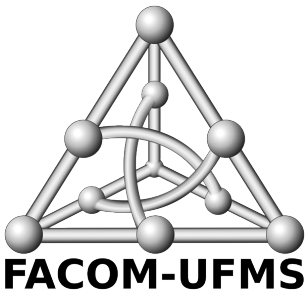
O Problema da Seleção de Segmentos Específicos: algoritmos e aplicações

Jean Alexandre Dobre

Dissertação Apresentada
à
Faculdade de Computação
da
Universidade Federal do Mato Grosso do Sul
para
obtenção do título
de
Mestre em Ciência da Computação



Orientador: Prof. Dr. Said Sadique Adi

Área de Concentração: Ciência da Computação

Campo Grande, Julho de 2017

O Problema da Seleção de Segmentos Específicos: algoritmos e aplicações

Esta versão da dissertação contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 31/07/2017.

Comissão Julgadora:

- Prof. Dr. Said Sadique Adi (orientador) - UFMS
- Profa. Dra. Nahri Balesdent Moreano - UFMS
- Pesquisador Dr. Leonardo Rippel Salgado - Embrapa

Agradecimentos

Primeiramente, gostaria de agradecer à minha esposa, Sheila e ao meu filho Kevin, que me apoiaram incondicionalmente durante o tempo em que estive dedicado no desenvolvimento deste trabalho;

Agradeço aos meus pais, Antônio e Roseli, pelo apoio durante toda minha vida, pelos valores morais que me ensinaram e pela compreensão que tiveram durante o período em que estive envolvido neste trabalho;

Agradeço muito ao meu orientador, professor Said Sadique Adi, pela paciência e atenção que teve comigo durante todo o período em que estivemos envolvidos. Sou profundamente grato por tudo que me ensinou, por suas ideias, dicas e correções, que me proporcionaram esclarecimento para o desenvolvimento e conclusão deste trabalho;

Ao meu amigo Emerson, agradeço pelo constante apoio e incentivo se fazendo presente em muitos momentos dessa longa trajetória;

Agradeço também aos colegas de trabalho na COIN/UFGD, Renato e Leandro, por ter disponibilizado equipamento de alta tecnologia, fundamental para a realização dos testes deste trabalho;

Por fim, agradeço ao chefe do setor de desenvolvimento da COIN/UFGD, Maic, pela compreensão e liberdade concedida para que pudesse realizar as reuniões necessárias para o desenvolvimento da dissertação.

Resumo

O Problema da Seleção de Segmentos Específicos consiste em, dadas duas ou mais sequências de DNA, encontrar o menor segmento em uma delas que tenha pelo menos k diferenças com relação a todos os segmentos das outras sequências. Esse problema é recorrente na Biologia, cuja solução possibilita, dentre outras coisas, uma amplificação precisa de regiões específicas de DNA em laboratório. Com isso é possível detectar e diagnosticar doenças infecciosas, e identificar o patógeno causador da infecção. O Problema da Seleção de Segmentos Específicos pode ser resolvido através de um problema computacional denominado Problema do *Primer* com k Diferenças. Embora ao longo do tempo algumas abordagens tenham sido propostas para esse último, com algoritmos cada vez mais eficientes, sabemos que alguns têm um alto custo de processamento e uso de memória, o que torna inviável sua aplicação na prática. Com isso em mente, e considerando que, até onde sabemos, nenhum estudo foi realizado para fazer uma comparação entre esses algoritmos, propomos aqui um estudo detalhado das diferentes abordagens conhecidas para resolver o Problema do *Primer* com k Diferenças, fazendo uma avaliação dos algoritmos relacionados com casos de testes artificiais e reais e utilizando o melhor deles no desenvolvimento de um sistema que possa ser utilizado de forma efetiva por biólogos e outros interessados na seleção de segmentos específicos.

Palavras-chave: Segmentos específicos, Amplificação, PCR, Região específica, *Primer* específico, Sonda.

Abstract

The Specific Segment Selection Problem takes as input two or more DNA sequences, and gives as output the shortest segment in one of them that has at least k differences from any segments of the other sequences. This problem is recurrent in Biology, whose solution enables, among other things, an accurate amplification of specific regions of DNA in laboratory. With this it is possible to detect and diagnose infectious diseases, and to identify the pathogen that causes the infection. The Specific Segment Selection Problem can be solved through a computational problem called k Diference *Primer* Problem. Although over time several approaches have been proposed to solve this last problem, with efficient algorithms, we know that some of them have a high cost of processing and memory usage, which makes its application impractical. With this in mind, and considering that, to the best of our knowledge, no studies have been carried out to compare these algorithms, we propose here a detailed study of the different known approaches to the k Diference *Primer* Problem, making an evaluation of the related algorithms by using artificial and real tests and using the best of them in the development of a software that can be used effectively by biologists and others interested in the selection of specific segments.
Keywords: Specific segments, Amplification, PCR, Specific region, Specific *Primer*, Probe.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Problemas que envolvem sequências são estudados frequentemente na área de Teoria da Computação e muitos algoritmos relacionados a esses problemas podem ser aplicados em outras áreas, tais como a Biologia Molecular, dando suporte a técnicas e procedimentos executados em laboratórios de análises de sequências genômicas. Dentre essas técnicas estão os testes para detecção de doenças ou de agentes causadores de doenças, que é algo difícil de ser executado por conta da necessidade de se identificar regiões específicas de DNA a serem amplificadas. Devido ao tamanho das sequências e à quantidade de sequências envolvidas, a identificação dessas regiões pode consumir muito tempo, tornando essa tarefa impossível de ser realizada manualmente.

Segmentos específicos são regiões de uma determinada sequência de DNA diferentes de qualquer outra região de outra(s) sequência(s) de DNA. Biólogos interessados na detecção (do agente causador) de doenças podem utilizar a técnica de PCR laboratorial para amplificar determinados trechos genômicos do agente causador da doença, possibilitando a sua identificação. Para isso, os trechos genômicos amplificados devem ser segmentos específicos, tornando sua identificação e seleção de suma importância para o sucesso da PCR e o diagnóstico correto da doença.

Tendo duas sequências em mãos, uma forma de selecionar segmentos específicos de uma delas seria analisando todos os segmentos possíveis dessa sequência na busca por um que não aparece na outra, o que seria inviável visto que o tamanho das sequências pode facilmente passar a quantidade de milhões de caracteres. Isso torna o desenvolvimento de algoritmos eficientes para a seleção de segmentos específicos um problema importante tanto do ponto de vista teórico quanto prático.

Os algoritmos existentes atualmente para o problema da seleção de segmentos específicos resolvem o seguinte problema computacional, conhecido como **Problema do *Primer* com k Diferenças**: dadas duas sequências α e β e um parâmetro k, encontre, para cada índice j de α, o menor segmento γ de α que se inicia em j com distância de edição de pelo menos k com relação a qualquer segmento de β. Esses algoritmos estão fortemente baseados na solução de um outro problema computacional, conhecido como **Problema do Casamento Inexato com até k Diferenças**, que consiste em encontrar ocorrências de uma sequência p em outra sequência t com no máximo k diferenças.

Em [2], Gusfield apresenta uma abordagem que resolve o Problema do *Primer* com k Diferenças eficientemente em tempo $O(kmn)$, onde $m = |\alpha|$, $n = |\beta|$ e k é a quantidade de diferenças. A solução consiste, basicamente, em invocar um algoritmo que resolve o Problema do Casamento Inexato com até k Diferenças para cada posição j de α e avaliar o resultado processado. Nesse contexto, existem técnicas que permitem melhorar a eficiência computacional de partes do algoritmo e que,

se aplicadas corretamente, podem tornar viável o uso do algoritmo em casos reais, que requerem um processamento massivo de dados em tempo hábil.

Dado o exposto acima, o objetivo principal deste trabalho é o estudo detalhado das diferentes abordagens que resolvem o Problema do *Primer* com k Diferenças, a implementação dos algoritmos relacionados e a comparação, em termos de tempo de execução e quantidade de memória utilizada, em casos de testes artificiais e reais. Faz parte também dos objetivos deste trabalho o desenvolvimento de uma ferramenta computacional com interface gráfica e funcionalidades que permitirá aos usuários usar todos os seus recursos afim de solucionar o Problema da Seleção de Segmentos Específicos.

1.1 Contribuições

Dentre as contribuições deste trabalho, listamos as principais como sendo:

- a implementação dos dois algoritmos que resolvem o Problema da Seleção do *Primer* com k Diferenças (PPkD);
- a geração de um conjunto de casos de testes com sequências artificiais e reais para avaliar os algoritmos que resolvem o PPkD, disponível em https://github.com/jeandobre/k-difference-prime/dados;
- a submissão de um artigo para publicação na décima edição da Revista Brasileira de Sistemas de Informação - isys, http://www.seer.unirio.br/index.php/isys, intitulado *Um sistema para auxiliar na seleção de regiões específicas de sequências biológicas*;
- o desenvolvimento de um sistema computacional para auxiliar biólogos na tarefa de seleção de segmentos específicos para as mais diversas atividades da PCR.

1.2 Organização do texto

Esta dissertação está dividia em cinco outros capítulos, além desta introdução, organizados da seguinte forma: o Capítulo 2 apresenta conceitos fundamentais, definições, problemas e algoritmos que estão relacionados diretamente ao problema objeto de estudo. O Capítulo 3 faz uma apresentação detalhada do problema da seleção de *primers* específicos e das principais abordagens, para resolvê-lo. O Capítulo 4 descreve a implementação dos algoritmos e os testes realizados, incluindo diversas tabelas e gráficos detalhando os resultados obtidos. O Capítulo 5 detalha o desenvolvimento do sistema computacional com todas as etapas e documentação envolvidas no processo. Finalmente, o Capítulo 6 apresenta as considerações finais e o que ainda pode ser explorado em trabalhos futuros.

Capítulo 2

Conceitos preliminares

Para que haja um melhor entendimento do problema que será abordado neste trabalho, assim como das suas aplicações, é necessário conhecer alguns conceitos básicos de Ciência da Computação e Biologia Molecular, bem como alguns algoritmos para problemas relacionados. Tais conceitos e algoritmos encontram-se detalhados neste capítulo, escrito com base em [2, 3, 4, 5, 6, 7, 8, 9, 10].

2.1 Biologia

Biologia Molecular é a área da Biologia que estuda os organismos em nível de moléculas, especificamente de suas células, com ênfase na formação, estrutura e função do seu material genético [11]. Ela corresponde a uma área ampla, que tem ligação íntima com a bioquímica e a genética. Nas próximas subseções descrevemos alguns dos principais conceitos da Biologia Molecular, detalhando como ocorre o processo de replicação de DNA na célula e o uso de *primers* em técnicas de replicação de DNA em laboratório.

2.1.1 A célula

A **célula** é considerada a unidade básica de constituição de todos os seres vivos, cujo desenvolvimento e manutenção depende do seu correto funcionamento [5]. Ela pode ser definida como um sistema estrutural, composto de um conjunto de elementos com funções específicas, envolto por uma membrana chamada **membrana celular** ou **membrana plasmática**, que mantém a sua integridade e proporciona trocas entre o meio e a célula.

Toda célula divide-se, basicamente, em duas regiões: núcleo e citoplasma. O **núcleo celular** é o centro de controle das atividades celulares e o armazenador das informações hereditárias que a célula transmite ao se reproduzir. O **citoplasma** é uma substância viscosa que preenche o espaço entre a membrana plasmática e o núcleo.

As células são classificadas em **procarióticas** e **eucarióticas** de acordo com características de sua organização estrutural. A célula cujo núcleo encontra-se misturado ao citoplasma é classificada como procariótica e a célula que possui seu núcleo, geralmente bem mais complexo, protegido e separado do citoplasma por uma membrana, denominada **membrana nuclear**, é classificada como eucariótica. Como exemplo podemos citar as bactérias como seres procarióticos e a maior parte dos seres vivos, como animais, plantas e fungos como seres eucarióticos.

Dentre os vários elementos constituintes das células destacam-se suas organelas (ex. mitocôndrias, ribossomos, complexo de Golgi, etc) e duas moléculas fundamentais para o seu desenvolvimento: a molécula de *ácido desoxirribonucleico* e a molécula de *ácido ribonucleico*.

2.1.2 Ácidos nucleicos

Os **ácidos nucléicos** são moléculas gigantes (macromoléculas), formadas por unidades monoméricas menores conhecidas como nucleotídeos. O ácido desoxirribonucléico, ou simplesmente DNA (do inglês, *deoxyribonucleic acid*) e o ácido ribonucléico, ou simplesmente RNA (do inglês, *ribonucleic acid*), são os dois tipos de ácidos nucléicos presentes na célula [1, 3, 5]. Cada nucleotídeo, por sua vez, é formado por três elementos:

- um radical "fosfato", derivado da molécula do ácido ortofosfórico (H_3PO_4);
- uma desoxirribose, ou seja, um açúcar do grupo das pentoses (monossacarídeos com cinco átomos de carbono);
- uma base orgânica nitrogenada, ou simplesmente base, divididas em dois grupos: Purinas (*Adenina* e *Guanina*), que possuem anel duplo de átomos de carbono e as Pirimidinas (*Citosina*, *Timina* e *Uracila*) que possuem anel simples.

Cada carbono da pentose que constitui os nucleotídeos é rotulado com 1', 2', 3', 4' e 5' de acordo com sua posição. Dentro do nucleotídeo O carbono 1' se liga com a base enquanto que o carbono 5' se liga ao fosfato. As bases que compõem o DNA são a (**A**) adenina, (**C**) citosina, (**G**) guanina e (**T**) timina, enquanto que as bases que compõem o RNA são a (**A**) adenina, (**C**) citosina, (**G**) guanina e (**U**) uracila.

A Figura 2.1 é uma representação da estrutura dos nucleotídeos que compõem o DNA.

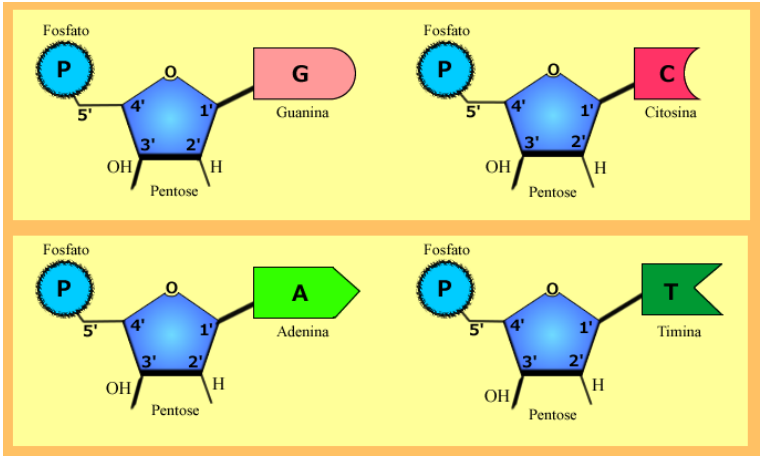


Figura 2.1: *Uma representação dos nucleotídeos de uma molécula de DNA. Extraído de [1].*

Uma **fita** do DNA é uma sequência de nucleotídeos dessa molécula unidos quimicamente pelos carbonos 3' e 5' da pentose, através do fosfato. Quando a ligação é feita entre o carbono 5' de um nucleotídeo e o carbono 3' do próximo nucleotídeo da fita, é dito que o sentido da fita é 5'→ 3'. Da mesma forma, quando a ligação é feita entre o carbono 3' de um nucleotídeo e o carbono 5' do próximo nucleotídeo da fita, é dito que o sentido da fita é 3'→ 5'. A extremidade da fita que possui um nucleotídeo em que o carbono 3' está livre é chamada de **extremidade 3'**, e a extremidade

da fita que possui um nucleotídeo em que o carbono 5' está livre é chamada de **extremidade 5'** [1, 12].

A estrutura do DNA consiste, basicamente, de duas fitas de nucleotídeos dispostas de forma antiparalelas, enroladas (enoveladas) uma sobre a outra em proteínas chamadas histonas, com aspecto de dupla hélice, compostas por milhares de nucleotídeos, a Figura 2.2 a seguir apresenta um esquema simplificado da estrutura do DNA.

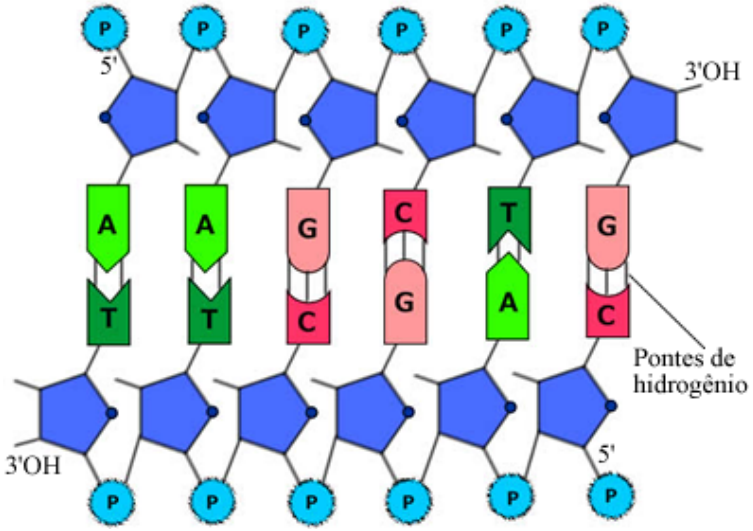


Figura 2.2: *A estrutura do DNA. Extraído de [1].*

A união das duas fitas é feita por pontes de hidrogênio, que ligam pares de bases específicas do DNA: AT, CG, TA e GC. Essas bases são chamadas **bases complementares**. As bases são sempre ligadas com o seu complemento. Com isso, conhecendo uma fita e o sentido dela, é possível determinar a fita complementar de DNA. Como exemplo, para um segmento de fita AATCGATGGA, o seu complemento será TTAGCTACCT. Por convenção, uma sequência de DNA é sempre representada no sentido 5'→ 3', ou seja a extremidade 5' está à esquerda na sequência e a extremidade 3' está à direita na sequência. Na Figura 2.2, é possível ver a ligação entre os nucleotídeos através do radical fosfato formando a fita simples e, por meio das pontes de hidrogênio, formando a dupla fita de DNA.

O **RNA** é um ácido nucleico semelhante ao DNA, mas com algumas características diferentes. No RNA a base Timina (T) é substituída pela base Uracila (U), que faz complementaridade de base com a Adenina (A). O RNA é formado por apenas uma fita e, em seus nucleotídeos, a pentose possui uma molécula de oxigênio adicional ligada ao carbono 2'. Na célula existem diferentes tipos de RNA, cada qual desempenhando uma função específica e bem definida [6].

2.1.3 Replicação do DNA

O processo de *replicação* ou *duplicação* do DNA é essencial para a manutenção da vida e ocorre na fase de duplicação celular. Esse processo envolve diversas enzimas que atuam sobre o DNA. Entre elas estão a **topoisomerase**, uma enzima que aumenta ou diminui a tensão das fitas enoveladas sendo assim capaz de desenrolar o DNA, a **helicase**, uma enzima capaz de quebrar as pontes de hidrogênio e assim separar as fitas de DNA e a **DNA polimerase**, uma enzima capaz de construir, ou seja, sintetizar uma nova fita de nucleotídeos a partir de um molde e um segmento iniciador

[1, 5].

No processo de replicação do DNA, primeiramente, as duas fitas que o compõe são separadas pela ação da topoisomerase e a helicase. Chamamos cada fita separada de **fita molde**. Em seguida, a DNA polimerase promove a ligação de novos nucleotídeos, com base na fita molde, a um segmento iniciador, construindo duas novas cadeias de nucleotídeos. Na Figura 2.3 é possível ver as principais enzimas envolvidas no processo de replicação de DNA na célula.

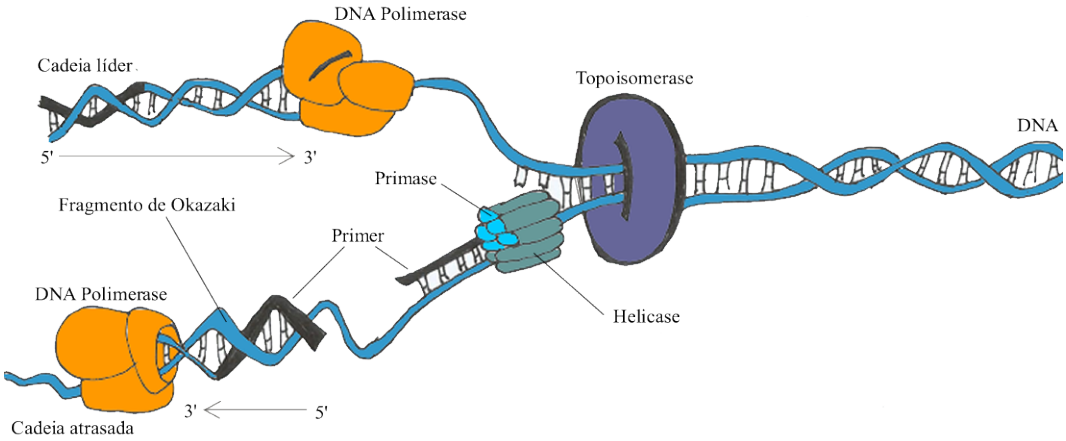


Figura 2.3: *Um exemplo de como ocorre o processo de replicação do DNA e as enzimas envolvidas.*

A síntese das cadeias complementares ocorre sempre no sentido 5'→ 3', ou seja, a DNA polimerase necessita de uma extremidade 3' livre para iniciar a síntese de DNA. Como as fitas do DNA possuem orientação opostas, apenas uma cadeia, chamada de **cadeia líder** é sintetizada de forma contínua. A outra cadeia, chamada de **cadeia atrasada**, é sintetizada de forma semicontínua.

A DNA polimerase necessita de um segmento iniciador, denominado *primer*, para iniciar a sua ação. Dentro da célula, o **primer** corresponde a um segmento de RNA produzido pela ação de uma enzima especial chamada **primase** em intervalos regulares [5]. O *primer* se liga à fita molde por complementaridade de bases, num processo denominado de **hibridização**. Um *primer* hibridiza à cadeia líder e a DNA polimerase sintetiza o restante da cadeia de forma contínua e rápida. Como a cadeia atrasada está no sentido 3'→ 5', ela é sintetizada pela DNA polimerase em fragmentos, denominados **fragmentos de Okazaki** que precisam, além de um *primer* em cada início de fragmento, da atuação de outras enzimas para unir os fragmentos e torná-los uma nova fita. A sintetização em forma de fragmentos torna lenta a construção da nova fita de DNA e, por conta disso, é dita atrasada em relação à cadeia líder.

Em laboratório, o *primer* pode ser sintetizado quimicamente a partir de uma parte conhecida da sequência de DNA de interesse. Esse *primer* sintético, também conhecido como **oligonucleotídeo**, é um pedaço pequeno de fita simples composto de bases de DNA complementares às bases de um dado segmento de DNA de interesse. A sua sintetização industrial permite, juntamente com a ação da DNA polimerase e outros compostos, executar o processo de replicação de DNA em laboratório.

2.1.4 Replicação de fragmentos de DNA em laboratório

Fora do processo natural de replicação de DNA na célula, existe uma técnica desenvolvida em laboratório para replicar fragmentos de DNA, sem a necessidade de uma célula viva, chamada de *reação em cadeia da polimerase*, ou simplesmente PCR (do inglês, *Polymerase Chain Reaction*)

[5]. O DNA de interesse que contém a região a ser replicado é chamado de **DNA alvo** ou **DNA molde**. Essa reação ocorre em tubos de ensaio contendo o DNA alvo e mais alguns compostos, como oligonucleotídeos e a enzima taq DNA polimerase (DNA polimerase termo resistente, ou seja, resistente à altas temperaturas) que faz a replicação do DNA. A PCR é usada habitualmente nos laboratórios de investigação médica e biológica para uma variedade de tarefas tais como a detecção de doenças hereditárias, a construção de árvores filogenéticas (árvores de relação entre espécies), a clonagem de genes, testes de paternidade, exames para detecção de agentes patogênicos entre outras [3].

A técnica de PCR é realizada em uma máquina chamada *termociclador*, onde são postos os microtubos de ensaio. No processo padrão da PCR, para replicar fragmentos de DNA, são necessários quatro principais componentes dentro dos microtubos, a saber, uma amostra de qualquer resíduo que contenha o DNA alvo, um par de oligonucleotídeos, a taq DNA polimerase e as **dNTP**'s (desoxirribonucleotídeos fosfatados) que são nucleotídeos disponíveis, a serem utilizados pela DNA polimerase para construir uma nova fita de DNA [13].

O processo de replicação consiste em três fases ou etapas. A primeira fase é a *desnaturação*, em que o DNA alvo é desnaturado por aquecimento da solução (a cerca de $95°C$) por aproximadamente 30 segundos. Como a estrutura em dupla hélice do DNA é mantida através de ligações por pontes de hidrogênio entre as bases pareadas, o calor causa a separação das suas duas fitas; a segunda fase é o anelamento ou *hibridização*, em que os *primers* se ligam por hibridação às respectivas regiões complementares em cada fita. Essa etapa ocorre a uma temperatura, não fixa, entre 45° e $65°C$, por aproximadamente um minuto [4]; a fase final é a extensão, ou *polimerização*, em que a taq DNA polimerase usa a extremidade 3' dos *primers* para formar uma nova cadeia com bases complementares ao molde. Essa fase ocorre a uma temperatura de $72°C$, por aproximadamente um minuto. O processo todo é então reiniciado e pode ser repetido muitas vezes até que se tenha a quantidade de fragmentos desejado [13, 14].

Cada ciclo da PCR duplica apenas a região que está entre os *primers forward* e *reverse* da sequência de DNA em estudo. Conforme o procedimento é realizado, os fragmentos sintetizados servem como modelo para novos fragmentos, de forma que, com alguns ciclos, o pedaço de DNA predominante na solução é idêntico à região alvo. Assim, após um número n de ciclos de PCR, tem-se idealmente 2^n regiões idênticas à região do DNA original compreendida entre os dois *primers*.

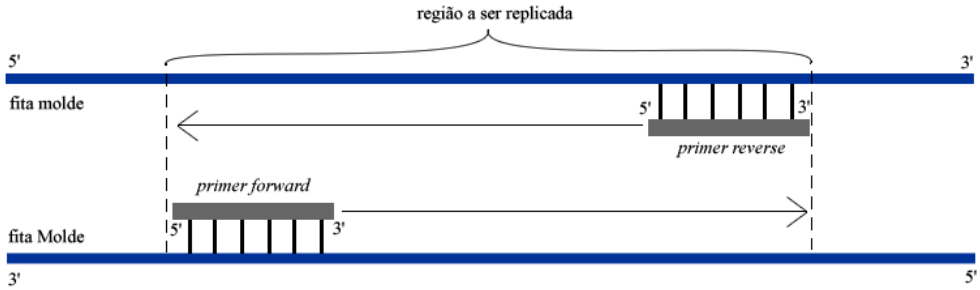


Figura 2.4: *Exemplo de uso de primers para replicação de DNA*

Neste procedimento é necessário um par de *primers* selecionados para delimitar o início e fim da região alvo da sequência de DNA que se quer amplificar. Chamamos de *primer forward* o *primer* que se liga à região inicial de uma das fitas molde e de *primer reverse*, o *primer* que se liga à região final da outra fita molde. A Figura 2.4 mostra um exemplo de como um par de *primers*, em

cor cinza, são posicionados. As setas representam a direção em que ocorre a sintetização das novas cadeias.

É impostante observar que, em várias tarefas que envolvem a PCR, como a detecção de doenças e/ou agentes infecciosos, é fundamental que os trechos a serem amplificados sejam segmentos específicos, ou seja, regiões de uma determinada sequência de DNA diferentes de qualquer outra região de outra(s) sequência(s) de DNA.

2.1.5 *Primers* e sondas

O processo de replicação de DNA através da PCR é utilizado amplamente em técnicas moleculares para as mais diversas finalidades. Em toda PCR é necessário um conjunto de *primers*, que são oligonucleotídeos sintéticos curtos, complementares à sequência molde, para permitir o inicio da formação da nova cadeia. O *primer* é extraído da região da sequência de DNA que se deseja amplificar, selecionado com base na técnica molecular que será executada. Além disso, um *primer* deve ter algumas características, que devem ser analisadas durante a seleção [15].

Um *primer* ideal possui as seguintes características:

- pode ser selecionado de qualquer ponto da sequência molde;
- ser o menor segmento possível, dado que o custo da sintetização do *primer* é diretamente proporcional ao tamanho do segmento; assim, quanto maior, mais caro [7];
- ter em sua composição entre 40% e 60% de bases C e G para aumentar a sua hibridização com o DNA molde, visto que o pareamento dessas bases é feita com três pontes de hidrogênio;
- ter em sua extremidade 3' uma das bases, C ou G, para que o pareamento seja estável.

Além disso, alguns cuidados devem ser tomados com relação à estrutura interna do *primer*, no intuito de não permitir que se alinhe consigo (*self-dimer*), com outro *primer* (*hetero-dimer*) ou em uma região diferente da região alvo no DNA molde (*primer* não específico). Para isso, deve ser evitado um conjunto de bases repetidas (*repeat*), uma sequência de repetições de uma única base (*runs*). Além disso, pelo menos metade do *primer* deve ser diferente de qualquer região que não se tenha interesse em replicar [2].

Um *primer* é específico se ele se pareia apenas com a região específica da qual foi selecionado na sequência de DNA e com nenhuma outra região desse DNA ou de qualquer outro DNA na solução. A especificidade do *primer* está relacionada diretamente ao seu tamanho, de forma que, quanto maior, mais específico. Contudo, o tamanho do *primer* impacta diretamente em seu custo de produção, ou seja, quanto maior mais caro.

A **sonda** (do inglês, *probe*) é um segmento complementar de DNA de tamanho variável (de 100 a 1.000 pares de bases), produzido sinteticamente e usado principalmente para detectar a presença de determinadas sequências de nucleotídeos no DNA alvo. Elas são marcadas radioativamente ou quimicamente, com isótopos fluorescentes, que permitem sua detecção e identificação visual, através de técnicas de imagens, para extrair informações genéticas ou identificar uma determinada sequência alvo entre um conjunto de outras sequências de DNA [6].

2.2 Computação

Bioinformática ou biologia computacional é a área da Computação aplicada à análise e modelagem de problemas relacionados à biologia, aplicando técnicas da informática para auxiliar a comparação de dados genéticos e genômicos bem como na compreensão dos aspectos evolutivos da biologia molecular. Nas próximas subseções descrevemos alguns dos principais conceitos relacionados a sequências computacionais e problemas relacionados.

2.2.1 Sequências

Uma **sequência** ou **cadeia** s construída sobre um alfabeto Σ, que corresponde a um conjunto finito de caracteres, nada mais é do que uma concatenação ordenada e finita de caracteres de Σ. O **tamanho** de uma sequência s é representado por $|s|$ e corresponde à quantidade de caracteres que ela possui. O **índice** i de uma sequência s representa a posição de um caractere nessa sequência, e denotamos por s_i o i-ésimo caractere dela. Denotamos por $t_{1..n}$ uma sequência t de tamanho n. Se uma sequência é vazia, ou seja, se $|s| = 0$, ela é representada por ϵ e chamada de **cadeia vazia**. Como exemplo das definições acima, a sequência s = TTAGCGAC corresponde a uma sequência construída sobre o alfabeto Σ = {A, C, G, T}. Observe que $|s| = 8$ e que s_2 = T e s_7 = A.

Uma **subsequência** s' de uma sequência s consiste em uma cópia de s contendo todos, alguns ou nenhum de seus caracteres. Uma **subcadeia** ou **segmento** de uma sequência s é um trecho de s com zero ou mais caracteres consecutivos. Se o primeiro caractere de uma subcadeia de s corresponde ao seu i-ésimo caractere e o último ao seu j-ésimo caractere, a subcadeia é denotada por $s_{i..j}$. Se $i > j$, o segmento corresponde à cadeia vazia ϵ. Como exemplo, para uma sequência s = TTAGCGAC, temos o segmento $s_{2..5}$ = TAGC e o segmento $s_{6..8}$ = GAC.

A **concatenação** de duas cadeias s e t, denotada por $s \bullet t$, é uma operação que gera uma terceira cadeia que consiste dos caracteres de s seguidos pelos caracteres de t. A concatenação da sequência s do exemplo anterior com a sequência t = AAACGCT, também construída sobre o alfabeto Σ = {A, C, G, T}, corresponde à sequência u = TTAGCGACAAACGCT. Observe que $|u| = |s| + |t|$, quando $u = s \bullet t$.

Um **prefixo** de uma sequência s é um segmento $s_{1..j}$ tal que $1 \leq j \leq |s|$. Se $j < 1$, tem-se um **prefixo vazio**, que corresponde à cadeia vazia. De forma semelhante, um **sufixo** de uma sequência s é um segmento $s_{i..|s|}$ tal que $1 \leq i \leq |s|$. Se $i > |s|$, tem-se um **sufixo vazio**, que corresponde à cadeia vazia. O i**-ésimo** sufixo de s, denotado por $s_{i..n}$, representa um sufixo iniciando na posição i de s. No caso da sequência s = TTAGCGAC dada como exemplo temos, para $j = 3$ o prefixo p = TTA, e para $i = 5$ o sufixo v = CGAC.

O **maior prefixo comum** ou LCP (do inglês, *Longest Commom Prefix*) entre duas sequências, $p_{1..m}$ e $t_{1..n}$, é o maior segmento $s_{1..k}$ tal que $0 \leq k \leq m, n$, onde $s_1 = p_1 = t_1$, $s_2 = p_2 = t_2$ .. $s_k = p_k = t_k$. Se $k = 0$ então $s = \epsilon$. Denotamos por $LCP_{p,t}(i, j)$ o maior prefixo comum entre $p_{i..m}$ e $t_{j..n}$. Para exemplificar, dadas as sequências p = CGTATT e t = GTATG, temos que o $LCP_{p,t}(1, 1) = \epsilon$ e que o $LCP_{p,t}(2, 1)$ = GTAT, ou seja $p_{2..5} = t_{1..4}$.

A **maior extensão comum** ou LCE (do inglês, *Longest Common Extension*) entre duas sequências, $p_{1..m}$ e $t_{1..n}$, é o tamanho do maior prefixo comum entre $p_{i..m}$ e $t_{j..n}$, ou seja, $|LCP_{p,t}(i, j)|$. Denotamos por $LCE_{p,t}(i, j)$ a maior extensão comum entre $p_{i..m}$ e $t_{j..m}$. Para exemplificar, dadas as sequências p = CGTATT e t = GTATG, temos que o $LCE_{p,t}(1, 1) = 0$ e que o $LCE_{p,t}(2, 1) = 4$.

2.2.2 Distância de edição

Em várias situações práticas, é necessário determinar quão parecida (ou diferente) são duas sequências, e para isso existem diversas medidas. Nesta seção será detalhada uma delas, conhecida como *distância de edição* ou *distância de Levenshtein* [16].

De forma bem simples, podemos definir a **distância de edição** ou **número mínimo de diferenças** entre duas sequências como sendo a menor quantidade possível de operações de *inserção*, *remoção* e *substituição* necessárias para transformar uma das sequências na outra [2, 16, 17]. Essas operações são chamadas de **operações de edição** e elas atuam sobre caracteres individuais de uma sequência. Como exemplo, tomando-se s = TACTG e t = ACGCT, a distância de edição entre elas corresponde a 4, relacionada às operações de remoção de s_1 = T, substituição de s_4 = T por G, substituição de s_5 = G por T e inserção de T ao final de s, totalizando 4 diferenças.

Dadas duas sequências s e t, $D(i, j)$ corresponde à distância de edição entre $s_{1..i}$ e $t_{1..j}$. Isto é, $D(i, j)$ denota o menor número de operações de edição necessárias para transformar os primeiros i caracteres de s nos primeiros j caracteres de t. Usando as sequências s e t do exemplo anterior, $D(2, 3)$ representa a distância de edição entre $s_{1..2}$ = TA e $t_{1..3}$ = ACG. Usando esta notação, se s tem tamanho m e t tem tamanho n, então a distância de edição entre s e t é precisamente o valor $D(m, n)$. Dessa forma, usando as sequências s e t do exemplo anterior, temos que $D(5, 5)$ = 4 diferenças.

Dadas as definições anteriores, o Problema da Distância de Edição pode ser assim definido:

> **Problema da Distância de Edição:** dadas duas sequências s e t, com $|s| = m$, $|t| = n$, determinar o menor número de operações de substituição, inserção e remoção necessárias para transformar s em t.

Um algoritmo para resolver o Problema da Distância de Edição emprega uma abordagem conhecida como **programação dinâmica**. Essa é uma técnica crucial para resolver problemas de otimização com base em recorrências matemáticas. Algoritmos para o cálculo da distância de edição, para a determinação da *sequência de fibonacci* e para *multiplicação de cadeia de matrizes* são exemplos de algoritmos baseados em programação dinâmica. Ela pode ser aplicada a problemas em que a solução ótima pode ser determinada a partir de soluções ótimas previamente determinadas e armazenadas para subproblemas do problema original, evitando recálculos [18]. Tal técnica consiste de três componentes essenciais:

- a **relação de recorrência**: estabelece uma *relação de recursividade* que, dada uma condição inicial, permite determinar o próximo termo em função dos antecessores.

No caso do Problema da Distância de Edição, dadas duas sequências s e t, as condições iniciais são $D(i, 0) = i$, para $0 \leq i \leq |s|$ e $D(0, j) = j$, para $0 \leq j \leq |t|$. Sobre a relação de recursividade para esse problema, é possível determinar o valor $D(i, j)$ através do cálculo da distância de edição envolvendo todos os prefixos de $s_{1..i}$ e $t_{1..j}$. Observe que existem três possibilidades para se determinar a distância de edição entre $s_{1..i}$ e $t_{1..j}$:

- determinar a distância de edição entre $s_{1..i}$ e $t_{1..j-1}$ e somar o valor 1, referente à inserção do caractere t_j na posição i de s (após deslocar o segmento $s_{i..m}$ uma posição à direita);

- determinar a distância de edição entre $s_{1..i-1}$ e $t_{1..j}$ e somar o valor 1, referente a remoção do caractere s_i;

- determinar a distância de edição entre $s_{1..i-1}$ e $t_{1..j-1}$ e somar o valor 1 se $s_i \neq t_j$, referente à substituição do caractere s_i por t_j;

A partir dessas possibilidades e observando que estamos em busca do menor número de operações que transformam s em t, podemos determinar a seguinte relação de recorrência para solução do Problema da Distância de Edição:

$$D(i,j) = min \begin{cases} D(i-1,j)+1 \\ D(i,j-1)+1 \\ D(i-1,j-1) \quad + \quad \alpha(i,j), \end{cases} \tag{2.1}$$

onde:

$$\alpha(i,j) = \begin{cases} 0 \quad para \quad s_i = t_j \\ 1 \quad para \quad s_i \neq t_j, \end{cases}$$

para todo $i \geq 0$ e $j \geq 0$.

- a **computação tabular**: permite armazenar os valores calculados a cada passo em uma matriz de dados.

No caso do Problema da Distância de Edição, é necessário uma matriz D de dimensão $(m+1) \times (n+1)$, denotada por $D[0..m; 0..n]$, para armazenar todos os valores computados, onde $D[i,j]$ armazena o valor $D(i,j)$.

- a **rastreabilidade**: consiste em identificar, para cada célula da matriz, qual célula anterior foi utilizada para o cálculo do seu valor. Isso é essencial, para se conseguir reconstruir a solução ótima do problema e uma das formas de se fazê-lo é adicionando ponteiros nas células da matriz de programação dinâmica.

No caso do Problema da Distância de Edição, é possível adicionar até três tipos de ponteiros em uma célula, representando cada caso da Recorrência 2.1. A seta $\uparrow$ indicaria que o valor foi gerado a partir da célula superior $D[i-1,j]$, representando a operação de remoção do caractere s_i, a seta $\nwarrow$ indicaria que o valor foi gerado a partir da célula $D[i-1,j-1]$, representando a operação de substituição de s_i por t_j, quando $s_i \neq t_j$, e nenhuma operação quando $s_i = t_j$, e a seta $\leftarrow$ indicaria que o valor foi gerado a partir da célula à esquerda $D[i,j-1]$, representando a operação de inserção do caractere t_j. A célula $D[i,j]$ pode apontar para uma, duas ou até três células, indicando os possíveis caminhos para a solução ótima. As células da coluna 0, $D[i,0]$, para $1 \leq i \leq m$, apontam sempre para a célula imediatamente acima. As célula da linha 0, $D[0,j]$, para $1 \leq j \leq n$, apontam sempre para a célula imediatamente à esquerda. $D[0,0]$ é a única célula da matriz que não aponta para nenhuma outra célula.

Após o cálculo de $D(|s|,|t|)$, pode-se construir a sequência ótima de operações de edição, necessárias para transformar a sequência s em t partindo-se de $D[|s|,|t|]$ e percorrendo-se a matriz no sentido contrário àquele utilizado para o seu preenchimento, ou seja, voltando-se nas células seguindo as setas.

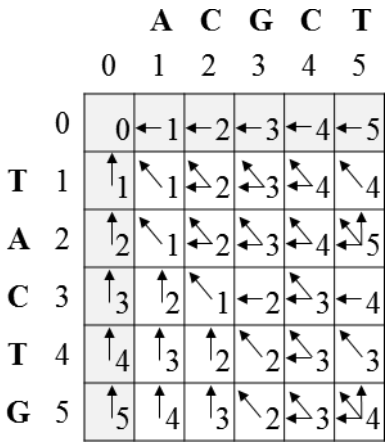


Figura 2.5: *Exemplo de matriz $D[0..m, 0..n]$ com rastreabilidade.*

A Figura 2.5 mostra a matriz D computada para as sequências do exemplo anterior, s = TACTG e t = ACGCT. O valor de cada célula $D[i, j]$ indica a distância de edição entre $s_{1..i}$ e $t_{1..j}$ e as setas representam a rastreabilidade para determinação da sequência ótima de operações. Por exemplo, o valor armazenado na célula $D[3, 5] = 4$ representa a menor quantidade de operações necessárias para transformar $s_{1..3}$ = TAC em $t_{1..5}$ = ACGCT. Partindo-se da célula $D[5, 5]$ e seguindo as setas utilizadas para o preenchimento de D, é possível obter as sequências de operações RNNSSI, RNNISS e SISNNR para transformar s em t, onde I, S, R representam as operações de inserção, substituição, remoção, respectivamente e N nenhuma operação.

Com base na Recorrência 2.1 detalhada anteriormente, o Problema da Distância de Edição entre uma sequência de tamanho m e uma sequência de tamanho n pode ser resolvido através da programação dinâmica em tempo e espaço $O(mn)$, e os passos dessa solução estão especificados no Algoritmo 1. Perceba que o espaço ocupado pela matriz de programação dinâmica usada pelo algoritmo pode ser reduzido para $O(n)$. Porém nesse caso não se consegue fazer a rastreabilidade e obter a sequência de operações. Isto pois apenas a linha prévia precisa ser armazenada no intuito de computar a próxima linha. Com isso, precisamos manter apenas um vetor D' de n posições, constantemente atualizado, para representar cada linha da matriz D [19].

Algoritmo 1 Algoritmo_Distância_de_Edição

Entrada: duas sequências s e t
Saída: distância de edição entre s e t

```
1:  m ← |s|;
2:  n ← |t|;
3:  para j ← 0 até n faça                                   ▷ Inicialização
4:      D[0, j] ← j;
5:  fim_para
6:  para i ← 1 até m faça
7:      D[i, 0] ← i;
8:  fim_para
9:  para i ← 1 até m faça                                   ▷ Execução
10:     para j ← 1 até n faça
11:         se s_i = t_j então
12:             α ← 0;
13:         senão
14:             α ← 1;
15:         fim_se
16:         D[i, j] ← min(D[i − 1, j] + 1, D[i, j − 1] + 1, D[i − 1, j − 1] + α);
17:     fim_para
18: fim_para
19: devolva D[m, n];
```

A partir do Algoritmo 1 é possível solucionar outros problemas mais complexos que utilizam a distância de edição como medida de comparação entre duas sequências. Um desses problemas é conhecido como Casamento Inexato com até k Diferenças e será apresentado em detalhes na seção 2.2.9.

2.2.3 Árvores

Uma **árvore** T é um conjunto finito de elementos denominados **nós**. Se $T = \emptyset$, então a árvore é vazia; senão, existe um nó especial r, chamado raiz de T e, possivelmente, outros nós, que constituem um conjunto vazio ou encontram-se divididos em $m \geq 1$ conjuntos distintos não vazios chamados de **subárvores** de r, cada qual sendo uma árvore. Cada nó possui um identificador associado, denominado **rótulo** do nó. Se v é um nó de uma árvore T então, a notação T_v indica a subárvore de T com raiz v. Uma árvore pode ser representada de forma hierárquica, através de um conjunto de vértices, que correspondem aos nós da árvore, conectados por meio de arestas.

Os nós raízes $w_1, w_2, ..., w_j$ das subárvores de T_v são chamados **filhos** de v. O nó v é chamado **pai** de $w_1, w_2, ..., w_j$ e os nós $w_1, w_2, ..., w_j$ são ditos **irmãos**. Se y pertence à subárvore T_v, y é **descendente** de v, e v é **ancestral** de y. Se $y \neq v$ então y é **descendente próprio** de v, e v é **ancestral próprio** de y. Uma **folha** é um nó que não possui descendentes próprios. Um nó que não é folha é chamado de **nó interno**. O **grau** de saída de um nó é dado pelo número de filhos que este nó possui. O grau da árvore é dado pelo seu nó de maior grau.

Um **caminho** em uma árvore T corresponde a uma sequência de nós distintos $v_1, v_2, ..., v_k$ em T, tal que existe a relação "é pai de" entre todos os nós consecutivos dessa sequência. Em um caminho cujo primeiro nó é v_1 e o último nó é v_k, é dito que v_1 alcança v_k, e vice-versa. Dado um caminho $v_1, v_2, ..., v_k$ em T, o valor $k - 1$ é o **comprimento** do caminho. O **nível** ou **profundidade** de um

nó v é dado pelo comprimento do caminho que vai da raiz até o nó v. O nível da raiz é 0. O nível da árvore é dado pelo seu nó de maior nível. A **altura** de um nó v é dada pela quantidade de nós do maior caminho de v até um de seus descendentes, com as folhas tendo altura igual a 1. A altura de uma árvore T é igual a altura do seu nó raiz, e é denotada por $h(T)$, enquanto que $h(v)$ representa a altura da subárvore de raiz v [20].

