Universidade Federal de Pernambuco - UFPE
Centro de Informática - CIn
Bacharelado em Ciência da Computação

Processamento de Cadeias de Caracteres

Ferramenta PMT

Recife, Abril de 2022

Universidade Federal de Pernambuco -
UFPE Centro de Informática - CIn

Bacharelado em Ciência da Computação

Disciplina: Processamento de Cadeias de Caracteres - IF767
Professor: Paulo Fonseca
Período Letivo: 2021.2
Alunos: Amanda Nunes Silvestre Costa e Rodrigo Farias Rodrigues Lemos

Sumário

1. Identificação

A equipe é formada por Amanda Nunes Silvestre Costa (ansc) e Rodrigo Farias Rodrigues Lemos (rfrl).

A maior parte do código foi feita utilizando os conceitos de pair programming, fazendo com que todos tenham uma boa noção e contribuição de boa parte do código, segue a lista de responsabilidades que utilizamos na divisão.

1.1 Amanda Costa

- Script para testes automatizados e criação de gráficos.
- Preparação de testes.
- Aho Corasick
- Wu Manber

1.2 Rodrigo Lemos

- Arquivo main com a CLI.
- Arquivo makefile com sistema de compilação.
- KMP
- Ukkonen

2. Implementação

2.1 Algoritmos de casamento exato

Foram implementados nesta etapa os algoritmos Aho-Corasick e Knuth-Morris-Pratt (KMP).

2.1.1 Aho-Corasick

Desenvolvido em 1975 por Alfred V. Aho e Margaret J. Corasick, o Aho-Corasick é um algoritmo de busca exata de padrão e tem como especialidade a otimização em casos de busca de múltiplos padrões em um mesmo texto.

Para fazer a busca o algoritmo utiliza princípios de Máquina de estados finitos e além dos tradicionais caminhos similares a de uma Trie, inclui também conexões adicionais em casos de falha de padrão, otimizando os casos de erros e fazendo o algoritmo não precisar retroceder nem computar a mesma posição do texto mais de uma vez.

2.1.1 Knuth-Morris-Pratt (KMP)

Desenvolvido por James H. Morris em 1970, o Knuth-Morris–Pratt (Popularmente apelidado de KMP) é um algoritmo de busca exata otimizado para a busca de um único padrão em um texto base.

A complexidade total do algoritmo é O(n + m) onde "n" é a quantidade de caracteres do texto base e "m" o tamanho do padrão a ser procurado. Assim como o Aho-Corasick, o KMP também possui uma fase de pré-processamento onde o algoritmo criará um vetor (chamaremos de borda) que armazenará o maior prefixo do padrão que termina na posição atual.

Passo a passo do algoritmo.

1) **Criação do vetor de bordas.**

A borda de um valor representa o tamanho da maior cadeia, terminando na posição atual e acabando antes da origem que também seja um prefixo da cadeia original.

c[i]	a	b	a	a	b	a
border[i]	-1	-1	0	0	1	2

2) **Busca pelo padrão.**

Nessa etapa do algoritmo percorremos o texto original para enfim realizar a busca pelo padrão, usaremos o vetor de bordas como uma otimização, já que ele indicará a partir de que posição do padrão deveremos retomar a busca quando encontrarmos uma falha.

2.2 Algoritmos de casamento aproximado

Foram implementados nesta etapa os algoritmos Ukkonen e Wu Manber.

2.2.1 Ukkonen

Proposto por Esko Ukkonen em 1995, o algoritmo homônimo ao criador consiste em um algoritmo online para criação de uma árvore de sufixos, utilizado aqui para busca aproximada de um dado padrão, com erro máximo definido.

O grande diferencial vem da construção da árvore, que começa implicitamente contendo o primeiro caractere do texto e, sequencialmente, são adicionados os sucessivos. Em suma, é criada uma árvore de sufixos implícita para cada prefixo do texto, onde a árvore do prefixo atual (T[i+1]) é construída a partir da árvore do prefixo anterior (T[i]).

A complexidade deste algoritmo é linear no tamanho do texto, mas o pré-processamento é exponencial em *r* (erro máximo permitido).

2.2.2 Wu-Manber

Desenvolvido por Sun Wu e Udi Manber em 1992, o algoritmo Wu-manber utiliza operações binárias para casamento aproximado de padrões. Sua lógica consiste na utilização de máscaras binárias:

(1) Entre o alfabeto e o padrão, e

(2) Entre o padrão e o texto onde para cada posição do texto terá um conjunto de máscaras binárias definido com base no conjunto da posição anterior, representando o reconhecimento dos prefixos do padrão terminando naquela posição, com erro de 0 até erro máximo definido.

