TPF - The Pattern Finder

Nome da ferramenta: TPF

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

Descrição do funcionamento da ferramenta

A ferramenta TPF (The Pattern Finder) foi desenvolvida utilizando a linguagem de programação C++. Implementamos os seguintes algoritmos estudados em sala de aula que serão separados em duas categorias:

Algoritmos de busca exata:

Boyer-Moore

Como foi visto em sala, o Boyer-Moore é um algoritmo eficiente, utilizado pelo próprio *grep*. Utilizamos o Boyer-Moore nos casos de busca exata, onde um único padrão é procurado no texto.

Aho-Corasick

Para mais de um padrão se fez necessário usar um algoritmo que, mesmo sendo um pouco mais complexo, consiga lidar com os padrões de forma eficiente. Sempre que há mais de um padrão a ser procurado, a ferramenta utiliza o Aho-Corasick para fazer o *pattern matching*.

Algoritmos de busca aproximada:

Wu-Manber

Para busca aproximada utiliza-se o Wu-Manber para entradas onde o erro não é muito significativo e o tamanho do padrão é consideravelmente grande. Em nossa implementação do Wu-Manber tivemos que criar um estrutura de dados (descrita na próxima parte do relatório) para generalizar um conjunto de bits, tal estrutura foi chamada de Bitset.

Sellers

Vimos que apesar de ser simples o algoritmo de Sellers funciona eficientemente para padrões de tamanho razoavelmente pequenos. Ele também tem a vantagem de sua complexidade não depender do valor máximo do erro. Desta forma o algoritmo de Sellers é usado sempre que o valor do erro é grande ou o tamanho da entrada é pequeno.

Detalhes de implementação

A leitura dos arquivos de entrada foi feita linha por linha. Decidimos utilizar essa estratégia pois é necessário imprimir a linha em que o padrão foi achado no arquivo e dessa forma não precisaríamos fazer um pós processamento.

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

tpf.cpp

Este é o arquivo de interface com o usuário, ele importa um segundo arquivo que contém a implementação dos algoritmos, *tpf_algorithm.h* (motor de busca). Seus objetivos incluem imprimir a ajuda, fazer *parse* nos argumentos de entrada, lidar com *wildcards* etc. Após isso ele irá decidir se a busca é exata ou não e chamar a função *tpf_find()* definida em *tpf_algorithms.h*.

tpf_algorithm.h

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

tpf_algorithm.cpp

Este arquivo contém o motor de busca em si. Implementa as funções definida em *tpf_algorithm.h*, onde a principal função é o *tpf_find()*, que serve de ligação entre a interface com o usuário (*tpf.cpp*) e suas funções. O parâmetro *tpf_type* define qual o tipo de busca e determina qual algoritmo será utilizado.

TreeAhoCorasick.h

Por questões de organização foi decidido criar uma estrutura para comportar todas as variáveis que definem a árvore do Aho-Corasick, 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.

Bitset.h

Da mesma forma, criamos uma estrutura de dados para representar o conjunto de bits usado pelo Wu-Manber de forma mais otimizada. Sabíamos que o C++ contém em sua biblioteca padrão a estrutura *bitset*[1], contudo este container tem a limitação de ter o tamanho determinado em tempo de compilação, o que tornou necessário uma reimplementação da estrutura.

[1] Referência do *bitset* - http://www.cplusplus.com/reference/bitset/bitset/bitset/

Bugs e limitações

Os bugs e erros encontrados durante a implementação da ferramenta foram consertados em tempo devido. Acreditamos que agora a ferramenta funciona de acordo com o esperado.