Trabalho Prático 3

Servidor de emails otimizado

Ítalo Dell'Areti

Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte – MG – Brasil

italodellareti@ufmg.br

1. Introdução.

O problema proposto é a criação de um simulador de um servidor de e-mails, que possui certas características predeterminadas como entregar e-mails, pesquisar e-mails e apagar e-mails. Ademais, a implementação deve possuir um foco na parte de gerenciamento adequado da memória do sistema,e na otimização da pesquisa por usuários e mensagens, otimizando o sistema, utilizando menos espaço e realizando consultas de forma mais eficiente e veloz. Portanto, a partir do input fornecido, contendo um arquivo de texto,que em cada linha representa uma operação e os resultados vão ser armazenados e confirmados pelo output console.

Para estruturar, vamos utilizar os tipos abstratos de dados, processos de busca já previamente estabelecidos para resolver este problema. Tendo em vista disso, este documento possui como objetivo explicitar o funcionamento desta implementação.

2. Implementação

O programa foi desenvolvido utilizando a linguagem de programação C++, compilada pelo G++ da GNU Compiler Collection. Ademais, o programa foi produzido e testado em um ambiente Linux distribuição Ubuntu 20.04 LTS utilizando Visual Studio Code.

2.1 - Modularização e Organização

Como exigido pela documentação, a principal estrutura do servidor de e-mails é uma tabela hash, que está em tabela_hash.cpp, tendo as entradas desta tabela aponta para uma árvore binária,que está em arvore_busca.cpp, e cada nó desta árvore armazena um email.

Temos como arquivos os headers arvore_busca.h, mensagem.h, tabela_hash.h, com as funções de criar as classes e estruturas, e classificando suas variáveis como públicas e privadas para o bom funcionamento do programa.

A implementação do programa foi realizada utilizando as classes **bst** (binary search tree) e **hash_table** e a estrutura **main.**

A classe **bst** é onde a estrutura de dados de árvore binária baseada em nós, onde todos os nós da subárvore esquerda possuem um valor numérico inferior ao nó raiz e todos os nós da subárvore direita possuem um valor superior ao nó raiz. É nesta estrutura que armazenamos os emails recebidos. Sendo assim, ele está responsável pela criação, remoção e busca dos emails.

A classe **hash_table** é a responsável por armazenar as caixas de mensagens dos usuários, sendo nela onde realizamos as operações de criar mensagens, remover mensagens e encontrar mensagens. Cada entrada da tabela hash apontará para a **bst**, como dito anteriormente, é onde fica armazenado os emails.

2.2 - Estrutura de Dados

Para atender as demandas em relação às operações necessárias, vamos analisar as principais funções do programa.

Em **arvore_busca.cpp** temos

bst::bst : Cria a árvore de pesquisa binária.

bst::insere : Insere a árvore de pesquisa binária.

bst::remove : Remove nó árvore de pesquisa binária.

bst::min : Busca o menor elemento.

bst::encontra : Busca na árvore .

Em **tabela_hash.cpp** temos

hash_table::hash_table : Cria a hash.

hash_table::insere : Insere no hash.

hash_table::remove : Remove elemento do hash.

hash_table::encontra : Busca elementos no hash.

hash_table::hash : Realiza a operação do hash - U % M

Na **main.cpp** temos apenas as chamadas de função e algumas estratégias de robustez para o bom funcionamento do programa.

3. Análise Complexidade

3.1 - Tempo

Após esta breve visão sobre o programa como um todo, agora vamos mostrar algumas ordens de complexidade das principais funções do programa, tendo em vista que seus impactos influenciam no tempo de execução do programa. Dessa forma, assim, a complexidade das principais funções são

Em **arvore_busca.cpp** temos

bst::bst : Custo O(1), criando a árvore.

bst::insere : Custo O(n)

bst::remove : Custo O(n)

bst::min : Custo O(n)

bst::encontra : Custo O(n)

Em **tabela_hash.cpp** temos

hash_table::hash_table : Custo O(n)

hash_table::insere : Custo O(1)

hash_table::remove : Custo O(1)

hash_table::encontra : Custo O(1)

hash_table::hash : Custo O(1)

3.2 - Espaço

No pior caso, a complexidade de espaço nas árvores binárias de busca e no hash tem custo O(n).

