# Servidor de e-mails

**Marcelo Andrade**

github.com/marceloanje
linkedin.com/in/marcelo-andrade-10334b160
marcelo.anje@outlook.com

## 1. Introdução

Um servidor de e-mails é um serviço de hospedagem, que armazena e-mails, que partem de uma origem para um destinatário, esse armazenamento pode ser locado em uma rede online ou uma rede local.

Como um servidor pode armazenar várias quantidades diferentes de usuários, e cada usuário pode ter uma quantidade variante de e-mails, uma saída encontrada para esse problema é utilizar uma alocação dinâmica de memória.

Sendo assim, neste projeto foi implementado um software que simula um servidor de e-mails com memória dinâmica.

A seção é onde está a descrição do método adotado, a seção três, a análise de complexidade, a seção quatro, as estratégias de robustez adotadas, a seção cinco, a análise experimental, e por fim, a seção seis, a conclusão.

## 2. Método

### 2.1 Implementação

O programa foi desenvolvido na linguagem C++, compilada pelo WSL da Ubuntu e os testes foram realizados em um emulador de sistemas operacionais (VirtualBox) com Linux Ubuntu 20.04.4, sendo emulado em um computador com Windows 10, Ryzen 7 2.3 GHz e 8GB de memória RAM.

### 2.2 Estrutura de dados

A implementação do programa teve como base a estrutura de dados do tipo lista, que se caracteriza por sua alocação dinâmica. O programa possui duas listas principais, que são a lista do tipo item *Email*, lista essa, que se encontra na classe *Usuario*, e a lista do tipo item *Usuario*, que se encontra na classe *Servidor*.

Ou seja, um servidor possui uma lista de usuários, em que cada usuário possui uma lista de e-mails.

## 2.3 Formatos de dados

O sistema foi feito de tal forma, que durante sua execução é recebido um caminho o qual deve se encontrar um arquivo de entrada no formato *.txt*, o qual contém os dados de entradas que são tratados internamente, dessa forma a realizar todo mecanismo de modificação, inserção, remoção e consulta dos usuários do servidor. O resultado final é expelido através da tela.

## 2.4 Classes e métodos

O programa foi dividido primordialmente em três classes diferentes: classe ***Email***, classe ***Usuario***, classe ***Servidor***. Além dessas três classes, há também o arquivo ***main.cpp***.

A classe ***Email*** possui dois construtores, um default e um paramétrico, também possui alguns métodos básicos *get* e *set*, para manipulação dos seus atributos privados, que são: *mensagem*, do tipo *string*, *origem*, do tipo *int*, *destino*, do tipo *int, prioridade,* do tipo *int* e *prox* que é um ponteiro do tipo *Email*. Essa classe é *friend* com a classe *Usuario*.

A classe ***Usuario*** possui dois construtores, um default e um paramétrico, possui um destrutor, também possui alguns métodos básicos *get* e *set*, para manipulação dos seus atributos privados, que são: *ID*, do tipo *int, primeiro*, ponteiro do tipo *Email, ultimo*, ponteiro do tipo *Email* e *prox*, ponteiro do tipo *Usuario*. Também possui alguns métodos como *InsereOrdenado*, que é usado para inserir objetos do tipo *Email* na lista de objetos que possui a classe *Usuario* de acordo com sua prioridade, *removePrimeiroEmail*, que remove o primeiro *Email* da lista, dentre outros. Essa classe possui uma lista de objetos do tipo *Email*, que é alocada dinamicamente. Essa classe é *friend* com a classe *Servidor*.

A classe ***Servidor*** possui dois construtores, um default e um paramétrico, possui um destrutor, também possui alguns métodos básicos *get* e *set*, para manipulação dos seus atributos privados, que são: *tamanho*, do tipo *int*, primeiro, ponteiro do tipo *Usuario*, ultimo, ponteiro do tipo *Usuario* e o método Posicional, que é usado para posicionar o item da lista que será modificado. Também possui alguns outros métodos como *S_InsereFinal*, que é usado para inserir um objeto ao final da lista, *removeUsuario*, que é usado para inserir um *Usuario* de acordo com o *ID* passado como parâmetro, *Existe*, que verifica se um *Usuario* já existe na lista de acordo com o *ID* passado como parâmetro. Essa classe possui uma lista de objetos do tipo *Usuario*, que é alocada dinamicamente.

