Criação de um software para geração de índices remissivos

Arthur Alexsander Martins Teodoro

[1]Instituto Federal de Minas Gerais - Campus Formiga

arthurmteodoro@gmail.com

Introdução

O trabalho proposto era a criação de um software para a criação de índices remissivos. Um índice remissivo é basicamente um arquivo que serve de referência de um texto. O índice remissivo é uma lista em ordem alfabética que lista onde cada palavra chave previamente definida se encontra no texto. Para a geração de tal, cada palavra lida no texto deve ser procurada na base de palavras chave, caso existindo, acrescentando a página(no caso deste trabalho, linha), onde tal palavra está. Porém, como visto, é necessário realizar várias buscas, o que acarreta em uma demora no processamento do software caso não seja usada uma estrutura de dados especializada em busca.

O objetivo deste trabalho é a comparação do uso de estruturas de dados para busca, logo foi desenvolvidas as estruturas tabela *hash* ou tabela de dispersão, lista encadeada, árvore binária de busca e árvore AVL. O funcionamento da árvore binária de busca consiste em manter que o filho com valor menor que o pai é inserido no folha da esquerda, e o filho com valor maior que o pai na folha da direita. Porém isso pode causar uma degeneração da árvore, uma vez, se um número relativo de entradas ordenadas forem inseridas na árvore, tal começa a se comportar como uma lista, fazendo que sua busca demore mais. Para a solução de tal problema, a inserção da árvore AVL realiza rotações, mantendo a propriedade de uma árvore binária de busca e balanceando a árvore. Uma árvore está balanceada quando a altura de suas subárvores diferem em módulo até uma unidade.

Implementação

Nesta seção deste documento será mostrada as escolhas de implementação, demonstração das estruturas de dados e demais questões de implementação.

Descrição sobre as decisões de projeto e implementação

Foi usada a linguagem de programação C para a implementação do trabalho. O trabalho foi codificado usando o editor de texto Sublime Text 3[1], foi usado o compilador GCC versão 5.4.0, sistema operacional GNU/Linux Ubuntu 16.04, e controle de versão usando método Git.

Foi criado uma TAD de lista encadeada(*lista.c, lista.h*) para o índice remissivo, uma tabela hash com Encadeamento Externo(*hashEncadeada.c, hashEncadeada.h*), uma TAD de árvore binária de busca(*arvoreBin.c, arvoreBin.h*) e uma árvore AVL(*arvoreAVL.c, arvoreAVL.h*. Além das TAD's, foi criado um arquivo que utiliza a TAD criada para criar o índice remissivo(*indice.c, indice.h*), sendo realizada neste arquivo a manipulação de strings, leitura e gravação de arquivos de saída. Foi criado um

[1]https://www.sublimetext.com/

arquivo que contém a função main() (*main.c*) que utiliza as função criadas no arquivo *indice.c*, não sendo necessário o usuário conhecer as minúcias do processo de criação do índice remissivo.

A função *hash* usada neste trabalho é uma função que multiplica o valor ASCII do caractere com o índice em que tal caractere se encontra. Esta função foi a que resultou em um melhor resultado em um trabalho realizado anteriormente. Já como tratamento de colisão, foi usado o encadeamento externo, este que também apresentou melhor resultado em trabalho anterior já citado.

Compilação e Execução

Foi criado um arquivo de *Makefile* usado para a compilação do sistema, porém este deve ser alterado dependendo da estrutura que deseja ser criada. Na compilação do arquivo *main.c*, é definida a constante *O*. Esta quando assume o valor 1, é utilizada a estrutura tabela *hash*, 2 é usada a lista encadeada, 3 a árvore binária de busca e 4, a árvore AVL.

Para a execução do software, deve ser passado por argumentos na linha de comando o nome de três arquivos, nesta ordem: a) Palavras Chave: arquivo que possui as palavras que devem ser indexadas, b) Texto: arquivo com o texto que se deseja criar o índice remissivo e c) Saída: arquivo com o índice remissivo criado com sucesso.

Após o término da execução será mostrado o tempo gasto na geração do índice remissivo, dado usado para a confecção deste documento.

Testes Realizados

Para a realização dos testes, foi usado o teste de número 5 disponibilizado pelo professor. Tal teste possui um número relativamente grande de palavras chave e um texto extenso, por isso foi o escolhido para a bateria de testes. Tais testes tinham como objetivo descobrir quais das estruturas seriam melhores na criação de um índice remissivo.

Execução	Hash com Tratamento Externo	Lista Encadeada	Árvore Binária de Busca	Árvore AVL
1	0,015211	0,033211	0,016916	0,016846
2	0,013989	0,035733	0,016496	0,016945
3	0,013835	0,03854	0,016632	0,016168
4	0,014551	0,033724	0,016706	0,01605
5	0,014572	0,035869	0,017626	0,018087
6	0,014228	0,035149	0,017098	0,016224
7	0,013819	0,03407	0,016666	0,015902
8	0,013971	0,033429	0,016446	0,02424
9	0,014073	0,033853	0,016768	0,016414
10	0,013844	0,033856	0,016848	0,015864
Média	**0,0142093**	**0,0347434**	**0,0168202**	**0,017274**
Desvio Padrão	**0,0004474419**	**0,0016321015**	**0,0003426257**	**0,0025366545**

Figura 1. Tabela com os valores de tempo das execuções do sistema

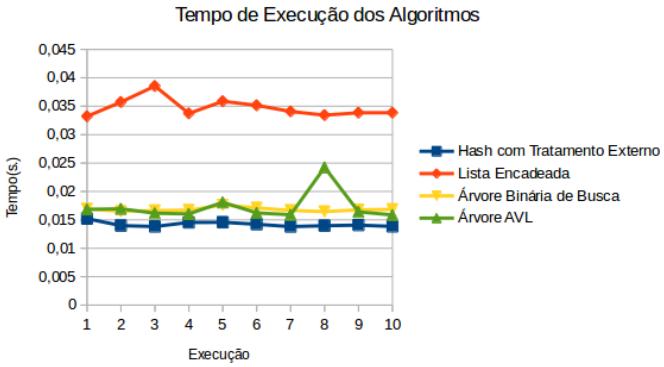

Figura 2. Gráfico dos tempos de execução

Também foi realizado testes com o arquivo texto com um número grande de linhas e o arquivo de palavras chave ordenados. Tal ordem fazia que a árvore binária de busca se degenerasse para uma lista encadeada.

