Documentação Trabalho Prático 3

Lucas Roberto Santos Avelar

Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG) - Belo Horizonte – MG – Brasil

lucasrsavelar@ufmg.br

1. Introdução

Esta documentação lida com o problema da simulação de um servidor de mensagens eletrônicas (*emails*). O objetivo é implementar um programa que receba um arquivo contendo comandos de envio, pesquisa e remoção de mensagens, além de argumentos passados pela linha de comando (nome dos arquivos de entrada e saída), analise todas essas entradas e variáveis e escreva no arquivo de saída os devidos retornos para cada comando de entrada (confirmando a entrega ou remoção de uma mensagem, imprimindo uma mensagem pesquisada ou expressando a ocorrência de algum erro na execução do comando). Para resolver o problema, foi escrito um código que utilizava principalmente dois algoritmos de pesquisa em memória primária: uma Tabela Hash, que representa o servidor de emails, e uma Árvore Binária de Pesquisa, que representa a caixa de entrada dos usuários. Foi implementada também uma estrutura auxiliar que representa as mensagens dos usuários, além de funções próprias de cada algoritmo para realizar as inserções, remoções e pesquisas de emails.

2. Implementação

O código organiza-se principalmente em 2 blocos. Dentro dos arquivos, encontram-se alguns comentários sobre as operações de cada função. A seguir, há uma análise mais detalhada.

O primeiro bloco consiste nos arquivos “funcoes.hpp” e “funcoes.cpp” que reúnem tanto a declaração e a implementação das diversas funções fundamentais para a execução do programa quanto a declaração das estruturas necessárias, bem como de suas variáveis. As estruturas consistem em Email, que armazena uma mensagem e dois

inteiros que são os identificadores do usuário e do email, além de servir como nó das árvores binárias e possuir dois apontadores de Email para direita e esquerda; em ArvoreBinaria, que é a implementação de uma árvore binária de pesquisa com funções de inserção, remoção e pesquisa de nós, além de um apontador para Email que representa a raiz da árvore e um booleano auxiliar; e em TabelaHash, que é a implementação de uma tabela hash com funções de inserção, remoção e pesquisa na tabela e um array de árvores binárias que corresponde à tabela.

Nas funções, fora das estruturas temos somente a funcaoDispersao, que recebe o identificador do usuário e o tamanho da tabela e retorna o resto do primeiro dividido pelo segundo, o que é o índice da Tabela Hash em que devem ser armazenadas as mensagens do usuário cujo identificador foi passado como parâmetro. Dentro de ArvoreBinaria, temos insersaoAuxiliar, que recebe um email, uma mensagem e os identificadores de email e de usuário e insere o email na árvore. Para isso, ela é chamada recursivamente até encontrar um ponteiro nulo, que é onde ela cria um novo nó (email) e atribui a ele as informações passadas (mensagem e identificadores). Enquanto não encontra esse ponteiro, a função caminha para a esquerda da árvore, caso o identificador do email novo seja menor que o atual, ou para a direita, caso contrário. A função pesquisaAuxiliar funciona de forma semelhante quanto à recursão, caminhando para a esquerda ou para a direita dependendo dos identificadores das mensagens e com a diferença de que chegar em um ponteiro nulo significa que a mensagem não existe, retornando então um email auxiliar com identificador negativo. Se o email atual possui tanto identificador do usuário quanto identificador do email iguais aos do procurado, significa que foi encontrado e o email, e ele é retornado. Porém, se o identificador do email for igual mas o do usuário for diferente, significa que aquele usuário não tem o email procurado, então é retornado também o email auxiliar. A função Antecessor recebe dois nós (emails) como parâmetros e ajuda no caso de remoção, quando o nó a ser removido tem dois filhos. Ela procura o filho mais à direita do nó à esquerda do que deve ser removido e, quando encontra, retorna atualizando e corrigindo as posições da árvore. A função remocaoAuxiliar recebe um email e os dois identificadores e possui funcionamento praticamente idêntico à de pesquisa, com a diferença de que ao encontrar o nó a ser removido, ele é eliminado e a árvore é corrigida (se o removido tem nenhum filho,

basta deletá-lo; com um filho basta subir esse filho e com dois filhos a função Antecessor é chamada). Finalmente, as funções insereArvore, removeArvore e Pesquisa apenas chamam suas respectivs funções auxiliares passando no parâmetro do email a raiz da árvore.

Dentro de TabelaHash, a função inicializaTabela recebe o tamanho da tabela como parâmetro e aloca um array de árvores binárias baseado nesse tamanho. A função insereHash recebe uma string, os dois identificadores e o tamanho da tabela, chama funcaoDispersao para obter o índice da tabela em que o email deve ser armazenado e chama a função insereArvore da árvore daquele índice. A função pesquisaHash funciona da mesma forma, com a diferença de que ela retorna o email pesquisado (ou um email auxiliar que indica a inexistência do pesquisado), e a função removeHash inicialmente também funciona da mesma forma, mas retorna o booleano da árvore binária que indica se a remoção ocorreu com sucesso (retorna true quando esse booleano é false) ou se houve algum erro (mensagem inexistente ou já apagada, retornando false quando o booleano é false).

