Pattern Matching Tool

Arthur Costa e Lucas Rufino

21 de Outubro de 2018

1 Identificação

Equipe composta pelos alunos Arthur Latache Pimentel Gesteira Costa (alpgc)
Lucas Alves Rufino (lar)

1.1 Separação de tarefas

As atividades foram separadas da sequinte maneira:

1.1.1 Arthur Costa

- Interface de comando
- Leitura de arquivo
- Make file
- Aho Corasik
- Ukkonen

1.1.2 Lucas Rufino

- Interface de comando (opção de arquivo de padrões)
- Adaptação de leitura de arquivos
- Procedimentos de testes de algoritmos
- Shift Or
- Wu Manber

2 Implementação

2.1 Algoritmos de casamento exato

Foram implementados dois algoritmos de casamento exato, o ShiftOr e o Aho-Corasik. O ShiftOr em geral limita o tamanho do padrão pelo tamanho de um inteiro na maquina, no entanto usamos um vetor de inteiros, o que ajuda a ter padrões maior, mas deixa o algoritmo mais lento. O Aho Corasic é um algoritmo que trabalha com varios padrões, e por isso tende a ter vantagem quando multiplos padrões são utilizados.

2.2 Algoritmos de casamento aproximado

Implementamos o Ukkonen e o Wu Manber para o casamento aproximado. O Ukkonen tende a ser mais rapido com distancias maiores, enquanto o Wu Manber fica proporcionalmente mais lento com distancias maiores. Utilizamos o vetor de inteiros no ukkonen para obter velocidade no ukkonen tambem.

2.3 Heurística de seleção de algoritmo

Como o Aho Corasick ficou melhor que todos os outros em performance por muito, ele é a escolha padrão caso seja casamento exato. No caso de matching aproximado o Ukkonen sai constantemente melhor para casos onde a distância de edição do error é menor ou igual a 1/3 do tamanho do padrão, nesse caso o Ukkonem é escolhido, quando a distância de edicão for maior que 1/3 que o tamanho do padrão, o algoritmo do Wu-Manber é utilizado. Essa foi a heuristica adotada para seleção no caso de não citação do algoritmo.

2.4 Detalhes de implementação relevantes

O leitor de entrada é online, armazenando apenas a linha atual na memoria, o programa foi pensando de forma a ter o minimo de OO, mas para manter o codigo o mais limpo possivel criamos uma struct para cada algoritmo de busca implementando uma superclasse abstrata. Usamos o getopt para ler os parametros de entrada.

O ShiftOr e o Wu Manber foram implementados usando um vetor para dar suporte a qualquer tamanho(Em contraste com a solucao original de usar apenas um inteiro como bit set). E o projeto foi implementando considerando apenas caracteres de 0 a 255, entao qualquer coisa maior que UTF-8 não roda corretamente nesse projeto.

3 Testes e Resultados

Os testes realizados foram conduzidos por scripts implementados em na linguagem python, o script é capaz de invocar a ferramenta PMT no console com

um determinado algoritmo, para um ou multiplos padrões de diferentes tamanhos, e um determinado erro, redirecioando as informações sobre tempo para a interface de usuário do console. O tempo de execução do algoritmo foi calculando utilizando o comando TIME, considerou-se o tempo real como a medida padrão para avalização de tempo. Os valores obtidos são a media de 5 execuções para cada comando executado.

Os testes também foram realizados com os comando grep e ggrep para casamento exato e agrep para casamento aproximado para realização de baselines de comparação. Todos os testes de análise de tempo de execução, em todas as execulções foi utilizado com a opção -c, ou seja, as saídas consistiram apenas nas quantidades de linhas com ocorrência do padrão.

Também foram realizados testes para verificação da corretude dos algoritmos. Com a modificação da semântica da opção -c, que passa a contar o número de linhas, e não de ocorrencia de padrões, foi possível validar algoritmos de busca exata, comparamos a contagem de linhas com ocorrências (opção -c) entre os algoritmos da nossa ferramenta e da ferramenta grep e agrep. Para os algoritmos de busca aproximada, também foi possível a realização o mesmo procedimento, isso foi possível devido a pre configuração do comando agrep com as opções -D1 -I1 -S1, que condiciona o agrep da forma como os pesos estão configurados nos algoritmos implementados. O número de linhas obtidos no -c com o grep, agrep e ggrep em sua grande maioria apresentou valores similires em muitos casos, contudo algumas poucos eram incoerente.

