UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
CIÊNCIAS DA COMPUTAÇÃO

Projeto 1 - Relatório

PEDRO HENRIQUE TÔRRES SANTOS (phts)

ULLAYNE FERNANDES FARIAS DE LIMA (uffl)

Recife, Pernambuco

2018

1. Identificação

O projeto foi desenvolvido por Pedro Henrique Tôrres Santos e Ullayne Fernandes Farias de Lima. A implementação dos algoritmos foi divida tal que cada integrante implementar um algoritmo de casamento aproximado e um de casamento exato. Portanto, Pedro implementou os algoritmos Boyer-Moore, Shift-Or, e Wu-Manber e Ullayne implementou os algoritmos Aho-Corasick e Ukkonen. Em outras etapas do projeto, houve contribuição de ambas as partes.

2. Implementação

2.1. Algoritmos de Casamento Exato

Os algoritmos de casamento exato utilizados no projeto foram Aho-Corasick, Boyer-Moore, Shift-or, e KMP. Para melhor compreensão das implementações feitas, será descrito informações importantes sobre a implementação de cada algoritmo.

2.1.1. Aho Corasick

O aho corasick foi implementado utilizando as funções *goto*, constrói as arestas da máquina de estados diretamente do padrão, *build_fail*, adiciona as arestas de falha a máquina de estados e Aho Corasick, encontra as ocorrências dos padrões no texto. Para representar a máquina de estados do algoritmo, é utilizado uma hash table onde relaciona o estado atual e o carácter da aresta com o próximo estado, a ideia é ‘estou no estado 1’ e a aresta é ‘c’, então o próximo estado é 2. Esta representação foi escolhida a fim de otimizar o tempo no momento da consulta do próximo estado, já que o tempo de acesso em uma hash table é constante, apesar da inserção em uma hash no pior caso é O(n) a otimização ainda apresentou o melhor resultado.

2.1.2. Shift-Or

O Shift-Or foi implementado utilizando um *bitset* de tamanho definido pela arquitetura na qual o código foi compilado. Como o tamanho do *bitset* precisa ser definido em tempo de compilação, foi feita a implementação de *dynamic_bitset* onde o tamanho seria definido em tempo de execução. Porém ao fim da implementação, o *dynamic_bitset* se

mostrou muito lento, fazendo com que ele não fosse integrado no projeto final e com que o tamanho máximo do padrão fosse definido pela arquitetura na qual o código fosse compilado. Os arquivos do *dynamic_bitset* permanecem do projeto de forma desconexa do binário compilado.

O pré-processamento do padrão é feito através da função *init_char_mask* que gera as máscaras correspondentes a cada um dos caracteres do padrão.

2.1.3. Boyer-Moore

O Boyer-Moore foi implementado usando funções auxiliares para o pré-processamento do padrão. O cálculo do bom-sufixo é feito a partir de *init_good_suffix* e *init_bad_char* é responsável por computar o mau-caractere. Já a borda do padrão é inicializada por *init_border*.

2.1.4. KMP

O KMP foi implementado utilizando as funções *init_next,* resposnável pelo pré-processamento do padrão para calcular todas as possíveis bordas e a função *kmp*, função que busca ocorrência do padrão no texto e caso ache armazena em um vetor de ocorrências.

Por questões de desempenho, a dupla escolheu não adiconar o algoritmos nos testes e na implementação da ferramenta, porém é possível encontrar a implementação na pasta source do projeto.

2.2. Algoritmo de Casamento Aproximado

Os algoritmos de casamento aproximado utilizados no projeto foram WuManber e o Ukkonen.

2.2.1. Ukkonen

O algoritmo Ukkonen foi implementado utilizando as funções *buil_ukk_fsm*, contrói a máquina de estados relativa ao padrão de entrada com a função auxiliar *diff* que verifica se dois caracteres são iguais ou diferentes e retorna um inteiro e a função *next_column* calcula o valor da próxima coluna baseado na função de transição do algoritmo. Para representar o algoritmo, escolhemos utilizar uma hash table no lugar da trie, já que fornecia acesso constante às próximas transições e na busca.

2.2.2. Wu-Manber

O Wu-Manber foi implementado através de uma extensão do Shift-Or, ou seja, todas as limitações e otimizações do Shift-Or também estão presentes nesse algoritmo. O pré-processamento é feito pelo Shift-Or e após isso o Wu-Manber utiliza as máscaras de caracteres do padrão para realizar a busca. A distância de edição é setada na inicialização do algoritmo, fazendo com que qualquer busca feita por aquela instância do algoritmo utilize a mesma distância de edição.