Já o segundo bloco corresponde ao arquivo “main.cpp”, onde está localizada a função main do programa. Primeiro, ela declara as variáveis de entrada e saída, obtém essas variáveis da linha de comando e cria os arquivos de entrada e saída basedo nessa entrada. O tamanho da Tabela Hash a ser criada é lido do arquivo de entrada, ela é criada e chama sua função de inicialização passando o tamanho como parâmetro. O programa então entra em laço até o final do arquivo para leitura das operações. Se a operação for do tipo ENTREGA, são lidos da entrada o identificador do usuário, o identificador do email e o número de palavras na mensagem. É lida a primeira palavra, ela é adicionada à string de mensagem e então inicia-se um laço para adicionar à string um espaço e a próxima palavra, até que todas sejam lidas. Chama-se então a função de inserção da Tabela Hash, a função de dispersão é chamada para retornar o índice e as informações necessárias são impressas no arquivo de saída. Se for uma CONSULTA, são lidos da entrada o identificador do usuário e o identificador do email e uma variável temporária do tipo Email recebe o retorno da função de pesquisa da Tabela Hash. Caso esse email temporário possua identificador válido (não negativo), a consulta ocorreu com sucesso e as informações são impressas no arquivo de saída. Do contrário, o email não existe/foi apagado e a mensagem de erro é impressa na saída. Finalmente, se a operação for APAGA são

lidos da entrada o identificador do usuário e o identificador do email e um booleano auxiliar recebe o retorno da função de remoção da Tabela Hash. Se esse booleano for verdadeiro, a remoção ocorreu com sucesso e as informações são impressas no arquivo de saída. Do contrário, o email não existe/foi apagado e a mensagem de erro é impressa na saída. Ao finalizar a entrada, os arquivos de entrada e saída são fechados e encerra-se o programa.

O programa foi testado em um computador com 4GB de memória RAM e processador intel Core i3, sistema operacional Windows 10 (mas a partir de um subsistema WSL Ubuntu), utilizando da linguagem C++ e com o compilador g++.

3. **Análise de Complexidade**

Complexidade de Tempo: dentro do arquivo “funcoes.cpp”, temos que a função funcaoDispersao apresenta complexidade de tempo O(1), visto que consiste apenas de um retorno. As funções auxiliares da estrutura ArvoreBinaria possuem complexidades de tempo que dependem da entrada. Tanto a inserção quanto a remoção e a pesquisa dependem do número de elementos dentro da árvore e do seu formato. O melhor caso dessas funções é O(1), e ocorre quando o elemento a ser pesquisado ou removido está na raiz e a inserção é também na raiz. O pior caso ocorre quando temos entradas ordenadas, pois a árvore binária torna-se então uma lista encadeada e o custo para pesquisar, remover ou inserir um elemento no final da lista é dado por O(n). Finalmente, o caso médio e também o caso mais comum/frequente em uma árvore ordenada é quando temos entradas variadas que fazem a árvore crescer tanto para a direita quanto para a esquerda, e nesse caso as funções de pesquisa, remoção e inserção apresentam complexidade O(log n). Como as funções principais de ArvoreBinaria e também as funções de TabelaHash são basicamente atribuições, retornos e chamadas das funções auxiliares, elas possuem a mesma complexidade que varia de acordo com a entrada. Finalmente, a função inicializaTabela consiste apenas em uma atribuição, então também tem complexidade O(1). No arquivo “main.cpp”, a maior complexidade encontra-se nos diversos laços. O primeiro deles é a leitura dos argumentos da linha de comando, com complexidade O(n/2) em que n é o número de argumentos; o próximo é o laço de leitura do arquivo de entrada que itera sobre todos os n elementos presentes no arquivo de texto, sendo assim O(n); e

também o laço de leitura da mensagem, com complexidade O(n – 1) para n palavras.

Complexidade de Espaço: as funções realizam as operações levando em conta uma Tabela Hash que aloca de forma dinâmica um array de árvores binárias. Tendo então uma tabela hash com tamanho x e levando em consideração que cada árvore binária tem tamanho n, a complexidade de espaço do programa é dada por O(x*n).

4. **Estratégias de Robustez**

As estratégias de robustez adotadas foram as seguintes:

- Considerar que a saída será dada num arquivo denominado “saida.txt”, caso o parâmetro de saída não seja informado.
- Tratar o caso em que mais de um usuário é mapeado para a mesma árvore binária e é feita uma consulta pelo mesmo email, mas passando como parâmetro diferentes identificadores de usuário (vide exemplo 3 das entradas de teste). Para isso, o programa verifica se tanto o identificador do usuário quanto o identificador do email são iguais aos do email pesquisado; se o identificador de email for igual mas o de usuário for diferente é retornado que o email procurado não existe

5. **Testes**

Os testes foram realizados com auxílio dos arquivos de entrada disponibilizados no Moodle. Os 4 arquivos de entrada, com diferentes operações e diferentes quantidades de chamada de cada operação, foram fornecidos ao programa. Além disso, foi utilizado um gerador de carga disponibilizado nos fóruns por um aluno da disciplina que também apresenta variações em relação às operações e à frequência de cada uma.