4. Análise Experimental

4.1 - Depuração de desempenho e Desempenho computacional

Mesmo com grandes quantidades de comandos e palavras, o **gprof** não começa a acumular tempo na análise, com o métodos em 0 segundos.

Isso ocorre devido a maioria das coisas serem realizadas em tempo de execução, então a única mudança que será notável é a quantidade de chamadas que será registrada no **gprof**

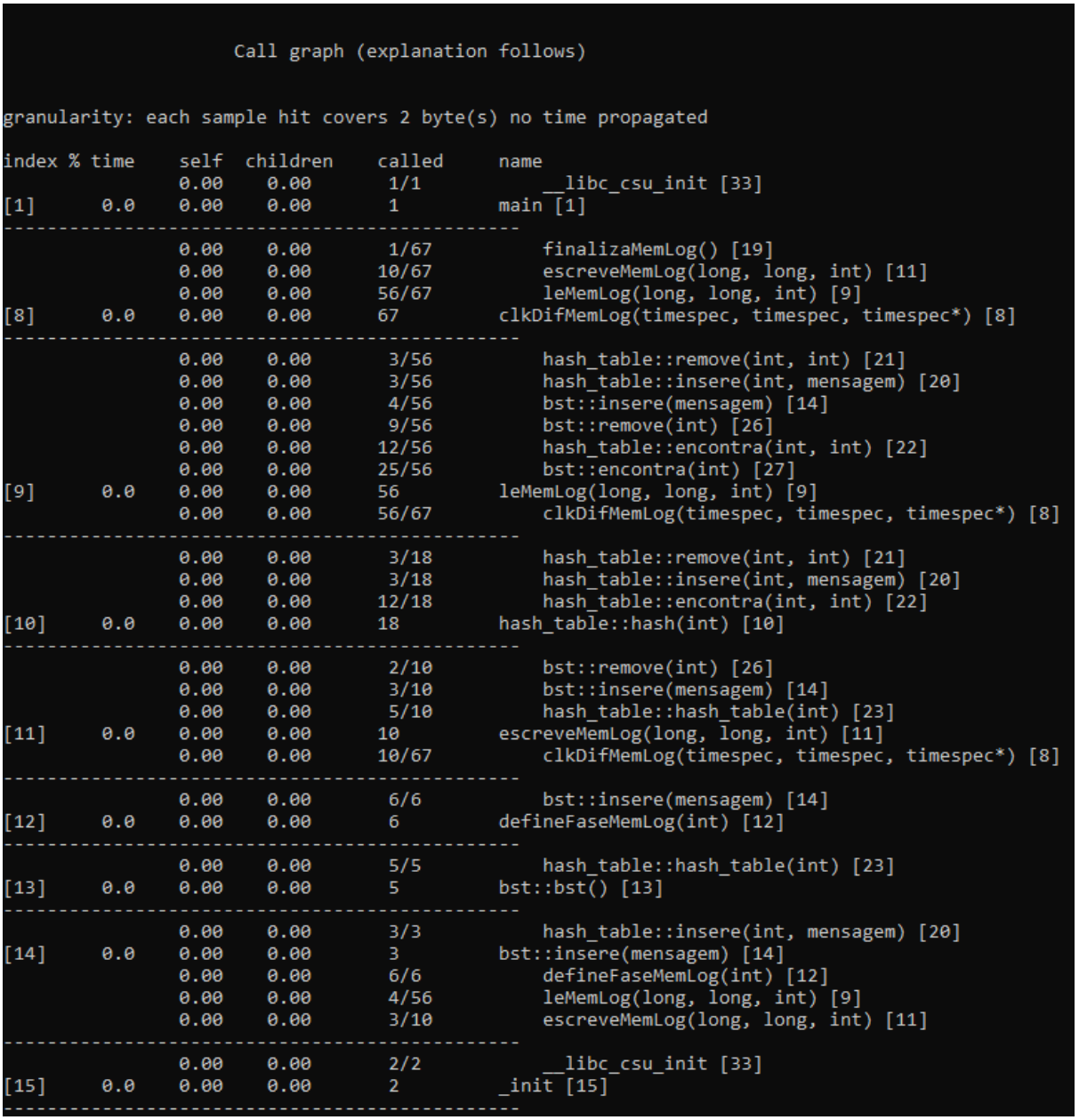
```
delareti@Delareti:~/tp3$ gprof bin/tp3 gmon.out
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00       67     0.00     0.00  clkDifMemLog(timespec, timespec, timespec*)
  0.00      0.00     0.00       56     0.00     0.00  leMemLog(long, long, int)
  0.00      0.00     0.00       18     0.00     0.00  hash_table::hash(int)
  0.00      0.00     0.00       10     0.00     0.00  escreveMemLog(long, long, int)
  0.00      0.00     0.00        6     0.00     0.00  defineFaseMemLog(int)
  0.00      0.00     0.00        5     0.00     0.00  bst::bst()
  0.00      0.00     0.00        3     0.00     0.00  bst::insere(mensagem)
  0.00      0.00     0.00        2     0.00     0.00  _init
  0.00      0.00     0.00        1     0.00     0.00  main
```