Sua complexidade fica em O(n*r), onde n é o número de caracteres no texto base, e r o erro máximo permitido.

2.3 Heurística de seleção de algoritmos

Utilizando os testes a serem abordados em seguida como base, esses foram os critérios para a escolha do algoritmo nos casos onde o usuário não optou por nenhum específico.

Em casos onde o erro máximo seja zero:

- Escolhemos o KMP em casos de busca por um único padrão
- Escolhemos o Aho Corasick em casos de múltiplos padrões.

Em casos onde o erro máximo seja maior que zero:

- Utilizamos o Ukkonen se a margem de erro for menor que 6.
- Utilizamos o Wu-Manber se a margem de erro for maior ou igual a 6

2.4 Detalhes de implementação relevantes

Em nossa implementação todo o processo de processamento dos padrões a serem procurados é feito antes da leitura dos textos independentemente do algoritmo utilizado, dessa forma criamos um pipeline que é otimizado e padronizado, facilitando a inserção de novos métodos de busca no futuro.

Nosso sistema foi feito otimizando a busca em textos com múltiplas linhas, o resultado final das ocorrências representará a quantidade de linhas contendo o padrão e não de fato o número de vezes que aquele padrão existe no texto base.

- Opções de linha de comando:
    - -a —algotithm *A*: realiza busca com algoritmo selecionado A, opção:
        - AhoCorasick;
        - BruteForce;

        - KMP;
        - Ukkonen; e
        - WuManber.
    - -c —count-only: imprime apenas a quantidade total de ocorrências do(s) padrões contidas no(s) arquivo(s) de texto.
    - -e —edit *r*: localizar todas as ocorrências aproximadas do padrão a uma distância de edição máxima r.
    - -p —pattern-file: arquivo contendo o padrão de busca desejado, aceita múltiplos padrões de uma vez, sendo um por linha do arquivo.
    - -h –help: retorna um guia com os possíveis comandos.
- Alfabeto de entrada
    - *getopt* utilizado para leitura dos argumentos do programa
    - O programa foi feito para funcionar utilizando como base textos com os padrões de caracteres ASCII.

3. Testes e Resultados

Os testes foram realizados através de um script python, utilizando a ferramenta GNU time para obtenção do tempo de execução. Para critério de comparação, foi utilizada a opção “count-only”, retornando apenas a quantidade total de ocorrências em cada algoritmo. Os resultados encontrados foram armazenados em uma tabela a fim de comparar o desempenho da execução de cada algoritmo quando enfrentando um determinado dataset junto com um padrão.

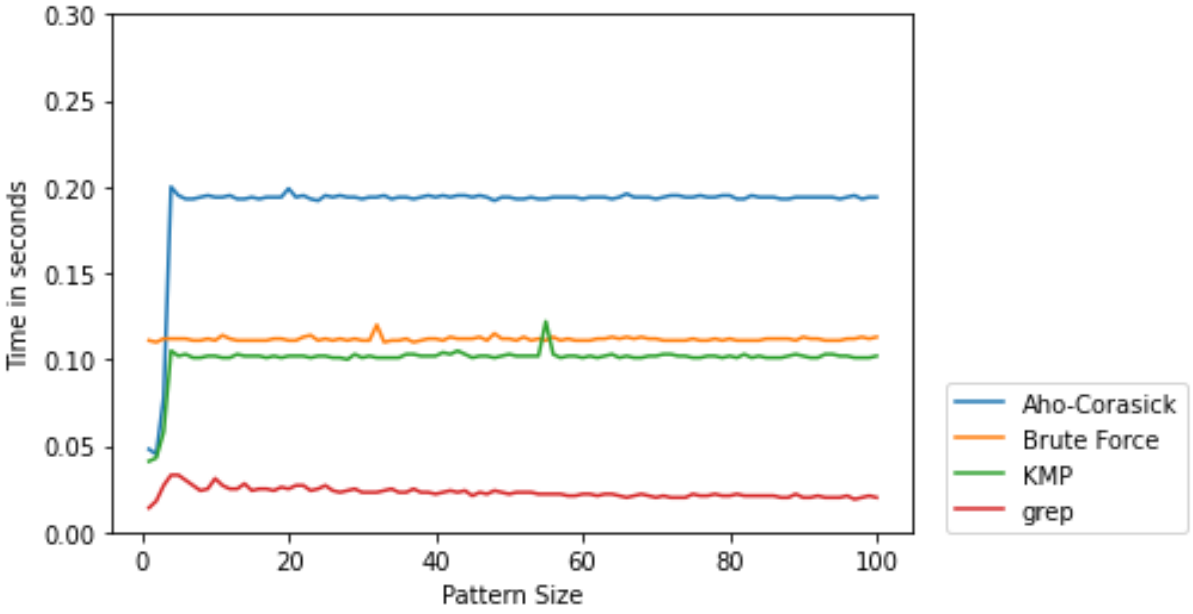
No caso do casamento exato, foi feito um comparativo de desempenho e corretude com a ferramenta *grep*. Já para o casamento aproximado, o intuito era que este comparativo fosse realizado com a ferramenta *agrep*, mas devido ao tamanho dos arquivos e seus tipos, isso não foi possível em todos os casos.

