Ferramenta PMT

1. Identificação

Este projeto foi desenvolvido por Heitor da Silva Santos e Leonardo dos Anjos Silva. Cada integrante da equipe ficou responsável por implementar um algoritmo de busca exata e um de busca aproximada, tendo Heitor implementado KMP e Sellers e Leonardo implementado Aho-Corasick e Wu-Manber. Outras funcionalidades foram desenvolvidas em conjunto.

2. Implementação

Foram implementados 4 (quatro) algoritmos, sendo 2 (dois) para busca exata - KMP e Aho-Corasick - e 2 (dois) para busca aproximada - Sellers e Wu-Manber. Quando um algoritmo não for especificado através da opção -a (--algorithm), a ferramenta decidirá o melhor algoritmo para usar através da seguinte análise:

- Se a opção -e (--edit) for utilizada, um dos algoritmos de busca aproximada será escolhido:
  - Se o tamanho do maior padrão for menor ou igual a 64 (sessenta e quatro), o algoritmo Wu-Manber será executado
  - Caso contrário, o algoritmo Sellers será executado
- Caso contrário, se o número de padrões que serão buscados é maior do que 1 (um), o algoritmo Aho-Corasick será executado
- Caso contrário, o algoritmo KMP será executado

2.1. Algoritmos de casamento exato

Foram implementados os algoritmos KMP e Aho-Corasick para realizar buscas exatas de padrões em arquivos de texto. Estes algoritmos não impõem nenhuma restrição adicional, de forma que podem ser executados para quaisquer padrões e arquivos de texto aceitos pela ferramenta.

Devido a natureza do algoritmo, quando não especificado um algoritmo através da opção -a, Aho-Corasick é preferido sobre KMP nas situações em que múltiplos padrões são fornecidos (através da especificação de um arquivo com a opção -p).

2.2. Algoritmos de casamento aproximado

Para realizar buscas aproximadas, foram implementados os algoritmos Sellers e Wu-Manber. O algoritmo de Wu-Manber, porém, só pode ser executado se todos os padrões tiverem tamanho menor ou igual a 64 (sessenta e quatro), de forma que a ferramenta avisa o usuário de tal restrição quando este tenta forçar a utilização desse algoritmo através da opção -a em um conjunto de padrões que contém um padrão com tamanho maior do que o máximo permitido.

Devido sua complexidade temporal $O(nr)$, sendo n o tamanho do texto e r a distância de edição máxima, e observando que a distância de edição máxima normalmente é um valor pequeno, o algoritmo de Wu-Manber é

preferido sobre o Sellers (de complexidade temporal *O(nm)*, sendo *n* o tamanho do texto e *m* o tamanho do padrão) quando não especificado um algoritmo através da opção -a.

2.3. Detalhes de implementação relevantes

Todos os algoritmos implementados recebem como entrada o texto e uma lista de padrões. Com exceção do Aho-Corasick, todos os algoritmos serão executados uma vez para cada padrão na lista. Optamos por construir a ferramenta dessa forma para facilitar a implementação de algumas funcionalidades, sem prejudicar a eficiência do programa.

A ferramenta lida apenas com caracteres da tabela ASCII, não tendo suporte a caracteres Unicode. Além disso, os algoritmos foram implementados de forma a **não** receberem um alfabeto como entrada. Sendo assim, todos os algoritmos assumem que o alfabeto utilizado são os caracteres da tabela ASCII.

Pensando no consumo de memória, optamos por processar cada arquivo de texto uma linha por vez, de forma que os algoritmos são executados uma vez para cada linha dos textos, salvo pré-processamentos sobre o(s) padrão(ões) fornecidos para a ferramenta.

Observando a ineficiência dos algoritmos de casamento aproximado sobre arquivos de texto muito grandes (constatado durante a fase de testes), foram necessárias algumas otimizações sobre os algoritmos de Sellers e de Wu-Manber. Em geral, essas otimizações consistiram em realizar menos operações de instanciação e/ou cópia de listas.

3. Testes e Resultados

Foram realizados testes de corretude e eficiência. Ambos foram assistidos por scripts em *bash* que executaram as ferramentas *pmt* e *grep*/*agrep* com as mesmas entradas.