2.3. Informações Importantes

Neste tópico, informaremos algumas conversões realizadas no projetos como alfabeto utilizado, como funciona a leitura de entrada e algumas otimizações feitas para melhorar o desempenho dos algoritmos.

2.3.1. Alfabeto Utilizado

O alfabeto utilizado é o *Extended ASCII*, visto que para o domínio do projeto não fazia-se necessário utilizar outro tipo de encoding. Portanto caracteres *multibyte* não serão representado por um único caractere no programa, podendo haver uma penalização no desempenho.

2.3.2. Leitura de Entrada

O parsing dos argumentos do programa foi feito através da utilização do *getopt*. Os argumentos então passam por uma pequena validação, que não prossegue com a execução do programa em casos de falha. Ao falhar o programa exibe a mensagem de help indicando os argumentos e valores aceitos.

Após isso os arquivos a serem utilizados pelo programa são abertos e ficam prontos para serem consumidos através de um *istream*.

2.3.3. Otimizações

Os algoritmos do projeto foram primeiramente implementados com foco na sua funcionalidade. Após isso eles sofreram diversas alterações com o intuito de diminuir seu tempo de execução. As principais otimizações foram: utilizar métodos *inline* onde fosse

possível, passar todos os parâmetros por referência, evitar ao máximo cópia de objetos, utilizar *iterator* e *for-each* onde fosse possível, utilização de *smart-pointers*, utilizar padrão *return early*, reestruturar código de forma a fazer com que o compilador pudesse realizar a maior quantidade possível de otimizações com flag *-O2*, e trocar estruturas de dados por outras que assintoticamente poderiam ter a mesma complexidade mas em casos específicos uma possuia vantagens em relação a outras.

3. Testes

Os testes foram realizados em duas computadores diferentes, o primeiro utilizando o sistema operacional Ubuntu 16.04 com processador intel Celeron 1.60 GHz com memória 4016 Mb DDR3 com o compilador (informações sobre o compilador), o segundo (informacoes sobre o notebook de peu).

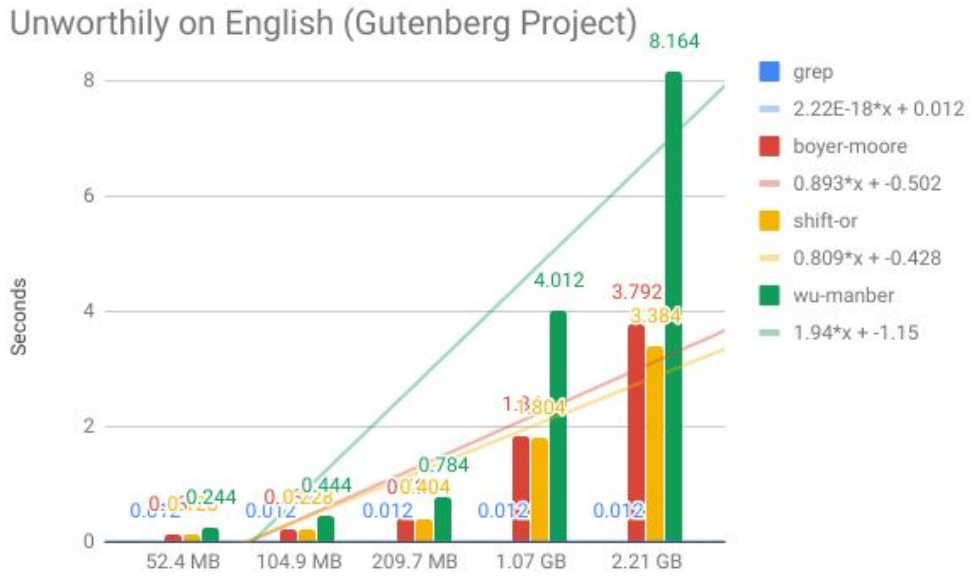
Para análise de desempenho, os algoritmos foram testados de dois modos diferentes. O primeiro modelo de teste consiste em variar o tamanho do texto utilizando o mesmo padrão de entrada para algoritmos exatos, para algoritmos aproximados os erros também variam entre {0, 1, 2, 3} . O segundo modelo consiste em variar o tamanho da entrada até 64 caracteres, como comentado acima é o limite do algoritmo shift-or.

Com objetivo de fazer uma análise mais precisa do tempo gasto por cada algoritmo, cada teste foi rodado três vezes e foi considerado o menor valor pois indica que os valores maiores possuíram alguma interferência externa. Além disso, os algoritmos de casamento exato foram comparados com o grep, para que fosse possível ter uma referência de desempenho, já para algoritmos de casamento aproximado a referência de desempenho foi feita com o agreep.