Os arquivos utilizados são de diferentes tamanhos e estão disponíveis no Pizza&Chili, já os padrões, foram obtidos de maneira aleatória nos arquivos. Os testes foram realizados em um MacBook Pro modelo 2019 com 8 GB de memória e um processador Intel Core i5 1,4 GHz.

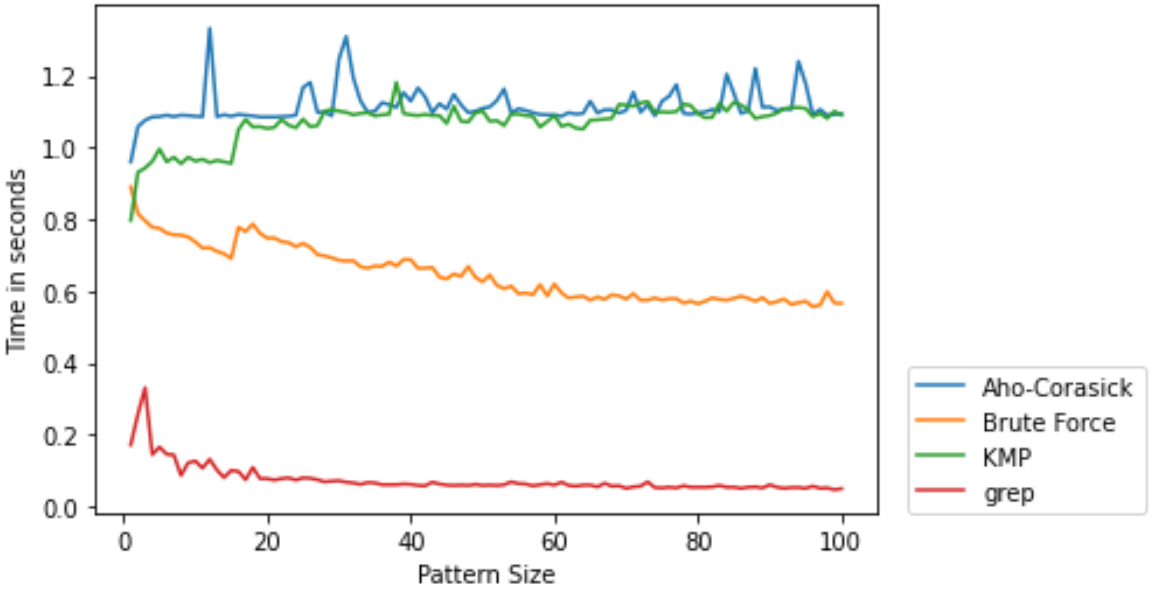
3.1 Algoritmos de Busca Exata

A fim de observar a aplicabilidade de cada algoritmo e entender quais os algoritmos ideais para determinadas situações, foram feitos testes utilizando os diferentes arquivos previamente mencionados e padrões (cada padrão gerado aleatoriamente a partir do seu próprio arquivo), variando o tamanho do prefixo do padrão de 1 até 100. Os critérios aqui utilizados foram tempo de desempenho por tamanho do padrão.