3.1 Dados

Para análise dos algoritmos foram utilizados as bases de dados do Pizza&Chilli (http://pizzachili.dcc.uchile.cl/texts.html). Do corpus obtido foi utilizado os seguinte arquivos:

- Pitches (Sequência de arquivos MIDI) 55,8MB
- Sources (Arquivo de código C /Java) 210,9MB
- DBLP.xml (Arquivo estruturado XML) 296,1MB
- DNA (Sequência de DNA) 403,9MB
- Proteins (Sequência de proteinas) 1,18GB
- English (Conjunto de arquivos em inglês) 2,21GB

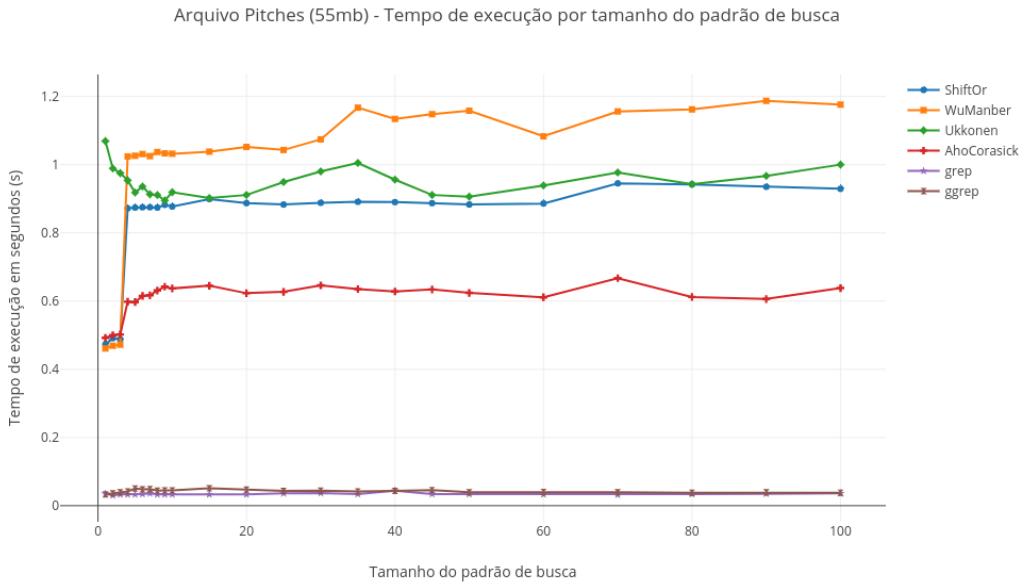
3.2 Ambiente de testes

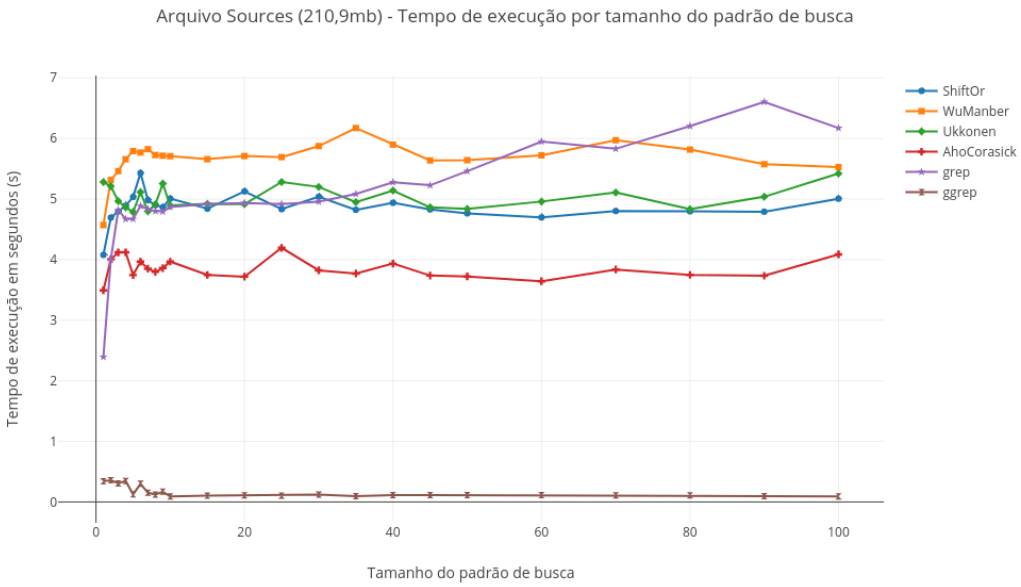
Os testes foram feitos em um computador iMac com sistema operacional macOS High Sierra v10.13.6, processador 2.7 GHz Intel Core i5, memória 8GB 1600 MHz DDR3 e com SSD de 256Gb. O compilador utilizado foi o g++ 4.2.1

