Processamento de Cadeias de Caracteres

Relatório do Projeto 1
2015.2

Guilherme Peixoto
Rafael Acevedo

Recife, 25 de Outubro de 2015

Sumário

1. Identificação

1.1 Equipe

Guilherme Palma Peixoto (gpp)
Rafael Acevedo de Aguiar (raa7)

1.2 Descrição da contribuição dos membros

Guilherme Peixoto: Implementação dos algoritmos de string matching exatos e execução dos testes relacionados a esses algoritmos

Rafael Acevedo: Implementação dos algoritmos de matching aproximado e execução dos testes relacionados a esses algoritmos

2. Implementação

2.1 Algoritmos implementados

Foram implementados 2 algoritmos para matching exato: o Algoritmo de Knuth-Morris-Pratt(KMP) e o Algoritmo de Aho-Corasick. Já para matching aproximado, foram implementados o Algoritmo de Sellers e o Algoritmo de Wu-Manber.

2.2 Situações nas quais cada algoritmo é empregado

No caso de matching exato, usamos o KMP quando o usuário entra com apenas um padrão. Quando há mais de um padrão, é usado o Aho-Corasick.

Para matching aproximado, utilizamos o Wu-Manber quando o padrão tem menos que 64 caracteres(tamanho da palavra de máquina) e o Sellers caso o padrão tenha 64 ou mais caracteres.

2.3 Detalhes de implementação relevantes

Na nossa implementação do algoritmo de Aho-Corasick, a função de transição entre os estados foi implementada como uma matriz (ao contrário de outra estrutura como map) pois esta executa acesso às transições em tempo constante.

No algoritmo de Wu-Manber, visto que o padrão sempre terá menos que 64 bits, foi usado unsigned long long ao invés de bitset para as máscaras de bits.

Durante a leitura das entradas, fazemos a chamada dos algoritmos quando uma linha do arquivo de texto é lida(evita que um arquivo inteiro seja carregado na memória). Caso haja algum pré-processamento, este é feito antes da leitura do arquivo de texto. Isso foi feito para evitar o carregamento de um arquivo de texto inteiro na memória.

O projeto foi desenvolvido para trabalhar exclusivamente com caracteres ASCII, já que alguns dos algoritmos implementados necessitam de saber o alfabeto de antemão para seu funcionamento.

3. Testes

A fim de testar a performance do projeto desenvolvido, podemos dividir os testes nas seguintes fases: busca exata e busca aproximada. A menos que explicitamente anotado, todos os testes foram feitos em: OS X 10.11, 2.6 GHz Intel Core i5, 8 GB 1600 MHz DDR3.

3.1 Busca Exata

3.1.1. KMP

Para medir a performance em busca exata, foi feita a busca por padrões de tamanhos diversos em bases de características diferentes: em textos de linguagem natural e bases de "caráter genético" (i.e.: as *strings* são padrões de genomas, proteínas, etc). A tabela a seguir mostra a performance de acordo com o tamanho de um padrão buscado e a base de texto. O algoritmo utilizado nesse caso é o KMP.

	englishTexts/bible.txt		genome/ecoli.txt		protein/hs.txt	
tam 5	0,045	0,002	0,056	0,004	0,041	0,001
tam 10	0,044	0,002	0,055	0,002	0,042	0,002
tam 50	0,044	0,002	0,058	0,002	0,043	0,002
tam 100	0,044	0,002	0,055	0,002	0,044	0,006

Para cada dupla de tamanho do padrão e base, há um par de valores associados: a média de tempo de execução (em segundos) e o desvio padrão associado em 100 execuções. Pode ser observado que a performance do algoritmo é estável, possuindo valores baixos de desvio padrão.

Além desses testes, foi feito uma série adicional de testes dentro do arquivo *genome/ecoli.txt* para avaliar melhor a influência do tamanho do padrão na busca.