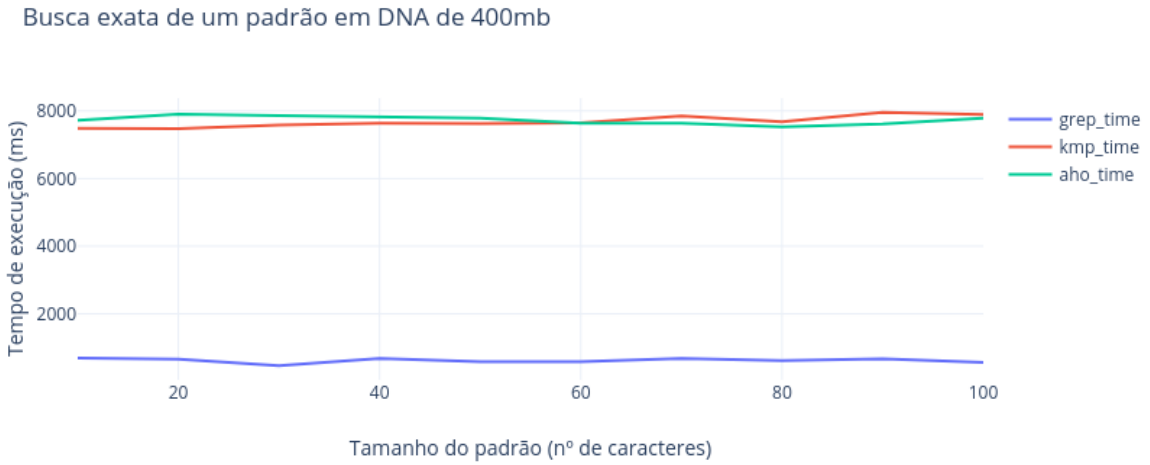
Para os testes de corretude, as ferramentas foram executadas **sem** a opção -c (que reportaria apenas o número de ocorrências dos padrões nos textos) e suas saídas foram direcionadas para arquivos distintos. Em seguida, esses arquivos eram comparados utilizando a ferramenta *diff*. Caso esta última reportasse alguma diferença entre os arquivos, o teste falhava. Durante a execução dos testes, foi constatado que a ferramenta *agrep* não garante que a saída conterá as linhas do texto na mesma ordem em que se encontram no arquivo de entrada, o que prejudicava a análise do *diff*. Dessa forma, fizemos uso ainda da ferramenta *sort* para ordenar as linhas das saídas do *pmt* e do *agrep*, antes da execução do *diff*.

Para os testes de eficiência, as ferramentas foram executadas **com** a opção -c, para diminuir o tempo de execução. Foi utilizada a ferramenta *time* para medir o tempo de execução de cada ferramenta. Cada teste foi realizado três vezes e foram calculadas as médias dos tempos de execução.

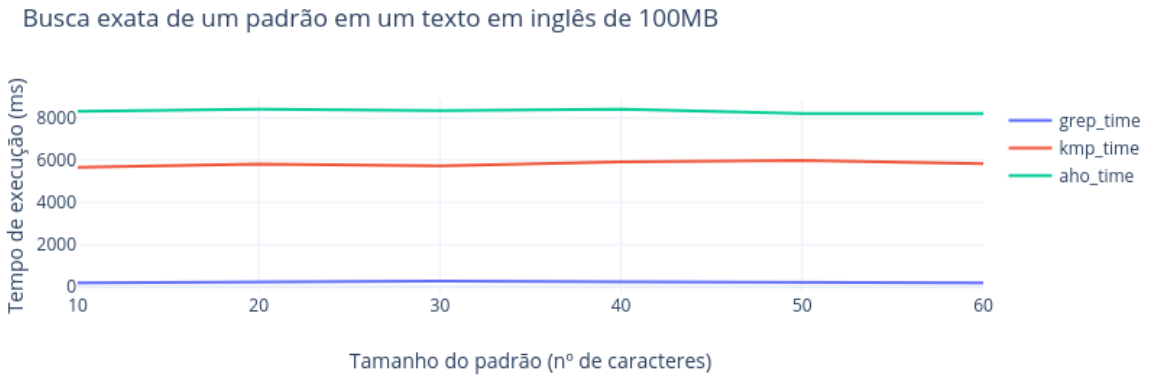
3.1. Algoritmos de casamento exato

Para o casamento exato, nós primeiramente fizemos testes com apenas um padrão a ser buscado no texto. Para tal, utilizamos os algoritmos