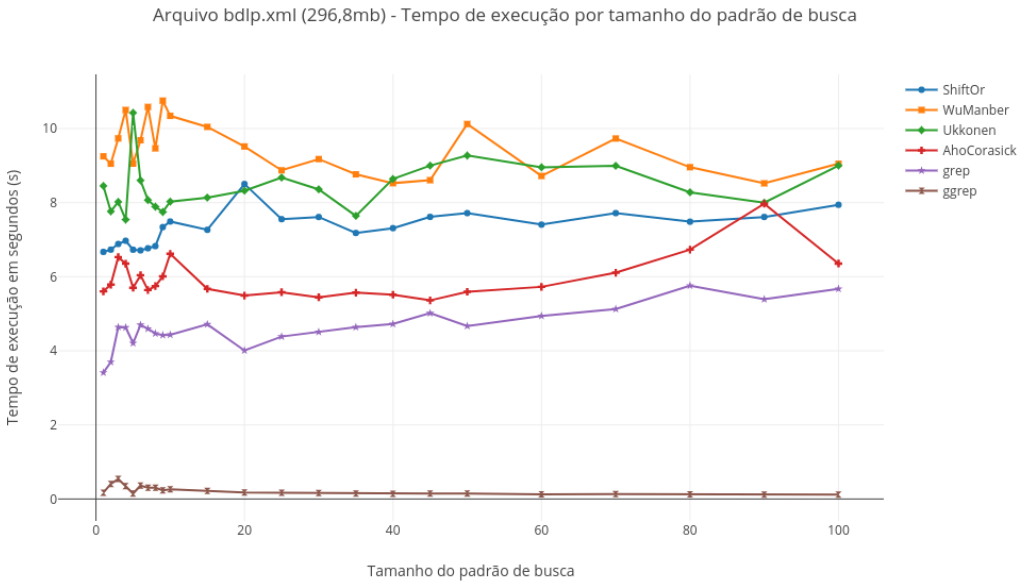
3.3 Descrição dos experimentos e resultados

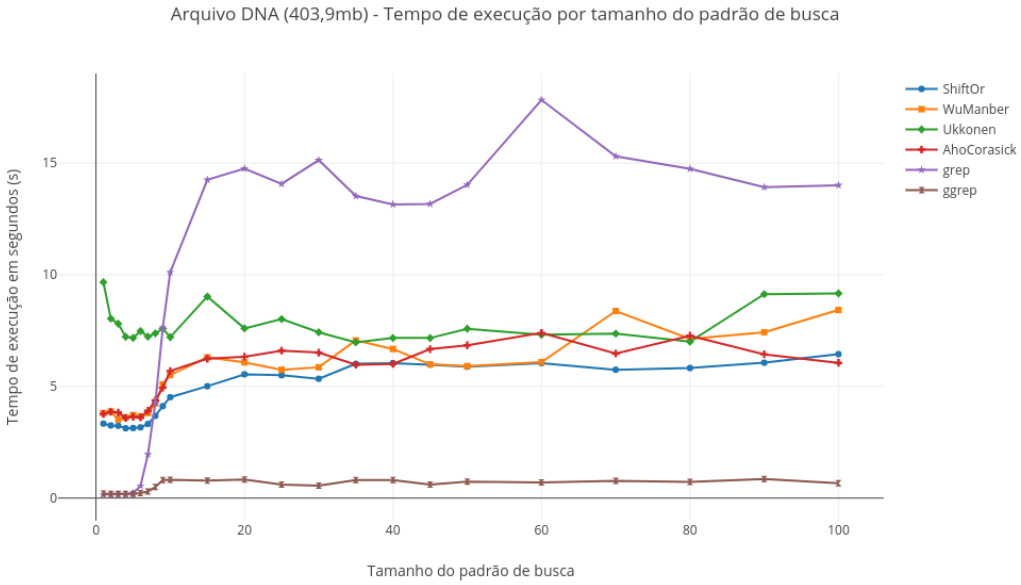
Para testar os algoritmos foram utilizados os arquivos citados acima. Inicialmente foi testado cada algoritmo separadamente, buscando um único padrão de busca, e depois com o auxilío da ferramenta como um todo foi testada a aplicação. Também foi testada a busca de muitos padrões em uma mesma entrada para avaliar o desenpenho dos algoritmos, no caso dos algoritmos cassamento aproximado, foram realizados testes com alteração do número de erro para um padrão de busca de tamanho fixo. Vale resaltar que todos os padrões de busca tem origem no proprio arquivo que esta sendo utilizado na busca, isso implica dizer que o casamento de padrão acontece pelo menos uma vez em cada execução da ferramenta.

