Trabalho Prático III: Recebimento e Download das Consciências

Juliana Assis Alves
Matrícula: 2018115787

Thursday 9th September, 2021

1 Introdução

O seguinte documento possui como proposito apresentar uma descrição acerca da implementação do sistema elaborado para o trabalho prático III. O problema em questão consiste na edificação de um sistema capaz de suportar o armazenamento de dados e operações dos mesmos.

Para efetuar tais propósitos, utilizou-se, majoritariamente, duas estruturas de dados distintas que trabalham de forma conjunta: uma árvore binária de busca e listas encadeadas, cada nó dá árvore binária possui, portanto, uma lista encadeada relacionada. As listas possuem em cada uma de suas células um valor em binário da consciência de determinada pessoa. O programa é divido, portanto, em 3 módulos.

1. Um módulo dedicado à implementação da lista das consciências.
2. Um módulo com a implementação da árvore binária de busca.
3. Um módulo de leitura do arquivo de entrada.

2 Método

2.1 BST

Nesta seção descreverá-se, um pouco sobre a estrutura de dados implementada. Uma BST- Binary Search Tree (árvore binária de busca) é utilizada para o armazenamento de dados e para realizar uma seguinte recuperação desses dados. Em uma BST cada nó contém um campo de comparação, podendo haver outras informações, além dos ponteiros para os seus dois filhos left (esquerda) e right (direita).

O campo de comparação especifica em geral uma chave que identifica de forma única um determinado nó dessa árvore. Na BST implementada o campo de comparação é o nome de cada uma das pessoas informadas no arquivo txt. Além do nome, cada nó da árvore possui um campo do valor associado ao somatório das consciências, uma lista encadeada de tais consciências e os ponteiros para esquerda e direita. A estrutura pode ser observada na figura abaixo:

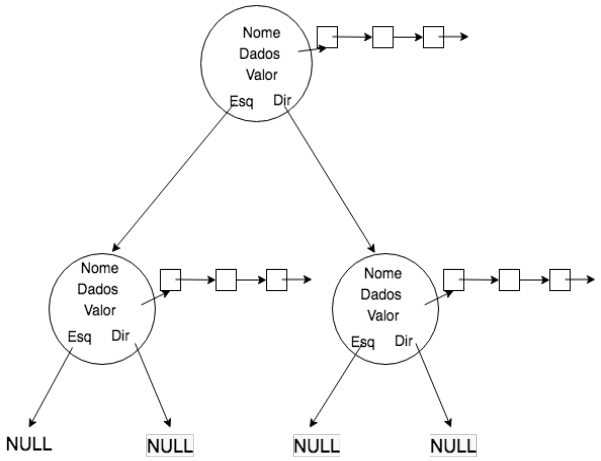

Figure 1: Esquema da árvore implementada

As funções empregadas para essa estrutura são:

- **void insert(Pointer *p, datatypet num):** A função de inserção recebe um tipo de dados datatype contendo o nome e os arquivos binários. Percorre a árvore binaria para nós internos indo tanto para a direita quanto para a esquerda. Caso o nó seja folha insere as informações contidas no datatype nessa folha. Cada nova inserção uma lista é criada. Caso o elemento a ser inserido já esteja presente na árvore apenas acrescenta-se o char do binário em sua lista relacionada.

- **void BST print inorder(Pointer *node)** Percorre a árvore em ordem exibindo suas informações. Inicia-se pelo filho à esquerda o nó interno e o filho à direita. É exibido na tela apenas o campo nome contido no nó.

- **void search(char *B, Pointer *P);** Realiza a busca de um elemento na árvore. Como parâmetro de busca usa-se uma string de nome. Caso a string seja compatível invoca-se a função de remoção.

