Ferramenta PMT

1. Identificação

A equipe é composta por Bruno de Melo Costa (bmc4) e Renato Henrique Alpes Sampaio (rhas). Bruno implementou os algoritmos de busca exata e aproximada, parte da CLI e a função de leitura do texto. Renato implementou os scripts de teste e apresentação dos dados, parte da CLI, o algoritmo de escolha de algoritmo a partir da entrada e escreveu a documentação.

2. Implementação

2.1. Descrição da ferramenta

Dois tipos de busca foram implementados no código (Busca Exata e Busca Aproximada) e para cada um foram realizados testes para decidir quais seriam os melhores algoritmos, considerando performance e limite. Apenas o algorimo Aho-Corasick pode fazer a busca de múltiplos padrões simultaneamente, todos os outros fazem a busca pelos texto com um padrão por vez. Cada algoritmo possuem duas versões: uma para quando a flag "--count" estiver presente no comando; e outra versão para quando a flag "--count" não estiver presente. A única diferença é que quando o "--count" não está presente, ele retorna imediatamente quando encontra uma ocorrência, o que aumenta a performance da busca. A decisão de criar duas funções é para evitar que a verificação do "--count" em cada interação diminui-se a performance do código. Além disso, todas as verificações de ocorrência recebem o atributo *[[unlikely]]* que auxilia o código a encontrar uma melhor performance, visto que ocorrer uma ocorrência em cada interação é improvável, no entanto, o uso desse atributo aumenta a requisição do compilador para suportar c++20.

2.1.1. Alfabeto

O alfabeto consiste na tabela completa ASCII. Para a conversão, converte-se diretamente do *char* para o *uint8_t*, podendo utilizar o charactere como um número utilizando-se o melhor desempenho para a conversão (nativo).

2.1.2. Leitura do texto

Para a leitura do texto de entrada, para cada arquivo, lê-se uma linha por vez e faz a busca para tal linha.

2.1.3. Estratégia dos Algoritmos de Busca Exata

Para a busca exata, foram implementados os seguintes algoritmos: Aho-Corasick, Boyer Moore, Knuth Morris Pratt e Shift Or. Através de testes realizados detalhados posteriormente, checamos da seguinte forma qual algoritmo escolher:

- ➢ Se tiver mais de um padrão, escolhe-se Aho-Corasick;
- ➢ Senão, se o tamanho do padrão for menor que 6, escolhe-se Shift Or;
- ➢ Senão, escolhe-se o Boyer Moore.

2.1.4. Estratégia dos Algoritmos de Busca Aproximada

Para a busca aproximada, foram implementados os seguintes algoritmos: Ukkonen, Sellers e Wu Manber. Através de testes realizados detalhados posteriormente, checamos da seguinte forma qual algoritmo escolher:

- ➢ Se o tamanho do padrão for maior que 128, escolhe-se o Sellers;
- ➢ Senão, se o tamanho do padrão for maior que 12, escolhe-se o Wu Manber;
- ➢ Senão, escolhe-se o Ukkonen.

2.1.5. Detalhamento dos algoritmos

Detalharemos agora partes relativas dos algoritmos.

2.1.5.1. Aho-Corasick

Utiliza-se de dois *vectors* (um para falha e outro para contar a quantidade de soluções) e uma trie (para a construção de estados e para apontar estados seguintes depedendo da entrada). A trie utiliza-se de *vectors* dentro de um vetor, os vetores internos tem tamanho fixo (do tamanho do alfabeto) e o externo é variável para suportar a variância da quantidade de estados. Internamente, utiliza-se um *vector* de *unordered_set* para quantificar a quantidade de

ocorrências em cada estado, que posteriormente vai ser salvo em um *vector* de *unsigned*, em favor da performance.

2.1.5.2. Sellers

A geração das próximas colunas são feitos *in-place* para evitar o realocamento de memória.

2.1.5.3. Shift Or

Por construção, o Shift Or tem uma restrição de tamanho do padrão, que varia dependendo do compilador. Se o compilador for gnu, o seu limite é 128 caracteres, caso contrário, o seu limite é 64 caracteres. Porém, mesmo com um maior limite no gnu, a performance do tamanho dos padrões acima de 64 caracteres possui uma perfomance notavemente inferior para o mesmo tamanho de entrada, visto que ambos os casos, possuem um performance constante.

