Universidade Federal de Pernambuco – UFPE

Centro de Informática – CIn

Processamento de Cadeia de Caracteres – IF767

Relatório de Projeto

Allyson Manoel Nascimento Venceslau - amnv
Vinicius Rebelo de Moraes - vrm

Recife, Outubro de 2017

Agradecimentos

Agradecimentos especiais a Allyson por ter sido o responsável pelo funcionamento dos algoritmos Aho Corasick e Sellers, como também a arquitetura da classe fachada e a Vinícius pelo trabalho nos algoritmos shift-or, wu-manber e na execução dos testes.

Sumário

1. Implementação

1.1 Algoritmos utilizados

Escolhemos implementar o shift-or e o Aho Corasick para a busca exata e o Seller e o Wu-Manber para a busca aproximada.

1.2 Descrição do funcionamento

Temos uma classe fachada, chamada Decisao que recebe os parâmetros dados pelo usuário e determina qual algoritmo deve ser usado e que funções dele devem ser utilizadas. Criamos uma classe abstrata Algorithm que eh classe base de todos os algoritmos implementados. Ela tem métodos implementados para leitura dos arquivos e os seguintes métodos abstratos (que sao implementados especificamente por cada algoritmo) relevantes: int count() para retornar quantas ocorrências houveram e void occ() que imprime as linhas das ocorrências.

1.2.1 Estruturas de dados usadas

As principais estruturas de dados utilizadas foram: map (na máscara de char do shiftor e wumanber), deque e vector.

1.2.2 Forma de leitura

Para leitura, passamos o caminho do arquivo para a classe algoritmo que por sua vez tem uma função para leitura linha a linha e então buscar o padrão em cada uma.

1.3 Estratégias para escolha dos algoritmos exatos

1.3.1 Exatos

Para a escolha dos exatos, primeiro verificamos se há mais de uma padrão, se sim o aho-Corasick eh a melhor escolha, pelo fato do seu grande processamento ocorrer apenas uma vez na criação da máquina de estados. Caso seja apenas um padrão, verificamos se o tamanho dele cabe na estrutura usada na máscara do shiftOr que eh um long long, ou seja vemos se a quantidade de bits do long long suporta o tamanho do padrão, caso suporte usamos o shift-or, senão o aho-corasick.

1.3.2 Aproximados

Já para os algoritmos exatos apenas fazemos a segunda verificação que ocorre nos exatos, se o tamanho do padrão cabe no long long, que também é usado pelo wu manber para guardar as máscaras dos caracteres. Se sim, utilizamos o wu manber, se não o seller.

1.4 Bugs e limitações

Uma limitação já abordada acima que afeta tanto o shiftOr quanto o wuManber, se deve a forma escolhida para representar uma máscara de um caractere, que é por meio de um long long, que serve como array de bits de cada caracter, porém o long long tem um tamanho fixo, que geralmente é de 64 bits, o que acaba limitando o tamanho do padrão que se pode trabalhar.

Uma outra limitação é o fato de não ser possível trabalhar com textos de diferentes encodings, o que fez com que fosse preciso deixar todos textos em utf-8 antes de começar os testes, pois senão os resultados não eram os mesmos do grep.

Infelizmente não conseguimos colocar os wildcards a tempo e pelo mesmo motivo, o parâmetro -p PATTERNFILE ficou funcionando apenas o aho-corasick.

Por fim, existem bugs com pelo menos os algoritmos aho-corasick e o seller, tentamos implementar ambos seguindo o que seus respectivos artigos sugeriram, porém não tivemos exatamente as saídas que se eram esperadas quando os testamos.

2. Testes e Resultados

Para fazer os testes, foram-se utilizadas scripts feitos em shell script que executavam o programa, passando diferentes parâmetros e padrões e salvando o tempo com a ferramenta GNU time, em seguida foi executado o GNU Plot para geração dos gráficos.

Os testes foram realizadas numa máquina com processador AMD A6-4400M 1.4-2.7GHz, 6GB ram e sistema operacional Debian 9. O dados utilizados para teste foram um conjunto de textos inglês[1], com aproximadamente 50MB, retirados de Pizza&Chili Corpus e um conjunto de textos de Shakespeare, com aproximadamente 5MB, retirados do github da disciplina[2].

