UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PMT

Primeiro Projeto de Processamento de Cadeias de Caracteres

Equipe

Carlos Henrique Gonçalves e Silva (chgs)

Pedro Tiago de V. S. R. Araújo (ptvsr)

25 de Outubro de 2015

Sumário

1. Identificação

1. **Identificação da Equipe**
    a. Carlos Henrique Gonçalves e Silva (chgs)
    b. Pedro Tiago de Vasconcelos Souza Rangel Araújo (ptvsr)

2. **Descrição de Contribuição**

Ambos os integrantes tiveram participação em todas as partes da implementação do projeto, tanto no “front-end” quanto no “back-end”.

Sendo mais detalhado, em relação aos algoritmos vistos em sala, Carlos foi o responsável pela implementação do *Aho-Corasick*, enquanto Pedro foi o responsável pela implementação dos algoritmos *Boyer-Moore* e *Sellers*.

Detalhando o “front-end”, ambos tiveram participação igual quanto à classe base do projeto, a ***pmt.cpp***, tanto na parte de leitura de arquivos e escrita dos resultados das pesquisas, como também o tratamento de opções/argumentos passados através da linha de comando (CLI).

2. Implementação

a. Descrição Geral do Sistema

O nosso sistema faz buscas exatas e aproximadas, em um ou mais arquivos de texto. Podendo ser executado através do seguinte comando :

$ ./bin/pmt [options] pattern textfile [textfile...]

As options podem ser:

a) **-e/--edit** (no caso de uma busca aproximada) seguida do tamanho da distância de edição;

b) **-p/--pattern** seguido do nome do arquivo exato, ou utilizando *wildcards* para arquivos que contêm os padrões a serem procurados.

c) **-c/--color** para a impressão dos resultados com o padrão destacado na cor vermelha.

Para pesquisar em mais de um arquivo de texto com o mesmo comando, basta adicioná-los ao fim do comando, ou usar *wildcards*.

i. Algoritmos Implementados

- Boyer-Moore
  Algoritmo utilizado quando usado para busca exata.
- Aho-Corasick
  Algoritmo utilizado quando usado busca exata e a opção **-p/--pattern** é ativada. Ou seja, quando procurado usando, possivelmente, mais de uma pattern.
- Sellers.
  Algoritmo utilizado quando usado para busca aproximada.

ii. Detalhe de implementação relevantes

- Estrutura de Dados:
  - Para o caso das estruturas de dados, foram utilizados primariamente *vector* e *set*. Apesar de não serem as mais eficientes para tal uso, conseguimos garantir que não exista um limite de buscas para o usuário, tanto com padrões quanto textos. O uso de *set* é feito para garantir que o usuário não faça a busca de um mesmo padrão mais de uma vez, visto que isso seria possível caso existisse o mesmo padrão em arquivos diferentes com o uso de wildcards, por exemplo.

- Estratégia de leitura das entradas
    - A leitura dos arquivos de texto é feita utilizando a função padrão do C++, *getline().* Nós lemos o arquivo de entrada linha a linha e passamos para o algoritmo correspondente. No caso de ser um arquivo com os padrões, eles são colocados em um *set* para assegurar que um padrão não vai ser colocado mais de uma vez e em seguida é passado ao algoritmo.

- Heurísticas para combinação dos algoritmos
    - Os algoritmos que utilizamos foram escolhidos com certa facilidade. O algoritmo **Boyer-Moore** foi escolhido por ser o mesmo utilizado pela ferramenta **GNU grep**. O algoritmo **Aho-Corasick** foi escolhido por sua capacidade de pesquisar mais de um padrão no texto ao mesmo tempo. Já para a busca aproximada, **Sellers** foi escolhido por, além de dificuldades de implementaçao que tivemos no Ukkonen e Wu-Manber, tinha um desempenho aceitável.

b. Detalhes de implementação relevantes

Foram utilizadas duas ferramentas auxiliares no desenvolvimento do projeto, sendo elas:

1. **GNU getopt**: ferramenta utilizada para obter as opções e argumentos passados ao programa através da interface em linha de comando (CLI).
2. **GNU glob**: ferramenta utilizada para tratar o casamento de *wildcards*. Vale notar que se foi feita uma modificação no funcionamento básico da ferramenta para que melhor se adequasse ao que nosso projeto propõe. Dada essa modificação (*override/inline*), o programa garante a utilização de *wildcards* tanto para arquivos de textos quando para arquivos de padrões, mas como sempre há um *trade-off*, o desempenho em geral foi um pouco prejudicado.

c. Bugs Conhecidos e limitações

Se foi detectado que nosso algoritmo aproximado (**Sellers**) tem seu resultado mais próximo do **agrep** quando o erro máximo tem diferença de 1. Por exemplo, no **pmt** com e_max 1, o resultado é mais próximo do **agrep** com e_max 2.

No ultrabook Samsung o **Aho-Corasick** sempre dá a resposta correta, porém está travando com os arquivos *english*, excetuando-se o *english.50MB*. No MacBook Pro, ele executa até o final.

Durante os testes, o algoritmo **Boyer-Moore**, enquanto testado com os arquivos *english* até o de 200MB, todos os resultados deram exatos. Quando testado com o arquivo o

english.1024MB e *english*, de 2048MB, das mais de 7 milhões de linhas no arquivo de saída, houveram apenas 6 diferentes.

3. Testes e Resultados

a. Descrição dos dados e ferramentas de comparação utilizados

i. Descrição dos Dados

Os dados utilizados para testes (dna e english) e comparações foram retirados do site Pizza&Chili, descrito na especificação do projeto. Os seguintes arquivos foram utilizados nos testes:

- dna
- dna.50MB
- dna.100MB
- dna.200MB
- english
- english.50MB
- english.100MB
- english.200MB
- english.2014MB
- Harry_Potter_1.txt
- Harry_Potter_2.txt
- Harry_Potter_3.txt
- Harry_Potter_4.txt

ii. Ferramentas de Comparação

Para comparação de resultados, foi utilizada a ferramenta ***diff*** do próprio Linux. Essa ferramenta compara dois arquivos linha a linha e diz se há diferenças entre os mesmos.

b. Descrição do ambiente de testes

Como ambiente de teste, foi utilizado um ultrabook Samsung utilizando um processador Intel Core i5 2467M de 1.60GHz e 8GB de memória RAM, e um MacBook Pro utilizando um processador Intel Core i5 4258U de 2.40GHz e 8GB de memória RAM.

O Sistema Operacional utilizado em ambos os computadores foi o Ubuntu 14.04 LTS 64-bit.

c. Descrição dos experimentos realizados

Todos os arquivos acima descritos foram executados nas ferramentas **grep** (para o matching exato), **agrep** (para o matching aproximado) e **pmt**, ferramenta desenvolvida como projeto da cadeira de Processamento de Cadeias de Caracteres.