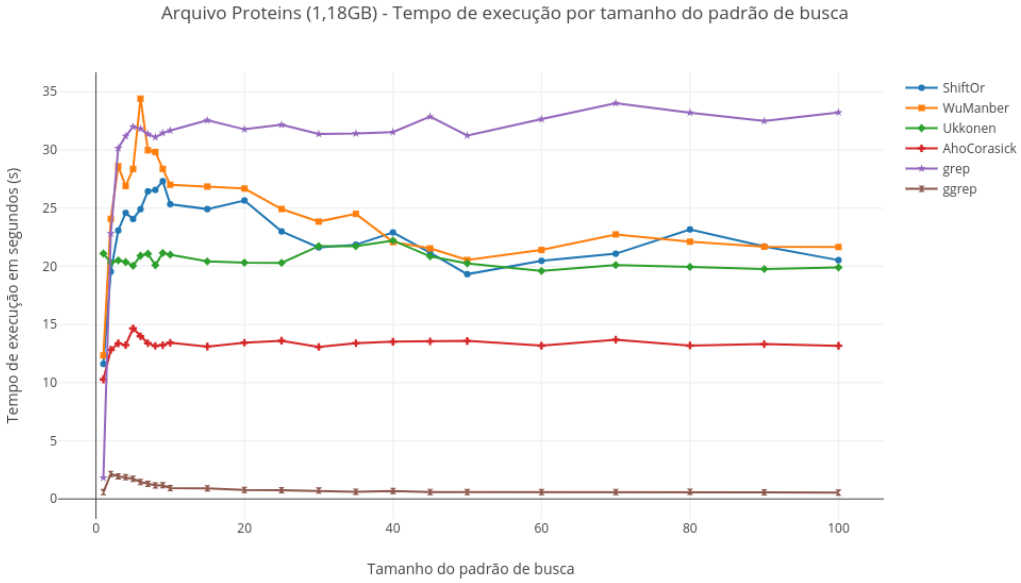
Para o primeiro teste foi utilizado todos os textos acima citados. Os padrões buscados no texto eram fragmentos do proprio texto, a busca foi realizada aplicando-se o padrão no comando com diferentes tamanhos variando de 1 a 9, de 10 a 50 em passos de 5 em 5, e de 50 a 100 em passos de 10 em 10. 23 testes eram execultados para cada algoritmo. Os resultados seguem nos gráficos abaixo. vale resalta que o grep de Mac costuma demostrar comportamento muito variável, em alguns casos ele não detecta o padrão retirado do proprio texto, ou possui um limiar onde sua eficiencia piora muito, no caso do algoritmos de casamento exato, o Aho se mostrou expressivamente melhor com relação ao ShiftOr, ja para o casamento aproximado, mesmo com a distancia de edição 0, o Ukkonem se mostrou mais eficiente que o WuManber.