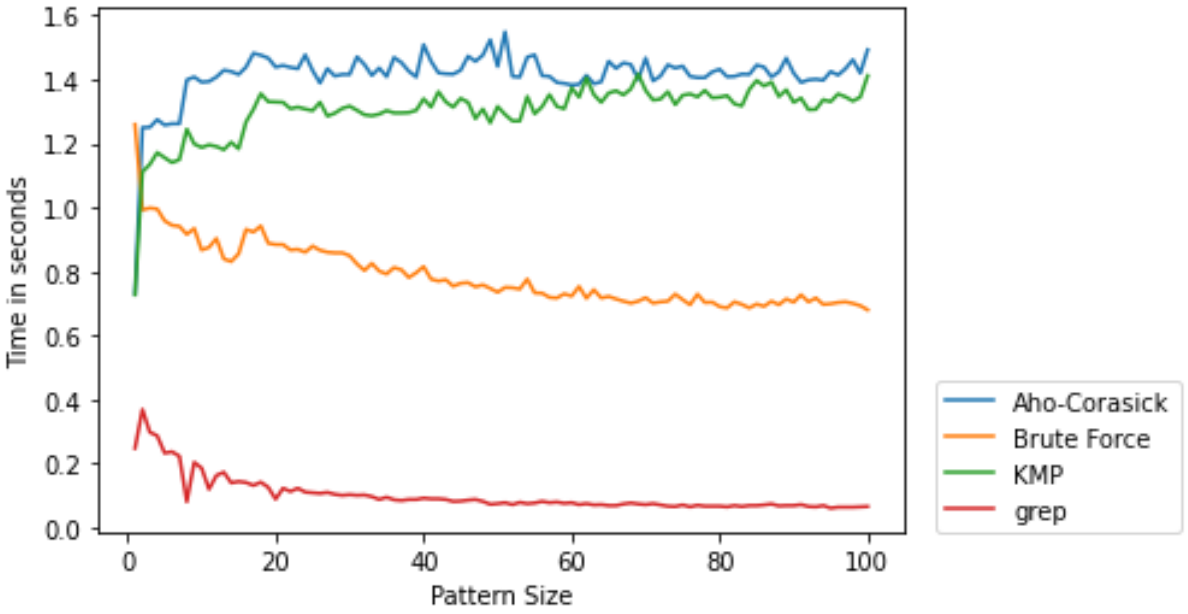
Teste utilizando arquivo *pitches* (56 mb)



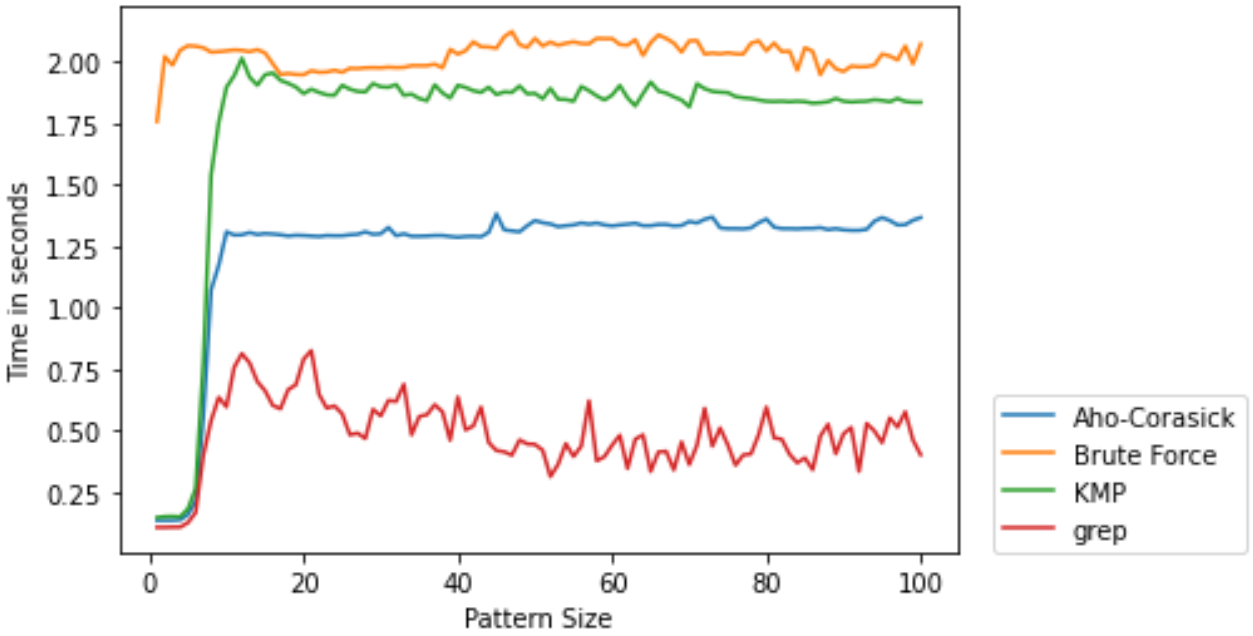
Teste utilizando arquivo sources (211 mb)



