Universidade Federal de Pernambuco
Centro de Informática

Relatório do primeiro projeto: Processamento de cadeias de caracteres

Professor:
Paulo Gustavo Soares Fonseca (paguso)

Sumário

Identificação

Integrantes:
Daniel de Jesus Oliveira (djo)
Rafael Nunes Galdino da Silveira (rngs)
Túlio Paulo Lages da Silva (tpls)

Daniel responsável pela implementação do Sellers e KMP.
Rafael responsável pela implementação do Aho-Corasick.
Túlio responsável pela implementação do Boyer-Moore.
Todos colaboraram na construção da infraestrutura de execução e do relatório.

Execução

Instruções de compilação

Para compilar o projeto, basta executar o comando `make` ou `make pmt`, e para executá-lo, rodar o executável `pmt` que será gerado no diretório `bin`. Para mais informações sobre o uso do programa, a opção `--help` está disponível.

Instruções de execução

Exemplo de uso:

`pmt [opções] [padrão] arquivo_de_entrada [outros_arquivos_de_entrada ...]" << endl;`

Opções:

- Configura a distância máxima de edição
  - `-e , --edit=< edit distance>`
- Especifica o arquivo de onde serão lidos os padrões a serem buscados (um por linha)
  - `-p , --pattern=< pattern file>`
- Habilita a opção *quiet* e o programa não imprimirá as saídas (útil para análise de tempo)
  - `-q, --quiet`
- Mostra uma versão em inglês destas instruções
  - `-h , --help`

Se um arquivo com padrões não for especificado, o primeiro argumento após as opções dado ao `pmt` será interpretado como o único padrão a ser buscado. Vários arquivos de entrada podem ser especificados.

Implementação

O pmt implementado pelo grupo inclui a implementação dos algoritmos *Knuth-Morris-Pratt* (KMP), *Boyer-Moore* (BM) e *Aho-Corasick* (AC) para casamento exato de padrões, e *Sellers* para casamento aproximado. De acordo com os parâmetros fornecidos na execução da aplicação, o algoritmo é escolhido:

- ❏ Se for especificada uma distância máxima de edição maior que zero, o algoritmo ***Sellers*** é escolhido;
- ❏ Caso contrário, se mais de um padrão for especificado, o algoritmo ***AC*** é escolhido;
- ❏ Caso contrário, se a opção -k estiver habilitada o algoritmo ***KMP*** é escolhido;
- ❏ Caso contrário, o algoritmo ***BM*** é escolhido.

O programa realiza a busca por todo o arquivo de entrada, fornecendo a saída apenas ao concluir a busca. Ao finalizar a execução da busca, a informação sobre as ocorrências encontradas é exibida, mostrando o número da linha da ocorrência e a posição onde a ocorrência começa (no caso de busca exata) ou termina (no caso de busca aproximada).

Estruturas de dados

Para o projeto como um todo uma ocorrência é representada por uma estrutura de classe chamada Occurrence que contém informações sobre a linha, posição na string e erro (se tiver) da dada ocorrência. Os algoritmos *KMP, BM* e *Sellers* não requerem nenhuma estrutura de dados especial nas suas formas mais simples, então só foram usadas estruturas triviais como arrays, strings e a classe vector da STL.

Para a implementação do algoritmo *Aho-Corasick*, além das estruturas triviais já citadas foi implementada também uma estrutura de Nó que tem conhecimento da sua transição de falha, transições válidas e uma lista de suas ocorrências. A lista de ocorrências de um nó foi implementada como uma lista encadeada, dessa forma as ocorrências de um nó podem ser encadeadas com as obtidas através das transições de falha.

Leitura das entradas

Para a leitura das entradas, foi criada uma classe chamada FileReader. O intuito de sua criação foi fornecer uma interface através da qual se pudesse ler facilmente os caracteres do arquivo aos poucos (lendo do arquivo uma quantidade limitada de caracteres de cada vez), assim tornando o consumo de memória do sistema um problema irrisório.

A adaptação dos algoritmos para lidar com o fato da leitura de cada linha não ser feita de uma vez trouxe dois problemas principais: o aumento da complexidade do código em si e

um overhead (devido às repetidas leituras e ao tratamento de casos problemáticos) que afeta negativamente a performance do algoritmo.

Testes e Resultados

Para os testes implementamos um resumido sistema com o intuito de coletar informações acerca da performance do pmt implementado.

