Ferramenta PMT

1. Identificação

A equipe é formada pelos alunos Mário Victor Gomes de Matos Bezerra e Pedro Kempter Brant. Mário implementou os algoritmos Aho Corasick, Shift Or, Wu Manber e a interface, e Pedro implementou os algoritmos KMP e Sellers.

2. Implementação

Detalhes de Implementação

Todos os algoritmos foram divididos em três partes: um construtor, que pré processa tudo que for necessário; uma função *count,* que conta o número total de ocorrências de um padrão em uma dada *string*; e, por fim, uma função *exists*, que determina se um padrão ocorre em uma dada *string.*

Para que seja possível processar arquivos de texto de uma tamanho considerável, o texto é processado linha a linha, logo as funções recebem como entrada uma referência de *string* de cada linha.

Representação alfabeto

O alfabeto considerado para este projeto foi o *extended ASCII,* já que é o suficiente para o escopo desse projeto.

Aho Corasick

Sua implementação consiste em: um construtor, que prepara a tabela de transição (*goto*), a de *fail* e armazena as ocorrências de cada estado; e duas funções, *count* e *exists*, que são responsáveis por percorrer a máquina de estados, representada pela tabela de transição, em procura de ocorrências.

Inicialmente, esse algoritmo começou fazendo uso de *unordered_map* para armazenar a tabela de transição, porém, após fazer *profiling* da aplicação, foi detectado que o *hash map* estava consumindo muito tempo de execução. Logo, começou-se a usar *vector* para representar a tabela de transição e *fail*, resultando

em uma melhora significativa de desempenho, caindo de, em média, 3 minutos para 12 segundos.

Shift Or

Sua implementação consiste em: um construtor, que prepara todas as máscaras binárias de todos os caracteres do alfabeto; e duas funções, *count* e *exists*, que são responsáveis por percorrer o texto, aplicando as máscaras dado o caracter lido. Além disso, o algoritmo só aceita padrões de até 64 caracteres, devido a estrutura de dados usada, *uint64_t*.

Foi feita uma tentativa de implementar a forma alternativa desse algoritmo, Shift And, porem foi notado que ele possui um detalhe que o faz rodar uma quantidade maior de comandos, mais especificamente, a checagem de um *if*. Enquanto que o Shift Or roda essa checagem *tamanho_do_padrão* vezes, o Shift And roda essa checagem *tamanho_do_texto* vezes. Dessa forma, foi-se decidido usar a implementação do Shift Or.

Wu Manber

Sua implementação consiste em: um construtor, que, similarmente ao algoritmo Shift Or, prepara todas as máscaras binárias de todos os caracteres do alfabeto; e duas funções, *count* e *exists*, que são responsáveis por percorrer o texto e procurar as ocorrências.

Inicialmente, esse algoritmo fazia uso de uma matriz, inicialmente vazia, que era preenchida com o comando *emplace*. Porém, após fazer *profiling* da aplicação, percebeu-se que a alocação de memória estava tomando tempo demasiado. Dessa forma, começou-se a declarar um *vector* de tamanho dois, que, ao invés de alocar memória a quantidade de caracteres do texto, só foi necessário alocar a memória uma única vez.

KMP

Sua implementação consiste em: 4 funções, *borders*, calcula um array de saltos possíveis (através do cálculo das bordas), *kmp,* compara o padrão com o texto fazendo saltos a partir das bordas pré calculadas.Além das funções*, count* e

exists que são responsáveis por retornar o número de ocorrências e se o padrão ocorre, respectivamente .

Sellers

Sua implementação consiste em: três funções, *sellers,* responsável por calcular a distância de Levenshtein (casamento aproximado) e retornar a posição das ocorrências.Além das funções*, count* e *exists* que são responsáveis por retornar o número de ocorrências e se o padrão ocorre, respectivamente .

Inicialmente, esse algoritmo foi implementado montando uma matriz de array n x m para calcular a distância. Porém, para minimizar o tempo e a memória gastos, a função sellers foi otimizada para só guardar a linha anterior e a atual com um uso de um vector 2 x m.

3. Testes

Os testes foram divididos em 2 módulos: busca exata, o qual varia-se o tamanho dos padrões de entrada e do texto; e busca aproximada, o qual varia-se o tamanho do texto e do distância de edição (0, 1 e 2).