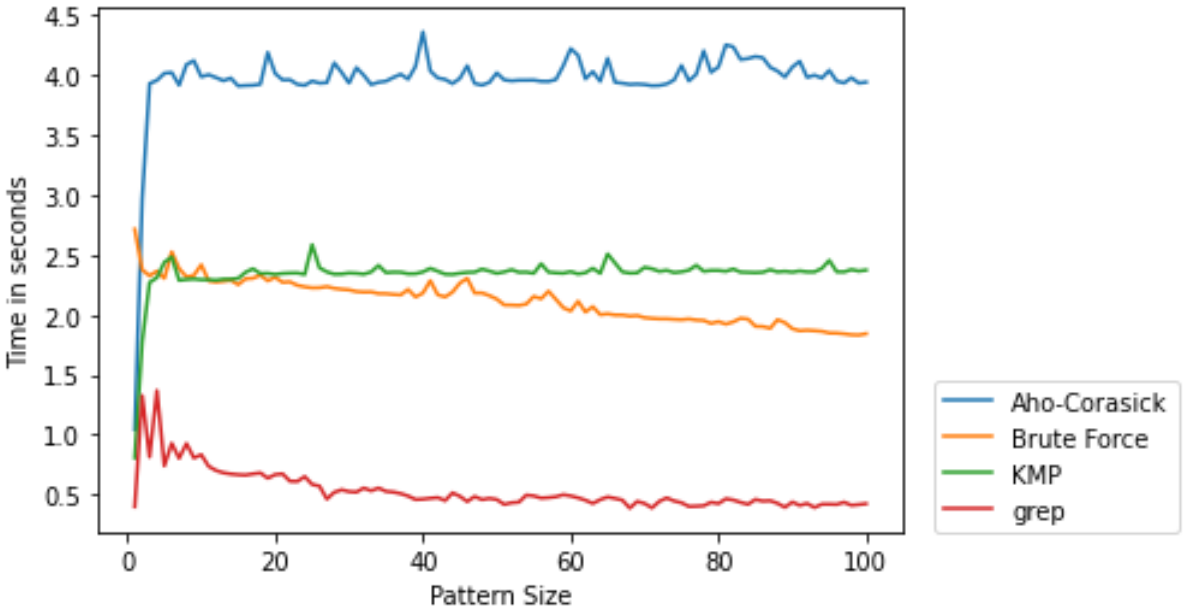
Teste utilizando arquivo dblp.xml (296 mb)



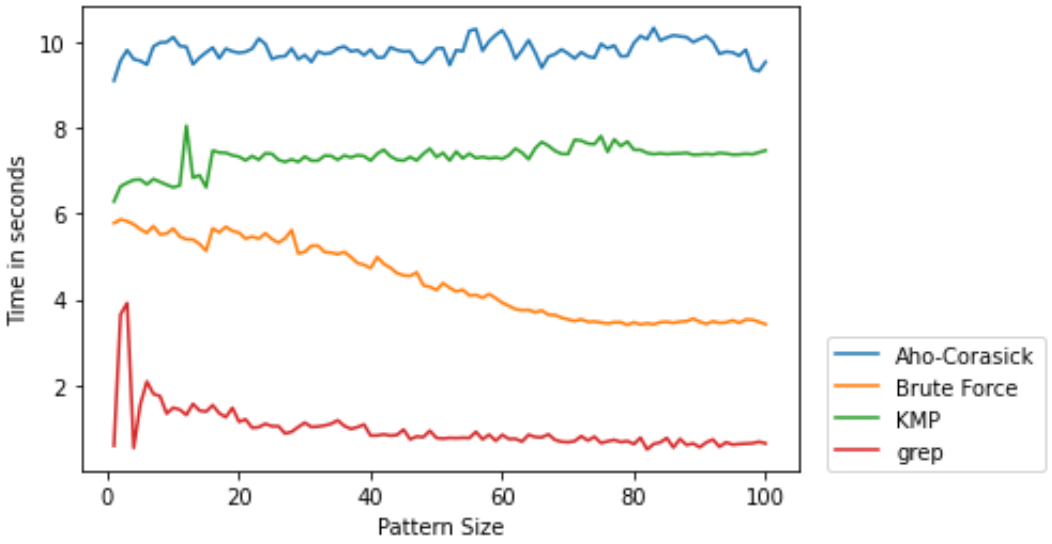
Teste utilizando arquivo DNA (404 mb)