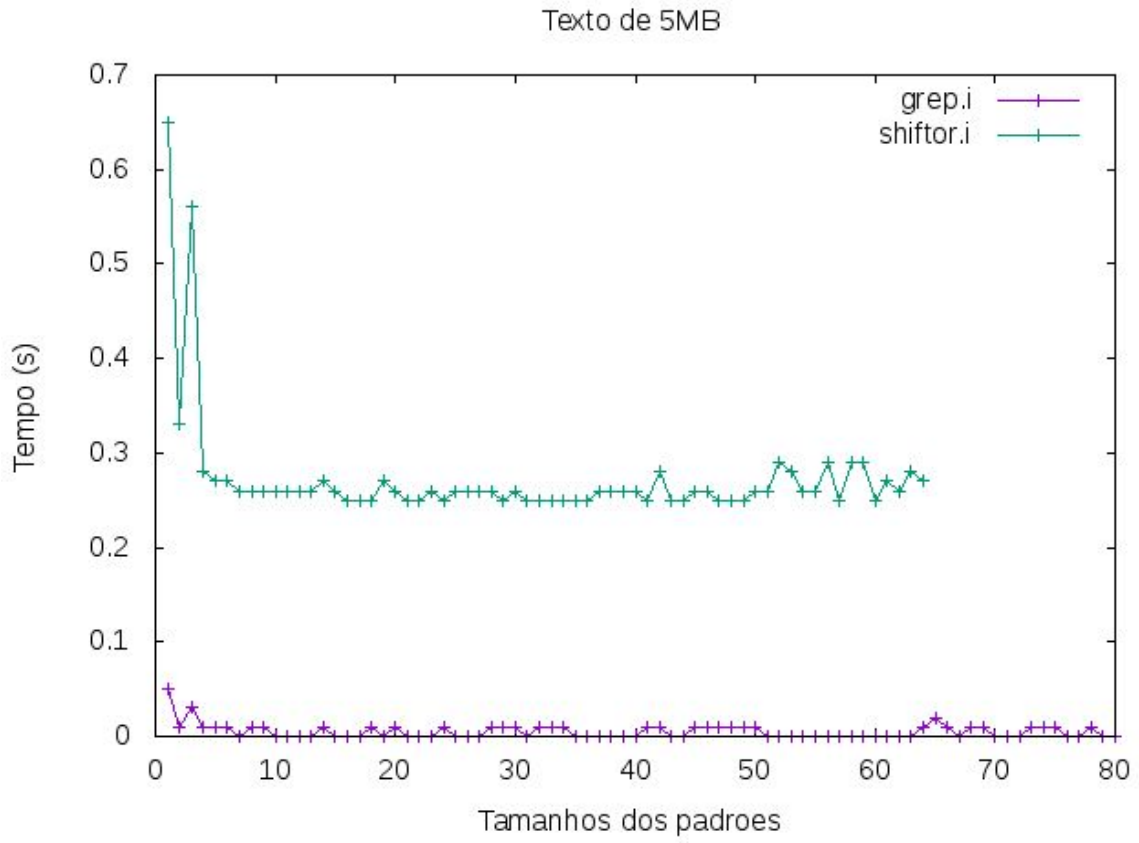
2.1 Testes em busca exata

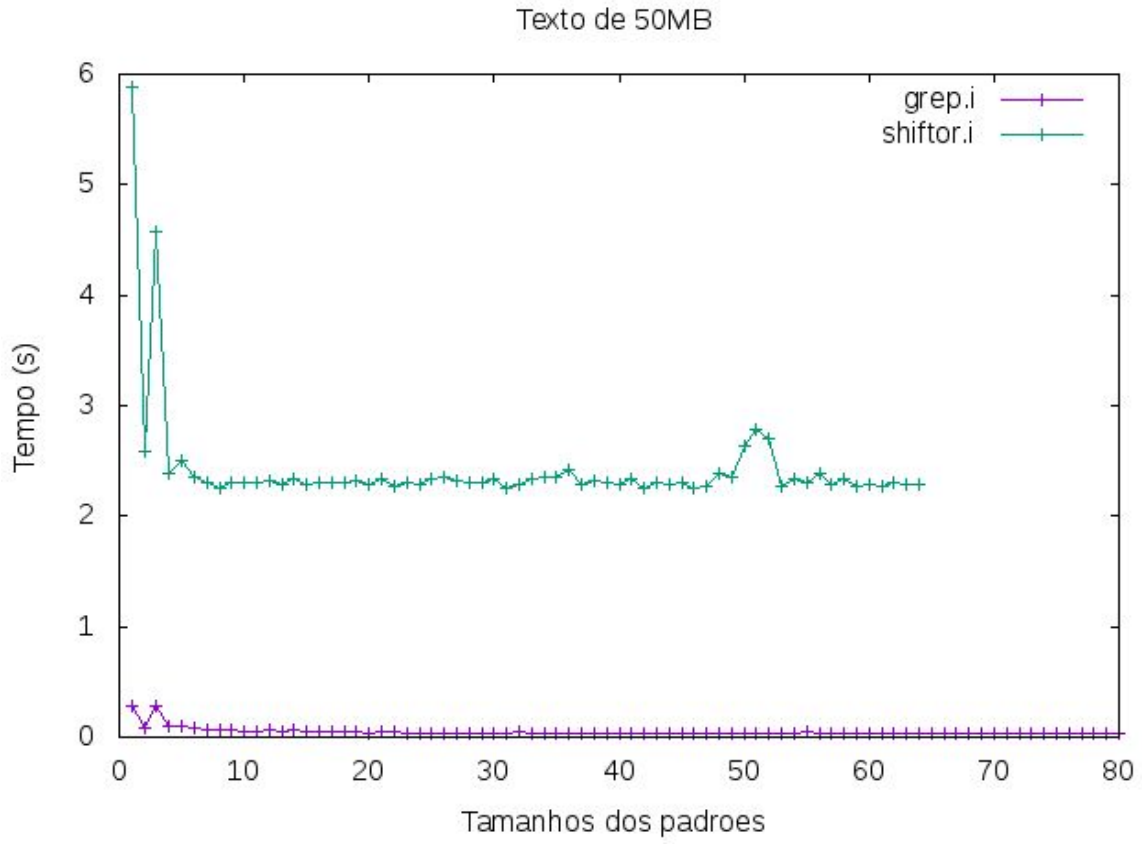
Rodamos cada algoritmo, com pmt -a algoritmo PAT FILE | wc -l, pois tiramos como medida de comparação quantas linhas eram detectadas em cada arquivo (o wc também foi usado com o grep pelo mesmo motivo). O que também ajudou a descobrirmos se nossos algoritmos retornavam resultados errados, o que aparentemente não ocorreu no shift-or,

[1] http://pizzachili.dcc.uchile.cl/texts/nlang, acesso em 15 de outubro de 2017.
[2] https://github.com/paguso/if76720172/tree/b780ae1afd23a9b71abc3fee44bea54c2e66406c/data, acesso em 15 de outubro de 2017.

contudo o Aho Corasick teve saídas tão erradas que achamos melhor não colocá-lo nos gráficos.

Com ambos conjuntos de dados, testamos padrões aleatórios de tamanhos 1 a 80.