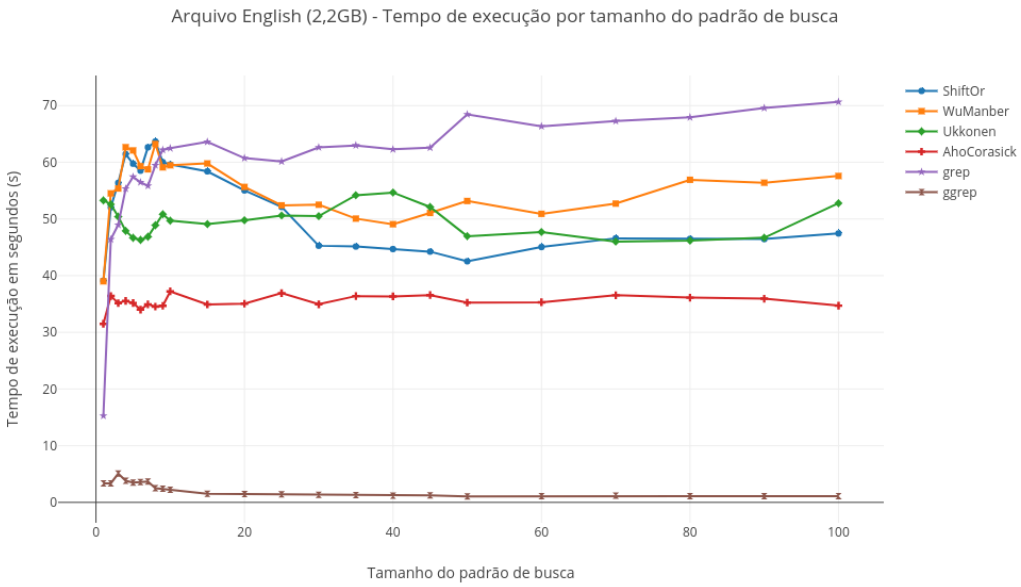



Arquivo Sources (210,9mb) - Tempo de execução por tamanho do padrão de busca
Tempo de execução em segundos (s)
Tamanho do padrão de busca
ShiftOr
WuManber
Ukkonen
AhoCorasick
grep
ggrep

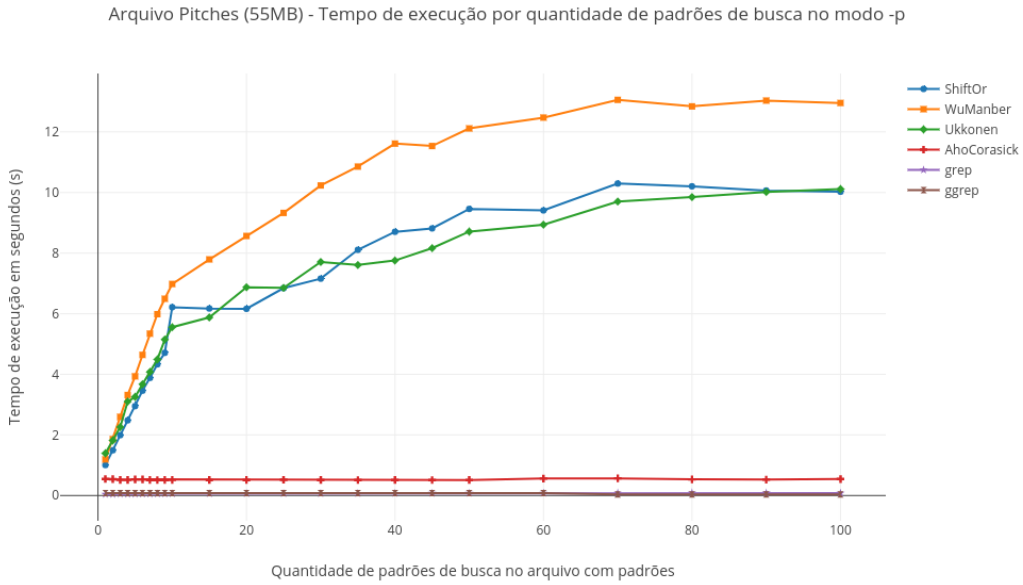

Arquivo bdlp.xml (296,8mb) - Tempo de execução por tamanho do padrão de busca
Tempo de execução em segundos (s)
Tamanho do padrão de busca
ShiftOr
WuManber
Ukkonen
AhoCorasick
grep
ggrep