Utilizado com a Entrada 1 (a maior fornecida).

Em relação ao desempenho computacional, apesar de ter-se utilizado a biblioteca **memlog** devido às estruturas de dados utilizadas, a performance terá apenas pequenas mudanças na quantidade de chamadas de função, pois está é uma implementação estática onde o tempo será zero, devido aos métodos serem executados em tempo de compilação.

4.2 - Acesso à Memória e Localidade de Referência

Tendo em vista o teste de entrada 1, vamos analisar as chamadas de métodos realizados pelo programa.



```
                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00       1/1           __libc_csu_init [33]
[1]     0.0     0.00    0.00       1         main [1]
-----------------------------------------------
                0.00    0.00       1/67          finalizaMemLog() [19]
                0.00    0.00      10/67          escreveMemLog(long, long, int) [11]
                0.00    0.00      56/67          leMemLog(long, long, int) [9]
[8]     0.0     0.00    0.00      67         clkDifMemLog(timespec, timespec, timespec*) [8]
-----------------------------------------------
                0.00    0.00       3/56          hash_table::remove(int, int) [21]
                0.00    0.00       3/56          hash_table::insere(int, mensagem) [20]
                0.00    0.00       4/56          bst::insere(mensagem) [14]
                0.00    0.00       9/56          bst::remove(int) [26]
                0.00    0.00      12/56          hash_table::encontra(int, int) [22]
                0.00    0.00      25/56          bst::encontra(int) [27]
[9]     0.0     0.00    0.00      56         leMemLog(long, long, int) [9]
                0.00    0.00      56/67          clkDifMemLog(timespec, timespec, timespec*) [8]
-----------------------------------------------
                0.00    0.00       3/18          hash_table::remove(int, int) [21]
                0.00    0.00       3/18          hash_table::insere(int, mensagem) [20]
                0.00    0.00      12/18          hash_table::encontra(int, int) [22]
[10]    0.0     0.00    0.00      18         hash_table::hash(int) [10]
-----------------------------------------------
                0.00    0.00       2/10          bst::remove(int) [26]
                0.00    0.00       3/10          bst::insere(mensagem) [14]
                0.00    0.00       5/10          hash_table::hash_table(int) [23]
[11]    0.0     0.00    0.00      10         escreveMemLog(long, long, int) [11]
                0.00    0.00      10/67          clkDifMemLog(timespec, timespec, timespec*) [8]
-----------------------------------------------
                0.00    0.00       6/6           bst::insere(mensagem) [14]
[12]    0.0     0.00    0.00       6         defineFaseMemLog(int) [12]
-----------------------------------------------
                0.00    0.00       5/5           hash_table::hash_table(int) [23]
[13]    0.0     0.00    0.00       5         bst::bst() [13]
-----------------------------------------------
                0.00    0.00       3/3           hash_table::insere(int, mensagem) [20]
[14]    0.0     0.00    0.00       3         bst::insere(mensagem) [14]
                0.00    0.00       6/6           defineFaseMemLog(int) [12]
                0.00    0.00       4/56          leMemLog(long, long, int) [9]
                0.00    0.00       3/10          escreveMemLog(long, long, int) [11]
-----------------------------------------------
                0.00    0.00       2/2           __libc_csu_init [33]
[15]    0.0     0.00    0.00       2         _init [15]
-----------------------------------------------
```