O **ancestral comum mais baixo** ou LCA (do inglês, *Lowest Common Ancestor*) de dois nós x e y, denotado por $LCA(x, y)$, é o nó mais profundo em T que é ancestral de ambos os nós x e y, ou seja, o LCA entre dois nós é o nó, de maior nível, ancestral de x e de y.

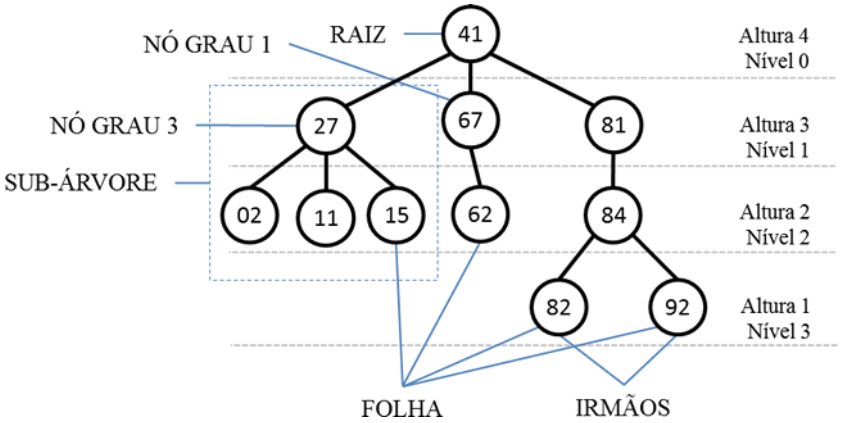


Figura 2.6: *Um exemplo de árvore de dados.*

Para exemplificar as definições acima, usaremos a Figura 2.6 que apresenta uma árvore simples, representada de forma hierárquica, em que cada círculo representa um nó da árvore e cada nó possui um rótulo, que vamos utilizar como sua identificação. O nó 41 é a raiz da árvore. Os nós 02, 11, 15, 62, 82 e 92 são folhas da árvore. O grau do nó 27 é 3 e do nó 67 é 1. O nível do nó 81, 67 e 27 é 1. O grau da árvore é 3, a altura da árvore é 4 e o nível da árvore é 3. Perceba que os nós 02, 11 e 15 são irmãos. O nó 81 é pai do nó 84 e é ancestral comum dos nós 82 e 92, mas o ancestral comum mais baixo dos nós 82 e 92 é o nó 84.

Um **percurso** em uma árvore é o ato de, a partir de um dado nó, geralmente o nó raiz, visitar todos os nós da árvore, por meio de suas arestas, sem repetir nó algum. Existem três formas sistemáticas de percorrer uma árvore, que são:

- pré-ordem (*preorder*): visitar a raiz, visitar a sub-árvore à esquerda e por último visitar a sub-árvore à direita;
- em ordem (*inorder*): visitar a sub-árvore à esquerda, visitar a raiz e por último visitar a sub-árvore à direita;
- pós-ordem (*postorder*): visitar a sub-árvore à esquerda, visitar à sub-árvore à direita e por último visitar a raiz;

2.2.4 Árvore de sufixo

Uma **árvore de sufixo** é uma árvore de dados construída a partir de sufixos de um determinado texto ou sequência. Dada uma sequência p, de tamanho n, construída sobre um alfabeto Σ, a árvore

de sufixo ST da sequência p, concatenado com o símbolo \$, é uma árvore com $n+1$ nós folhas, enumerados de 1 a $n+1$. Cada sufixo $p_{i..n}$ de p define exatamente uma folha da árvore, ou seja, cada nó folha recebe um índice de p como rótulo que representa o sufixo $p_{i..n}$ de p. Cada nó interno pode ter dois ou mais filhos e cada aresta de ST representa um segmento não vazio de p [21]. O grau de cada nó da árvore é zero, quando o nó é uma folha, ou maior ou igual a dois quando o nó é interno.

A partir da raiz da árvore de sufixo ST, denotada por $r[ST]$, todos os sufixos de p com mesmo prefixo compartilham o mesmo caminho nessa árvore. Cada aresta de ST é rotulada por um segmento de p, tal que, para um nó v os rótulos das arestas que ligam v a seus filhos são diferenciados pelos seus i caracteres iniciais. Na verdade, cada aresta é rotulada com o índice do caractere inicial e final de seu rótulo. Dessa forma cada rótulo consome espaço constante de 2 inteiros para sua representação [22].

O **segmento do caminho** é um segmento obtido da concatenação em ordem dos rótulos de todas as arestas que estão nesse caminho. Se o caminho tem apenas o nó v e v é raiz da árvore, é dito que o segmento do caminho é vazio. Em um caminho da raiz até um nó folha v, um segmento de caminho corresponde a um sufixo $p_{i..n}$ de p, pois o nó folha v representa um sufixo de p começando na posição i. Em um caminho da raiz até um nó interno w, a quantidade de caracteres do segmento do caminho, representa a profundidade ou nível do nó w.

A Figura 2.7 apresentada a seguir mostra uma árvore de sufixo ST construída para a sequência p = ACGTTACGTGA\$. Nela podemos ver cada sufixo $p_{i..n}$ representado por um nó rotulado com o índice i de p. Os rótulos de arestas são apresentados para facilitar a visualização dos sufixos e a subárvore z mostra como são armazenado os rótulos de arestas. Ainda é possível ver um caminho da raiz até o nó y, cuja concatenação em ordem dos rótulos das arestas é ACGT, a quantidade de caracteres é 4 e corresponde ao nível do nó y. Em um caminho da raiz até o nó 8, o segmento de caminho é igual ao prefixo $p_{8..n}$, ou seja, GTGA\$.

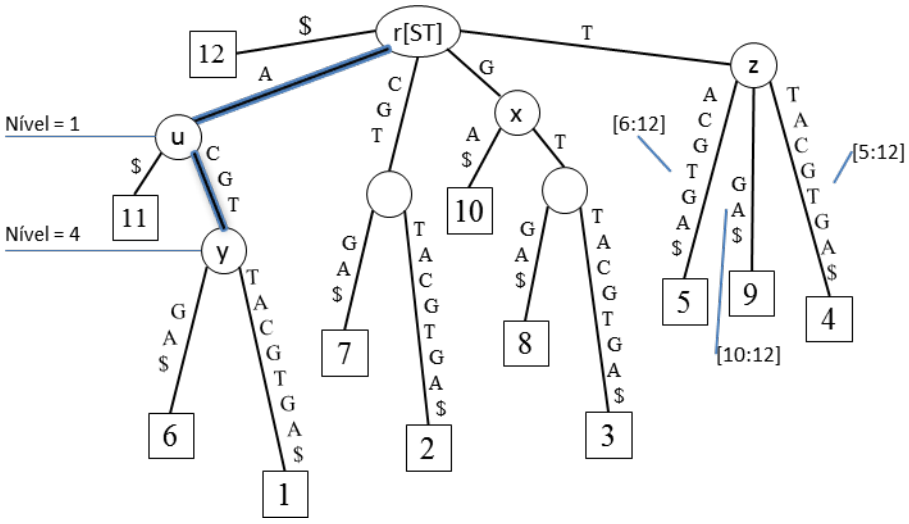


Figura 2.7: *Árvore de sufixo ST para a sequência p = ACGTTACGTGA\$.*

A construção da árvore de sufixo ST para uma sequência p de tamanho n pode ser feita em tempo $O(n)$, através de um algoritmo descrito por Ukkonen em [21]. Esse algoritmo funciona adicionando cada sufixo de p em ordem crescente de seu comprimento, ou seja, um caractere por vez, e é chamado por isso de algoritmo *On-line*. O estado inicial da árvore ST, apenas com sua raiz, representa o

sufixo vazio ϵ. Primeiramente, é construída a árvore ST_1 para o caractere p_1, depois adiciona-se o caractere p_2 para obter a árvore de sufixo ST_2 para a sequência p_1p_2, e assim sucessivamente até ser construída a árvore ST para a sequência $p_{1..n}$ a partir da árvore ST_{n-1} para $p_{1..n-1}$.

Para construir uma árvore de sufixo para duas sequências p e t, chamada de árvore de sufixo generalizada, basta concatenar $s = p \bullet \# \bullet t \bullet \$$, onde os símbolos $\#$ e \$ não pertencem ao alfabeto Σ e construir a árvore de sufixo para a sequência s.

2.2.5 Vetor de sufixo

O **Vetor de Sufixo** (do inglês, *Suffix Array*), proposto por Manber e Meyers em [23], é um vetor com os índices de todos os sufixos de uma sequência organizados de forma a expô-los em ordem lexicográfica. Mais detalhadamente, dada uma sequência $t_{1..n}$, construída sobre um alfabeto Σ, onde $t_n = \$$, e \$ não está contido em Σ e precede qualquer outro símbolo contido em Σ. O vetor de sufixo $sa[1..n]$ de t é um vetor de n números inteiros j, que representam os sufixos de t, organizados em ordem lexicográfica. Ou seja, $t_{sa[1]..n} < t_{sa[2]..n} < .. < t_{sa[n]..n}$ [24].

Em conjunto ao vetor de sufixo é comum utilizar o **Vetor de Sufixo Inverso** (do inglês, *Inverse Suffix Array*), denotado por $sa^{-1}[1..n]$, onde $sa^{-1}[i]$ representa a posição do i-ésimo sufixo no vetor de sufixo sa. Ou seja, para todo inteiro $i \in [1..n]$, definimos como $sa^{-1}[i]$ o inteiro j tal que $sa[j] = i$. Um **Vetor LCP**, usado frequentemente em conjunto ao vetor de sufixo, é um vetor que armazena o tamanho do maior prefixo comum entre dois sufixos consecutivos de sa, ou seja, $LCP[i]$ representa o LCE entre $t_{sa[i-1]..n}$ e $t_{sa[i]..n}$, formalmente $LCP[i] = LCE_{t,t}(sa[i-1], sa[i])$. Por definição, $LCP[1] = 0$ [25].

Para exemplificar, dada a sequência t = ACGTTACGTGA\$, o alfabeto $\Sigma = \{ACGT\}$, a ordem lexicográfica \$ < A < C < G < T, mostramos na Tabela 2.1 o vetor de sufixo sa de t, o vetor de sufixo inverso sa^{-1} e o vetor LCP. A coluna *Sufixo i* apresenta todos os sufixos $t_{i..n}$ para cada posição i de t e a coluna *Sufixo* $sa[i]$ apresenta todos os sufixos em ordem lexicográfica. Perceba que $sa[2] = 11$ e representa o sufixo $t_{11..12}$ = A\$ de t que, de acordo com a ordem lexicográfica, está à frente do sufixo $sa[3]$, que representa o sufixo $t_{6..12}$ = ACGTGA\$ de t. O vetor de sufixo inverso $sa^{-1}[1] = 4$ e corresponde ao sufixo $t_{1..12}$ está na posição 4 no vetor de sufixo sa, ou seja $sa[4] = 1$. Na coluna $LCP[i]$, vemos o LCE entre os sufixos $t_{sa[3]..n}$ e $t_{sa[4]..n}$ que corresponde a 4 caracteres iguais, representado pelos caracteres sublinhados, ACGT, na coluna *Sufixo* $sa[i]$.

O vetor de sufixo, o vetor de sufixo inverso e o vetor LCP, de uma sequência de tamanho n sobre um alfabeto Σ pode ser construído pelo algoritmo proposto por Larsson e Sadakane em [26], que constrói esses vetores em tempo $O(n \log n)$ no pior caso e $O(n)$ em tempo médio. Tal algoritmo particiona a sequência em subconjuntos, e os ordena recursivamente com o algoritmo *SuffixSort*, baseado no algoritmo *Quicksort* de ordenação de elementos [26]. Outro algoritmo para construir os vetores é apresentado por Aluru em [27] em tempo $O(n)$.

2.2.6 Consulta mínima de intervalo (RMQ)

Perceba que, dado um vetor LCP e um vetor de sufixo inverso sa^{-1}, para resolver o problema da maior extensão comum, precisamos encontrar o menor valor em um intervalo de valores. Assim, a solução do LCE passa a depender da solução de outro problema, conhecido como Consulta Mínima de Intervalo (do inglês, *Range Minimum Query* ou RMQ), definido da seguinte forma:

Tabela 2.1: *Um exemplo de vetor de sufixo sa, vetor de sufixo inverso sa^{-1} e vetor LCP para uma sequência $t = ACGTTACGTGA\$$.*

Posição i	**Sufixo** i	$sa[i]$	**Sufixo** $sa[i]$	$sa^{-1}[i]$	$LCP[i]$
1	ACGTTACGTGA$	12	$	4	0
2	CGTTACGTGA$	11	A$	6	0
3	GTTACGTGA$	6	ACGTGA$	9	1
4	TTACGTGA$	1	ACGTTACGTGA$	12	4
5	TACGTGA$	7	CGTGA$	10	0
6	ACGTGA$	2	CGTTACGTGA$	3	3
7	CGTGA$	10	GA$	5	0
8	GTGA$	8	GTGA$	8	1
9	TGA$	3	GTTACGTGA$	11	2
10	GA$	5	TACGTGA$	7	0
11	A$	9	TGA$	2	1
12	$	4	TTACGTGA$	1	1

> **Problema da Consulta Mínima de Intervalo** $RMQ_a(i, j)$: dado um vetor $a[1..n]$ de n elementos, e dois índices i e j, com $1 \leq i \leq j \leq n$, encontrar a posição do menor elemento no subvetor $a[i..j]$, ou seja, $RMQ_a(i, j) = min_{i \leq k \leq j}(a_k)$ [28].

Basicamente, dado um vetor qualquer, para resolver o RMQ procuramos pela posição do menor elemento entre duas posições específicas dentro do vetor. Para exemplificar o problema, dado um vetor LCP do exemplo da Tabela 2.1, o valor do menor elemento entre a posição 1 e 5 é 0 e está na posição 4, ou seja, $RMQ_{LCP}(1, 5) = 4$.

Um algoritmo ingênuo para resolver o RMQ, dado um vetor a e dois índices i e j, baseia-se em percorrer a da posição i até j, devolvendo a posição x do primeiro menor valor encontrado. Tal algoritmo poderia levar tempo proporcional ao tamanho do vetor a no pior caso, ou seja, $O(|a|)$.

Seguindo a notação proposta em [29], é dito que um algoritmo com tempo de pré-processamento $p(n)$ e tempo de consulta $q(n)$ tem a complexidade $\langle p(n), q(n)\rangle$. Dessa forma, o algoritmo descrito anteriormente teria a complexidade $\langle O(1), O(n)\rangle$ por conta de não precisar de pré-processamento.

Um algoritmo mais eficiente para resolver esse problema requer a construção de uma estrutura de dados auxiliar que permite computar o RMQ em tempo constante. Uma solução trivial, com complexidade $\langle O(n^3), O(1)\rangle$, consiste em construir uma matriz $M[1..n; 1..n]$ para armazenar todas as possíveis n^2 consultas, construída em tempo $O(n^3)$. Tal solução não é viável para grandes valores de n. Assim, embora o tempo para determinar o RMQ seja $O(1)$, tal solução não é eficiente [30, 31].

Uma outra abordagem, usando matriz de espalhamento [1], com complexidade $\langle O(n \log n), O(1)\rangle$ utiliza a ideia de pré-computar toda consulta que é potencia de dois, isto é, para todo i entre 1 e n e todo j entre 1 e $\log n$, encontramos o menor elemento no bloco iniciando em i que seja igual a 2^j, ou seja, em uma matriz $M[1..n; 1..\log n]$ computamos todos os valores $M[i, j] = argmin_{k=i..i+2^j-1}(a_k)$ [31]. Vale ressaltar que consideramos neste trabalho logaritmos tomados na base 2.

Usando a programação dinâmica, computamos os valores de $M[i, j]$ encontrando o menor valor entre dois blocos, um de tamanho 2^j e o outro de tamanho 2^{j-1}. Mais formalmente, temos a seguinte recorrência:

[1] (do inglês, *Sparse Table*) é um método de estruturar os dados de forma a facilitar a consulta aos seus elementos em tempo constante.

$$M[i,j] = \begin{cases} M[i,j-1], & \text{se } a[M[i,j-1]] \leq a[M[i+2^{j-1}-1,j-1]] \\ M[i+2^{j-1}-1,j-1], & \text{caso contrário} \end{cases}$$

Para exemplificar, apresentamos na Figura 2.8 um vetor a e o preenchimento da Matriz M na posição 2.

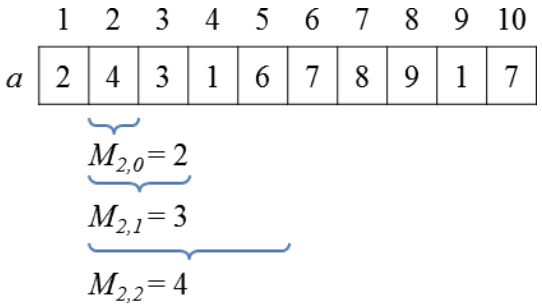


Figura 2.8: *Exemplo de construção de blocos usando matriz de espalhamento.*

Para localizar qualquer $RMQ_a(i,j)$, selecionamos dois blocos sobrepostos que cobrem inteiramente o sub-vetor $[i..j]$. Seja 2^k o tamanho do maior bloco que encaixa no intervalo de i até j, isto é, seja $k = \lfloor \log(j-i) \rfloor$. Assim, o $RMQ_a(i,j)$ pode ser determinado obtendo o mínimo entre dois blocos, usando a seguinte fórmula:

$$RMQ_a(i,j) = \begin{cases} M[i,k], & \text{se } a[M[i,k]] \leq a[M[j-2^k+1,k]] \\ M[j-2^k+1,k], & \text{caso contrário} \end{cases}$$

2.2.7 Vetor de sufixo comprimido

O **vetor de sufixo comprimido** ou *CSA* (do inglês, *Compressed Suffix Array*), proposto por Grossi e Vitter em [32] é uma estrutura de dados que reduz consideravelmente o tamanho do vetor de sufixo, ao custo de um aumento no tempo de acesso aos dados. A redução é feita por compressão, baseada no método de Burrows-Wheeler ou *BWT* (do inglês, *Burrows-Wheeler transform*) onde é possível representar um texto t com $O(n \log |\Sigma|)$ bits, codificando cada símbolo em $\log |\Sigma|$ bits [26, 32, 33].

Um vetor de sufixo comprimido para um texto t é uma estrutura de dados de indexação própria, que provê três principais funções para recuperação dos dados originais. Seja sa o vetor de sufixo de t e sa^{-1} o vetor de sufixo inverso de sa. A função $lookup(i)$, devolve $sa[i]$ em tempo $O(\log^{\epsilon} n)$ onde $0 < \epsilon < 1$, a função $inverse(i)$, devolve $sa^{-1}[i]$ em tempo $O(\log^{\epsilon} n)$ e a função $\Psi(i)$, devolve $sa^{-1}[sa[i]+1]$ em tempo $O(1)$ [32].

Algoritmos para construção dessa estrutura em tempo $O(n \log n)$ e acesso aos dados em $O(\log^{\epsilon} n)$ foram propostos por Larsson e Sadakane em [26] e Sirén em [33].

2.2.8 Árvore de sufixo comprimida

Sadakane em [34] apresenta a **Árvore de Sufixo Comprimida** ou *CST* (do inglês, *Compressed Suffix Trees*) que é considerada como uma estrutura de dados de indexação própria, de forma comprimida, que representa uma árvore de sufixo e simula suas funcionalidades convencionais [35].

A árvore de sufixo comprimida é construída com base no vetor de sufixo comprimido. Além desse, outros vetores são empregados afim de obter todas as funcionalidades da árvore de sufixo. O vetor de **parêntesis balanceados** representa os nós da árvore através de um par de parêntesis aberto e fechado "(...)", construído por um percurso em pré-ordem. Este vetor provê três funções para navegação na árvore. Seja p um ou mais símbolos armazenados nesse vetor. A função $select_p(i)$, que devolve a posição da i-ésima ocorrência de p, a função $inorder(i)$, que devolve a ocorrência de ")(", tal que $x = select_{)(}(i)$, e a função $rank_p(i)$, que devolve o número de ocorrências de p até a posição i. O **vetor de altura**, denotado por $Hgt[1..n]$, é exatamente o vetor LCP e permite a travessia na árvore das folhas até a sua raiz (travessia *bottom-up*).

Um nó v, que representa uma posição i do texto t, é obtido da seguinte forma:

$$i' = sa^{-1}[i];$$
$$v = select_{(}(i' + 1);$$

A função $depth(v)$ representa a profundidade da sequência de um nó interno v e devolve a altura de v através do vetor Hgt. A função $parent(v)$ devolve o nó w pai de v. A função $lca(v, w)$ utiliza um vetor auxiliar P' de inteiros, tal que $P'[i] = rank_{(}(i) - rank_{)}(i)$, que é preparado para determinar o RMQ. Com isso, o $lca(v, w) = parent(RMQ_{P'}(v, w) + 1)$.

Tais funcionalidade podem ser computadas teoricamente em tempo $O(\log^e n)$, onde e é uma constante entre $0 < e \leq 1$, mas que na prática pode consumir muito tempo, devido à quantidade de conversões necessárias para se obter os dados originais. A Figura 2.9 ilustra uma árvore de sufixo para o texto t = ACGTTACGTGA$ e sua representação comprimida.

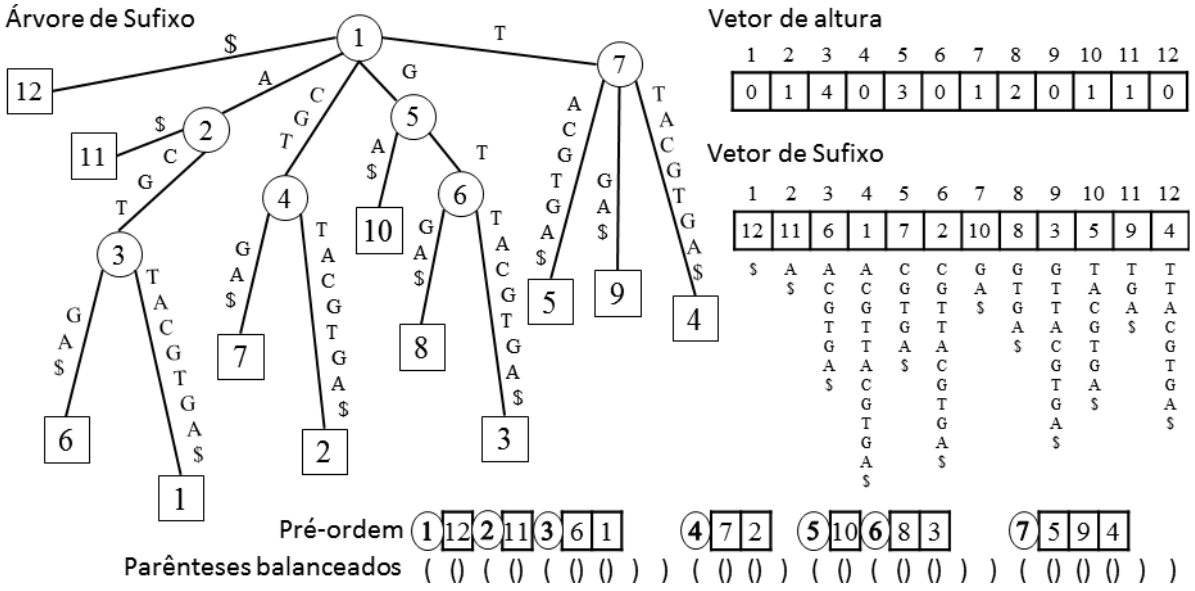


Figura 2.9: *Árvore de sufixo construída para a sequência t = ACGTTACGTGA$, e sua representação em forma de vetores.*

2.2.9 O Problema do Casamento Inexato com até k Diferenças

O Problema do Casamento Inexato com até k Diferenças, também conhecido na literatura como Problema do Casamento Inexato ou Problema do Casamento de Sequências Aproximadas,

consiste em encontrar ocorrências de uma sequência, chamada *padrão*, em outra, frequentemente maior, chamada *texto*, com até k diferenças. Para exemplificar, dadas as sequências p = ACT e t = AGTTTTC, há ocorrência de p em t terminando na posição 3 de t com uma diferença (caractere C substituído por G) e terminando na posição 4 de t com duas diferenças (caracteres AC substituídos por GT). Definimos formalmente o problema como:

> **Problema do Casamento Inexato com até k Diferenças (PCIkD):** dada uma sequência p, de tamanho m, uma sequência t, de tamanho n, e um inteiro $k \geq 1$, encontrar todas as ocorrências de p em t com até k diferenças [9].

Existem diversos algoritmos na literatura para resolver esse problema, que empregam as mais diversas técnicas, na tentativa de diminuir o uso de memória e, principalmente, o tempo de busca. Uma das soluções propostas adapta o Algoritmo 1 de forma a permitir a busca por uma sequência p em t com até k diferenças. A primeira modificação consiste em alterar a inicialização da primeira linha da matriz D, preenchendo-a com zero ($D[0, j] \leftarrow 0$, para $0 \leq j \leq n$), fazendo com que $D[i, j]$ represente agora o menor número de diferenças entre $p_{1..i}$ e qualquer segmento do texto terminando em t_j. A segunda modificação está na avaliação do resultado computado na matriz D. Após ela ser preenchida, se $D[m, j] \leq k$, para algum j entre 1 e n, então existe uma ocorrência com até k diferenças de $p_{1..m}$ finalizando em t_j. Caso contrário, pode-se afirmar que tal ocorrência não existe.

Para exemplificar o uso do algoritmo 1 adaptado na solução do PCIkD, dadas as sequências p = AATA e t = AAACGTAA, o valor $k = 2$ e a matriz $D[0..4, 0..8]$ apresentada na Tabela 2.2, observe que existem ocorrências da sequência p na sequência t com até k diferenças terminando em t_2, t_3, t_4, t_5, t_7 e t_8.

Tabela 2.2: *Exemplo de uma matriz D com inicialização modificada.*

			A	**A**	**A**	**C**	**G**	**T**	**A**	**A**
		0	1	2	3	4	5	6	7	8
	0	0	0	0	0	0	0	0	0	0
A	1	1	0	0	0	1	1	1	0	0
A	2	2	1	0	0	1	2	2	1	0
T	3	3	2	1	1	1	2	2	2	1
A	4	4	3	2	1	2	2	3	2	2

Landau e Vishkin apresentaram em [36] um algoritmo de complexidade de tempo $O(kmn)$ e de espaço $O(n)$, que utiliza a ideia de Ukkonen, descrita em [37], adaptada para a busca de padrão em texto com até k diferenças. Esse algoritmo realiza sua tarefa utilizando-se das diagonais da matriz D usada para o cálculo da distância de edição entre duas sequências. Segue abaixo os detalhes desse algoritmo.

Uma **diagonal** d da matriz D consiste de todas as células $D[i, l]$ tal que $l - i = d$. A **diagonal principal** é a diagonal 0 de D, composta das células $D[i, i]$ onde $0 \leq i \leq m \leq n$ [38]. Observe que, para células consecutivas em qualquer diagonal, a diferença entre $D[i, l]$ e $D[i-1, l-1]$ é 0 ou 1. Para um número de diferenças e e uma diagonal d, definimos uma matriz $L[-(k+1)..n+1; -1..k]$, onde $L[d, e]$ denota a *maior linha* i tal que $D[i, l] = e$ e $D[i, l]$ está na diagonal d. A definição de $L[d, e]$ implica que e é o menor número de diferenças entre $p_{1..L[d,e]}$ e qualquer segmento terminando em $t_{d+L[d,e]}$. Outra implicação é que $p_{L[d,e]+1} \neq t_{L[d,e]+d+1}$. Note que $d+L[d, e]$ corresponde exatamente

a l. Assim, para resolver o Problema do Casamento Inexato com até k Diferenças, basta procurar por valores de $L[d, e]$ que alcançam a linha m tal que $e \leq k$ [9].

A matriz L pode ser computada por indução em e, usando a definição de que $L[d, e]$ contém o valor da maior linha i de D tal que $D[i, l] = e$ e $D[i, l]$ está na diagonal d. Dadas uma diagonal d e um número de diferenças e, suponha que para todo $x < e$ e toda diagonal y, $L[y, x]$ foi corretamente computado. Suponha também que $L[d, e]$ armazena o valor i, ou seja, i é a maior linha de D tal que $D[i, l] = e$ e $D[i, l]$ está na diagonal d. Observando qualquer célula $D[i, l]$ na diagonal d, para $1 \leq i \leq m$ e $1 \leq l \leq n$, percebe-se que $D[i, l]$ foi preenchida com o valor e utilizando-se uma (ou mais) das seguintes células:

1. $D[i-1, l-1]$ (predecessor de $D[i, l]$ na diagonal d) com o valor $e-1$ e $p_i \neq t_l$;
2. $D[i, l-1]$ (predecessor de $D[i, l]$ na diagonal *"abaixo"* de d, na linha i) com o valor $e-1$;
3. $D[i-1, l]$ (predecessor de $D[i, l]$ na diagonal *"acima"* de d, na coluna l) com o valor $e-1$;
4. $D[i-1, l-1]$ (predecessor de $D[i, l]$ na diagonal d) com o valor e e $p_i = t_l$.

Isso implica que se pode iniciar em $D[i, l]$ e seguir seu predecessor na diagonal d pelo caso 4 (enquanto o valor for e) até que a primeira ocorrência do caso 1, 2 ou 3 (quando o valor for $e-1$) seja encontrada. Invertendo-se essa ideia, é possível computar $L[d, e]$ usando $L[d-1, e-1]$, $L[d, e-1]$ e $L[d+1, e-1]$ para inicializar uma variável, que é então incrementada de um em um até atingir o valor correto de $L[d, e]$. Em outras palavras, é possível computar $L[d, e]$ inicializando uma variável com o maior valor dentre $L[d-1, e-1]$, $L[d, e-1]+1$ e $L[d+1, e-1]+1$ e seguir na mesma diagonal d, incrementando a variável em uma unidade enquanto $p_{row+1} = t_{d+row+1}$ [9].

Para resolver o PCIkD, procuramos algum $L[d, e]$ igual a m tal que e seja $\leq k$, o que significa que o padrão $p_{1..m}$ ocorre no texto finalizando em t_{d+m} com e diferenças. Em caso de não existir nenhum $L[d, e] = m$ tal que $e \leq k$, pode ser afirmado que não há ocorrência de p em t com no máximo k diferenças.

Os passos da abordagem descrita acima para preenchimento da matriz L e resolução do PCIkD encontram-se detalhados no Algoritmo 2. Os valores de inicialização servem apenas para controlar os limites da matriz e não interferem no cálculo do $L[d, e]$. Vale ressaltar também que não há necessidade de analisar todas as diagonais da matriz D, com isso as diagonais $d > n - m + k + 1$ e $d < -k$ não tem utilidade alguma para a solução do problema.

Algoritmo 2 Algoritmo_K_Difference_Inexact_Match

Entrada: duas sequências p, t e um inteiro $k > 0$
Saída: todas as ocorrências de p em t com até k diferenças

```
1:  m ← |p|;
2:  n ← |t|;
3:  para d ← 0 até n faça                                   ▷ Inicialização
4:      L[d, −1] ← −1;
5:  fim_para
6:  para d ← −(k + 1) até −1 faça
7:      L[d, |d| − 1] ← |d| − 1;
8:      L[d, |d| − 2] ← |d| − 2;
9:  fim_para
10: para e ← −1 até k faça
11:     L[n + 1, e] ← −1;
12: fim_para
13: para e ← 0 até k faça                                   ▷ Execução
14:     para d ← −e até n − m + k + 1 faça
15:         row ← max(L[d, e − 1] + 1, L[d − 1, e − 1], L[d + 1, e − 1] + 1);
16:         row ← min(row, m);
17:         enquanto row < m e row + d < n e p_{row+1} = t_{row+1+d} faça
18:             row ← row + 1;
19:         fim_enquanto
20:         L[d, e] ← row;
21:         se L[d, e] = m então
22:             escreva Existe uma ocorrência de p finalizando em t_{d+m};
23:         fim_se
24:     fim_para
25: fim_para
```

Um exemplo de matriz L preenchida com base nas diagonais de D pode ser visto na Tabela 2.3. Nesse exemplo, são utilizadas as mesmas sequências do exemplo anterior, ou seja p = AATA, t = AAACGTAA e $k = 2$. A diagonal 0, com 2 diferenças, representada por $L[0, 2] = 4$, corresponde à ocorrência de $p_{1..4}$ terminando em t_{4+0}. A diagonal 5, com 1 diferença, representada por $L[5, 1] = 3$ corresponde à ocorrência de $p_{1..3}$ terminando em t_8. A diagonal -1 alcança a linha m com apenas uma diferença, ou seja, $L[-1, 1] = 4$, que corresponde à ocorrência de $p_{1..4}$ terminando em t_3. Perceba que o tamanho desta nova matriz é relativamente menor do que a matriz D.

O Algoritmo 2 proposto originalmente roda em tempo $O(kmn)$, já que, para cada uma das $k + 1$ diferenças, é computado o valor de $L[d, e]$ em $n - m + k + 1$ diagonais, e em cada diagonal, a variável row atinge, no máximo, o valor m. Landau e Vishkin propõe ainda em [36] uma nova versão do algoritmo que determina a extensão comum mais longa em tempo constante. Isso é feito utilizando-se árvore de sufixo, que permite a busca ao ancestral comum mais baixo de dois nós em tempo $O(1)$. Essa estratégia pode ser vista com maiores detalhes na seção a seguir, que apresenta também outras estruturas de dados, tais como o vetor de sufixo e a árvore de sufixo comprimida que podem ser aplicadas para obtenção, em tempo constante, do LCE.

Tabela 2.3: *Exemplo de matriz $L[d, e]$ apresentando valores computados com base nas diagonais da matriz D.*

	-1	**0**	**1**	**2**
-3	-	-	1	2
-2	-	0	1	4
-1	-1	0	4	-
0	-1	2	3	4
1	-1	2	3	4
2	-1	1	2	3
3	-1	0	1	4
4	-1	0	1	4
5	-1	0	3	-
6	-1	2	-	-
7	-1	1	-	-
8	-1	0	-	-
9	-1	-1	-1	-1

Trabalhos relacionados

Além dos algoritmos já apresentados para resolver o PCIkD, outros algoritmos foram propostos ao longo do tempo. Esses algoritmos utilizam diferentes técnicas de programação, com o emprego de autômatos e/ou uso de paralelismo [19, 39], ou foram escritos com base em outras medidas de distância, como a distância de *Hamming* [40] ou o q-grams [19]. Além disso, existem diversas estruturas de dados criadas para suportar indexação de sequências e recuperação de dados em tempo de execução [41], usadas em algoritmos mais recentes. A seguir, apresentaremos os principais algoritmos, que utilizam a técnica de programação dinâmica, encontrados na literatura. Nessa apresentação considere p (padrão) uma sequência de tamanho m, t (texto) uma sequência de tamanho n e k o número máximo de diferenças permitidos.

Em 1985, Landau e Vishkin apresentaram em [40] um algoritmo com complexidade de tempo $O(m^2 + k^2 n)$ que indexa p em uma matriz $LM[0..m-1; 0..m-1]$, de forma que $LM[i, j] = LCE_{p,p}(i, j)$, que representa dados estatísticos de casamento do padrão com ele mesmo. Esses dados são utilizados em um algoritmo de análise do texto, que consome tempo $O(k^2 n)$. No mesmo ano, eles apresentaram, em [10], o mesmo algoritmo melhorado, sem o uso da matriz LM, com complexidade de tempo $O(k^2 n)$. Posteriormente, em 1986, Landau e Vishkin apresentaram em [36] e em [42] o Algoritmo 2, mais refinado, com complexidade de tempo $O(kmn)$ e complexidade de espaço $O(n)$.

Em 1988, Galil e Giancarlo apresentam em [43] algoritmos paralelos e seriais, que empregam estruturas de dados como a árvore de sufixo, construída apenas com t, sobrepondo pedaços dessa sequência de tamanho $O(m)$, com complexidade de tempo igual ao dos algoritmos de Landau e Vishkin, mas lentos na prática, conforme os próprios autores concluem.

Em 1989, Galil e Park apresentam em [44] um algoritmo com complexidade de tempo $O(kn)$ e complexidade de espaço $O(m^2)$, baseado na ideia de Landau e Vishkin [40] de análise de t, utilizando a matriz LM com dados estatísticos de p. A principal ideia é computar três diagonais vizinhas da matriz L ao mesmo tempo, fazendo uma cuidadosa verificação das diagonais mais relevantes, ou seja, daquelas que representam uma igualdade entre o padrão e o texto. Em 1990, Chang e Lawler em [45] substituíram a matriz LM por uma árvore de sufixo e um algoritmo de determinação do

ancestral comum mais baixo em tempo constante, reduzindo o espaço extra $O(m^2)$ para $O(m)$.

Takaoka, em 1994, apresenta em [46] uma simplificação do algoritmo de Chang e Lawler adotando o conceito de amostragem, do inglês *sampling*, depois chamado de q-sample. A ideia é tomar pequenas amostras espaçadas ao longo do texto e, com a ajuda de um pré-processamento de p, descartar de forma rápida várias posições não candidatas ao casamento aproximado entre p e t. A complexidade de tempo desse algoritmo é de $O((kn^2/m^2)\log m)$.

Publicado em 1998, mas apresentado pela primeira vez em um documento técnico no ano de 1986, Myers em [47] propõe um algoritmo que compartilha a mesma ideia de computar novas diagonais usando as diagonais prévias já computadas, de forma incremental. Analisando cuidadosamente o valor encontrado na diagonal de maior alcance, ou seja, o valor h, resultado da computação da primeira diagonal que obtém o valor n ou m, ele conclui que apenas as diagonais entre -h e h precisam ser computadas, já que fora delas tal valor não é encontrado. Dessa forma, a computação das diagonais é realizada incrementalmente, chamadas h-ondas, em cada diagonal d entre $-h$ e h. Tal algoritmo necessita de espaço extra $O(n)$ e roda em tempo $O(kn)$.

Em 2006, Miranda apresenta em [38] uma variante do algoritmo de Landau e Vishkin que substitui a árvore de sufixo usada na determinação da maior extensão comum por vetores de sufixo, que também permite a determinação da maior extensão comum em tempo constante.

Na Tabela 2.4 listamos os principais algoritmos desenvolvidos ao longo do tempo, agrupados pelas técnicas empregadas, mostrando o autor e o artigo de referência.

Tabela 2.4: *Principais algoritmos publicados para resolver o PCIkD utilizando a técnica de programação dinâmica.*

Ano	**Computação direta**	**Pré-Processamento**		
		Padrão	**Texto**	**Padrão e Texto**
1985	Landau e Vishkin [10]	Landau e Vishkin [40] Matriz ML		
1986	Landau e Vishkin [9, 36]			Landau e Vishkin [9, 36] Árvore de Sufixo
1988		Landau, Vishkin e Uzi [42]	Galil e Giancarlo [43] Árvore de Sufixo	
1989		Galil e Park [44] Matriz ML + Árvore de Sufixo		
1990		Chang e Lawler [45] Árvore de Sufixo		
1994		Takaoka [46]		
1998		Myers [47]		
2006				Miranda [38] Vetor de Sufixo

Estruturas de dados na fase de pré-processamento

Na parte central do Algoritmo 2 (linhas 17 e 18), é realizada a computação da maior extensão comum entre p e t de forma iterativa, ou seja, incrementando a variável row enquanto p_{i+row} seja igual a t_{j+row} e row é menor que os tamanhos de p e t. Tais passos, claramente, tomam tempo $O(m)$.

Contudo, é possível determinar o valor de $LCE_{s,t}(i,j)$ em tempo $O(1)$ empregando uma estrutura de dados capaz de armazenas todos os resultados possíveis e devolvê-los, sob consulta em tempo constante. Tais estruturas de dados, como *Árvores de Sufixo* [2, 7] e *Vetores de Sufixo* [38], são construídas em tempo hábil e permitem a determinação da maior extensão comum em tempo inferior ao do algoritmo iterativo. Quando utilizadas na solução de um problema maior, é necessária uma fase de pré-processamento para construir tais estruturas de dados. A Tabela 2.5 a seguir apresenta as principais estruturas de dados usadas para esse fim e que foram selecionadas para os testes neste trabalho.

Tabela 2.5: *Estruturas de dados com seus tamanhos, tempo de construção e tempo de consulta.* Σ *representa o alfabeto finito,* n *o tamanho da sequência, e um valor variando de 0 até 1 e,* v *a quantidade de nós da árvore.*

Estrutura	**Tamanho**	**Tempo de Construção**	**Tempo de Consulta (*Query*)**
Arranjos de sufixo	$4n + \log n$	$O(n \log n)$	$O(1)$
Árvore de sufixo	$16n + 20v$	$O(n)$	$O(1)$
Árvore de sufixo comprimida	$(8n \log \lvert\Sigma\rvert)$	$O(n)$	$O(\log^e n)$

Para computar a maior extensão comum através de uma árvore de sufixo, precisamos construir uma árvore de sufixo generalizada ST para as sequências p e t. Após sua construção, é possível determinar o $LCE_{p,t}(i,j)$ encontrando o ancestral comum mais baixo dos nós x e y, que representam os índices i e j. Na verdade, a profundidade do ancestral comum mais baixo dos nós x e y corresponde ao $LCE(i,j)$, ou seja, se $w = LCA(x,y)$ então o nível de w é exatamente o $LCE_{p,t}(i,j)$. Perceba que o nó w representa indiretamente o $LCP_{p,t}(i,j)$, pois o segmento de caminho do nó raiz até o nó w é exatamente o $LCP_{p,t}(i,j)$. Assim o $LCE_{p,t}(i,j) = profundidade(LCA(x,y))$, onde o nó x representa o sufixo $p_{i..n}$ e o nó y representa o sufixo $t_{j..n}$.

É possível obter o $LCA(x,y)$ em tempo $O(1)$ de forma direta, deslocando bits dos rótulos dos nós folhas, que representam as posições i de p e j de t. Em [2, 22] podemos encontrar maiores detalhes sobre como construir a árvore de sufixo e obter o LCA de quaisquer dois nós da árvore em tempo $O(1)$, além da descrição em detalhes sobre a etapa de pré-processamento para permitir obter o LCE em tempo constante.

Para computar o $LCE_{p,t}(i,j)$ utilizando um vetor de sufixo, é necessário construir o vetor de sufixo, o vetor de sufixo inverso e o vetor LCP para duas sequências, p e t, concatenando-as com dois símbolos, $\#e\$$, não pertencentes ao alfabeto Σ, tal que, $s = p \bullet \# \bullet t \bullet \$$. Construído o vetor LCP, é possível determinar o $LCE_{p,t}(i,j)$ de p e t, buscando o menor valor no intervalo desse vetor que vai da posição $sa^{-1}[i]+1$ a $sa^{-1}[j]$, se $sa[i] < sa[j]$, ou seja, $LCE_{p,t}(i,j) = minlcp(sa^{-1}[i]+1, sa^{-1}[j])$. Para exemplificar, seja p = TGCAT, t = ACTGGCG e s = TGCT#ACTGGCG\$. A Tabela 2.6 apresenta os vetores necessários para computar o $LCE_{p,t}(i,j)$.

Usando a Tabela 2.6 vamos exemplificar como encontrar o LCE de $p_{3..5}$ e $t_{3..7}$, onde $i = 3$ e $j = 3$. Note que é necessário somar a j o tamanho da sequência $p + 1$, para referenciar a posição correspondente na sequência concatenada s, tal que, $j = j + |p| + 1$ e com isso, $j = 3 + 5 + 1$.

Tabela 2.6: *Exemplo de vetor de sufixo, vetor de sufixo inverso e vetor LCP para duas sequências.*

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
s	T	G	C	A	T	#	A	C	T	G	G	C	G	$
sa	14	6	7	4	3	12	8	13	2	11	10	5	1	9
sa^{-1}	13	9	5	4	12	2	3	7	14	11	10	6	8	1
LCP	0	0	0	1	0	1	1	0	1	2	1	0	1	2

$$\begin{aligned} LCE_{p,t}(3,3) &= Minlcp(sa^{-1}[3]+1, sa^{-1}[9]) \\ &= Minlcp(5+1, 14) \\ &= Minlcp(6, 14) \\ &= 0 \text{ (na posição 8)} \end{aligned}$$

Um algoritmo ingênuo para determinar o LCE através do vetor LCP e do vetor sa^{-1} é apresentado no Algoritmo 3, que recebe tais vetores e dois índices i de p e j de t, como parâmetros, devolvendo o menor valor encontrado no intervalo, que representa o $LCE_{p,t}(i,j)$, em tempo $O(n)$.

Algoritmo 3 Algoritmo_DirectMin

Entrada: vetores sa^{-1}, LCP e dois índices i e j
Saída: O tamanho do LCE de i e j

```
1: low ← min(sa^-1[i], sa^-1[j]);
2: high ← max(sa^-1[i], sa^-1[j]);
3: t ← LCP[low + 1]
4: para k ← low + 2 até high faça
5:     se LCP[k] < t então
6:         t ← LCP[k];
7:     fim_se
8: fim_para
9: devolva t
```

Para resolver o problema do LCE através do RMQ, após construir a matriz de espalhamento de um vetor LCP, temos que $LCE_{p,t}(i,j) = RMQ_{LCP}(sa^{-1}[i]+1, sa^{-1}[j])$ [25]. O algoritmo é apresentado em Algoritmo 4.

Algoritmo 4 Algoritmo_RMQ

Entrada: vetores sa^{-1}, RMQ e dois índices i e j
Saída: O LCE de i e j

```
1: low ← min(sa_i^-1, sa_j^-1);
2: high ← max(sa_i^-1, sa_j^-1);
3: devolva RMQ(low + 1, high)
```

É possível computar o $LCE_{p,t}(i,j)$ em uma árvore de sufixo comprimida, de forma direta por meio das funções $depth$ e $lca(x,y)$, de forma que $LCE_{p,t}(i,j) = depth(lca(v,w))$.

Capítulo 3

O Problema da Seleção de Segmentos Específicos

Neste capítulo será abordado o problema biológico que serve de motivação para o desenvolvimento deste trabalho, a saber, o Problema da Seleção de Segmentos Específicos, assim como o problema computacional que o modela, denominado Problema do *Primer* com k Diferenças. Sobre esse último problema, apresentaremos sua definição formal e algumas abordagens para resolvê-lo.

3.1 Motivação e definição do problema

Em laboratórios de biologia molecular, a PCR é uma técnica básica que permite aos biólogos realizar uma ampla variedade de procedimentos, dentre eles a detecção e o diagnóstico de doenças infecciosas e hereditárias. Por ser um método direto e de alta sensibilidade, ele é capaz de identificar patógenos (bactéria, fungo, parasita), fornecendo informações precisas de tipo, quantidade e presença na amostra analisada. Identificar o tipo correto de patógeno permite um tratamento seletivo e precoce em casos de doenças cujos sintomas ainda não se manifestaram, evitando o uso indevido de medicamentos e até mesmo a morte do organismo.