6. **Análise Experimental**

A análise de Desempenho Computacional foi efetuada após submeter o programa a entradas com diferentes tamanhos para a tabela e

diferentes quantidades de operações. Primeiro, foram analisados os resultados das 4 entradas de testes disponibilizadas no minhaUFMG. Nenhuma delas foi suficiente para provocar um uso do programa que registrasse dados válidos, ficando a seguir o respectivo resultado de cada uma:

```
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       34     0.00     0.00  Email::Email()
  0.00      0.00     0.00       31     0.00     0.00  Email::~Email()
  0.00      0.00     0.00       26     0.00     0.00  bool std::operator==<char, std::char_traits<char>, std::a
  0.00      0.00     0.00       23     0.00     0.00  ArvoreBinaria::ArvoreBinaria()
  0.00      0.00     0.00       22     0.00     0.00  funcaoDispercao(int, int)
  0.00      0.00     0.00        8     0.00     0.00  TabelaHash::InsereHash(std::__cxx11::basic_string<char, s
  0.00      0.00     0.00        8     0.00     0.00  ArvoreBinaria::insereArvore(std::__cxx11::basic_string<ch
  0.00      0.00     0.00        8     0.00     0.00  ArvoreBinaria::insercaoAuxiliar(Email*&, std::__cxx11::ba
  0.00      0.00     0.00        5     0.00     0.00  TabelaHash::PesquisaHash(int, int, int)
  0.00      0.00     0.00        5     0.00     0.00  ArvoreBinaria::pesquisaAuxiliar(Email*, int, int)
  0.00      0.00     0.00        5     0.00     0.00  ArvoreBinaria::Pesquisa(int, int)
  0.00      0.00     0.00        5     0.00     0.00  Email::operator=(Email&&)
  0.00      0.00     0.00        4     0.00     0.00  Email::Email(Email const&)
  0.00      0.00     0.00        1     0.00     0.00  _GLOBAL__sub_I__Z15funcaoDispercaoii
  0.00      0.00     0.00        1     0.00     0.00  _GLOBAL__sub_I_main
  0.00      0.00     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
  0.00      0.00     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
  0.00      0.00     0.00        1     0.00     0.00  TabelaHash::RemoveHash(int, int, int)
  0.00      0.00     0.00        1     0.00     0.00  TabelaHash::inicializaTabela(int)
  0.00      0.00     0.00        1     0.00     0.00  ArvoreBinaria::removeArvore(int, int)
  0.00      0.00     0.00        1     0.00     0.00  ArvoreBinaria::remocaoAuxiliar(Email*&, int, int)
  0.00      0.00     0.00        1     0.00     0.00  Email::Email(Email&&)
```

```
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       22     0.00     0.00  Email::Email()
  0.00      0.00     0.00       20     0.00     0.00  Email::~Email()
  0.00      0.00     0.00       16     0.00     0.00  bool std::operator==<char, std::char_traits<char>, std::al
  0.00      0.00     0.00       13     0.00     0.00  funcaoDispercao(int, int)
  0.00      0.00     0.00        5     0.00     0.00  TabelaHash::InsereHash(std::__cxx11::basic_string<char, st
  0.00      0.00     0.00        5     0.00     0.00  ArvoreBinaria::insereArvore(std::__cxx11::basic_string<cha
  0.00      0.00     0.00        5     0.00     0.00  ArvoreBinaria::insercaoAuxiliar(Email*&, std::__cxx11::bas
  0.00      0.00     0.00        3     0.00     0.00  TabelaHash::PesquisaHash(int, int, int)
  0.00      0.00     0.00        3     0.00     0.00  ArvoreBinaria::pesquisaAuxiliar(Email*, int, int)
  0.00      0.00     0.00        3     0.00     0.00  ArvoreBinaria::Pesquisa(int, int)
  0.00      0.00     0.00        3     0.00     0.00  Email::operator=(Email&&)
  0.00      0.00     0.00        2     0.00     0.00  ArvoreBinaria::ArvoreBinaria()
  0.00      0.00     0.00        2     0.00     0.00  Email::Email(Email const&)
  0.00      0.00     0.00        1     0.00     0.00  _GLOBAL__sub_I__Z15funcaoDispercaoii
  0.00      0.00     0.00        1     0.00     0.00  _GLOBAL__sub_I_main
  0.00      0.00     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
  0.00      0.00     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
  0.00      0.00     0.00        1     0.00     0.00  TabelaHash::inicializaTabela(int)
  0.00      0.00     0.00        1     0.00     0.00  Email::Email(Email&&)
```

Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
0.00	0.00	0.00	19	0.00	0.00	Email::Email()
0.00	0.00	0.00	17	0.00	0.00	Email::~Email()
0.00	0.00	0.00	16	0.00	0.00	bool std::operator==<char, std::char_traits<char>, std::a
0.00	0.00	0.00	13	0.00	0.00	funcaoDispercao(int, int)
0.00	0.00	0.00	5	0.00	0.00	TabelaHash::InsereHash(std::__cxx11::basic_string<char, s
0.00	0.00	0.00	5	0.00	0.00	ArvoreBinaria::insereArvore(std::__cxx11::basic_string<ch
0.00	0.00	0.00	5	0.00	0.00	ArvoreBinaria::insercaoAuxiliar(Email*&, std::__cxx11::ba
0.00	0.00	0.00	3	0.00	0.00	TabelaHash::PesquisaHash(int, int, int)
0.00	0.00	0.00	3	0.00	0.00	ArvoreBinaria::pesquisaAuxiliar(Email*, int, int)
0.00	0.00	0.00	3	0.00	0.00	ArvoreBinaria::Pesquisa(int, int)
0.00	0.00	0.00	3	0.00	0.00	Email::operator=(Email&&)
0.00	0.00	0.00	2	0.00	0.00	Email::Email(Email const&)
0.00	0.00	0.00	1	0.00	0.00	_GLOBAL__sub_I__Z15funcaoDispercaoii
0.00	0.00	0.00	1	0.00	0.00	_GLOBAL__sub_I_main
0.00	0.00	0.00	1	0.00	0.00	__static_initialization_and_destruction_0(int, int)
0.00	0.00	0.00	1	0.00	0.00	__static_initialization_and_destruction_0(int, int)
0.00	0.00	0.00	1	0.00	0.00	TabelaHash::inicializaTabela(int)
0.00	0.00	0.00	1	0.00	0.00	ArvoreBinaria::ArvoreBinaria()
0.00	0.00	0.00	1	0.00	0.00	Email::Email(Email&&)

Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
0.00	0.00	0.00	45	0.00	0.00	Email::~Email()
0.00	0.00	0.00	41	0.00	0.00	bool std::operator==<char, std::char_traits<char>, std::
0.00	0.00	0.00	36	0.00	0.00	Email::Email()
0.00	0.00	0.00	22	0.00	0.00	funcaoDispercao(int, int)
0.00	0.00	0.00	12	0.00	0.00	TabelaHash::PesquisaHash(int, int, int)
0.00	0.00	0.00	12	0.00	0.00	ArvoreBinaria::pesquisaAuxiliar(Email*, int, int)
0.00	0.00	0.00	12	0.00	0.00	ArvoreBinaria::Pesquisa(int, int)
0.00	0.00	0.00	12	0.00	0.00	Email::operator=(Email&&)
0.00	0.00	0.00	7	0.00	0.00	Email::Email(Email const&)
0.00	0.00	0.00	5	0.00	0.00	ArvoreBinaria::ArvoreBinaria()
0.00	0.00	0.00	5	0.00	0.00	Email::Email(Email&&)
0.00	0.00	0.00	3	0.00	0.00	TabelaHash::InsereHash(std::__cxx11::basic_string<char,
0.00	0.00	0.00	3	0.00	0.00	TabelaHash::RemoveHash(int, int, int)
0.00	0.00	0.00	3	0.00	0.00	ArvoreBinaria::insereArvore(std::__cxx11::basic_string<c
0.00	0.00	0.00	3	0.00	0.00	ArvoreBinaria::removeArvore(int, int)
0.00	0.00	0.00	3	0.00	0.00	ArvoreBinaria::remocaoAuxiliar(Email*&, int, int)
0.00	0.00	0.00	3	0.00	0.00	ArvoreBinaria::insercaoAuxiliar(Email*&, std::__cxx11::b
0.00	0.00	0.00	1	0.00	0.00	_GLOBAL__sub_I__Z15funcaoDispercaoii
0.00	0.00	0.00	1	0.00	0.00	_GLOBAL__sub_I_main
0.00	0.00	0.00	1	0.00	0.00	__static_initialization_and_destruction_0(int, int)
0.00	0.00	0.00	1	0.00	0.00	__static_initialization_and_destruction_0(int, int)
0.00	0.00	0.00	1	0.00	0.00	TabelaHash::inicializaTabela(int)

A seguir, o programa foi submetido a entradas geradas a partir de um gerador de carga, disponibilizado nos fóruns por um estudante. Inicialmente, foram entradas com tamanho de tabela e número de comandos iguais à 100, 200, 300, 500 e 1000; novamente sem dados significativos. As tabelas encontram-se a seguir, respectivamente:

Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
0.00	0.00	0.00	188	0.00	0.00	bool std::operator==<char, std::char_traits<char>, std::a
0.00	0.00	0.00	183	0.00	0.00	Email::Email()
0.00	0.00	0.00	171	0.00	0.00	Email::~Email()
0.00	0.00	0.00	169	0.00	0.00	funcaoDispercao(int, int)
0.00	0.00	0.00	100	0.00	0.00	ArvoreBinaria::ArvoreBinaria()
0.00	0.00	0.00	43	0.00	0.00	TabelaHash::InsereHash(std::__cxx11::basic_string<char, s
0.00	0.00	0.00	43	0.00	0.00	ArvoreBinaria::insereArvore(std::__cxx11::basic_string<ch
0.00	0.00	0.00	43	0.00	0.00	ArvoreBinaria::insercaoAuxiliar(Email*&, std::__cxx11::ba
0.00	0.00	0.00	31	0.00	0.00	TabelaHash::PesquisaHash(int, int, int)
0.00	0.00	0.00	31	0.00	0.00	ArvoreBinaria::pesquisaAuxiliar(Email*, int, int)
0.00	0.00	0.00	31	0.00	0.00	ArvoreBinaria::Pesquisa(int, int)
0.00	0.00	0.00	31	0.00	0.00	Email::operator=(Email&&)
0.00	0.00	0.00	30	0.00	0.00	Email::Email(Email&&)
0.00	0.00	0.00	26	0.00	0.00	TabelaHash::RemoveHash(int, int, int)
0.00	0.00	0.00	26	0.00	0.00	ArvoreBinaria::removeArvore(int, int)
0.00	0.00	0.00	26	0.00	0.00	ArvoreBinaria::remocaoAuxiliar(Email*&, int, int)
0.00	0.00	0.00	1	0.00	0.00	_GLOBAL__sub_I__Z15funcaoDispercaoii
0.00	0.00	0.00	1	0.00	0.00	_GLOBAL__sub_I_main
0.00	0.00	0.00	1	0.00	0.00	__static_initialization_and_destruction_0(int, int)
0.00	0.00	0.00	1	0.00	0.00	__static_initialization_and_destruction_0(int, int)
0.00	0.00	0.00	1	0.00	0.00	TabelaHash::inicializaTabela(int)
0.00	0.00	0.00	1	0.00	0.00	Email::Email(Email const&)

Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
0.00	0.00	0.00	417	0.00	0.00	bool std::operator==<char, std::char_traits<char>, std::a
0.00	0.00	0.00	343	0.00	0.00	Email::~Email()
0.00	0.00	0.00	335	0.00	0.00	Email::Email()
0.00	0.00	0.00	332	0.00	0.00	funcaoDispercao(int, int)
0.00	0.00	0.00	200	0.00	0.00	ArvoreBinaria::ArvoreBinaria()
0.00	0.00	0.00	72	0.00	0.00	TabelaHash::RemoveHash(int, int, int)
0.00	0.00	0.00	72	0.00	0.00	ArvoreBinaria::removeArvore(int, int)
0.00	0.00	0.00	72	0.00	0.00	ArvoreBinaria::remocaoAuxiliar(Email*&, int, int)
0.00	0.00	0.00	68	0.00	0.00	TabelaHash::PesquisaHash(int, int, int)
0.00	0.00	0.00	68	0.00	0.00	ArvoreBinaria::pesquisaAuxiliar(Email*, int, int)
0.00	0.00	0.00	68	0.00	0.00	ArvoreBinaria::Pesquisa(int, int)
0.00	0.00	0.00	68	0.00	0.00	Email::Email(Email&&)
0.00	0.00	0.00	68	0.00	0.00	Email::operator=(Email&&)
0.00	0.00	0.00	60	0.00	0.00	TabelaHash::InsereHash(std::__cxx11::basic_string<char, st
0.00	0.00	0.00	60	0.00	0.00	ArvoreBinaria::insereArvore(std::__cxx11::basic_string<cha
0.00	0.00	0.00	60	0.00	0.00	ArvoreBinaria::insercaoAuxiliar(Email*&, std::__cxx11::bas
0.00	0.00	0.00	1	0.00	0.00	_GLOBAL__sub_I__Z15funcaoDispercaoii
0.00	0.00	0.00	1	0.00	0.00	_GLOBAL__sub_I_main
0.00	0.00	0.00	1	0.00	0.00	__static_initialization_and_destruction_0(int, int)
0.00	0.00	0.00	1	0.00	0.00	__static_initialization_and_destruction_0(int, int)
0.00	0.00	0.00	1	0.00	0.00	TabelaHash::inicializaTabela(int)

Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
0.00	0.00	0.00	603	0.00	0.00	bool std::operator==<char, std::char_traits<char>, std::a
0.00	0.00	0.00	517	0.00	0.00	Email::Email()
0.00	0.00	0.00	513	0.00	0.00	Email::~Email()
0.00	0.00	0.00	499	0.00	0.00	funcaoDispercao(int, int)
0.00	0.00	0.00	300	0.00	0.00	ArvoreBinaria::ArvoreBinaria()
0.00	0.00	0.00	103	0.00	0.00	TabelaHash::InsereHash(std::__cxx11::basic_string<char, s
0.00	0.00	0.00	103	0.00	0.00	ArvoreBinaria::insereArvore(std::__cxx11::basic_string<ch
0.00	0.00	0.00	103	0.00	0.00	ArvoreBinaria::insercaoAuxiliar(Email*&, std::__cxx11::ba
0.00	0.00	0.00	99	0.00	0.00	TabelaHash::PesquisaHash(int, int, int)
0.00	0.00	0.00	99	0.00	0.00	ArvoreBinaria::pesquisaAuxiliar(Email*, int, int)
0.00	0.00	0.00	99	0.00	0.00	ArvoreBinaria::Pesquisa(int, int)
0.00	0.00	0.00	99	0.00	0.00	Email::Email(Email&&)
0.00	0.00	0.00	99	0.00	0.00	Email::operator=(Email&&)
0.00	0.00	0.00	97	0.00	0.00	TabelaHash::RemoveHash(int, int, int)
0.00	0.00	0.00	97	0.00	0.00	ArvoreBinaria::removeArvore(int, int)
0.00	0.00	0.00	97	0.00	0.00	ArvoreBinaria::remocaoAuxiliar(Email*&, int, int)
0.00	0.00	0.00	1	0.00	0.00	_GLOBAL__sub_I__Z15funcaoDispercaoii
0.00	0.00	0.00	1	0.00	0.00	_GLOBAL__sub_I_main
0.00	0.00	0.00	1	0.00	0.00	__static_initialization_and_destruction_0(int, int)
0.00	0.00	0.00	1	0.00	0.00	__static_initialization_and_destruction_0(int, int)
0.00	0.00	0.00	1	0.00	0.00	TabelaHash::inicializaTabela(int)

Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
0.00	0.00	0.00	1019	0.00	0.00	bool std::operator==<char, std::char_traits<char>, std::a
0.00	0.00	0.00	848	0.00	0.00	Email::Email()
0.00	0.00	0.00	845	0.00	0.00	Email::~Email()
0.00	0.00	0.00	837	0.00	0.00	funcaoDispercao(int, int)
0.00	0.00	0.00	500	0.00	0.00	ArvoreBinaria::ArvoreBinaria()
0.00	0.00	0.00	174	0.00	0.00	TabelaHash::RemoveHash(int, int, int)
0.00	0.00	0.00	174	0.00	0.00	ArvoreBinaria::removeArvore(int, int)
0.00	0.00	0.00	174	0.00	0.00	ArvoreBinaria::remocaoAuxiliar(Email*&, int, int)
0.00	0.00	0.00	164	0.00	0.00	TabelaHash::InsereHash(std::__cxx11::basic_string<char, s
0.00	0.00	0.00	164	0.00	0.00	ArvoreBinaria::insereArvore(std::__cxx11::basic_string<ch
0.00	0.00	0.00	164	0.00	0.00	ArvoreBinaria::insercaoAuxiliar(Email*&, std::__cxx11::ba
0.00	0.00	0.00	161	0.00	0.00	TabelaHash::PesquisaHash(int, int, int)
0.00	0.00	0.00	161	0.00	0.00	ArvoreBinaria::pesquisaAuxiliar(Email*, int, int)
0.00	0.00	0.00	161	0.00	0.00	ArvoreBinaria::Pesquisa(int, int)
0.00	0.00	0.00	161	0.00	0.00	Email::Email(Email&&)
0.00	0.00	0.00	161	0.00	0.00	Email::operator=(Email&&)
0.00	0.00	0.00	1	0.00	0.00	_GLOBAL__sub_I__Z15funcaoDispercaoii
0.00	0.00	0.00	1	0.00	0.00	_GLOBAL__sub_I_main
0.00	0.00	0.00	1	0.00	0.00	__static_initialization_and_destruction_0(int, int)
0.00	0.00	0.00	1	0.00	0.00	__static_initialization_and_destruction_0(int, int)
0.00	0.00	0.00	1	0.00	0.00	TabelaHash::inicializaTabela(int)

```
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     2020     0.00     0.00  bool std::operator==<char, std::char_traits<char>, std::a
  0.00      0.00     0.00     1744     0.00     0.00  Email::~Email()
  0.00      0.00     0.00     1731     0.00     0.00  Email::Email()
  0.00      0.00     0.00     1657     0.00     0.00  funcaoDispercao(int, int)
  0.00      0.00     0.00     1000     0.00     0.00  ArvoreBinaria::ArvoreBinaria()
  0.00      0.00     0.00      339     0.00     0.00  TabelaHash::PesquisaHash(int, int, int)
  0.00      0.00     0.00      339     0.00     0.00  ArvoreBinaria::pesquisaAuxiliar(Email*, int, int)
  0.00      0.00     0.00      339     0.00     0.00  ArvoreBinaria::Pesquisa(int, int)
  0.00      0.00     0.00      339     0.00     0.00  Email::Email(Email&&)
  0.00      0.00     0.00      339     0.00     0.00  Email::operator=(Email&&)
  0.00      0.00     0.00      333     0.00     0.00  TabelaHash::RemoveHash(int, int, int)
  0.00      0.00     0.00      333     0.00     0.00  ArvoreBinaria::removeArvore(int, int)
  0.00      0.00     0.00      333     0.00     0.00  ArvoreBinaria::remocaoAuxiliar(Email*&, int, int)
  0.00      0.00     0.00      326     0.00     0.00  TabelaHash::InsereHash(std::__cxx11::basic_string<char, s
  0.00      0.00     0.00      326     0.00     0.00  ArvoreBinaria::insereArvore(std::__cxx11::basic_string<ch
  0.00      0.00     0.00      326     0.00     0.00  ArvoreBinaria::insercaoAuxiliar(Email*&, std::__cxx11::ba
  0.00      0.00     0.00        1     0.00     0.00  _GLOBAL__sub_I__Z15funcaoDispercaoii
  0.00      0.00     0.00        1     0.00     0.00  _GLOBAL__sub_I_main
  0.00      0.00     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
  0.00      0.00     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
  0.00      0.00     0.00        1     0.00     0.00  TabelaHash::inicializaTabela(int)
```