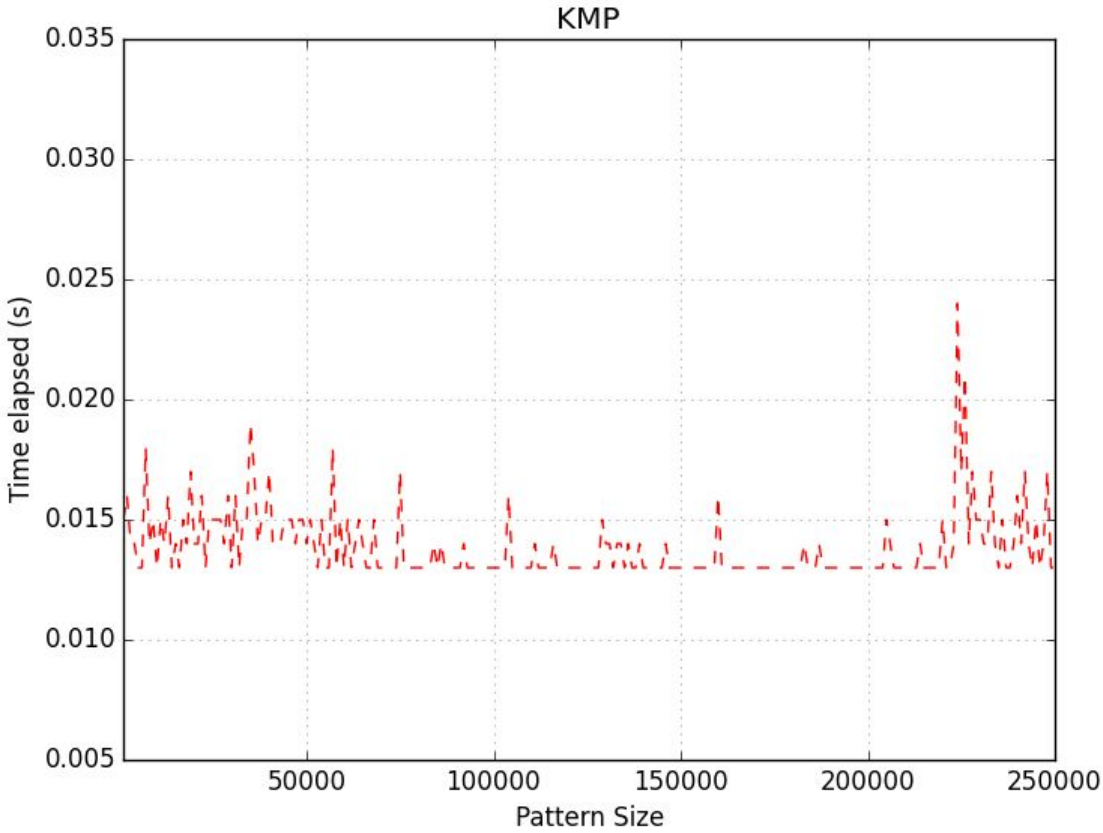

Gráfico (1): Performance do algoritmo KMP em relação ao tamanho do padrão a ser buscado.

De acordo com o gráfico acima, podemos observar como o tamanho do padrão afeta a performance. O tamanho inicial foi 1000 caracteres e foi iterado de 1000 em 1000 até o tamanho 250,000. Devido à pequena escala representada, podemos observar apesar de alguns picos de diferença na performance, a performance é aproximadamente constante. Por fim, também foi testado a performance do KMP de acordo com o tamanho do texto. Para o teste, foi utilizada uma string de padrão (aleatória) de tamanho 5. A cada iteração, foram geradas N strings aleatórias de tamanho 50 para o padrão ser buscado em cada, tal que N varia entre 1000 e 50000 (de 1000 em 1000). De acordo com os resultados de expostos no gráfico abaixo, podemos observar a linearidade esperada da performance do algoritmo de acordo com o tamanho do texto a ser buscado.

