Ferramenta IPMT (Indexed Pattern Matching Tool)

Trabalho da Disciplina IF 767 - Processamento de Cadeias de Caracteres

Professor Paulo Fonseca - paguso@cin.ufpe.br

Alunos:

Lucas de Souza Albuquerque - lsa2@cin.ufpe.br

Vítor Travassos Castelo Branco - vtcb@cin.ufpe.br

Introdução e Identificação

Este relatório tem como intenção detalhar o processo de desenvolvimento e algumas escolhas tomadas na implementação de uma ferramenta *ipmt* para indexação, armazenagem e busca de padrões em um arquivo de texto. Mencionamos a lógica por trás das implementações dos diferentes algoritmos de indexação e compressão/descompressão, e o método de busca de padrões utilizando-se o índice, analisando suas aplicações e seus prós e contras. Por questão de tempo, a versão atual do relatório não possui a geração de testes e posterior análise dos resultados e estatísticas obtidos, e isto seria o próximo passo óbvio para o projeto.

A ferramenta segue as regras e diretrizes definidas na especificação do projeto, sendo desenvolvida em C++ e possuindo uma **CLI,** ou **Interface em Linha de Comando**, pela qual o usuário interage com suas funções. A ferramenta possui dois modos principais de funcionamento: o primeiro destes modos é o modo **INDEX**, que recebe pelo menos um arquivo de texto e realiza os algoritmos de indexação e compressão sobre este arquivo, gerando um arquivo índice *.idx*. O segundo dos modos é o modo **SEARCH**, que recebe um arquivo índice e pelo menos um padrão, e procura as ocorrências destes padrões usando o índice pré-processado associado ao arquivo *.idx*.

Além destas funcionalidades básicas, o usuário pode, através de comandos opcionais adicionais, controlar alguns parâmetros da indexação/busca e acessar as seguintes funcionalidades extras da ferramenta:

- **-h, --help,** que mostra uma pequena documentação sobre a ferramenta com parte das informações aqui mencionadas, para ajudar o usuário na utilização do programa por meio da **CLI**.
- **-i, --indextype,** que permite o usuário à especificar o tipo de indexação que ele deseja no programa. Como esperado, este comando adicional é restrito para o modo **INDEX.**

- **-z, --compression,** que permite o usuário à especificar o algoritmo que será usado para comprimir o arquivo de texto para posterior busca de padrões offline. Como esperado, este comando adicional é restrito para o modo **INDEX.**
- E por último, **-p, --pattern**, que permite o usuário a fornecer um arquivo com múltiplos padrões (sendo um padrão por linha) para busca de suas ocorrências em um arquivo índice. Como esperado, este comando adicional é restrito para o modo **SEARCH.**

É importante notar que, por ora, só tem um índice implementado **(Suffix Array)** e um algoritmo de compressão **(LZ77)**, e consequentemente as flags estão sendo ignorados, mas sua existência é importante para intuitos de melhorar a escalabilidade do projeto, e, possivelmente, em um tempo futuro, expandir a ferramenta e adicionar novos algoritmos sem ter que mexer em todas as etapas do fluxo de funcionamento do programa e diminuir a probabilidade de erros e conflitos. Esta filosofia também se encontra em outras partes do programa, como por exemplo, na existência de headers para índices (**index.h**, com métodos de construção, busca, serialização…) e codificadores (**encoder.h**).

Considerando então todas as funcionalidades, a ferramenta então pode ser executada de três maneiras:

- `ipmt index [OPTION]... [TEXTFILE]`**,** para indexar um arquivo de texto.
- `ipmt search [OPTION]... [PATTERN] [INDEXFILE]`**,** para procurar as ocorrências de um padrão em um arquivo de índice *.idx.*
- E, por último, `ipmt search [OPTION]... -p [PATTERNFILE] [INDEXFILE]`, para procurar as ocorrências de vários padrões de um arquivo *patternfile* em um outro arquivo de índice *.idx*.

O projeto então inclui um *flag parser* e um *input parser* para analisar o input do usuário na **Interface em Linha de Comando** e chamar os algoritmos que correspondem a funcionalidade desejada.

A ferramenta foi desenvolvida em conjunto pelos alunos **Lucas de Souza Albuquerque** (lsa2) e **Vítor Travassos Castelo Branco** (vtcb). A interface de linha de comando **CLI** *(command line interface)*, o fluxo básico do programa e o desenvolvimento do relatório foram feitos em conjunto por ambos os membros da equipe. A realização de testes e a plotagem e análise de resultados também serão realizadas em conjunto em um tempo posterior à apresentação.

Implementação da Ferramenta

As seções abaixo tem como objetivo detalhar a implementação da ferramenta, analisando os algoritmos implementados e mencionando seus específicos, e ressaltando decisões feitas na implementação para as diferentes etapas do fluxo de Indexed Pattern Matching *(indexação, compressão, busca…)*. Analisaremos também limitações e problemas encontrados no trabalho e possíveis melhorias para o projeto em algum momento futuro.

