Trabalho Prático 3

Servidor de emails otimizado

Leandro Freitas de Souza – 2021037902

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

leandrofreitas@dcc.ufmg.br

1. Introdução

O principal objetivo desse trabalho foi a implementação de um servidor de email otimizado. Para o sucesso dessa implementação, foram usadas estruturas de dados aprendidas na disciplina.

2. Método

O programa foi desenvolvido na linguagem C++, e compilado pelo compilador G++, da GNU Compiller Collection. Além disso, a máquina utilizada no desenvolvimento do programa possui 8GB de memória RAM, e um processador Intel Core I5, de 11ª Geração.

2.1. Estruturas de Dados

Na construção do programa foram usadas duas principais estruturas de dados: uma tabela de hashing e uma árvore binária de pesquisa.

- Hashing

A tabela hashing é uma estrutura de dados de pesquisa, que através de uma função de transformação de uma chave, mapeia cada chave em um novo endereço.

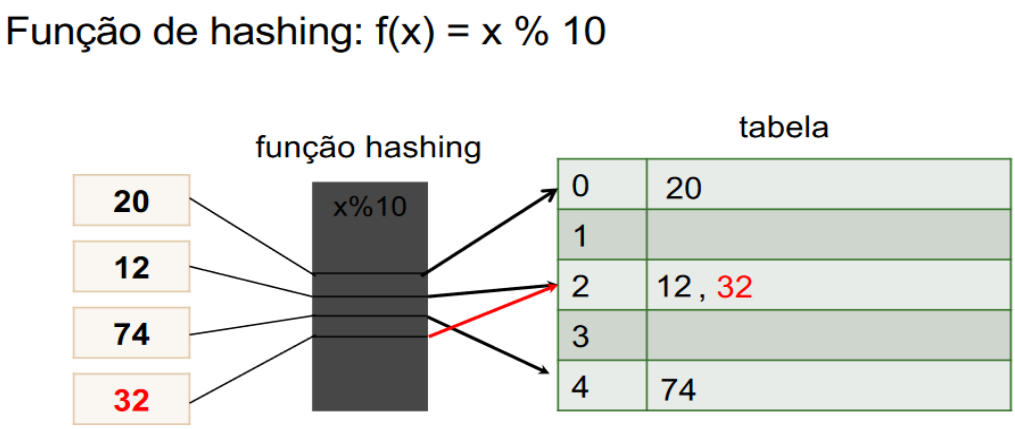

Imagem 1: Tabela hashing

A utilidade dessa estrutura se deu pelo armazenamento e fácil acesso dos usuários do email. Cada endereço da tabela hashing é direcionado a uma árvore binária de pesquisa, que guarda os emails correspondentes aos usuários referentes à entrada.

- Árvore Binária de Pesquisa

Uma árvore binária de pesquisa é utilizada devido ao acesso em tempo médio logarítmico dos emails armazenados, de forma a possibilitar a sua busca. Essa estrutura consiste de um grafo em formato de árvore binária cujos nós são ordenados de forma que, para um nó qualquer, todos os emails armazenados à direita desse nó possuem chaves maiores, enquanto todos os emails armazenados à esquerda desse nó possuem chaves menores.

Por mais que um endereço da tabela de hash possa conter mais de um usuário, não há colisão nos emails, uma vez que os números de mensagem são distintos, mesmo para usuários diferentes.

O uso dessas estruturas de dados permitiu a fácil implementação do trabalho, com o auxílio de algumas funções contidas nessas classes.