Execução	Hash com Tratamento Externo	Lista Encadeada	Árvore Binária de Busca	Árvore AVL
1	0,408612	0,606808	0,722468	0,409977
2	0,398697	0,620699	0,745717	0,407966
3	0,40049	0,633559	0,750038	0,414127
4	0,402469	0,628755	0,726519	0,412051
5	0,410396	0,616839	0,739116	0,406334
6	0,419836	0,615899	0,725551	0,407636
7	0,39887	0,634629	0,754958	0,413168
8	0,406953	0,629076	0,740903	0,4103
9	0,399497	0,627735	0,735516	0,406308
10	0,40139	0,625689	0,720251	0,407862
Média	**0,404721**	**0,6239688**	**0,7361037**	**0,4095729**
Desvio Padrão	**0,0067714428**	**0,0087742472**	**0,0120875688**	**0,0028069053**

Figura 3. Tabela com os valores de tempo das execuções do sistema com entradas ordenadas

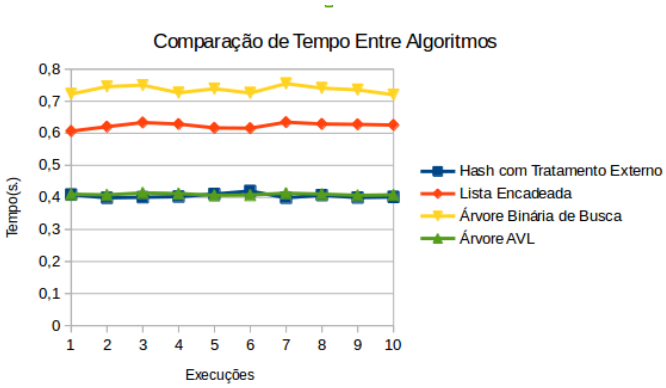

Figura 4. Gráfico dos tempos de execução com entradas ordenadas

Estudo de Complexidade

As funções de inserção quanto busca no método de hash possui complexidade média de *O(1)*, somente no caso da lista encadeada que a busca terá caso médio de *O(n)*. As funções de manipulação de string, na maioria das vezes, percorre toda a string, fazendo com que sua complexidade seja *O(n)*. A função de leitura do arquivo de palavras chave possui complexidade de O(n), uma vez que lê o arquivo de palavras até o fim. Já a função de

geração do índice remissivo é O(n), uma vez que cada linha é lida e tratada. Já o arquivo de saída também possui complexidade de *O(n*log n)*, uma vez que é necessário retornar uma lista além da ordenação do vetor para gerar o índice em ordem alfabética. O código em suma possui complexidade de *O(n*log n)* quando é usada as estruturas hash ou lista encadeada, uma vez que o algoritmo de ordenação *quicksort* possui tal complexidade.

Já nos casos das árvores, o resultado é diferente, uma fez que não é necessário a ordenação das palavras chaves, sendo necessário somente o caminhamento em tal. Os algoritmos de inserção e inserção a árvore AVL possui melhor resultado. A inserção e busca em uma árvore binária de busca possui complexidade *O(n)* no pior caso e *O(log n)* no caso médio. Já a árvore AVL, sua busca e inserção possui como pior e caso médio complexidade *O(log n)*. Isso faz com que a complexidade do sistema como um todo usando árvores é *O(log n)*.

Análise dos Resultados

Na primeira bateria de testes, pode constatar que a lista encadeada foi a estrutura que apresentou pior resultado. Muito próximo umas das outras, as árvores. Isso deve ao fato de que a árvore binária de busca não se encontrava muito desbalanceada. Por isso, como a inserção se um nó da árvore é mais lento na AVL, a AVL teve o terceiro melhor tempo médio e a árvore binária de busca se encontrou na segunda posição. A hash foi a que apresentou melhor resultado.

Já na segunda bateria de testes, com o arquivo de texto maior e palavras chaves ordenadas, o resultado foi diferente. Como esta ordem de palavras chave forçava a degeneração da árvore binária de busca, ela obteve o pior tempo. Em seguida, com o terceiro melhor tempo, a lista encadeada. A árvore AVL ficou com o segundo melhor tempo médio, muito próximo à hash. Porém, o desvio médio da AVL foi menor, o que representa que tal se manteve mais estável nos testes.

Logo se conclui que, para entradas pequenas, a AVL não é uma boa escolha, uma vez que o tempo de balanceamento sobrepõe o tempo de outras estruturas. Porém, quando não se sabe que tipo de entrada poderá ser recebida, tal estrutura é recomendada, uma vez que não existe risco da mesma se degenerá para uma lista encadeada. Porém, a tabela de dispersão se manteve como melhor estrutura. Mas tal possui como dificuldade de implementação a descoberta de uma função que cause poucas colisões.

Conclusão

Foi concluído que as duas melhores estruturas de dados para a construção de um índice remissivo é a tabela de dispersão ou uma árvore AVL. Também durante este trabalho foi reforçada a implementação de árvores, algoritmos de balanceamento e principalmente um ganho de experiência para escolhas de estruturas de dados.