Uma entrada com 10000 espaços na Tabela Hash e 10000 comandos foi, finalmente, capaz de gerar dados úteis:

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 50.09      0.01     0.01    22278     0.22     0.22  bool std::operator==<char, std::char_traits<char>, std::a
 50.09      0.01     0.01     3340     1.50     1.50  Email::operator=(Email&&)
  0.00      0.01     0.00    17978     0.00     0.00  Email::~Email()
  0.00      0.01     0.00    17937     0.00     0.00  Email::Email()
  0.00      0.01     0.00    16622     0.00     0.00  funcaoDispercao(int, int)
  0.00      0.01     0.00    10000     0.00     0.00  ArvoreBinaria::ArvoreBinaria()
  0.00      0.01     0.00     3342     0.00     0.00  TabelaHash::RemoveHash(int, int, int)
  0.00      0.01     0.00     3342     0.00     0.00  ArvoreBinaria::removeArvore(int, int)
  0.00      0.01     0.00     3342     0.00     0.00  ArvoreBinaria::remocaoAuxiliar(Email*&, int, int)
  0.00      0.01     0.00     3340     0.00     0.00  TabelaHash::PesquisaHash(int, int, int)
  0.00      0.01     0.00     3340     0.00     0.00  ArvoreBinaria::pesquisaAuxiliar(Email*, int, int)
  0.00      0.01     0.00     3340     0.00     0.00  ArvoreBinaria::Pesquisa(int, int)
  0.00      0.01     0.00     3340     0.00     0.00  Email::Email(Email&&)
  0.00      0.01     0.00     3299     0.00     0.00  TabelaHash::InsereHash(std::__cxx11::basic_string<char, s
  0.00      0.01     0.00     3299     0.00     0.00  ArvoreBinaria::insereArvore(std::__cxx11::basic_string<ch
  0.00      0.01     0.00     3299     0.00     0.00  ArvoreBinaria::insercaoAuxiliar(Email*&, std::__cxx11::ba
  0.00      0.01     0.00        1     0.00     0.00  _GLOBAL__sub_I__Z15funcaoDispercaoii
  0.00      0.01     0.00        1     0.00     0.00  _GLOBAL__sub_I_main
  0.00      0.01     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
  0.00      0.01     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
  0.00      0.01     0.00        1     0.00     0.00  TabelaHash::inicializaTabela(int)
```

E uma entrada com 50000 posições e 50000 comandos gerou os melhores dados:

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 50.09      0.02     0.02    16360     1.22     1.22  ArvoreBinaria::Pesquisa(int, int)
 25.05      0.03     0.01    16825     0.60     0.60  ArvoreBinaria::remocaoAuxiliar(Email*&, int, int)
 25.05      0.04     0.01                             main
  0.00      0.04     0.00   106263     0.00     0.00  bool std::operator==<char, std::char_traits<char>, std::al
  0.00      0.04     0.00    87718     0.00     0.00  Email::Email()
  0.00      0.04     0.00    87357     0.00     0.00  Email::~Email()
  0.00      0.04     0.00    83452     0.00     0.00  funcaoDispercao(int, int)
  0.00      0.04     0.00    50000     0.00     0.00  ArvoreBinaria::ArvoreBinaria()
  0.00      0.04     0.00    16825     0.00     0.60  TabelaHash::RemoveHash(int, int, int)
  0.00      0.04     0.00    16825     0.00     0.60  ArvoreBinaria::removeArvore(int, int)
  0.00      0.04     0.00    16721     0.00     0.00  TabelaHash::InsereHash(std::__cxx11::basic_string<char, st
  0.00      0.04     0.00    16721     0.00     0.00  ArvoreBinaria::insereArvore(std::__cxx11::basic_string<cha
  0.00      0.04     0.00    16721     0.00     0.00  ArvoreBinaria::insercaoAuxiliar(Email*&, std::__cxx11::bas
  0.00      0.04     0.00    16360     0.00     1.22  TabelaHash::PesquisaHash(int, int, int)
  0.00      0.04     0.00    16360     0.00     0.00  ArvoreBinaria::pesquisaAuxiliar(Email*, int, int)
  0.00      0.04     0.00    16360     0.00     0.00  Email::Email(Email&&)
  0.00      0.04     0.00    16360     0.00     0.00  Email::operator=(Email&&)
  0.00      0.04     0.00        1     0.00     0.00  _GLOBAL__sub_I__Z15funcaoDispercaoii
  0.00      0.04     0.00        1     0.00     0.00  _GLOBAL__sub_I_main
  0.00      0.04     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
  0.00      0.04     0.00        1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
  0.00      0.04     0.00        1     0.00     0.00  TabelaHash::inicializaTabela(int)
```