2.1.5.4. Ukkonen

No Ukkonen, utiliza-se uma trie no mesmo modelo transcrito no Aho-Corasick, mas, além disso, foi implementado um *vector* para saber se o estado é final. Para evitar que estados (colunas) iguais fossem repetidos na trie, utiliza-se um *unordered_map*, pois diferentemente do *map* a sua falta de ordenação permitiria uma melhor performance. Porém, para implementar o *unordered_map*, precisariamos implementar uma função de hash, pois a coluna é constituída por um *vector* de unsigned. Para calcular o hash, multiplica-se o valor de cada célula por um número constante pseudo-aleatório previamente processado.

2.1.5.5. Wu Manber

O Wu Manber, de forma resumida, utiliza-se das técnicas do Sellers e possui as limitações descritas no Shift Or.

3. Testes e resultados

Foram utilizados arquivos de texto em inglês (Arquivo de 1024MB da fonte mencionada na especificação do projeto e arquivo de 1300MB para testes de AI disponível em https://mystic.the-eye.eu/public/AI/pile/test.jsonl.zst) e um arquivo de 200MB de sequências de proteínas, retirado da fonte mencionada na especificação. As ferramentas utilizadas para comparação no benchmarking foram grep para casamento exato e agrep para casamento aproximado. Todos os testes foram realizados com a flag --count.

Os testes foram realizados em um computador rodando Manjaro Linux 5.10.105-1 com CPU Intel Core i5-8250U @ 1.60GHz, memória DDR4 2400 MHz de 8GiB e SSD ST500LM021 de 6Gb/s. O compilador utilizado foi o GCC 11.2.0-4

Todos os scripts shell e python utilizados nos testes estão disponíveis na pasta /test do projeto.

3.1 Casamento Exato:

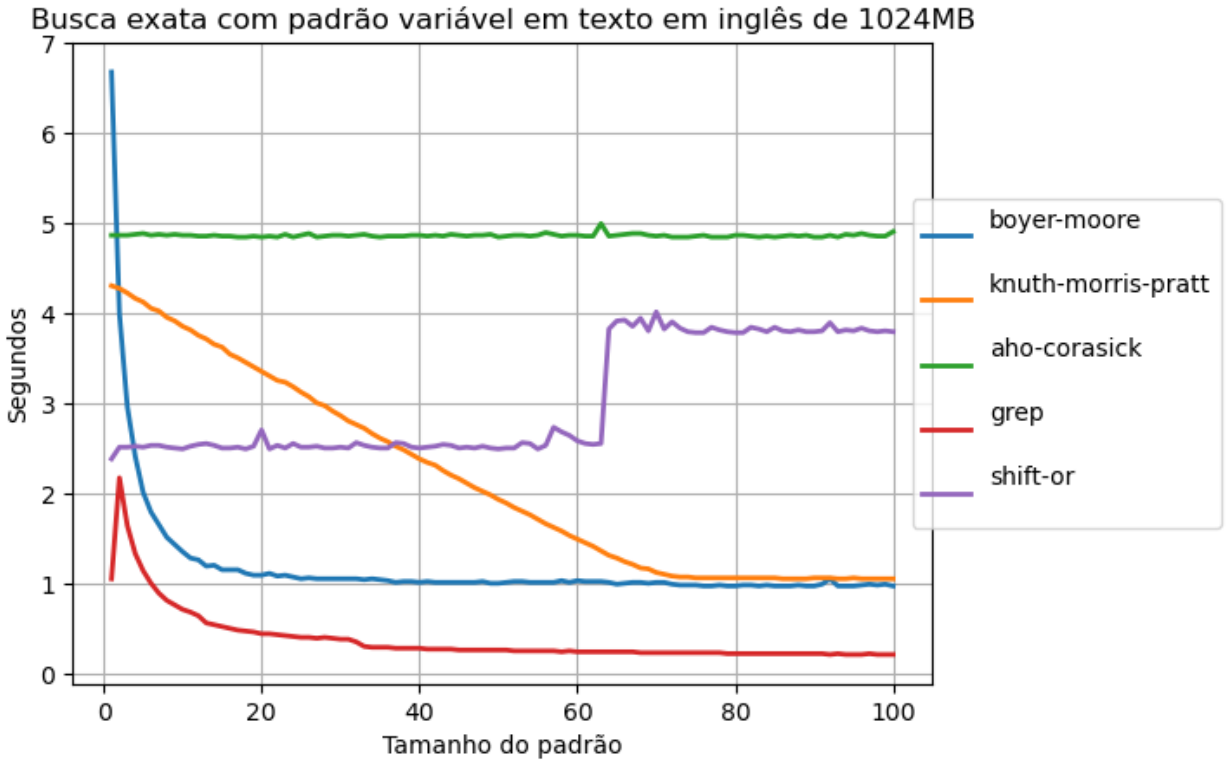
Foram realizadas duas séries de testes de casamento exato, uma com apenas um padrão por vez e uma com múltiplos padrões. Para o teste de padrão único, foi utilizado o texto em inglês de 1024MB e foram testados os algoritmos Boyer-Moore, KMP, Aho-Corasick e Shift-Or, bem como a ferramenta grep. Foram feitos 3 testes para cada tamanho de padrão entre 1 e 100 (inclusivo), tomando-se a média dos três testes como resultado final do tamanho. Para o teste de padrões múltiplos, foi utilizado o mesmo texto e foram testados os algoritmos Aho-Corasick e a ferramenta grep. Foram feitos 3 testes para cada quantidade de padrões (padrões de tamanho 5) entre 1 e 80 (inclusivo), tomando-se a média dos três testes como resultado final da quantidade.