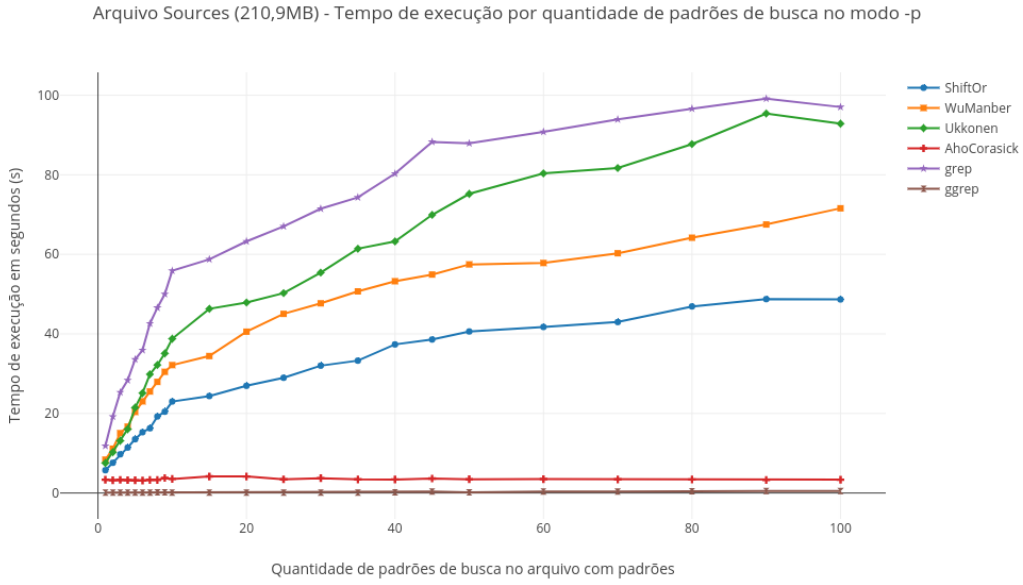

Arquivo DNA (403,9mb) - Tempo de execução por tamanho do padrão de busca
Tempo de execução em segundos (s)
Tamanho do padrão de busca
ShiftOr
WuManber
Ukkonen
AhoCorasick
grep
ggrep
0
5
10
15
0
20
40
60
80
100


Arquivo Proteins (1,18GB) - Tempo de execução por tamanho do padrão de busca
Tempo de execução em segundos (s)
Tamanho do padrão de busca
ShiftOr
WuManber
Ukkonen
AhoCorasick
grep
ggrep
0
5
10
15
20
25
30
35
0
20
40
60
80
100



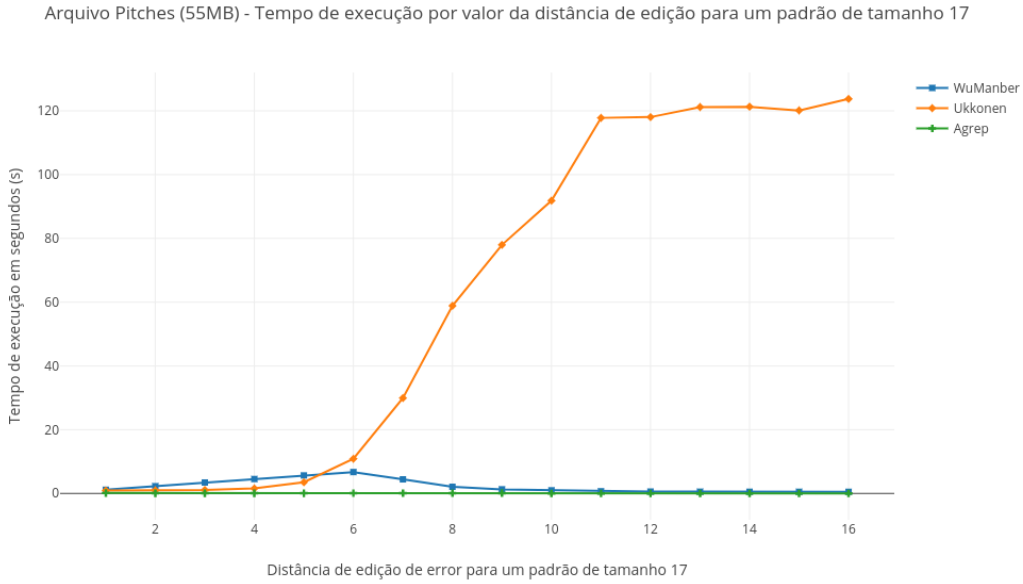
Para o segundo teste, foram utilizados os textos Pitches e Source. Foi processado varios arquivos com numero de padrão variando entre 1 a 100 padrões, todos contidos dentro dos respectivos textos. O objetivo desse experiemnto é validar a eficiência do Aho Corasick para processar multiplos padrões em pipeline, ou seja, ao mesmo tempo. Como esperado o Aho demostrou resultados muito expressivo com relação aos outros algoritmos. Esse resultado define a heristica para casos onde de deseja realizar um casamento extado de padrão.