Teste utilizando arquivo proteins (1.2 gb)



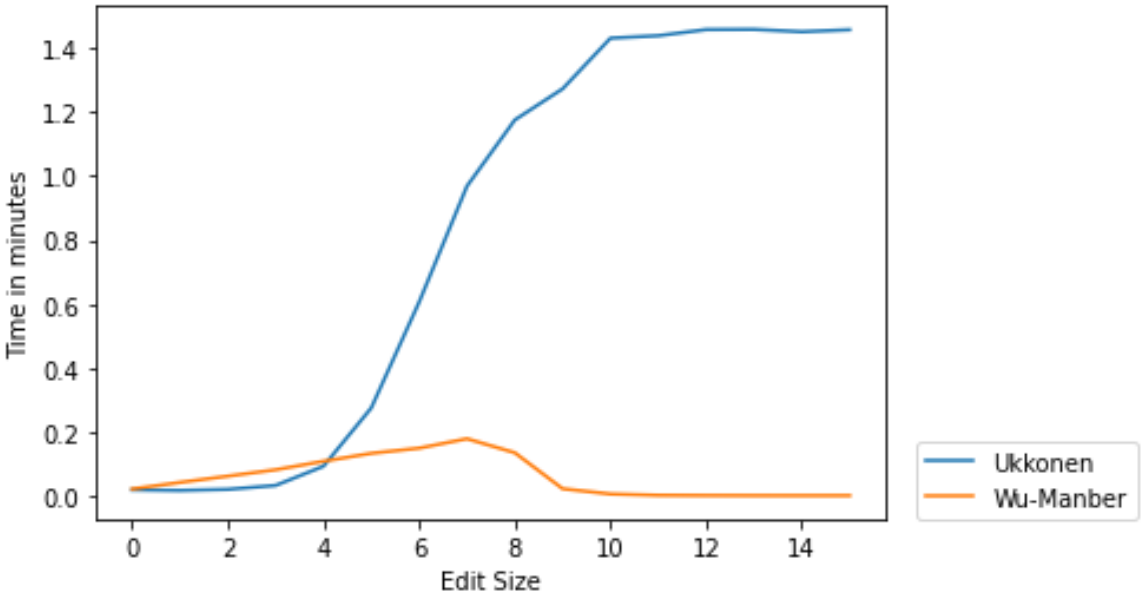
Teste utilizando arquivo english (2.2 gb)