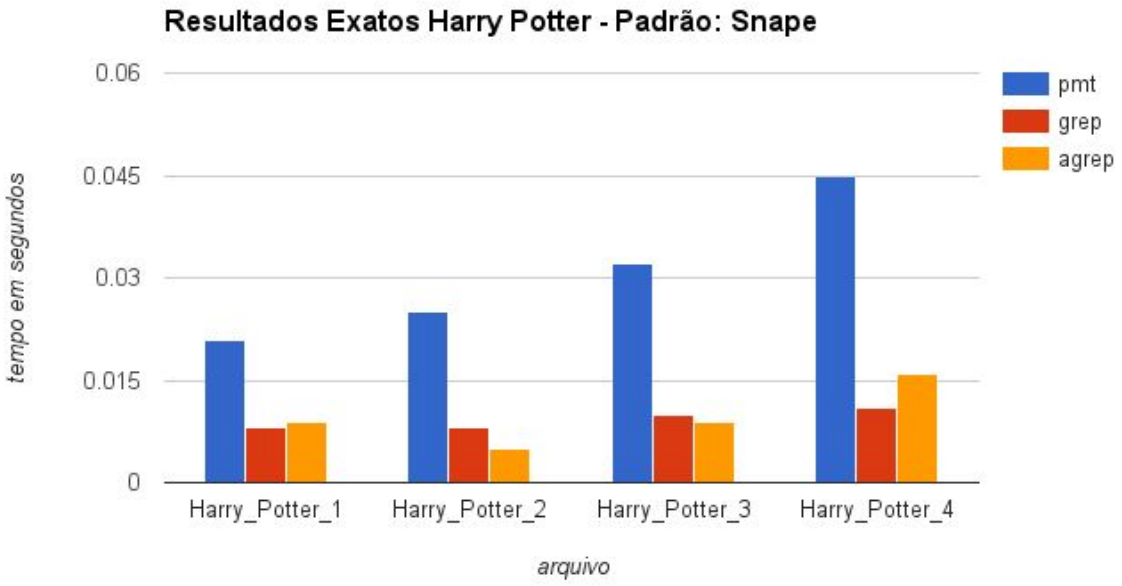
Todas as execuções dos testes foram feitas utilizando o mesmo comando para todos os programas:

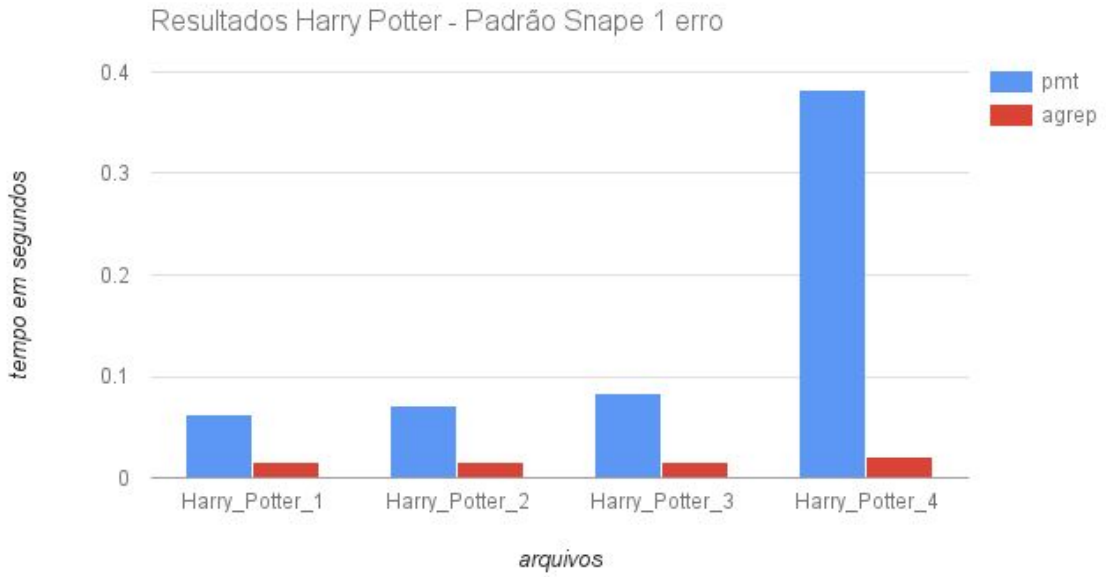
$ time [ferramenta] [opção] [padrão] [arquivo texto] > [arquivo saida]