Como já visto, a técnica da PCR é essencialmente utilizada para replicar regiões específicas de sequências de DNA e tendo uma amostra do sangue de um espécime (animal, planta ou micro-organismo), é possível diagnosticar se há infecção no espécime e identificar qual é o agente causador da infecção [48].

Quando um espécime é infectado, células do agente infeccioso podem ser encontradas em seu fluído. Logo, é possível encontrar pelo menos dois DNAs no fluído do espécime: o seu próprio e o do agente infeccioso. É possível extrair os DNAs das células por meio de técnicas específicas. Para identificar a infecção é necessário amplificar uma região específica do DNA do agente infeccioso para possibilitar sua visualização em gel de agarose. Assim, é necessário conhecer uma região específica na sequência de DNA do agente infeccioso, ou seja, uma região que não seja igual a outra região de outro DNA presente no fluído.

Dado o que foi exposto, a tarefa de seleção de segmentos específicos do DNA alvo tem um papel fundamental no sucesso da PCR e é imprescindível para se evitar o desperdício de tempo e recursos envolvidos no processo. Na prática, ela é uma tarefa difícil de ser executada por conta do tamanho das sequências e da quantidade de possibilidades, ou seja, de segmentos de uma sequência. Ademais,

é necessário que o segmento não seja somente diferente, mas que tenha pelo menos uma certa quantidade de diferenças em relação a qualquer outro segmento de outras sequências. Informalmente, tendo em mãos duas ou mais sequências de DNA, onde uma delas representa o DNA do agente patógeno e será usada como sequência alvo, precisamos encontrar o menor segmento dessa sequência que tenha k diferenças em relação a qualquer segmento das outras sequências. Esse segmento, se encontrado, é considerado um segmento específico. Formalmente, o problema biológico de seleção de segmentos específicos pode ser definido como segue:

> **O Problema da Seleção de Segmentos Específicos**: dadas duas ou mais sequências de DNA, encontrar o menor segmento em uma delas que tenha pelo menos k diferenças com relação a todos os segmentos das outras sequências.

A solução deste problema pode ser utilizada da seguinte forma:

- como região alvo a ser replicada; o segmento específico serve como modelo para o desenho de *primers* afim de se obter o *primer forward* e o *primer reverse*, e o tamanho do segmento representa o tamanho do produto (pb) da PCR.
- como sonda; o segmento específico pode ser um candidato a sonda.
- como *primer*; o segmento específico pode ser um candidato a *primer*.

O Problema da Seleção de Segmentos Específicos pode ser modelado pelo problema computacional que será apresentado a seguir: o Problema do *Primer* com k Diferenças [2].

3.2 O Problema do *Primer* com k Diferenças

Informalmente, no Problema do *Primer* com k Diferenças, dadas duas sequências, queremos encontrar segmentos pequenos de uma delas que tenham pelo menos k diferenças com relação a qualquer segmento da outra sequência. Tal problema pode ser definido formalmente da seguinte forma:

> **Problema do *Primer*[1] com k Diferenças (PPkD):** dadas duas sequências α e β e um inteiro $k > 0$ encontrar, para cada índice j de α, o menor segmento γ de α que se inicia em j e que tenha pelo menos k diferenças com relação a qualquer segmento de β [2].

Como exemplo de entrada e saída do PPkD, dadas duas sequências α = CCCGGCCC, β = CCCGTGCCC e um valor $k = 1$, na posição 1 de α teríamos o resultado γ = CCCGG, na posição 2 o resultado γ = CCGG, na posição 3 o resultado γ = CGG, na posição 4 o resultado γ = GG e, da posição 5 até a posição 8 a resposta é ε. Já para o parâmetro $k = 2$ a resposta é ε para todas as posições de α, pois não há nenhum segmento de α que tenha pelo menos 2 diferenças com relação a β[2].

[1] Optamos por manter o nome original do problema definido por Gusfield em [2], mas entendendo que sua solução possibilita também, obter segmentos específicos.

[2] Para uma melhor leitura do texto, a partir daqui, ao dizermos que um segmento possui pelo menos k diferenças com relação a β, estamos nos referindo a todos os segmentos dessa sequência

3.3 Abordagens para solucionar o problema

Para resolver o PPkD basta utilizarmos os algoritmos apresentados no Capítulo 2 para resolver o PCIkD invertendo-se o ponto de vista associado aos seus resultados. Isto é, ao invés de procurar por ocorrências de casamentos inexatos, recusamos todos os resultados que tiverem menos que k diferenças. Ou seja, ignoramos qualquer índice j de α que não seja o início de um segmento com pelo menos k diferenças com relação a β.

De forma mais detalhada, o método baseado na solução do PCIkD para resolver o PPkD consiste em três passos. No primeiro passo, para cada índice j de α, com $1 \leq j \leq (m-k)$, passamos os parâmetros $\alpha_{j..m}$ como p, β como t e k. No segundo passo, executamos um dos dois algoritmos que resolvem o PCIkD e, no terceiro passo, analisamos a matriz de programação dinâmica resultante da execução do algoritmo para encontrar o menor prefixo da sequência $\alpha_{j..m}$ (se existir) que tenha distância de edição de pelo menos k com relação à β.

Chamaremos de **abordagem 1** a análise de dados da matriz D, resultante da execução do Algoritmo 1 devidamente modificado para resolver o PCIkD, e de **abordagem 2** a análise de dados da matriz $L[d, e]$, resultante da execução do Algoritmo 2. Vale destacar que, nessa abordagem, a quantidade de diagonais d analisadas pelo Algoritmo 2, na linha 14, deve ser alterada para variar de $-e$ até n e assim determinar corretamente o tamanho do segmento [7]. Cada abordagem possui tempo de execução e uso de memória diferente, mas seguem a mesma estratégia, definida nos três passos descritos anteriormente.

As abordagens descritas encontram-se detalhadas a seguir. Nelas, a ocorrência de segmento específico para cada posição j da sequência α é representada por uma determinada linha, (da matriz D ou L) devolvida ao final da execução de uma das abordagens. Esse número corresponde ao tamanho do segmento específico. Caso essa linha não exista para um certo j, o valor devolvido será -1 e o processo é interrompido pois, se não houve ocorrência de segmento específico para essa posição de α, também não haverá para as posições seguintes. A versão final do algoritmo para resolver o PPkD está descrito no Algoritmo 5.

Algoritmo 5 Algoritmo_k_Difference_Primer

Entrada: duas sequências α e β, e um inteiro k
Saída: Todas as ocorrências r de *primer*

1: $m \leftarrow |a|$; ▷ Inicialização
2: **para** $j \leftarrow 1$ **até** $m - k$ **faça** ▷ Execução
3: $r \leftarrow$ (Abordagem1($\alpha_{j..m}, \beta, k$) ou Abordagem2($\alpha_{j..m}, \beta, k$);
4: **se** $r \neq -1$ **então**
5: **Imprimir** $j; r; \alpha_{j..j+r}$
6: **senão**
7: **Terminar**;
8: **fim_se**
9: **fim_para**

3.3.1 Abordagem 1

Dada a matriz D computada pelo Algoritmo 1, a abordagem 1 consiste em percorrer todas as linhas de D, a partir da linha k, procurando pela primeira linha r da matriz D tal que $D[r, l] \geq k$ para todo $1 \leq l \leq n$, e devolver $\gamma = p_{1..r}$ se essa linha existir. Se $r = m$ e houver algum valor

menor que k nesta linha, pode-se afirmar que não existe um segmento de α começando em j com pelo menos k diferenças com relação a β.

Para exemplificar, vamos utilizar as mesmas entradas dadas no exemplo anterior, ou seja, α = CCCGGCCC, β = CCCGTGCCC, $k = 1$ e a execução para a posição $j = 4$. Neste caso, a abordagem 1 recebe como parâmetros p = GGCCC, t = CCCGTGCCC e $k = 1$. A matriz $D[0..6; 0..10]$ resultante do Algoritmo 1 modificado encontra-se representada na Tabela 3.1. Nela, podemos ver o menor número de diferenças entre $p_{1..i}$ e qualquer segmento do texto terminando em t_l para todo i e l. Seguindo a abordagem 1, observamos que na linha 1 não é a linha r, pois $D[1, 4] = 0$ e $D[1, 6] = 0$, valores menores que k. Já a linha 2 é uma linha r, pois todas as células dessa linha tem valores maior ou igual a k. Logo, se $r = 2$ então a ocorrência de segmento específico é GG, ou seja, $\gamma = p_{1..2}$.

Tabela 3.1: *Exemplo de matriz D para encontrar ocorrências.*

			C	**C**	**C**	**G**	**T**	**G**	**C**	**C**	**C**
		0	1	2	3	4	5	6	7	8	9
	0	0	0	0	0	0	0	0	0	0	0
G	1	1	1	1	1	0	1	0	1	1	1
G	2	2	2	2	2	1	1	1	1	2	2
C	3	3	2	2	2	2	2	2	1	1	2
C	4	4	3	2	2	3	3	3	2	1	1
C	5	5	4	3	2	3	4	4	3	2	1

Se α tem tamanho m e β tem o tamanho n, então o PPkD pode ser resolvido, adaptando o Algoritmo 1, em tempo $O(m^2n)$ e uso de memória $O(n)$. A versão final do algoritmo da abordagem 1 é apresentada no Algoritmo 6.

3.3.2 Abordagem 2

Dada a matriz $L[d, e]$ computada pelo Algoritmo 2, com a quantidade de diagonais d devidamente estendida para o intervalo $-k \leq d \leq n$, a abordagem 2 consiste primeiramente em percorrer todas as células $L[d, e]$, tal que $e < k$, procurando pelo valor m. Se existe algum $L[d, e]$ tal que $e < k$ com o valor m, isso significa que existe uma ocorrência de $p_{1..m}$, terminando em $t_{L[d,e]+d}$, com menos que k diferenças, e pode ser afirmado que não existe um segmento de α começando em j com pelo menos k diferenças com relação à β. Essa verificação simples pode ser adicionada ao Algoritmo 2 enquanto a matriz é preenchida, após a computação da variável row. Se $row = m$ e $e < k$, então o processamento pode ser interrompido pois é certo que não há ocorrência. Caso não exista nenhum par de valores d e e tais que $L[d, e] = m$ e $e < k$, pode-se concluir que há ocorrência de segmento específico.

Nesse caso, percorremos a coluna $k - 1$ de L no intuito de obter a maior linha r armazenada em algum $L[d, k - 1]$, para $-k \leq d \leq n$, incrementá-la em uma unidade fazendo-se $r = r + 1$ e devolver $\gamma = p_{1..r}$. Isso funciona pois cada $L[d, e]$ representa uma ocorrência de $p_{1..L[d,e]}$ finalizando em $t_{d+L[d,e]}$, com exatamente e diferenças. Por definição, sabemos que $p_{L[d,e]+1} \neq t_{L[d,e]+d+1}$, logo $p_{1..L[d,e]+1}$ tem $e + 1$ diferenças com relação a qualquer segmento finalizando em $t_{L[d,e]+d+1}$.

Para exemplificar, definimos α = CAACAC, β = CGTCCTGCC, $k = 3$ e a posição $j = 1$. Vamos executar a Abordagem 2, que recebe os parâmetros p = CAACAC ($m = 6$), t = CGTCCTGCC ($n = 9$) e $k = 3$. A matriz $L[d, e]$ resultante da execução do Algoritmo 2 modificado encontra-se

Algoritmo 6 Algoritmo_Abordagem_1

Entrada: duas sequências a e b, e um inteiro $k > 0$
Saída: linha r

```
1:  m ← |a|;                                                    ▷ Inicialização
2:  n ← |b|;
3:  r ← −1;
4:  para l ← 0 até n faça
5:      D'[l] ← 0;
6:  fim_para
7:  para i ← 1 até m e r = −1 faça                              ▷ Execução
8:      pivo ← i;
9:      passou ← VERDADEIRO;
10:     para l ← 1 até n faça
11:         se a_i = b_l então α ← 0;
12:         senão α ← 1;
13:         fim_se
14:         aux ← min(D'[l] + 1, pivo + 1, D'[l − 1] + α);
15:         D'[l − 1] ← pivo;
16:         pivo ← aux;
17:         se passou = VERDADEIRO e pivo < k então passou ← FALSO;
18:         fim_se
19:     fim_para
20:     D'[n] ← aux;
21:     se passou = VERDADEIRO então r ← i;
22:     fim_se
23: fim_para
24: devolva r;
```

representada na Tabela 3.2. Nela, observamos primeiramente que nenhum $L[d, e]$ tem o valor m nas colunas 0, 1 e 2 e portanto há ocorrência obtida procurando pelo maior valor na coluna $k - 1$, que é 4, nas diagonais 0, 1, 4 e 5. Assim, $r = 4 + 1$ e $\gamma = p_{1..5}$, ou seja, γ = CAACA.

Se α tem tamanho m e β tem o tamanho n, então o PPkD pode ser resolvido em tempo $O(km^2n)$ e espaço $O(nk)$, utilizando o algoritmo ingênuo para determinação do LCE. Quando o LCE entre duas sequências é determinado em tempo constante, utilizando uma estrutura de dados como árvores de sufixo ou vetor de sufixo, o tempo total do algoritmo passa a ser $O(kmn)$ [2]. A versão final do algoritmo da abordagem 2 é descrito no Algoritmo 7.

3.4 Implementações utilizadas

Procuramos implementar os algoritmos mais conhecidos para resolver o PPkD até o momento. Ao todo, foram implementados cinco programas diferentes baseados nos algoritmos para resolver o PPkD. São quatro programas baseados na abordagem 2, onde cada um deles tem um algoritmo diferente que determina o LCE entre duas sequências, e um programa baseado na abordagem 1. Atribuímos a sigla KDP, de *k-difference primer*, seguida de um número sequencial, para cada implementação.

Todos os programas foram implementados na linguagem C++, com base em [49, 50, 51], que permite uma estruturação de classes e reaproveitamento de objetos. Com relação às estruturas de

Tabela 3.2: *Exemplo de matriz $L[d, e]$ para encontrar ocorrências.*

	-1	**0**	**1**	**2**	**3**
-4	-	-	-	2	3
-3	-	-	1	2	4
-2	-	0	1	3	4
-1	-1	0	2	3	6
0	-1	1	2	4	5
1	-1	0	1	4	5
2	-1	0	2	3	4
3	-1	1	2	3	6
4	-1	1	2	4	5
5	-1	0	1	4	-
6	-1	0	2	3	-
7	-1	1	2	-	-
8	-1	1	-	-	-
9	-1	0	-	-	-
10	-1	-1	-1	-1	-1

dados mais complexas, no caso as árvores de sufixo, árvores de sufixo comprimidas e vetores de sufixo, fizemos uso de bibliotecas prontas, de outros autores, que em geral costumam ser implementações eficientes, testadas e estáveis, prontas para serem utilizadas. A Tabela 3.3 apresenta os cinco programas desenvolvidos com detalhes sobre eles.

Tabela 3.3: *Diferentes implementações realizadas*

Nome	**Algoritmo**	**Estrutura de dados**	**Complexidade**	
			Tempo	**Espaço**
KDP1	Abordagem 1	-	$O(m^2n)$	$O(n)$
KDP2	Abordagem 2	-	$O(km^2n)$	$O(k+n)$
KDP3	Abordagem 2	Árvore de sufixo comprimida	$O(kmn \log n)$	$O(k+n+\|CST\|)$
KDP4	Abordagem 2	Vetor de sufixo	$O(kmn)$	$O(k+n+4n \log n)$
KDP5	Abordagem 2	Árvore de sufixo	$O(kmn)$	$O(k+n+\|ST\|)$
Observações: m representa o tamanho da sequência α, n o tamanho da sequência β, k é a quantidade de diferenças, CST corresponde a $O(n \log \|\Sigma\|) + 6n + o(n)$ e ST corresponde a $20v + 4n + 12n$, onde v representa o número de nós da árvore.				

3.4.1 Organização estrutural

A linguagem de programação C++ permite o desenvolvimento orientado a objetos. Como todos os programas compartilham um conjunto de funcionalidades, utilizamos um agrupamento de dados que são comuns a todos os programas, em forma de classes hierárquicas, como biblioteca base, que permitiu a abstração dos dados e funcionalidades gerais e a implementação do comportamento de cada algoritmo diretamente no programa. Isso facilitou a manutenção, a padronização de código, a forma de uso de todas os programas e, principalmente, permitiu avaliar apenas a diferença entre as abordagens e as estruturas de dados empregadas. Na Figura 3.1 é apresentado o diagrama de classes que, representa a organização de classes da biblioteca `classes.h`, utilizada em todas os programas implementados. As próximas seções tratarão de cada implementação utilizada durante o desenvolvimento dos programas.

Algoritmo 7 Algoritmo_Abordagem_2

Entrada: duas sequências a, b e um inteiro $k > 0$
Saída: linha r

```
1:  m ← |a|;                                                    ▷ Inicialização
2:  n ← |b|;
3:  passou ← VERDADEIRO;
4:  para d ← −(k + 1) até n + 1 faça
5:      L'[d] ← −1;
6:  fim_para
7:  para e ← 0 até k − 1 e passou = VERDADEIRO faça             ▷ Execução
8:      pivo ← −1;
9:      L'[−e − 1] ← e;
10:     L'[−e − 2] ← e;
11:     para d ← −e até n e passou = VERDADEIRO faça
12:         row ← max(L'[d] + 1, L'[d − 1], L'[d + 1] + 1);
13:         row ← min(row, m);
14:         row ← row + LCE(a_row, b_row+d);
15:         se row = m então passou ← FALSO;
16:         fim_se
17:         L'[d − 1] ← pivo;
18:         se row > r então r ← row;
19:         fim_se
20:         pivo ← row;
21:     fim_para
22:     L'[d] ← row;
23: fim_para
24: se passou = VERDADEIRO então r ← r + 1;
25: senão
26:     r ← −1;
27: fim_se
28: devolva r;
```

3.4.2 Árvore de sufixo comprimida

A implementação de árvore de sufixo comprimida utilizada no programa KDP3 foi aquela disponibilizada pelo grupo de pesquisa SuDS Project, do Departamento de Ciência da Computação da Universidade de Helsinki (https://www.cs.helsinki.fi/group/suds/cst/). Publicada em [35], desenvolvida em C++ e distribuída como biblioteca na versão 1.1, e ela implementa o algoritmo proposto por Sadakane em [34] com suporte a todas as funcionalidades típicas oferecidas por uma árvore de sufixo.

Não foi necessário nenhum código adicional para utilizar a biblioteca, que já fornece uma função própria para computar a maior extensão comum entre duas sequências. Foi necessário apenas a configuração inicial, no arquivo `configuration.h`, do tamanho da palavra que o sistema operacional suporta, W = 32 *bits* ou W = 64 *bits*. Como os testes foram executados em um processador de 64 *bits*, o tamanho da palavra foi configurada em W = 64. O espaço utilizado pelos dados desta estrutura é extremamente comprimido, com tamanho $O(n \log |\Sigma|) + 6n + o(n)$

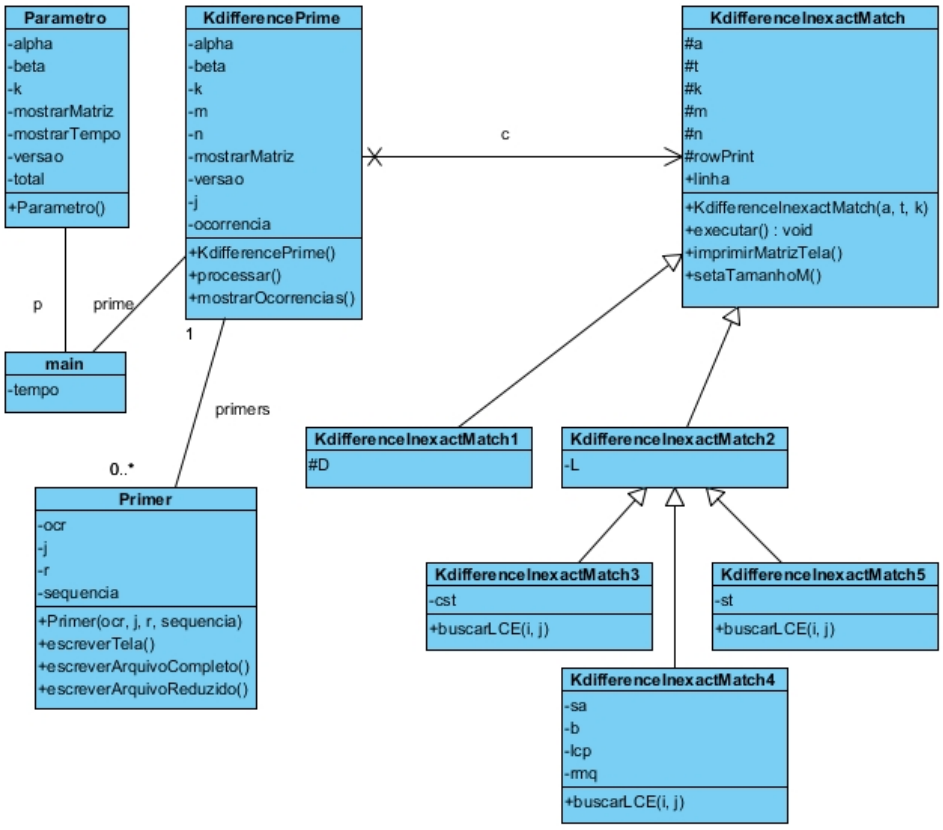


Figura 3.1: *Diagrama de classes da estrutura organizacional dos programas implementados.*

3.4.3 Vetor de sufixo

No programa KDP4, a implementação do vetor de sufixo utilizada foi aquela disponível no repositório https://github.com/chaemon/library/tree/master/string. Ela utiliza o algoritmo proposto por Larson e Sadakane [26] para a construção do vetor de sufixo, o algoritmo de Kasai *et al.* [52] para construção do vetor LCP e o algoritmo do RMQ. Foi necessária apenas a implementação do Algoritmo 4 para computar o maior extensão comum entre duas sequências em tempo constante. O tamanho total da estrutura de dados é $3n + O(n) + O(n \log n)$.

3.4.4 Árvore de sufixo

No programa KDP5, utilizamos a biblioteca *strmat*, disponibilizada em http://web.cs.ucdavis.edu/~gusfield/strmat.html por Dan Gusfield, que implementa o algoritmo de Unkkonen para construção da árvore de sufixo, e é preparada para a determinação da maior extensão comum entre duas sequências em tempo constante. Originalmente a biblioteca é disponibilizada na linguagem C e foi necessário algumas adaptações técnicas para poder integrar a biblioteca ao programa escrito em C++. Nenhuma correção ou mudança nos algoritmos originais, da biblioteca, foi necessário. O tamanho total do uso desta estrutura de dados para alfabetos pequenos (DNA e RNA), é, teoricamente, por volta de $20v + 4n + 12n$, onde n representa o tamanho da sequência p e v representa o número de nós da árvore T.

Tabela 3.4: *Tabela de parâmetros criados para usar os programas*

Nome	Descrição	Exemplo de uso	obrig.
-a	Indica a sequência alfa: arquivo ou sequência de caracteres.	`-a CCCGGCCC` `-a arquivo`	SIM
-b	Indica a sequência beta: arquivo ou sequência de caracteres.	`-b CCCGTGCCC` `-b arquivo`	SIM
-k	Quantidade de diferenças desejada.	`-k 10`	SIM
-vs[%]	Número da versão do programa, apenas KDP1 e KDP2. 1: Matriz D (KDP1) e L (KDP2) com uma linha (Padrão); 2: Matriz D (KDP1) com duas linhas e L (KDP2) completa; 3: Matriz D (KDP1) completa.	`-vs1`, `-vs2` ou `-vs3`	NÃO
-sm	Mostrar a matriz de programação dinâmica na tela, independente da versão.	`-sm`	NÃO
-st	Mostrar o tempo de execução do algoritmo.	`-st`	NÃO
-ts[%]	Tipo de saída de resultados: 1 = detalhada, 2 = completa e 5 = simples (padrão). detalhada: posição j; tamanho do segmento; completo: posição j; tamanho do segmento; segmento específico. simples: posição j; tamanho do segmento	`-ts1` ou `-ts2`	NÃO
-sf	Nome do arquivo de saída: nome da pasta e arquivo em que será salvo os resultados; (criação automático na pasta `saida` com a seguinte nomenclatura: `aM_bN_k_nome_programa`).	`-sf arquivo_saida`	NÃO
-log	Mostrar detalhes de etapas de processamento de dados.	`-log`	NÃO

3.4.5 Compilação e uso

Para compilar os arquivos fontes `kdifferenceprime1.cpp`, `kdifferenceprime2.cpp` e `kdifferenceprime4.cpp`, referente aos programas KDP1, KDP2 e KDP4, utilizamos a instalação padrão do gcc para o sistema operacional Debian Linux, executando-o com os parâmetros `-O3` e `-std=c++11`. O parâmetro `-O3` é necessário para ganho de performance durante a compilação e o parâmetro `-std=c++11` é necessário para indicar ao compilador a versão 11 da biblioteca std que foi utilizada no código. Para compilar os arquivos fontes `kdifferenceprime3.cpp` e `kdifferenceprime5.cpp`, referentes aos programas KDP3 e KDP5, foi necessário a criação dos arquivos `makefilek3` e `makefilek5`, por conta do uso de bibliotecas externas. Tais arquivos, para serem compilados, necessitam do comando make com o parâmetro `-f`. Ressaltamos que o arquivo fonte `kdifferenceprime3.cpp` depende da biblioteca encontrada na pasta `cst_v_1_1`, o arquivo `kdifferenceprime4.cpp` depende da biblioteca `rmq-sa.h`, e o arquivo `kdifferenceprime5.cpp` depende da biblioteca encontrada na pasta `strmat`. Todos os arquivos fontes dependem da biblioteca `classes.h`.

Para facilitar o uso e testes dos aplicativos, criamos diversos parâmetros. A Tabela 3.4 mostra todos os parâmetros criados, bem como uma descrição detalha do uso de cada um deles.

Capítulo 4

Análise Experimental e Resultados

Como análise experimental, avaliamos o tempo e espaço utilizados pelos programas que implementam os algoritmos 6 e 7, apresentados no Capítulo 3. A primeira bateria de testes foi realizada com dados artificiais, gerados de forma aleatória, com base no alfabeto $\Sigma = \{A, C, G, T\}$, que corresponde ao alfabeto utilizado na representação de DNAs. A segunda bateria de testes foi realizada com sequências correspondentes de genes específicos de diversos seres. Os resultados dessa análise são descritos em detalhe neste capítulo.

4.1 Análise e comparação de resultados

Para fazer a análise experimental dos programas implementados, foram utilizados dados gerados aleatoriamente para criar sequências artificiais e dados reais de sequências genéticas. O uso desses dois tipos diferentes de sequência permite uma análise do comportamento dos programas em diversas situações. Os testes experimentais foram realizados em um servidor Intel Xeon(R) E5-4650 2.7 GHZ, com 20 MB de memória *cache*, 95 GB de memória RAM e 8 núcleos de processamento, dedicado exclusivamente para os testes. O sistema operacional utilizado foi o Debian GNU/Linux 8 (jessie). Tomamos o cuidado de manter no máximo 7 processos ativos rodando ao mesmo tempo no servidor, onde um núcleo de processamento ficou sempre disponível para o sistema operacional.

O tempo de processamento apresentado nas tabelas a seguir, levam em consideração apenas a execução das abordagens 1 e 2 e não incluem a etapa de pré-processamento. O uso de memória é obtido através da função *getrusage*, que solicita ao sistema operacional o tamanho do espaço alocado para o programa que está em execução, ao final da execução do programa.

Vamos apresentar a seguir uma série de tabelas e gráficos gerados a partir dos testes realizados com dados artificiais e dados reais, juntamente a algumas observações, conjecturas e conclusões baseados nos dados obtidos.

4.2 Dados artificiais

Para a realização dos testes com dados artificiais, foi desenvolvido um script escrito na linguagem C++, para gerar as sequências de dados. Este programa utiliza, basicamente, a função *random* para gerar um número aleatório entre 1 e 4, cada qual representando uma letra do alfabeto $\Sigma = \{A, C, G, T\}$ (1 = A, 2 = C, 3 = G e 4 = T). Optamos por executar 3 séries de testes artificiais com dezenove sequências, cujos tamanhos variam de 1000 a 10000 caracteres, em intervalos de 1000

Tabela 4.1: *Sequências diferentes geradas aleatoriamente,* $k = 30$.

	KDP1		KDP2		KDP3		KDP4		KDP5		
Tamanho	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Ocr
1.000	0:00:01	3360	0:00:00	3360	0:00:39	3384	0:00:01	3708	0:00:01	5373	930
2.000	0:00:01	3360	0:00:01	3360	0:02:54	3397	0:00:04	3708	0:00:06	8449	1931
3.000	0:00:02	3360	0:00:03	3360	0:07:02	3612	0:00:06	3916	0:00:15	11723	2926
4.000	0:00:04	3711	0:00:05	3683	0:12:23	4026	0:00:13	4321	0:00:26	14997	3930
5.000	0:00:06	3916	0:00:08	4005	0:20:06	4504	0:00:23	4251	0:00:44	18040	4929
6.000	0:00:09	4196	0:00:12	4327	0:28:20	4872	0:00:33	4613	0:01:04	21235	5927
7.000	0:00:11	4549	0:00:17	4508	0:41:01	5276	0:00:46	4952	0:01:28	24248	6928
8.000	0:00:15	4820	0:00:21	4848	0:55:40	5632	0:01:01	5265	0:01:59	27263	7925
9.000	0:00:19	5167	0:00:27	5119	1:10:10	5970	0:01:17	5676	0:02:29	30324	8928
10.000	0:00:25	5385	0:00:33	5416	1:27:07	6384	0:01:34	6096	0:03:11	33433	9920
20.000	0:01:33	8169	0:02:14	8181	6:10:34	8250	0:06:21	9613	0:12:50	64389	19920
30.000	0:03:33	11036	0:05:02	11023	13:53:27	11740	0:14:23	13192	0:32:15	95240	29925
40.000	0:06:18	14037	0:08:55	14035	28:02:37	15300	0:25:26	17223	1:02:17	126044	39928
50.000	0:09:55	16856	0:14:04	16867	43:18:09	18834	0:40:01	20739	1:38:16	156707	49925
60.000	0:14:22	19685	0:20:03	19623	65:41:08	22430	1:00:04	37709	2:31:37	187785	59925
70.000	0:20:10	22781	0:27:31	22805	91:12:08	26104	1:24:06	28756	3:26:49	218845	69927
80.000	0:26:28	25592	0:35:56	25551	122:10:41	29602	1:50:38	32479	4:39:07	250233	79925
90.000	0:33:22	28620	0:45:42	28573	154:54:52	35264	2:36:12	36251	6:11:57	281035	89924
100.000	0:41:23	31389	0:58:07	31387	181:23:25	653005	2:56:52	49103	7:37:11	312087	99924

caracteres, e de 10000 a 100000 caracteres, em intervalos de 10000 caracteres. O valor de k foi fixado nos valores 30, 300 e 600 diferenças, para compreender o uso do sistema para todos os tamanhos de regiões específicas. A primeira série de testes foi realizada com sequências α e β diferentes, geradas aleatoriamente. A segunda série de testes foi realizada com sequências α e β iguais e a terceira série de testes foi realizada com sequências α e β completamente diferentes, em que a sequência α é constituída apenas pelo caractere 'A' e a sequência β apenas pelo caractere 'C'.

Cada série de testes foi executado três vezes e extraída a média do tempo de execução e da memória utilizada. Nas Tabelas 4.1, 4.2 e 4.3 são apresentados os resultados dos diferentes testes realizados com dados artificiais, para $k = 30$. Nas Tabelas 4.4, 4.5 e 4.6 apresentamos resultados dos diferentes testes realizados para $k = 300$ e nas Tabelas 4.7, 4.8 e 4.9 apresentamos resultados dos diferentes testes realizados para $k = 600$ com dados artificiais.

As tabelas são compostas por doze colunas, dispostas na seguinte ordem e com as seguintes informações:

1. Uma coluna denominada Tamanho com o tamanho da entrada do problema, ou seja, os tamanhos de α e β, para a execução do algoritmo; perceba que o tamanho de α e β são sempre iguais;

2. Cinco colunas denominadas Tempo com o tempo de execução no formato de hora: minuto: segundo, e cinco colunas denominadas Mem. com a memória física utilizada, em *bytes*; as colunas são agrupadas em cinco diferentes implementações;

3. Uma coluna denominada Ocr com a quantidade de ocorrências encontradas naquela execução.

Observe os tempos absurdamente altos obtidos pelo KDP3 e apresentados nas Tabelas 4.1, 4.2 e 4.3, referente aos primeiros testes executados. Acreditamos que essa ineficiência deve-se à compressão de dados aplicada pela estrutura da árvore de sufixo comprimida, que não permite o acesso direto ao dado, necessitando de vários passos de conversões para se obter o valor original necessário para computar a maior extensão comum. Como estamos interessados nos melhores algoritmos, e percebendo que o programa KDP3 não é competitivo na prática, o retiramos dos demais testes.

A Figura 4.1 apresenta o gráfico do tempo de execução dos programas KDP1, KDP2, KDP4 e KDP5, com dados gerados aleatoriamente para $k = 30$. As Figuras 4.2 e 4.3 apresentam os gráficos de tempo de execução dos testes para $k = 30$, referente a sequências iguais e sequências completamente diferentes, respectivamente.

Tabela 4.2: *Sequências iguais, $k = 30$.*

	KDP1		KDP2		KDP3		KDP4		KDP5		
Tamanho	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Ocr
1.000	0:00:00	3780	0:00:00	3780	0:00:00	3283	0:00:00	3780	0:00:00	6100	0
2.000	0:00:00	3780	0:00:00	3780	0:00:00	3312	0:00:00	3780	0:00:00	9724	0
3.000	0:00:00	3780	0:00:00	3780	0:00:00	3312	0:00:00	3780	0:00:00	13388	0
4.000	0:00:00	3780	0:00:00	3780	0:00:00	3312	0:00:00	3780	0:00:00	16928	0
5.000	0:00:00	3780	0:00:00	3780	0:00:00	3312	0:00:00	3780	0:00:00	20464	0
6.000	0:00:00	3780	0:00:00	3780	0:00:00	3312	0:00:00	3780	0:00:00	24152	0
7.000	0:00:00	3780	0:00:00	3780	0:00:00	3312	0:00:00	3780	0:00:00	27772	0
8.000	0:00:00	3780	0:00:00	3780	0:00:00	3328	0:00:00	3796	0:00:00	31480	0
9.000	0:00:00	3780	0:00:00	3780	0:00:00	3417	0:00:00	3984	0:00:00	35004	0
10.000	0:00:00	3780	0:00:00	3780	0:00:00	3451	0:00:00	4132	0:00:00	38644	0
20.000	0:00:00	3780	0:00:00	3780	0:00:00	4345	0:00:00	5860	0:00:00	74584	0
30.000	0:00:02	3780	0:00:00	3780	0:00:00	5201	0:00:00	7620	0:00:00	110588	0
40.000	0:00:03	3780	0:00:00	3780	0:00:00	6421	0:00:00	9712	0:00:00	147044	0
50.000	0:00:05	3780	0:00:00	3780	0:00:00	7224	0:00:00	11464	0:00:00	183036	0
60.000	0:00:08	3780	0:00:00	3780	0:00:00	8875	0:00:00	13316	0:00:00	219060	0
70.000	0:00:12	3780	0:00:00	3780	0:00:00	10055	0:00:00	15472	0:00:00	255072	0
80.000	0:00:16	3780	0:00:00	3780	0:00:00	11716	0:00:00	17368	0:00:00	291100	0
90.000	0:00:20	3780	0:00:00	3780	0:00:00	12451	0:00:00	19236	0:00:00	327024	0
100.000	0:00:26	3780	0:00:00	3780	0:00:00	14565	0:00:00	21008	0:00:00	362924	0

Tabela 4.3: *Sequências completamente diferentes, $k = 30$.*

	KDP1		KDP2		KDP3		KDP4		KDP5		
Tamanho	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Ocr
1.000	0:00:00	3423	0:00:00	3429	0:00:36	3422	0:00:01	3407	0:00:01	7049	971
2.000	0:00:01	3423	0:00:01	3429	0:02:28	3528	0:00:04	3407	0:00:07	11893	1971
3.000	0:00:01	3423	0:00:02	3429	0:06:30	3902	0:00:08	3613	0:00:16	16679	2971
4.000	0:00:02	3452	0:00:03	3451	0:11:53	4378	0:00:14	4075	0:00:28	21485	3971
5.000	0:00:03	3768	0:00:05	3708	0:19:07	4866	0:00:21	4505	0:00:46	26247	4971
6.000	0:00:04	4033	0:00:07	3939	0:25:19	4972	0:00:31	4923	0:01:08	30972	5971
7.000	0:00:05	3964	0:00:10	3977	0:35:19	5314	0:00:40	5101	0:01:30	35524	6971
8.000	0:00:06	4357	0:00:13	4299	0:47:23	5730	0:00:53	5551	0:01:57	40192	7971
9.000	0:00:08	4524	0:00:16	4565	1:04:45	6116	0:01:06	6057	0:02:27	45055	8971
10.000	0:00:10	4871	0:00:20	4819	1:21:31	6542	0:01:23	6461	0:03:05	49815	9971
20.000	0:00:39	7155	0:01:21	7129	5:09:29	10210	0:05:36	10406	0:12:36	97255	19971
30.000	0:01:25	9683	0:03:03	9725	12:06:00	13634	0:13:05	14524	0:26:58	144329	29971
40.000	0:02:34	12220	0:05:12	12219	21:51:02	17478	0:23:24	19020	0:50:55	191809	39971
50.000	0:03:58	14484	0:08:25	14503	36:44:58	21186	0:36:26	22960	1:19:14	238933	49971
60.000	0:05:45	17021	0:11:58	17000	50:33:47	24556	0:52:41	27203	1:58:46	286369	59971
70.000	0:07:48	19345	0:16:50	19331	76:38:02	27988	1:11:44	31724	4:13:38	333492	69971
80.000	0:09:55	21849	0:21:48	21876	101:20:12	32168	1:30:28	36025	6:01:04	380935	79971
90.000	0:13:21	24084	0:26:18	24183	129:52:38	37815	1:54:03	39981	7:55:18	426963	89971
100.000	0:16:32	26672	0:32:22	26660	152:40:39	42821	2:21:09	44349	10:03:36	461477	99971

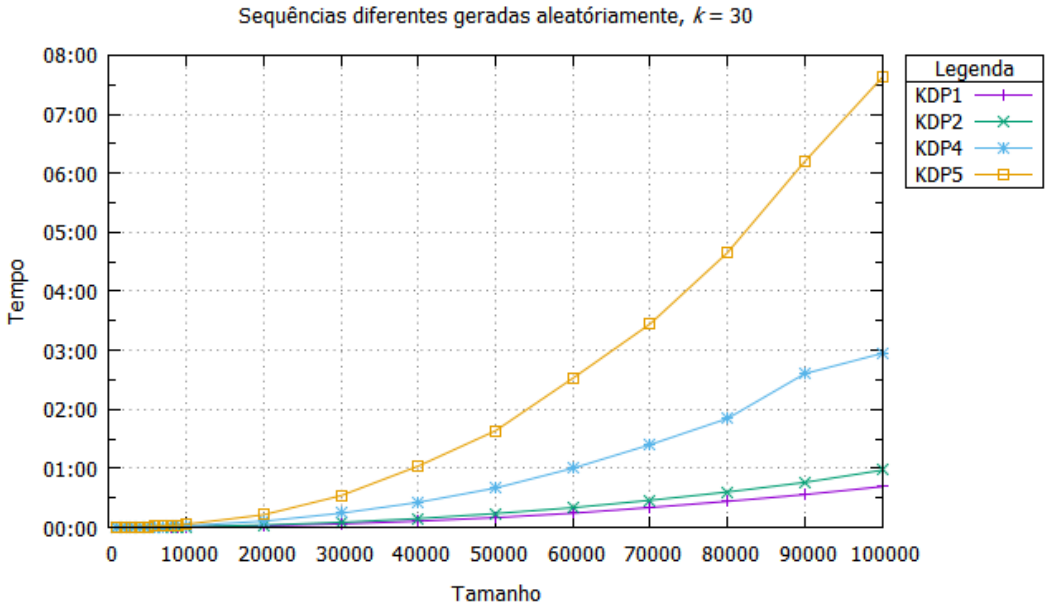


Figura 4.1: *Gráfico de tempo de execução das sequências diferentes, geradas aleatoriamente, $k = 30$.*

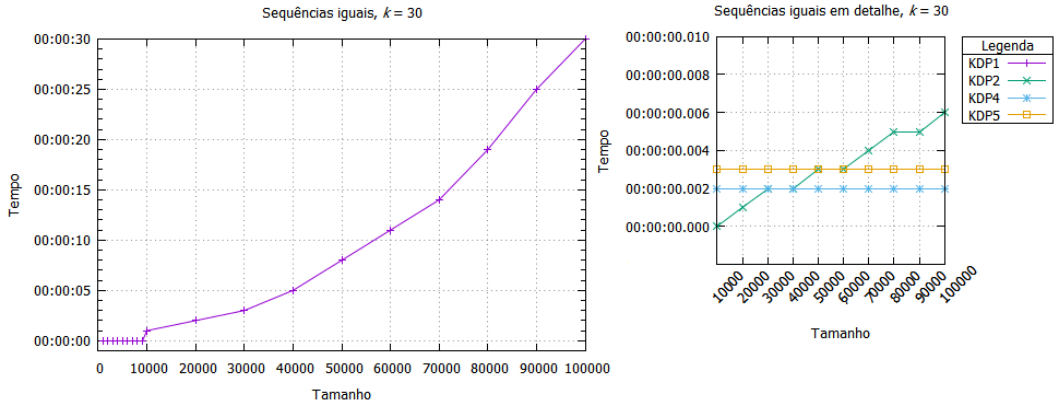


Figura 4.2: *Gráfico de tempo de execução de sequências iguais,* $k = 30$.

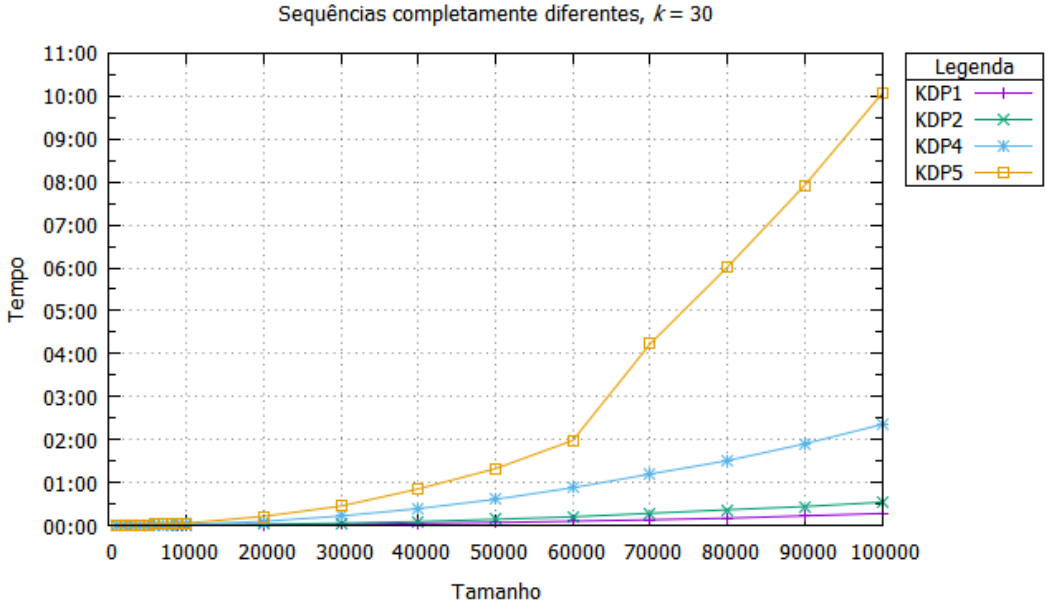


Figura 4.3: *Gráfico de tempo de execução das sequências completamente diferentes,* $k = 30$.

Tabela 4.4: *Sequências diferentes geradas aleatoriamente,* $k = 300$.

	KDP1		KDP2		KDP4		KDP5		
Tamanho	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Ocr
1.000	00:00:01	4892	00:00:01	4892	00:00:03	4892	00:00:05	6.000	358
2.000	00:00:06	5264	00:00:09	5288	00:00:22	5504	00:00:39	10800	1378
3.000	00:00:16	7352	00:00:22	7324	00:01:04	7752	00:01:53	15852	2371
4.000	00:00:28	9536	00:00:44	9524	00:02:02	10084	00:03:38	20756	3371
5.000	00:00:48	11376	00:01:09	11316	00:03:21	12188	00:06:26	25448	4367
6.000	00:01:04	13336	00:01:47	13376	00:05:00	14424	00:09:35	30356	5368
7.000	00:01:26	15508	00:02:21	15504	00:06:48	16660	00:12:57	35264	6374
8.000	00:01:50	17736	00:03:14	17688	00:09:04	18912	00:17:39	40084	7370
9.000	00:02:25	19772	00:03:58	19728	00:11:35	21288	00:22:20	45068	8362
10.000	00:02:53	21628	00:05:07	21700	00:14:27	23252	00:29:55	49600	9365
20.000	00:12:32	42600	00:20:11	42440	01:04:52	45764	01:59:00	98700	19366
30.000	00:27:47	63200	00:45:55	63252	02:24:43	68128	04:43:56	147532	29361
40.000	00:51:49	84140	01:27:51	84108	04:28:34	90912	08:36:51	196168	39367
50.000	01:21:06	104980	02:16:18	105064	06:55:52	113588	14:35:15	244824	49359
60.000	01:51:28	125840	03:03:23	125948	10:08:01	135852	21:31:33	293844	59359
70.000	02:32:21	146816	04:14:54	146860	14:20:40	159196	39:04:10	343028	69358
80.000	03:18:55	167660	05:55:46	167700	19:00:28	181912	57:41:59	392236	79351
90.000	04:25:17	188324	07:34:50	188260	24:04:10	204300	75:28:01	441116	89350
100.000	05:14:54	146860	09:21:50	209512	31:12:10	227132	89:38:52	490272	99362

Tabela 4.5: *Sequências iguais,* $k = 300$.