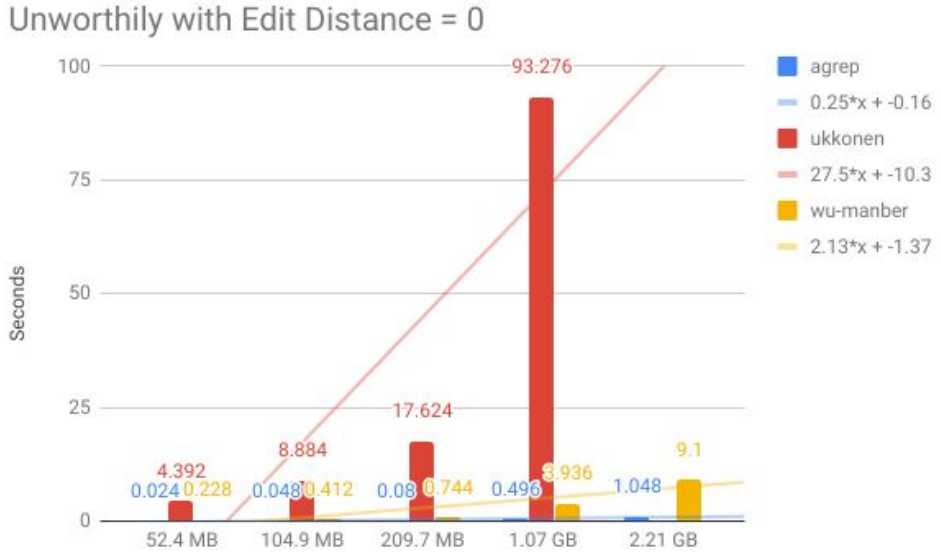
A análise da corretude dos algoritmos de casamento exato foi feita com o grep, para cada padrão as linhas de ocorrência eram comparadas para verificar se o algoritmo conseguiu atingir todos os matchs.

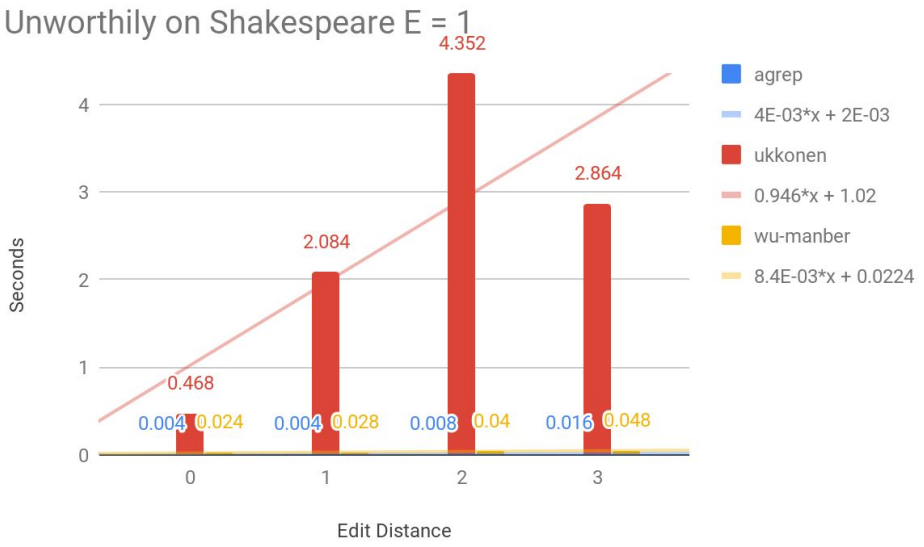
3.1. Teste com a variação do tamanho do texto