3.2 Casamento Aproximado:

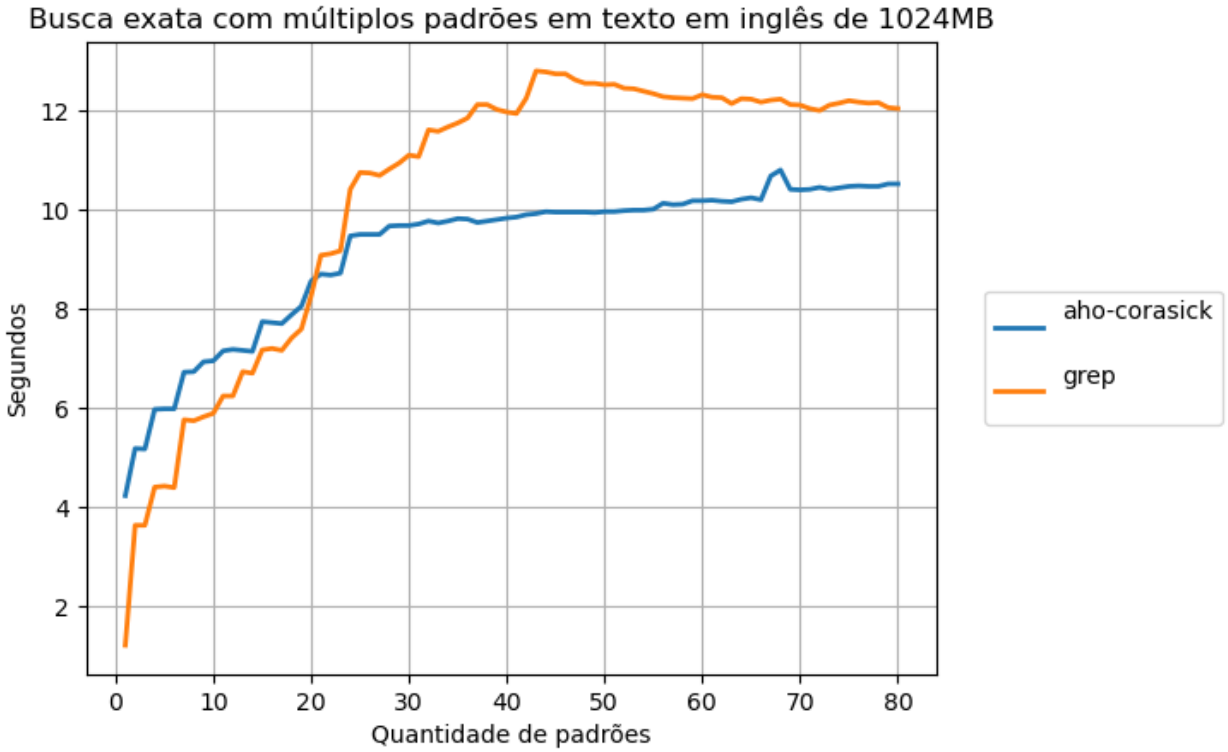
Para o casamento aproximado também foram realizadas duas séries de testes, uma com erro fixo e uma com erro variável. Para o teste de erro fixo foi utilizado o texto de proteínas truncado para 100MB e foram testados os algoritmos Sellers, Ukkonen e Wu-Manber, bem como a ferramenta agrep. Foi utilizado um erro fixo 5 e foram feitos 3 testes para cada tamanho de

padrão entre 6 e 66 (excetuando-se agrep, que só tem suporte até tamanho 32 e ukkonen, que se torna muito custoso depois do tamanho 18), tomando-se a média dos três testes como resultado final do tamanho. Para o teste de erro variável foi utiliazado o mesmo texto e foram testados os algoritmos Sellers, Ukkonen e Wu-Manber (agrep não permite erro acima de 8). O erro utilizado foi o tamanho do padrão - 1 e foram feitos 3 testes para cada tamanho de padrão entre 1 e 30 (excetuando-se Ukkonen), tomando-se a média dos três testes como resultado final do tamanho.