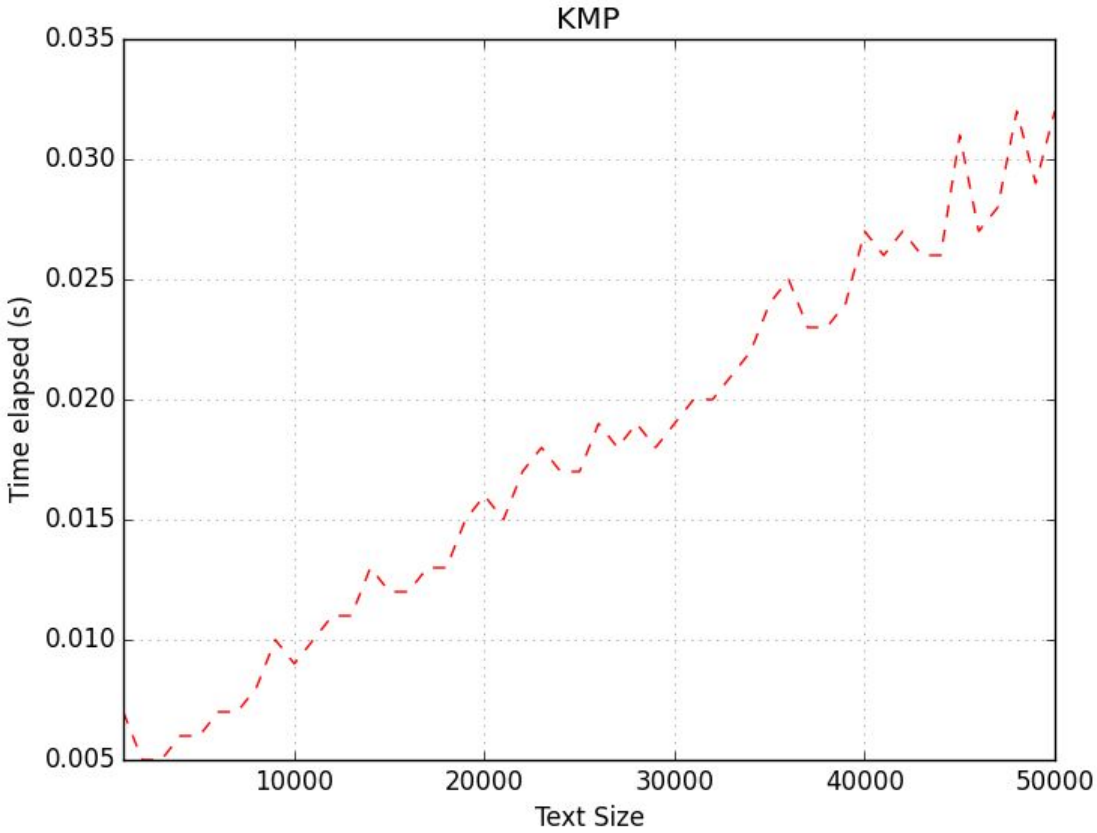

Gráfico (2): Performance do algoritmo KMP em relação ao tamanho do texto de entrada.

3.1.2. Aho Corasick

Também foi testada a performance ao procurar simultaneamente por vários padrões no mesmo texto, na qual utiliza-se o Aho-Corasick para realizar essa busca. Para o teste, foram utilizadas strings aleatórias. Para o conjunto de padrões, foram geradas 1000 strings de tamanho 5, 10, 50 e 100 aleatoriamente, enquanto para o conjunto de padrões foram geradas 40,000 strings de tamanho 200. Os resultados são reportados de forma análoga, ao longo de 100 execuções para cada configuração:

Performance Aho-Corasick		
Tamanho do Padrão	Tempo médio execução (s)	Desvio Padrão
5	0.130	0.007
10	0.128	0.006

50	0.144	0.014
100	0.154	0.014

Em adição aos experimentos acima, foi gerado um gráfico que mostra a performance do algoritmo de acordo com a variação do tamanho do padrão (em uma única execução para cada tamanho). Para cada tamanho de padrão (variando de 5 a 100), foi gerado 100 strings aleatoriamente para ser buscado na mesma base utilizada acima (40,000 strings de tamanho 200). O gráfico abaixo mostra que a performance do algoritmo é aproximadamente constante de acordo com o tamanho do padrão a ser buscado.