Percebe-se que em praticamente metade do tempo empregado pelo programa, a função de Pesquisa da Árvore Binária estava em execução. É compreensível, dado que dependendo da entrada, essa é uma das funções que apresenta maior complexidade (O(n) no pior caso). O mesmo é válido para a função de remoção, cuja análise é equivalente à função de pesquisa. É válido também ressaltar que funcaoDispersao, mesmo sendo chamada um número relativamente alto de vezes (mais de 83 mil), praticamente não gastou tempo de execução, confirmando sua complexidade O(1). No geral, é possível afirmar que o programa executa com velocidade consideravelmente boa (uma entrada de 50 mil gastou apenas 0,04 segundos) dado que as funções empregadas na Tabela Hash e na Árvore Binária apresentam complexidades simples (O(log n) na maioria dos casos), e mesmo a complexidade mais alta (O(n)) é bastante aceitável.

7. **Conclusões**

O trabalho consistiu na implementação de um programa que simula um servidor de emails a partir da utilização de uma Tabela Hash de Árvores Binárias e que recebe como entrada um arquivo de texto contendo diversas operações que esse servidor de emails deve suportar (como entrega, pesquisa e remoção de mensagens), além de imprimir num arquivo de saída mensagens de êxito ou de fracasso dependendo do resultado de cada operação. Acredito que novamente houve um pouco

mais de simplicidade de realização em comparação com os trabalhos práticos anteriores (principalmente o primeiro), creio que muito por conta da aproximação do fim do semestre e pela falta de tempo para desenvolver/corrigir atividades mais complexas. Além disso, já ter visto as estruturas que deveriam ser implementadas no trabalho em sala de aula foi um fator crucial para o maior entendimento do que deveria ser feito em cada etapa do desenvolvimento. Ao contrário da minha escolha no trabalho prático anterior, dessa vez optei por utilizar um array alocado de forma dinâmica pois acredito que, pelo contexto, era o mais simples e econômico (em termos de memória) a ser feito. Diferentemente do TP2, que envolvia muitas trocas de posições e elevava exponencialmente a importância dos índices; no TP3 as principais funções (inserção, remoção e pesquisa) não fazem grande exigência de índices, permitindo o uso da alocação dinâmica de forma mais tranquila e até mesmo evitando o desperdício de espaço.

8. **Bibliografia**

- Material disponibilizado no minhaUFMG (slides sobre Tabela Hash e Árvores Binárias de pesquisa)
- Código proveniente das PA's anteriores (beecrowd, principalmente sobre Árvores Binárias de pesquisa)
- Código proveniente de Trabalhos Práticos anteriores (principalmente sobre argumentos da linha de comando)
- https://m.cplusplus.com/articles/DEN36Up4/
- https://www.geeksforgeeks.org/command-line-arguments-in-c-cpp/

9. **Instruções para compilar e executar**

- Antes de tudo, **favor acessar o diretório “bin”, presente dentro do diretório TP, e colocar nele o(s) arquivo(s) de entrada**
- Retorne ao diretório raiz TP
- Escreva e execute no terminal o comando “make”
- Acesse novamente o diretório “bin”
- Escreva e execute no terminal o comando “./tp3.exe”, em conjunto com os argumentos da linha de comando. Isso significa:

  [-i/-I] nome do arquivo de entrada.txt **(Favor usar arquivos .txt!)**

  [-o/-O] nome do arquivo de saída

  Exemplo de entrada: ./tp3.exe -i input.txt -o saida.txt

Observação: caso o parâmetro de arquivo de saída não seja informado, ele será considerado como “saida.txt”. É fundamental informar o arquivo de entrada!

A ordem em que os parâmetros serão passados não é relevante, desde que o indicador esteja seguido do parâmetro correto. Ou seja, para o exemplo acima:

./tp3.exe -o saida.txt -i input.txt também funciona

./tp3.exe -I input.txt -O saida.txt também funciona

./tp3.exe -O saida.txt -i input.txt também funciona

./tp3.exe -i saida.txt -o input.txt **NÃO!**

Também não faz diferença usar letras maiúsculas ou minúsculas nos indicadores, nem misturar maiúsculas com minúsculas.

- O arquivo de saída será gerado, normalmente, dentro do diretório “bin”
- Para novas entradas, acesse o diretório “bin” e execute novamente a linha de comandos