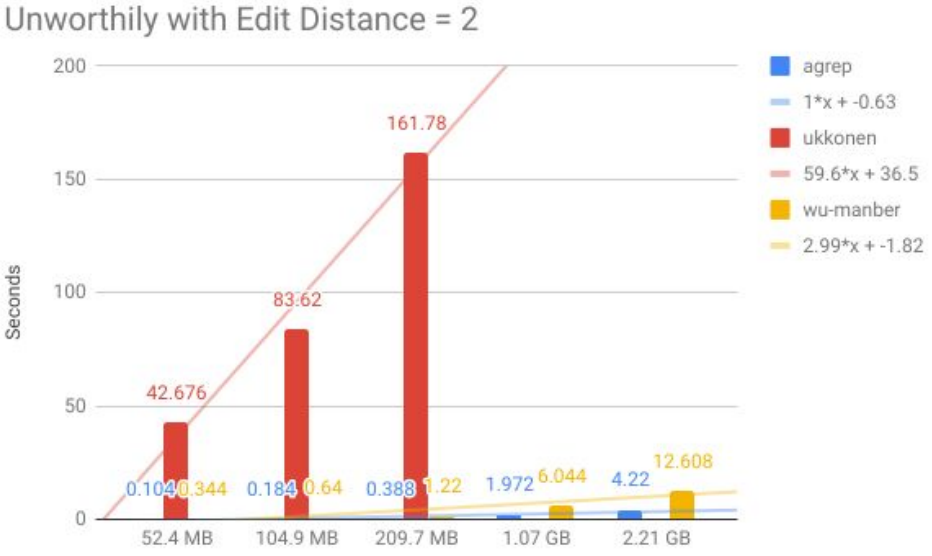
Para o teste com a variação do tamanho do texto foi utilizado o padrão “unworthily”. A seguir, é mostrado o desempenho de cada algoritmo no conjunto de testes realizados para a variação do tamanho do texto. Para casamento exato, o gráfico a seguir mostra a variação de desempenho dos algoritmos boyer-moore, shift-or, wu-manber e o grep.

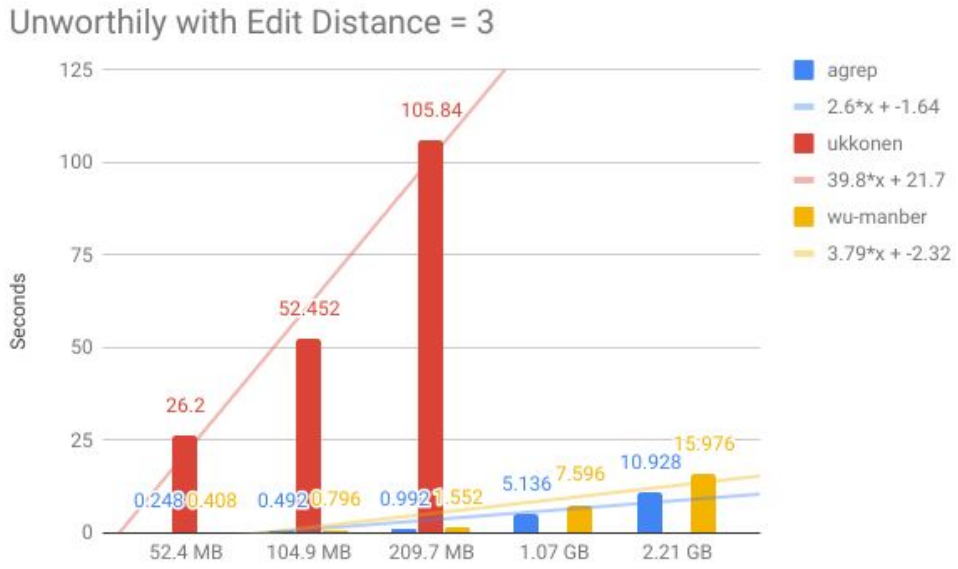


A comparação dos algoritmos de casamento aproximados foi feita variando os erros para {0, 1, 2, 3}. A seguir, é possível observar as comparações dos algoritmos ukkonen, wu-manber e agrep.