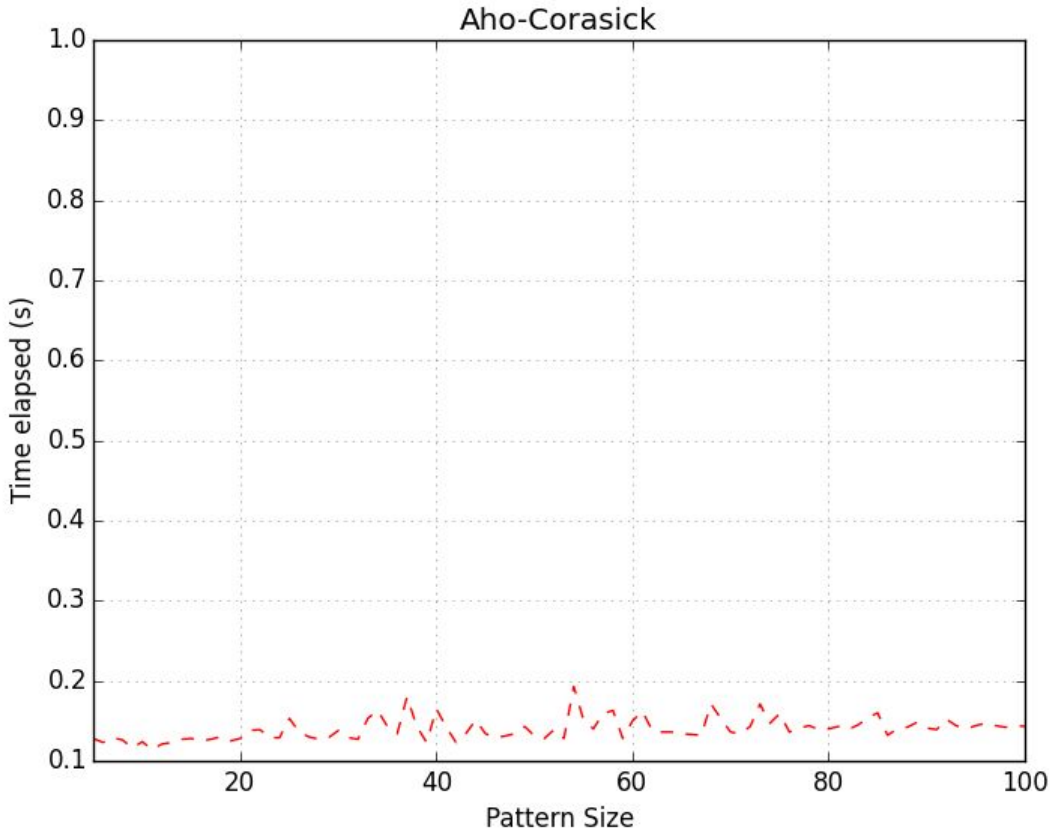

Gráfico (3): Performance do algoritmo Aho-Corasick em relação ao tamanho do padrão a ser buscado.

Similar ao teste realizado com o KMP, também medimos a performance do Aho-Corasick em relação ao tamanho do texto a ser buscado. O experimento foi realizado de forma análoga, no entanto ao invés de buscar por um único padrão de tamanho 5 em cada iteração, foram buscados por 100 padrões de tamanho 5 a cada iteração. Também pode ser confirmado que o algoritmo apresenta performance linear de acordo com o tamanho do texto a ser buscado.