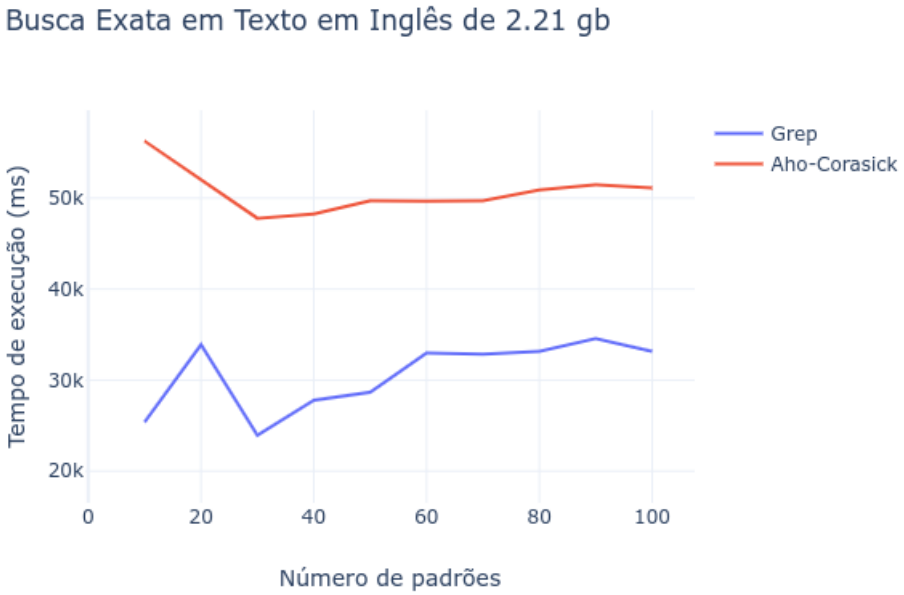
KMP e Aho-Corasick. Foram utilizados 10 diferentes padrões de tamanhos de 10 até 100 caracteres, a serem buscados em um arquivo com uma sequência de DNA de 400MB.



Depois, nós também utilizamos um texto em inglês de 100MB para pesquisar um padrão de tamanho entre 10 e 60 caracteres, também utilizando KMP e Aho-Corasick.



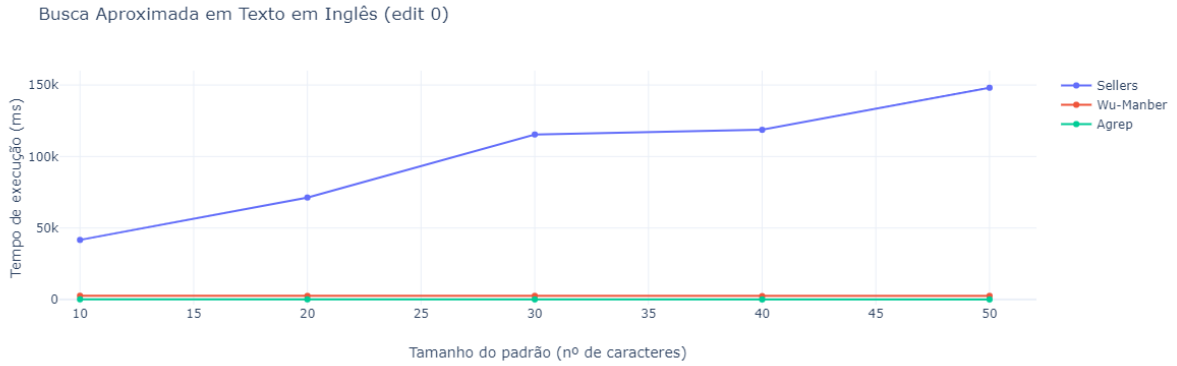
Nosso último teste com casamento exato foi procurar vários padrões em um arquivo de texto. Nós utilizamos um texto em inglês de 2.21GB e o algoritmo que nós utilizamos foi o Aho-Corasick.