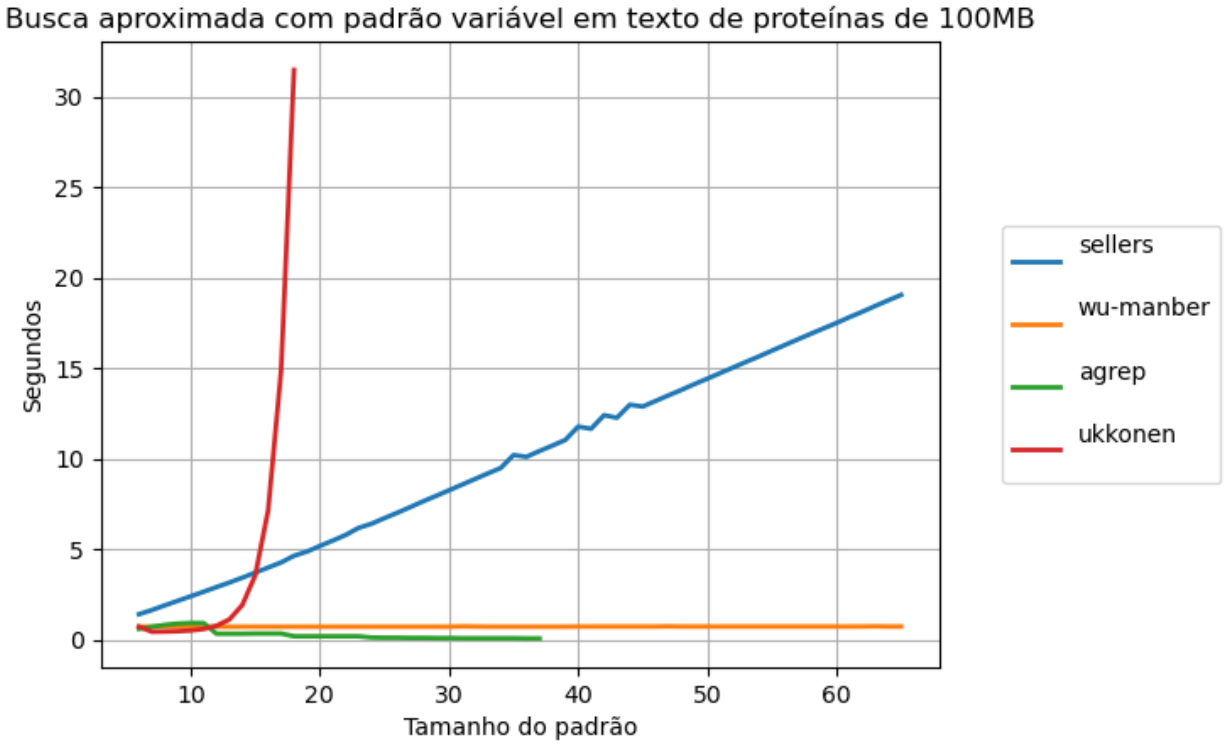
3.3 Resultados e discussão



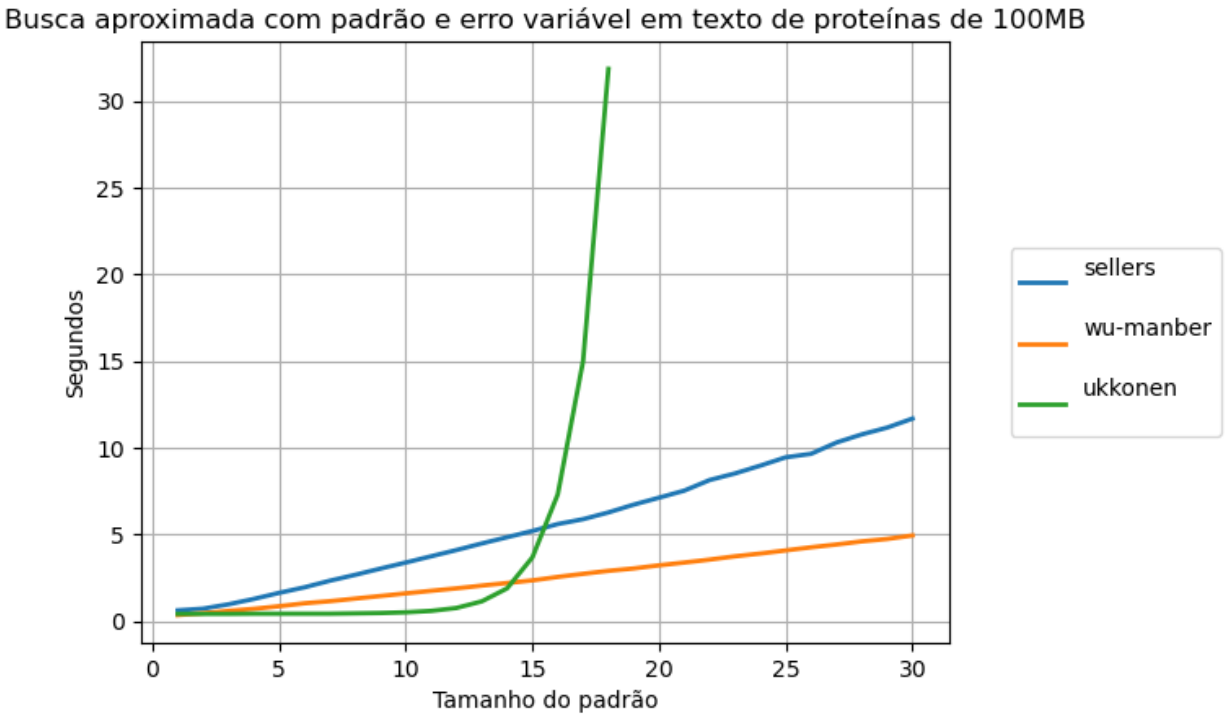
Nesse teste é possível observar que o Aho-Corasick não tem um desempenho tão bom para o caso em que só é utilizado um padrão, embora mantenha seu tempo constante. Também fica claro que o Shift-Or e até mesmo o KMP tem desempenho bom para padrões pequenos, só se tornando mais vantajoso o uso do Boyer-Moore para padrões com tamanho acima de 6. Destaca-se que a implementação do Boyer-Moore utilizada no grep tem desempenho melhor que a implementação utilizada no PMT. Além disso, no Shift-Or percebe-se a diferença de performance entre os tamanhos de padrão 64 e 65, isso se deve ao fato de ocorrer uma mudança de *int64_t* para *int128_t*, que mesmo sendo suportado, tem performance inferior.



Nesse teste fica clara a eficiência do Aho-Corasick para busca de múltiplos padrões, superando até mesmo a implementação do grep para mais de 20 padrões.



O teste de busca aproximada mostra que exceto para padrões muito pequenos (menores que 12) o tempo constante do Wu-Manber supera muito o tempo linear do Sellers e o exponencial do Ukkonen.



Embora o Wu-Manber não seja constante quando o erro é variável, os resultados ainda demonstram que esse algoritmo tem desempenho melhor que o Sellers e o Ukkonen para padrões com tamanho maior que 12.

4. Conclusões

Para busca exata, só vale a pena utilizar o Aho-Corasick quando é necessário buscar mais que três padrões, visto que é mais rápido rodar o Shift-Or para um padrão duas vezes do que rodar o Aho-Corasick para dois padrões (para padrões menores que 64). Para apenas um padrão o Boyer-Moore tem um desempenho muito superior para padrões de tamanho médio ou maior. Não foram encontrados casos de uso para o KMP, visto que mesmo na breve janela em que ele é mais eficiente que o Boyer-Moore, o Shift-Or tem desempenho muito melhor.

No caso da busca aproximada, só foi encontrado um caso de uso para o Sellers quando o tamanho do padrão é maior que 128, caso em que não é possível utilizar o Wu-Manber. Para tamanhos menores que 128 e maiores que 12, o tempo constante do Wu-Manber tem uma performace muito superior aos outros dois algoritmos, mesmo com erro variáve. O Ukkonen tem um caso de uso muito específico quando o tamanho do padrão é menor que 12, situação na qual ele ultrapassa a performance do Wu-Manber.