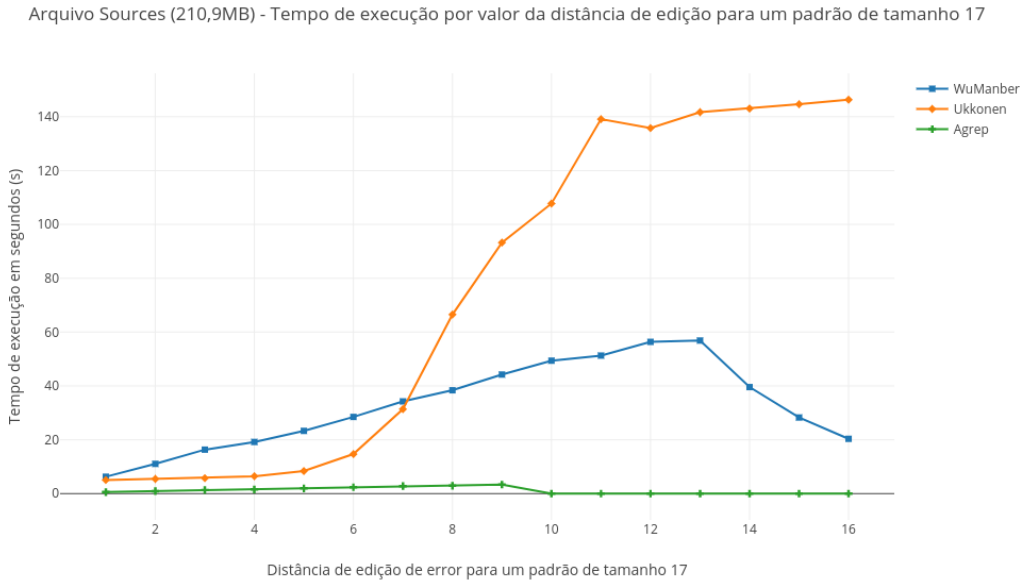




Para os algoritmos de cassamento aproximado, os testes foram realizados variando o numero de erro de edição para um padrão fixo de tamanho 17, no caso, variamos o valor da edição entre 0 ate 16 e analisamos o comportamento do tempo de execução. O teste foi realizado com os textos Pitches e Source, com pradrões contidos nos respectivos textos. Como resultado podesse observar

que o Wu Manber possui melhor desempenho nos caso nos casos de aumento do número de edição, notasse que quando a distância de edição é menor ou igual a 1/3 do tamanho do padrão o Ukkonen se sai melhor, no outro caso, o Wu Manber se sai melhor. resultados podem ser vistos nos graficos abaixo:





4 Conclusões

Analisando comparativamente os algoritmos implementados quanto a busca exata podemos concluir que o Aho corasick ganhou com destaca em tempo de execução, além da possibilidade de execução desse algoritmo com multiplos padrões me pipeline o que permite autissima eficiência em muitos casos. Vale resaltar que em comparação com o grep implementado no Mac OS ele demostrou resultados muito bons.

Quanto aos algoritmos com casamento aproximado de padrão podemos notar que para distância de edição pequenas com relação ao padrão, o ukkonen se sai muito bem, pois necessita de pouco processamento inicial, contudo, para distancia de edição elevada, o tamanho do preprocessamento do padrão se torna muito custoso, sendo desvantajoso, Já o WuManber tira proveito dessa caracterista, podendo executar rapidamente com relação ao Ukkonen ja que não possui muito preprocessamento de padrão, e conforme aumentamos o número de edição, mais padrões são encontrados, podendo retornar, se nem se quer ler toda a linha.