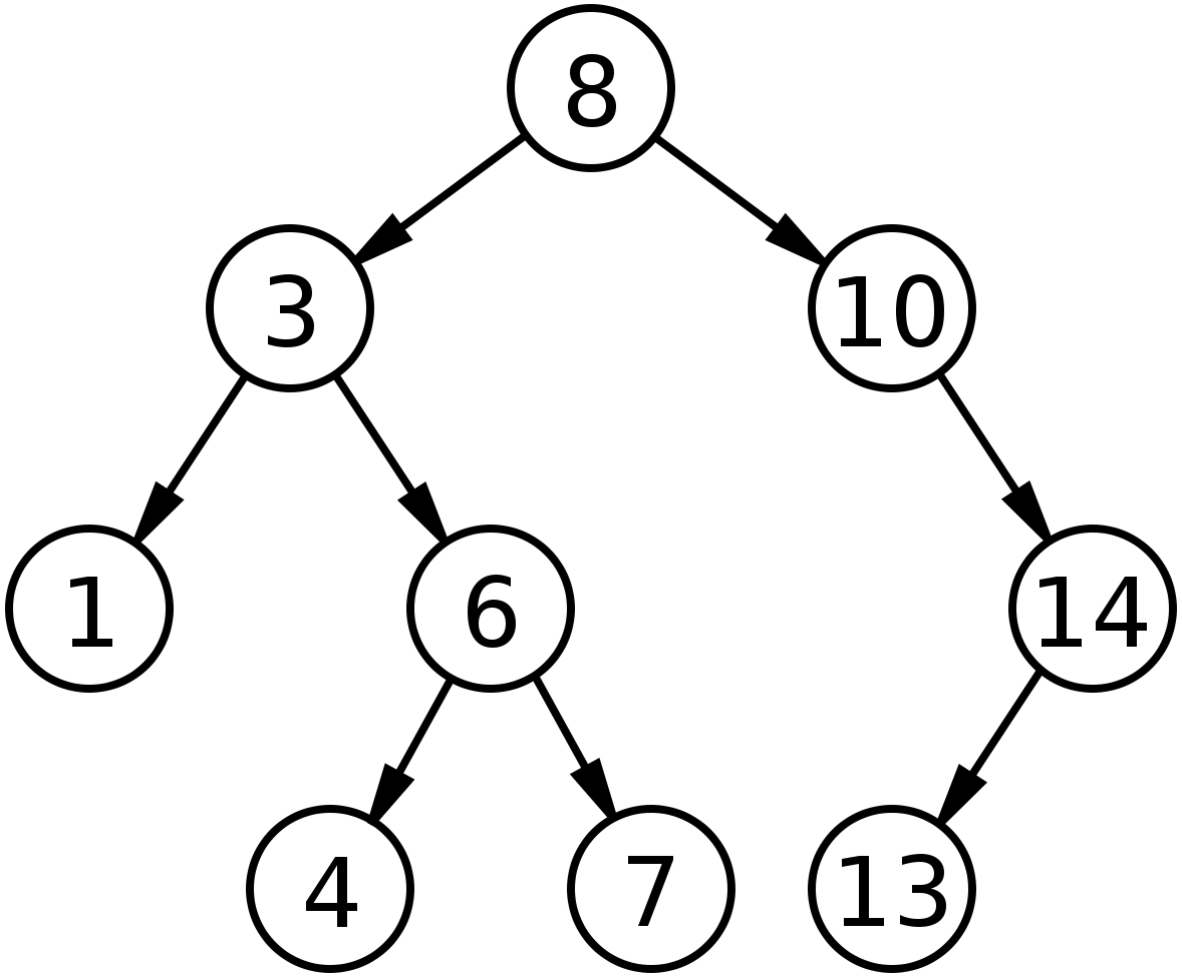

Imagem 2: exemplo de uma árvore binária de pesquisa

2.2 Funções

O programa conta com três funções principais, que correspondem a cada uma das operações que o código deve realizar: enviar um email, consultar um email e deletar um email.

A função de inserir, em primeiro lugar acessa a posição de um usuário na tabela hash e insere um email na árvore binária correspondente. Essa função se dá de forma recursiva e possui tempo médio de ordem logarítmica.

A função de consulta, após encontrar o usuário correspondente na tabela hash, procura, por meio do id da mensagem, a mensagem correspondente. Ela retorna a mensagem que foi buscada, e, caso não haja essa mensagem, ela retorna uma mensagem com a chave “-1” e

com texto vazio. Essa função também possui complexidade de tempo média logarítmica.

A função de remoção remove, de forma recursiva um email com a chave dada. Para manter a estrutura da árvore, ela substitui o nó removido pelo último nó à esquerda, caso o nó removido tenha dois filhos.

3. Análise de Complexidade

3.1. Tempo

Para analisarmos a complexidade de tempo, é necessário tomar em conta que cada operação do código, independentemente de sua função realizada, possui complexidade média O(log(n)), sendo n o número de elementos presentes na árvore binária de consulta, uma vez que a consulta na tabela hashing possui complexidade O(1).

Dessa maneira, para inserir n elementos na árvore, a complexidade do programa é, com certeza, nlog(n), uma vez que a complexidade aumenta para a inserção de novos elementos.

Para a realização de uma operação, seja de consulta ou de remoção, a sua complexidade será O(log(n)).

Dessa maneira, a complexidade assintótica média para o programa, para m consultas ou remoções e n inserções será O((n+m)log(n)) = O(max(n, m)log(n)).

3.2. Espaço

A inserção e remoção de emails na árvore não requer nenhum tipo de espaço auxiliar, então, seja n o número de emails, e H a chave hashing, a complexidade de espaço do código será O(n + H).

4. Estratégias de Robustez

Para garantir a robustez do programa, foram seguidas algumas estratégias.

Uma das estratégias utilizadas na leitura dos emails foi ler o seu conteúdo palavra a palavra, e caso o email tenha mais palavras que o informado na sua inserção, elas serão ignoradas, mas o funcionamento do código não será comprometido, uma vez que ele lerá as palavras até o final do arquivo ou encontrar outro comando. Além disso, a árvore binária, por possuir alocação dinâmica, pode atingir um tamanho muito grande.

5. Análise Experimental

Para a avaliação experimental do trabalho, além da execução do código com os casos testes disponíveis no Moodle, foi realizada uma testagem com entradas grandes, para verificar se o programa consegue armazenar uma quantidade grande de mensagens.

Além disso, foi feita uma análise de tempo e de localidade referência a partir de gráficos gerados pelo gprof, que confirmaram a complexidade de tempo e memória esperados do código, além de contribuir com a sua robustez.

6. Conclusão

A partir da realização desse trabalho, foi possível aprofundar meus conhecimentos sobre árvores binárias, e principalmente de hashing. A implementação não foi um grande desafio, embora inicialmente houve dificuldade no entendimento do funcionamento do programa. Dessa maneira, na minha opinião todos os objetivos do trabalho foram cumpridos, uma vez que ele funciona de acordo com o especificado em sua descrição.

7. Referências

https://virtual.ufmg.br/20221/course/view.php?id=11607

Apêndice: Instruções para a execução do programa

Para a execução do programa, basta, em primeiro lugar, utilizar o terminal para acessar o diretório “TP” e executar o comando “make all”. Uma vez executado, será gerado um executável no diretório “bin” com o nome “tp.exe”. Ao executar esse arquivo, devem ser fornecidos dois parâmetros: o nome do arquivo com as informações de entrada, precedido por –i e o nome do arquivo de saída, precedido por –o.