Relatório de Trabalho

INF01203 - 2020/2

Turma A - Prof. Viviane Moreira

Alunos: Victor Furusho Vally, Vinicius Carra

Para este trabalho, foi decidida a implementação usando estruturas AVL e ARN. Tal decisão decorre do fato que a inserção deve ser mais rápida na ARN e a pesquisa deve ser mais rápida na AVL. O código fonte e arquivos que foram utilizados para testes podem ser encontrados no GitHub, plataforma utilizada no workflow do trabalho, no link https://github.com/viniciuscarra/Trabalho-Estrutura-de-Dados

Implementação

A implementação da AVL ficou por conta do Vinicius e a implementação da ARN ficou por conta do Victor.

Foi usado um TAD para tratar os tweets, onde estrutura idTweets é uma lista duplamente encadeada circular que contém os ids numéricos dos tweets. Esse tipo de estrutura foi usada pois possibilita fazer as inserções/consultas da lista apenas em seu head, assim, ao final da inserção, a lista estará ordenada de forma decrescente de acordo com os IDs dos tweets, ganhando assim performance por não precisar percorrer a lista em tempo de inserção de nodos. O fato dela ser circular permite com que em tempo de consulta, a impressão seja feita começando pelo último elemento da lista e passando sempre para o ponteiro anterior, sendo assim, será impressa a lista em ordem crescente. Em um primeiro momento a lista era simplesmente encadeada, e para poder imprimir em ordem crescente, chamadas recursivas da função eram feitas, porém para palavras com um alto número de IDs acabavam causando um erro no programa, o que levou à necessidade de mudar a estrutura de dados para armazenagem dos IDs.

O TAD dos tweets contém as funções stringToLower, que converte as caracteres de uma string para caixa baixa, funcMax, que retorna o maior de dois números, initIDL, que inicializa uma lista de ids, imprimeIDL, que imprime os elementos de uma lista de ids e insereIDL, para inserir ids na lista. Ambas implementações de árvores utilizam este TAD para tratar os tweets.

Para iniciar o programa, é necessário passar como argumentos o arquivo com a base de tweets, um arquivo de entrada contendo as palavras que deseja-se buscar, um arquivo de saída, onde os resultados serão gravados e por fim, mas não menos importante, um código, referente à qual tipo de indexação será usada: 1 para AVL ou 2 para Rubro Negras.

Então é realizada a tokenização das palavras contidas no arquivo base, elas são passadas para a função de inserção da árvore selecionada. Após toda a inserção, é manipulado o arquivo de entrada, de tal forma que cada linha dele servirá como busca na árvore. Na busca, será impresso no arquivo de saída a palavra que deseja-se encontrar, e caso haja ocorrências dela, também será impresso os IDs dos tweets em que ocorrem. Após percorrer toda lista de busca, será impresso dados sobre a árvore, como o número de nodos e a altura da árvore e o número de comparações e rotações realizadas durante o processo de inserção. Também é apresentado o número de comparações feitas no processo de busca.

Implementação AVL

A AVL foi implementada apenas com uma estrutura, contendo as seguintes variáveis:

- palavra: Ponteiro para char contendo a palavra contida naquele nodo.
- idTweets: Ponteiro para a lista encadeada contendo os IDs dos tweets com a dada palavra.
- altura: Inteiro contendo a atual altura do nodo.
- esq: Ponteiro para o nodo à esquerda.
- dir: Ponteiro para o nodo à direita

A inserção de uma palavra se dá através da função insereNodoAVL(). Na função, é feito a comparação lexicográfica da palavra que deseja-se inserir com a palavra presente no atual nodo da árvore, utilizando a função strcmp(). Com essa comparação é possível determinar, caso as palavras sejam diferentes, para qual direção (esquerda ou direita) deve-se seguir para achar o lugar certo da inserção. Caso as palavras forem iguais, é checado se o atual ID já está na lista, caso não esteja, é inserido.

Depois que o novo nodo é inserido, é feito o rebalanceamento da árvore. Para isso, é calculado o fator de balanceamento do nodo atual e dos nodos filhos (visto que é mantido controle da altura de cada nodo, basta fazer a altura do nodo esquerdo diminuída pela altura do nodo direito). Com essas informações é possível determinar, caso seja necessário, qual tipo de rotação deve-se fazer.

Durante cada rotação, além de mudar a posição dos nodos na árvore, a altura de cada um deles é atualizada, para manter essa parte da estrutura sempre consistente, pois todo balanceamento depende desta informação

Implementação ARN

A ARN foi implementada usando duas estruturas, uma para a árvore e uma para os nós. A estrutura da árvore é chamada rbt para red-black tree e contém as seguintes variáveis:

- root: a raiz da árvore;

- nodes: o número de nós contido na árvore;
- rotations: rotações realizadas durante a inserção;
- comps: o número de strcmp realizados durante a inserção.

A estrutura do nó é chamada de rbNode para red-black node, e contém as seguintes variáveis:

- word : um ponteiro para char que contém a palavra que o nó representa;
- ids: um ponteiro para a estrutura de lista encadeada usada para armazenar os Ids dos tweets;
- color: um char que contém a cor do nó atual;
- a: ancestral do nó;
- r: filho direito do nó;
- l: filho esquerdo do nó.

A inserção é feita a partir da função rbInserir, que toma a árvore como parâmetro e chama rbInsert, que é a função de inserção recursiva. rbInsert é apenas uma função de inserção em ABP comum, mas com o adicional do caso em que a palavra do nó e a palavra sendo inserida são iguais, ou seja, strcmp retorna 0, o que chama o método de inserção de id na lista de ids dos tweets. Também tem alguns argumentos a mais, como o ancestral do nó atual, e o novo nó inserido para passar para à função de arrumar a árvore.

Após cada inserção, a árvore é reajustada para manter as regras da rubro-negra. A função arrumarArvore é chamada a partir do nó recém inserido, cada uma das regras é avaliada e colorações e rotações são aplicadas devidamente.

Durante cada rotação é necessário mudar tanto os filhos, quanto os ancestrais e recolorir os nós, adicionando complexidade ao algoritmo. Por esse motivo, a re-coloração é feita em uma função auxiliar, depois, chama as funções de rotação puras nos nós adequados.

Performance

Quanto à performance: em um processador Ryzen 5 3600, com a base completa, a fase de inserção leva, em média, 6,65 segundos em AVL e 5,5 segundos em ARN. A fase de consulta leva, em média, 0,19 segundos nas duas, com a pesquisa de 36 palavras. Tal lista de buscas não mostra uma diferença substancial nesta fase do programa por ser muito pequena.

Foi medida a performance usando a biblioteca time.h, marcando o início da inserção usando a variável start e a função clock(). Depois da inserção, é subtraído start do clock() atual e dividido por CLOCKS_PER_SEC, o que converte clocks do processador em segundos, então é atribuído à variável timeIn. O mesmo é feito para a

pesquisa, mas é atribuído à variável timeOut. As duas variáveis então são imprimidas na saída.

Nas circunstâncias da base completa, a diferença é pequena, pois um milhão de tweets é uma quantidade relativamente pequena e fácil de processar para processadores modernos. Também deve-se levar em conta que as duas árvores podem ficar muito parecidas dependendo da ordem de inserção dos dados, fazendo com que os tempos de pesquisa, que deveriam ser menores em AVL, fiquem virtualmente iguais. Porém, com um número maior de dados para processar, as vantagens e desvantagens seriam amplificadas e as diferenças de tempo de inserção e pesquisa entre a AVL e ARN seriam maiores.