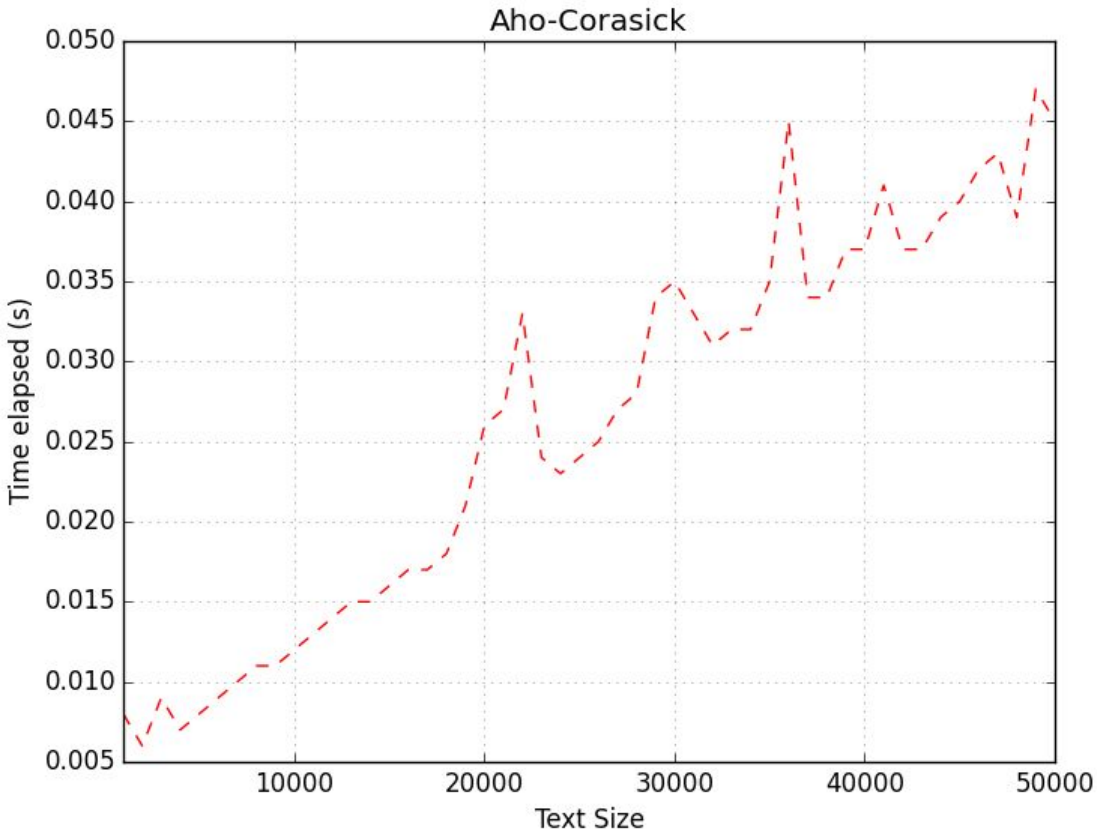

Gráfico (4): Performance do algoritmo Aho-Corasick em relação ao tamanho do texto de entrada.

3.2 Busca Aproximada

Os testes realizados para busca aproximada são similares aos realizados para busca exata. A fim de medir a performance em busca exata, primeiro é feito a busca por um padrão em textos que possuem características distintas, de forma análoga ao experimento inicial. Para padrões de tamanho inferior a 64 caracteres, Wu-Mamber é utilizado como algoritmo; caso contrário a busca é feita utilizando o Sellers. Na tabela abaixo, observamos a performance do Wu-Mamber.

3.2.1 Wu-Mamber

Para cada configuração são feitas 100 execuções e analisada a média do tempo de execução e o desvio padrão. Para cada documento a ser buscado, um único padrão é buscado (todos os padrões são coerentes com o texto a ser buscado), no entanto o erro é variado.

Performance Wu-Mamber						
Erro	**englishTexts\bible.txt**		**genome\ecoli.txt**		**protein\hs.txt**	
K	**Média**	**DP**	**Média**	**DP**	**Média**	**DP**
1	0,020s	0,001	0,046s	0,004	0,034s	0,004
3	0,021s	0,001	0,046s	0,002	0,034s	0,002
5	0,020s	0,0009	0,052s	0,006	0,034s	0,002

Também podemos analisar a performance do Wu-Mamber de acordo com a variação do tamanho do padrão, com a restrição de que o tamanho do padrão não deve conter mais que 64 caracteres.