Quanto aos arquivos, foram utilizados dois conjuntos de arquivos, que variam de tamanho (128MB, 256MB, 512MB, 1024MB e 2048MB) e conteúdo (xml da Wikipédia e DNA gerado aleatoriamente).

Enquanto que se usou o *grep* para comparação de eficiência dos algoritmos de busca exata, usou-se o *agrep* para os algoritmos de busca aproximada.

3.0 Especificação do Computador

3.1 Busca Exata

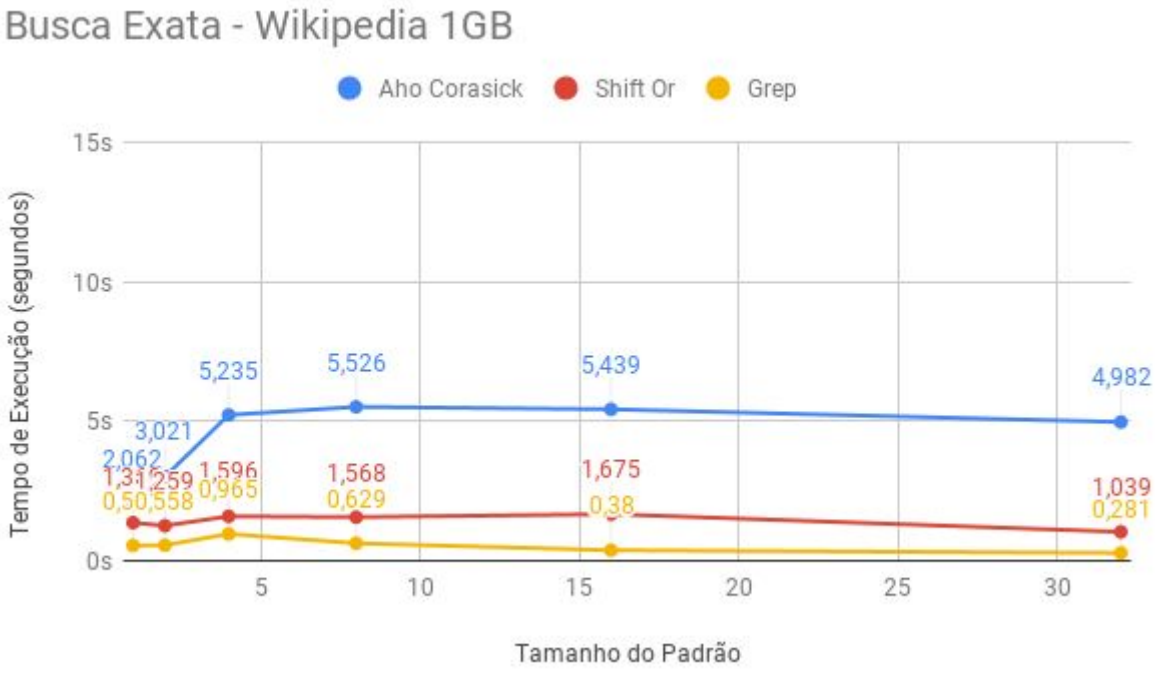
Neste módulo foram feitos diversos testes, enquanto se variava tamanho do arquivo e do padrão, os quais eram:

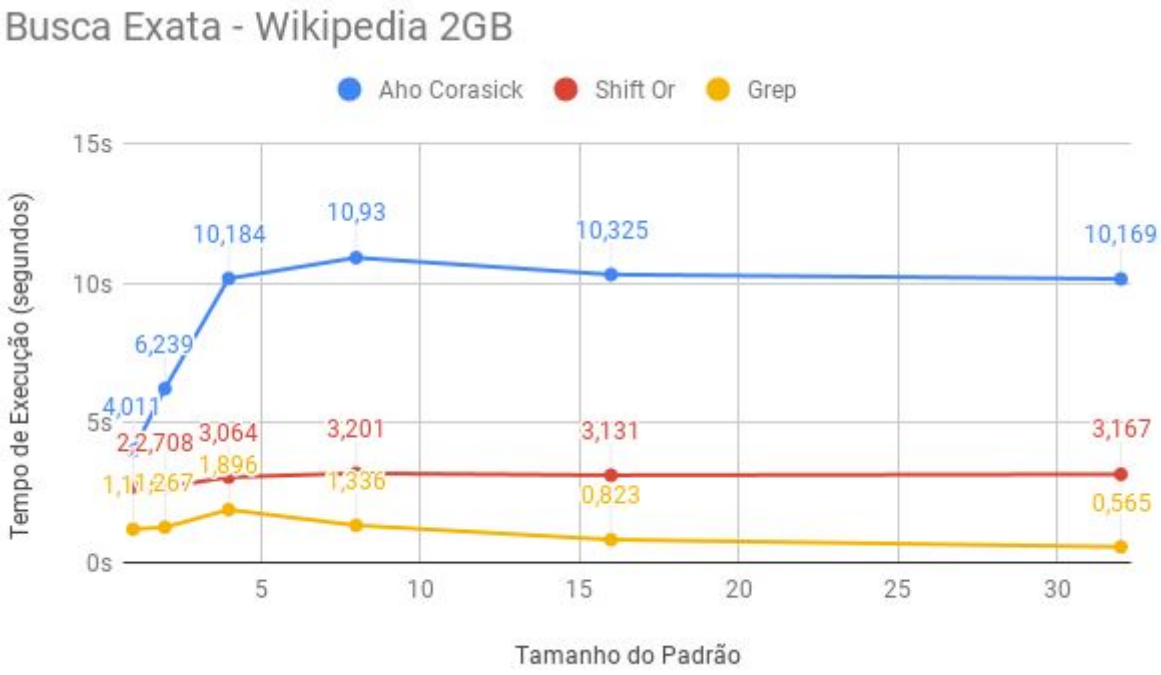
- a
- th
- does
- revision
- misunderstanding
- January]], [[February]],[[March

Os padrões foram escolhidos de acordo com a frequência com que eles aparecem nos textos, além de variar de tamanho entre 1, 2, 4, 8, 16 e 32.