	KDP1		KDP2		KDP4		KDP5		
Tamanho	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Ocr
1.000	00:00:00	3748	00:00:00	3604	00:00:00	3748	00:00:00	6144	0
2.000	00:00:00	3748	00:00:00	3604	00:00:00	3748	00:00:00	9788	0
3.000	00:00:00	3748	00:00:00	3604	00:00:00	3748	00:00:00	13348	0
4.000	00:00:00	3748	00:00:00	3604	00:00:00	3748	00:00:00	16904	0
5.000	00:00:00	3748	00:00:00	3604	00:00:00	3748	00:00:00	20560	0
6.000	00:00:00	3748	00:00:00	3604	00:00:00	3748	00:00:00	24212	0
7.000	00:00:00	3748	00:00:00	3604	00:00:00	3748	00:00:00	27740	0
8.000	00:00:00	3748	00:00:00	3604	00:00:00	3756	00:00:00	31468	0
9.000	00:00:00	3748	00:00:00	3604	00:00:00	3996	00:00:00	34111	0
10.000	00:00:00	3748	00:00:00	3604	00:00:00	4104	00:00:00	37650	0
20.000	00:00:01	3748	00:00:00	3604	00:00:00	5860	00:00:00	65599	0
30.000	00:00:03	3748	00:00:00	3604	00:00:00	7500	00:00:00	101575	0
40.000	00:00:05	3748	00:00:00	3604	00:00:00	9780	00:00:00	137806	0
50.000	00:00:08	3748	00:00:00	3604	00:00:00	11560	00:00:00	173993	0
60.000	00:00:12	3748	00:00:00	3604	00:00:00	13228	00:00:00	210082	0
70.000	00:00:16	3748	00:00:00	3604	00:00:00	15576	00:00:00	246127	0
80.000	00:00:20	3748	00:00:00	3604	00:00:00	17468	00:00:00	279045	0
90.000	00:00:24	3748	00:00:00	3604	00:00:00	19252	00:00:00	327132	0
100.000	00:00:30	3748	00:00:00	3748	00:00:00	21020	00:00:00	362968	0

Tabela 4.6: *Sequências completamente diferentes,* $k = 300$.

	KDP1		KDP2		KDP4		KDP5		
Tamanho	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Ocr
1.000	00:00:00	3740	00:00:02	3740	00:00:06	3740	00:00:09	7664	701
2.000	00:00:03	4120	00:00:07	4236	00:00:28	4464	00:00:54	13216	1701
3.000	00:00:08	5304	00:00:16	5248	00:01:05	5752	00:02:22	18772	2701
4.000	00:00:13	6532	00:00:28	6628	00:02:00	7144	00:04:02	24656	3701
5.000	00:00:22	7632	00:00:45	7568	00:03:10	8376	00:06:40	30132	4701
6.000	00:00:30	8748	00:01:03	8712	00:04:38	9620	00:09:25	35756	5701
7.000	00:00:42	9788	00:01:27	9656	00:06:17	10828	00:13:29	41196	6701
8.000	00:00:54	10876	00:01:58	10716	00:08:19	11996	00:17:17	46828	7701
9.000	00:01:08	11868	00:02:29	11788	00:10:33	13244	00:22:31	52388	8701
10.000	00:01:29	12932	00:02:58	12932	00:13:07	14540	00:27:46	57888	9701
20.000	00:05:54	23444	00:12:12	23600	00:53:06	26784	01:48:13	113292	19701
30.000	00:13:03	34432	00:27:43	34476	01:59:23	39320	04:10:29	169184	29701
40.000	00:24:05	45060	00:48:47	44984	03:30:12	51764	07:43:01	224664	39701
50.000	00:36:38	55936	01:16:50	55932	05:29:46	64368	11:35:32	280308	49701
60.000	00:52:40	66512	01:52:38	66416	08:01:14	76720	16:24:11	335868	59701
70.000	01:11:09	77332	02:29:18	77340	10:53:40	89728	33:29:53	391548	69701
80.000	01:34:29	88048	03:15:42	88044	14:06:54	102088	52:58:42	447028	79701
90.000	01:57:47	98892	04:04:03	98848	17:51:57	114756	70:04:10	508410	89701
100.000	02:25:17	109492	05:21:34	109484	22:03:54	127076	81:52:00	562016	99701

Perceba que, quando as sequências são iguais (Tabelas 4.2, 4.5 e 4.8), não há ocorrências, e quando as sequências são completamente diferentes (Tabelas 4.3, 4.6 e 4.9), existe uma ocorrência de segmento para cada $i - k + 1$ posição de α. Esses testes simulam as situações de melhor caso e de pior caso, respectivamente.

Com base nos testes realizados com sequências diferentes geradas aleatoriamente e sequências completamente diferentes, percebemos que o programa KDP1, que implementa o Algoritmo 6, é ligeiramente mais rápido que o programa KDP2, que implementa o Algoritmo 7, o que condiz com a diferença de complexidade teórica desses algoritmos. Mais detalhadamente, ele é cerca de 30% mais rápido, na média, quando $k = 30$, cerca de 61% mais rápido, na média, quando $k = 300$ e cerca de 63% mais rápido, na média, quando $k = 600$. Ambos são muito superiores aos algoritmos KDP4 e KDP5 em cerca de mais de 280% na média. A implementação baseada em árvore de sufixo mostrou-se a pior de todas ao lidar com sequências diferentes e sequências completamente diferentes. Nesse caso, nada justifica seu uso pois, mesmo tendo uma computação constante da maior extensão comum entre duas sequências, o tempo dessa computação é alto na prática além de um alto consumo de memória para manter a estrutura de dados. O vetor de sufixo com a computação do RMQ mostrou-se mais competitivo na prática do que a árvore de sufixo, usando menos espaço e podendo ser construído de forma mais rápida. O tempo de computação da maior extensão comum utilizando a árvore de sufixo ou o vetor de sufixo é constante o que fica claro olhando-se os valores nas Tabelas 4.2, 4.5 e 4.8. Perceba nelas o tempo constante dos programas e o crescente uso de memória no KDP4 e KDP5, que não ocorre no KDP1 e KDP2.

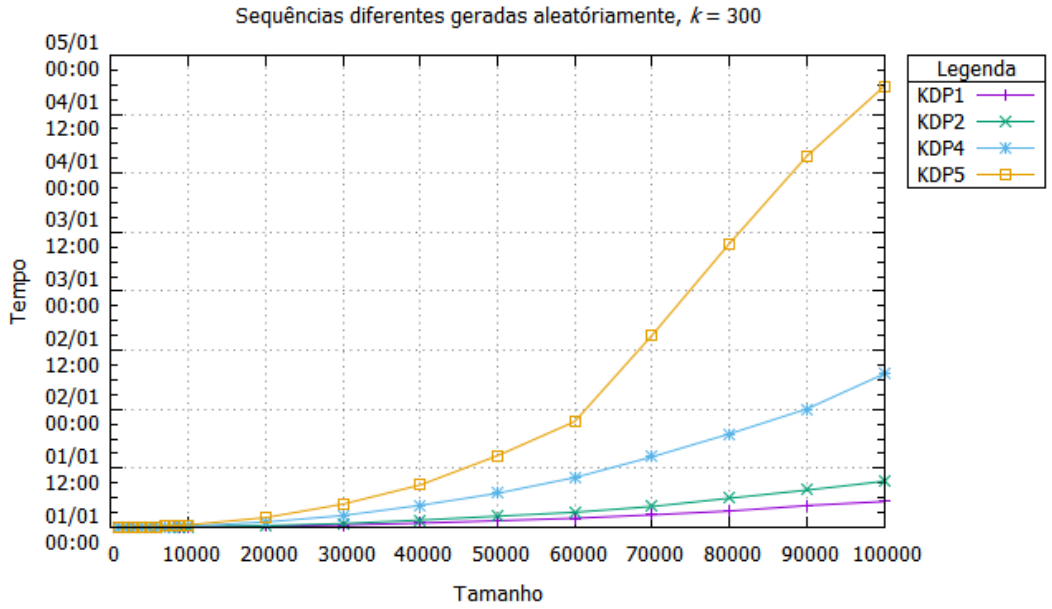


Figura 4.4: *Gráfico de tempo de execução das sequências diferentes, geradas aleatoriamente,* $k = 300$.

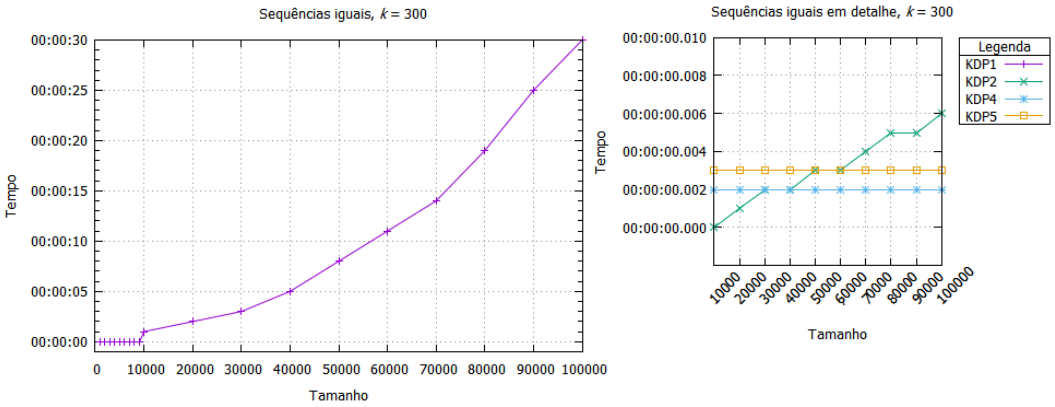


Figura 4.5: *Gráfico de tempo de execução de sequências iguais,* $k = 300$.

Tabela 4.7: *Sequências diferentes geradas aleatoriamente,* $k = 600$.

	KDP1		KDP2		KDP4		KDP5		
Tamanho	**Tempo**	**Mem.**	**Tempo**	**Mem.**	**Tempo**	**Mem.**	**Tempo**	**Mem.**	**Ocr**
1.000	00:00:00	3789	00:00:01	3789	00:00:00	3680	00:00:00	5356	0
2.000	00:00:06	5501	00:00:09	5516	00:00:23	5836	00:00:41	11192	769
3.000	00:00:19	9200	00:00:32	9179	00:01:23	9644	00:02:44	17740	1755
4.000	00:00:40	13153	00:01:07	13112	00:03:02	13804	00:05:55	24448	2754
5.000	00:01:07	16883	00:01:54	16884	00:05:12	17732	00:11:00	30932	3759
6.000	00:01:43	20843	00:02:49	20839	00:07:57	21812	00:16:52	37752	4754
7.000	00:02:28	24800	00:04:00	24784	00:11:27	25884	00:23:00	44532	5763
8.000	00:03:17	28521	00:05:21	28544	00:15:50	29688	00:30:37	50984	6755
9.000	00:04:13	32375	00:06:57	32431	00:20:06	33924	00:41:35	57712	7755
10.000	00:05:14	36451	00:08:40	36485	00:25:47	37816	00:52:04	64380	8747
20.000	00:22:48	75055	00:39:29	74980	01:50:36	78408	03:52:52	131176	18754
30.000	00:52:47	113888	01:29:12	113784	04:23:44	118852	10:20:51	198176	28751
40.000	01:36:10	152800	02:41:57	152792	08:27:37	159476	19:22:09	264760	38756
50.000	02:34:42	191888	04:23:38	191924	12:49:02	200076	33:06:40	331708	48753
60.000	03:42:05	230700	06:08:21	230664	18:43:52	240980	40:49:50	398620	58741
70.000	04:53:36	269564	08:31:51	269608	27:03:03	281972	63:00:26	465844	68733
80.000	06:30:45	308632	11:10:45	308720	38:07:15	322888	96:01:35	533100	78733
90.000	08:13:26	347528	13:58:39	347560	46:23:41	363384	142:37:25	600044	88732
100.000	10:01:21	386396	17:23:37	386420	59:30:52	404108	191:14:11	667124	98722

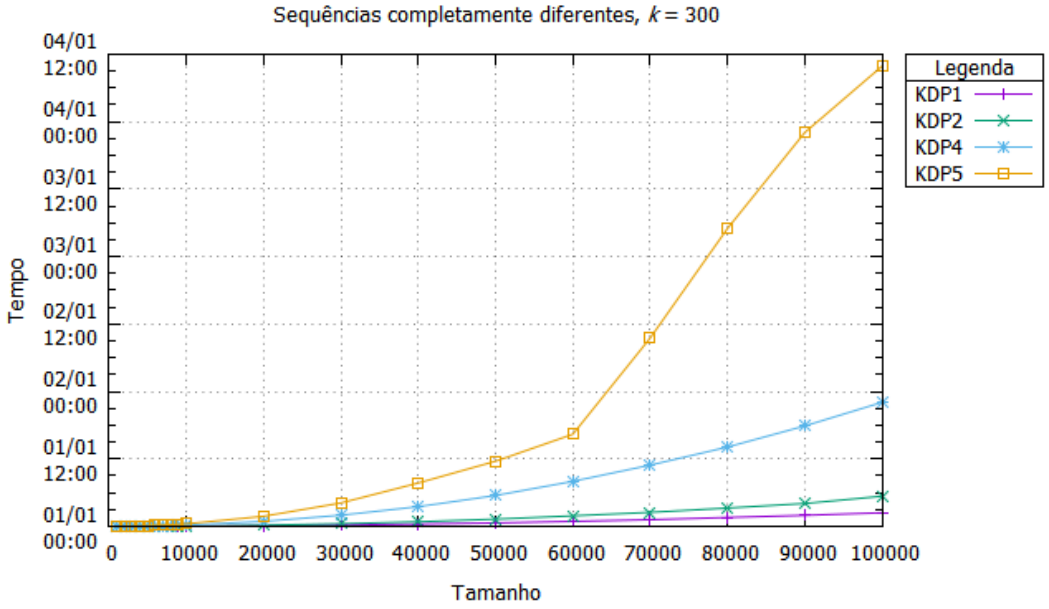


Figura 4.6: *Gráfico de tempo de execução das sequências completamente diferentes, $k = 300$.*

Tabela 4.8: *Sequências iguais, $k = 600$.*

	KDP1		KDP2		KDP4		KDP5		
Tamanho	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Ocr
1.000	00:00:00	3880	00:00:00	3880	00:00:00	3488	00:00:00	6164	0
2.000	00:00:00	3880	00:00:00	3880	00:00:00	3493	00:00:00	9760	0
3.000	00:00:00	3880	00:00:00	3880	00:00:00	3493	00:00:00	13391	0
4.000	00:00:00	3880	00:00:00	3880	00:00:00	3493	00:00:00	16924	0
5.000	00:00:00	3880	00:00:00	3880	00:00:00	3493	00:00:00	20519	0
6.000	00:00:00	3880	00:00:00	3880	00:00:00	3520	00:00:00	24213	0
7.000	00:00:00	3880	00:00:00	3880	00:00:00	3649	00:00:00	27817	0
8.000	00:00:00	3880	00:00:00	3880	00:00:00	3839	00:00:00	31408	0
9.000	00:00:00	3880	00:00:00	3880	00:00:00	4000	00:00:00	34110	0
10.000	00:00:00	3880	00:00:00	3880	00:00:00	4227	00:00:00	37705	0
20.000	00:00:02	3880	00:00:00	3880	00:00:00	5909	00:00:00	65599	0
30.000	00:00:03	3880	00:00:00	3880	00:00:00	7652	00:00:00	101614	0
40.000	00:00:05	3880	00:00:00	3880	00:00:00	9675	00:00:00	137857	0
50.000	00:00:07	3880	00:00:00	3880	00:00:00	11435	00:00:00	173963	0
60.000	00:00:10	3880	00:00:00	3880	00:00:00	13156	00:00:00	210059	0
70.000	00:00:14	3880	00:00:00	3880	00:00:00	15717	00:00:00	246285	0
80.000	00:00:18	3880	00:00:00	3880	00:00:00	17531	00:00:00	279211	0
90.000	00:00:27	3880	00:00:00	3884	00:00:00	19383	00:00:00	327300	0
100.000	00:00:30	3880	00:00:00	3888	00:00:00	21293	00:00:00	363184	0

Tabela 4.9: *Sequências completamente diferentes, $k = 600$.*

	KDP1		KDP2		KDP4		KDP5		
Tamanho	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Tempo	Mem.	Ocr
1.000	00:00:01	3800	00:00:02	3808	00:00:03	3880	00:00:05	7940	401
2.000	00:00:05	5300	00:00:10	5176	00:00:33	5564	00:01:02	14284	1401
3.000	00:00:11	7152	00:00:23	7160	00:01:33	7484	00:03:24	20684	2401
4.000	00:00:22	9196	00:00:44	9160	00:03:05	9736	00:06:27	27252	3401
5.000	00:00:35	11076	00:01:05	11124	00:05:02	11768	00:12:15	33652	4401
6.000	00:00:53	13212	00:01:33	13092	00:07:56	14140	00:17:16	40192	5401
7.000	00:01:12	15088	00:02:09	15048	00:11:06	16060	00:26:20	46512	6401
8.000	00:01:35	17092	00:02:50	17112	00:14:23	18364	00:31:17	53028	7401
9.000	00:02:03	18988	00:03:54	19068	00:18:03	20480	00:44:56	59420	8401
10.000	00:02:33	20760	00:09:07	20772	00:23:09	22552	00:51:55	65884	9401
20.000	00:10:18	40648	00:18:00	40640	01:35:58	43956	03:55:56	130456	19401
30.000	00:24:05	60252	00:40:32	60252	03:40:35	65216	08:40:52	194932	29401
40.000	00:42:36	79784	01:12:25	79888	06:37:42	36704	16:05:17	259540	39401
50.000	01:08:01	99368	01:52:45	99760	10:22:49	107900	35:52:18	323976	49401
60.000	01:36:02	119340	02:51:34	119216	16:04:51	129604	46:36:07	388632	59401
70.000	02:08:50	138944	03:42:32	138900	22:34:53	151268	60:36:51	453140	69401
80.000	02:51:44	158424	04:49:32	138900	29:25:31	172528	93:11:31	517564	79401
90.000	03:36:36	178332	06:05:43	178320	37:01:03	194192	135:08:27	586420	89401
100.000	04:25:46	197896	07:30:54	197968	44:51:33	363384	180:08:44	650360	99401

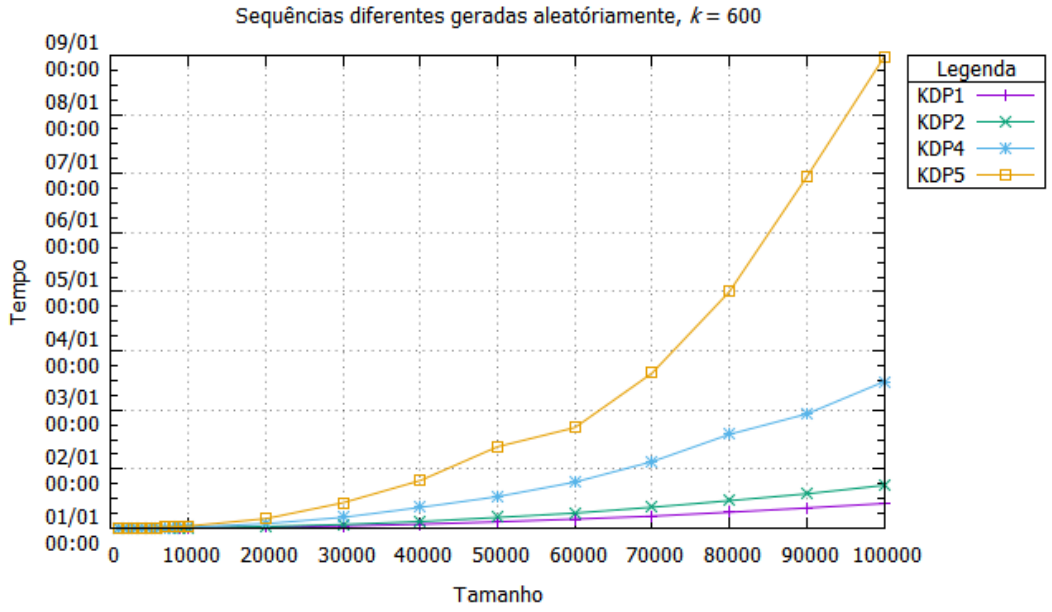


Figura 4.7: *Gráfico de tempo de execução das sequências diferentes, geradas aleatoriamente,* $k = 600$.

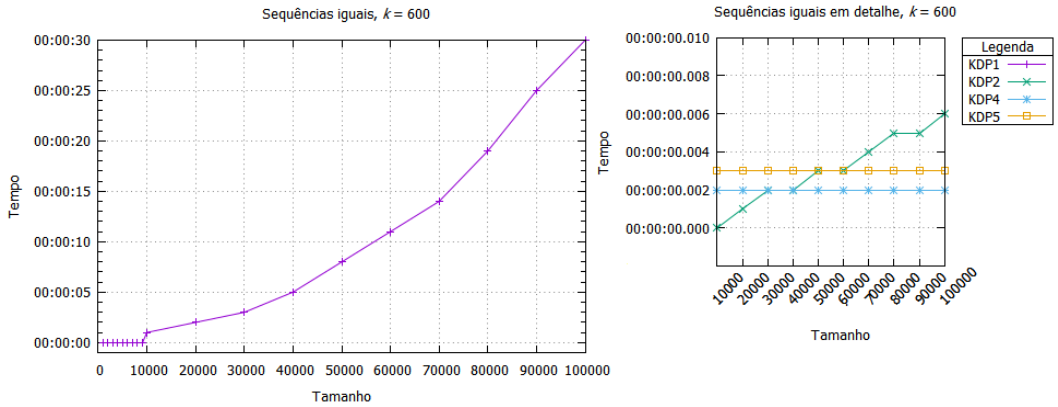


Figura 4.8: *Gráfico de tempo de execução de sequências iguais,* $k = 600$.

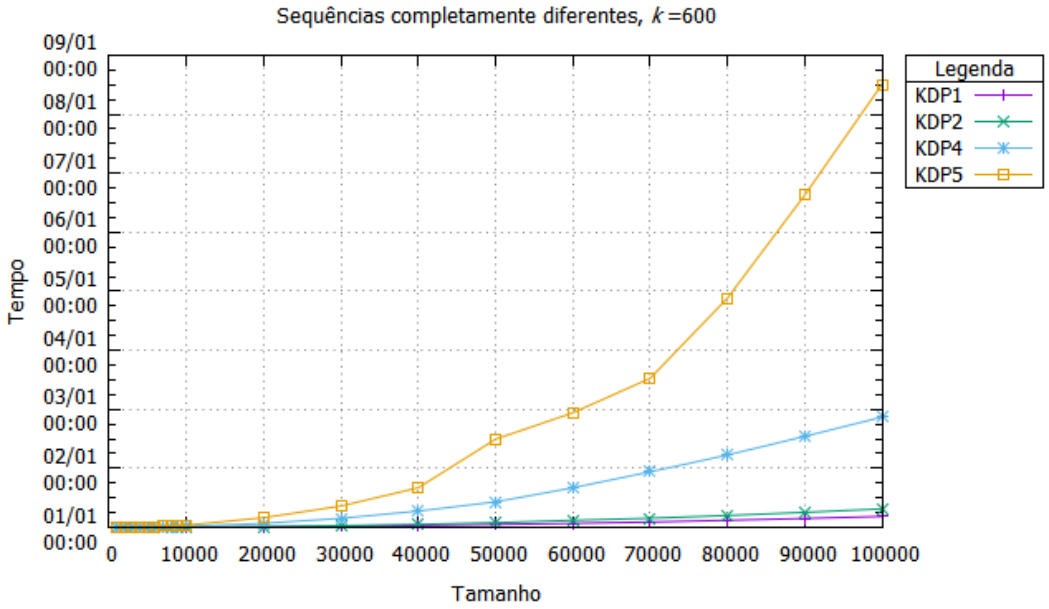


Figura 4.9: *Gráfico de tempo de execução das sequências completamente diferentes,* $k = 600$.

Já para os casos de testes com sequências iguais, percebemos que o programa KDP1 é o mais lento de todos. Neste caso, como as sequências são iguais, a abordagem 2 possui uma larga vantagem sobre a abordagem 1: ao computar as primeiras células da matriz L, o fim de α já é encontrado, ao passo que, na abordagem 1 é necessário computar toda a matriz D para encontrar o fim de α. Perceba que, nesses casos de testes, o programa KDP2, mesmo tendo a implementação ingênua da maior extensão comum, consegue competir com os programas KDP4 e KDP5.

Acreditamos que a vantagem em tempo obtida pelo programa KDP2 em relação aos programas KDP4 e KDP5 nos testes com sequências completamente diferentes e sequências aleatórias, e igualado nos testes com sequências iguais, pode ser explicada pela organização de memória dos processadores atuais, que devido ao avanço tecnológico obtido com a diminuição dos transistores, possibilitou a adição de memórias de acesso rápido junto à CPU, chamadas de memórias *cache*. Dessa forma, as memórias passaram a ser organizadas por hierarquia de proximidade física ao processador, onde quanto mais próxima estiver, mais rápido é o acesso a ela. O tamanho dessa memória porém é limitado e por conta disso existe um controlador que gerencia os dados nelas armazenadas. Esse controlador sincroniza os dados das memórias *cache* com a memória principal. Caso o dado buscado não seja encontrado na memória *cache*, o controlador busca na memória principal. Como o tempo de sincronização das memórias leva vários ciclos de *clock* do processador, o controlador faz uma previsão de uso dos dados e sempre traz para a memória *cache* registros de memória próximos. Essa organização favorece os programas KDP1 e KDP2, que fazem uso apenas das matrizes de programação dinâmica, onde a próxima célula da matriz quase sempre está na memória *cache* [53].

No caso do KDP4 e do KDP5, suas estruturas de dados possuem uma organização não sequencial na memória do computador e o acesso aos dados é direto para cálculo do RMQ no vetor de sufixo ou cálculo do LCA na árvore de sufixo, podendo ocasionar vários *cache miss*, onde o dado não está na memória *cache* e o controlador precisa buscar na memória principal, o que prejudica muito o desempenho [54, 55]. Somando-se a isso a grande quantidade de computação do LCE envolvida no processamento das sequências, temos a diferença de tempo considerável entre esses programas e o

KDP2.

A quantidade de memória utilizada pelos programas KDP1, em todos os casos de teste quando $k = 30$, é praticamente idêntico ao que utiliza o programa KDP2. Nesses casos, o programa KDP4 utiliza, na média, cerca de 50% mais memória que os programas KDP2 e KDP1 e o programa KDP5 utiliza, na média cerca de 870% mais memória que o programa KDP4. Quando $k = 300$, a quantidade de memória utilizada pelos programas KDP1 e KDP2, na média, é idêntica, enquanto que o programa KDP4 utiliza, na média, cerca de 14% mais memória que os programas KDP2 e KDP1 e o programa KDP5 utiliza, na média, cerca de 329% mais memória que o programa KDP4. Quando $k = 600$, a quantidade de memória utilizada pelos programas KDP1 e KDP2, na média, é idêntica, enquanto que, o programa KDP4 utiliza, na média, cerca de 18% mais memória que os programas KDP2 e KDP1 e o programa KDP5 utiliza, na média, cerca de 177% mais memória que o programa KDP4. Nos casos onde as sequências são iguais, para todos os valores de k, observamos que o consumo de memória dos programas é mínimo, por conta de não haver ocorrências, e os programas KDP4 e KDP5, consomem memória proporcional ao tamanho da entrada.

O tamanho médio da maior extensão comum entre duas sequências também deve ser levado em conta na comparação da eficiência dos programas. Como já observado por Ilie em [25], o tamanho médio da maior extensão comum entre duas sequências é, em geral, pequeno. Com isso, o algoritmo ingênuo para o cálculo do LCE, cuja complexidade teórica é $O(n)$, passa a ser muito mais rápido na prática. Isso pôde ser visto nos nossos testes com sequências diferentes geradas aleatoriamente, onde o tamanho máximo do LCE dessas sequências, na média, é de 13 caracteres e a média do tamanho da maior extensão comum, levando em consideração os valores iguais a zero, é de 0.33 e de 1.33, levando em consideração apenas valores maiores que zero, o que faz com que o programa KDP2, que calcula de forma direta o LCE, leve grande vantagem sobre os programas KDP4 e KDP5.

Devemos considerar também em nossa análise a similaridade entre as sequências. Obtivemos a porcentagem de similaridade entre as sequências α e β usadas nos testes através do programa EMBOSS Needle para sequências que, juntas, somam tamanho menor ou igual a 100000 caracteres e do programa EMBOSS Stretcher para sequências que, juntas, somam tamanho maior que 100000 caracteres, ambos adquirido no endereço ftp://emboss.open-bio.org/pub/EMBOSS/. Tais programas usam o algoritmo de Needleman-Wunsch [56] para determinar o alinhamento global ótimo de duas sequências e, com base nele, calcular um valor, em porcentagem, representando a similaridade entre elas. Para as sequências geradas aleatoriamente obtivemos as seguinte similaridades entre α e β (de mesmo tamanho): tamanho 1000 com similaridade 47%, tamanho 2000 com similaridade 46.3%, tamanho 3000 com similaridade 45.9%, tamanho 4000 com similaridade 46.5%, tamanho 5000 com similaridade 47.3%, tamanho 6000 com similaridade 46.8%, tamanho 7000 com similaridade 47%, tamanho 8000 com similaridade 46.2%, tamanho 9000 com similaridade 46.6%, tamanho 10000 com similaridade 46.6%, tamanho 20000 com similaridade 46.6%, tamanho 30000 com similaridade 46.7%, tamanho 40000 com similaridade 47.5%, tamanho 50000 com similaridade 47.5%, tamanho 60000 com similaridade 47.6%, tamanho 70000 com similaridade 47.5%, tamanho 80000 com similaridade 47.4%, tamanho 90000 com similaridade 47.4% e tamanho 100000 com similaridade 47.4%. Percebemos com isso que, em casos onde a similaridade entre as sequências é de 100%, o KDP2 é o programa mais rápido. Já em casos onde a similaridade entre as sequências varia de 0% a 47.6%, ou seja, menor que 50%, o KDP1 é o programa mais rápido.

Outra análise que devemos fazer é sobre a real complexidade do programa KDP1 que imple-

menta a abordagem 1, pois a principio identificamos que o Algoritmo 6 possui complexidade de tempo de pior caso $O(m^2n)$, o que, na prática, não foi alcançada. Nos testes avaliamos três casos: sequências diferentes, sequências iguais e sequências completamente diferentes. Quando as sequências são iguais, apenas a primeira posição da sequência α é avaliada, ou seja, a matriz D é percorrida por completo, uma vez, o que consome tempo $O(mn)$. Isso pode ser visto nas Tabelas 4.2, 4.5 e 4.8, que apresentam a execução dos testes com sequências iguais para $k = 30$, $k = 300$ e $k = 600$, respectivamente, onde o tempo de execução do programa KDP1 não se altera, independentemente do valor de k. Quando as sequências são completamente diferentes, todas as ocorrências tem tamanho k e, claramente a complexidade de tempo nesse caso, é $O(mkn)$, onde a matriz D é preenchida m vezes, e sempre até a linha k de D. Quando as sequências são diferentes, a matriz D é preenchida m vezes, até uma certa linha e raramente atinge a última linha da matriz. Mais detalhadamente, quando $k = 30$, esse preenchimento termina entre as linhas 65 e 91, observando que o tamanho médio das ocorrências nesse caso é de 74 caracteres. Quando $k = 300$, esse preenchimento termina entre as linhas 611 e 663, com o tamanho médio das ocorrências de 635 caracteres. Quando $k = 600$, esse preenchimento termina entre as linhas 1228 e 1291, com o tamanho médio das ocorrências de 1251 caracteres. Essas considerações nos permitem afirmar que, para esses casos de testes, a complexidade de tempo do KDP1, na prática é $O(mkn)$.

4.3 Dados reais

Os testes realizados com sequências artificiais geradas de forma aleatória apresentaram resultados importantes mas que não abrangem todos os casos possíveis. Isso pois, as sequências de DNA de seres vivos não se enquadram em nenhum dos três testes realizados com dados aleatórios: elas não são exatamente iguais, não são completamente diferentes e não são sequências aleatórias. Por conta disso, realizamos uma nova bateria de testes com dados reais, que é importante para avaliar o comportamento dos programas com sequências reais e que nos dará um resultado mais próximo do uso prático dos programas.

Para executar os testes com dados reais, utilizamos dez sequências do *Homo sapiens*, cada uma delas contendo um gene específico, de diversos tamanhos, a saber, *hsarhgdig* com 4398 caracteres, *hsascl2* com 4455 caracteres, *hsankrd43* com 5457 caracteres, *hsa1bg* com 8694 caracteres, *hsalg10b* com 14972 caracteres, *hsbad* com 16877 caracteres, *hsankr10* com 38530 caracteres, *hsascc2* com 51655 caracteres, *hsasz1* com 66302 caracteres e *hsaff4* com 90284 caracteres. Foram executadas três séries de testes, variando o valor de k e comparando cada uma das sequências com todas as outras. Cada série de testes foi executado três vezes e extraída a média do tempo de execução e da quantidade de memória utilizada. Os resultados parciais, contendo apenas as sequências *hsarhgdig*, *hsankr10* e *hsaff4* que representam casos pequeno, médio e grande, respectivamente, são apresentados nas Tabelas 4.10 para $k = 30$, Tabela 4.11 para $k = 300$ e Tabela 4.12 para $k = 600$. Os demais testes são apresentados no Apêndice B.

As tabelas são compostas por treze colunas, na seguinte ordem e com as seguintes informações:

1. Duas colunas, denominadas Nome e Tam., com a identificação e o tamanho da sequência α para a execução do algoritmo;

2. Duas colunas, denominadas Nome e Tam., com a identificação e o tamanho da sequência β para a execução do algoritmo;

3. Oito colunas, denominadas Tem. e Mem., apresentando, respectivamente, o tempo de execução, no formato hora:minuto:segundo, e a quantidade de memória física utilizada, em *bytes*, agrupadas pelas quatro diferentes implementações;

4. Uma coluna, denominada Ocr., com a quantidade de ocorrências encontradas naquela execução.

Tabela 4.10: *Sequências reais de genes do* Homo sapiens*, tamanho pequeno, médio e grande,* $k = 30$.

α		β		KDP1		KDP2		KDP4		KDP5		
Nome	**Tam.**	**Nome**	**Tam.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Ocr**
hsarhgdig	4398	hsascl2	4455	00:00:05	3740	00:00:07	3716	00:00:18	4436	00:00:34	16699	4326
hsarhgdig	4398	hsankrd43	5457	00:00:06	4228	00:00:07	4276	00:00:21	4992	00:00:41	18576	4324
hsarhgdig	4398	hsa1bg	8694	00:00:10	4276	00:00:12	4256	00:00:34	5264	00:01:04	23480	4245
hsarhgdig	4398	hsalg10b	14972	00:00:13	4128	00:00:18	4240	00:01:01	5732	00:02:01	32548	4250
hsarhgdig	4398	hsbad	16877	00:00:19	3729	00:00:24	3744	00:01:09	5428	00:02:09	36243	4237
hsarhgdig	4398	hsankr10	38530	00:00:37	4604	00:00:48	4540	00:02:37	8064	00:05:44	66748	4245
hsarhgdig	4398	hsascc2	51655	00:00:55	3835	00:01:11	3811	00:03:26	8363	00:07:31	87596	4240
hsarhgdig	4398	hsasz1	66302	00:01:09	4119	00:01:26	4103	00:04:26	10124	00:10:01	107500	4237
hsarhgdig	4398	hsaff4	90284	00:01:23	4716	00:01:49	4716	00:06:32	12736	00:18:49	148080	4231
hsankr10	38530	hsarhgdig	4398	00:00:38	13440	00:00:53	13492	00:02:47	16764	00:05:50	75587	38462
hsankr10	38530	hsascl2	4455	00:00:36	14885	00:00:52	14981	00:02:45	16736	00:05:36	75481	38462
hsankr10	38530	hsankrd43	5457	00:00:50	13496	00:01:09	13453	00:03:31	17067	00:07:07	76983	38459
hsankr10	38530	hsa1bg	8694	00:01:15	18720	00:01:44	18640	00:05:34	22580	00:11:37	87516	38461
hsankr10	38530	hsalg10b	14972	00:02:22	19020	00:03:14	18948	00:09:56	23304	00:20:32	96304	38459
hsankr10	38530	hsbad	16877	00:02:44	13724	00:03:20	13737	00:10:52	18267	00:23:16	95708	38459
hsankr10	38530	hsascc2	51655	00:08:54	13756	00:11:05	13833	00:33:31	21495	01:19:11	147093	38456
hsankr10	38530	hsasz1	66302	00:10:55	14109	00:14:45	14113	00:43:41	22951	01:46:28	167469	38453
hsankr10	38530	hsaff4	90284	00:15:23	20136	00:19:17	20212	01:02:42	31072	03:09:26	213624	38454
hsaff4	90284	hsarhgdig	4398	00:01:36	28232	00:02:08	28165	00:06:26	36279	00:14:16	171645	90212
hsaff4	90284	hsascl2	4455	00:01:33	27987	00:02:23	27933	00:06:40	36005	00:14:40	171381	90217
hsaff4	90284	hsankrd43	5457	00:02:04	27945	00:02:45	27953	00:08:09	36299	00:12:45	124749	90217
hsaff4	90284	hsa1bg	8694	00:03:34	44280	00:04:04	44356	00:13:34	52692	00:29:06	194188	90214
hsaff4	90284	hsalg10b	14972	00:06:39	29284	00:07:52	29299	00:23:44	38213	00:51:13	188425	90214
hsaff4	90284	hsbad	16877	00:08:16	30069	00:08:28	30055	00:26:36	39139	00:58:51	193788	90216
hsaff4	90284	hsankr10	38530	00:17:34	46015	00:19:52	45983	01:01:54	41244	02:19:13	223804	90213
hsaff4	90284	hsascc2	51655	00:25:22	46783	00:25:43	41377	01:23:26	43165	03:41:13	245777	90213
hsaff4	90284	hsasz1	66302	00:31:38	41593	00:35:09	30671	01:54:35	44559	04:15:46	265832	90212

Os gráficos 4.10, 4.11 e 4.12 representam os dados da Tabela 4.10 para os testes com as sequências hsarhgdig, hsankr10 e hsaff4, respectivamente, quando $k = 30$. Os gráficos 4.13, 4.14 e 4.15 representam os dados da Tabela 4.11 para os testes com as sequências hsarhgdig, hsankr10 e hsaff4, respectivamente, quando $k = 300$ e os gráficos 4.16, 4.17 e 4.18 representam os dados da Tabela 4.12 para os testes com as sequências hsarhgdig, hsankr10 e hsaff4, respectivamente, quando $k = 600$.

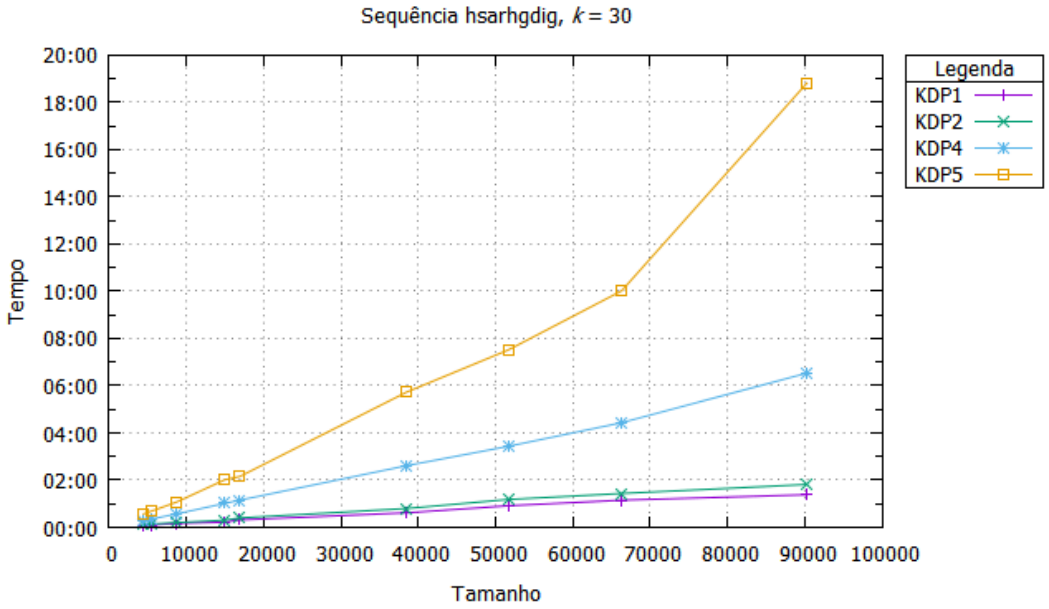


Figura 4.10: *Gráfico de tempo de execução da sequência hsarhgdig,* $k = 30$.

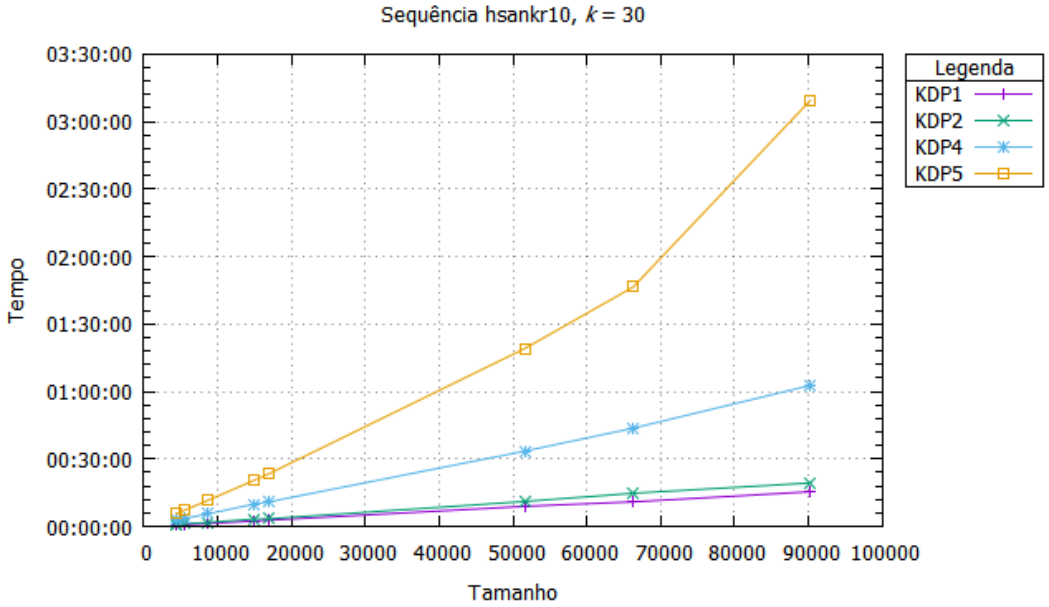


Figura 4.11: *Gráfico de tempo de execução da sequência hsankr10,* $k = 30$.

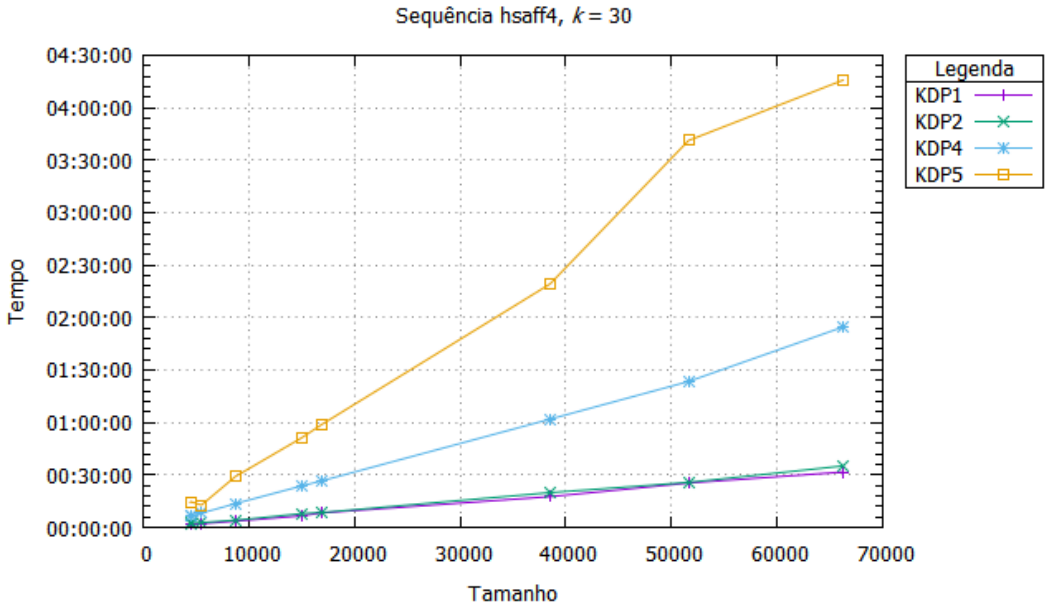


Figura 4.12: *Gráfico de tempo de execução da sequência hsaff4,* $k = 30$.

Tabela 4.11: *Sequências reais de genes do* Homo sapiens*, tamanho pequeno, médio e grande,* $k = 300$.