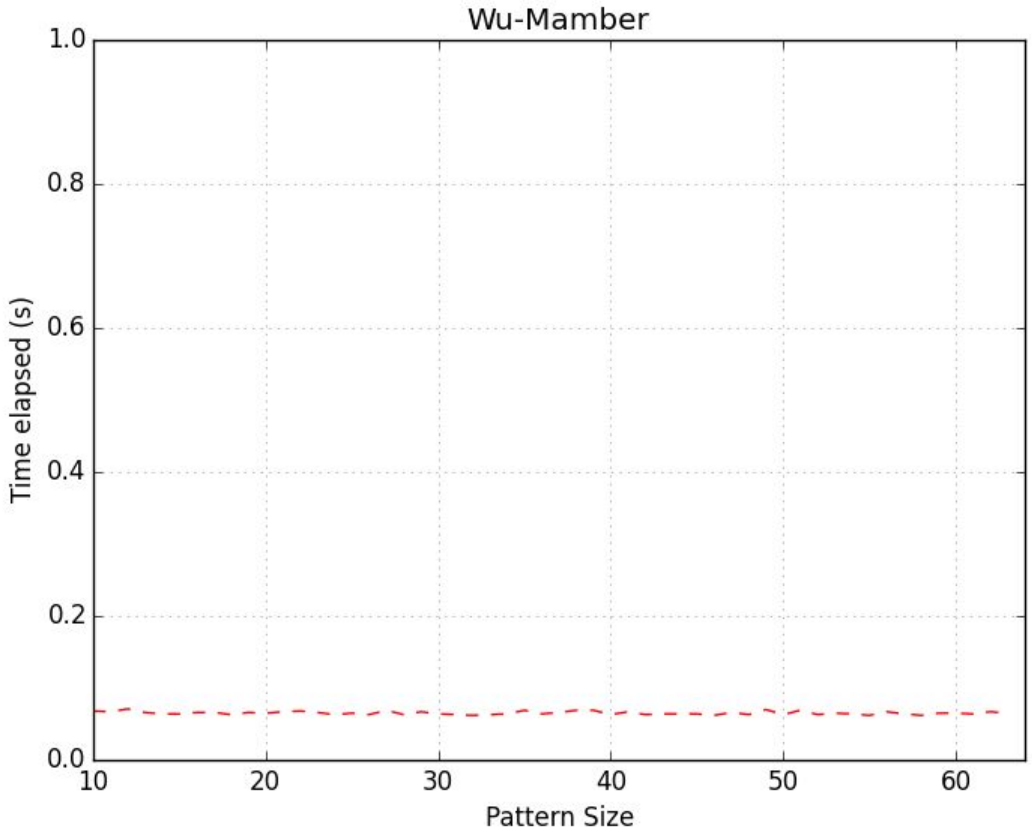

Gráfico (5): Performance do algoritmo Wu-Mamber em relação ao tamanho do padrão a ser buscado.

Nesse teste, foi realizado a busca por um padrão aleatório de tamanho 10 a 64 caracteres com o erro limite igual a 1 dentro de 40,000 strings aleatórias de tamanho 200 cada. Novamente, podemos observar que dentro dos limites suportados pela implementação realizada do Wu-Mamber, o tamanho do padrão não prejudica a performance.