Busca Exata - Wikipedia 256MB

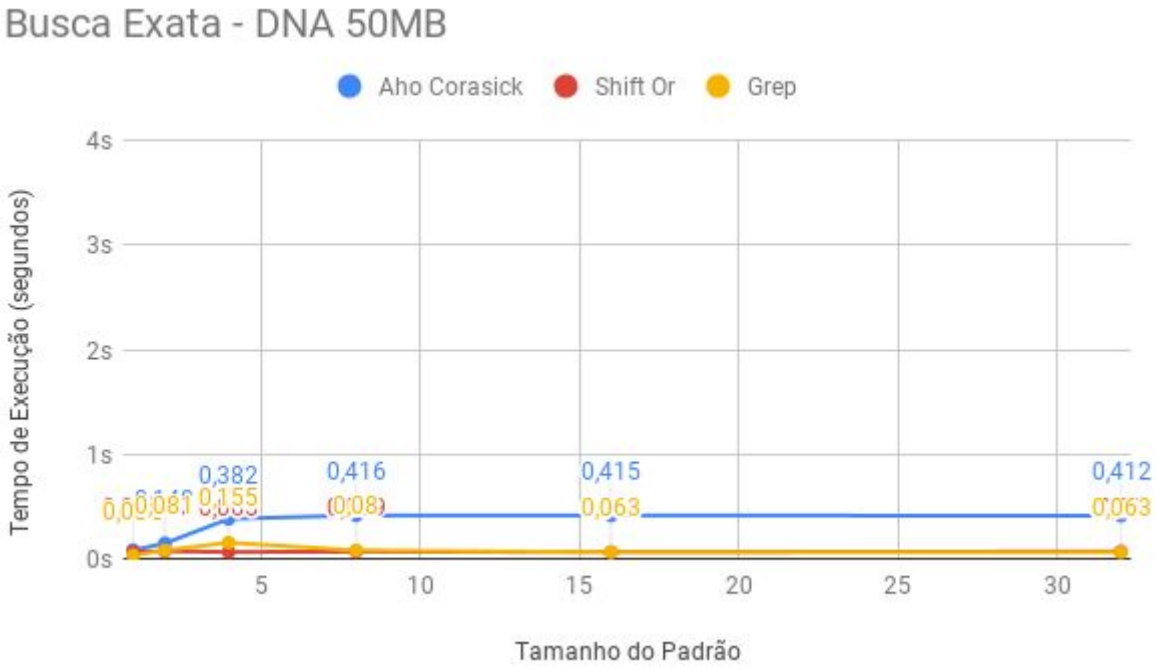
Busca Exata - Wikipedia 512MB

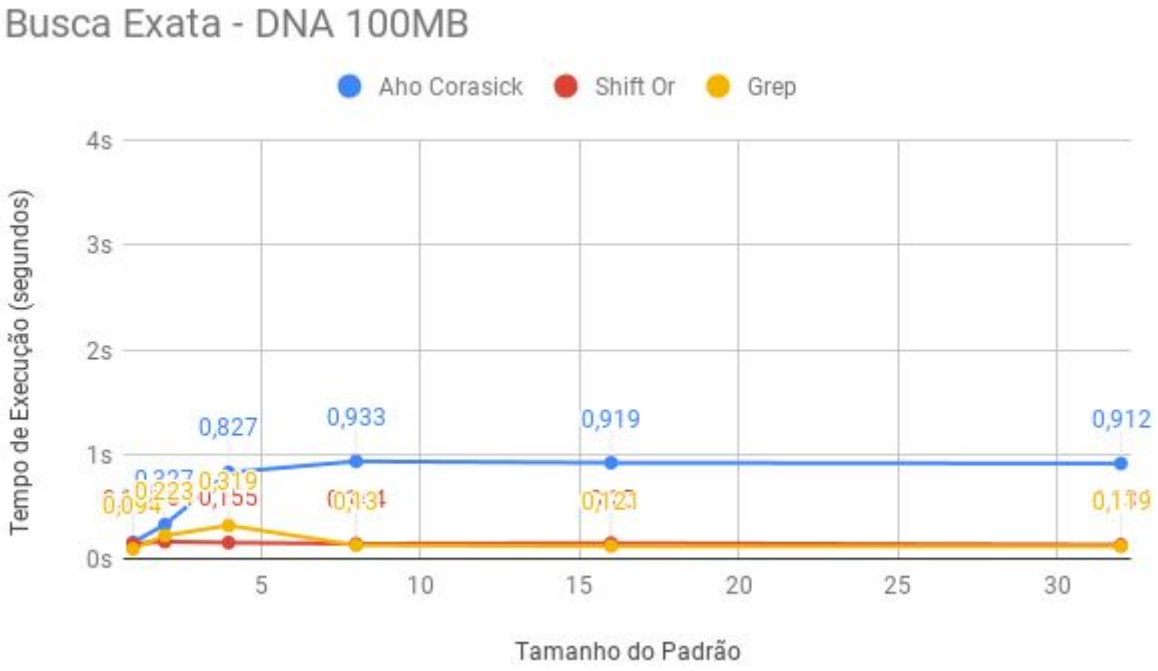


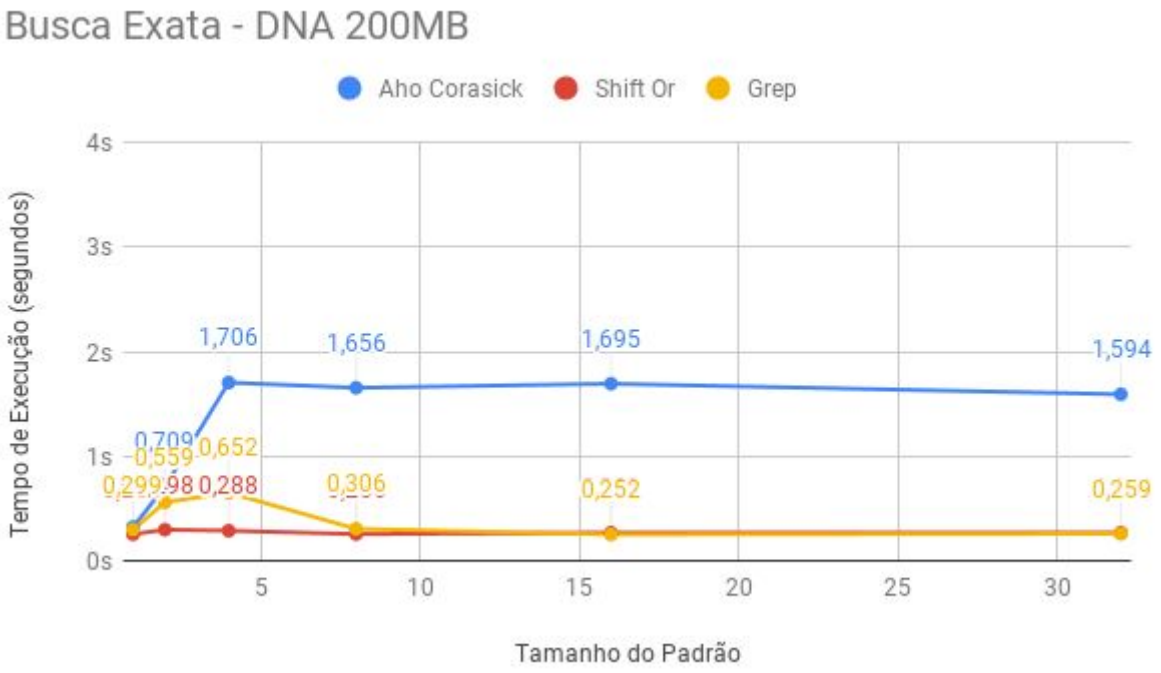


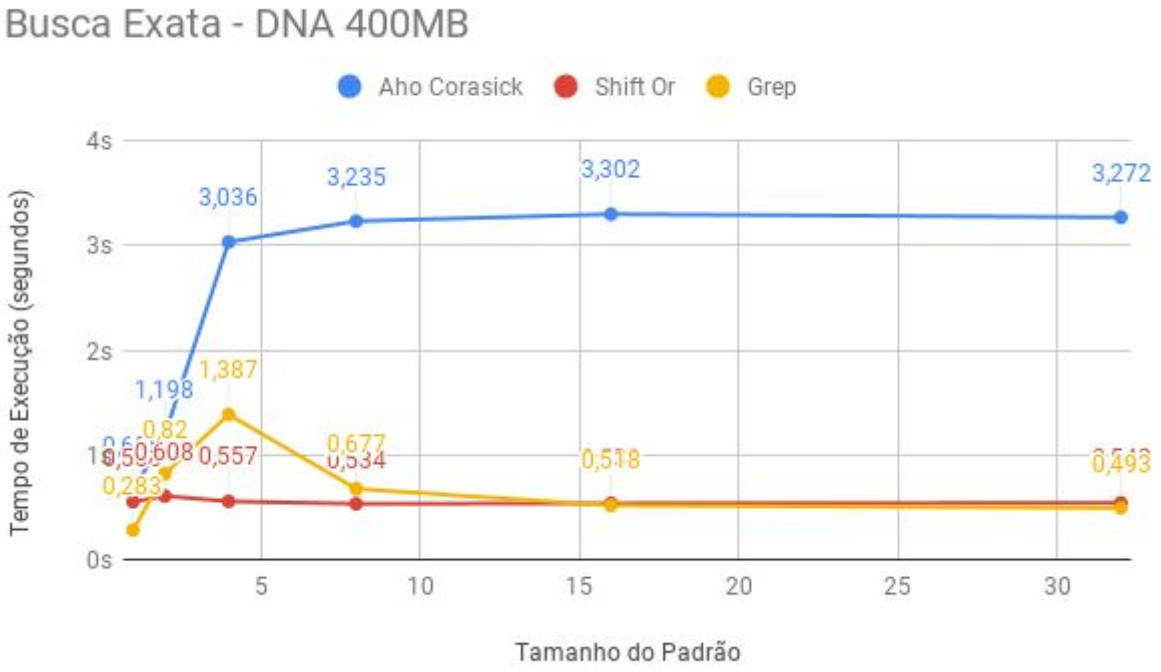
Visto os resultados, podemos observar que o comportamento entre os diferentes tamanhos de arquivo se mantém para os algoritmos, sendo o mais eficiente o Grep. O Shift Or se mostra mais eficiente que o Aho Corasick, não se distanciando muito do Grep.

Também foi possível notar que a partir do padrão de 4 caracteres, o Aho Corasick se demonstrou mais estável.