α		β		KDP1		KDP2		KDP4		KDP5		
Nome	**Tam.**	**Nome**	**Tam.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Ocr**
hsarhgdig	4398	hsascl2	4455	00:00:33	10524	00:00:54	10528	00:02:24	11312	00:04:50	23476	3750
hsarhgdig	4398	hsankrd43	5457	00:00:39	10508	00:01:02	10448	00:03:08	11336	00:05:42	24960	3747
hsarhgdig	4398	hsa1bg	8694	00:01:07	10352	00:01:38	10192	00:05:11	11312	00:09:24	29836	3704
hsarhgdig	4398	hsalg10b	14972	00:01:40	13128	00:02:43	10364	00:08:57	11844	00:18:28	38572	3768
hsarhgdig	4398	hsbad	16877	00:02:01	10380	00:03:12	10332	00:09:21	12028	00:18:51	42736	3702
hsarhgdig	4398	hsankr10	38530	00:04:32	10688	00:07:09	10632	00:19:14	18720	00:42:09	154952	3722
hsarhgdig	4398	hsascc2	51655	00:05:58	10428	00:09:47	10388	00:29:59	14928	01:12:09	94044	3708
hsarhgdig	4398	hsasz1	66302	00:07:37	10428	00:11:31	10500	00:39:07	16512	01:34:11	113832	3720
hsarhgdig	4398	hsaff4	90284	00:11:13	10748	00:16:47	10752	00:52:59	29026	02:32:37	337482	3717
hsankr10	38530	hsarhgdig	4398	00:04:51	75400	00:08:05	75288	00:23:58	78664	00:50:35	137448	37945
hsankr10	38530	hsascl2	4455	00:05:07	75860	00:08:27	75880	00:24:01	79484	00:54:14	137832	37942
hsankr10	38530	hsankrd43	5457	00:06:37	80252	00:10:18	80376	00:29:45	83956	01:07:46	143868	37894
hsankr10	38530	hsa1bg	8694	00:10:32	82060	00:16:59	82040	00:46:54	98982	01:37:53	383632	37906
hsankr10	38530	hsalg10b	14972	00:18:45	82896	00:29:40	82800	01:18:42	101564	03:23:26	160204	37884
hsankr10	38530	hsbad	16877	00:20:00	81580	00:33:14	81688	01:34:28	86192	03:42:56	163640	37911
hsankr10	38530	hsascc2	51655	01:03:40	83324	01:42:25	83276	05:08:41	90920	11:14:42	216336	37892
hsankr10	38530	hsasz1	66302	01:24:22	84868	02:11:05	84944	06:45:22	93752	16:04:18	238656	37877
hsankr10	38530	hsaff4	90284	01:55:53	85568	03:02:18	85549	07:52:19	132040	18:19:55	907798	37882
hsaff4	90284	hsarhgdig	4398	00:11:21	174476	00:19:46	174432	00:59:51	182456	02:09:58	317936	89698
hsaff4	90284	hsascl2	4455	00:12:04	175304	00:19:23	175296	00:59:00	183348	02:18:16	318596	86698
hsaff4	90284	hsankrd43	5457	00:15:02	185088	00:24:44	184992	01:14:14	193140	02:49:42	329976	89648
hsaff4	90284	hsa1bg	8694	00:27:11	204288	00:40:00	204512	01:43:34	243104	03:42:05	895806	89646
hsaff4	90284	hsalg10b	14972	00:46:13	199828	01:09:59	199856	03:32:29	208732	08:10:46	358864	89602
hsaff4	90284	hsbad	16877	00:54:24	207488	01:16:55	207444	03:55:18	216500	09:32:32	370928	89626
hsaff4	90284	hsankr10	38530	02:01:28	209412	02:56:13	209328	09:38:29	220272	21:30:37	402892	89613
hsaff4	90284	hsascc2	51655	02:50:30	219080	03:59:17	210948	13:07:53	223521	29:52:06	426000	89610
hsaff4	90284	hsasz1	66302	03:34:10	212900	05:18:15	212772	18:00:17	226680	39:37:25	447792	89601

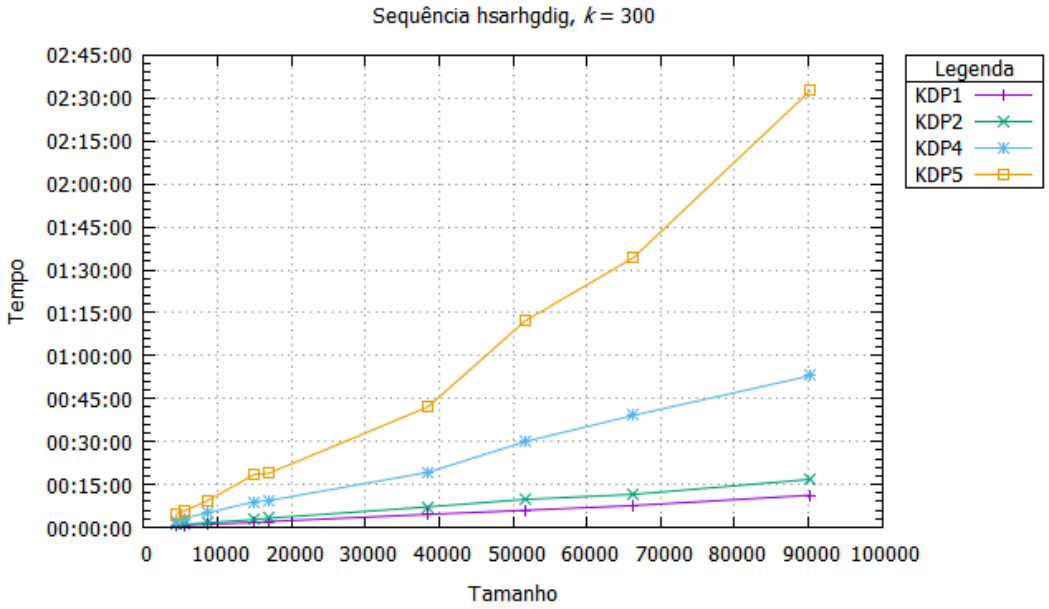


Figura 4.13: *Gráfico de tempo de execução da sequência hsarhgdig,* $k = 300$.

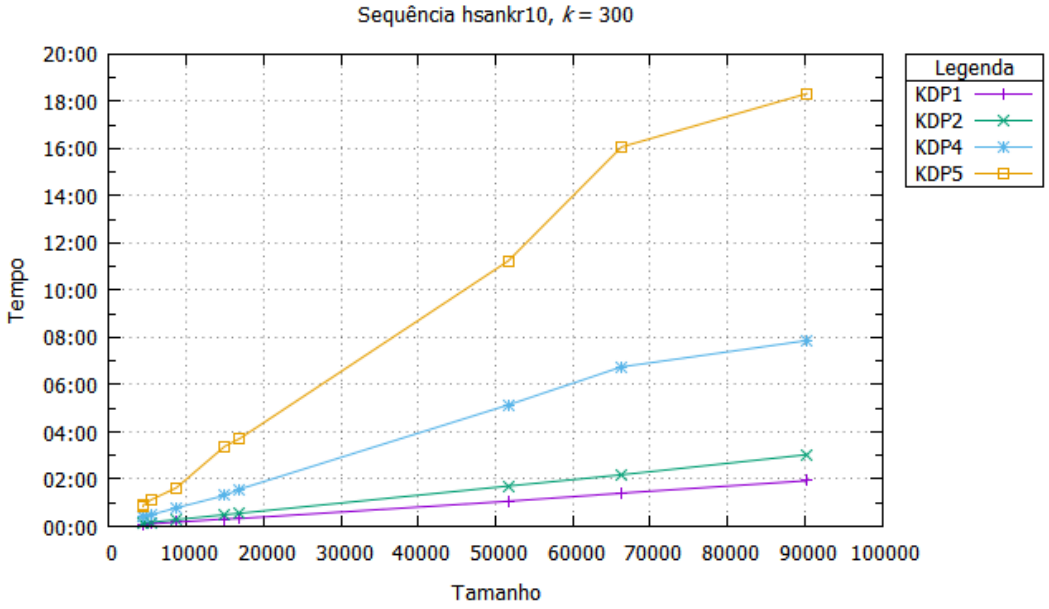


Figura 4.14: *Gráfico de tempo de execução da sequência hsankr10,* $k = 300$.

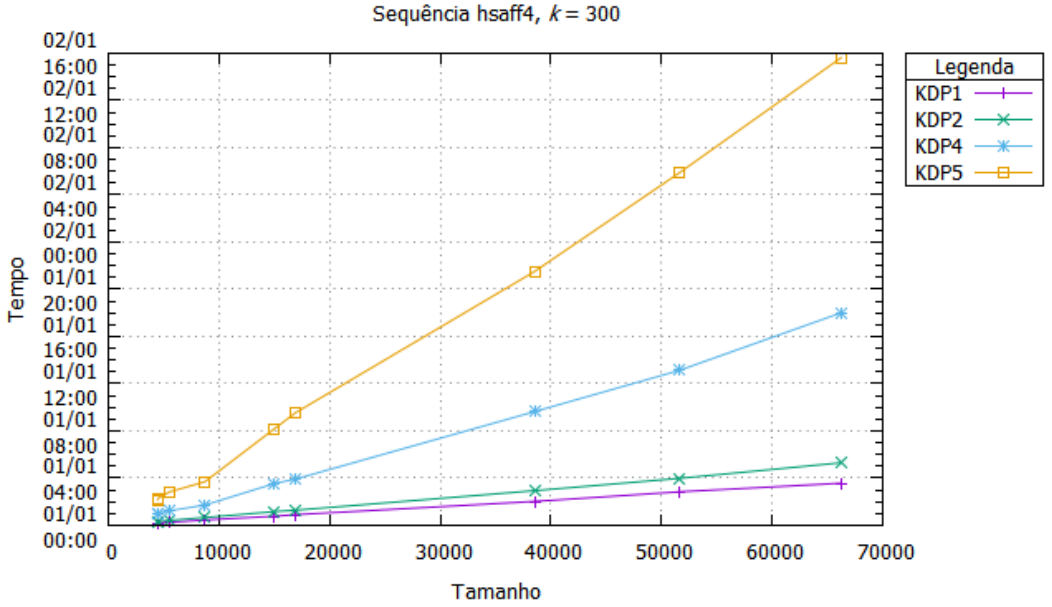


Figura 4.15: *Gráfico de tempo de execução da sequência hsaff4,* $k = 300$.

Tabela 4.12: *Sequências reais de genes do* Homo sapiens*, tamanho pequeno, médio e grande,* $k = 600$.

α		β		KDP1		KDP2		KDP4		KDP5		
Nome	**Tam.**	**Nome**	**Tam.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Ocr**
hsarhgdig	4398	hsascl2	4455	00:00:57	15024	00:01:43	14968	00:04:14	15748	00:07:54	28052	3119
hsarhgdig	4398	hsankrd43	5457	00:01:04	15020	00:01:42	14934	00:05:09	16204	00:09:21	35676	3110
hsarhgdig	4398	hsa1bg	8694	00:01:42	14728	00:02:29	14500	00:07:53	16092	00:14:19	42448	3082
hsarhgdig	4398	hsalg10b	14972	00:02:45	14476	00:04:29	11428	00:14:46	13060	00:30:28	42532	3210
hsarhgdig	4398	hsbad	16877	00:03:23	14808	00:05:49	14868	00:16:45	16556	00:33:51	47332	3091
hsarhgdig	4398	hsankr10	38530	00:07:48	15108	00:12:18	15028	00:33:06	26462	01:12:31	219032	3097
hsarhgdig	4398	hsascc2	51655	00:10:21	14856	00:17:18	14900	00:56:34	19488	02:04:19	98472	3113
hsarhgdig	4398	hsasz1	66302	00:13:14	14684	00:21:41	14688	01:14:09	20604	02:54:49	118076	3184
hsarhgdig	4398	hsaff4	90284	00:18:26	15172	00:27:35	15178	01:27:04	40974	04:10:48	476392	3132
hsankr10	38530	hsarhgdig	4398	00:09:54	135628	00:17:57	135576	00:55:12	139204	01:50:28	197672	37403
hsankr10	38530	hsascl2	4455	00:09:51	137500	00:17:55	137504	00:54:48	141040	01:51:09	199428	37376
hsankr10	38530	hsankrd43	5457	00:13:23	145952	00:22:52	145920	01:09:27	149544	02:21:05	209700	37267
hsankr10	38530	hsa1bg	8694	00:19:58	147984	00:32:12	147954	01:28:54	178498	03:05:33	691824	37309
hsankr10	38530	hsalg10b	14972	00:35:22	150160	00:55:57	149980	02:28:27	183976	06:23:43	290198	37260
hsankr10	38530	hsbad	16877	00:42:15	147528	01:11:21	147564	03:27:57	152076	07:29:23	229508	37324
hsankr10	38530	hsascc2	51655	02:10:06	150216	03:34:54	150208	11:34:22	157620	26:36:07	283208	37295
hsankr10	38530	hsasz1	66302	02:48:00	153256	04:35:38	153404	15:17:47	162100	35:12:13	306860	37260
hsankr10	38530	hsaff4	90284	03:39:14	154072	05:44:53	154040	14:53:33	237748	44:18:36	1634564	37263
hsaff4	90284	hsarhgdig	4398	00:21:26	319492	00:31:02	319412	01:41:36	327488	04:17:19	462904	89138
hsaff4	90284	hsascl2	4455	00:20:58	323652	00:33:30	323612	01:36:25	331676	03:51:52	467072	89123
hsaff4	90284	hsankrd43	5457	00:29:52	342080	00:40:33	342112	02:09:07	350228	05:00:31	487156	89401
hsaff4	90284	hsa1bg	8694	00:53:02	367116	01:18:02	367520	03:22:03	436868	07:13:16	1609804	89017
hsaff4	90284	hsalg10b	14972	01:20:10	363792	02:37:07	363804	06:35:49	372700	16:58:06	522924	88983
hsaff4	90284	hsbad	16877	01:33:09	375920	02:01:50	375988	07:12:55	384992	16:36:01	539364	89032
hsaff4	90284	hsankr10	38530	03:37:55	375420	05:06:35	375448	19:10:18	386312	46:33:42	568924	88991
hsaff4	90284	hsascc2	51655	04:44:53	380200	06:21:06	380200	26:12:57	392792	58:49:35	595376	89007
hsaff4	90284	hsasz1	66302	06:22:56	381048	08:29:18	380932	31:21:21	394952	75:11:25	616004	88955

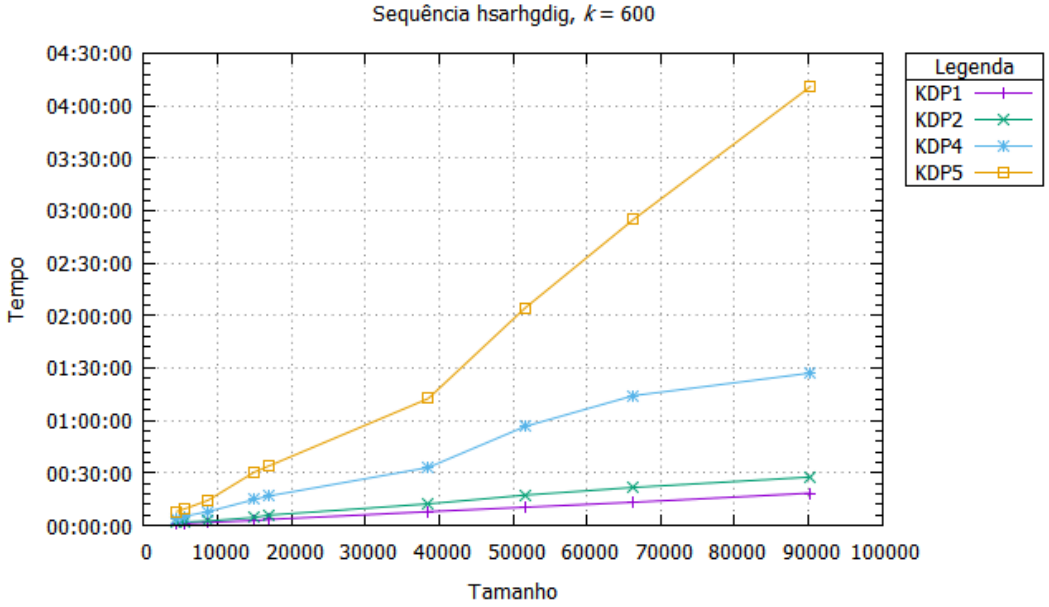


Figura 4.16: *Gráfico de tempo de execução da sequência hsarhgdig,* $k = 600$.

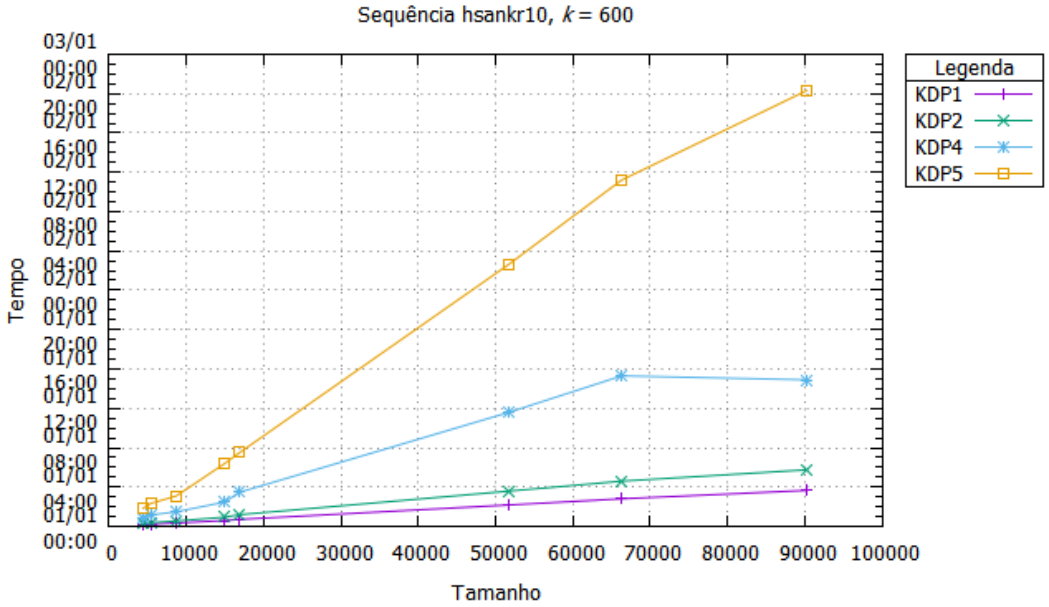


Figura 4.17: *Gráfico de tempo de execução da sequência hsankr10,* $k = 600$.

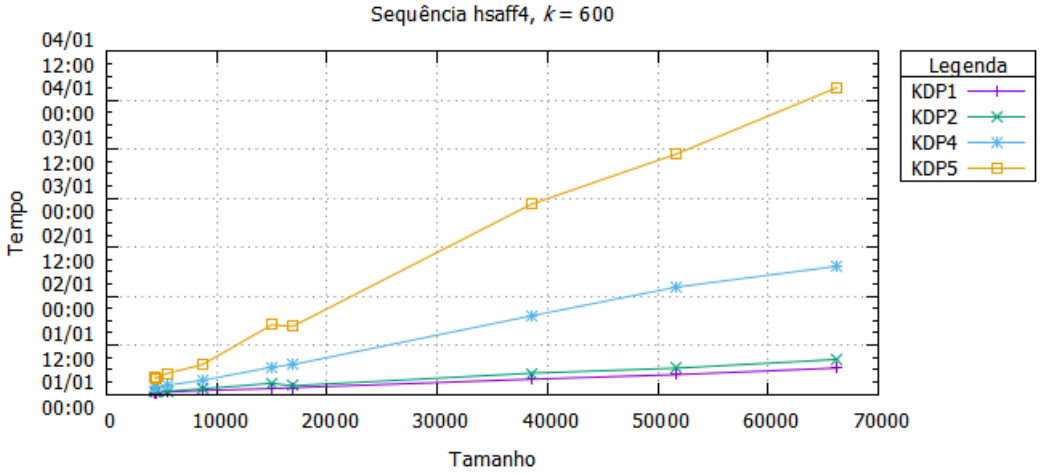


Figura 4.18: *Gráfico de tempo de execução da sequência hsaff4, $k = 600$.*

Verificamos que o comportamento dos programas é similar àquele obtido com dados artificias gerados de forma aleatória, e com isso conclusões parecidas podem ser obtidas. Perceba que, para $k = 30$, o programa KDP1 é cerca de 30% mais rápido, na média, que o programa KDP2. Esse, por sua vez, é cerca de 295%, mais rápido, na média, que o programa KDP4. Quando $k = 300$, o programa KDP1 é cerca de 57% mais rápido que o programa KDP2, na média, e o programa KDP2 é cerca de 360% mais rápido que o programa KDP4, na média. Quando $k = 600$, o programa KDP1 é cerca de 59% mais rápido que o programa KDP2, que é cerca de 380% mais rápido que o programa KDP4.

Tabela 4.13: *Similaridade entre as sequências reais de genes do Homo sapiens, tamanho pequeno, médio e grande.*

α		β		Similaridade
Nome	**Tam.**	**Nome**	**Tam.**	
hsaff4	90284	hsascl2	4455	3.3%
hsarhgdig	4398	hsaff4	90284	3.4%
hsaff4	90284	hsarhgdig	4398	3.4%
hsaff4	90284	hsankrd43	5457	4%
hsarhgdig	4398	hsasz1	66302	4.3%
hsarhgdig	4398	hsascc2	51655	6%
hsaff4	90284	hsa1bg	8694	6.6%
hsarhgdig	4398	hsankr10	38530	7.6%
hsankr10	38530	hsarhgdig	4398	7.6%
hsankr10	38530	hsascl2	4455	7.6%
hsankr10	38530	hsankrd43	5457	9.1%
hsaff4	90284	hsalg10b	14972	14.1%
hsaff4	90284	hsbad	16877	15.5%
hsankr10	38530	hsa1bg	8694	15.9%
hsarhgdig	4398	hsalg10b	14972	18.1%
hsarhgdig	4398	hsbad	16877	18.5%
hsankr10	38530	hsalg10b	14972	25%
hsankr10	38530	hsbad	16877	29.6%
hsankr10	38530	hsaff4	90284	31.1%
hsaff4	90284	hsankr10	38530	31.1%
hsarhgdig	4398	hsa1bg	8694	34.2%
hsaff4	90284	hsascc2	51655	38.4%
hsankr10	38530	hsasz1	66302	38.8%
hsankr10	38530	hsascc2	51655	43.7%
hsarhgdig	4398	hsankrd43	5457	43.9%
hsaff4	90284	hsasz1	66302	44.8%
hsarhgdig	4398	hsascl2	4455	48.4%

O programa KDP5 é o programa mais lento no geral e o que mais utiliza memória, tornando o seu uso injustificável nesse caso. O mesmo ocorre com o programa KDP4, que é cerca de três vezes mais lento que o KDP1 e o KDP2. Como já dito, acreditamos que essa vantagem do KDP1 e KDP2 seja obtida por conta dos algoritmos implementados por eles, que utilizam apenas estruturas de dados sequenciais, não necessitando fazer acesso direto a endereços que não estão na memória

cache, ao contrário dos programas KDP4 e KDP5, cuja computação do LCE pode ocasionar vários *cache miss*, prejudicando o desempenho.

A quantidade de memória utilizada pelo programas KDP1, quando $k = 30$, é praticamente idêntico ao que utiliza o programa KDP2. Nesses casos, o programa KDP4 utiliza, na média, cerca de 34% mais memória que os programas KDP1 e KDP2 e o programa KDP5 utiliza, na média cerca de 547% mais memória que o programa KDP4. Quando $k = 300$, a quantidade de memória utilizada pelos programas KDP1 e KDP2, na média, é praticamente idêntica, enquanto que o programa KDP4 utiliza, na média, cerca de 14% mais memória que os programas KDP1 e KDP2 e o programa KDP5 utiliza, na média, cerca de 207% mais memória que o programa KDP4. Quando $k = 600$, a quantidade de memória utilizada pelos programas KDP1 e KDP2 é, na média, idêntica, enquanto que o programa KDP4 utiliza, na média, cerca de 33% mais memória que os programas KDP1 e KDP2 e o programa KDP5 utiliza, na média, cerca de 165% mais memória que o programa KDP4. Tais resultados são, novamente, parecidos aos obtidos pelos testes com sequências artificiais.

Outro fator importante já destacado é o tamanho médio da maior extensão comum entre duas sequências que, como já observado, é pequeno em sequências com baixa similaridade, onde o tamanho máximo do LCE nessas sequências é cerca de 32 caracteres, na média. Já a média do tamanho da maior extensão comum, levando em consideração os valores iguais a zero, é de 0.33 e de 1.35, levando em consideração apenas valores maiores que zero. Acreditamos que devido a isso, o algoritmo ingênuo de computação do LCE seja melhor que os algoritmos que utilizam estruturas de dados como árvores de sufixo e vetores de sufixo.

Percebemos ainda que os resultados obtidos nos testes com dados reais são parecidos aos testes com dados artificiais. Acreditamos que isso ocorra por conta da baixa similaridade entre as sequências, que varia de 3.3% a 48.4%, com uma média de 20.5%. De forma detalhada, as similaridades entre as sequências reais, para os casos pequeno, médio e grande, são apresentadas na Tabela 4.13. Os dados completos podem ser vistos no Apêndice B, Tabela B.4. Perceba que nenhuma das sequências usadas nos testes reais atinge 50% de similaridade com qualquer outra, muito parecida aos testes com dados artificiais.

Finalmente, a análise da complexidade de tempo, na prática, do programa KDP1, $O(mkn)$, obtida nos testes artificias quando as sequências têm baixa similaridade é confirmada nos testes com sequencias reais. Observamos novamente que a matriz D é preenchida até uma certa linha. Mais detalhadamente, considerando todos os testes, quando $k = 30$, o preenchimento termina, na maioria das vezes, entre as linhas 67 e 108, observando que o tamanho médio das ocorrências nesse caso é de 81 caracteres. Quando $k = 300$, o preenchimento termina, na maioria das vezes, entre as linhas 567 e 750, observando que o tamanho médio das ocorrências nesse caso é de 654 caracteres. Quando $k = 600$, o preenchimento termina, na maioria das vezes, entre as linhas 1106 e 1419, com o tamanho médio das ocorrências de 1268 caracteres.

4.4 Dados reais complementares

Ao final dos testes realizados com dados artificiais e reais, percebemos que não havia nenhum caso onde as sequências envolvidas possuíam mais do que 50% de similaridade entre elas. Por conta disso resolvemos executar testes complementares com sequências reais, cuidadosamente selecionadas, no intuito de obter casos onde as sequências tenham entre 50% e 99% de similaridade e assim uma

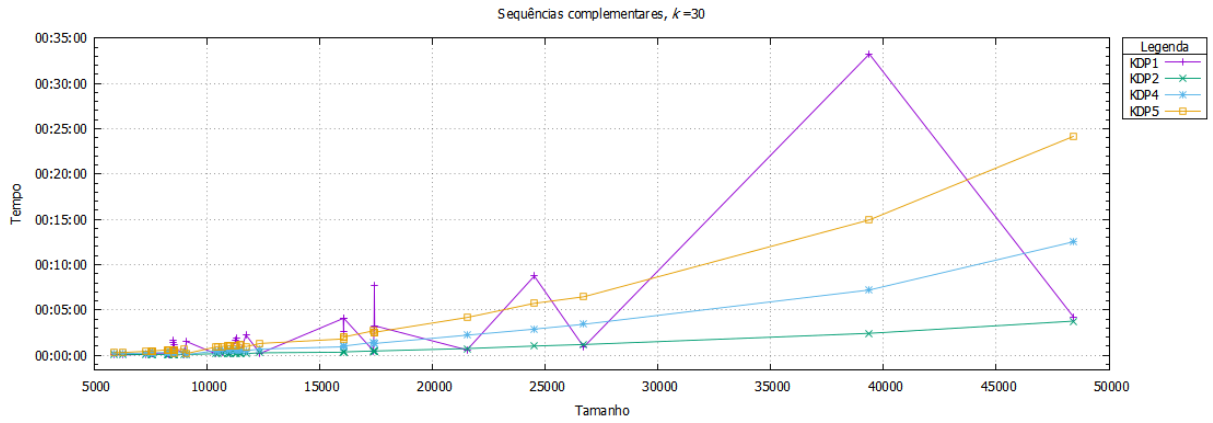


Figura 4.19: *Gráfico de tempo de execução dos testes reais complementares,* $k = 30$.

conclusão mais precisa sobre o comportamento dos programas.

Para executar os testes complementares com dados reais utilizamos 45 sequências, obtidas do repositório de genes do NCBI no endereço https://www.ncbi.nlm.nih.gov/, utilizando a ferramenta BLAST para encontrar sequências similares. Das sequências obtidas, 17 são do *Homo sapiens* (hs), 13 são do *Mus musculus* (mm), 4 são do *Pan troglodytes* (pt), 1 é do *Macaca fascicularis* (mf), 1 é do *Gorila gorila gorila* (ggg), 1 é do *Bison bison bison* (bbb), 1 é do *Cystic fibrosis* (cf), 1 é do *Papio hamadryas* (ph), 1 é do *Pongo abelii* (pa). Cada uma delas contém um gene específico, de tamanho específico, e foram agrupadas em pares conforme a similaridade entre elas, totalizando 90 casos de testes, onde a sequência é usada um vez como α e outra como β. Todos os casos podem ser vistos na Tabela 4.14.

Foram executadas três séries de testes, variando o valor de k, onde cada caso de teste foi processado três vezes, extraindo-se a média do tempo de execução e da quantidade de memória utilizada. Os resultados desses testes são apresentados nas Tabelas 4.15, para $k = 30$, 4.16, para $k = 300$ e 4.17, para $k = 600$.

As Figuras 4.19, 4.20 e 4.21 apresentam os gráficos do tempo de execução dos programas com base nos dados das Tabelas 4.15, 4.16 e 4.17, respectivamente. Para obter tais gráficos, somamos o tamanho das sequências α e β e extraímos a média dos tempos nos casos com tamanho iguais, afim de garantir uma visualização precisa dos dados.

Perceba que houve muitos casos em que não foi obtida nenhuma ocorrência de segmentos específicos. Mais detalhadamente, quando $k = 300$, alguns pares com mais do que 80% de similaridade não apresentaram ocorrências de segmentos específicos. Quando $k = 600$, não foi encontrada nenhuma ocorrência de segmento específico para os pares com mais do que 90% de similaridade e para alguns pares com mais do que 80% de similaridade. Em todos esses casos, acreditamos que o tamanho da sequência alvo seja extremamente pequeno para obter segmentos específicos com 300 e 600 diferenças, visto que, quando há uma alta similaridade entre as sequências a quantidade de diferenças é menor e, com isso, o tamanho do segmento específico é maior, necessitando de sequências grandes para ser obtido.

Com base nos testes complementares realizados com sequências reais, percebemos que o programa KDP1 é ligeiramente mais rápido que o programa KDP2 até uma certa similaridade entre as sequências. Mais precisamente, quando $k = 30$ o programa KDP1 é cerca de 37% mais rápido, na

Tabela 4.14: *Similaridade entre as sequências reais complementares.*

α		β		$\alpha + \beta$	**Similaridade**
Nome	**Tam.**	**Nome**	**Tam.**		
bet1mm	12024	bet1hs	14691	26715	50,50%
bet1hs	14691	bet1mm	12024	26715	50,50%
dppa5mm	3099	dppa5hs	3167	6266	52,50%
dppa5hs	3167	dppa5mm	3099	6266	52,50%
dnajc4hs	6001	dnajc4mm	6362	12363	54,20%
dnajc4mm	6362	dnajc4hs	6001	12363	54,20%
kcnk4hs	10712	kcnk4mm	10827	21539	55%
kcnk4mm	10827	kcnk4hs	10712	21539	55%
hoxa4cf	3300	hoxa4mm	4017	7317	56,40%
hoxa4mm	4017	hoxa4cf	3300	7317	56,40%
nr2e1mm	23619	nr2e1hs	24800	48419	58,30%
nr2e1hs	24800	nr2e1mm	23619	48419	58,30%
csf2hs	4375	csf2mm	4630	9005	60%
csf2mm	4630	csf2hs	4375	9005	60%
or51a7hs	2940	or51a7mm	2940	5880	62,50%
or51a7mm	2940	or51a7hs	2940	5880	62,50%
hoxa4cf	3300	hoxa4hs	4274	7574	62,50%
hoxa4cf	3300	hoxa4ph	4275	7575	62,50%
hoxa4hs	4274	hoxa4cf	3300	7574	62,50%
hoxa4ph	4275	hoxa4cf	3300	7575	62,50%
hoxa4cf	3300	hoxa4pa	4249	7549	62,80%
hoxa4cf	3300	hoxa4pt	4271	7571	62,80%
hoxa4pa	4249	hoxa4cf	3300	7549	62,80%
hoxa4pt	4271	hoxa4cf	3300	7571	62,80%
cdx2mm	8351	cdx2hs	9041	17392	63,10%
cdx2hs	9041	cdx2mm	8351	17392	63,10%
gpr137mm	5411	gpr137hs	5634	11045	66,70%
gpr137hs	5634	gpr137mm	5411	11045	66,70%
hoxa6hs	4254	hoxa6mm	4261	8515	67,40%
hoxa6mm	4261	hoxa6hs	4254	8515	67,40%
ctgfmm	5175	ctgfhs	5204	10379	68,50%
ctgfhs	5204	ctgmm	5175	10379	68,50%
lrrc4bbb	5055	lrrc4mm	5622	10677	70,70%
lrrc4mm	5622	lrrc4bbb	5055	10677	70,70%
hoxa4mm	4017	hoxa4hs	4274	8291	71,70%
hoxa4hs	4274	hoxa4mm	4017	8291	71,70%
hoxa4mm	4017	hoxa4ph	4275	8292	72%
hoxa4ph	4275	hoxa4mm	4017	8292	72%
hoxa4mm	4017	hoxa4pt	4271	8288	72,10%
hoxa4pt	4271	hoxa4mm	4017	8288	72,10%
hoxa4mm	4017	hoxa4pa	4249	8266	72,20%
hoxa4pa	4249	hoxa4mm	4017	8266	72,20%
lrrc4bbb	5055	lrrc4pa	5861	10916	73,30%
lrrc4bbb	5055	lrrc4hs	5879	10934	73,30%
lrrc4pa	5861	lrrc4bbb	5055	10916	73,30%
lrrc4hs	5879	lrrc4bbb	5055	10934	73,30%
lrrc4bbb	5055	lrrc4mf	5426	10481	78,90%
lrrc4mf	5426	lrrc4bbb	5055	10481	78,90%
lrrc4mf	5426	lrrc4mm	5622	11048	80,30%
lrrc4mm	5622	lrrc4mf	5426	11048	80,30%
kcnk4ggg	7352	kcnk4pa	8715	16067	81,40%
kcnk4pa	8715	kcnk4ggg	7352	16067	81,40%
lrrc4mm	5622	lrrc4pa	5861	11483	81,90%
lrrc4mm	5622	lrrc4hs	5879	11501	81,90%
lrrc4pa	5861	lrrc4mm	5622	11483	81,90%
lrrc4hs	5879	lrrc4mm	5622	11501	81,90%
kcnk4ggg	7352	kcnk4pt	8702	16054	82,80%
kcnk4pt	8702	kcnk4ggg	7352	16054	82,80%
ctsdpt	11284	ctsdhs	13238	24522	82,80%
ctsdhs	13238	ctsdpt	11284	24522	82,80%
kcnk4ggg	7352	kcnk4hs1	8730	16082	82,90%
kcnk4hs1	8730	kcnk4ggg	7352	16082	82,90%
lrrc4mf	5426	lrrc4pa	5861	11287	89,40%
lrrc4pa	5861	lrrc4mf	5426	11287	89,40%
lrrc4mf	5426	lrrc4hs	5879	11305	89,70%
lrrc4hs	5879	lrrc4mf	5426	11305	89,70%
hoxa4pa	4249	hoxa4ph	4275	8524	96,10%
hoxa4hs	4274	hoxa4ph	4275	8549	96,10%
hoxa4ph	4275	hoxa4pa	4249	8524	96,10%
hoxa4ph	4275	hoxa4hs	4274	8549	96,10%
kcnk4pt	8702	kcnk4pa	8715	17417	96,30%
kcnk4pa	8715	kcnk4pt	8702	17417	96,30%
kcnk4pa	8715	kcnk4hs1	8730	17445	96,40%
kcnk4hs1	8730	kcnk4pa	8715	17445	96,40%
hoxa4pt	4271	hoxa4ph	4275	8546	96,60%
hoxa4ph	4275	hoxa4pt	4271	8546	96,60%
hoxa4pa	4249	hoxa4hs	4274	8523	97,20%
hoxa4hs	4274	hoxa4pa	4249	8523	97,20%
hoxa4pa	4249	hoxa4pt	4271	8520	97,70%
hoxa4pt	4271	hoxa4pa	4249	8520	97,70%
lrrc4pa	5861	lrrc4hs	5879	11740	97,90%
lrrc4hs	5879	lrrc4pa	5861	11740	97,90%
il3pt	4548	il3hs	4550	9098	98%
il3hs	4550	il3pt	4548	9098	98%
ddx18pt	19676	ddx18hs	19699	39375	98,20%
ddx18hs	19699	ddx18pt	19676	39375	98,20%
kcnk4pt	8702	kcnk4hs1	8730	17432	98,60%
kcnk4hs1	8730	kcnk4pt	8702	17432	98,60%
hoxa4pt	4271	hoxa4hs	4274	8545	99,10%
hoxa4hs	4274	hoxa4pt	4271	8545	99,10%

Tabela 4.15: *Sequências reais complementares com similaridade acima de 50%, $k = 30$.*

α		β		KDP1		KDP2		KDP4		KDP5		
Nome	**Tam.**	**Nome**	**Tam.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Ocr**
or51a7hs	2940	or51a7mm	2940	00:00:03	2845	00:00:03	2839	00:00:10	3280	00:00:17	11639	2855
or51a7mm	2940	or51a7hs	2940	00:00:04	2848	00:00:03	2837	00:00:09	3280	00:00:17	11637	2865
dppa5mm	3099	dppa5hs	3167	00:00:02	2583	00:00:04	2579	00:00:10	3035	00:00:19	11652	3027
dppa5hs	3167	dppa5mm	3099	00:00:02	2581	00:00:04	2579	00:00:11	3039	00:00:18	11651	3093
hoxa4cf	3300	hoxa4mm	4017	00:00:07	3380	00:00:05	3372	00:00:14	3912	00:00:27	14576	3193
hoxa4cf	3300	hoxa4pa	4249	00:00:12	4172	00:00:05	4164	00:00:16	4720	00:00:27	16008	3124
hoxa4cf	3300	hoxa4pt	4271	00:00:13	4436	00:00:05	4432	00:00:16	4980	00:00:27	16348	3113
hoxa4cf	3300	hoxa4hs	4274	00:00:12	4176	00:00:05	4168	00:00:16	4716	00:00:27	16040	3116
hoxa4cf	3300	hoxa4ph	4275	00:00:12	4176	00:00:05	4168	00:00:16	4716	00:00:28	16056	3113
hoxa4mm	4017	hoxa4cf	3300	00:00:06	3648	00:00:05	3640	00:00:14	4176	00:00:26	14840	3950
hoxa4mm	4017	hoxa4pa	4249	00:00:09	3648	00:00:06	3644	00:00:19	4276	00:00:36	16340	3950
hoxa4mm	4017	hoxa4pt	4271	00:00:09	3652	00:00:06	3644	00:00:19	4284	00:00:35	16368	3937
hoxa4mm	4017	hoxa4hs	4274	00:00:08	3652	00:00:06	3644	00:00:18	4284	00:00:34	16364	3937
hoxa4mm	4017	hoxa4ph	4275	00:00:09	4548	00:00:07	3908	00:00:19	4548	00:00:35	16652	3931
hoxa4pa	4249	hoxa4cf	3300	00:00:11	4708	00:00:06	4696	00:00:15	5252	00:00:29	16540	4178
hoxa4pa	4249	hoxa4mm	4017	00:00:09	3916	00:00:06	3904	00:00:19	4548	00:00:35	16604	4164
hoxa4pa	4249	hoxa4pt	4271	00:01:31	23192	00:00:05	23184	00:00:15	23844	00:00:27	38340	3008
hoxa4pa	4249	hoxa4hs	4274	00:01:24	18968	00:00:06	18696	00:00:17	19356	00:00:37	33804	3669
hoxa4pa	4249	hoxa4ph	4275	00:00:51	11840	00:00:05	11832	00:00:16	12492	00:00:31	26652	3270
hoxa6hs	4254	hoxa6mm	4261	00:00:09	3919	00:00:07	3911	00:00:20	4572	00:00:36	16852	4187
hoxa6mm	4261	hoxa6hs	4254	00:00:09	3920	00:00:07	3911	00:00:20	4571	00:00:36	16849	4154
hoxa4pt	4271	hoxa4cf	3300	00:00:11	4704	00:00:05	4700	00:00:16	5248	00:00:28	16620	4200
hoxa4pt	4271	hoxa4mm	4017	00:00:09	3916	00:00:07	3908	00:00:19	4548	00:00:35	16628	4194
hoxa4pt	4271	hoxa4pa	4249	00:01:43	25568	00:00:06	25556	00:00:20	26484	00:00:38	40980	4167
hoxa4pt	4271	hoxa4hs	4274	00:01:08	16328	00:00:02	18164	00:00:13	16984	00:00:13	31748	1466
hoxa4pt	4271	hoxa4ph	4275	00:00:57	13160	00:00:07	13152	00:00:18	13548	00:00:36	27996	3886
hoxa4hs	4274	hoxa4cf	3300	00:00:10	4705	00:00:05	4699	00:00:16	5249	00:00:28	16569	4203
hoxa4hs	4274	hoxa4mm	4017	00:00:08	3916	00:00:06	3820	00:00:18	4552	00:00:34	16636	4197
hoxa4hs	4274	hoxa4pa	4249	00:01:27	19232	00:00:07	19224	00:00:20	20148	00:00:38	34596	4170
hoxa4hs	4274	hoxa4pt	4271	00:01:09	16328	00:00:03	18168	00:00:07	16984	00:00:14	31748	1466
hoxa4hs	4274	hoxa4ph	4275	00:00:50	11576	00:00:06	11568	00:00:19	12232	00:00:36	26364	3914
hoxa4ph	4275	hoxa4cf	3300	00:00:10	4708	00:00:05	4700	00:00:16	5252	00:00:28	16584	4204
hoxa4ph	4275	hoxa4mm	4017	00:00:09	3916	00:00:07	3908	00:00:19	4552	00:00:34	16656	4192
hoxa4ph	4275	hoxa4pa	4249	00:00:59	13424	00:00:06	13152	00:00:20	14072	00:00:38	28240	4203
hoxa4ph	4275	hoxa4pt	4271	00:00:57	13156	00:00:06	13148	00:00:19	13548	00:00:35	27996	3882
hoxa4ph	4275	hoxa4hs	4274	00:00:50	11576	00:00:06	11568	00:00:19	12232	00:00:35	26360	3906
csf2hs	4375	csf2mm	4630	00:00:06	3389	00:00:07	3383	00:00:23	3816	00:00:41	16536	4300
il3pt	4548	il3hs	4550	00:01:33	21611	00:00:05	25828	00:00:11	22303	00:00:21	37671	2192
il3hs	4550	il3pt	4548	00:01:30	21084	00:00:04	20812	00:00:11	21512	00:00:21	37144	2138
csf2mm	4630	csf2hs	4375	00:00:06	3396	00:00:08	3387	00:00:23	4081	00:00:40	16797	4553
lrrc4bbb	5055	lrrc4mf	5426	00:00:26	6303	00:00:11	6299	00:00:30	7095	00:00:58	23015	4872
lrrc4bbb	5055	lrrc4mm	5622	00:00:21	5508	00:00:11	5505	00:00:31	6319	00:00:58	22200	4922
lrrc4bbb	5055	lrrc4pa	5861	00:00:28	6303	00:00:11	6299	00:00:32	7127	00:01:00	23580	4874
lrrc4bbb	5055	lrrc4hs	5879	00:00:29	6303	00:00:12	6297	00:00:33	7125	00:01:02	23647	4875
ctgfmm	5175	ctgfhs	5204	00:00:11	3927	00:00:10	3919	00:00:29	4716	00:00:54	19445	5107
ctgfhs	5204	ctgmm	5175	00:00:11	3925	00:00:10	4184	00:00:29	4716	00:00:54	19443	5108
gpr137mm	5411	gpr137hs	5634	00:00:13	4192	00:00:12	4183	00:00:34	5024	00:01:03	20688	5336
lrrc4mf	5426	lrrc4bbb	5055	00:00:25	6565	00:00:11	6563	00:00:30	7357	00:00:56	23279	5270
lrrc4mf	5426	lrrc4mm	5622	00:00:27	6304	00:00:12	6299	00:00:33	7135	00:01:03	23903	5253
lrrc4mf	5426	lrrc4pa	5861	00:01:43	17127	00:00:11	17121	00:00:32	17984	00:01:02	36684	4871
lrrc4mf	5426	lrrc4hs	5879	00:02:00	19504	00:00:11	19585	00:00:32	20360	00:01:03	39425	4844
lrrc4mm	5622	lrrc4bbb	5055	00:00:20	5511	00:00:12	5683	00:00:31	6319	00:01:00	22204	5552
lrrc4mm	5622	lrrc4mf	5426	00:00:27	6568	00:00:12	6559	00:00:34	7401	00:01:04	24171	5550
lrrc4mm	5622	lrrc4pa	5861	00:00:29	6855	00:00:13	6561	00:00:37	7435	00:01:09	24775	5550
lrrc4mm	5622	lrrc4hs	5879	00:00:29	6568	00:00:13	6559	00:00:36	7440	00:01:12	24824	5550
gpr137hs	5634	gpr137mm	5411	00:00:13	4191	00:00:11	4183	00:00:34	5028	00:01:02	20688	5558
lrrc4pa	5861	lrrc4bbb	5055	00:00:25	6567	00:00:12	6559	00:00:33	7395	00:01:01	23843	5722
lrrc4pa	5861	lrrc4mf	5426	00:01:35	17393	00:00:12	17391	00:00:32	18248	00:01:03	36943	5302
lrrc4pa	5861	lrrc4mm	5622	00:00:28	6572	00:00:13	6565	00:00:36	7437	00:01:07	24784	5690
lrrc4pa	5861	lrrc4hs	5879	00:02:12	22676	00:00:11	22669	00:00:30	23299	00:00:57	43365	4526
lrrc4hs	5879	lrrc4bbb	5055	00:00:26	6565	00:00:13	6563	00:00:33	7396	00:01:03	23916	5740
lrrc4hs	5879	lrrc4mf	5426	00:01:53	20036	00:00:13	20031	00:00:33	20888	00:01:07	39427	5290
lrrc4hs	5879	lrrc4mm	5622	00:00:28	6571	00:00:12	6564	00:00:35	7440	00:01:07	24828	5708
lrrc4hs	5879	lrrc4pa	5861	00:02:16	22676	00:00:12	22845	00:00:31	23828	00:00:58	43631	4540
dnajc4hs	6001	dnajc4mm	6362	00:00:12	3932	00:00:16	4191	00:00:43	4861	00:01:20	22512	5919
dnajc4mm	6362	dnajc4hs	6001	00:00:12	4193	00:00:16	4189	00:00:41	5128	00:01:18	22515	6290
kcnk4ggg	7352	kcnk4pt	8702	00:04:23	31135	00:00:19	31127	00:00:54	32325	00:01:47	59601	5359
kcnk4ggg	7352	kcnk4pa	8715	00:02:47	18464	00:00:24	18457	00:01:04	19661	00:02:03	46136	6380
kcnk4ggg	7352	kcnk4hs1	8730	00:05:04	33631	00:00:22	33771	00:01:00	34701	00:01:59	62151	6012
cdx2mm	8351	cdx2hs	9041	00:00:29	5535	00:00:28	5528	00:01:25	6899	00:02:40	31259	8276
kcnk4pt	8702	kcnk4ggg	7352	00:03:45	31140	00:00:21	31484	00:00:58	32339	00:01:50	59608	6714
kcnk4pt	8702	kcnk4pa	8715	00:03:09	20851	00:00:29	20845	00:01:18	22223	00:02:33	51231	7697
kcnk4pt	8702	kcnk4hs1	8730	00:07:41	50947	00:00:27	50940	00:01:14	53376	00:02:25	82052	7365
kcnk4pa	8715	kcnk4ggg	7352	00:02:25	19259	00:00:24	19255	00:01:06	20463	00:02:06	46932	7743
kcnk4pa	8715	kcnk4pt	8702	00:03:06	20851	00:00:28	20845	00:01:17	22223	00:02:29	51232	7702
kcnk4pa	8715	kcnk4hs1	8730	00:03:16	21629	00:00:29	21636	00:01:19	23016	00:02:34	52176	7727
kcnk4hs1	8730	kcnk4ggg	7352	00:03:04	34307	00:00:22	34301	00:01:03	35769	00:02:03	62947	7378
kcnk4hs1	8730	kcnk4pt	8702	00:07:43	51739	00:00:27	51645	00:01:15	52847	00:02:29	82844	7377
kcnk4hs1	8730	kcnk4pa	8715	00:03:15	21644	00:00:28	21639	00:01:17	23016	00:02:30	52175	7741
cdx2hs	9041	cdx2mm	8351	00:00:29	6067	00:00:29	6059	00:01:25	7432	00:02:44	31791	8969
kcnk4hs	10712	kcnk4mm	10827	00:00:35	6080	00:00:44	6071	00:02:14	7848	00:04:12	37892	10631
kcnk4mm	10827	kcnk4hs	10712	00:00:36	6815	00:00:44	6335	00:02:12	8023	00:04:09	37984	10757
ctsdpt	11284	ctsdhs	13238	00:09:04	37767	00:01:00	37763	00:02:48	39676	00:05:31	80761	10535
bet1mm	12024	bet1hs	14691	00:00:52	6619	00:01:09	7307	00:03:25	7265	00:06:26	45692	11948
ctsdhs	13238	ctsdpt	11284	00:08:25	39357	00:01:03	39352	00:02:58	41267	00:05:58	82353	13162
bet1hs	14691	bet1mm	12024	00:00:58	7687	00:01:14	7677	00:03:25	9764	00:06:28	46755	14617
ddx18pt	19676	ddx18hs	19699	00:33:14	97227	00:02:27	97485	00:07:16	100432	00:15:37	166791	16795
ddx18hs	19699	ddx18pt	19676	00:33:15	96963	00:02:23	97837	00:07:09	100960	00:14:16	166261	16827
nr2e1mm	23619	nr2e1hs	24800	00:04:18	14631	00:03:46	14624	00:11:51	18563	00:24:00	86928	23546
nr2e1hs	24800	nr2e1mm	23619	00:04:13	15164	00:03:46	15156	00:13:12	19101	00:24:18	87464	24723