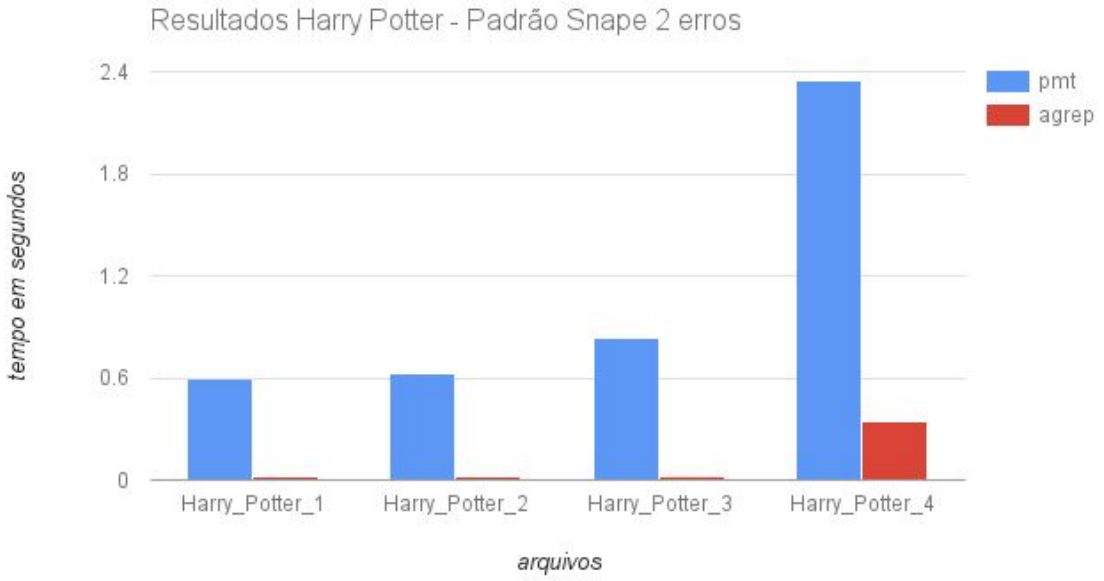
onde **time** é a ferramenta do Linux para cálculo do tempo de execução, **ferramenta** é o programa utilizado (grep, agrep ou pmt), **opção** é para o caso de matching aproximado, onde se da a distância de edição máxima (e_max), **padrão** é o padrão a ser buscado no arquivo **arquivo texto** e **arquivo saida** é para onde será mandado o output das ferramentas para comparação a partir da ferramenta **diff**.