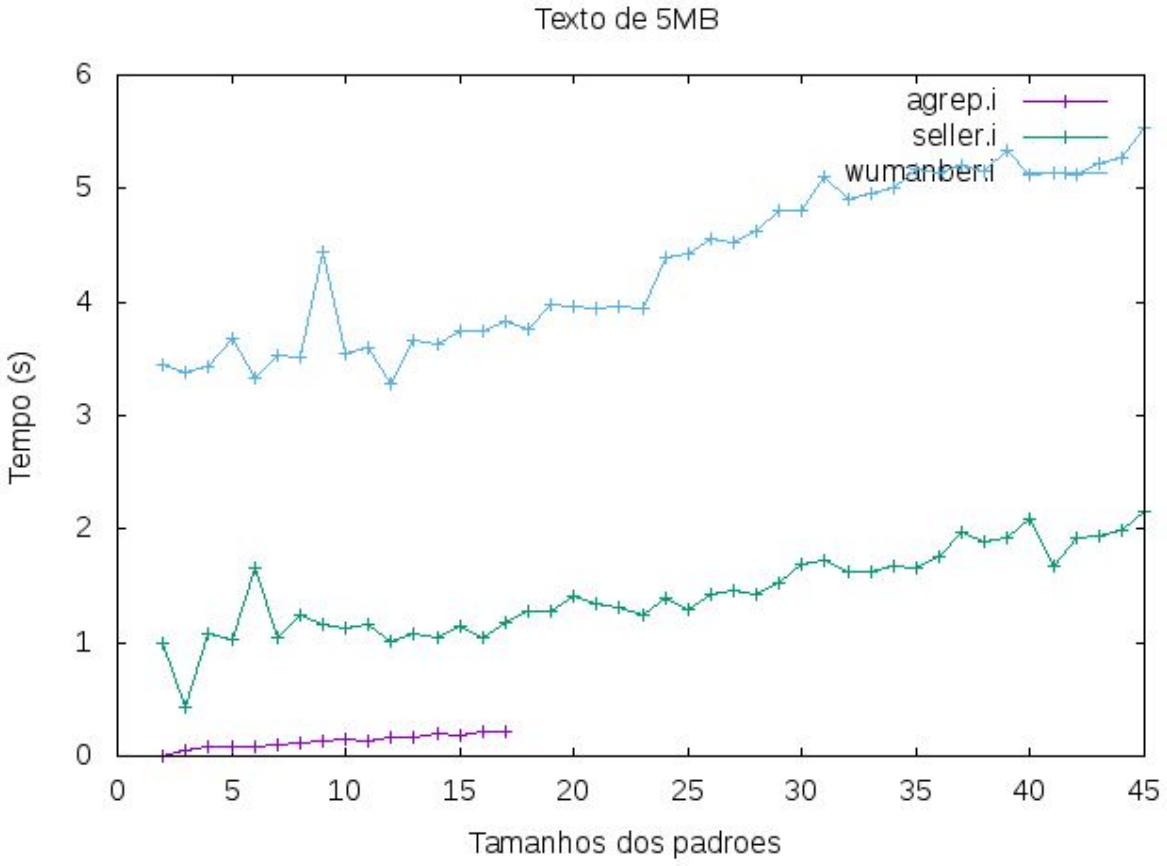


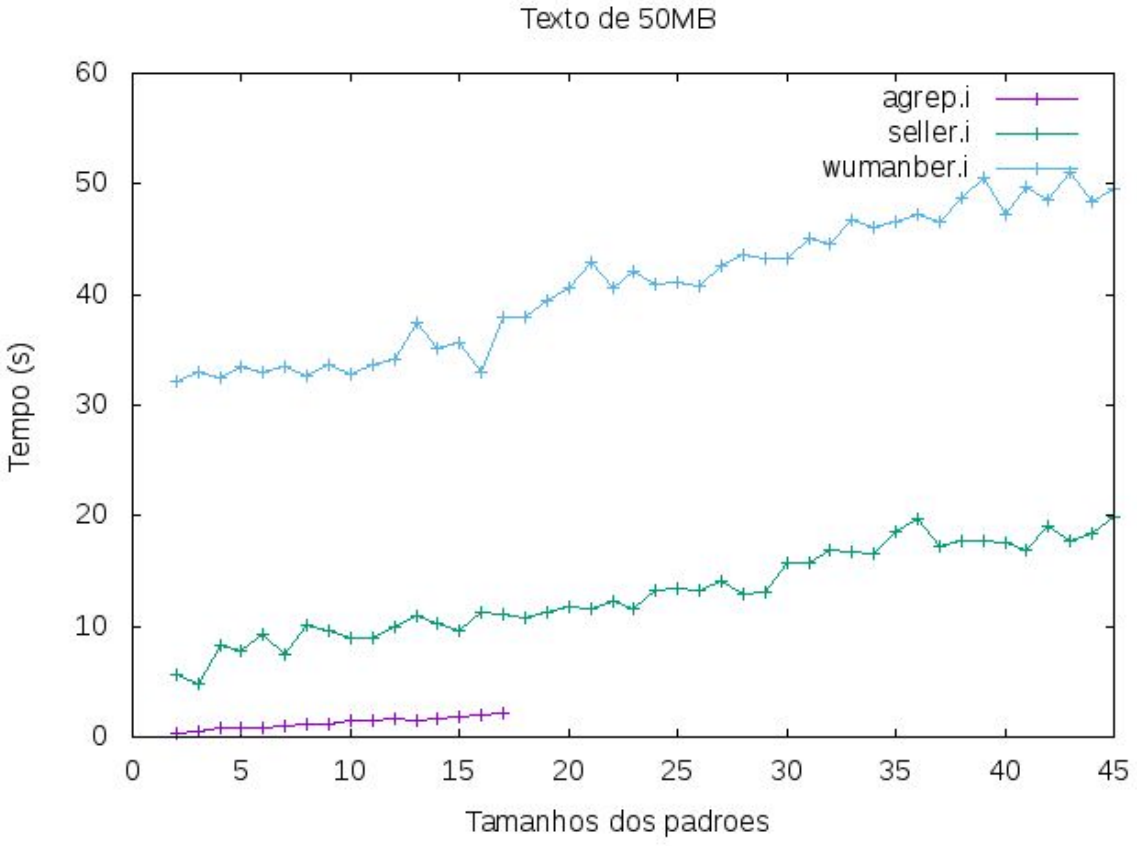
2.2 Testes em busca aproximada

Já para a busca aproximada, utilizamos os mesmos conjunto de dados e métodos da exata, com a diferença do intervalo do tamanho de padrões que foi de 2 a 45 e da adição da taxa de erro, que optamos por ser o (tamanho do padrão + 1)/2, pois assim ficamos com resultados entre o caso médio e o pior caso (que seria uma taxa de erro do tamanho do padrão - 1). O sellers teve algumas saídas erradas, porém ele se aproxima do resultado correto, o que nos fez pensar que sua versão 100% nao teria um desempenho muito diferente e que valeria a pena utilizá-lo para comparação.

Curiosamente, o agrep tem uma limitação de máximo de erros, que parece variar de acordo com sua implementação[3]. No ambiente que testamos, o agrep era limitado a um erro máximo de 8.

[3] https://linux.die.net/man/1/agrep, acesso em 15 de outubro de 2017.


Texto de 5MB
agrep.i
seller.i
wumanber.i
Tempo (s)
Tamanhos dos padroes
0
1
2
3
4
5
6
0
5
10
15
20
25
30
35
40
45


Texto de 50MB
agrep.i
seller.i
wumanber.i
Tempo (s)
Tamanhos dos padroes
0
10
20
30
40
50
60
0
5
10
15
20
25
30
35
40
45

2.3 Conclusões

Para a busca exata, percebemos que o tempo do grep e do shiftor tendem a nao mudar muito com o aumento dos padrões (pelo menos até o tamanho 80), houve um gasto de tempo maior para ambos no começo, porém verificamos que isso foi devido a maior ocorrência dos padrões iniciais (by, the, ...).

E na busca aproximada, foi notável que o aumento do tempo ocorreu em função da taxa de erro máximo e dos tamanhos dos padrões que foram aumentando.

Vimos que o desempenho do Wu Manber foi muito pior que o dos outros 2, possivelmente devido a taxa de erros que era alta para ele e o grep ficou muito à frente de todos em termos de desempenho, um dos fatores que acreditamos ter contribuído, mas não tão expressivamente, foi a leitura linha a linha que não tinha um buffer de tamanho explícito para leitura.

Verificamos também que ao compilar nosso programa com o parâmetro -O3 no g++, para que o gcc faça ainda mais otimizações que o normal, o desempenho do programa melhorou consideravelmente.