Tabela 4.16: *Sequências reais complementares com similaridade acima de 50%, $k = 300$.*

α		β		KDP1		KDP2		KDP4		KDP5		
Nome	**Tam.**	**Nome**	**Tam.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Ocr**
or51a7hs	2940	or51a7mm	2940	00:00:23	8124	00:00:26	8119	00:01:10	8560	00:02:02	16920	2194
or51a7mm	2940	or51a7hs	2940	00:00:23	8128	00:00:27	8119	00:01:12	8560	00:02:07	16919	2276
dppa5mm	3099	dppa5hs	3167	00:00:19	6807	00:00:34	6801	00:01:26	7260	00:02:30	15875	2492
dppa5hs	3167	dppa5mm	3099	00:00:19	7072	00:00:34	7067	00:01:26	7525	00:02:29	16140	2557
hoxa4cf	3300	hoxa4mm	4017	00:00:34	8395	00:00:30	8391	00:01:24	8925	00:02:33	19595	1770
hoxa4cf	3300	hoxa4pa	4249	00:00:25	6021	00:00:15	6544	00:00:43	7095	00:01:19	17855	793
hoxa4cf	3300	hoxa4pt	4271	00:00:25	6292	00:00:15	6456	00:00:44	6829	00:01:12	18464	770
hoxa4cf	3300	hoxa4hs	4274	00:00:25	6288	00:00:16	6811	00:00:44	7355	00:01:14	17888	795
hoxa4cf	3300	hoxa4ph	4275	00:00:24	6380	00:00:16	6544	00:00:42	7096	00:01:12	18431	791
hoxa4mm	4017	hoxa4cf	3300	00:00:40	11832	00:00:46	12088	00:02:04	12620	00:03:47	23292	3419
hoxa4mm	4017	hoxa4pa	4249	00:00:55	12628	00:00:53	12535	00:02:27	13255	00:04:25	25315	2974
hoxa4mm	4017	hoxa4pt	4271	00:00:54	12364	00:00:51	12359	00:02:20	12996	00:04:18	25080	2945
hoxa4mm	4017	hoxa4hs	4274	00:01:00	12364	00:00:48	12353	00:02:24	12996	00:04:22	25079	2971
hoxa4mm	4017	hoxa4ph	4275	00:00:55	12361	00:00:51	12356	00:02:20	12996	00:04:22	25103	2956
hoxa4pa	4249	hoxa4cf	3300	00:01:02	18172	00:00:57	17056	00:02:34	18979	00:04:16	30267	3659
hoxa4pa	4249	hoxa4mm	4017	00:01:05	13683	00:01:00	13679	00:02:22	14316	00:04:40	26371	3132
hoxa4pa	4249	hoxa4pt	4271	00:00:00	1284	00:00:00	1280	00:00:00	1932	00:00:00	16428	0
hoxa4pa	4249	hoxa4hs	4274	00:00:00	1284	00:00:00	1276	00:00:00	1932	00:00:00	16384	0
hoxa4pa	4249	hoxa4ph	4275	00:00:00	1280	00:00:00	1280	00:00:00	1936	00:00:00	16096	0
hoxa6hs	4254	hoxa6mm	4261	00:00:53	12368	00:00:56	12360	00:02:42	13019	00:04:59	25301	3434
hoxa6mm	4261	hoxa6hs	4254	00:00:51	11840	00:00:53	11831	00:02:32	12667	00:04:38	24769	3262
hoxa4pt	4271	hoxa4cf	3300	00:01:08	18700	00:00:51	18607	00:02:20	19244	00:04:13	30348	3675
hoxa4pt	4271	hoxa4mm	4017	00:01:03	13947	00:00:55	13943	00:02:34	14583	00:04:28	28213	3198
hoxa4pt	4271	hoxa4pa	4249	00:00:00	1284	00:00:00	1280	00:00:00	1936	00:00:00	16432	0
hoxa4pt	4271	hoxa4hs	4274	00:00:00	1284	00:00:00	1280	00:00:00	1940	00:00:00	16704	0
hoxa4pt	4271	hoxa4ph	4275	00:00:00	1284	00:00:00	1280	00:00:00	1940	00:00:00	16120	0
hoxa4hs	4274	hoxa4cf	3300	00:00:59	18171	00:00:49	18255	00:02:13	18711	00:04:01	30299	3683
hoxa4hs	4274	hoxa4mm	4017	00:01:13	13860	00:00:59	13764	00:02:30	14584	00:04:58	26667	3220
hoxa4hs	4274	hoxa4pa	4249	00:00:00	1284	00:00:00	1280	00:00:00	1936	00:00:00	16380	0
hoxa4hs	4274	hoxa4pt	4271	00:00:00	1280	00:00:00	1280	00:00:00	1940	00:00:00	16704	0
hoxa4hs	4274	hoxa4ph	4275	00:00:01	1284	00:00:00	1280	00:00:00	1940	00:00:00	16068	0
hoxa4ph	4275	hoxa4cf	3300	00:01:06	18435	00:01:00	16847	00:02:26	18713	00:04:04	30052	3682
hoxa4ph	4275	hoxa4mm	4017	00:01:01	13947	00:01:01	15349	00:02:29	14584	00:04:50	26688	3199
hoxa4ph	4275	hoxa4pa	4249	00:00:00	1284	00:00:00	1276	00:00:00	1936	00:00:00	16100	0
hoxa4ph	4275	hoxa4pt	4271	00:00:00	1284	00:00:00	1276	00:00:00	1940	00:00:00	16116	0
hoxa4ph	4275	hoxa4hs	4274	00:00:00	1280	00:00:00	1280	00:00:00	1936	00:00:00	16072	0
csf2hs	4375	csf2mm	4630	00:00:50	10777	00:01:12	10775	00:03:13	11472	00:05:50	24191	3661
il3pt	4548	il3hs	4550	00:00:00	1284	00:00:00	1276	00:00:00	1980	00:00:00	17608	0
il3hs	4550	il3pt	4548	00:00:00	1288	00:00:00	1276	00:00:00	1976	00:00:00	17612	0
csf2mm	4630	csf2hs	4375	00:01:02	11229	00:01:09	11305	00:03:11	12004	00:05:52	24717	3844
lrrc4bbb	5055	lrrc4mf	5426	00:02:35	28479	00:01:22	28208	00:03:43	28743	00:07:06	44925	3616
lrrc4bbb	5055	lrrc4mm	5622	00:02:15	23464	00:01:37	23456	00:04:16	24269	00:08:02	40151	3868
lrrc4bbb	5055	lrrc4pa	5861	00:02:41	27688	00:01:26	27857	00:03:57	28512	00:07:28	45227	3490
lrrc4bbb	5055	lrrc4hs	5879	00:02:52	27949	00:01:25	27680	00:03:54	28775	00:07:32	45293	3465
ctgfmm	5175	ctgfhs	5204	00:01:30	16600	00:01:31	16593	00:04:22	17395	00:08:12	32115	4520
ctgfhs	5204	ctgmm	5175	00:01:31	16599	00:01:28	16589	00:04:14	17387	00:07:54	32116	4406
gpr137mm	5411	gpr137hs	5634	00:01:55	15808	00:02:03	15800	00:05:08	16639	00:09:34	32828	4758
lrrc4mf	5426	lrrc4bbb	5055	00:02:46	26601	00:01:32	29531	00:03:54	30064	00:07:43	45984	4020
lrrc4mf	5426	lrrc4mm	5622	00:02:59	30328	00:01:46	30059	00:04:41	31156	00:08:37	48192	4132
lrrc4mf	5426	lrrc4pa	5861	00:00:01	1292	00:00:00	1288	00:00:00	2144	00:00:00	20844	0
lrrc4mf	5426	lrrc4hs	5879	00:00:00	1288	00:00:00	1288	00:00:00	2180	00:00:00	20952	0
lrrc4mm	5622	lrrc4bbb	5055	00:02:27	25044	00:02:00	25040	00:04:40	25855	00:09:09	41739	4679
lrrc4mm	5622	lrrc4mf	5426	00:03:20	31645	00:02:08	31640	00:04:48	32219	00:08:45	48985	4410
lrrc4mm	5622	lrrc4pa	5861	00:03:35	32439	00:01:48	32435	00:05:08	33308	00:09:40	50648	4388
lrrc4mm	5622	lrrc4hs	5879	00:03:15	32703	00:01:57	32429	00:05:00	33576	00:09:19	50960	4388
gpr137hs	5634	gpr137mm	5411	00:01:54	17392	00:02:05	17383	00:05:59	18227	00:10:01	33887	4992
lrrc4pa	5861	lrrc4bbb	5055	00:02:30	29800	00:01:31	30411	00:04:11	30627	00:07:55	47603	4413
lrrc4pa	5861	lrrc4mf	5426	00:00:16	3665	00:00:08	3663	00:00:21	4519	00:00:39	23216	323
lrrc4pa	5861	lrrc4mm	5622	00:03:19	34028	00:01:49	34021	00:04:54	34896	00:09:12	52239	4544
lrrc4pa	5861	lrrc4hs	5879	00:00:00	1296	00:00:00	1292	00:00:00	2184	00:00:00	21988	0
lrrc4hs	5879	lrrc4bbb	5055	00:02:35	30328	00:01:42	30409	00:04:17	31155	00:08:04	47673	4329
lrrc4hs	5879	lrrc4mf	5426	00:00:12	3403	00:00:08	3573	00:00:19	4253	00:00:37	23060	304
lrrc4hs	5879	lrrc4mm	5622	00:03:14	33533	00:01:47	33667	00:04:51	35160	00:09:24	52547	4560
lrrc4hs	5879	lrrc4pa	5861	00:00:00	1296	00:00:00	1292	00:00:00	2184	00:00:01	21988	0
dnajc4hs	6001	dnajc4mm	6362	00:01:25	13700	00:02:15	13693	00:05:59	14631	00:11:06	32016	4990
dnajc4mm	6362	dnajc4hs	6001	00:01:25	14228	00:02:25	14221	00:06:04	15160	00:11:41	32548	5425
kcnk4ggg	7352	kcnk4pt	8702	00:00:00	1308	00:00:00	1300	00:00:00	2500	00:00:00	29508	0
kcnk4ggg	7352	kcnk4pa	8715	00:00:01	1304	00:00:00	1300	00:00:00	2504	00:00:00	28980	0
kcnk4ggg	7352	kcnk4hs1	8730	00:00:00	1308	00:00:00	1300	00:00:00	2500	00:00:00	29680	0
cdx2mm	8351	cdx2hs	9041	00:04:10	23225	00:05:04	23215	00:14:24	24585	00:25:51	48948	7709
kcnk4pt	8702	kcnk4ggg	7352	00:00:30	5001	00:00:36	5261	00:01:40	6203	00:03:18	33471	1202
kcnk4pt	8702	kcnk4pa	8715	00:03:41	16361	00:00:34	25305	00:01:35	17735	00:03:01	46744	940
kcnk4pt	8702	kcnk4hs1	8730	00:00:01	1316	00:00:01	1312	00:00:00	2688	00:00:00	32684	0
kcnk4pa	8715	kcnk4ggg	7352	00:00:43	6852	00:00:41	6844	00:01:53	8055	00:03:33	34524	1328
kcnk4pa	8715	kcnk4pt	8702	00:03:40	16364	00:00:34	16444	00:01:34	17733	00:03:03	46743	940
kcnk4pa	8715	kcnk4hs1	8730	00:02:50	12668	00:00:25	12661	00:01:10	14040	00:02:18	43197	695
kcnk4hs1	8730	kcnk4ggg	7352	00:00:31	5004	00:00:36	5263	00:01:41	6203	00:03:19	33377	1193
kcnk4hs1	8730	kcnk4pt	8702	00:00:00	1320	00:00:00	1316	00:00:00	2692	00:00:00	32688	0
kcnk4hs1	8730	kcnk4pa	8715	00:02:48	12668	00:00:26	12663	00:01:10	14037	00:02:18	43197	695
cdx2hs	9041	cdx2mm	8351	00:03:45	25604	00:04:52	24811	00:14:07	26967	00:25:49	51325	8363
kcnk4hs	10712	kcnk4mm	10827	00:05:07	25879	00:07:05	25869	00:22:00	27559	00:39:16	57515	9945
kcnk4mm	10827	kcnk4hs	10712	00:05:19	26141	00:07:36	26136	00:23:42	27821	00:44:35	57783	10157
ctsdpt	11284	ctsdhs	13238	00:07:30	21928	00:01:04	21921	00:02:16	23836	00:04:55	64921	856
bet1mm	12024	bet1hs	14691	00:08:01	27737	00:10:55	27732	00:35:24	29816	01:05:25	66811	11367
ctsdhs	13238	ctsdpt	11284	00:47:56	176639	00:11:29	176897	00:29:43	178548	00:59:34	219899	12593
bet1hs	14691	bet1mm	12024	00:08:04	34087	00:12:23	34079	00:35:05	36161	01:05:16	71041	14043
ddx18pt	19676	ddx18hs	19699	00:54:47	97491	00:04:21	98015	00:12:32	100692	00:27:42	167056	2935
ddx18hs	19699	ddx18pt	19676	00:56:29	97756	00:04:21	98191	00:12:53	100960	00:27:32	167319	2933
nr2e1mm	23619	nr2e1hs	24800	00:30:29	66112	00:40:42	66103	02:07:46	70044	04:13:52	138408	22966
nr2e1hs	24800	nr2e1mm	23619	00:32:27	70339	00:41:53	70332	02:52:08	97065	04:41:53	142640	24122

Tabela 4.17: *Sequências reais complementares com similaridade acima de 50%, $k = 600$.*

α		β		KDP1		KDP2		KDP4		KDP5		
Nome	**Tam.**	**Nome**	**Tam.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Tem.**	**Mem.**	**Ocr**
or51a7hs	2940	or51a7mm	2940	00:00:21	7599	00:00:23	7591	00:00:51	7503	00:01:31	16125	825
or51a7mm	2940	or51a7hs	2940	00:00:20	7332	00:00:26	7063	00:01:03	7943	00:01:48	16127	974
dppa5mm	3099	dppa5hs	3167	00:00:33	9180	00:00:52	9180	00:02:08	9635	00:03:46	18251	1874
dppa5hs	3167	dppa5mm	3099	00:00:39	9709	00:01:01	9792	00:02:16	10168	00:04:01	18777	1975
hoxa4cf	3300	hoxa4mm	4017	00:00:24	7339	00:00:23	7071	00:00:59	7343	00:01:49	18803	670
hoxa4cf	3300	hoxa4pa	4249	00:00:00	1276	00:00:00	1268	00:00:00	1820	00:00:00	13108	0
hoxa4cf	3300	hoxa4pt	4271	00:00:00	1276	00:00:00	1272	00:00:00	1816	00:00:00	13188	0
hoxa4cf	3300	hoxa4hs	4274	00:00:00	1276	00:00:00	1272	00:00:00	1820	00:00:01	13140	0
hoxa4cf	3300	hoxa4ph	4275	00:00:00	1276	00:00:00	1272	00:00:00	1820	00:00:00	13152	0
hoxa4mm	4017	hoxa4cf	3300	00:00:59	18432	00:01:17	19483	00:03:19	19487	00:05:57	30155	2844
hoxa4mm	4017	hoxa4pa	4249	00:00:56	14476	00:00:52	14997	00:02:22	15368	00:04:26	27692	1499
hoxa4mm	4017	hoxa4pt	4271	00:01:01	14740	00:00:57	14733	00:02:20	15105	00:04:23	27192	1478
hoxa4mm	4017	hoxa4hs	4274	00:00:56	15001	00:00:49	14731	00:02:23	15633	00:04:21	27984	1506
hoxa4mm	4017	hoxa4ph	4275	00:00:56	15003	00:00:52	14471	00:02:20	15899	00:04:17	27740	1478
hoxa4pa	4249	hoxa4cf	3300	00:01:18	23979	00:01:24	23797	00:03:28	24260	00:06:22	35548	3083
hoxa4pa	4249	hoxa4mm	4017	00:00:59	16059	00:00:54	16317	00:02:20	16953	00:04:16	28745	1587
hoxa4pa	4249	hoxa4pt	4271	00:00:00	1280	00:00:00	1280	00:00:00	1936	00:00:00	16432	0
hoxa4pa	4249	hoxa4hs	4274	00:00:00	1284	00:00:00	1280	00:00:00	1936	00:00:00	16384	0
hoxa4pa	4249	hoxa4ph	4275	00:00:00	1280	00:00:00	1280	00:00:00	1932	00:00:00	16100	0
hoxa6hs	4254	hoxa6mm	4261	00:01:25	15271	00:01:36	15528	00:04:09	16187	00:07:14	28205	2425
hoxa6mm	4261	hoxa6hs	4254	00:01:28	16063	00:01:31	16844	00:03:53	17505	00:06:47	29260	2319
hoxa4pt	4271	hoxa4cf	3300	00:01:18	23977	00:01:23	23887	00:03:27	24260	00:06:23	35891	3094
hoxa4pt	4271	hoxa4mm	4017	00:01:02	16851	00:00:56	16583	00:02:26	17487	00:04:27	29061	1665
hoxa4pt	4271	hoxa4pa	4249	00:00:00	1284	00:00:00	1280	00:00:00	1936	00:00:00	16432	0
hoxa4pt	4271	hoxa4hs	4274	00:00:01	1280	00:00:00	1280	00:00:00	1940	00:00:01	16704	0
hoxa4pt	4271	hoxa4ph	4275	00:00:00	1284	00:00:00	1280	00:00:00	1940	00:00:00	16120	0
hoxa4hs	4274	hoxa4cf	3300	00:01:19	23981	00:01:23	23973	00:03:30	24519	00:06:22	35840	3100
hoxa4hs	4274	hoxa4mm	4017	00:01:26	16848	00:01:16	17023	00:02:30	17223	00:04:45	29044	1691
hoxa4hs	4274	hoxa4pa	4249	00:00:00	1280	00:00:00	1280	00:00:00	1936	00:00:00	16384	0
hoxa4hs	4274	hoxa4pt	4271	00:00:00	1284	00:00:00	1276	00:00:00	1940	00:00:00	16700	0
hoxa4hs	4274	hoxa4ph	4275	00:00:00	1284	00:00:00	1276	00:00:00	1940	00:00:00	16068	0
hoxa4ph	4275	hoxa4cf	3300	00:01:18	23715	00:01:25	23973	00:03:32	24259	00:06:24	35857	3097
hoxa4ph	4275	hoxa4mm	4017	00:01:01	16588	00:00:56	16845	00:02:26	17224	00:04:33	29856	1660
hoxa4ph	4275	hoxa4pa	4249	00:00:00	1284	00:00:00	1280	00:00:00	1936	00:00:01	16100	0
hoxa4ph	4275	hoxa4pt	4271	00:00:00	1284	00:00:00	1280	00:00:00	1940	00:00:00	16120	0
hoxa4ph	4275	hoxa4hs	4274	00:00:01	1284	00:00:00	1280	00:00:00	1940	00:00:00	16068	0
csf2hs	4375	csf2mm	4630	00:01:46	15271	00:01:44	15263	00:05:24	15959	00:09:46	28680	2846
il3pt	4548	il3hs	4550	00:00:00	1288	00:00:00	1280	00:00:00	1980	00:00:00	17612	0
il3hs	4550	il3pt	4548	00:00:00	1288	00:00:00	1280	00:00:00	1980	00:00:00	17612	0
csf2mm	4630	csf2hs	4375	00:01:42	16593	00:02:11	16587	00:05:45	17284	00:10:00	30000	3077
lrrc4bbb	5055	lrrc4mf	5426	00:01:15	15015	00:00:39	15009	00:01:46	15807	00:03:26	31727	854
lrrc4bbb	5055	lrrc4mm	5622	00:02:53	29799	00:02:11	29793	00:06:06	30343	00:11:02	46485	2763
lrrc4bbb	5055	lrrc4pa	5861	00:01:13	13168	00:00:44	13339	00:01:41	13989	00:03:16	30443	746
lrrc4bbb	5055	lrrc4hs	5879	00:00:56	10000	00:00:29	9995	00:01:13	10821	00:02:19	27343	545
ctgfmm	5175	ctgfhs	5204	00:02:29	29533	00:02:29	29791	00:06:59	30585	00:12:40	45313	3615
ctgfhs	5204	ctgmm	5175	00:02:23	29271	00:02:16	29000	00:06:29	29797	00:11:56	44785	3380
gpr137mm	5411	gpr137hs	5634	00:02:40	29008	00:03:52	28469	00:08:25	29311	00:15:57	45240	3781
lrrc4mf	5426	lrrc4bbb	5055	00:02:22	22671	00:01:10	22664	00:03:06	23464	00:05:25	39381	1434
lrrc4mf	5426	lrrc4mm	5622	00:03:12	30063	00:01:54	30059	00:04:53	31160	00:08:56	47664	2124
lrrc4mf	5426	lrrc4pa	5861	00:00:00	1292	00:00:00	1284	00:00:00	2148	00:00:00	20844	0
lrrc4mf	5426	lrrc4hs	5879	00:00:00	1292	00:00:00	1284	00:00:00	2148	00:00:00	20952	0
lrrc4mm	5622	lrrc4bbb	5055	00:03:28	35344	00:02:40	35515	00:07:33	36151	00:13:15	52299	3538
lrrc4mm	5622	lrrc4mf	5426	00:03:47	36664	00:02:07	36657	00:05:55	37500	00:10:59	54267	2753
lrrc4mm	5622	lrrc4pa	5861	00:03:52	35871	00:02:19	36131	00:06:05	37004	00:11:30	54341	2624
lrrc4mm	5622	lrrc4hs	5879	00:03:34	35344	00:01:57	35336	00:05:41	36215	00:10:46	53600	2536
gpr137hs	5634	gpr137mm	5411	00:03:35	32176	00:03:15	32165	00:09:04	33012	00:15:44	36700	3997
lrrc4pa	5861	lrrc4bbb	5055	00:01:50	23464	00:01:19	23545	00:03:32	24291	00:06:35	40739	1777
lrrc4pa	5861	lrrc4mf	5426	00:00:02	1556	00:00:01	1548	00:00:03	2408	00:00:05	21100	20
lrrc4pa	5861	lrrc4mm	5622	00:03:37	36404	00:02:08	36400	00:05:21	37272	00:10:20	54616	2515
lrrc4pa	5861	lrrc4hs	5879	00:00:00	1296	00:00:00	1292	00:00:00	2184	00:00:00	21992	0
lrrc4hs	5879	lrrc4bbb	5055	00:01:36	20031	00:01:05	20115	00:03:05	20859	00:05:42	37377	1567
lrrc4hs	5879	lrrc4mf	5426	00:00:00	1296	00:00:00	1288	00:00:00	2148	00:00:01	20948	3
lrrc4hs	5879	lrrc4mm	5622	00:03:36	36931	00:01:55	36923	00:05:28	37800	00:10:18	55184	2518
lrrc4hs	5879	lrrc4pa	5861	00:00:00	1292	00:00:00	1296	00:00:01	2184	00:00:00	21984	0
dnajc4hs	6001	dnajc4mm	6362	00:02:16	20035	00:03:45	20031	00:10:08	20967	00:18:40	38350	4196
dnajc4mm	6362	dnajc4hs	6001	00:02:22	21884	00:04:02	21876	00:10:38	22815	00:19:51	40204	4709
kcnk4ggg	7352	kcnk4pt	8702	00:00:01	1308	00:00:00	1304	00:00:00	2496	00:00:00	29512	0
kcnk4ggg	7352	kcnk4pa	8715	00:00:00	1308	00:00:00	1304	00:00:00	2504	00:00:00	29980	0
kcnk4ggg	7352	kcnk4hs1	8730	00:00:00	1304	00:00:00	1300	00:00:00	2500	00:00:00	29684	0
cdx2mm	8351	cdx2hs	9041	00:06:10	37480	00:07:51	37471	00:23:33	38844	00:45:41	63203	6747
kcnk4pt	8702	kcnk4ggg	7352	00:00:39	5796	00:00:40	5789	00:02:31	6996	00:04:46	34000	901
kcnk4pt	8702	kcnk4pa	8715	00:00:01	1320	00:00:00	1316	00:00:00	2688	00:00:00	31704	0
kcnk4pt	8702	kcnk4hs1	8730	00:00:01	1320	00:00:00	1312	00:00:00	2688	00:00:00	1308	0
kcnk4pa	8715	kcnk4ggg	7352	00:00:51	7643	00:01:01	7637	00:03:05	8844	00:05:28	35315	1015
kcnk4pa	8715	kcnk4pt	8702	00:00:01	1316	00:00:00	1316	00:00:01	2692	00:00:01	31700	0
kcnk4pa	8715	kcnk4hs1	8730	00:00:01	1320	00:00:00	1312	00:00:00	2692	00:00:00	31848	0
kcnk4hs1	8730	kcnk4ggg	7352	00:00:43	5793	00:00:52	5792	00:02:31	6996	00:05:04	34171	892
kcnk4hs1	8730	kcnk4pt	8702	00:00:01	1316	00:00:00	1316	00:00:00	2692	00:00:00	32688	0
kcnk4hs1	8730	kcnk4pa	8715	00:00:00	1320	00:00:00	1312	00:00:00	2692	00:00:00	31852	0
cdx2hs	9041	cdx2mm	8351	00:06:21	44081	00:08:02	43811	00:24:21	45181	00:45:14	69543	7395
kcnk4hs	10712	kcnk4mm	10827	00:08:42	43512	00:13:21	42504	00:41:43	44192	01:16:27	74145	8979
kcnk4mm	10827	kcnk4hs	10712	00:10:00	43039	00:12:49	43029	00:41:08	44719	01:18:29	74680	9255
ctsdpt	11284	ctsdhs	13238	00:00:01	1340	00:00:00	1332	00:00:00	3248	00:00:00	44336	0
bet1mm	12024	bet1hs	14691	00:13:05	47275	00:21:37	47268	01:05:14	49352	02:01:08	86345	10700
ctsdhs	13238	ctsdpt	11284	00:47:34	184117	00:19:40	184291	00:50:58	186204	01:38:58	227292	11958
bet1hs	14691	bet1mm	12024	00:13:29	59431	00:22:01	59423	01:07:36	61507	02:05:55	98500	13412
ddx18pt	19676	ddx18hs	19699	00:00:01	1400	00:00:00	1392	00:00:01	4604	00:00:01	70964	0
ddx18hs	19699	ddx18pt	19676	00:00:02	1400	00:00:01	1388	00:00:00	4604	00:00:01	70964	0
nr2e1mm	23619	nr2e1hs	24800	00:56:20	118116	01:13:16	117845	04:17:29	121787	08:10:09	190151	21853
nr2e1hs	24800	nr2e1mm	23619	00:58:05	127363	01:14:21	127353	04:00:15	131565	08:04:47	199927	23020

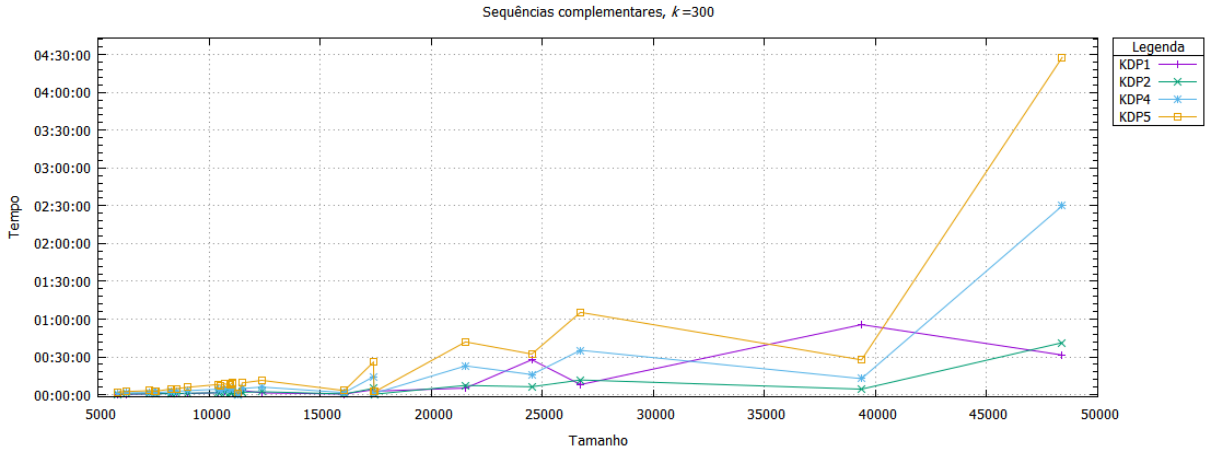


Figura 4.20: *Gráfico de tempo de execução dos testes reais complementares,* $k = 300$.

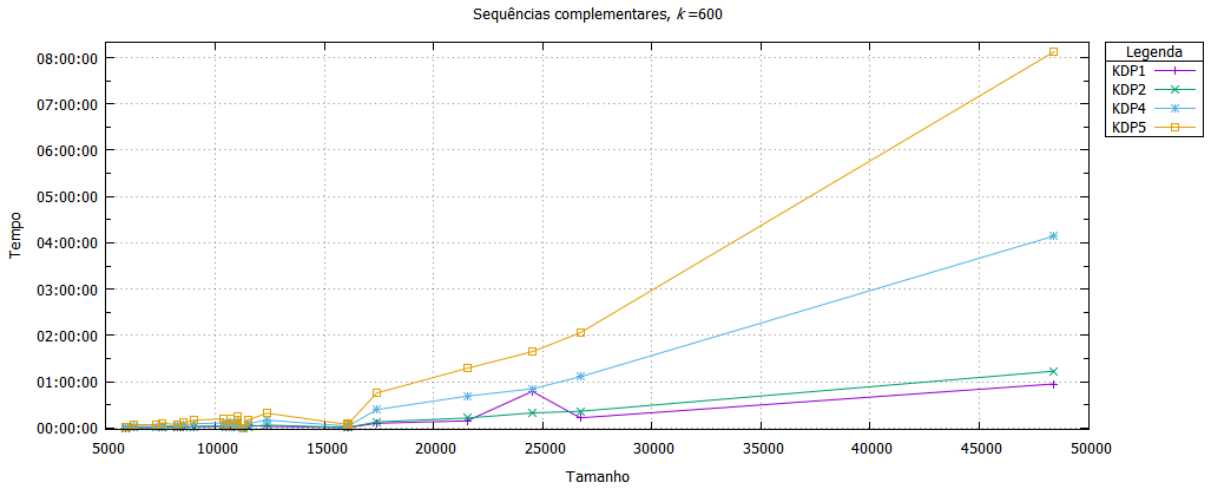


Figura 4.21: *Gráfico de tempo de execução dos testes reais complementares,* $k = 600$.

média, nos casos de similaridade entre 50% e 55%. Nos casos de similaridade entre 56,4% e 62,5% os programas são praticamente iguais em tempo. Nos demais casos, o programa KDP2 é cerca de 485% mais rápido que o programa KDP1. Nos casos de similaridade entre 82,8% e 98,2%, o programa KDP1 foi o mais lento de todos, com tempos absurdamente altos em relação ao KDP2. Observamos que os programas KDP4 é cerca de 190% mais lento que o KDP2 na média, e o programa KDP5 é cerca de 87% mais lento que o programa KDP4.

Quando $k = 300$ o programa KDP1 é cerca de 29%, na média, mais rápido que o programa KDP2, nos casos de similaridade entre 50% e 62,5% e alguns poucos casos até 72%. Nos demais casos, o programa KDP2 é cerca de 162%, na média, mais rápido que o programa KDP1. O programa KDP4 é cerca de 174% mais lento que o KDP2 na média, e o programa KDP5 é cerca de 86% mais lento que o programa KDP4.

Quando $k = 600$ o programa KDP1 é cerca de 30%, na média, mais rápido que o programa KDP2, nos casos de similaridade entre 50% e 67,4%, com alguns casos em que o KDP2 é mais rápido. Nos demais casos, o programa KDP2 é cerca de 45%, na média, mais rápido que o programa KDP1, observando que, devido à quantidade de diferenças, não foi possível obter nenhuma ocorrência em casos acima de 89,7% de similaridade entre as sequências. No geral, os programas KDP4 e KDP5 são, novamente, os mais lentos e os que mais consomem memória. O programa KDP4 é cerca de 190% mais lento que o KDP2 na média, e o programa KDP5 é cerca de 86% mais lento que o programa KDP4.

Observamos que a quantidade de memória utilizada pelos programas KDP1 e KDP2 é ainda praticamente idêntico em todos os casos, porém a diferença para os demais programas diminuiu consideravelmente. Mais detalhadamente, quando $k = 30$, o programa KDP4 utiliza, na média, cerca de 6% mais memória que os programas KDP2 e KDP1 e o programa KDP5 utiliza, na média cerca de 140% mais memória que o programa KDP4. Quando $k = 300$, o programa KDP4 utiliza, na média, cerca de 5% mais memória que os programas KDP2 e KDP1 e o programa KDP5 utiliza, na média, cerca de 103% mais memória que o programa KDP4. Quando $k = 600$, o programa KDP4 utiliza, na média, cerca de 33% mais memória que os programas KDP2 e KDP1 e o programa KDP5 utiliza, na média, cerca de 101% mais memória que o programa KDP4. Esses resultados nos permitem comprovar que a eficiência da árvore de sufixo e do vetor de sufixo está diretamente relacionado com a similaridade entre as sequências: quanto mais similares as sequências, menor será o consumo de memória.

O tamanho médio da maior extensão comum entre duas sequências, extraído dos pares de sequências dos testes dessa seção, é ligeiramente maior que o dos testes anteriores. Aqui o tamanho máximo, na média, é de 112 caracteres e a média do tamanho da maior extensão comum, levando em consideração valores iguais a zero, é de 0.35 e de 1.38, levando em consideração apenas valores maiores que zero. Observamos que a diferença de tempo do programa KDP2, que calcula de forma direta o LCE, diminuiu para os programas KDP4 e KDP5 e acreditamos que isso se deve ao aumento do tamanho do LCE.

A similaridade entre as sequências utilizadas nos casos de testes reais complementares aumentou, na média, consideravelmente em relação aos testes anteriores, sendo de 77% de similaridade. Isso nos permitiu observar um comportamento inesperado do KDP1, que era o melhor em todos os testes anteriores, e um comportamento mais regular do programa KDP2. Sobre o KDP1, acreditamos que o tamanho das ocorrências, com um aumento considerável em relação aos testes anteriores, influenciou

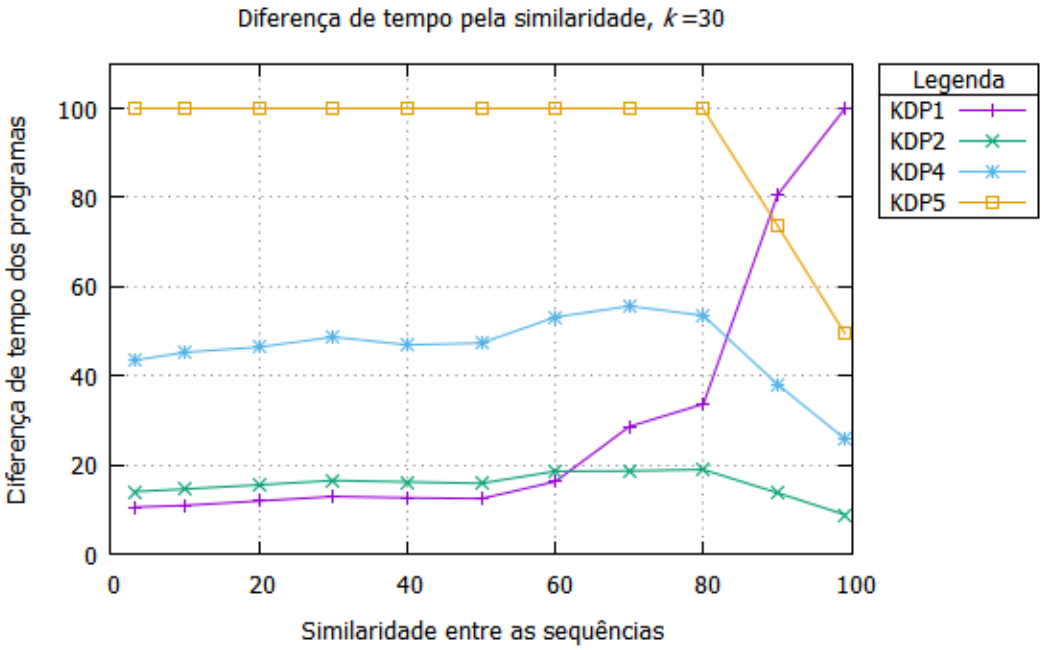


Figura 4.22: *Gráfico de diferença de tempo dos programas por similaridade entre as sequências,* $k = 30$.

diretamente o tempo de execução, por conta do preenchimento da matriz D. Mais detalhadamente, quando $k = 30$ o preenchimento termina entre as linhas 64 e 3387, observando que o tamanho médio das ocorrências nesse caso é de 346 caracteres. Quando $k = 300$, esse preenchimento termina entre as linhas 600 e 3148, observando que o tamanho médio das ocorrências nesse caso é de 970 caracteres. Quando $k = 600$, esse preenchimento termina entre as linhas 1397 e 4086, observando que o tamanho médio das ocorrências nesse caso é de 1887 caracteres.

Com os dados obtidos dos diferentes casos de testes realizados foi possível identificar que o programa KDP1 é mais rápido que o programa KDP2 até uma certa similaridade entre as sequências envolvidas no processamento. Na tentativa de identificar e entender melhor até que ponto um programa é melhor que o outro e obter uma melhor conclusão, elaboramos os gráficos das Figuras 4.22, 4.23 e 4.24, representando todos os casos de testes reais realizados, quando $k = 30$, $k = 300$ e $k = 600$ respectivamente, agrupados por similaridade entre as sequências e a diferença de tempo entre os programas. Para obter tais gráficos, juntamos os 180 casos de testes com sequências reais, ordenados pela similaridade entre as sequências, e calculamos a diferença de tempo, em porcentagem, entre cada programa executado, onde o maior tempo representa 100%. Após essa etapa, agrupamos os casos de testes em intervalos de 10% de similaridade, ou seja, de 0 a 10, de 10 a 20, e assim por diante, até o caso de maior similaridade obtida.

Com isso, podemos observar na Figura 4.22 que, quando $k = 30$, o programa KDP1 é mais rápido que os demais programas entre 0 e 62% de similaridade entre as sequências e o mais lento de todos quando a similaridade é acima de 90%. Na Figura 4.23, quando $k = 300$, o programa KDP1 é mais rápido que os demais programas entre 0 e 68% e o mais lento de todos quando a similaridade é acima de 95%. Na Figura 4.24, quando $k = 600$, o programa KDP1 é mais rápido que os demais programas entre 0 e 72%. Perceba que o programa KDP2, no geral, é o programa mais regular em termos de tempo de execução e uso de memória, mantendo um desempenho próximo ao melhor no intervalo de similaridade entre 0 e 72% e sendo o melhor quando a similaridade é maior que 72%.

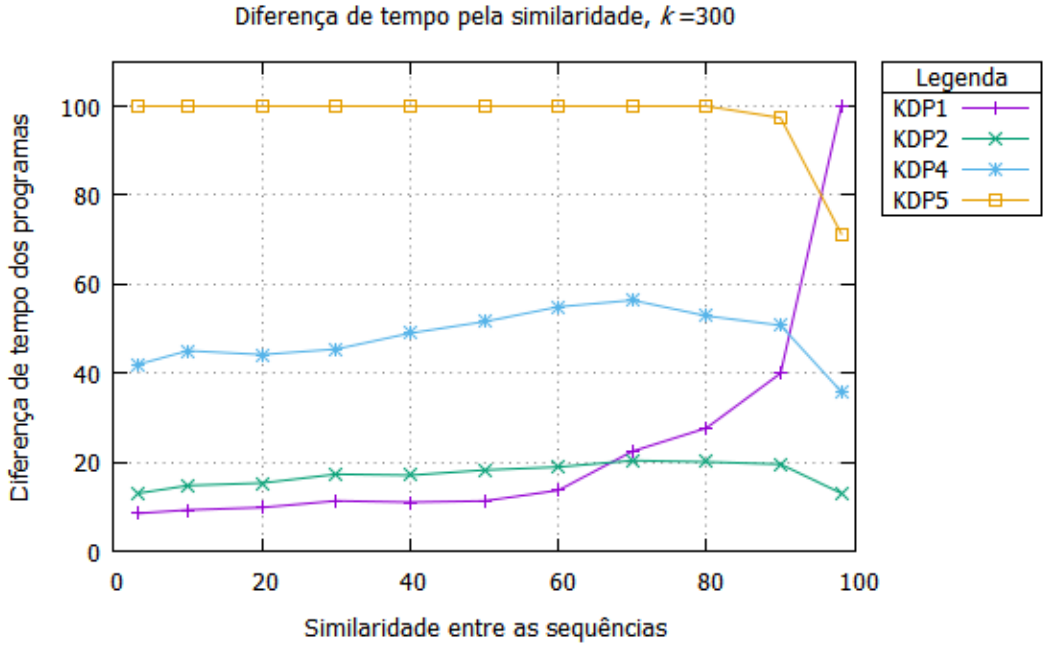


Figura 4.23: *Gráfico de diferença de tempo dos programas por similaridade entre as sequências,* $k = 300$.

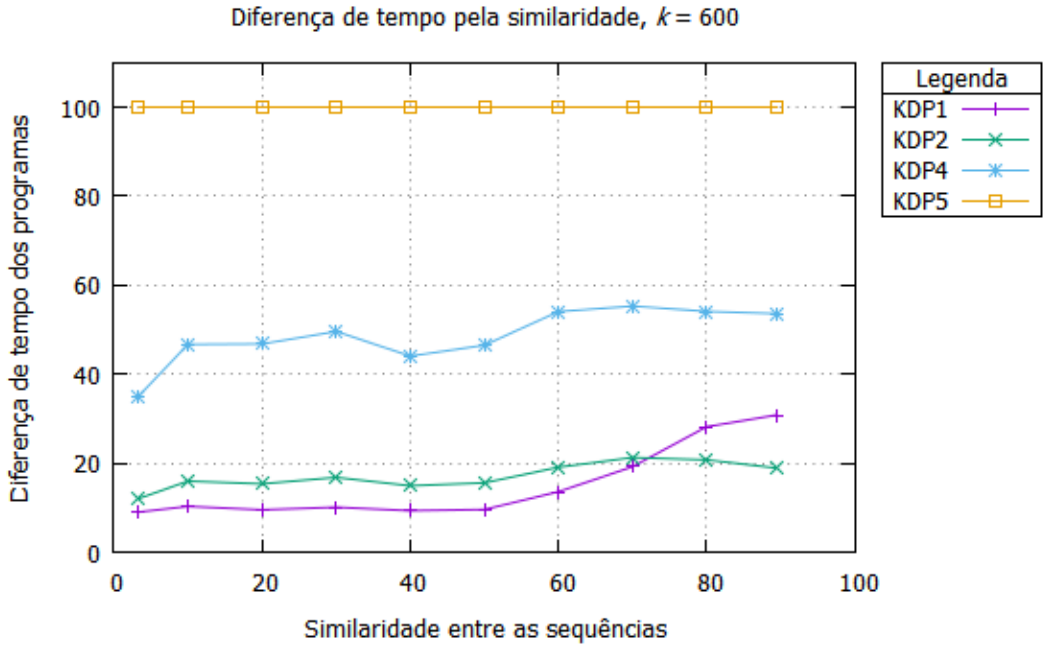


Figura 4.24: *Gráfico de diferença de tempo dos programas por similaridade entre as sequências,* $k = 600$.

