Trabalho Prático 2 da disciplina de Estrutura de Dados

Alex Eduardo Alves dos Santos
Matrícula: 2021032145

Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil
alexeduardoaeas@ufmg.br

1. Introdução

O problema proposto foi criar um simulador de um servidor de emails. O sistema simulado tem suporte à entrega, consulta e remoção de mensagens para usuários.

Para resolver o problema citado foi utilizado uma tabela Hash combinado com uma Árvore Binária, permitindo que a entrega, consulta e remoção de mensagens fossem feitas de maneira eficiente.

2. Implementação

O programa foi desenvolvido na linguagem C++, compilada pelo compilador G++ da GNU Compiler Collection.

2.1 Estrutura de Dados

A implementação do programa teve como base 3 estruturas de dados, uma tabela hash, que armazena em cada posição da tabela uma árvore binária de pesquisa onde em cada nó é armazenado o item, no caso o email. Essa combinação de estruturas de dados permite que a pesquisa seja bastante eficiente, assim como a remoção e a inserção, em comparação com outras estruturas, essas possuem características específicas que as tornam mais adequadas para o problema proposto.

A tabela hash foi implementada em uma classe, e os índices de cada posição da tabela foi dado como o resto da divisão entre a chave de usuário e o tamanho da tabela, espera se que o tamanho da tabela seja sempre <=100, pois é iniciada 100 posições para serem usadas na tabela, alterações simples na Classe Hash_LE podem aumentar ou diminuir esse valor, caso seja do conhecimento prévio do programador o tamanho que a tabela deve ter. Além disso alguns métodos de inserção e remoção também estão implementados nessa Classe, esses métodos retornam valores que deixam o programa mais robustos, informando se o procedimento foi realizado com sucesso ou se houve falhas.

A árvore binária de pesquisa foi implementada para tratar de colisões na tabela hash, ou seja, quando duas ou mais chaves de usuários possuem o mesmo resto da divisão pelo tamanho da tabela, nesse caso, os emails devem ser armazenados na mesma posição da tabela e para permitir que isso não fosse um problema uma árvore binária de pesquisa armazena todos os valores em cada índice da tabela hash, ou seja, cada posição da tabela hash possui uma árvore que trata as entradas. A escolha da árvore binária de pesquisa é muito adequada para esse problema, pois as operações de pesquisa, remoção e inserção, são feitas de maneira bastante elegante por essa estrutura de dados. Essa árvore funciona

armazenando os E Mails através da chave que cada um possui em seus nós, a forma de caminhar nesse tipo de estrutura de dados é muito melhor do que em uma lista encadeada, por exemplo, pois não precisa percorrer todos os elementos para encontrar o que quer, exceto em raras exceções de inserções ordenadas. Por fim, métodos recursivos de inserção, pesquisa e remoção que compõem essa árvore, é capaz de resolver os problemas propostos de maneira adequada.

Por fim, a estrutura de dados mais simples, feita em uma Classe TipoItem, apenas armazena e retorna os dados do Email, como o identificador do usuário que possui o email, o corpo do email, e a sua própria chave. Esses dados são de extrema importância para o funcionamento correto das outras Estruturas de dados, pois para armazenar um email na tabela hash, é necessário saber qual usuário possui esse email, e para que dois usuarios nao possuam o mesmo e mail, devido às colisões, é verificado se o usuário que está consultando o email, é o que possui esse email através do identificador de usuário que essa estrutura possui.

2.2 Pesquisa

Uma das partes mais fundamentais do programa é realizar a pesquisa de maneira consistente e eficiente, portanto, para realizar a pesquisa são realizados alguns passos importantes.

A pesquisa é feita através das chaves de usuário e do email, a chave de usuário é utilizada na função hash da tabela para identificar em qual posição da tabela os e mails daquele usuário estão armazenados, em seguida a chave do email é utilizada na pesquisa da árvore binária, e ao ser encontrada o item que possui a chave desse email é retornado, no entanto ainda são feitas as verificações se a chave do usuário é a mesma chave de usuário que possui aquele email. E caso a chave não exista um item vazio é retornado indicando que não foi encontrado um email com aquela chave.

