UNIVERSIDADE FEDERAL DE MINAS GERAIS

Trabalho Prático 2

Aluno: João Vítor Santana Depollo

Matrícula: 2021039751

Email: joaovitorsantanadepollo@gmail.com

1. **Introdução**

   Neste trabalho prático, foi passado como tarefa aos alunos a criação de um sistema de e-mails que armazenasse os e-mails de forma que a procura e o envio de um e-mail entre usuários seja otimizado. Como solução, será implementado uma estrutura de dados conhecida como “HashMap” ele armazena outro tipo abstrato de dados de forma que o sistema não fica contido em uma enorme lista de informações e que possamos “cortar o caminho” quando quisermos recuperar determinada informação. Exemplo: Temos 10 listas e cada uma armazena um intervalo diferente de 100 números. Para pesquisarmos se um determinado elemento existe em nosso HashMap, não precisaremos de checar todos os elementos até encontrá-lo, basta irmos na lista em que o elemento se encontra dentro do intervalo do TAD que implementamos.

2. **Método**

   Dentro do HashMap, Árvores Binárias de Pesquisa foram escolhidas como TAD para a pesquisa. Em que durante a execução do programa, passamos um número “U” que representa o número de um usuário. Contextualizando, suponhamos que dentro do HashMap teremos 10 árvores e o U do e-mail é 14, será utilizado o resto da divisão de U por 10 para determinar em qual árvore ele será inserido, no caso, 4.

   Para facilitar na busca dos e-mails será implementado uma BST(Binary Search Tree) dentro de nosso HashMap. Uma Árvore Binária de Pesquisa tem uma forma característica de armazenar a informação que precisamos, ao invés de irmos enfileirando um elemento atrás de outro, verificamos se um identificador(definido pelo desenvolvedor) é menor do que o primeiro nó da árvore(raiz), se sim, inserimos a esquerda dele esse, caso contrário, o inserimos a direita. Além disso, caso já exista um nó para direção que fomos, fazemos essa

mesma verificação até chegar nas folhas dessa árvore(onde não há elementos filhos). Nesse projeto, os nós foram implementados de forma que armazenaremos o e-mail com o identificador do destinatário, e-mail e o corpo do e-mail. Dessarte, um e-mail que possui o mesmo identificador de outra lista, não será considerado como igual, pois fazemos também uma comparação entre o índice do “HashMap” que foi passado na criação do e-mail. Na operação de remoção de um e-mail é feita a rotação dos nós filhos do nó removido, para manter a propriedade da BST.

Funcionamento do código principal:

Nesse sistema, existem 4 classes, sendo elas: email, node, tree e hash. A classe e-mail tem o identificador do e-mail, do usuário e armazena o texto da mensagem. A descrição das outras classes já foi feita anteriormente. A respeito dos métodos e das operações realizadas dentro do programa, temos as operações básicas de busca, inserção e remoção dos e-mails. Todas essas operações olham pelo identificador do nó para se direcionarem para qual direção a função vai seguir(direita ou esquerda). Dessa forma, os casos médios da busca pelos e-mails tem uma performasse melhor.

No código principal, temos a leitura dos valores de entrada e a inicialização do HashMap, é lido a quantidade de BSTs da aplicação, em que esse número é utilizado para alocar dinamicamente o elemento da classe. Se for chamada a operação de entrega, armazenamos o valor da quantidade de palavras do corpo do e-mail e é feito um loop para concatenar todas as entradas na variável email.

3. **Instruções de compilação e execução**

Na pasta raiz do projeto, basta utilizar o comando make, no terminal, que ele irá compilar o programa e também irá utilizar o arquivo entrada.txt como entrada para o .exe e jogará o output para o arquivo saída.txt. **IMPORTANTE**, o arquivo de entrada tem que ser fornecido pelo usuário e tem que estar na pasta raiz do projeto. Além disso, dentro do makefile tem a opção de compare, que olha dentro de 2 pastas input e output. Dessa forma, ele executa o programa com os arquivos de entrada dentro dessa pasta e compara com as saídas fornecidas dentro da pasta output. Caso seja interessante o uso dessa opção, é importante que o usuário crio essas pastas e coloquem os arquivos com o nome “entrada_numero.txt” e “ saida_numero.txt” em que número será um valor inteiro. Por padrão o compare do makefile compara até o número 8, caso o número de testes seja maior ou menor, é necessário que o usuário faça as alterações necessárias ao makefile.

4. **Análise de Complexidade**

Hash_LE::Insere:

A função usa da recursividade para percorrer o TAD, conferindo O(1) para a complexidade de tempo, porém para a complexidade de espaço ela possui os seguintes custos das chamadas recursivas:

No melhor caso, a árvore não contem nenhum elemento, ou seja, O(1).

No caso médio, a árvore tende ao balanceamento, o que confere $O(\log_n)$ para a função.

No pior caso, a árvore está degenerada, com um galho que tem a altura igual a n elementos que a árvore possui, ou seja, O(n).

HASH_LE:Verify:

Essa função, chama a função "**pesquisa"** de casa árvore, e elas não são recursivas. Então tem uma complexidade de tempo representada da seguinte forma:

Melhor caso: O(1)

Caso médio: $O(\log_n)$

Pior caso: O(n)

HASH_LE:Remove:

As chamadas são recursivas, logo tem esta complexidade de espaço:

Melhor caso: O(1) – o Nó é uma folha

Caso médio: $O(\log_n)$- o Nó está na metade da árvore

Pior caso: O(n) – O nó é a raiz

HASH_LE::Hash:

O(1) em todos os casos.

5. **Conclusão**

Para percorrer a árvore de forma mais clara, a recursividade acaba sendo um recurso bastante interessante de ser trabalhado nesta aplicação, pois a parte de

remoção se torna mais complexa de ser implementada, devido a complexidade de rotacionar a árvore sem usar uma forma recursiva para tal. Além disso, a implementação com o uso de Hash se mostrou muito prática após a criação da Árvore Binária de Pesquisa, conferindo elegância e uma performasse maior no algoritmo para pesquisa e inserção de novos e-mails em nosso sistema. Um desafio maior foi lidar com possíveis casos de colisão, em que e-mails podem ter o mesmo identificador e isso dificultou a distinção dos mesmos, para tal foi necessário criar o atributo “part” que representa o parâmetro do usuário que é utilizado para escolher qual BST que seŕa utilizada pelo códio. Como por exemplo, o caso de quando temos somente 2 árvores no HashMap, ou seja, nossa aplicação calcula o número mod 1 para inserir um e-mail no sistema. Logo, um e-mail que recebeu o código 0,12,3 ou qualquer outro número sempre estará na primeira árvore, mesmo que os e-mais sejam diferentes. Logo, também foi necessária comparação que esse parâmetro para certificar de que em uma consulta, eu estou tratando do e-mail correto.

6. **Bibliografia**

https://www.geeksforgeeks.org/binary-search-tree-set-2-delete/

https://www.cs.usfca.edu/~galles/visualization/BST.html

https://visualgo.net/en/bst?slide=1