4.4.1 Teste prático

Após a realização dos testes com sequências reais, executamos uma nova série de testes com duas sequências de bactérias (arquivos `mycobacterium_bovis` e `mycobacterium_tuberculosis`), representando o DNA completo de *Mycobacterium bovis* (GenBank: AP010918.1) e *Mycobacterium tuberculosis* (GenBank: AP018033.1), com tamanhos de 4.371.711 e 4.399.916 caracteres respectivamente, no intuito de obter o tempo de execução dos programas em casos práticos. Neste caso, as sequências envolvidas possuem 99,95% de similaridade [57].

O teste prático foi realizado com $k = 30$ e a sequência mycobacterium_bovis como α. O tempo de execução do programa KDP1 foi de 684 horas, 05 minutos e 14 segundos, do programa KDP2 foi de 359 horas, 11 minutos e 21 segundos, do programa KDP4 foi de 419 horas e 45 segundos e do programa KDP5 foi de 500 horas, 03 minutos e 02 segundos. Perceba que o KDP1 foi o mais lento de todos, confirmando a conclusão obtida nos testes com sequências reais. O KDP2 foi o programa mais rápido, seguido do programa KDP4 e KDP5. Os resultados obtidos neste teste, confirmam que o KDP2 é o programa mais regular em termos de tempo e o mais adequado para a condução de experimentos práticos.

Capítulo 5

Desenvolvimento do Sistema

A proposta deste trabalho inclui ainda o desenvolvimento de um sistema WEB cliente/servidor que permita aos interessados em encontrar uma região específica, o uso simples e efetivo do melhor algoritmo implementado. Neste capítulo será relatado em detalhes o desenvolvimento desse sistema, desde seu projeto até sua implantação.

5.1 Metodologia

Existem várias abordagens e estrategias diferentes para desenvolvimento de sistemas, que depende muito do projeto em si e da equipe disponível. Neste projeto optou-se pela metodologia ágil de desenvolvimento. A **metodologia ágil** pode ser definida como um conjunto de processos de desenvolvimento de *software* que visa simplificar as etapas, com entregas frequentes de *software* funcional e foco na iteração entre indivíduos [58]. O que resulta em uma documentação sucinta, além de etapas do projeto desenvolvidas concorrentemente pela mesma pessoa, que assume diferentes papéis. Nesta metodologia, os usuários possuem papel importantíssimo no projeto, e todas as etapas são baseadas em suas perspectivas e atividades executadas no sistema.

Separamos o desenvolvimento do sistema em três etapas principais: projeto de *software*, desenvolvimento e implantação. Todas essas etapas são dependentes uma da outra e cíclicas, ou seja, o projeto pode mudar conforme o desenvolvimento, o desenvolvimento pode mudar conforme a implantação, e assim por diante. A definição dos recursos necessários para o desenvolvimento do sistema é descrita na Seção 5.2. A etapa de projeto, em que é realizado o levantamento de requisitos e a modelagem de dados, é descrita na Seção 5.3. A etapa de desenvolvimento é descrita na Seção 5.4 e a etapa de implantação, com detalhes de como e onde usar o *software*, é descrita na Seção 5.5.

5.2 Recursos

Nesta seção, vamos definir os recursos necessários para o desenvolvimento do *software*, a infraestrutura mínima necessária para suportar o sistema e as ferramentas utilizadas ao longo do projeto.

Primeiramente, é necessário definir a plataforma computacional em que o *software* será instalado, ou seja, o conjunto de componentes, bibliotecas, *softwares* e *hardwares*, que provêm recursos e facilitam o desenvolvimento de um novo sistema. Nesse sentido, a Web tem se tornado a plataforma padrão de desenvolvimento, sendo independente de sistema operacional e acessível por qualquer

dispositivo móvel. Decidimos então pelo desenvolvimento do sistema como aplicativo para ser acessível pela Web, com o nome de **Web Specific Segment Selection** ou apenas **w3s**, uma alusão ao seu objetivo e a plataforma escolhida.

Outro fator importante a definir antes do desenvolvimento do sistema é a arquitetura de *software*, que consiste dos componentes de *software*, suas propriedades externas, e seus relacionamentos com outros sistemas. Apresentamos na Figura 5.1 a arquitetura detalhada de interação entre os sistemas, onde podemos ver a etapa de processamento, que usa diretamente o programa KDP1 ou KDP2 e gera um arquivo com os resultados. O banco de dados serve como suporte ao sistema Web, e será utilizado para guardar os dados do processamento. A arquitetura proposta é muito flexível, facilitando sua manutenção e possibilitando a substituição de qualquer componente. Caso surja uma implementação nova e mais eficiente do algoritmo para resolver o PPkD, será possível substituir apenas a camada de processamento, sem necessidade de recodificação.

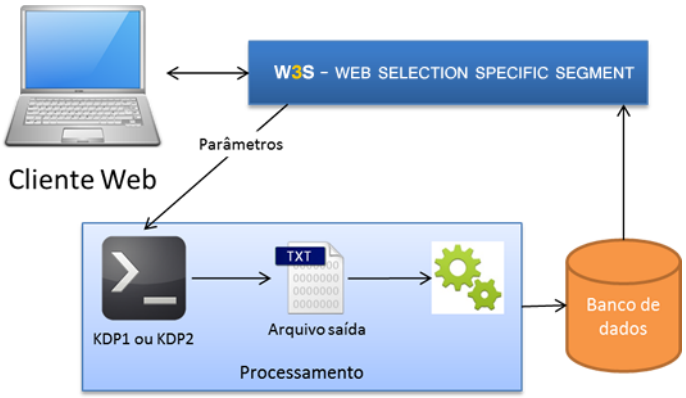


Figura 5.1: *A arquitetura detalhada do sistema.*

Com base nas definições anteriores, selecionamos então os *softwares* que utilizaremos neste projeto, são eles:

- A ferramenta CASE[1] Visual Paradgma, versão 10, para a diagramação de modelos e processos de negócio;
- A linguagem de programação JAVA, versão 8, com o *framework* PLAY, versão 1.4.4 para desenvolver o aplicativo WEB, obtido em https://www.playframework.com/download;
- O PostgreSQL, versão 9.4, como sistema gerenciador de banco de dados, obtido em https://www.postgresql.org/download/;
- O bootstrap, versão 3.3.7, como framework visual, obtido em http://getbootstrap.com/;
- O repositório público GIT como controlador de versão de código no endereço https://github.com/jeandobre/web-select-primer para salvar os arquivos fontes do sistema;
- O IntelliJ IDEA, versão 2016.3.4, como IDE[2], obtido em https://www.jetbrains.com/idea/download/;

[1]Ferramenta CASE (do inglês *Computer-Aided Software Engineering*) é uma classificação para softwares que auxiliam atividades de engenharia de software, desde análise de requisitos e modelagem até programação e testes.

[2]IDE (do inglês *Integrated Development Environment*) é um software que reúne ferramentas de apoio à codificação de softwares, visando aumentar a produtividade no desenvolvimento de novos sistemas computacionais.

- O *Bizage Modeler*, versão 3.1.0 para modelar o diagrama de atividades do sistema, obtido em http://www.bizagi.com/pt/produtos/bpm-suite/modeler.

5.3 Projeto

Um **projeto de software** é uma descrição da estrutura de *software* a ser implementada, dos dados que são parte do sistema, das *interfaces* entre os componentes do sistema e, algumas vezes, dos algoritmos utilizados. Na fase de projeto, executamos a atividade de especificação de *software*, que destina-se a estabelecer quais funções são requisitadas para o sistema e as restrições suas operações. O resultado dessa fase são diagramas, listas e modelos que traduzem e descrevem, nos mínimos detalhes, o *software* que deverá ser desenvolvido na fase seguinte.

A tarefa de seleção de segmentos específicos geralmente é guiado pela experiência e conhecimento do biólogo sobre aquela determinada sequência de DNA, de forma que ele pode conhecer uma região específica a ser amplificada na PCR. Porém, na maioria dos casos, é uma tarefa extremamente difícil, que vai além da experiência do biólogo, principalmente quando há mais de duas sequências envolvidas. O sistema w3s vai auxiliá-los nessa tarefa, pois vai entregar uma lista de ocorrências de segmento específico, computadas de forma exata.

Nesse ponto, cada ocorrência obtida através da execução do KDP1 ou KDP2 é um segmento específico. Cabe ao usuário verificar se este segmento está apto a ser utilizado. Perceba que o w3s não substitui ferramentas computacionais já utilizadas por biólogos. Pelo contrário, vai precisar do suporte delas, pois as ocorrências de segmento específico deverão ser submetidas à uma ferramenta de desenho de *primers*. O sistema w3s é uma ferramenta computacional especializada e auxiliará apenas na tarefa de seleção de segmentos específicos. A Figura 5.2 apresenta o diagrama de atividades com as principais funcionalidades que usuários necessitam do sistema.

Definimos então o *software* como um sistema para auxiliar biólogos na atividade de selecionar segmentos específicos a partir de sequências de DNA. Por essa definição, percebemos que os usuários interessados neste sistema são biólogos que precisam identificar regiões específicas de forma exata para as mais diversas atuações da PCR.

5.3.1 Objetivos

O objetivo geral do sistema é identificar segmentos específicos de forma precisa. Vamos utilizar o diagrama de atividades para representar as principais atividades que o usuário executará no sistema afim de cumprir esse objetivo.

Um **diagrama de atividades** é essencialmente um gráfico de fluxo das atividades que devem ser executadas, mostrando a ordem e o fluxo entre uma e outra atividade, sendo empregado, principalmente, para modelar os aspectos dinâmicos do sistema. Geralmente utilizamos a notação UML[3] (do inglês, *Unified Modeling Language*) para desenhar tal diagrama.

Dentro do diagrama de atividades é possível ver diversos fluxos de trabalho. Chamamos de **fluxo principal** a sequência de atividades no processo, do inicio até o fim, necessárias para alcançar o objetivo principal do sistema. Os demais fluxos que existirem dentro do diagrama são chamados de **fluxos alternativos**.

[3]Na área de Engenharia de *Software*, UML é uma linguagem de modelagem que permite representar um sistema de forma padronizada (com intuito de facilitar a compreensão na fase de projeto do sistema.)

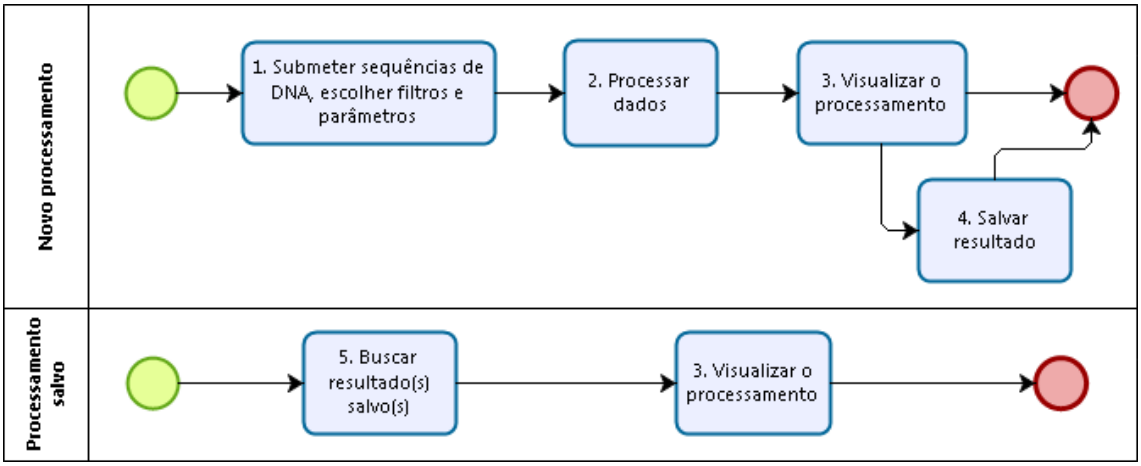


Figura 5.2: *O diagrama de atividades mostrando os fluxos de atividades que devem ser suportados pelo sistema.*

O diagrama de atividades do sistema w3s pode ser visto na Figura 5.2, que apresenta as principais atividades que os usuários necessitam do sistema. O diagrama apresenta dois fluxos: novo processamento e processamento salvo. O primeiro é o fluxo principal utilizado na primeira execução do sistema para obtenção dos resultados. O segundo é um fluxo alternativo, e só ocorre quando o usuário salvar os resultados oriundos de uma primeira execução do sistema, para uma busca futura por resultados salvos.

Cada atividade do usuário deve ser suportada pelo sistema. Assim, o próximo passo no desenvolvimento do sistema é o levantamento de requisitos para cada atividade, que será detalhado na próxima seção.

5.3.2 Requisitos

Um **requisito de software**, ou apenas **requisito**, é a descrição das funções, objetivos, propriedades e restrições que o sistema deve possuir para satisfazer contratos, padrões ou especificações de acordo com o(s) usuário(s). De forma mais geral, um requisito é uma condição necessária para satisfazer os objetivos do *software*. Os requisitos podem ser classificados em **requisitos funcionais**, que detalham o que o sistema deve fazer, ou seja, suas funções, e **requisitos não-funcionais**, que detalham os critérios que qualificam os requisitos funcionais, como qualidade ou usabilidade.

Na etapa de levantamento de requisitos é essencial a participação do(s) usuário(s) do sistema, a fim de descobrir maiores informações sobre o domínio da aplicação, que serviços o sistema deve fornecer, o desempenho exigido do sistema, as restrições de hardware e demais informações. Os requisitos foram levantados com ajuda de um profissional biólogo que trabalha diretamente na área de atuação do sistema. A lista dos principais requisitos do sistema é apresentado a seguir. A lista detalhada dos requisitos funcionais e não funcionais, classificadas por versão do software, pode ser vista no Apêndice A.

1. Submeter sequências de DNA, escolher filtros e parâmetros

    1.1. Na tela inicial do sistema, o usuário carrega arquivos de DNA, ou digita/copia sequências de texto de DNA, no formado FASTA ou texto contínuo, escolhe os parâmetros e filtros, seleciona um dos programas KDP1 ou KDP2, e submete os dados para processamento;

i. O usuário pode adicionar mais de uma sequência para comparação com a sequência alvo, ou seja, o sistema deve permitir múltiplas sequências β;

ii. O usuário pode marcar a opção para ver a maior/menor ocorrência;

iii. O usuário pode filtrar o tipo de processamento, escolhendo processar todas as posições da sequência alvo ou apenas um determinado intervalo;

iv. O usuário pode filtrar o resultado mostrando apenas as ocorrências com um certo tamanho de caracteres;

v. O usuário pode filtrar o resultado mostrando apenas as ocorrências com uma certa distancia entre elas.

1.2. O sistema recebe os dados e arquivos de DNA, e carrega os arquivos para o servidor de dados.

2. Processar os dados;

2.1. O sistema deve fazer a validação dos dados submetidos;

2.2. O sistema deve converter arquivos FASTA para arquivos de texto;

2.3. O sistema deve processar os dados usando um dos programas selecionados pelo usuário;

2.4. O sistema deve guardar os resultados obtidos em banco de dados.

3. Visualizar o processamento;

3.1. O usuário visualiza os dados do processamento e todas as ocorrências em formato de barras/sequência;

3.2. O usuário pode visualizar os dados do processamento e todas as ocorrências em forma de lista de dados;

3.3. O usuário pode copiar uma ocorrência.

4. Salvar resultado;

4.1. Se decidir por salvar o resultado do processamento efetuado, o usuário informa os dados de identificação do processamento e salva o resultado.

5. Buscar resultado(s) salvo(s).

5.1. Se decidir por buscar o processamento salvo, o usuário informa os dados de identificação do processamento;

5.2. O sistema lista os processamentos salvos com base nos dados informados pelo usuário;

5.3. O usuário seleciona um processamento para visualizar seus resultados e o sistema executa o requisito 3.

Dado a sua importância no contexto do sistema, o requisito 3.1 merece uma explicação mais detalhada. Visando facilitar a visualização dos resultados oriundos do processamento e sabendo que pode haver muitas ocorrências para uma sequência alvo, estabelecemos como requisito adicional o desenvolvimento de um componente que apresenta a sequência alvo em formato de barra e permite visualizar cada candidato a segmento específico ao selecionar a posição inicial daquela ocorrência. De forma mais detalhada, o componente inclui os seguintes requisitos:

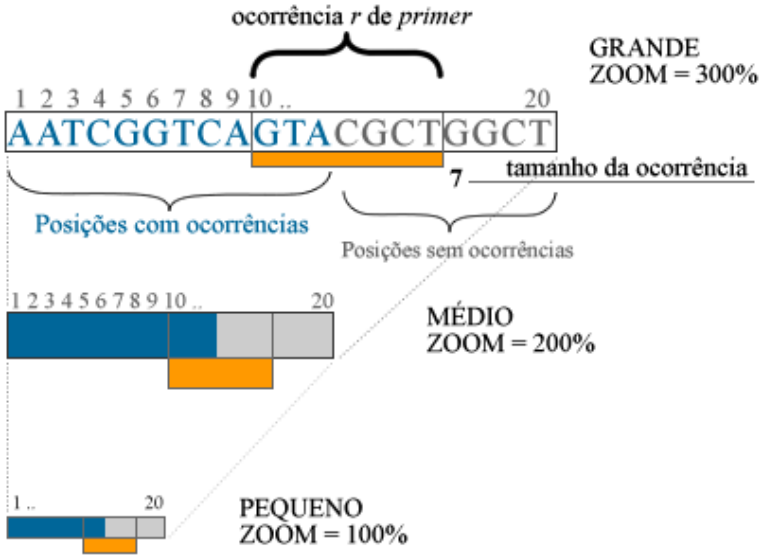


Figura 5.3: *O projeto do componente visual de apresentação da sequência alvo.*

- Apresentar a sequência alvo em forma de barra, com opção de rolagem horizontal, quando a sequência extrapolar o tamanho da largura da tela;
- Cada posição da sequência alvo que tiver uma ocorrência deve ser marcada com uma cor diferente da sequência;
- Ao clicar na posição marcada de cor diferente, o componente deve mostrar o tamanho da ocorrência para esta posição bem como o segmento;
- Permitir a visualização do resultado em três tamanhos, pequeno, médio e grande, onde cada letra da sequência, no tamanho grande, corresponde a 10 *pixels*[4] de largura na tela, em tamanho médio, corresponde a 2 *pixels* e em tamanho pequeno, a 1 pixel.

Um esquema visual, para facilitar o entendimento e compreensão das funcionalidades do componente, é apresentado na Figura 5.3.

5.3.3 Modelagem de dados

A partir dos requisitos gerais, é possível elaborar outro diagrama fundamental para o projeto do sistema, o modelo de dados, que é usado, frequentemente, para descrever a estrutura da informação que está sendo processada. Neste caso, vamos utilizar o **diagrama de entidades e relacionamentos** (DER) que é um modelo conceitual utilizado na Engenharia de *Software* para descrever os objetos (entidades) envolvidos em um domínio de negócios, com suas características (atributos) e como eles se relacionam entre si (relacionamentos).

Como o sistema tem o requisito de salvar o resultado, para posterior recuperação, é fundamental o uso de banco de dados. A Figura 5.4 apresenta o modelo de entidade e relacionamento que contempla todos os requisitos levantados para este sistema. Cada entidade é representada em cor

[4]Um pixel é o menor elemento num dispositivo de exibição.

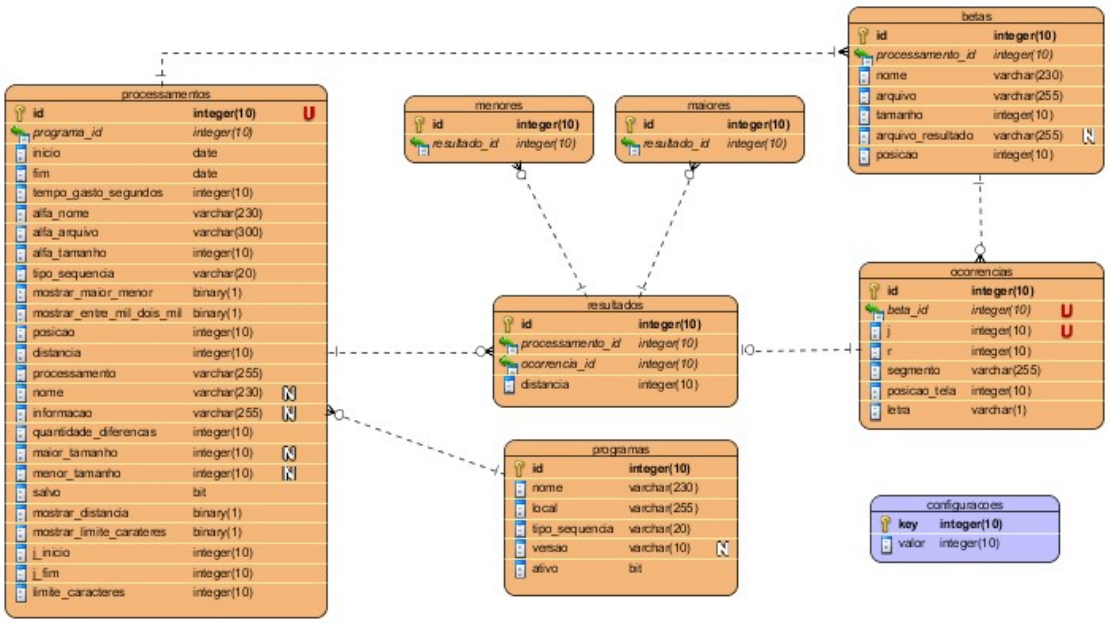


Figura 5.4: *Diagrama de entidade e relacionamento do sistema.*

laranja por um nome único, relacionada a uma outra entidade representada por uma linha pontilhada. A entidade *configuracoes* serve apenas para armazenar dados de configuração do sistema e não se relaciona a nenhuma outra entidade no sistema.

5.4 Desenvolvimento

Na fase de desenvolvimento, também conhecida como fase de codificação, é onde ocorre a transformação dos requisitos, levantados na fase de projeto, em código de máquina. Esta fase é direcionada pelos requisitos do sistema e validada pelos casos de testes, que são elaborados em conjunto com a codificação. Um **caso de teste** é um conjunto de condições ou situações usadas para testar as funcionalidades de um *software*, visando descobrir defeitos e garantir seu funcionamento correto.

Como o sistema w3s é basicamente uma *interface* para uso dos programas KDP1 e KDP2, desenvolvidos em C++, as funcionalidades que serão aqui detalhadas, tem como base o resultado gerado pela execução de um desses programas. Assim, o sistema desenvolvido em JAVA executa os algoritmos desenvolvidos em C++, passando os devidos parâmetros para essa execução e, ao final, abre o arquivo de resultados gerado. Perceba que a integração entre o sistema e o programa acontece aqui, onde a comunicação entre eles ocorre primeiramente, por parâmetros usados na execução dos algoritmos em *prompt* de comando e, posteriormente, ao termino da execução, pelo arquivo de resultados, escrito no padrão: posição j de α; tamanho do segmento; segmento. Para exemplificar, suponha uma execução com os seguintes parâmetros: `-a CCCGGCCC -b CCCGTGCCC -k 1 -sf teste.txt`. O resultado escrito no arquivo `teste.txt` será:

$$j; tamanho; sequencia$$
$$0; 5; CCCGG$$
$$1; 4; CCGG$$
$$2; 3; CGG$$
$$3; 2; GG$$

A partir desse arquivo, fazemos o carregamento das ocorrências para uma lista, chamada aqui de lst. Perceba que lst terá tamanho máximo $m - k$, dado que m é tamanho da sequência alvo e k a quantidade de diferenças. No exemplo dado anteriormente, lst tem 4 elementos, onde a posição 0 do arquivo corresponde à posição 1 da sequência α no sistema. A primeira linha do arquivo é o cabeçalho e será ignorada pelo sistema.

A seguir apresentamos detalhadamente os requisitos solicitados pelo usuário e as soluções encontradas parar explorar os resultados decorrentes da solução do PPkD. Para isso, redefinimos a sequência α e a sequência β, dados como entrada do problema, como sequência alvo e sequência de comparação, respectivamente.

No requisito 1.1.i, o sistema deve permitir que o usuário carregue mais de uma sequência de comparação, ou seja, que ele compare a sequência alvo com mais de uma sequência de comparação. Mais detalhadamente, seja B um conjunto com 2 sequências. Neste caso, é necessário comparar a sequência alvo com B_1 e com B_2.

Para suportar a funcionalidade mencionada, executamos o algoritmo que resolve o PPkD para cada sequência do conjunto B e guardamos a lista de resultados. Após o processamento, é necessário selecionar, para cada posição j da sequência α, a maior ocorrência nessa posição, em cada lista de resultado, desde que essa posição tenha ocorrência em todos os resultados. Se não houver uma ocorrência para a posição j da sequência alvo em todos os resultados, então afirmamos que não há uma ocorrência que inicia na posição j e que tenha k diferenças com relação a todas as sequências do conjunto B.

Para exemplificar, suponha o processamento envolvendo uma determinada sequência alvo e um conjunto B de três sequências. Chamamos de r_1 a lista de ocorrências resultante da comparação da sequência alvo com B_1, r_2 a lista resultante da comparação da sequência alvo com B_2 e r_3 a lista resultante da comparação da sequência alvo com B_3. Usando a ideia apresentada acima, preenchemos uma lista lst com os valores oriundos dos resultados de cada uma das execuções, representados na Tabela 5.1. Em negrito estão os valores escolhidos para compor a lista lst. O valor -1 representa uma não ocorrência. Perceba que na posição 1, o maior valor, 18, está na posição 1 de $r3$, na posição 2, o maior valor, 12, está na posição 2 de $r2$, na posição 3, o maior valor, 19, está na posição 3 de $r1$ e a partir da posição 4 não há mais ocorrências.

A demonstração da correção do algoritmo é simples. Ainda utilizando a Tabela 5.1 perceba que, escolhendo como resultado final o valor 15, que corresponde ao tamanho do menor segmento começando na posição 1 da sequência alvo com k diferenças em relação à sequência B_1, ele não corresponde a uma solução do problema ao considerarmos a sequência B_3. Esse segmento não possui pelo menos k diferenças com relação a B_3, pois o menor segmento iniciando na posição 1 com k diferença em relação a B_3 tem tamanho 18. Com isso, selecionar o tamanho da maior

Tabela 5.1: *Exemplo de ocorrências de múltiplos resultados, onde cada coluna representa uma posição j de α e cada linha um resultado da comparação de α com uma sequência do conjunto B.*

	1	2	3	4	5	6	7	8
$r1$	15	11	**19**	19	18	19	**-1**	**-1**
$r2$	16	**12**	15	**-1**	**-1**	**-1**	-1	-1
$r3$	**18**	11	17	16	-1	-1	-1	-1
lst	18	12	19	-1	-1	-1	-1	-1

ocorrência para a posição j da sequência alvo garante que todos os outros segmentos tenham pelo menos k diferenças com relação a qualquer sequência do conjunto B.

Um algoritmo para resolver essa variação do PPkD, utilizando uma das abordagens já discutidas nos capítulos anteriores, tomaria tempo $O(l(abd)+o)$, onde l representa a quantidade de sequências do conjunto B, abd o tempo da abordagem aplicada e o o maior número de ocorrências. Utilizando a abordagem 1, a complexidade é $O(l(m^2n) + o)$ e utilizando a abordagem 2, a complexidade é $O(l(km^2n) + o)$, onde m é o tamanho da sequência alvo, n é o tamanho de cada sequência em B e k é a quantidade de diferenças.

Para o requisito 1.1.ii, o sistema deve selecionar a maior e a menor ocorrência dentre todas as ocorrências, apresentando ao usuário todas as posições em que elas ocorreram. Para suportar essa funcionalidade, basta criar duas variáveis, *maior* e *menor*, e setar os valores como $maior \leftarrow -1$ e $menor \leftarrow m$, onde m representa o tamanho da sequência alvo. Tais valores indicam os limites mínimo e máximo que podem ser encontrados na lista, visto que nunca encontraremos um valor menor que -1 e maior que m. É necessário manter duas listas, *pg* e *pp*, para guardar as posições em que a maior e a menor ocorrência aparecem, respectivamente. O algoritmo deve então percorrer lst comparando o valor de cada posição i com as variáveis *maior* e *menor*, atualizando *maior* com o maior valor em lst e *menor* com o menor valor em lst. Obtendo o maior e o menor valor da lista, é necessário percorrer mais uma vez lst. Caso $lst_i = maior$, guardamos a posição i em *pg*. Caso $lst_i = menor$, guardamos a posição i em *pp*. Tal algoritmo toma tempo $O(o)$, onde o representa o tamanho da lista lst.

No requisito 1.1.iii, o sistema deve permitir que o usuário escolha um intervalo de posições da sequência alvo para o processamento, ou seja, o usuário pode selecionar um intervalo $[x..y]$ da sequência alvo que deseja comparar com as sequências do conjunto B.

Tal funcionalidade foi codificada diretamente nos programas desenvolvidos em C++, onde adicionamos um novo parâmetro, $-j$, precedido de dois valores, posição de inicio x e posição de fim y. Seja $m = |\alpha|$, com x e y representando um intervalo da sequência α, temos que $x < y$, $x \geq 1$ e $y \leq m$. Assim, modificamos apenas o laço **para** do Algoritmo 5, na linha 2 que inicia na posição 1 e vai até $m - k$, para iniciar na posição x e ir até a posição y. No caso do usuário não passar o parâmetro $-j$, percorre-se toda a sequência alvo por padrão.

No requisito 1.1.iv, o usuário deseja filtrar a lista de resultados, selecionando um valor que representa o tamanho máximo da ocorrência que ele está procurando. O sistema deve então selecionar na lista de resultados lst apenas as ocorrências que tenham o tamanho menor ou igual ao valor selecionado pelo usuário. Para implementar essa funcionalidade, percorremos lst removendo todas as ocorrências que são maiores que o valor escolhido pelo usuário. Mais detalhadamente, seja lst a lista de resultados oriundos de um dado processamento, com m representando o tamanho da sequência α e max um valor inteiro setado pelo usuário, tal que $0 < max \leq m$. Para cada i em

lst, perguntamos se $lst_i > max$, e caso a resposta seja positiva, removemos a ocorrência na posição i. Assim, ao fim, lst conterá apenas as ocorrências menores ou iguais ao valor setado em max. Claramente tal algoritmo toma tempo tempo $O(o)$ onde o representa o tamanho da lista lst.

Já no requisito 1.1.v, o sistema deve permitir ao usuário adicionar um valor que representa a distância entre uma e outra ocorrência, ou seja, o usuário necessita de dois segmentos específicos, com uma certa distância (por exemplo 200 caracteres) entre eles. O sistema deve então marcar todas as ocorrências que distam um certo valor z do fim de uma para o inicio da outra, e apresentar ao usuário as duas, quando uma delas for selecionada. Seja m o tamanho da sequência alvo e z um inteiro tal que $0 < z \leq m$. Para encontrar tais ocorrências, basta percorrer a lista lst e, para cada posição i, verificar se existe uma ocorrência na posição $i + r + z$, onde r representa o tamanho da ocorrência na posição i. Se existir, marcamos as posições i e $i + z$. Ao final, as posições sem marcação são removidas enquanto que as posições marcadas são mostradas ao usuário. Um algoritmo que suporta a funcionalidade aqui descrita toma tempo $O(o + d)$ onde o representa o tamanho da lista lst e d a quantidade de elementos a serem removidos de lst após as marcações.

Perceba que todas as funcionalidades aqui detalhadas podem ser combinadas de forma que, caso o usuário desejar, as cinco podem ser utilizadas ao mesmo tempo.

5.5 Implantação

A fase de implantação consiste, basicamente, na elaboração e execução de testes para a detecção de defeitos e validação. Os testes de validação se destinam a demonstrar que um sistema cumpre com suas especificações enquanto que os testes de de detecção de defeitos tem como finalidade expor defeitos latentes em um *software* antes de sua entrega. Também ocorre a integração do código e implantação para uso em ambiente de teste durante os diversos testes do sistema, e em ambiente de produção, quando o sistema está apto para uso.

Separamos as funcionalidades do sistema em versões usuais de *software*, ou seja, a cada versão, o sistema deve ser passível de uso, mesmo com um conjunto pequeno de requisitos implementados. Assim, dividimos a lista detalhada de requisitos, que pode ser vista em Apêndice A em cinco versões do sistema. Ao final da quarta versão, foi realizado um teste de aceitação juntamente a todos os envolvidos do projeto do sistema. Como o sistema estava com quase todas as funcionalidades implementadas, testadas e operantes, foi possível que o usuário final do sistema executasse um procedimento completo de seleção de segmentos específicos. Isso foi de fundamental importância, visto que o usuário detectou algumas funcionalidades necessárias que o sistema não disponibilizava e pôde validar outras funcionalidades, mostrando pontos de acerto e pontos a serem melhorados.

Ao final, o código fonte do sistema foi carregado para o repositório GIT nos endereços https://github.com/jeandobre/web-select-primer e https://github.com/jeandobre/k-difference-prime, onde encontram-se disponíveis para implantação e uso. Vamos a seguir explicar sucintamente como fazer para implantar o *software* em ambiente de uso final e ressaltar a importância de um profissional experiente em integração e *middleware* para realizar tal tarefa.

Para rodar o sistema, é necessário ter um ambiente de servidor de dados já configurado, preferencialmente com linux, JAVA 8, o banco de dados PostgreSQL 9.4, o servidor de aplicação PLAY 1.4.4, o g++, para compilar os programas desenvolvidos em C++ e o GIT. É necessário copiar o código fonte para pastas locais do servidor, compilar o código fonte dos programas

`kdifferenceprime1.cpp` e `kdifferenceprime2.cpp` e executar o arquivo `banco.sql` diretamente no servidor de banco de dados, arquivo esse que cria o banco de dados. A configuração do servidor PLAY para acesso ao banco de dados criado é realizada no arquivo `application.conf`, que está dentro da pasta `conf` do projeto. A configuração do local de execução dos programas em C++, da pasta de carregamento de arquivos e da pasta em que serão salvos os arquivos de saída, é realizada na entidade configurações diretamente no banco de dados. Esses parâmetros podem ser visualizados através do sistema acessando o link *configuracao*.

Capítulo 6

Conclusões e Perspectivas Futuras

Neste trabalho abordamos o problema biológico de seleção de segmentos específicos através de algumas soluções existentes para o problema computacional denominado Problema do *Primer* com k Diferenças e depois realizamos uma pesquisa na literatura buscando por algoritmos que resolvem o problema computacional. Basicamente, duas soluções foram explanadas, com a segunda possuindo três variações, diferentes na forma que determinam a maior extensão comum entre duas sequências. Propomos então implementar os algoritmos correspondentes, submetendo-os a testes com dados artificiais e reais, e selecionar o melhor deles para o desenvolvimento de uma ferramenta computacional afim de auxilar os biólogos na busca por segmentos específicos para as mais diversas atividades em laboratórios.

Os testes serviram para avaliar, além das abordagens, as diferenças entre as estruturas de dados que servem para a determinação da maior extensão comum entre duas sequências. Avaliamos o emprego da árvore de sufixo comprimida para computação da maior extensão comum e, ao realizarmos a primeira bateria de testes, percebemos que o seu uso é impraticável, dado o alto tempo consumido na execução dos testes, embora o consumo de memória seja o menor das estruturas de dados testadas.

A árvore de sufixo e o vetor de sufixo são, na prática, lentos para a determinação da maior extensão comum entre duas sequências. O uso dessas estruturas no Algoritmo 7, justifica-se pelo fato de melhorarem a complexidade teórica de tempo através da computação em tempo constante da maior extensão comum. Na prática, porém, o tamanho da maior extensão comum entre duas sequências é relativamente pequeno e, por conta disso, o algoritmo que se utiliza da computação direta passa a ser mais rápido que aqueles que se utilizam dessas estruturas de dados.

Uma surpresa encontrada durante o trabalho foi o programa que emprega o Algoritmo 6 ser o mais rápido na grande maioria dos testes realizados, mesmo tendo complexidade de tempo $O(m^2n)$. Porém, ao fazer uma nova análise do algoritmo, percebemos que o tamanho das ocorrências em sequências com similaridade abaixo de 60% é, na média, cerca de duas a três vezes o valor de k. Isso faz com que a matriz D seja preenchida até uma certa linha e raramente atinja seu tamanho máximo. Isso faz com que o tempo, na prática, do algoritmo seja de $O(mkn)$ o que condiz com o tempo de execução dos testes realizados.

Outro ponto importante a ser destacado é o sistema computacional desenvolvido para permitir ao usuário a facilidade de uso do melhor algoritmo implementado. O sistema foi criado para auxiliar os usuários na busca por segmentos específicos, com uma arquitetura que permite a evolução do sistema e integração de novas abordagens de forma fácil. Acreditamos que possa se tornar uma

ferramenta para auxiliar no processo de seleção de segmentos específicos para as mais diversas atuações da PCR.

Contudo ainda há pontos a serem melhorados na ferramenta, como a opção de registrar a escolha de um determinado segmento como candidato para uso, facilitando ao usuário identificar quais segmentos já foram utilizados e se houve sucesso ou não. Além disso, o sistema carece de um processamento assíncrono de sequências, que atualmente interrompe o seu uso quando as sequências são grandes. O ideal é colocar em *background* o processamento da sequência, avisar o usuário do tempo estimado de processamento e liberar o sistema para uso. Outra melhoria importante é a substituição do componente de visualização das ocorrências na sequência por um componente mais completo em funcionalidades e que suporta grandes sequências, como o *NCBI sequence viewer*, disponível em https://www.ncbi.nlm.nih.gov/projects/sviewer/ e distribuído para uso de forma gratuita.

Percebemos ao longo do trabalho, que existem várias soluções para o Problema do Casamento Inexato com até k Diferenças que empregam novas abordagens e são, teoricamente, mais rápidos aos que foram aqui empregados. Achamos que, com o devido estudo, é possível como trabalhos futuros criar um novo algoritmo para resolver o Problema do *Primer* com k Diferenças utilizando essas soluções.

Também como trabalhos futuros está o desenvolvimento de heurísticas para o Problema do *Primer* com k Diferenças, que permitam o processamento de sequências grandes em tempo hábil, devolvendo soluções biologicamente úteis.

Uma outra possibilidade de trabalho futuro é a implementação de algoritmos que resolvem o Problema do *Primer* com k Diferenças que utilizam outras medidas de comparação de sequências, como a Distância de *Hamming*. A implementação de tais algoritmos, e a adição deles à ferramenta computacional, pode permitir aos usuários a escolha da medida desejada para comparação das sequências, de acordo com a aplicação e das sequências envolvidas.

Os resultados obtidos neste trabalho mostram que cumprimos o objetivo de estudar detalhadamente o Problema do *Primer* com k Diferenças, principalmente no que diz respeito à sua aplicabilidade prática no problema biológico da seleção de segmentos específicos e temos em mente trabalhos futuros que podem estender este estudo, tanto em seu contexto teórico quanto prático.

Apêndice A

Lista de requisitos completos

- Versão 0.1

    1. Na tela inicial, o usuário poderá adicionar um arquivo de texto como parâmetro de sequência alfa
    2. Na tela inicial, o usuário poderá adicionar um ou mais arquivos de texto como parâmetro de sequência beta. O sistema não deve permitir que seja adicionado o mesmo arquivo mais de uma vez.
    3. Na tela inicial, quando o usuário adicionar mais de um arquivo, eles deverão ser organizados em forma de fila, ou seja, o mais recente vai para o final da lista.
    4. Na tela inicial, o usuário poderá adicionar a quantidade de diferenças em um campo de texto.
    5. Na tela inicial, o usuário poderá selecionar o tipo de sequência da seguinte forma: sequências diferentes; sequencia.diferente sequências parecidas e; sequencia.parecida Com base na escolha do usuário, o sistema deverá utilizar o algoritmo KDP1 para sequências diferentes e o algoritmo KDP2 para sequências parecidas.
    6. Na tela inicial, o usuário poderá escolher entre processar todas as posições da sequência alvo ou selecionar uma posição específica de início e fim.
    7. Na tela inicial, o usuário poderá submeter os parâmetros clicando no botão Processar. O sistema deverá submeter os dados para efetuar o processamento e mostrar em uma tela escura a mensagem: “Aguarde, processando...”
    8. Fazer o carregamento do arquivo que o usuário selecionou em sequência alvo e sequências para um repositório temporário no servidor previamente configurado. *diretorio.entrada*
    9. Executar o programa de processamento, selecionado na funcionalidade 5, via console e guardar os resultados em um repositório previamente configurado. *diretorio.saida*
    10. Na tela do resultado, o sistema deverá apresentar os parâmetros escolhidos pelo usuário e carregar a lista de resultados salvo no repositório previamente configurado. *diretorio.saida*
    11. Na tela de resultados, o usuário poderá fazer um novo processamento clicando no botão Novo Processamento. O sistema deverá retornar a tela inicial com todos os campos vazios.

12. Na tela inicial, o usuário poderá remover um arquivo de sequência já carregado clicando no botão lixeira. O sistema não deverá fazer o carregamento dos arquivos removidos pelo usuário.

- Versão 0.2

1. O sistema deverá apresentar qualquer erro durante o processamento em uma tela, informando detalhadamente o erro que ocorreu, com opção de o usuário iniciar novamente o processamento. /erro.html
2. Os campos sequência alvo, sequências e quantidade de diferenças são obrigatório.
3. O campo de quantidade de diferenças não pode permitir ser digitado letras e símbolos, apenas valores numéricos.
4. O campo de posição específica da sequência alvo não pode permitir ser digitado letras e símbolos, apenas valores numéricos.
5. O campo de distância da posição não pode permitir ser digitado letras e símbolos, apenas valores numéricos.
6. A quantidade de diferenças deve ser maior que zero e menor que o tamanho da sequência alvo.
7. A posição e a distância deverão estar dentro do tamanho da sequência alvo, ou seja, a posição deve ser maior que zero e menor que o tamanho da sequência alvo.
8. Se o usuário escolher uma página inexistente, erro 404 o sistema deverá apresentar uma tela de erro com a opção de o usuário iniciar novamente o processamento. /404.html
9. Na tela de resultados, o usuário poderá visualizar as ocorrências de candidatos a segmento de forma gráfica. O componente mostra em azul as ocorrências na sequência. O tamanho é representado na cor laranja. Quando a sequência for maior que o tamanho da tela, deverá ser apresentada uma barra de rolagem.
10. Na tela de resultados, o usuário poderá selecionar qualquer posição da sequência para visualizar a ocorrência.
11. Na tela de resultados, quando o usuário selecionar uma posição que tenha ocorrência, o sistema deverá apresentar a posição, o tamanho e a sequência.
12. Na tela inicial, o usuário poderá marcar a opção de visualizar a menor e a maior ocorrência na tela de resultados.
13. Na tela de resultados, se a funcionalidade 13 for marcada, o sistema deverá apresentar em cor vermelha a maior ocorrência (Posição e tamanho) e uma marcação no componente de sequência relativo a posição.
14. Na tela de resultados, se a funcionalidade 13 for marcada, o sistema deverá apresentar em cor verde a menor ocorrência (Posição e tamanho) e uma marcação no componente de sequência relativo a posição.
15. Na tela de resultados, o usuário poderá guardar o processamento em banco de dados. O sistema deverá solicitar, em uma janela *pop-up*, o nome do usuário (campo obrigatório) e uma informação adicional com opções de *Guardar o resultado* ou *Cancelar*. O botão

Guardar o resultado só será ativado, quando houver mais de 5 caracteres digitados no campo nome do usuário.

16. Na tela de resultados, quando o usuário salvar um resultado, o sistema deverá guardar todos os dados apresentados no resultado em banco de dados.

- Versão 0.3

1. O sistema deverá identificar cada usuário por meio do *id* da sessão e salvar cada resultado processado utilizando o *id* da sessão.
2. O sistema deverá processar mais de uma sequência com a sequência alvo e apresentar o resultado mesclado, com as seguintes regras: Cada posição da sequência alvo poderá ter uma ocorrência em cada arquivo de resultado. Será então selecionado o maior tamanho de cada posição; Se a posição da sequência alvo não tiver uma ocorrência, então ela deve ser descartada; A quantidade total de ocorrências será igual a menor quantidade de ocorrências encontradas em uma das sequências.
3. Na tela de resultados, o sistema deverá apresentar a quantidade de ocorrências encontrado que contenham k diferenças com relação a(s) sequência(s) selecionadas pelo usuário.
4. Na tela de resultados, o sistema deverá apresentar as sequências selecionadas pelo usuário da seguinte forma: (nome da sequência, tamanho e quantidade de ocorrências).
5. Na tela de resultados, o sistema deverá permitir ao usuário clicar sobre o link de ocorrências e visualizar todas as ocorrências em forma de tabela. A tabela deverá conter as seguintes informações: quando apenas uma sequência beta: (Posição, distancia geral, sequência) quando mais de uma sequência beta: (Posição, distância, sequência, posição beta1, posição beta$N$..)
6. Na tela de resultados, quando houver mais de uma sequência beta, o sistema deverá permitir ao usuário clicar sobre o link de ocorrências por sequência beta e permitir visualizar todas as ocorrências em forma de tabela. A tabela deverá conter as seguintes informações: (Posição, distancia geral, sequência)
7. Na tela inicial, na tela de resultados e na tela de erro o usuário poderá recuperar um processamento guardado clicando no botão Recuperar processamento.
8. Na tela de recuperar processamento, o usuário poderá pesquisar por nome do usuário, nome de qualquer sequência envolvida e data. O sistema deverá apresentar uma lista em ordem decrescente de data e hora.
9. Na tela de recuperar processamento, o usuário poderá remover qualquer processamento da lista clicando no botão de lixeira.
10. Na tela de recuperar processamento, o usuário poderá selecionar qualquer processamento da lista clicando no botão de visualizar o processamento. O sistema deverá apresentar o resultado conforme foi salvo.