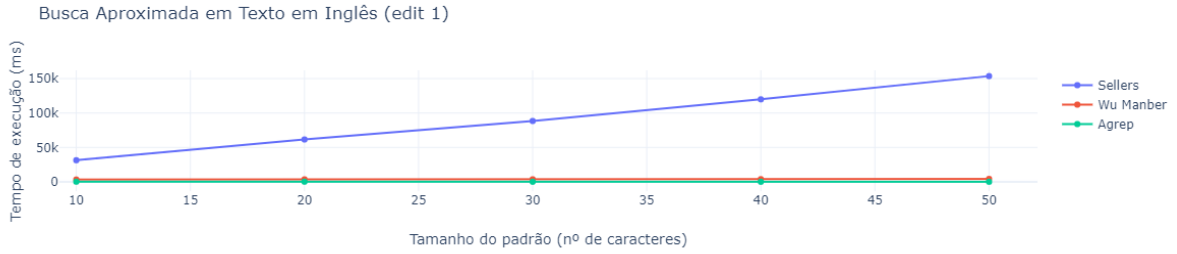


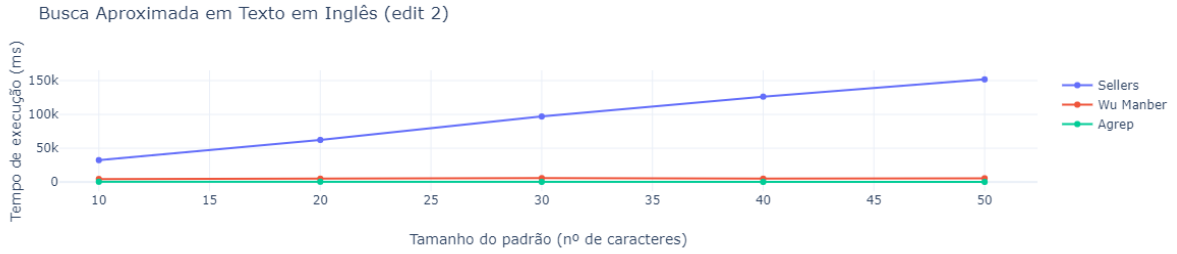
Após todos esses testes, concluímos que nos casos de pesquisa com um padrão, o KMP performa, pelo menos, tão bem quanto o Aho-Corasick, tendo um tempo de execução 25% menor nos casos em que o alfabeto é maior. Já no caso de estarmos pesquisando por vários padrões, primeiro percebemos que o tempo de execução do Aho-Corasick cresce lentamente conforme o número de padrões aumenta, enquanto que o tempo do *grep* cresce mais rapidamente. Isso é esperado uma vez que o *grep* utiliza como base o algoritmo de Boyer-Moore, que não aceita múltiplos padrões, então ele precisa executar o algoritmo uma vez para cada padrão, enquanto o Aho-Corasick faz um pré-processamento sobre todos os padrões, possibilitando uma busca mais otimizada sobre o texto.

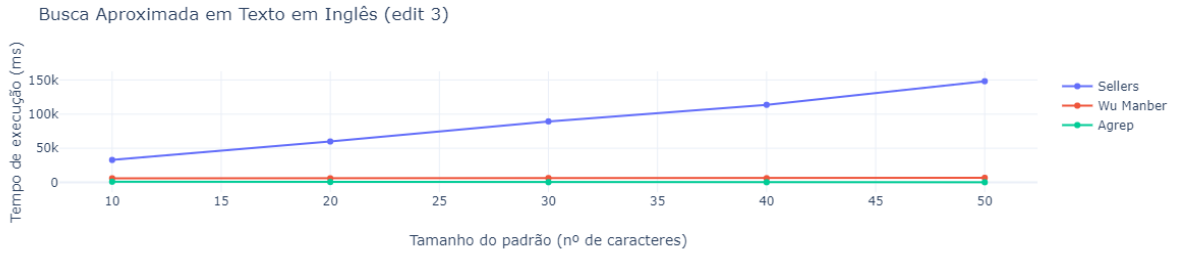
3.2. Algoritmos de casamento aproximado

Executamos testes com os algoritmos de Sellers e Wu-Manber, com tamanhos de padrão variando de 5 em 5 desde 10 até 50 e distâncias de edição máxima variando de 1 em 1 desde 0 até 3. Comparamos os resultados dos dois algoritmos entre si e com o agrep. Os resultados são mostrados a seguir.









É possível perceber que o Wu-Manber se comporta de forma constante para uma mesma distância de edição máxima. Enquanto isso, o tempo de execução do Sellers cresce à medida que o tamanho do padrão cresce. Essas observações são esperadas, dada as complexidades temporais dos algoritmos.

4. Conclusão

Com a execução dos testes, podemos verificar que o Aho-Corasick tem uma performance similar à do KMP. Isso se deve ao fato de que os algoritmos têm complexidade semelhante, sendo a do KMP $O(m+n)$ e a do Aho-Corasick $O(m+n+z)$, sendo m, n e z, o tamanho do padrão, o tamanho do texto e o número de ocorrências do padrão no texto, respectivamente.

Com relação aos algoritmos de casamento aproximado, percebemos que o Wu-Manber performa melhor que o Sellers em todas as situações em que a distância de edição máxima é um valor pequeno. Isso vai de acordo com as complexidades temporais dos dois algoritmos, $O(nr)$ para o Wu-Manber e $O(nm)$ para o Sellers, onde n, m e r denotam o tamanho do texto, o tamanho do padrão e a distância de edição máxima, respectivamente. Porém o Wu-Manber tem uma limitação, podendo ser utilizado apenas com padrões de tamanho máximo menor ou igual a 64 (sessenta e quatro). Esta limitação pode ser retirada utilizando-se estruturas como *bitset* em C++, mas não sem prejudicar a eficiência do algoritmo.