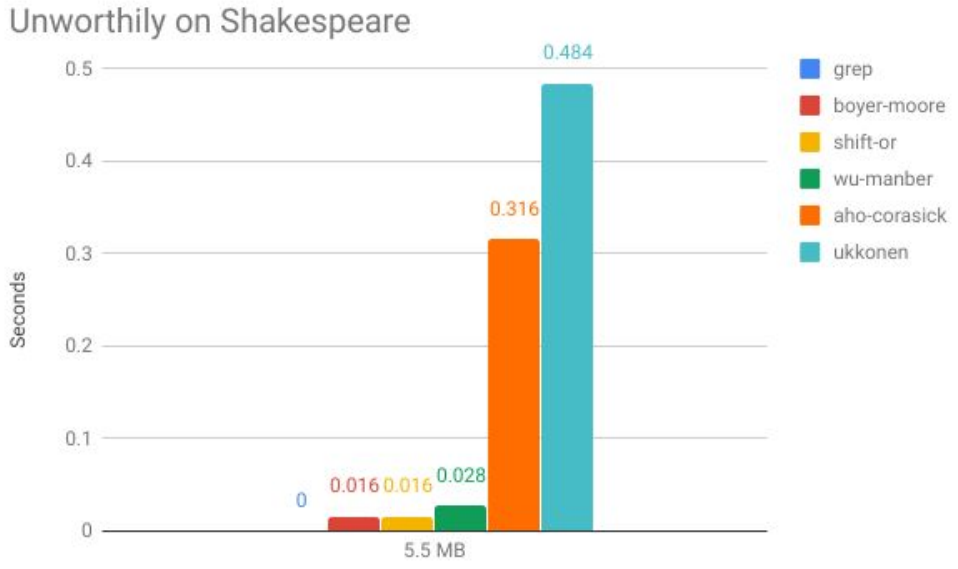



Unworthily on Shakespeare E = 1
Seconds
Edit Distance
0
1
2
3
4
0.468
2.084
4.352
2.864
0.004
0.024
0.004
0.028
0.008
0.04
0.016
0.048
agrep
4E-03*x + 2E-03
ukkonen
0.946*x + 1.02
wu-manber
8.4E-03*x + 0.0224


Unworthily with Edit Distance = 2
Seconds
0
50
100
150
200
42.676
83.62
161.78
0.104
0.344
0.184
0.64
0.388
1.22
1.972
6.044
4.22
12.608
52.4 MB
104.9 MB
209.7 MB
1.07 GB
2.21 GB
agrep
1*x + -0.63
ukkonen
59.6*x + 36.5
wu-manber
2.99*x + -1.82

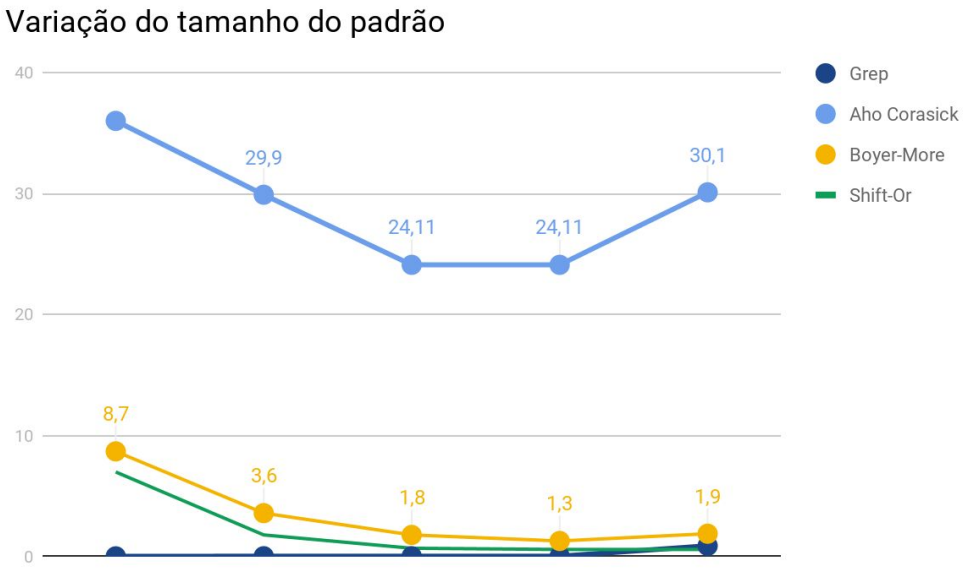


Além dos testes realizados separadamente, também foi feita a comparação dos algoritmos de casamento aproximado com erro igual a zero com o desempenho dos algoritmos de casamento exato.



3.2. Teste com a variação do padrão

Para analisar o comportamento dos algoritmos de casamento exato com variação de padrão, foram feitas variações de padrões nos tamanhos {1, 2, 4, 8, 16} a fim de observar o tempo gasto com relação a ferramenta grep (indicada na cor azul escuro).



4. Resultados

Após a realização dos dois tipos de testes, é possível observar tendências de comportamento dos algoritmos, contudo todos possuíram desempenho inferior ao grep e os algoritmos que possuíam máquinas de estados obtiveram menor desempenho dado a necessidade de atualização, adição e remoção de arestas que, no modelo utilizados, foram operações mais custosas. Além disso, também é notável o desempenho do algoritmo shift-or que possui um desempenho bastante próximo do grep.

As análises citadas acima é possível extrair pelos gráficos dos testes realizados, porém dados os testes feitos durante o desenvolvimento do projeto, a dupla conseguiu notar que o Aho-corasick possuía o desempenho melhor para múltiplos padrões considerando o Boyer-Moore e o shift-or.

Os algoritmos de distância aproximada, é possível notar que o wu-manber obteve o melhor desempenho em todas as situações de testes em relação ao ukkonen.