- Versão 0.4

1. Na tela de resultados, no componente visual da sequência, o usuário poderá aumentar o zoom da sequência clicando no botão +, diminuir o zoom da sequência clicando no botão -. O componente deverá ter 3 opções de zoom:
   - mínimo: 1 caractere representado por um pixel;
   - médio: 1 caractere representado por 2 pixels;
   - máximo: 1 caractere no tamanho 10 pts = 13 pixels..
2. Na tela de resultados, o sistema deverá apresentar no componente visual da sequência, todas as posições em que ocorreram o menor e o maior candidato. No rótulo verde e vermelho, deverá aparecer todas as posições em forma de números.
3. Na tela de resultados, no componente visual da sequência, quando o usuário mover o mouse ao longo da sequência, deverá ser apresentado o índice posicionado pelo mouse, tanto nas ocorrências quanto na sequência.
4. Na tela inicial, quando o usuário submeter dados para o processamento, o usuário poderá cancelar o processamento atual clicando no botão cancelar de cor vermelha.
5. Na tela de visualização de ocorrências por arquivo beta, o carregamento da lista de dados deverá ser por *ajax/json* sob demanda.
6. Na tela de visualização de resultados por arquivo processamento, o carregamento da lista de dados deverá ser por *ajax/json* sob demanda.
7. Na tela de visualização de processamentos, o carregamento da lista de dados deverá ser por *ajax/json* sob demanda.
8. A lista de ocorrência deve ser iniciada em 1.

- Versão 0.5

1. Na tela inicial o usuário poderá adicionar como entrada da sequência alvo ou das sequências que serão comparadas, um arquivo de texto de DNA ou um arquivo FASTA ou digitar a sequência diretamente em um campo de texto. O sistema deverá aceitar a entrada e fazer as devidas conversões de dados. O componente deverá submeter o arquivo imediatamente via AJAX e apresentar um resumo ao usuário. Nome do arquivo, tamanho da sequência e tipo (texto digitado, arquivo de texto ou arquivo FASTA)
2. No processamento de dados, quando o usuário escolher filtrar os resultados o sistema deverá remover do resultado as ocorrências que estão fora do filtro selecionado pelo usuário.

Apêndice B

Tabelas completas de testes reais

Tabela B.1: *Sequências reais de genes do* Homo sapiens, $k = 30$.

Alfa		Beta		KDP1		KDP2		KDP4		KDP5		
Nome	Tam.	Nome	Tam.	Tem.	Mem.	Tem.	Mem.	Tem.	Mem.	Tem.	Mem.	Ocr
hsascl2	4455	hsarhgdig	4398	00:00:05	4168	00:00:06	4116	00:00:18	4868	00:00:34	17264	4379
hsascl2	4455	hsankrd43	5457	00:00:05	4200	00:00:08	4276	00:00:22	4952	00:00:46	18664	4382
hsascl2	4455	hsa1bg	8694	00:00:09	4176	00:00:13	4240	00:00:39	5148	00:01:10	23424	4380
hsascl2	4455	hsalg10b	14972	00:00:13	4180	00:00:20	4212	00:00:59	5684	00:02:02	32328	4379
hsascl2	4455	hsbad	16877	00:00:18	3729	00:00:25	3728	00:01:09	5431	00:02:14	36271	4380
hsascl2	4455	hsankr10	38530	00:00:38	4556	00:00:49	4560	00:02:44	8072	00:05:59	66620	4380
hsascl2	4455	hsascc2	51655	00:00:52	4520	00:01:15	4624	00:03:48	9220	00:08:23	87980	4376
hsascl2	4455	hsasz1	66302	00:01:09	4116	00:01:29	4115	00:04:42	10161	00:12:09	107427	4381
hsascl2	4455	hsaff4	90284	00:01:38	4960	00:01:58	4908	00:06:57	12860	00:16:27	148404	4377
hsankrd43	5457	hsarhgdig	4398	00:00:05	3960	00:00:07	3996	00:00:22	4789	00:00:39	18351	5391
hsankrd43	5457	hsascl2	4455	00:00:05	3939	00:00:08	3976	00:00:22	4829	00:00:41	18417	5391
hsankrd43	5457	hsa1bg	8694	00:00:11	4772	00:00:17	4744	00:00:44	5912	00:01:30	25460	5386
hsankrd43	5457	hsalg10b	14972	00:00:17	4776	00:00:27	4760	00:01:17	6456	00:02:35	34544	5383
hsankrd43	5457	hsbad	16877	00:00:21	4047	00:00:28	4079	00:01:21	5773	00:02:42	37829	5385
hsankrd43	5457	hsankr10	38530	00:00:47	5060	00:01:06	4980	00:03:21	8692	00:07:08	68668	5378
hsankrd43	5457	hsascc2	51655	00:01:11	4383	00:01:29	4351	00:04:31	8947	00:10:10	89093	5383
hsankrd43	5457	hsasz1	66302	00:01:30	4433	00:01:57	4367	00:05:49	10472	00:13:24	109211	5378
hsankrd43	5457	hsaff4	90284	00:01:49	5252	00:02:27	5208	00:08:11	13256	00:22:16	150080	5379
hsa1bg	8694	hsarhgdig	4398	00:00:10	4803	00:00:15	4845	00:00:37	5829	00:01:13	24131	8629
hsa1bg	8694	hsascl2	4455	00:00:10	4811	00:00:14	4777	00:00:39	5871	00:01:10	24033	8628
hsa1bg	8694	hsankrd43	5457	00:00:12	4784	00:00:17	4824	00:00:46	5947	00:01:27	25521	8619
hsa1bg	8694	hsalg10b	14972	00:00:36	5104	00:00:43	5155	00:02:08	6977	00:04:10	39535	8619
hsa1bg	8694	hsbad	16877	00:00:44	5145	00:00:50	5147	00:02:23	7128	00:04:31	44047	8620
hsa1bg	8694	hsankr10	38530	00:01:38	5151	00:01:55	5153	00:05:42	9055	00:11:39	73684	8611
hsa1bg	8694	hsascc2	51655	00:02:15	5203	00:02:32	5160	00:07:42	10084	00:16:40	95473	8617
hsa1bg	8694	hsasz1	66302	00:02:53	5244	00:03:14	5276	00:09:55	11615	00:24:23	115299	8616
hsa1bg	8694	hsaff4	90284	00:04:03	5551	00:04:32	5521	00:13:50	13901	00:33:02	155391	8614
hsalg10b	14972	hsarhgdig	4398	00:00:15	6796	00:00:21	6753	00:01:04	8108	00:02:00	34945	14891
hsalg10b	14972	hsascl2	4455	00:00:15	6751	00:00:21	6799	00:01:06	8292	00:02:04	34885	14897
hsalg10b	14972	hsankrd43	5457	00:00:21	6784	00:00:29	6777	00:01:22	8447	00:02:34	36493	14898
hsalg10b	14972	hsa1bg	8694	00:00:30	8756	00:00:42	8756	00:02:09	10568	00:04:14	43192	14896
hsalg10b	14972	hsbad	16877	00:01:04	6752	00:01:21	6801	00:04:04	9281	00:08:08	54803	14897
hsalg10b	14972	hsankr10	38530	00:02:27	6837	00:03:10	6888	00:09:49	11223	00:20:54	84504	14899
hsalg10b	14972	hsascc2	51655	00:03:17	7123	00:04:16	7123	00:13:21	12603	00:30:47	105929	14901
hsalg10b	14972	hsasz1	66302	00:05:11	7253	00:05:57	10368	00:17:30	14092	00:41:51	126255	14899
hsalg10b	14972	hsaff4	90284	00:05:37	9444	00:07:08	9436	00:23:57	18392	01:00:06	168524	14899
hsbad	16877	hsarhgdig	4398	00:00:16	9580	00:00:27	9560	00:01:08	11236	00:02:17	42104	16811
hsbad	16877	hsascl2	4455	00:00:17	9284	00:00:25	9212	00:01:11	11056	00:02:15	41860	16809
hsbad	16877	hsankrd43	5457	00:00:22	9364	00:00:28	9292	00:01:26	11036	00:02:44	43128	16805
hsbad	16877	hsa1bg	8694	00:00:44	10312	00:00:52	10432	00:02:23	12424	00:05:03	49352	16804
hsbad	16877	hsalg10b	14972	00:01:04	10128	00:01:16	10060	00:03:52	12620	00:07:55	58168	16806
hsbad	16877	hsankr10	38530	00:03:03	10956	00:03:43	10959	00:10:50	15416	00:22:30	92960	16806
hsbad	16877	hsascc2	51655	00:04:25	11176	00:05:10	11228	00:13:59	17072	00:30:29	114864	16803
hsbad	16877	hsasz1	66302	00:05:19	10996	00:05:55	11024	00:19:04	18080	00:43:56	134448	16804
hsbad	16877	hsaff4	90284	00:08:40	11563	00:12:22	11492	00:26:33	20756	01:38:53	70880	16805
hsascc2	51655	hsarhgdig	4398	00:00:54	23800	00:01:18	23700	00:03:35	28460	00:07:53	107400	51586
hsascc2	51655	hsascl2	4455	00:00:54	17091	00:01:13	17117	00:03:34	21692	00:07:37	100517	51586
hsascc2	51655	hsankrd43	5457	00:01:06	23560	00:01:31	23488	00:04:31	28196	00:10:11	108336	51888
hsascc2	51655	hsa1bg	8694	00:01:51	25636	00:02:26	25640	00:07:12	30572	00:15:56	115632	51583
hsascc2	51655	hsalg10b	14972	00:03:07	24852	00:03:58	24848	00:12:45	30552	00:28:15	123736	51583
hsascc2	51655	hsbad	16877	00:04:06	17920	00:04:31	17872	00:13:36	23769	00:29:32	121783	51583
hsascc2	51655	hsankr10	38530	00:09:03	26272	00:10:30	26108	00:33:52	33920	01:18:07	159292	51583
hsascc2	51655	hsasz1	66302	00:16:24	18247	00:18:20	18223	00:55:30	28239	02:14:29	192673	51577
hsascc2	51655	hsaff4	90284	00:21:56	27360	00:24:48	27320	01:21:55	39812	04:07:45	242432	51582
hsasz1	66302	hsarhgdig	4398	00:00:58	28716	00:01:23	28660	00:04:40	34880	00:10:02	132208	66241
hsasz1	66302	hsascl2	4455	00:00:59	28892	00:01:31	28972	00:04:47	34820	00:10:20	132268	66240
hsasz1	66302	hsankrd43	5457	00:01:20	29720	00:01:50	29608	00:06:12	36004	00:12:46	134696	66226
hsasz1	66302	hsa1bg	8694	00:02:19	30760	00:03:00	30772	00:09:24	37084	00:20:34	140588	66228
hsasz1	66302	hsalg10b	14972	00:03:54	30984	00:05:27	31060	00:16:39	37880	00:37:37	150164	66227
hsasz1	66302	hsbad	16877	00:04:39	21791	00:05:51	21860	00:17:35	28869	00:39:11	145259	66228
hsasz1	66302	hsankr10	38530	00:10:44	32164	00:13:43	32128	00:43:53	41032	01:39:48	185428	66227
hsasz1	66302	hsascc2	51655	00:14:29	31912	00:18:38	31796	01:01:18	41920	02:22:36	206256	66222
hsasz1	66302	hsaff4	90284	00:26:27	32972	00:32:14	32944	01:49:02	46848	05:03:41	267876	66221

Tabela B.2: *Sequências reais de genes do* Homo sapiens, $k = 300$.

Alfa		Beta		KDP1		KDP2		KDP4		KDP5		
Nome	Tam.	Nome	Tam.	Tem.	Mem.	Tem.	Mem.	Tem.	Mem.	Tem.	Mem.	Ocr
hsascl2	4455	hsarhgdig	4398	00:00:33	10868	00:00:55	10804	00:02:32	11564	00:04:48	23840	3816
hsascl2	4455	hsankrd43	5457	00:00:45	3826	00:01:08	10608	00:03:06	11364	00:06:04	24888	3826
hsascl2	4455	hsa1bg	8694	00:01:06	10508	00:01:43	10612	00:05:15	11484	00:09:50	29852	3827
hsascl2	4455	hsalg10b	14972	00:01:49	10240	00:02:42	10292	00:08:41	11768	00:18:45	38320	3830
hsascl2	4455	hsbad	16877	00:02:03	10596	00:03:18	10568	00:09:19	12184	00:20:59	43140	3813
hsascl2	4455	hsankr10	38530	00:04:43	10940	00:07:36	10940	00:23:44	14308	00:49:32	72984	3828
hsascl2	4455	hsascc2	51655	00:06:28	10624	00:10:06	10648	00:32:39	15228	01:03:00	206792	3827
hsascl2	4455	hsasz1	66302	00:07:54	10700	00:12:09	10764	00:39:31	16596	01:30:23	114088	3825
hsascl2	4455	hsaff4	90284	00:10:45	11032	00:17:28	11064	00:58:35	18980	02:21:12	154312	3820
hsankrd43	5457	hsarhgdig	4398	00:00:39	12424	00:01:06	12456	00:03:06	13244	00:05:51	26660	4869
hsankrd43	5457	hsascl2	4455	00:00:40	12292	00:01:06	12444	00:03:02	13136	00:05:53	26860	4873
hsankrd43	5457	hsa1bg	8694	00:01:19	12628	00:02:14	12664	00:06:29	13676	00:12:38	33424	4822
hsankrd43	5457	hsalg10b	14972	00:02:18	12428	00:03:49	12440	00:10:25	16800	00:21:42	42061	4808
hsankrd43	5457	hsbad	16877	00:02:33	12332	00:04:34	12423	00:11:54	14252	00:23:53	46256	4843
hsankrd43	5457	hsankr10	38530	00:06:00	12980	00:09:37	13032	00:30:38	16608	01:02:51	76536	4812
hsankrd43	5457	hsascc2	51655	00:08:09	12804	00:12:36	12732	00:38:25	17304	01:24:13	97544	4827
hsankrd43	5457	hsasz1	66302	00:10:00	12776	00:16:03	12720	00:49:04	18808	01:45:02	117664	4804
hsankrd43	5457	hsaff4	90284	00:14:10	13128	00:21:53	13044	01:03:49	33136	02:53:38	375144	4814
hsa1bg	8694	hsarhgdig	4398	00:01:06	18912	00:01:50	18904	00:05:17	20048	00:10:18	38304	8071
hsa1bg	8694	hsascl2	4455	00:01:08	18904	00:01:52	18960	00:05:06	19972	00:10:26	38232	8083
hsa1bg	8694	hsankrd43	5457	00:01:22	19016	00:02:12	19056	00:06:11	20096	00:12:47	39676	8064
hsa1bg	8694	hsalg10b	14972	00:03:57	19456	00:06:00	19608	00:17:25	21380	00:35:33	54088	8036
hsa1bg	8694	hsbad	16877	00:04:48	20412	00:06:54	20360	00:19:44	22360	00:41:02	59356	7965
hsa1bg	8694	hsankr10	38530	00:10:22	20416	00:15:24	20412	00:45:50	24248	01:42:23	88976	7981
hsa1bg	8694	hsascc2	51655	00:14:10	20676	00:21:02	20736	01:02:39	25540	02:26:05	110728	7956
hsa1bg	8694	hsasz1	66302	00:17:30	20280	00:26:05	20168	01:19:53	26496	03:08:29	130360	7949
hsa1bg	8694	hsaff4	90284	00:24:51	21092	00:36:03	21052	01:50:29	29412	04:37:08	170924	7969
hsalg10b	14972	hsarhgdig	4398	00:01:48	29080	00:03:03	29028	00:08:55	30600	00:18:40	57208	14337
hsalg10b	14972	hsascl2	4455	00:01:51	29592	00:03:02	29564	00:09:07	31204	00:18:33	57672	14335
hsalg10b	14972	hsankrd43	5457	00:02:28	32196	00:03:54	32236	00:11:04	33816	00:23:48	61972	14327
hsalg10b	14972	hsa1bg	8694	00:03:53	32284	00:06:14	32240	00:19:23	34064	00:37:52	66648	14336
hsalg10b	14972	hsbad	16877	00:07:28	31648	00:11:48	31624	00:37:08	34304	01:17:36	79688	14341
hsalg10b	14972	hsankr10	38530	00:28:32	33119	00:31:00	33064	01:21:51	37416	03:11:31	110556	14324
hsalg10b	14972	hsascc2	51655	00:23:04	32508	00:38:23	32508	01:50:49	38168	04:20:52	131408	14353
hsalg10b	14972	hsasz1	66302	00:31:22	33708	00:50:31	33680	02:25:37	40592	06:03:00	152832	14356
hsalg10b	14972	hsaff4	90284	00:43:39	34060	01:09:18	34064	03:06:10	66332	07:47:01	607788	14345
hsbad	16877	hsarhgdig	4398	00:02:08	35472	00:03:46	35448	00:10:50	37172	00:21:36	67864	16226
hsbad	16877	hsascl2	4455	00:02:13	35456	00:03:51	35436	00:11:29	36804	00:21:37	67936	16238
hsbad	16877	hsankrd43	5457	00:02:41	35496	00:04:32	35472	00:13:25	37144	00:26:57	69196	16249
hsbad	16877	hsa1bg	8694	00:04:43	39440	00:07:22	39356	00:21:48	41440	00:43:33	78376	16238
hsbad	16877	hsalg10b	14972	00:08:12	37292	00:12:14	37284	00:29:43	46470	01:00:52	214178	16262
hsbad	16877	hsankr10	38530	00:21:19	39420	00:33:06	39792	01:15:43	55468	02:37:08	334470	16238
hsbad	16877	hsascc2	51655	00:30:20	41004	00:43:04	40972	01:36:02	62638	03:29:21	421426	16236
hsbad	16877	hsasz1	66302	00:38:02	39956	00:55:11	40056	02:16:24	65696	05:15:00	488542	16253
hsbad	16877	hsaff4	90284	00:54:17	41940	01:17:03	41960	02:46:08	75278	10:18:52	257066	16262
hsascc2	51655	hsarhgdig	4398	00:07:03	104016	00:11:22	103968	00:35:39	108380	01:13:47	187652	51036
hsascc2	51655	hsascl2	4455	00:06:49	103848	00:11:30	103800	00:33:01	108512	01:13:08	187244	51033
hsascc2	51655	hsankrd43	5457	00:08:31	105036	00:14:46	105096	00:44:46	109712	01:34:49	189876	51025
hsascc2	51655	hsa1bg	8694	00:16:00	114580	00:23:27	114544	01:11:36	119620	02:35:17	204648	51011
hsascc2	51655	hsalg10b	14972	00:24:45	110304	00:40:01	110436	01:41:07	135608	03:43:56	549194	51041
hsascc2	51655	hsbad	16877	00:30:37	116836	00:44:14	116724	02:09:57	122708	05:11:54	220388	51026
hsascc2	51655	hsankr10	38530	01:06:26	115768	01:40:55	115720	05:36:16	123420	12:52:50	248780	51028
hsascc2	51655	hsasz1	66302	01:57:02	116456	02:54:08	116392	08:59:04	126420	22:58:54	290844	51027
hsascc2	51655	hsaff4	90284	02:41:40	119228	04:09:04	119224	10:03:54	173488	16:26:35	1056456	51023
hsasz1	66302	hsarhgdig	4398	00:07:58	12508	00:13:59	125184	00:45:05	131232	01:38:16	228720	65751
hsasz1	66302	hsascl2	4455	00:08:41	126592	00:14:33	126612	00:46:06	132824	01:37:38	230132	65742
hsasz1	66302	hsankrd43	5457	00:10:55	137924	00:18:26	137900	00:57:27	144016	01:45:01	625098	65663
hsasz1	66302	hsa1bg	8694	00:18:30	142220	00:29:05	142200	01:33:08	148536	03:20:17	252008	65670
hsasz1	66302	hsalg10b	14972	00:31:44	143232	00:53:58	143220	02:15:38	175114	06:20:53	262264	65647
hsasz1	66302	hsbad	16877	00:36:06	139652	00:55:19	139620	02:51:47	146592	06:25:37	263008	65693
hsasz1	66302	hsankr10	38530	01:23:16	145784	02:16:01	145784	05:40:29	185978	12:54:13	840458	65651
hsasz1	66302	hsascc2	51655	01:49:18	142936	02:56:23	143020	07:42:34	187760	17:56:11	923838	65679
hsasz1	66302	hsaff4	90284	03:22:57	148524	05:30:45	148600	13:56:47	211022	14:50:38	1206654	65657

Tabela B.3: *Sequências reais de genes* Homo sapiens, $k = 600$.

Alfa		Beta		KDP1		KDP2		KDP4		KDP5		
Nome	Tam.	Nome	Tam.	Tem.	Mem.	Tem.	Mem.	Tem.	Mem.	Tem.	Mem.	Ocr
hsascl2	4455	hsarhgdig	4398	00:00:58	15208	00:01:37	15118	00:04:27	16180	00:08:26	33360	3194
hsascl2	4455	hsankrd43	5457	00:01:14	15368	00:01:52	42608	00:05:06	45646	00:09:59	99968	3204
hsascl2	4455	hsa1bg	8694	00:01:56	15060	00:03:01	15208	00:09:14	16458	00:17:17	42784	3207
hsascl2	4455	hsalg10b	14972	00:03:04	14580	00:04:33	14654	00:14:39	16754	00:31:39	54562	3255
hsascl2	4455	hsbad	16877	00:03:34	15056	00:05:53	15120	00:17:52	16712	00:35:31	47664	3195
hsascl2	4455	hsankr10	38530	00:08:28	15576	00:13:39	15576	00:42:36	20372	01:28:55	103912	3213
hsascl2	4455	hsascc2	51655	00:10:57	15068	00:17:06	15102	00:55:17	21598	01:46:41	293292	3216
hsascl2	4455	hsasz1	66302	00:13:24	14992	00:21:39	23192	01:14:49	20980	02:50:13	118340	3232
hsascl2	4455	hsaff4	90284	00:18:24	15540	00:29:54	15584	01:40:16	26734	04:01:41	217369	3226
hsankrd43	5457	hsarhgdig	4398	00:01:09	18792	00:02:04	18792	00:05:36	19592	00:10:38	33008	4314
hsankrd43	5457	hsascl2	4455	00:01:14	18640	00:02:12	18732	00:05:42	19536	00:10:55	33160	4311
hsankrd43	5457	hsa1bg	8694	00:02:12	18960	00:03:44	19014	00:10:50	20534	00:21:07	50182	4232
hsankrd43	5457	hsalg10b	14972	00:03:43	18544	00:06:10	18562	00:16:50	25068	00:35:04	62760	4178
hsankrd43	5457	hsbad	16877	00:04:27	18764	00:07:55	18784	00:24:27	20600	00:46:57	52592	4245
hsankrd43	5457	hsankr10	38530	00:10:20	19400	00:16:34	19478	00:52:45	24822	01:48:15	114392	4180
hsankrd43	5457	hsascc2	51655	00:13:40	19108	00:23:31	19076	01:12:04	23828	02:38:32	103832	4222
hsankrd43	5457	hsasz1	66302	00:17:09	18804	00:29:59	18900	01:36:51	24876	03:46:58	123792	4181
hsankrd43	5457	hsaff4	90284	00:24:19	19460	00:37:34	19334	01:49:32	49118	04:58:02	556086	4190
hsa1bg	8694	hsarhgdig	4398	00:01:58	31124	00:02:50	31048	00:07:28	32176	00:18:12	50480	7461
hsa1bg	8694	hsascl2	4455	00:01:49	31124	00:03:15	31164	00:07:36	32216	00:16:24	50380	7485
hsa1bg	8694	hsankrd43	5457	00:02:19	31456	00:03:35	31400	00:11:26	32544	00:21:04	51996	7467
hsa1bg	8694	hsalg10b	14972	00:06:07	31072	00:08:54	31180	00:29:55	33080	01:08:10	65628	7415
hsa1bg	8694	hsbad	16877	00:07:28	32716	00:10:17	32808	00:33:01	34868	01:12:18	71696	7283
hsa1bg	8694	hsankr10	38530	00:16:44	32224	00:23:02	32268	01:25:13	36220	03:18:13	100888	7366
hsa1bg	8694	hsascc2	51655	00:24:16	32832	00:31:08	32768	01:50:38	37644	04:37:42	122908	7276
hsa1bg	8694	hsasz1	66302	00:28:11	32052	00:39:21	31996	02:23:44	38252	06:02:50	141644	7353
hsa1bg	8694	hsaff4	90284	00:40:19	33772	00:53:35	33704	03:40:40	41920	15:28:28	183448	7315
hsalg10b	14972	hsarhgdig	4398	00:03:18	49916	00:05:07	49852	00:17:31	51592	00:32:11	74184	13788
hsalg10b	14972	hsascl2	4455	00:03:14	51016	00:05:47	51064	00:17:23	52560	00:33:20	79168	13786
hsalg10b	14972	hsankrd43	5457	00:04:22	55696	00:06:26	55700	00:18:56	57580	00:36:33	85644	13771
hsalg10b	14972	hsa1bg	8694	00:07:34	55220	00:12:09	55142	00:37:46	58262	01:13:47	113996	13735
hsalg10b	14972	hsbad	16877	00:14:39	54408	00:25:08	54408	01:19:20	56952	02:39:02	102556	13761
hsalg10b	14972	hsankr10	38530	00:32:00	57684	00:46:38	57640	02:34:30	62068	05:35:33	135116	13774
hsalg10b	14972	hsascc2	51655	00:40:40	55736	01:01:41	55748	03:40:50	61552	08:17:25	154736	13765
hsalg10b	14972	hsasz1	66302	00:54:59	58264	01:34:29	58236	04:51:02	65196	11:34:06	177428	13743
hsalg10b	14972	hsaff4	90284	01:19:25	58348	02:06:05	58354	05:38:43	113632	14:09:41	1041196	13763
hsbad	16877	hsarhgdig	4398	00:04:11	61608	00:07:23	61564	00:21:15	64558	00:42:21	117868	15585
hsbad	16877	hsascl2	4455	00:04:12	61364	00:07:18	61326	00:21:45	63696	00:40:57	117580	15605
hsbad	16877	hsankrd43	5457	00:05:07	61472	00:08:39	61429	00:25:35	64324	00:51:23	119834	15599
hsbad	16877	hsa1bg	8694	00:08:57	66880	00:13:59	66740	00:41:22	70270	01:22:38	132904	15622
hsbad	16877	hsalg10b	14972	00:14:36	63688	00:21:47	63674	00:52:55	79362	01:48:22	365778	15684
hsbad	16877	hsankr10	38530	00:40:48	66588	01:03:21	67214	02:24:55	93696	05:00:45	564986	15605
hsbad	16877	hsascc2	51655	00:56:04	69228	01:19:36	69172	02:57:30	105752	06:26:57	711504	15625
hsbad	16877	hsasz1	66302	01:10:19	67560	01:42:01	67728	04:12:11	111082	09:42:23	826056	15668
hsbad	16877	hsaff4	90284	01:38:34	71016	02:19:54	71052	05:01:40	127466	18:43:44	435282	15620
hsascc2	51655	hsarhgdig	4398	00:12:53	189548	00:20:46	189452	01:05:08	197508	02:14:05	341964	50434
hsascc2	51655	hsascl2	4455	00:13:53	190092	00:24:40	190136	01:13:14	194756	02:32:53	273420	50441
hsascc2	51655	hsankrd43	5457	00:16:39	192416	00:28:52	192530	01:27:31	200978	03:05:22	347830	50432
hsascc2	51655	hsa1bg	8694	00:27:47	205580	00:40:43	205518	02:04:20	214626	04:29:39	367186	50413
hsascc2	51655	hsalg10b	14972	00:46:54	200076	01:15:50	200316	03:11:36	245972	07:04:21	996158	50452
hsascc2	51655	hsbad	16877	00:58:02	211280	01:30:50	211136	04:52:14	217188	10:20:57	314744	50423
hsascc2	51655	hsankr10	38530	02:05:17	207776	03:10:19	207692	10:34:09	221510	24:17:27	446510	50433
hsascc2	51655	hsasz1	66302	03:43:28	208712	06:04:25	208712	20:20:31	218652	50:36:15	383172	50430
hsascc2	51655	hsaff4	90284	05:10:07	214820	07:57:46	214820	19:18:26	312584	31:32:31	1903476	50423
hsasz1	66302	hsarhgdig	4398	00:15:45	227696	00:27:39	227886	01:29:08	2388940	03:14:16	4163626	65239
hsasz1	66302	hsascl2	4455	00:16:09	231868	00:27:04	231914	01:25:44	243276	03:01:35	421512	65215
hsasz1	66302	hsankrd43	5457	00:21:56	253780	00:37:02	253729	01:55:26	264996	03:31:00	1150182	65022
hsasz1	66302	hsa1bg	8694	00:36:31	257752	00:57:24	257726	03:03:50	269196	06:35:20	456736	65086
hsasz1	66302	hsalg10b	14972	01:04:09	263036	01:49:06	263010	04:34:11	321588	12:49:58	481618	64999
hsasz1	66302	hsbad	16877	01:08:39	254572	01:55:52	254668	06:19:18	261680	13:54:08	378068	65121
hsasz1	66302	hsankr10	38530	02:43:29	266548	04:27:03	266548	11:08:30	340034	25:20:04	1536676	65020
hsasz1	66302	hsascc2	51655	03:33:01	260492	05:43:46	260648	15:01:30	342182	34:57:23	1683638	65094
hsasz1	66302	hsaff4	90284	06:39:14	270228	10:50:38	270364	27:26:05	383940	29:12:00	2195412	65013

Tabela B.4: *Similaridade entre as sequências reais de genes do* Homo sapiens

Sequência		Ocorrências, k = 30			Ocorrências, k = 300			Ocorrências, k = 600			Sim.
alfa	*beta*	Maior	Menor	Média	Maior	Menor	Média	Maior	Menor	Média	
4455	4398	96	65	77	721	614	667	1346	1244	1302	48,4%
4455	5457	93	64	76	726	612	661	1365	1227	1308	43,5%
4455	8694	92	67	76	693	622	653	1310	1237	1265	34,8%
4455	14972	86	66	74	663	603	630	1222	1162	1191	17,8%
4455	16877	88	68	77	679	620	647	1312	1232	1266	18,4%
4455	38530	89	68	77	722	612	658	1363	1223	1298	7,6%
4455	51655	87	70	77	660	614	638	1264	1213	1235	5,9%
4455	66302	85	66	75	664	606	628	1231	1164	1197	4,5%
4455	90284	89	69	76	700	612	649	1326	1214	1264	3,3%
5457	4398	93	57	72	725	550	623	1365	1090	1225	43,9%
5457	4455	91	57	72	722	559	624	1355	1114	1226	43,5%
5457	8694	93	66	75	697	596	645	1325	1202	1261	39,1%
5457	14972	89	66	75	673	593	640	1293	1168	1232	22,0%
5457	16877	92	66	75	679	593	633	1313	1186	1243	21,1%
5457	38530	88	68	78	725	600	660	1394	1182	1290	9,1%
5457	51655	87	70	77	672	601	637	1278	1209	1241	7,0%
5457	66302	89	68	77	672	605	642	1301	1174	1241	5,3%
5457	90284	92	70	78	706	604	656	1341	1197	1277	4,0%
8694	4398	153	57	75	701	547	635	1323	1116	1242	39,1%
8694	4455	88	61	73	690	567	628	1323	1134	1233	34,8%
8694	5457	89	65	74	696	590	636	1316	1194	1257	34,2%
8694	14972	235	66	81	841	600	652	1443	1177	1245	35,7%
8694	16877	234	67	90	838	610	698	1551	1227	1343	34,8%
8694	38530	236	69	89	800	611	686	1423	1216	1307	15,9%
8694	51655	211	70	91	909	611	703	1509	1211	1343	11,4%
8694	66302	224	69	89	821	606	682	1430	1183	1285	8,1%
8694	90284	230	70	93	904	614	708	1678	1230	1359	6,6%
14972	4398	152	56	67	644	520	567	1198	1018	1106	18,1%
14972	4455	88	56	68	641	535	576	1200	1070	1130	17,8%
14972	5457	86	64	74	664	587	638	1293	1173	1256	22,0%
14972	8694	235	65	76	847	594	641	1439	1192	1244	35,7%
14972	16877	226	66	76	790	582	624	1368	1168	1221	43,1%
14972	38530	228	69	80	821	602	659	1457	1196	1290	25,0%
14972	51655	259	69	79	835	609	639	1420	1196	1246	18,3%
14972	66302	181	68	81	825	603	667	1439	1197	1306	18,5%
14972	90284	234	71	82	850	608	667	1474	1203	1300	14,1%
16877	4398	236	66	95	700	572	627	1302	1149	1225	18,5%
16877	4455	87	59	73	668	579	622	1288	1172	1221	18,4%
16877	5457	92	65	73	679	593	624	1318	1180	1225	21,1%
16877	8694	236	66	95	827	610	706	1503	1226	1337	34,8%
16877	14972	226	66	86	799	597	663	1384	1162	1262	43,1%
16877	38530	251	68	100	865	612	708	1457	1214	1330	29,6%
16877	51655	265	69	105	947	616	737	1565	1215	1388	25,4%
16877	66302	313	66	100	929	600	712	1519	1168	1339	19,7%
16877	90284	326	67	108	1067	613	750	1729	1222	1419	15,5%
51655	4398	160	55	73	686	553	610	1291	1109	1198	6,0%
51655	4455	89	55	71	661	560	610	1264	1131	1200	5,9%
51655	5457	88	64	73	665	589	619	1268	1173	1217	7,0%
51655	8694	211	64	86	873	591	682	1506	1185	1308	11,4%
51655	14972	260	67	81	819	597	654	1420	1183	1267	18,3%
51655	16877	263	67	90	962	601	697	1619	1200	1344	25,4%
51655	38530	246	68	90	951	613	688	1556	1203	1320	43,7%
51655	66302	250	67	91	937	609	692	1538	1200	1324	44,5%
51655	90284	307	69	94	1047	614	710	1707	1211	1366	38,4%
66302	4398	161	52	67	676	523	569	1215	1037	1110	4,3%
66302	4455	88	57	68	648	536	577	1245	1072	1133	4,5%
66302	5457	92	65	74	675	594	636	1302	1178	1251	5,3%
66302	8694	227	64	79	882	604	658	1500	1196	1270	8,1%
66302	14972	179	67	80	807	608	664	1439	1194	1300	18,5%
66302	16877	319	66	81	897	591	644	1577	1173	1253	19,7%
66302	38530	266	69	84	922	617	677	1521	1215	1317	38,8%
66302	51655	247	70	84	921	609	661	1549	1201	1282	44,5%
66302	90284	335	69	88	946	617	690	1617	1211	1335	44,8%

Referências Bibliográficas

[1] RIBEIRO, M. C. M. *Genética Molecular*. Florianópolis: Universidade Federal de Santa Catarina. Biologia/EaD/UFSC, 2009. Citado 4 vezes nas páginas vi, 4, 5 e 6.

[2] GUSFIELD, D. *Algoritms on strings, trees, and sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997. Disponível em: <https://books.google.com.br/books?id=Ofw5w1yuD8kC>. Citado 7 vezes nas páginas 1, 3, 8, 10, 25, 28 e 31.

[3] SO BIOLOGIA. *A multiplicação dos fragmentos de DNA*. 2016. Disponível em: <http://www.sobiologia.com.br/conteudos/Biotecnologia/PCR.php>. Acesso em: 08 jun. 2016. Citado 3 vezes nas páginas 3, 4 e 7.

[4] MULLIS, K. B. F. F. et al. Specific enzymatic amplification of DNA in vitro: the polymerase chain reaction. In: *Cold Spring Harbor symposia on quantitative biology*. NY: Cold Spring Harbor Laboratory Press, 1986. v. 51, p. 263–273. Citado 2 vezes nas páginas 3 e 7.

[5] ALBERTS, B. et al. *Biologia Molecular da Célula*. Artmed Editora, 2009. Disponível em: <https://books.google.com.br/books?id=bGzbgGZ_A9UC>. Citado 4 vezes nas páginas 3, 4, 6 e 7.

[6] ROBERTIS, E. D.; HIB, J. *De Robertis Bases da Biologia celular e molecular*. 3. ed. Rio de Janeiro: Guanabara Koogan, 2001. Citado 3 vezes nas páginas 3, 5 e 8.

[7] ITO, M. et al. Polynomial-time Algorithms for Computing Characteristic Strings. *Systems and Computers in Japan*, Scripta Technico, Inc., v. 26, n. 3, p. 30–38, 1995. Citado 4 vezes nas páginas 3, 8, 25 e 29.

[8] CABELLO, G. M. *Diagnóstico de doenças genéticas: métodos de rastreamento*. 2016. Disponível em: <http://www.dbbm.fiocruz.br/ghente/ciencia/genetica/diagnostico.htm>. Acesso em: 17 mar. 2016. Citado na página 3.

[9] LANDAU, G. M.; VISHKIN, U. Fast parallel and serial approximate string matching. *Journal of Algorithms*, v. 10, n. 2, p. 157–169, 1989. Disponível em: <http://www.sciencedirect.com/science/article/pii/0196677489900102>. Citado 4 vezes nas páginas 3, 20, 21 e 24.

[10] LANDAU, G. M.; VISHKIN, U.; NUSSINOV, R. An efficient string matching algorithm with k differences for nucleotide and amino acid sequences. *Nucleic Acids Research*, v. 14, n. 1, p. 31–46, 1986. Disponível em: <http://dx.doi.org/10.1093/nar/14.1.31>. Citado 3 vezes nas páginas 3, 23 e 24.

[11] CARVALHO, G. M. *Biologia Molecular*. 2016. Disponível em: <http://biologia-molecular.info/>. Acesso em: 02 abr. 2016. Citado na página 3.

[12] LIMA, L. I. S. de. *O Problema do Alinhamento de Segmentos*. Dissertação (Mestrado) — Universidade Federal de Mato Grosso do Sul, 2013. Citado na página 5.

[13] MONTERA, L.; NICOLETTI, M. The pcr primer design as a metaheuristic search process. *Artificial Intelligence and Soft Computing–ICAISC 2008*, Springer, p. 963–973, 2008. Citado na página 7.

[14] MULLIS, K. B. et al. PCR-the polymerase chain reaction. *Trends in Genetics*, Elsevier Science Publishers (Biomedical Division), The Netherlands Amsterdam, v. 11, n. 6, p. 249–249, 1995. Citado na página 7.

[15] ABD-ELSALAM, K. A. Bioinformatic tools and guideline for pcr primer design. *African Journal of Biotechnology*, Academic Journals (Kenya), v. 2, n. 5, p. 91–95, 2003. Citado na página 8.

[16] RISTAD, E. S.; YIANILOS, P. N. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, v. 20, n. 5, p. 522–532, 1998. Citado na página 10.

[17] MASEK, W. J.; PATERSON, M. S. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, Elsevier, v. 20, n. 1, p. 18–31, 1980. Citado na página 10.

[18] CORMEN, T. et al. *Algoritmos: Teoria e Prática*. 3. ed. Rio de Janeiro: Campus, 2012. Citado na página 10.

[19] NAVARRO, G. A guided tour to approximate string matching. *ACM Computing Surveys (CSUR)*, ACM, v. 33, n. 1, p. 31–88, 2001. Citado 2 vezes nas páginas 12 e 23.

[20] SZWARCFITER, J. L.; MARKENZON, L. *Estruturas de Dados e seus Algoritmos*. [S.l.]: Livros Técnicos e Científicos, 1994. Citado na página 14.

[21] UKKONEN, E. On-line construction of suffix trees. *Algorithmica*, Springer, v. 14, n. 3, p. 249–260, 1995. Citado na página 15.

[22] MIRANDA, R. C. de C. *Um algoritmo para pesquisa aproximada de padrões baseado no método de Landau e Vishkin e uso de arranjos de sufixos para reduzir o uso de espaço*. Dissertação (Mestrado) — Universidade em Brasília, 2006. Citado 2 vezes nas páginas 15 e 25.

[23] MANBER, U.; MYERS, G. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, SIAM, v. 22, n. 5, p. 935–948, 1993. Citado na página 16.

[24] HUYNH, T. N. et al. Approximate string matching using compressed suffix arrays. *Theoretical Computer Science*, Elsevier, v. 352, n. 1, p. 240–249, 2006. Citado na página 16.

[25] ILIE, L.; NAVARRO, G.; TINTA, L. The longest common extension problem revisited and applications to approximate string searching. *Journal of Discrete Algorithms*, Elsevier, v. 8, n. 4, p. 418–428, 2010. Citado 3 vezes nas páginas 16, 26 e 45.

[26] LARSSON, N. J.; SADAKANE, K. Faster suffix sorting. *Theoretical Computer Science*, Elsevier, v. 387, n. 3, p. 258–272, 2007. Citado 3 vezes nas páginas 16, 18 e 34.

[27] KO, P.; ALURU, S. Space efficient linear time construction of suffix arrays. *Journal of Discrete Algorithms*, Elsevier, v. 3, n. 2, p. 143–156, 2005. Citado na página 16.

[28] FISCHER, J.; HEUN, V. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM Journal on Computing*, SIAM, v. 40, n. 2, p. 465–492, 2011. Citado na página 17.

[29] BENDER, M. A. et al. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, Elsevier, v. 57, n. 2, p. 75–94, 2005. Citado na página 17.

[30] DANIEL, P. *Range Minimum Query and Lowest Common Ancestor*. 2017. Disponível em: <https://www.topcoder.com/community/data-science/data-science-tutorials/range-minimum-query-and-lowest-common-ancestor/>. Citado na página 17.

[31] BENDER, M. A.; FARACH-COLTON, M. The LCA problem revisited. In: SPRINGER. *Latin American Symposium on Theoretical Informatics*. [S.l.], 2000. p. 88–94. Citado na página 17.

[32] GROSSI, R.; VITTER, J. S. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, SIAM, v. 35, n. 2, p. 378–407, 2005. Citado na página 18.

[33] SIRÉN, J. Compressed suffix arrays for massive data. In: SPRINGER. *International Symposium on String Processing and Information Retrieval*. [S.l.], 2009. p. 63–74. Citado na página 18.

[34] SADAKANE, K. Compressed suffix trees with full functionality. *Theory of Computing Systems*, Springer, v. 41, n. 4, p. 589–607, 2007. Citado 2 vezes nas páginas 18 e 33.

[35] VÄLIMÄKI, N. et al. Engineering a compressed suffix tree implementation. *Journal of Experimental Algorithmics (JEA)*, ACM, v. 14, p. 2, 2009. Citado 2 vezes nas páginas 18 e 33.

[36] LANDAU, G. M.; VISHKIN, U. Introducing efficient parallelism into approximate string matching and a new serial algorithm. *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, ACM New York, p. 220–230, 1986. Citado 4 vezes nas páginas 20, 22, 23 e 24.

[37] UKKONEN, E. Finding approximate patterns in strings. *Journal of algorithms*, Elsevier, v. 6, n. 1, p. 132–137, 1985. Citado na página 20.

[38] MIRANDA, R. C. de C.; AYALA-RINCÓN, M.; SOLON, L. Modifications of the Landau-Vishkin Algorithm Computing Longest Common Extensions via Suffix Arrays and Efficient RMQ Computations. In: SPRINGER. *Brazilian Symposium on Bioinformatics*. [S.l.], 2005. p. 210–213. Citado 3 vezes nas páginas 20, 24 e 25.

[39] UKKONEN, E.; WOOD, D. Approximate string matching with suffix automata. *Algorithmica*, Springer, v. 10, n. 5, p. 353–364, 1993. Citado na página 23.

[40] LANDAU, G. M.; VISHKIN, U. Efficient string matching in the presence of errors. In: IEEE. *26th Annual Symposium on Foundations of Computer Science*. [S.l.], 1985. p. 126–136. Citado 2 vezes nas páginas 23 e 24.

[41] NAVARRO, G. et al. Indexing methods for approximate string matching. *IEEE Data Eng. Bull.*, v. 24, n. 4, p. 19–27, 2001. Citado na página 23.

[42] LANDAU, G. M.; VISHKIN, U. Fast string matching with k differences. *Journal of Computer and System Sciences*, Elsevier, v. 37, n. 1, p. 63–78, 1988. Citado 2 vezes nas páginas 23 e 24.

[43] GALIL, Z.; GIANCARLO, R. Data structures and algorithms for approximate string matching. *Journal of Complexity*, Elsevier, v. 4, n. 1, p. 33–72, 1988. Citado 2 vezes nas páginas 23 e 24.

[44] GALIL, Z.; PARK, K. An improved algorithm for approximate string matching. *SIAM Journal on Computing*, SIAM, v. 19, n. 6, p. 989–999, 1990. Citado 2 vezes nas páginas 23 e 24.

[45] CHANG, W. I.; LAWLER, E. L. Sublinear approximate string matching and biological applications. *Algorithmica*, Springer, v. 12, n. 4, p. 327–344, 1994. Citado 2 vezes nas páginas 23 e 24.

[46] TAKAOKA, T. Approximate pattern matching with samples. In: SPRINGER. *International Symposium on Algorithms and Computation*. [S.l.], 1994. p. 234–242. Citado na página 24.

[47] LANDAU, G. M.; MYERS, E. W.; SCHMIDT, J. P. Incremental string comparison. *SIAM Journal on Computing*, SIAM, v. 27, n. 2, p. 557–582, 1998. Citado na página 24.

[48] NASCIMENTO, F. M. S. do. *Aplicação da técnica PCR para detecção de bactérias potencialmente patogênicas em um sistema UASB-lagoas de polimento para tratamento de esgoto doméstico*. Dissertação (Mestrado) — Universidade Federal de Minas Gerais, 2008. Citado na página 27.

[49] LOUDON, K. *Dominando algoritmos com C*. São Paulo: Editora Ciencia Moderna Ltda, 2000. Citado na página 31.

[50] DEITEL, M.; DEITEL, P. *C++ como programar. 5a edição*. São Paulo: Editora Pearson, 2005. Citado na página 31.

[51] SUTTER, H. *Programação avançada em C++*. São Paulo: Pearson-Mahron Books, 2006. Citado na página 31.

[52] KASAI, T. et al. Linear-time longest-common-prefix computation in suffix arrays and its applications. In: SPRINGER. *Annual Symposium on Combinatorial Pattern Matching*. [S.l.], 2001. p. 181–192. Citado na página 34.

[53] HENNESSY, J. L.; PATTERSON, D. A. *Organização e Projeto de Computadores: a interface hardware/software*. [S.l.]: Elsevier Brasil, 2014. Citado na página 44.

[54] PROSSER, P. *External Memory Computing*. Citado na página 44.

[55] SAIKKONEN, R.; SOISALON-SOININEN, E. Cache-sensitive memory layout for binary trees. In: SPRINGER. *Fifth IFIP International Conference On Theoretical Computer Science–TCS 2008*. [S.l.], 2008. p. 241–255. Citado na página 44.

[56] NEEDLEMAN, S. B.; WUNSCH, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, Elsevier, v. 48, n. 3, p. 443–453, 1970. Citado na página 45.

[57] CASTELÃO, A. B. C. *Contribuições da análise genômica para o controle da tuberculose bovina*. Tese (Doutorado) — Universidade Federal de Mato Grosso do Sul, 2016. Citado na página 64.

[58] PRESSMAN, R.; MAXIM, B. *Engenharia de Software-8ª Edição*. [S.l.]: McGraw Hill Brasil, 2016. Citado na página 65.