Ao analisar a imagem, observando a chamada dos métodos e as chamadas de funções. Como a quantidade de chamadas esperada e a quantidade apresentada são condizentes, e tomos as funções subordinadas aos métodos pais, podemos

inferir que a localidade de referência está alinhada do que se esperava e comprovar que os métodos são chamados de forma correta, sem chamadas desnecessárias ou utilização de recursos indesejados. O gráfico de distância de pilha não foi inserido na documentação pois ficou bem distante os valores de pilha.

5. Robustez

Algumas estratégias de robustez já foram citadas e explicadas durante a documentação, mas agora vamos ver algumas que ainda não foram citadas ou são de grande importância para o programa. Para lidar com argumentos inválidos como erro ao abrir o arquivo de entrada, nome do arquivo não informado e erro ao abrir arquivo de saída e remover o espaço no início da mensagem são tratados na **main.**

Além disso, outras estratégias adotadas foram colocar algumas variáveis de classes como privadas, para não ocorrer erros de modificação por funções não permitidas.

6. Testes

Para realizar os teste de funcionamento com as entradas fornecidas pelos professores

6.1 - Caso entrada 1

Entrada fornecida

23

ENTREGA 5 103 6 Bom dia, meu amigo! Tudo bom?

ENTREGA 6 104 8 Boa tarde, minha amiga! Vou bem e você?

ENTREGA 5 105 18 Também! Não estou encontrando meu casaco, será que o deixei na sua casa no fim de semana passado?

ENTREGA 6 106 8 É um casaco longo e vermelho com bolsos?

ENTREGA 5 107 3 Sim, este mesmo.

ENTREGA 6 108 6 Está aqui, vou guardá-lo para você.

ENTREGA 5 109 8 Obrigada! Vou buscá-lo no próximo sábado. Até lá!

ENTREGA 6 110 1 Até!

CONSULTA 5 103

CONSULTA 6 104

CONSULTA 5 107

APAGA 5 103

CONSULTA 5 103

CONSULTA 6 110

Saída terminal

```
delareti@Delareti:~/tp3$ ./bin/tp3 -i entrada.txt -o saida.txt
delareti@Delareti:~/tp3$ cat saida.txt
OK: MENSAGEM 103 PARA 5 ARMAZENADA EM 5
OK: MENSAGEM 104 PARA 6 ARMAZENADA EM 6
OK: MENSAGEM 105 PARA 5 ARMAZENADA EM 5
OK: MENSAGEM 106 PARA 6 ARMAZENADA EM 6
OK: MENSAGEM 107 PARA 5 ARMAZENADA EM 5
OK: MENSAGEM 108 PARA 6 ARMAZENADA EM 6
OK: MENSAGEM 109 PARA 5 ARMAZENADA EM 5
OK: MENSAGEM 110 PARA 6 ARMAZENADA EM 6
CONSULTA 5 103: Bom dia, meu amigo! Tudo bom?
CONSULTA 6 104: Boa tarde, minha amiga! Vou bem e você?
CONSULTA 5 107: Sim, este mesmo.
OK: MENSAGEM APAGADA
CONSULTA 5 103: MENSAGEM INEXISTENTE
CONSULTA 6 110: Até!
delareti@Delareti:~/tp3$
```

6.2 - Caso entrada 2

Entrada fornecida

2

ENTREGA 3 111 4 Olá Professor, tudo bem?

