

IPMT - Indexed Pattern Matching Tool

Nome da ferramenta: IPMT

Grupo: Danilo Neves Ribeiro (dnr2)
Vitor Hugo Antero de Melo (vham)

Descrição do funcionamento da ferramenta

A ferramenta IPMT foi desenvolvida utilizando a linguagem de programação C++. Implementamos os seguintes algoritmos estudados em sala de aula que serão separados em duas categorias:

Algoritmo de indexação:

Árvore de Sufixos (Ukkonen)

As árvores de sufixo, baseada no algoritmo de Ukkonen [1] como vistas em sala, são mais eficientes na complexidade de tempo para construção de índices. Utilizamos esta opção para fazer a indexação do texto e ela se mostrou eficiente. Em [2] há um detalhamento da construção da árvore, e nosso algoritmo foi baseado neste site.

Array de Sufixos

O array de sufixo implementado foi baseado no algoritmo tradicional introduzido por Manber & Myers (1990), e tem a complexidade de construção em $O(n * \log(n))$ - onde n é o tamanho da string – e de busca de $O(m * \log(n) + occ)$ - onde m é o tamanho do padrão e occ é o número de ocorrências. Apesar de ter uma construção e busca um pouco mais lento o algoritmo implementado tem a vantagem de consumir um pouco menos de memória do que a árvore de sufixos. A implementação do algoritmo foi baseada na descrição contida no livro em [3].

Algoritmo de compressão/descompressão:

LZW

Para realizar a compressão e descompressão dos arquivos e das estruturas de dados, utilizamos o algoritmo de Lempel-Ziv-Welch (LZW), que é comumente utilizado em compressão de arquivos GIF. Nós seguimos as dicas de implementação encontradas em [4]. Utilizamos a estrutura *map* para o dicionário tanto na compressão quanto na descompressão.

Detalhes de implementação

Os arquivos de entrada são carregados completos na memória, em uma única string. Decidimos utilizar essa estratégia pois é necessário varrer a string de entrada para criar a árvore de sufixo de uma única vez.

A implementação do projeto segue a estrutura descrita na especificação. Os arquivos do código fonte se encontram na pasta *src/*, descritos a seguir:

ipmt.cpp

Este é o arquivo de interface com o usuário, ele importa um segundo arquivo que contém a implementação dos algoritmos, *ipmt_algorithm.h*. Seus objetivos incluem imprimir a ajuda, fazer *parse* nos argumentos de entrada, lidar com *wildcards* etc. Ele também verifica se a execução deve ser do tipo *index* ou *search*, e chama as funções correspondentes.

ipmt_algorithm.h

Define algumas constantes e as assinaturas dos métodos que são implementados em *ipmt_algorithm.cpp*.

ipmt_algorithm.cpp

Este arquivo contém os algoritmos de compressão e descompressão, e a lógica que chama as funções para criar índices e executar buscas. Implementa as funções definidas em *ipmt_algorithm.h*, onde as principais funções são *ipmt_index_**() e *ipmt_search()*, que servem de ligação entre a interface com o usuário (*ipmt.cpp*) e suas funções.

SuffixTree.cpp

Por questões de organização foi decidido criar uma estrutura para comportar todas as variáveis que definem a árvore de sufixos, bem como funções e variáveis que ajudam a construí-la. Caso essa estrutura não fosse criada seria necessário passar várias estruturas e *arrays* por parâmetro, ou fazer tudo em um único método, o que avaliamos inadequado. A estrutura da árvore foi representada por um array de arestas, já que desta forma seria mais direto a codificação e decodificação. Além disso foi utilizado a estrutura *map* de *c++* para mapear quais as arestas que saiam de um determinado nó, considerando um determinado caractere (esta estrutura é feita para busca em tempo logarítmico, talvez usar um *hashmap* aumentaria a performance). Para serializar a árvore criamos uma string, onde o primeiro inteiro representa o número de nós na árvore, e os demais *n-1* inteiros representam as arestas (com informação do nó inicial, no final label da aresta).

SuffixArray.cpp

Para construção do array de sufixo foi criado os arrays para armazenar os sufixos, os rankings e arrays temporários. A busca dos padrões é feita fazendo uma busca binária sobre o índice do array do sufixo.

Bugs e limitações

Observamos que para muitos arquivos pequenos a compressão fica maior do que o arquivo original. Isso ocorre devido à baixa quantidade de padrões repetidos. Outra limitação ocorre no tamanho do dicionário não ser limitado, o que pode causar estouro de pilha dependendo do tamanho da entrada e da memória disponível durante a execução.

Outro bug foi que quando compilado no Linux a árvore joga um exceção na hora de dar free na memória alocada. Este erro não ocorre no Windows (sistema operacional que utilizamos para fazer o código). Para solucionar o problema tivemos que forçar o encerramento do programa. Isso não afeta o resultado final, contudo não é possível indexar mais de um arquivo de texto na mesma execução.

Referências

- [1] - E. Ukkonen. On-line construction of suffix trees. Algorithmica, 14(3):249-260, September 1995.
- [2] - <http://marknelson.us/1996/08/01/suffix-trees/>
- [3] – S.Halim. Competitive Programming 3 – The new Lower Bound of Programming Contests. 2014.
- [4] - <https://www.cs.duke.edu/csed/curious/compression/lzw.html>