O arquivo ***main.cpp*** é onde ocorre a execução do programa em si, essa parte recebe o arquivo de entrada em formato *.txt*, fornecido pelo usuário, lê todo esse arquivo, cria um *Servidor*, uma para ser usada como global, é ocorre toda a manipulação da lista de *Usuario*. Nesse arquivo, também ocorre a impressão da saída do programa em tela.

## 3. Análise de complexidade

A análise de complexidade do programa foi analisada os métodos relevantes de cada classe e o *main*. Dessa forma, sendo analisada no âmbito do tempo e espaço.

## 3.1 Complexidade temporal

Na classe ***Email***, não há nenhum tipo de comparação, somente atribuições por meio dos métodos *set*, *get* e dos construtores, podendo ser considerados como O(1).

Na classe ***Usuario***, temos os métodos *get*, *set* (exceto exceções que serão trabalhadas a seguir) e construtor que possuem apenas algumas atribuições, logo, complexidade O(1).

*~Usuario* aciona o método *U_Limpa* que possui ordem de complexidade O(n) e um "delete", O(1), sendo assim, suas ordens de complexidade são somadas - O(n);

*getPrimeiroEmail* possui apenas algumas comparações que são O(1), mas aciona internamente o método *getMensagem* que tem ordem de complexidade O(1), sendo assim, suas ordens de complexidade são somadas - O(1);

*U_Limpa* possui algumas atribuições e um loop "while" para realizar comparação e atribuição - O(n);

*U_Vazia* possui apenas uma comparação - O(1);

*U_InsereFinal* possui apenas algumas atribuições, mas aciona internamente os métodos construtor *Email* e *U_Vazia*, que tem ordem de complexidade respectivamente, O(1) e O(1), sendo assim, suas ordens de complexidade são somadas - O(1);

*InsereOrdenado* possui algumas atribuições e um loop "while" para realizar comparação e atribuição, mas aciona internamente os métodos construtor *Email* e *U_Vazia*, que tem ordem de complexidade respectivamente, O(1) e O(1), e alguns métodos *get* e *set*, que tem ordem de complexidade como, O(1), sendo assim, suas ordens de complexidade são somadas - O(n);

*removePrimeiroEmail* possui apenas algumas comparações que são O(1), mas aciona internamente o método *Tamanho* que tem ordem de complexidade O(n), sendo assim, suas ordens de complexidade são somadas - O(n);

*Tamanho* possui algumas atribuições e um loop "while" para realizar comparação e atribuição, mas aciona internamente o método *U_Vazia*, que tem ordem de complexidade, O(1), sendo assim, suas ordens de complexidade são somadas - O(n);

Para saber a complexidade da classe, basta somar a complexidade de cada método, que resulta em O(n).

Na classe ***Servidor***, temos os métodos construtores que possuem algumas atribuições e podem ser considerados como O(1).

*~Servidor* aciona o método *S_Limpa* que possui ordem de complexidade O(n) e um "delete", O(1), sendo assim, suas ordens de complexidade são somadas - O(n);

*setUsuario* aciona internamente os métodos *Posiciona* e *InsereOrdenado*, que tem ordem de complexidade respectivamente, O(n) e O(n), sendo assim, suas ordens de complexidade são somadas - O(n);

*S_getPrimeiroEmail* aciona internamente os métodos *Posicona* e *getPrimeiroEmail*, que tem ordem de complexidade respectivamente, O(n) e O(1), sendo assim, suas ordens de complexidade são somadas - O(n);

*Posiciona* possui algumas atribuições e um loop "while" para realizar comparação e atribuição, mas aciona internamente o método *getID*, que tem ordem de complexidade, O(1), sendo assim, suas ordens de complexidade são somadas - O(n);

*S_Limpa* possui algumas atribuições e um loop "while" para realizar comparação e atribuição - O(n);

*S_Vazia* possui apenas uma comparação - O(1);