ENTREGA 4 112 8 Boa noite! Tudo bem. Como posso te ajudar?

ENTREGA 3 113 12 Gostaria de entender melhor a nota do minha segunda prova, por favor.

ENTREGA 4 114 14 Certo, passe na minha sala na quinta-feira às 9 horas que vamos discutir juntos.

ENTREGA 3 115 1 Obrigado!

CONSULTA 4 114

CONSULTA 4 115

CONSULTA 3 115

Saída terminal

```
delareti@Delareti:~/tp3$ ./bin/tp3 -i entrada.txt -o saida.txt
delareti@Delareti:~/tp3$ cat saida.txt
OK: MENSAGEM 111 PARA 3 ARMAZENADA EM 1
OK: MENSAGEM 112 PARA 4 ARMAZENADA EM 0
OK: MENSAGEM 113 PARA 3 ARMAZENADA EM 1
OK: MENSAGEM 114 PARA 4 ARMAZENADA EM 0
OK: MENSAGEM 115 PARA 3 ARMAZENADA EM 1
CONSULTA 4 114: Certo, passe na minha sala na quinta-feira às 9 horas que vamos discutir juntos.
CONSULTA 4 115: MENSAGEM INEXISTENTE
CONSULTA 3 115: Obrigado!
delareti@Delareti:~/tp3$
```

6.3 - Caso entrada 3

Entrada fornecida

1

ENTREGA 1 116 14 Boa tarde José! Amanhã preciso que você busque uma encomenda na filial da Pampulha.

ENTREGA 2 117 9 Boa tarde Antônio! Certo, qual o endereço da loja?

ENTREGA 1 118 10 Avenida Antônio Carlos, 1000. Chegar lá e procurar por Flávio.

ENTREGA 2 119 5 Ok, estarei lá às 8h.

ENTREGA 1 120 1 Obrigado.

CONSULTA 1 116

CONSULTA 1 117

CONSULTA 2 117

Saída terminal

```
delareti@Delareti:~/tp3$ ./bin/tp3 -i entrada.txt -o saida.txt
delareti@Delareti:~/tp3$ cat saida.txt
OK: MENSAGEM 116 PARA 1 ARMAZENADA EM 0
OK: MENSAGEM 117 PARA 2 ARMAZENADA EM 0
OK: MENSAGEM 118 PARA 1 ARMAZENADA EM 0
OK: MENSAGEM 119 PARA 2 ARMAZENADA EM 0
OK: MENSAGEM 120 PARA 1 ARMAZENADA EM 0
CONSULTA 1 116: Boa tarde José! Amanhã preciso que você busque uma encomenda na filial da Pampulha.
CONSULTA 1 117: MENSAGEM INEXISTENTE
CONSULTA 2 117: Boa tarde Antônio! Certo, qual o endereço da loja?
delareti@Delareti:~/tp3$
```

6.4 - Caso entrada 4

Entrada fornecida

5

ENTREGA 5 121 10 Temos um presente para você! Clique aqui para saber mais.

ENTREGA 5 122 10 Aqui está a fatura digital do seu cartão de crédito.

ENTREGA 4 123 6 Nova mensagem no fórum do Moodle.