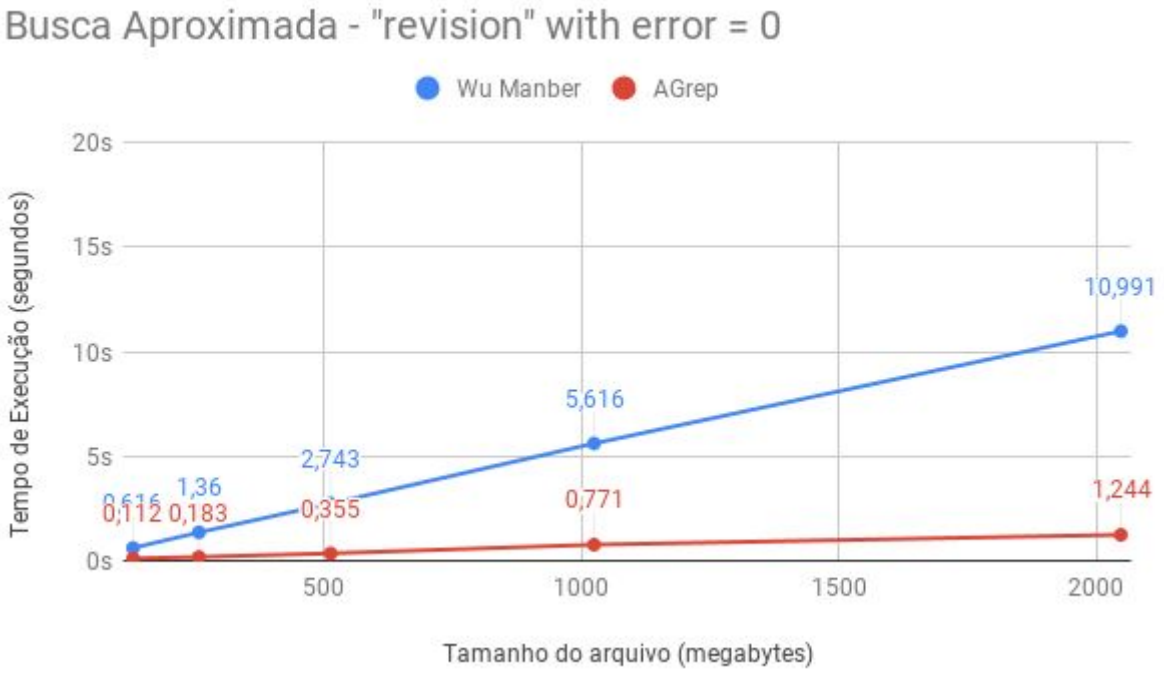


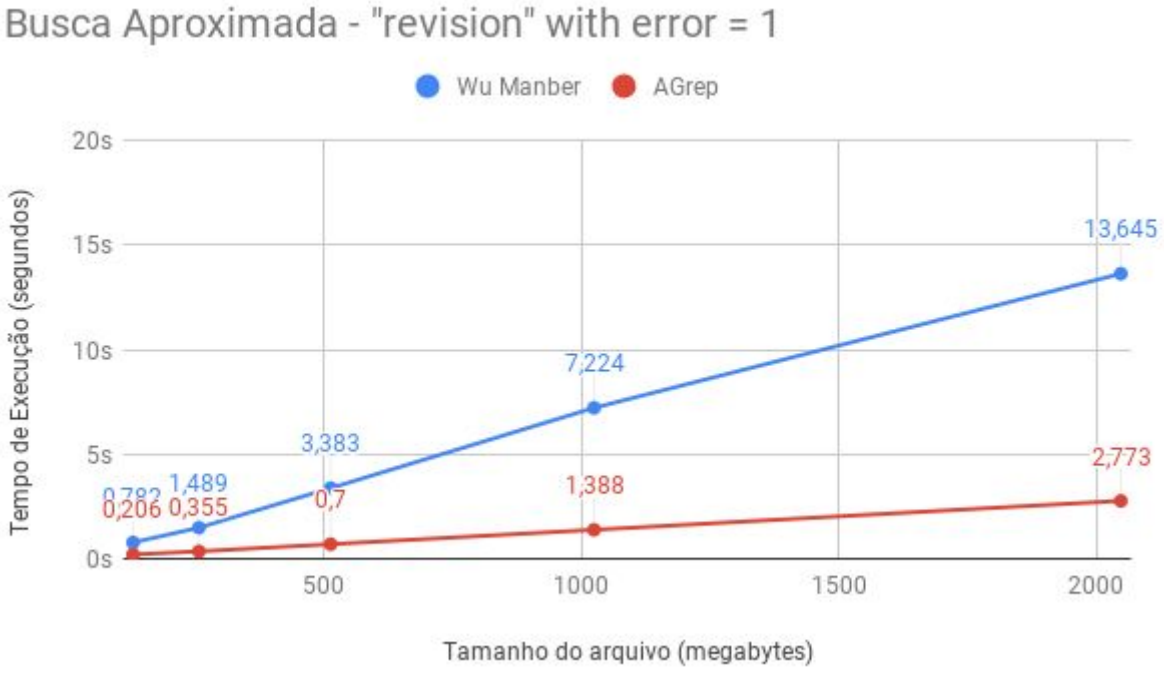


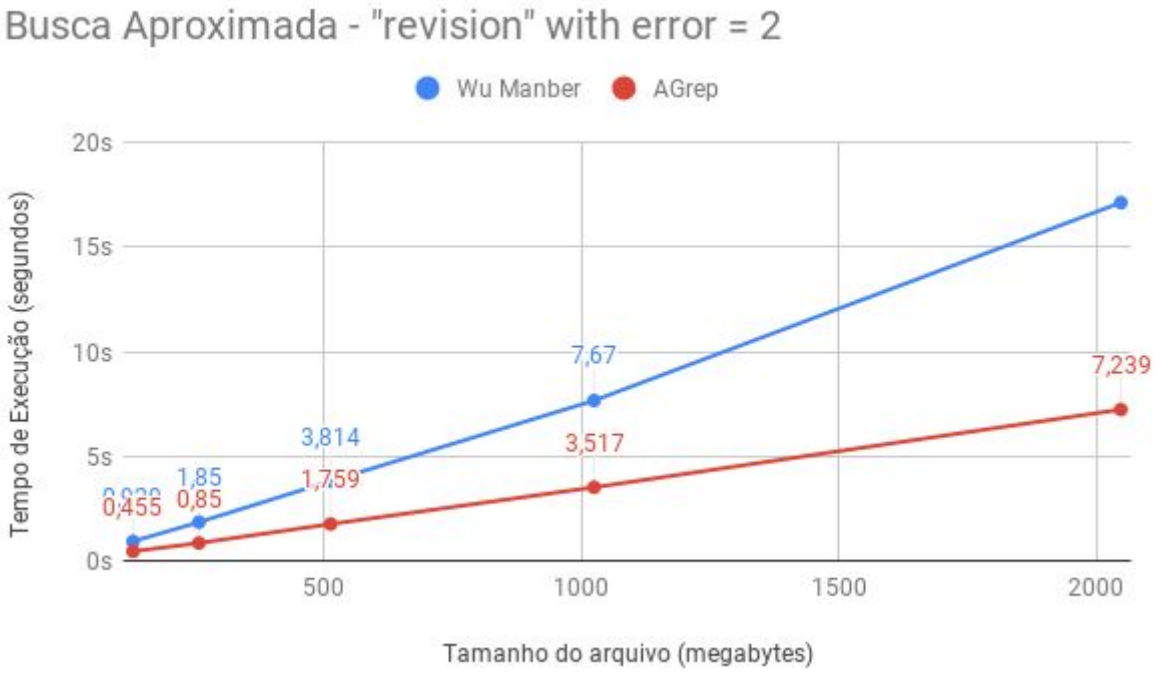
Visto os resultados, podemos perceber que o Shift Or se mostrou mais estável que o Grep e por vezes superando em termo de eficiência. Já o Aho Corasick manteve seu comportamento do último conjunto de teste, mostrando-se mais lento que os outros algoritmos. Também se mostrando mais estável a partir do

padrão de tamanho 4, já *grep* se mostrou mais estável após o padrão de tamanho 8.

3.2 Busca Aproximada







Visto os resultados, pode-se notar que o comportamento de crescimento do Wu Manber é bem similar ao do *agrep,* diferenciando-se pela eficiência. Note que o comportamento de ambos se mantém para distância de edição diferentes. Sendo mais custoso o processamento a medida que o erro aceitado aumenta.

4 Conclusão

A análise dos gráficos gerados pelos testes nos permite tirar algumas conclusões sobre os algoritmos em questão.

Todas as implementações se mostraram com um desempenho médio inferior ao apresentado pelo grep, porém em alguns casos o Shift Or se mostrou mais eficiente. Já o Aho Corasick teria vantagem ao analisar múltiplos padrões simultaneamente.

O Wu Manber se mostrou menos performático que o agrep, porém com uma curva de crescimento similar e com desempenho não tão discrepante.