- **void Father(Pointer q, Pointer *r);** Função auxiliar da remoção Utilizada quando o nó a ser removido possui dois filhos. Percorre-se a direita do filho à esquerda do nó que será removido. Realiza a conexão dos ponteiros e libera a memória alocada do ponteiro do nó que será removido

- **void Remove(char *x, Pointer *P);** Função usada para remover um elemento da árvore. Percorre-se a árvore até o nó que se deseja remover. Caso não encontre retorna com mensagem de erro. Verifica-se, então, o estado dos filhos. Caso não haja filhos à direita substitui o mesmo pelo filho à esquerda. Não havendo filhos à esquerda substitui pelo mais á direita de seus descendentes. Nos demais casos substitui pelo filho à esquerda.

2.2 Lista Encadeada

Cada um dos nós da árvore representam uma pessoa. Cada pessoa pode ter partes de sua consciência enviada aos poucos, portanto implementa-se uma lista encadeada de consciências em cada nó da árvore. As funções relacionadas a essa estrutura de dados são as seguintes:

- **void Allocate List(List **L);** Cria uma lista vazia.
- **int Insert List(List **L, char *Bin);** Insere um novo item na lista. São inseridos tanto os valores em binário quanto em decimal depois de convertidos.
- **int Bin to int(char *Bin);** Converte um valor em binário no tipo string, para um valor em decimal do tipo int. Realiza módulos e divisões consecutivas da string até que o decimal seja encontrado.
- **void show List(List **DL);** Exibe o conteúdo da lista em decimal (Já convertido). Função usada para testes do comportamento do algoritmo.

2.3 Leitor

O módulo de leitura fraciona as linhas do arquivo, identificando quais os nomes e dados respectivos que serão armazenados na árvore. Recebe como parâmetro o nome de um arquivo e o modifica para que o arquivo em seu diretório seja aberto de forma adequada. Lê a primeira linha do arquivo e armazena o numero de inserções. Realiza as N inserções particionando a string em duas, uma para nome e outra para dados. Percorre a árvore em ordem e, por fim, realiza o envio das consciências fazendo do buscas e remoções na árvore e exibe o último elemento restante a ser enviado.

3 Estudo da complexidade de tempo e espaço

Para a analise da complexidade de tempo do projeto é importante dividir o custo por cada um dos módulos construídos.

3.1 Complexidade de tempo

3.1.1 Leitor

O leitor, executa a leitura dos nomes e dos dados, realizando tratamento de strings nas N linhas do arquivo de entrada. Considerando que as tarefas de convocação de funções e os tratamentos de strings, feitas a cada linha lida, são de custo constante, a complexidade para esse módulo pode ser descrita como o $O(N)$;

3.1.2 BST

A ideia principal da BST é a divisão em ramificações, portanto quanto mais dados forem inseridos maior será a altura H dessa árvore. A análise de complexidade de tempo nessa estrutura é significativa na maioria das funções que envolvem a estrutura. Dessa forma, as seguintes funções possuem o mesmo aspecto de busca e translado na árvore. Apresentando, portanto, complexidades muito próximas:

1. Na etapa de **inserção** é necessário encontrar o melhor local para que o novo nó seja inserido; Na inserção sempre ocorrerá de alocar o novo nó em uma posição folha da árvore, sendo necessário atravessar toda a árvore.

    - No melhor caso, a complexidade de tempo seria de $O(1)$, em que o nó a ser inserido é o nó raiz da árvore.
    - No pior caso, a complexidade de tempo gasta para a pesquisa é de $O(H)$ em que H é a altura da árvore.

2. Na **pesquisa** é necessário percorrer a árvore a fim de verificar a existência do nó buscado Para pesquisa há os seguintes casos:

    - No melhor caso, o nó buscado é o nó raiz e a complexidade de tempo é de $O(1)$.
    - Pior caso, em que o nó compatível esteja nas últimas posições, a complexidade de tempo gasta para a pesquisa é de $O(H)$ em que H é a altura da árvore.
    - No caso médio, a complexidade de tempo gasta para as comparações é de $O(logN)$ sendo N o número de nós, de acordo com Ziviani[1].

3. Para **imprimir** a árvore é necessário percorrer toda a árvore. No caso da impressão, seu comportamento é sempre de pior caso, pois a árvore será totalmente percorrida. Portanto sua complexidade irá variar conforme o seu balanceamento.