CONSULTA 5 121

CONSULTA 5 122

CONSULTA 4 123

APAGA 5 121

CONSULTA 5 121

CONSULTA 5 122

CONSULTA 4 123

APAGA 5 122

CONSULTA 5 121

CONSULTA 5 122

CONSULTA 4 123

APAGA 5 123

CONSULTA 5 121

CONSULTA 5 122

CONSULTA 4 123

Saída terminal

```
delareti@Delareti:~/tp3$ ./bin/tp3 -i entrada.txt -o saida.txt
delareti@Delareti:~/tp3$ cat saida.txt
OK: MENSAGEM 121 PARA 5 ARMAZENADA EM 0
OK: MENSAGEM 122 PARA 5 ARMAZENADA EM 0
OK: MENSAGEM 123 PARA 4 ARMAZENADA EM 4
CONSULTA 5 121: Temos um presente para você! Clique aqui para saber mais.
CONSULTA 5 122: Aqui está a fatura digital do seu cartão de crédito.
CONSULTA 4 123: Nova mensagem no fórum do Moodle.
OK: MENSAGEM APAGADA
CONSULTA 5 121: MENSAGEM INEXISTENTE
CONSULTA 5 122: Aqui está a fatura digital do seu cartão de crédito.
CONSULTA 4 123: Nova mensagem no fórum do Moodle.
OK: MENSAGEM APAGADA
CONSULTA 5 121: MENSAGEM INEXISTENTE
CONSULTA 5 122: MENSAGEM INEXISTENTE
CONSULTA 4 123: Nova mensagem no fórum do Moodle.
ERRO: MENSAGEM INEXISTENTE
CONSULTA 5 121: MENSAGEM INEXISTENTE
CONSULTA 5 122: MENSAGEM INEXISTENTE
CONSULTA 4 123: Nova mensagem no fórum do Moodle.
delareti@Delareti:~/tp3$
```

Todos os testes fornecidos foram testados e as saídas do programa produzido foram iguais aos inputs/ Outputs fornecidos.

Vídeo de alguns dos testes funcionando.

Google Drive:
https://drive.google.com/file/d/1STZFusCKCcD2nm0xLayRkQwR84MwlSrt/view?usp=sharing

Dropbox:
https://www.dropbox.com/s/dsgpkp4wo6rfp1b/TP3%20FINAL%20TESTE.mp4?dl=0

7. Conclusão

Ao final deste trabalho, podemos inferir que este possui o propósito de abordar, trabalhar e ampliar a visão sobre o funcionamento de busca em árvores binária e utilização de hash e como elas se relacionam em uma aplicação, neste caso, o armazenamento e a busca dos elementos que eram realizadas com a junção dessas duas estruturas.

A implementação desta proposta, além de trabalharmos novas estruturas de dados que foram abordadas em sala de aula, trouxe também uma visão mercadológica e estratégica, pois neste caso, não tínhamos tanta liberdade para a escolha das estruturas e as classes que deveriam ser implementadas. Dessa forma, facilitando na implementação das estruturas preestabelecidas, mas permitindo um pouco de criatividade dentro do escopo exigido. Sendo assim, uma experiência de trabalho distinta dos outros realizados anteriormente e podendo praticar as mais diversas operações e modificações dentro destas estruturas.

Bibliografia

BACKES, A. R. (2018). *Linguagem C - Completa e Descomplicada*

Geeks for Geeks - Hashing Data Structure

Geeks for Geeks - Binary Search Tree

GNUPLOT - A Brief Manual and Tutorial

GPROF Tutorial – How to use Linux GNU GCC Profiling Tool

Instruções para compilação e execução

Para executar o programa basta dar um comando **make** para executar tudo. Alguns comandos adicionais são.

make all - compila tudo
make distclean - remove objetos apenas
make clean - remove objetos e executável

Após executar o make, basta executar este comando na pasta raiz

Exemplo: ./bin/tp3 -i entrada.txt -o saida.txt

Sendo -i (Arquivo de entrada para ser processado)
-o (Endereço do arquivo de saída)

Passando o executável que está na pasta /bin e os parâmetros de arquivo de entrada, arquivo de saída, tamanho mediana e tamanho partição. Lembrando que para o normal funcionamento você deverá colocar valores válidos ou o programa irá retornar um aviso.

Sempre que mudar quaisquer valores de entrada, você deverá executar novamente o arquivo da pasta bin com seus respectivos parâmetros para atualizar os valores da saída para os valores atuais.

Não remover o arquivo log_mem.txt pois este foi utilizado para implementar a biblioteca **memlog** para estimar a performance do programa, apesar de não ter sido eficiente devido às estruturas de dados utilizadas.