2.2 Inserção

A inserção de um novo email, é feita de maneira simples e rápida, primeiro é chamada a função hash com o valor da chave do usuário para encontrar o valor de índice da tabela, e em seguida, é inserido um novo nó na árvore binária de pesquisa a partir da chave do email, em particular esse nó sempre é uma nova folha, ou seja, é necessário percorrer a árvore a partir raiz até o nó que será o pai da nova inserção.

2.3 Remoção

Para apagar um email, é necessário antes encontrá-lo, para isso o método pesquisa é chamado antes de tudo, se o email existe então ele será apagado, para isso, é necessário encontrá lo pelos métodos descritos na operação de pesquisa, com a diferença que ao fim, o método irá apagar o nó correspondente da árvore, e retornará se a operação foi bem sucedida.

3. Análise de Complexidade

3.1 Tempo

Basicamente o programa realiza apenas 3 operações, insere, pesquisa e remove. Não se sabe de antemão quantas serão feitas ao longo da execução, portanto será considerado a ordem de complexidade de cada operação individualmente, e assume se que elas poderão se

repetir um número constante de vezes. Além disso, as outras operações com ordem O(1) que estão na main também serão desconsideradas na análise.

3.1.1 Insere

Para efetuar a entrada de um novo email, o método Insere é chamado na main o primeiro passo do método, é realizar a busca pelo elemento que vai ser inserido na tabela pois se ele já estiver presente não será alocado novamente. Mas se ele está sendo inserido pela primeira vez, então será feito o método que calcula o hash, e posteriormente será chamado o método Insere da árvore que está na posição da tabela Hash indicada.

Esse método apenas chama o InsereRecursivo, que basicamente caminha entre os nós da árvore binária até encontrar uma posição vazia no qual o nova chave pode ser inserida, esse operação custa O(n), caso a arvore tenha se degenerado e se tornado uma lista encadeada, mas no caso médio a complexidade é O(logn) e esse será o caso que em geral deve acontecer.

3.1.2 Pesquisa

Como dito anteriormente o Pesquisa é fundamental para que ele e os outros métodos funcione corretamente, o pesquisa é chamado na main quando é necessário buscar as informações de um email, além de ser chamado nos outros métodos para garantir que eles funcionem da forma correta, ao ser chamado ele calcula a função hash que custa O(1) e chama o Pesquisa da árvore binária correspondente, no qual chama o PesquisaRecursivo, esse método assim como o insere caminha na arvore buscando um elemento que tenha a chave buscada, portanto o pior caso acontece também na árvore degenerada e o caso médio são os mesmos do método de inserção.

3.1.3 Remove

Quando é necessário apagar um Email antes de tudo, o método Remove é chamado, esse método primeiramente realiza uma pesquisa para descobrir se o email existe e pode ser removido, caso seja ele exista então ele será removido chamando o método remove na árvore binária encontrada através da função hash. Em seguida é chamado o remove recursivo que assim como os outros a complexidade assintótica está atrelada ao caminhamento pela árvore, sendo assim o pior caso seria em uma árvore degenerada que o item a ser removido se encontra no nível mais baixo, sendo assim O(n), no entanto esperasse que o caso médio seja mais recorrente, sendo assim O(logn).

No final, tem-se que

Pior Caso:
- Insere: O(n)+O(n)
- Pesquisa: O(n)
- Remove: O(n)+O(n)

Caso Médio:
- Insere: O(logn)+O(logn)
- Pesquisa: O(logn)
- Remove: O(logn)+O(logn)

3.2 Espaço

Vamos considerar que cada email ocupe um espaço de memória ou que cada árvore ocupe a quantidade de nós inseridos que ela tem, e a tabela hash ocupa 100 espaços de memória, tal que cada um inicializa uma árvore com 0 nós. Logo o espaço ocupado pelo programa é

O(100)(Posições da Tabela Hash)+O(n) (E Mails armazenados)

Essa definição é para n <=100, tal que apenas um email foi inserido em cada posição da tabela e portanto é a raiz das suas próprias árvores.