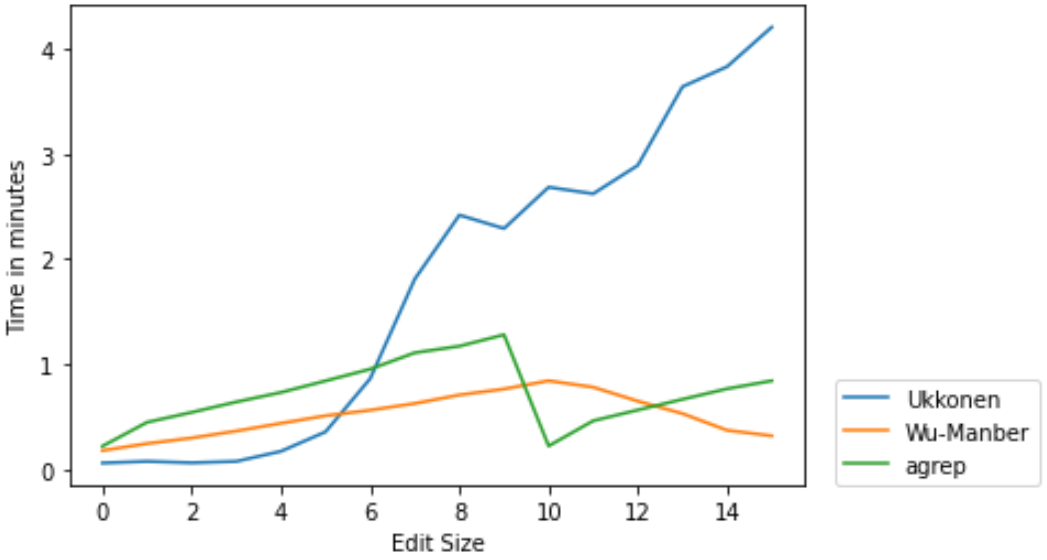
3.1.2 Algoritmos de Busca Aproximada

Com o mesmo objetivo dos testes do tópico 3.1.2, os testes desta etapa foram realizados com padrões de tamanho fixo *m* (16 caracteres), alterando *r* (erro máximo permitido) de 0 até *m*-1. O gráfico do agrep foi omitido dos dois últimos gráficos, visto que o sistema não suporta alguns dos caracteres utilizados no arquivo a ser testado.

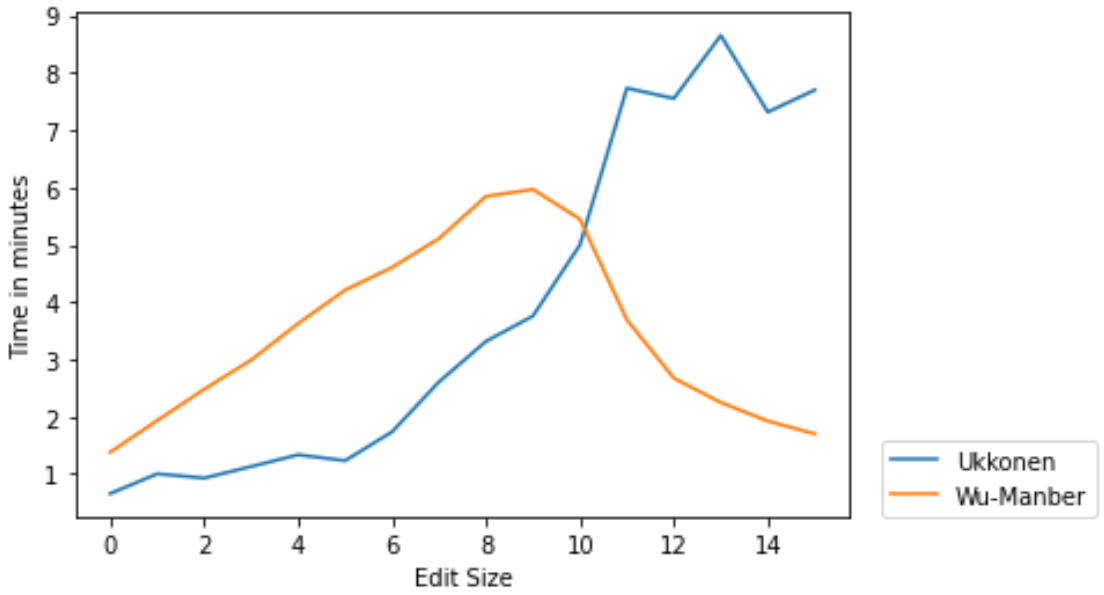
Teste utilizando arquivo pitches (56 mb)



Teste utilizando arquivo sources (211 mb)



Teste utilizando arquivo english (2.2 gb)



4. Conclusões

Em conclusão nós vimos uma aproximação bastante forte entre os algoritmos de busca exata, tanto o KMP quanto o Aho Corasick se saíram bem no decorrer dos testes, foi também feita a inclusão de um algoritmo de força bruta para comparação. É notório também como o arquivo de entrada pode afetar a performance dos algoritmos, a depender dos caracteres, tamanho e formato dos textos.

Nos casos de busca aproximada foi testado principalmente o quanto o tempo de execução varia a depender do tamanho da distância de edição enviada. A complexidade exponencial do Ukkonen se torna um fator determinante para fazer nosso projeto optar pelo Wu-Manber em casos de maior erro enviado.

Detectamos diversas oportunidades de otimização enquanto desenvolvemos os algoritmos, como por exemplo modificar o uso de vector por array, e tirar proveito de ponteiros e memsets. Se fizermos isso podemos otimizar mais ainda os algoritmos implementados e chegar ainda mais próximo do tempo de execução do grep. Fica também em aberto a possibilidade de adicionarmos mais dos algoritmos vistos em sala de aula no futuro, fizemos nosso código de uma forma que seja simples e intuitiva essa inserção.