    - Caso a árvore não esteja balanceada: a complexidade de tempo gasta para sua impressão de $O(H)$ em que H é a altura da árvore.
    - Caso a árvore esteja balanceada: a complexidade de tempo gasta para impressão será de $O(logN)$ em que N é o número de nodos presentes na árvore.

4. Na **remoção** é necessário percorrer até o nó que deverá ser removido, a primeira etapa da remoção é uma busca na árvore binária. Portanto, possui as mesmas complexidades da pesquisa.

    - No melhor caso, o nó buscado é o nó raiz e a complexidade de tempo é de $O(1)$.

- Pior caso, em que o nó compatível esteja nas últimas posições, a complexidade de tempo gasta para a pesquisa é de $O(H)$ em que H é a altura da árvore.
- No caso médio, a complexidade de tempo gasta para as comparações é de $O(logN)$ sendo N o número de nós, de acordo com Ziviani[1].

Entretanto, na remoção caso o nó a ser removido possua dois filhos é necessário escolher o maior filho a esquerda para ser colocado na posição do nó que será removido, tendo portanto que realizar um translado da subarvore á direita do filho à esquerda desse nó. Para tal, a complexidade de tempo associada é de $O(ND)$ em que ND são os N filhos à direita do filho à esquerda do nó que será removido.

3.1.3 Lista Encadeada

Cada um dos nós da árvore possui uma lista encadeada para armazenar as consciências. Na lista encadeada as inserções são realizadas em sua calda. Portanto, a complexidade de tempo relacionada a essa estrutura seria a de inserção equivalente à $O(N)$ em que N é o número de células da lista. Uma vez que para inserir é necessário percorrer a lista até sua calda passando pelos seus N nós já devidamente inseridos.

Além da inserção, outra operação relacionada com as listas que possui custo de tempo significativo é a operação de conversão de binário para decimal. Essa, por sua vez, será impactada pelo tamanho S d string que representa o binário, uma vez que os módulos e as divisões são feitas S vezes em loop. Portanto a complexidade de tempo associada é de $O(S)$.

3.2 Complexidade de espaço

Todo o sistema é operado sobre dois tipos de estruturas básicas. Uma árvore binária de busca em que cada nó representa uma pessoa e uma lista encadeada associada à cada nó para armazenar os respectivos arquivos binários. A lista encadeada de cada nó aumenta de acordo com a quantidade de inserções solicitadas para esse mesmo nó. Dado N, o número de inserções de uma mesma pessoa na árvore a complexidade de espaço da lista encadeada é de $O(N)$.

No caso da árvore, seu tamanho aumenta conforme a quantidade de pessoas diferentes que estão definidas no arquivo de entrada. Portanto, seja M, o montante de pessoas distintas que devem ser armazenadas na árvore, a complexidade de espaço para essa estrutura será de $O(M)$. Supondo que cada um dos M nós aloca da árvore possuam no máximo M células criadas em suas respectivas listas. A complexidade total será:

$$O(M * N)$$

4 Conclusões e considerações finais

O trabalho foi concluído com exito. Implementou-se uma árvore binária de busca em que cada nó possui uma lista encadeada. A árvore é preenchida via operação de leitura de um arquivo. Na edificação do trabalho foram empregados os seguintes conceitos vistos em aulas:

1. Árvore binária de busca
2. Listas encadeadas
3. Análise de complexidade de algoritmos
4. Padrões de código

. Usou-se diversos módulos, cada qual com seus respectivos métodos para a implementação completa do sistema. Em diversos trechos, foi mantida os *defines* de debug para que a lógica do código possa ser analisada, caso seja a vontade do leitor.

5 Referências

1. Ziviani, Nivio. Projeto de algoritmos: com implementações em Pascal e C. Vol. 2. Luton: Thomson, 2004.
2. Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2009). Introduction to algorithms. MIT press.

6 Compilação e Execução

Para compilar e testar o sistema é necessário que os seguinte passos sejam seguidos:

1. (../src/) **make**
2. **make nome.txt**

nome.txt refere-se ao nome do texto que será usado para ser executado no sistema.