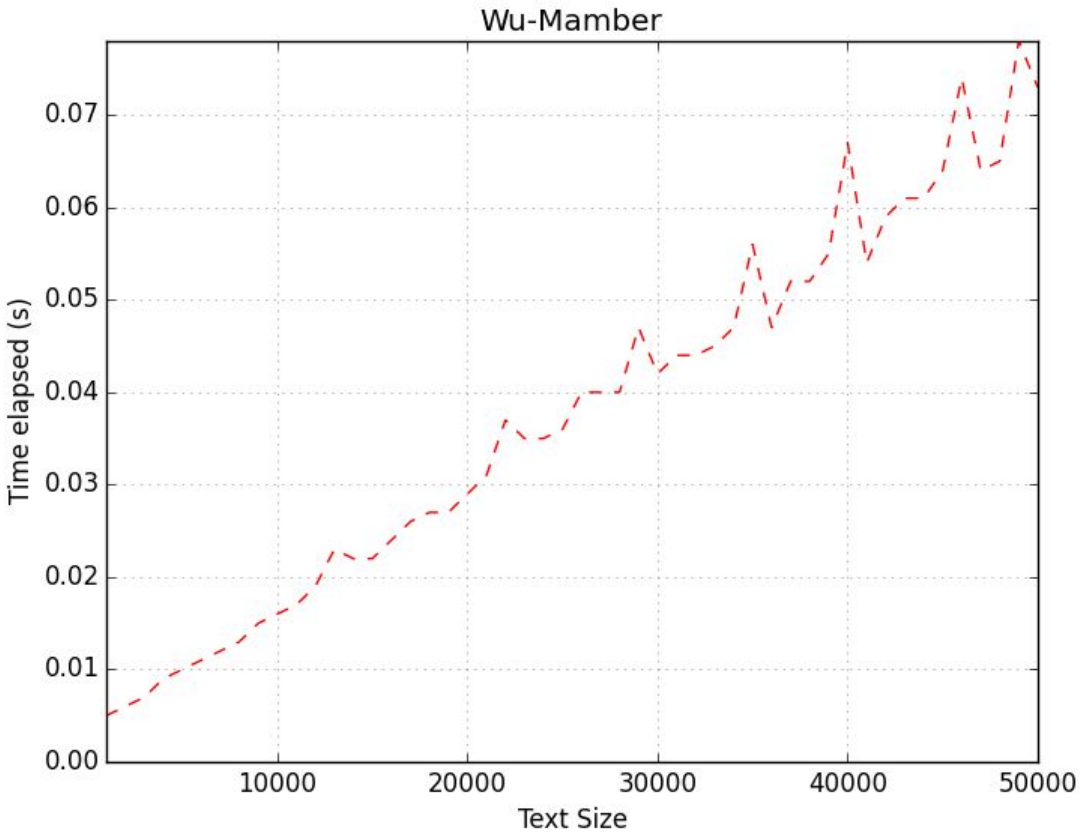

Gráfico (6): Performance do algoritmo Wu-Mamber em relação ao tamanho do texto de entrada.

No gráfico acima, observa-se que o algoritmo responde em performance linear em relação ao crescimento do tamanho do texto a ser buscado o padrão. O tamanho do texto varia de 1000 a 50000 linhas (50 caracteres cada) e é feita a busca aproximada por um padrão de tamanho 5 com erro limite igual a 3.

3.2.2. Sellers

Para testar o algoritmo Sellers, foi fixado o tamanho do padrão em 100 (uma vez que para qualquer padrão de tamanho inferior a 65 o Wu-Mamber é acionado) e foi feita a busca aproximada em três bases diferentes, analogamente aos experimentos anteriores.

Performance Sellers						
Erro	**englishTexts\bible.txt**		**genome\ecoli.txt**		**protein\hs.txt**	
K	**Média**	**DP**	**Média**	**DP**	**Média**	**DP**
10	0.065	0.007	0.043	0.006	0.031	0.002
25	0.062	0.003	0.043	0.003	0.032	0.003

50	0.064	0.007	0.044	0.007	0.032	0.002

Observamos também de acordo com o gráfico a seguir o impacto (ou ausência de) que a variação no tamanho do padrão provoca na performance do algoritmo. Para os valores testados, podemos perceber que, assim como nos outros experimentos, o tamanho do padrão pouco influencia na performance do algoritmo.

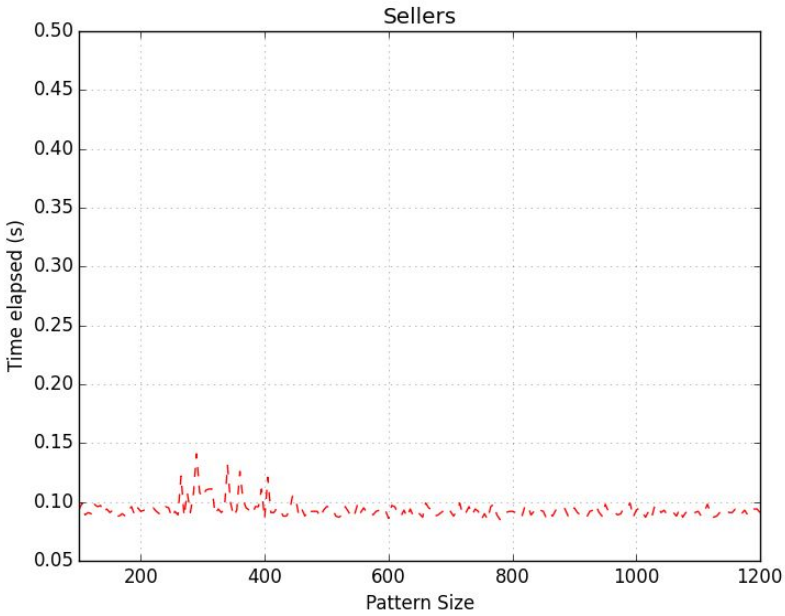

Gráfico (7): Performance do algoritmo Sellers em relação ao tamanho do padrão a ser buscado.