d. Dados e Resultados

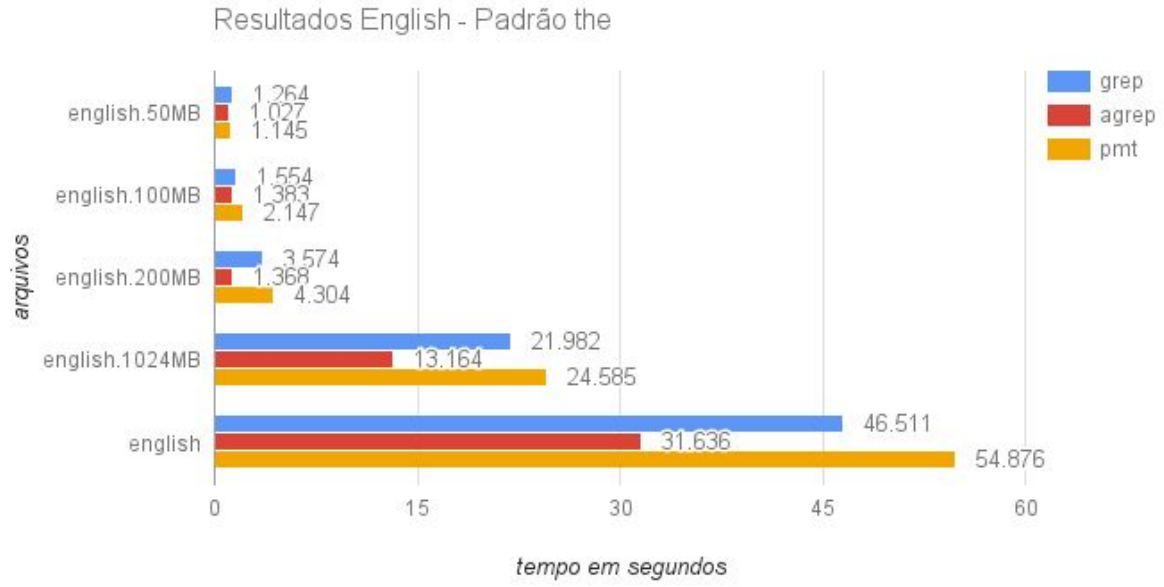
- Arquivos Harry Potter





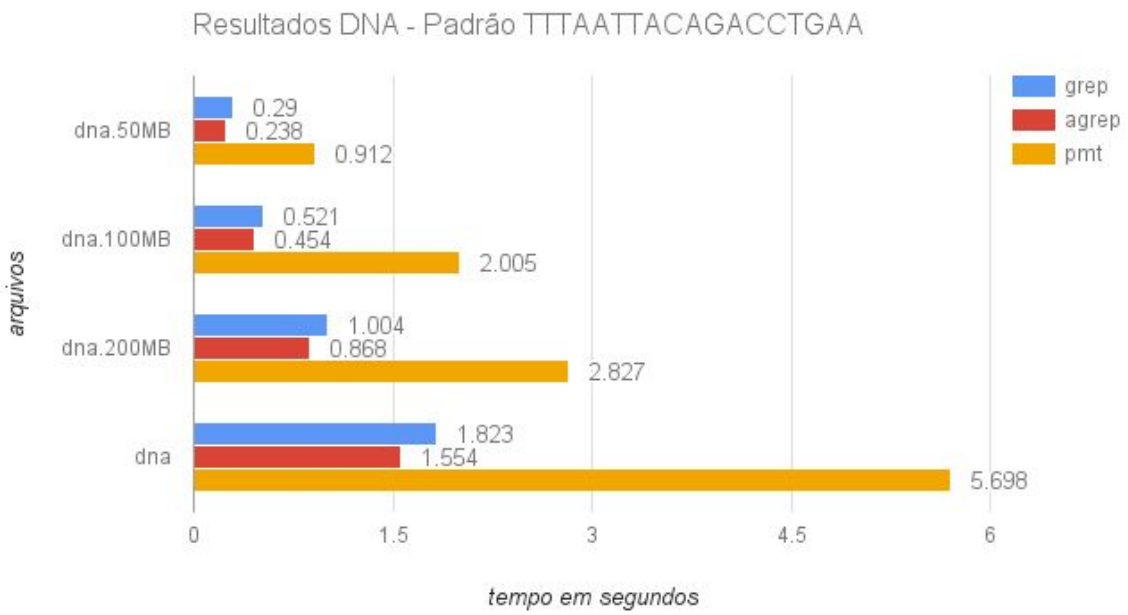


- Arquivos English



Olhar o tópico “Bugs Conhecidos e limitações” para mais informações à respeito dos arquivos English.

- Arquivos DNA



e. Conclusão

Como conclusão, vimos o quão otimizados são os algoritmos nativos do Linux para *string matching*. Mesmo utilizando um algoritmo de tempo linear quanto ao tamanho do texto e do padrão, Aho-Corasick com tempo O(n + m), não conseguimos chegar nem perto do desempenho do **grep**, que utiliza o algoritmo de Boyer-Moore. Quanto ao desempenho aproximado, utilizamos um algoritmo de tempo O(m*n), Sellers, em comparação ao Wu-Manber utilizado no **agrep**.

Todos os resultados dos testes podem ser encontrados lo link [3] nas referências.

4. Referências

[1] http://pizzachili.dcc.uchile.cl/texts.html
[2] http://cdn.preterhuman.net/texts/literature/books_by_title/G%20-%20M/

[3]
https://docs.google.com/a/cin.ufpe.br/spreadsheets/d/1WnemlumyXNpUkwh8ba3_gUQk37BHi54Uj3FfRkiN_Hk/edit?usp=sharing