Para n>100 os mesmos 100 espaços da tabela Hash serão criados, no entanto as árvores de cada lugar da tabela, terá um número que não é possível calcular, de maneira simples e objetiva, supondo que será distribuído de forma uniforme então cada árvore terá n/100 itens inseridos portanto a complexidade espacial seria

O(100)(Posições da Tabela Hash)+O(n) (E Mails armazenados)

Como as inserções quase nunca seguem um padrão rigoroso, pode acontecer de alguma posição da tabela hash ficar vazia, no entanto isso seria pouco significativo para a análise. No fim a complexidade espacial está atrelada a quantidade de Emails armazenados sendo assim **O(n).**

4. Estrategias de Robustez

Em casos de operações inválidas foram tomadas medidas para evitar que elas sejam efetuadas, a fim de manter o programa robusto, e de maneira elegante avisos são dados ao usuário para ele saber o que aconteceu.

Essas estratégias se trata basicamente da verificação da possibilidade de realizar a operação antes de realizá-la, como a verificação da existência do item antes de tentar removê-lo, podendo retornar o aviso que o item não foi removido pois não existia. A verificação de um item antes de inseri-lo, caso o item já estivesse inserido, tentar inseri-lo novamente poderia causar problemas e confusão, portanto a operação não é realizada e um aviso que o item já foi inserido é emitido. Além da operação de consulta, caso o email não exista a consulta falha e retorna que o email não foi encontrado.

5. Conclusão

Após a implementação do programa, pode se notar que havia partes complementares para o desenvolvimento do trabalho. A primeira era a implementação correta da tabela hash, de modo que permitisse que os emails fossem inseridos através do ID de Usuário, e pudessem ser acessados posteriormente.

A segunda era a de realizar o tratamento de colisões nessa tabela Hash, utilizar a árvore binária de pesquisa de maneira consistente, podendo inserir os emails a partir do seu ID, permitir que o acesso e remoção deles fossem feitos da maneira correta e entender os motivos dela se adequar tão bem para o problema foi de grande importância.

Após isso, ao combinar essas duas estruturas de dados tão poderosas, resolver o problema de maneira elegante ficou muito mais simples e elegante, agora que as operações são feitas rapidamente.

Ao fim desse trabalho, ficou ainda mais evidente a importância dos testes de unidade, da refatoração, da modularização e planejamento do código. Ambas práticas possibilitaram

um desenvolvimento mais produtivo/consciente de que, com a implementação de novas funcionalidades ou com a refatoração de trechos do código, as funções permaneciam corretas e funcionais.

Referências

Ziviani, N. (2006). Projetos de Algoritmos com Implementações em Java e C + +: Capítulo 3: Estruturas de Dados Básicas . Editora Cengage

Chaimowicz, L. and Prates, R. (2020). Slides virtuais da disciplina de estruturas de dados. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.

Instruções para compilação e execução

- Acesse o diretorio [TP/];
- Cole o arquivo de entrada, ex: [SuaEntrada.i];
- Utilizando um terminal, execute o arquivo [MakeFile] utilizando o seguinte comando: <make>;
- Com esse comando, o programa irá compilar gerando os arquivos .o dentro do diretório [TP/obj/] e o [run.out] dentro do diretório [TP/bin/];
- Com o programa devidamente compilado, utilizando um terminal execute o arquivo [run.out] utilizando o seguinte comando:
  <./bin/run.out -o [NomeSaida.o] -i [SuaEntrada.i]
- Após a execução desse comando será gerado o arquivo [NomeSaida.o] no diretório raiz do programa.