É possível ver também que, de acordo com o esperado, o algoritmo obtém uma performance linear de acordo com o tamanho do texto, como observado no gráfico a seguir.

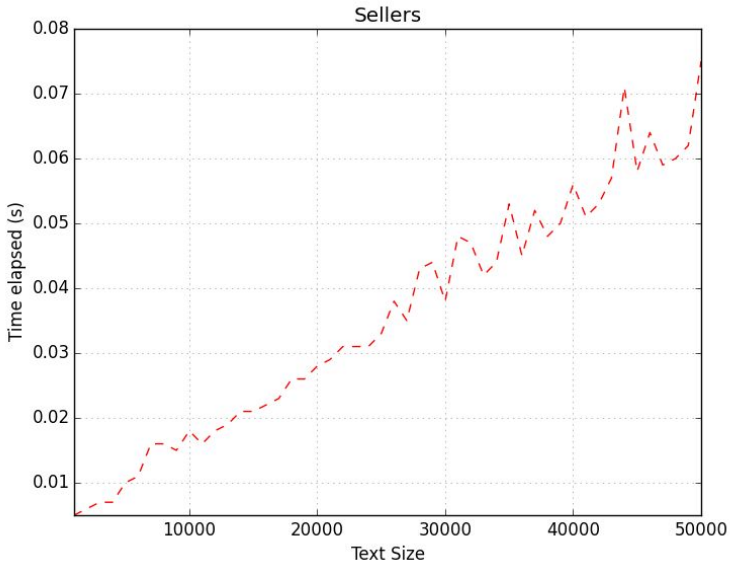

Gráfico (8): Performance do algoritmo Sellers em relação ao tamanho do texto de entrada.

3.3 Benchmarks

Na fase de testes comparamos também a performance da nossa implementação com a da ferramenta grep (versão: (BSD grep) 2.5.1-FreeBSD, OS X 10.11) para busca de padrões exatos. Cada teste foi executado 100 vezes e foi obtida a média e desvio padrão do tempo de execução. Mostramos os resultados a seguir:

3.3.1: Busca por um padrão

Nesse caso de teste, foi feita a busca por um padrão gerado aleatoriamente de tamanho 5 dentro de um arquivo de texto gerado aleatoriamente que contém 400,000 linhas com 200 caracteres cada. Os resultados foram:

Ferramenta	Média	Desvio Padrão
grep	0.136s	0.041
pmt\kmp	0.089s	0.012

3.3.2: Busca por vários padrões

Nesse caso de teste, foi feita a busca por 100 padrões gerados aleatoriamente de tamanho 5 dentro de um arquivo de texto gerado aleatoriamente que contém 400,000 linhas com 200 caracteres cada. A opção "-f" foi passada ao grep para que fosse feita a busca por múltiplos padrões. Os resultados foram:

Ferramenta	Média	Desvio Padrão
grep	3.960s	0.090
pmt\aho-corasick	0.117s	0.002

A queda de performance drástica do grep nesse caso deve-se ao fato de que a opção "-f" para procurar por vários padrões não é feita de forma "simultânea", ao contrário de como é realizado pelo algoritmo de Aho-Corasick.

3.3.3: Busca aproximada

O *benchmark* de comparação para a busca aproximada foi o *agrep* 4.17-2_i386, VMWare ubuntu 14.04, 4GB RAM, Intel i7. Foi buscado por um padrão aleatório de tamanho 32 dentro de um arquivo gerado aleatoriamente que contém 400,000 linhas com 200 caracteres cada. Os resultados foram:

Ferramenta	Média	Desvio Padrão
agrep	0.670s	0.001
pmt\wu_mamber	0.100s	0.007

Apenas o Wu-Mamber foi empregado nesse caso de de teste porque a ferramenta *agrep* dá suporte apenas para padrões que contém apenas até 32 caracteres. É importante notar também que a performance do agrep é influenciada pelo fato do teste ter sido executado em uma configuração de máquina virtual, ao invés de ter sido executado diretamente pelo sistema operacional.