Algoritmo de Compressão

O algoritmo principal usado para compressão de arquivos foi o algoritmo **LZ77** (Lempel-Ziv). Ele utiliza as partes que já foram lidas de um arquivo como um dicionário, substituindo as próximas ocorrências das mesmas sequências de caracteres pela posição da sua última ocorrência. Para limitar o espaço de busca e endereçamento, as ocorrências anteriores são limitadas por uma *sliding window* que tem tamanho fixo e caminha sobre o arquivo, delimitando qual área será analisada ao redor de um certo caractere. O tamanho da janela acaba, então, sendo um dos principais fatores para ajustar a performance do algoritmo.

A complexidade encontrada para o algoritmo LZ77, então, para o melhor match *BestMatch*, foi $O(|window|^2)$), ou seja, o **tamanho da janela ao quadrado.** Na ferramenta também está incluso um código em $O(|alphabet| * |window|)$, ou seja, o **tamanho do alfabeto * tamanho da janela**, mas esta versão ainda precisaria ser mais testada para garantir que ela funciona corretamente e que ela não possui nenhum erro de implementação. Ela com certeza apresentaria melhorias significativas sobre o LZ77 original, dependente apenas de $|window|$ - exceto em casos em que o tamanho da janela deslizante, a *sliding window*, seja muito pequena. Porém, casos em que uma janela muito pequena sejam úteis são raros, já que o algoritmo LZ77 em si depende na utilização de ocorrências anteriores de sequências de caracteres no texto, o que se torna mais difícil conforme você diminui o tamanho da janela.

Como será mencionado com mais detalhes na seção de problemas e limitações, o algoritmo LZ77 implementado acabou por se mostrar ineficiente em termos de compressão, ou seja, ele acabou gerando um arquivo bem maior do que o esperado, efetivamente não conseguindo comprimir o arquivo o suficiente. Por esses motivos, não se executa a compressão sobre o arquivo indexado, chegando-se na conclusão de que o LZ77, em seu estado atual, não traz benefícios suficientes para compensar sua execução.

Algoritmo de Indexação

A estrutura utilizada para a indexação do texto foi a Array de Sufixos (*Suffix Array)*, que é uma array ordenada de todos os sufixos de uma string bem usada para indexação completas de texto e compressão de dados, sendo uma alternativa simples à *Suffix Tree* que ainda é relativamente eficiente em questão de espaço. Também tentamos implementar uma *Suffix Tree* na ferramenta, mas existem erros em sua construção e por isso a Array de Sufixos é a única estrutura de indexação utilizada por ora. O projeto, porém, apresenta escalabilidade suficiente para que uma *Suffix Tree* correta possa ser adicionada sem grandes problemas.

Para a Suffix Array implementada, a complexidade de sua construção é $O(N * log(N))$, e a sua complexidade posterior em busca é $O(M * log(N))$, sendo **N** o tamanho do texto e **M** o tamanho da padrão em questão. Não implementada ainda é uma versão da busca sobre a Suffix Array com complexidade linear, mas ao longo dos últimos dias vimos analisando sua viabilidade de implementação e seus benefícios.

Limitações, bugs e Trabalhos Futuros

Por meio de análises durante o desenvolvimento da ferramenta, percebemos uma limitação significativa relacionada ao algoritmo de compressão utilizado, o **LZ77**. Em termos de compressão, este algoritmo se mostrou ineficiente para o índice gerado pelo *Suffix Array*, resultando em um arquivo razoavelmente maior do que o arquivo esperado (e desejado). Uma explicação possível para o problema se encontra na natureza da Array de Sufixos em si: como a array é uma sequência de números inteiros distintos, eles diminuem o número de repetições no qual o **LZ77** depende, mesmo após a codificação da array em bytes. Os prós e os contras do algoritmo de compressão foram comparados, e no final, por causa deste problema, a implementação atual do programa não realiza compressão no arquivo.

Por exemplo, para um *texto* com **N** bytes e um *texto* + *suffix array* relacionada com **5N** bytes, se percebeu que o *texto + suffix array*, comprimidos pelo LZ77 quando o mesmo não encontra muitas repetições, pode chegar em até **10N** bytes. Se a compressão resultante fosse no geral menor dos que os **5N** bytes ocupados pelo texto e pelo array de sufixos, valeria a pena considerar sua execução na ferramenta, mas infelizmente este não foi o caso.

Testes e Resultados

Como mencionado, por questão de tempo, não conseguimos executar muitos testes e gerar muitos resultados para análise até o momento da apresentação adiantada do projeto. Para um projeto como esse, porém, é importante executar testes de nossa ferramenta em várias situações, para tentar caracterizar os algoritmos em funções de seus parâmetros. Também é importante comparar tais resultados de nossa ferramenta com outras, sejam elas ferramentas padrão como o *grep* e o *gzip*, ou ferramentas disponíveis em bibliotecas de terceiros, usando estas outras como o *benchmark*. Com isso em mente, seria importante realizar uma análise de resultados mais profunda em alguma oportunidade futura.