O parâmetro para a comparação de performance do sistema foi o grep. Para tal, foram aferidos o tempo de cada algoritmo implementado por nós (KMP, BM, Aho e Sellers) e o tempo de execução do grep para os mesmos inputs. E os resultados foram comparados na tabela e gráfico a seguir:

Cada Os testes foram executados com os seguintes inputs:

- *Padrões:* aleatórios e de *tamanho* crescente na ordem
  - {T | 2^x, x está no intervalo [1, 4]}
- *Dataset*: percorreu-se todos os arquivos disponíveis no link abaixo como arquivo de texto
  - Dna (50MB até 200MB): http://pizzachili.dcc.uchile.cl/texts.html

Os resultados estão compilados nos gráficos a seguir:

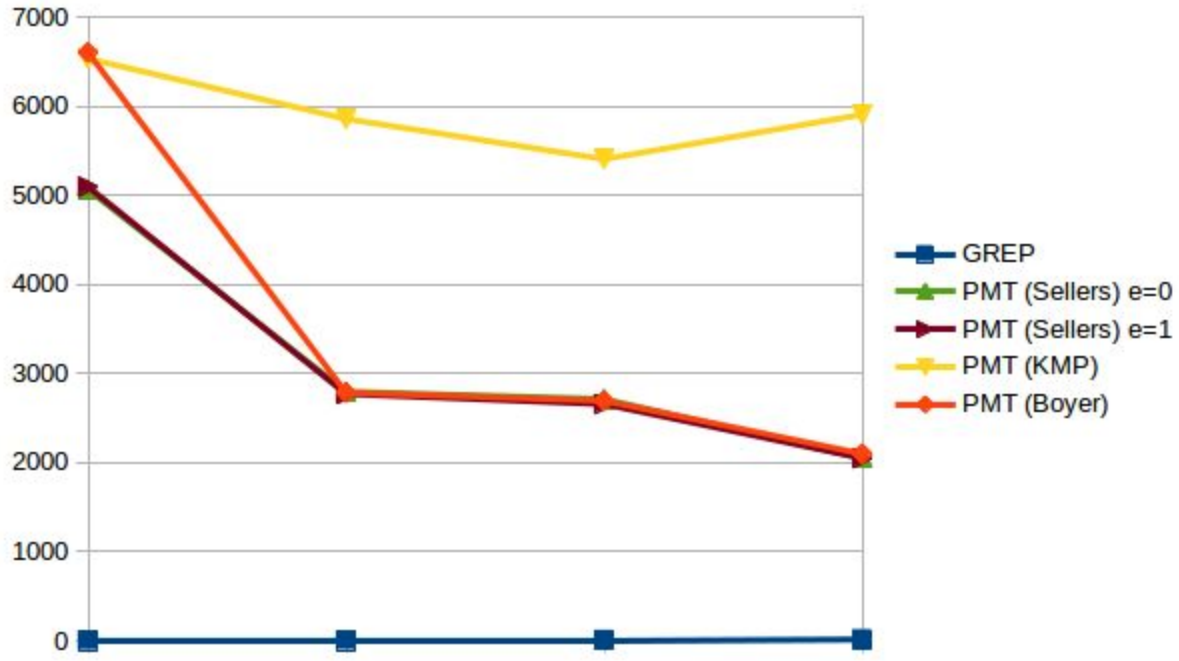

Imagem 1: Testes utilizando a base dna.50MB com padrões de tamanho 2, 4, 8, e 16

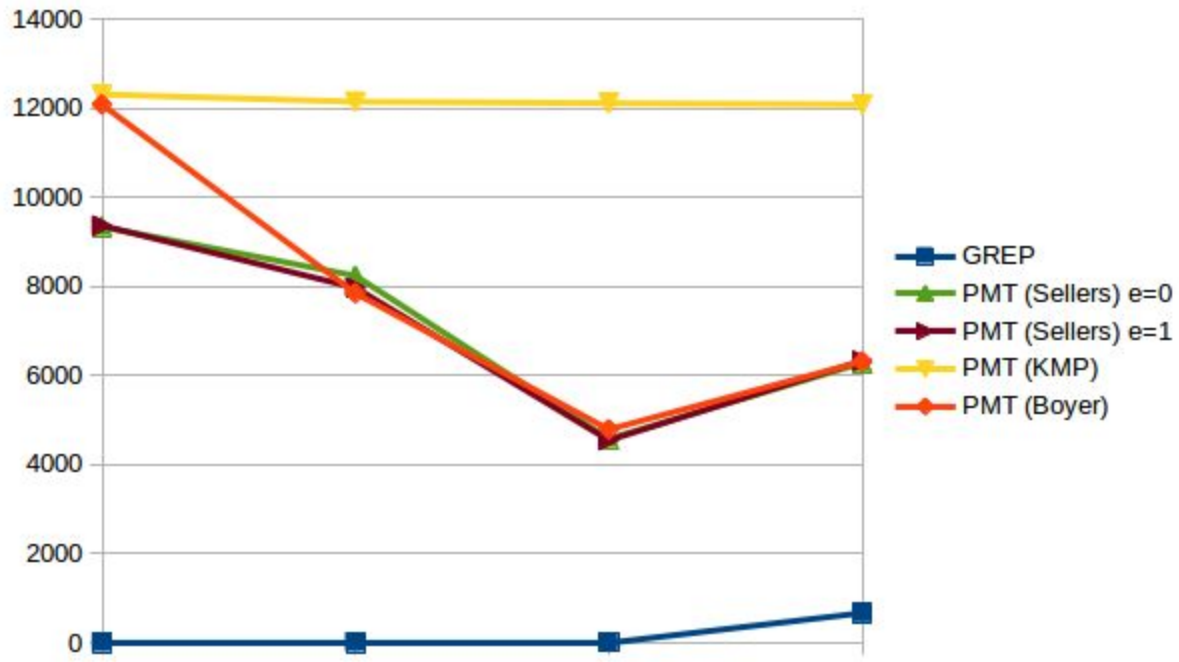

Imagem 2: Testes utilizando a base dna.100MB com padrões de tamanho 2, 4, 8, e 16

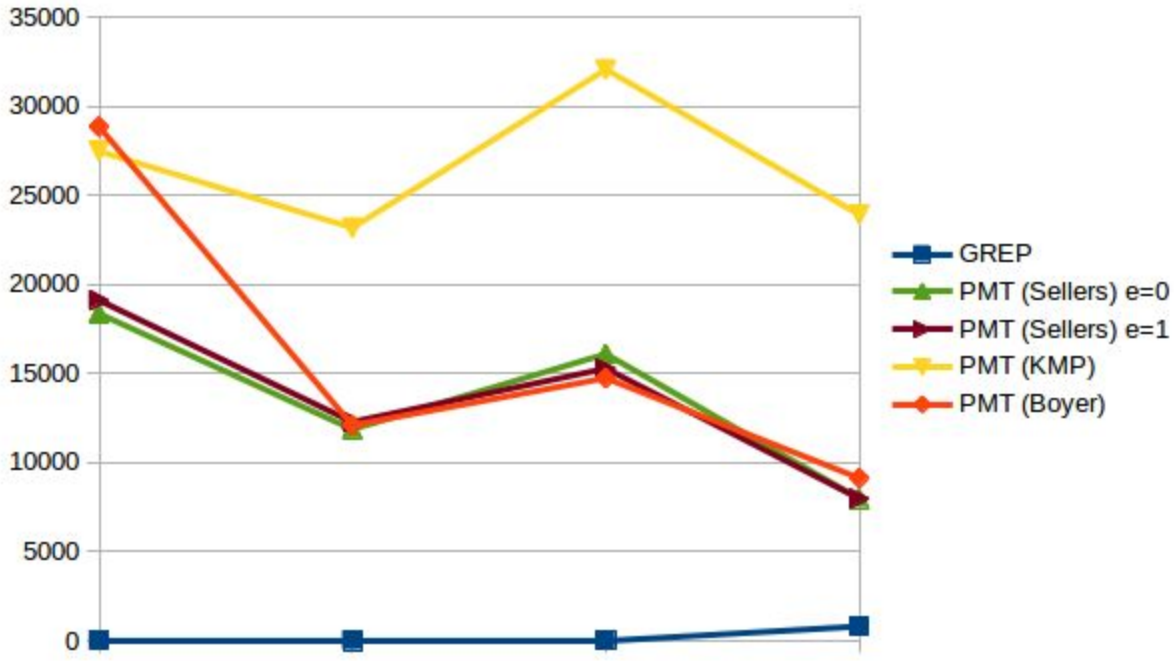

Imagem 3: Testes utilizando a base dna.100MB com padrões de tamanho 2, 4, 8, e 16

Apesar de a implementação ainda deixar bastante a desejar, é possivel verificar como a implementaçao do Boyer Moore fica mais rápida de acordo com o crescimento do tamanho do padrão.