*S_InsereFinal* possui algumas atribuições e um loop “while” para realizar comparação e atribuição, mas aciona internamente o método construtor *Email*, que tem ordem de complexidade respectivamente, O(1), sendo assim, suas ordens de complexidade são somadas - O(n);
*S_removePrimeiroEmail* possui apenas algumas atribuições e comparações que são O(1), mas aciona internamente os métodos *getID*, *Posiciona*, *removePrimeiroEmail*, que tem ordem de complexidade respectivamente, O(1), O(n) e O(n), sendo assim, suas ordens de complexidade são somadas - O(n);
*removeUsuario* possui algumas atribuições e um loop “while” para realizar comparação e atribuição, mas aciona internamente o método *getID*, que tem ordem de complexidade respectivamente, O(1), sendo assim, suas ordens de complexidade são somadas - O(n);
*existe* possui algumas atribuições e um loop “while” para realizar comparação e atribuição, mas aciona internamente os métodos *getID*, *U_getProx*, que tem ordem de complexidade respectivamente, O(1) e O(1), sendo assim, suas ordens de complexidade são somadas - O(n);
*usuarioVazio* possui apenas algumas atribuições e comparações que são O(1), mas aciona internamente os métodos *getID*, *Posiciona*, *U_Vazia*, que tem ordem de complexidade respectivamente, O(1), O(n) e O(1), sendo assim, suas ordens de complexidade são somadas - O(n);

Para saber a complexidade da classe, basta somar a complexidade de cada método, que resulta em O(n).

O ***main*** possui um loop while, logo a complexidade é O(n). Dentro do *main* são acionados alguns métodos das classes do sistema, como a maior ordem de complexidade das classes é O(n), dessa forma podemos considerar que o *main* como um todo possui ordem de complexidade $O(n^2)$.

Com isso, tem se as ordens de complexidade de todos os métodos e partes do programa, como a ordem de complexidade total é a soma de todas ordens, pode-se concluir que a ordem de complexidade total do sistema é $O(n^2)$.

## 3.2 Complexidade espacial

A análise de espaço pode se avaliar o principal meio de armazenamento do sistema, que a lista de objetos do tipo *Usuario*, que cada objeto dessa lista possui outra lista de objetos do tipo *Email,* dessa forma, podemos considerar que o espaço ocupado é $\mathbf{O(n^2)}$, pois a lista possui somente duas dimensões. Pode se analisar também que a complexidade não irá se alterar independe do tamanho das listas, visto que, $c*O(f(n)) = O(f(n))$.

## 4. Estratégias de robustez

Como se trata de manipulação de arquivos, e com isso cadastro, remoção, e manipulação de informações, algumas estratégias foram utilizadas. Sempre que o programa identifica alguma não conformidade, é retornada uma mensagem de erro em tela.

Em caso de tentativa de cadastro de um ID já existente, o programa retorna uma mensagem de erro, “ERRO: CONTA ID JA EXISTENTE”.

Em caso de tentativa de remoção de um cadastro não existente ou já excluído, o programa retorna uma mensagem de erro, “ERRO: CONTA ID NAO EXISTE”.

Em caso de tentativa de enviar um email a um cadastro não existente ou já excluído, o programa retorna uma mensagem de erro, “ERRO: CONTA ID NAO EXISTE”.

Em caso de tentativa de consulta de um cadastro não existente ou já excluído, o programa retorna uma mensagem de erro, “ERRO: CONTA ID NAO EXISTE”.

## 5. Conclusões

Esse trabalho teve como objetivo a implantação de um sistema de servidor de emails, utilizando alocação dinâmica, a partir disso foi desenvolvido um programa em C++ para atender a todas as especificações propostas.

Com esse desenvolvimento foi possível assimilar bem novos conceitos, como o de listas encadeadas e sua manipulação.

## Referências

Ziviani, N., **Projeto de Algoritmos com Implementações em Pascal e C**, 3ª Edição, Cengage Learning, 2011.

## Instruções de compilação e execução

Para a execução do programa sigas os passos descritos abaixo, utilizando o Makefile para isso:

- Utilizando o terminal acesse o diretório;
- Execute o arquivo Makefile utilizando o seguinte comando: “make”;
- Após esse comando, na pasta raiz, utilize o seguinte comando “run.out *diretório_do_arquivo_de_entrada*”;
- com esse comando deve aparecer em tela o resultado final;