**Trabalho Prático da disciplina de Estruturas de Dados**

**Luiz Felipe dos Santos Alves**

Departamento de Ciência da Computação -Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte – MG – Brasil

luiz49138@ufmg.br

# **Introdução**

Este trabalho trata do assunto de elaborar um sistema de servidor de e-mails semelhante ao da Google. Para a tratativa do problema é preciso criar um servidor que dê suporte à entrega e consulta de mensagens de e-mail para os usuários cadastro nele. Contando também com gerenciamento de contas de usuários, ou seja, na adição e remoção de usuários. Vale salientar que nesse sistema não pode possuir dois mesmos usuários, ou seja, cada usuário do sistema é único e não pode cadastra-lo novamente, e a caixa de e-mail que cada usuário além de se inicializar vazia, ela possui prioridade nas mensagens que são entregues aos e-mails, se por acaso as mensagens tiverem a mesma prioridade a ordem deve seguir pela mensagem que estiver a mais tempo na caixa, ou seja, a primeira que foi enviada tem mais prioridade que a última que foi enviada.

As divisões desse trabalho está dívida da seguinte forma: a segunda divisão trata da implementação, onde mostra como foi organizado o programa, a estrutura de dados e quais ferramentas utilizadas para compilação, a terceira fala sobre a análise de complexidade tanto de tempo quanto de espaço e a quarta é a conclusão.

# **Implementação**

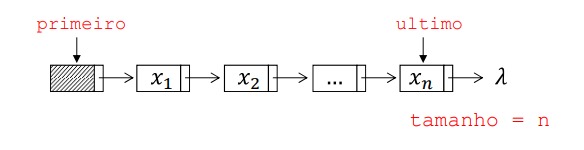
O programa foi desenvolvido na linguagem C++, compilada pelo compilador G++ da GNU Compiler Collection.

## **Estrutura de Dados**

O desenvolvimento do programa teve como base a estrutura de dados de uma lista encadeada. A escolha dessa estrutura de dados para tratar os dois tipos de armazenamento de dados, tanto a caixa de e-mail quanto o usuário do sistema, foi pelo fato que o custo da remoção de elementos no início é constante. Além de ter fácil acesso aos elementos de uma posição desejada, podendo organizar a caixa de e-mails de acordo com a prioridade da mensagem deixando sempre as prioritárias no início da lista.

Essa estrutura de dados foi montada em duas classes, uma chamada Emails e outra chamada Lista. Como visto em aula, implementou-se a lista com a chamada “célula-cabeça” antes do primeiro elemento, visando facilitar e, assim, reduzir o custo assintótico de algumas operações. Além disso, um dos membros da lista guarda a quantidade de elementos que ela possui, evitando um custo linear desnecessário (de iterar pela lista inteira contando o número de elementos) caso não houvesse esse atributo com essa informação.

Um diagrama esquemático da lista encadeada implementada pode ser visto na figura.



### Figure 1. Diagrama da Lista

As principais funções, como os construtores e os destrutores, as operações de inserção e remoção nas várias posições da lista são adaptações dos algoritmos vistos em sala.

## **Classes**

No desenvolvimento, foram montadas quatro classes principais. A primeira chamada Emails é a lista encadeada abordada acima, com as modificações feitas para que seus ponteiros sejam um objeto que guarda o usuário. OS dois ponteiros que ela possui, um aponta para o primeiro usuário, e o outro para o ultimo usuário e seu atributo de tamanho.

Os métodos que a classe Emails possui, funciona para o gerenciamento do sistema, como por exemplo o método de cadastraEmail e consultaEmail que recebe como parâmetro de entrada os ID que deseja cadastrar ou consultar.

A classe de usuários chamada UserEmail possui os seguintes atributos, um ponteiro para sua caixa de mensagem, outro ponteiro para indicar o próximo e-mail da lista, e o seu ID, e ela possui os métodos uteis como o getCaixamsg que retorna um ponteiro para o início de sua caixa de mensagem e o método de consultar a caixa de mensagem que imprimi suas mensagens de acordo com sua prioridade.

As outras duas classes foram criadas para o armazenamento das mensagens e sua prioridade, a primeira chama Lista, e como o nome já diz ela também segue a estrutura de uma lista encadeada, com dois ponteiros para suas mensagens iniciais e finais e o tamanho que guarda a quantidade de mensagem que esse usuário possui. Um dos métodos dessa classe, faz a inserção do objeto mensagem de acordo com a prioridade que ela tem, guardando sempre a com maior prioridade no início e a menor no final. E o objeto mensagem é uma classe que guarda a string da mensagem, a sua prioridade e o ponteiro para a próxima mensagem, e ela se chama TipoCelula.

## **Gerenciamento do sistema**

O gerenciamento dos e-mails se dá por quatro tipos de comandos, são ele o comando de cadastrar, de remover, de enviar um e-mail e de consultar a caixa de mensagem de um e-mail. Para executar essas operações, a primeira entrada deve ser o comando que deseja operar e a segunda entrada é o ID do usuário que deseja realizar a operação, no caso do envio de mensagem a entrada tem mais três parâmetros de entrada que são a prioridade a mensagem que deseja enviar e o indicativo de fim da mensagem.

Detalhando as operações: CADASTRA ID – a primeira informação é a operação que deseja realizar e nesse caso é o cadastro, a segunda é o ID que deseja cadastrar no sistema, caso já exista um usuário deve retorna uma mensagem de erro. REMOVE ID – seguindo o mesmo padrão da operação de cadastrar, ela procura no sistema o ID informado para realizar a remoção, caso não ache ela retorna uma mensagem de erro. ENTREGA ID PRI MSG FIM – nessa operação o ID informado é o usuário que irá receber a mensagem, o parâmetro PRI é a prioridade da mensagem, MSG é a mensagem que deseja enviar para o ID, e o FIM é uma marcação para informar ao sistema que chegou no final da mensagem. CONSULTA ID – também seguindo o padrão de cadastro, a consulta busca no sistema o ID informado e entra na sua caixa de mensagem imprimindo a mensagem com maior prioridade, caso houver duas mensagens com a mesma prioridade, a mensagem que será impressa é a que tem mais tempo dentro da caixa de e-mail.

# **Análise Complexidade**

## **Tempo**

Para a análise de complexidade de tempo na operação de cadastro, a princípio vamos cadastrar um usuário no sistema, sabendo que o sistema não tem usuário logo ele não entra na verificação de duplicidade de usuário tendo então um custo constante para armazenar esse primeiro usuário, ou seja, *O*(1). Se o sistema já possuir alguns usuários esse custo vai para *O*(n) onde n é o número de usuários que o sistema contém. Como o custa para adicionar o usuário é *O*(1) e para verificar a duplicidade é *O*(n), então temos a seguinte analise de tempo para o cadastro:

*O*(*1*)+ *O*(*n*) = *O*(*n*)

Para a remoção de um usuário temos o seguinte fluxo, primeiro fazemos uma busca no sistema para ver a existência desse usuário com o custo de melhor caso *O*(1) quando o usuário for o primeiro no cadastro e custo de *O*(n) no pior caso, quando ele ou está na última posição ou ele não existe no sistema. Caso sua existência é comprovada, precisamos achar sua posição e em sequência pegar o elemento anterior a ele para fazer a remoção, o pior caso para a remoção tem custo de *O*(n²), que é quando o elemento está na última posição, então vou precisar percorre toda a lista para passar a sua posição para a função que retorna à posição anterior a ele, e essa função percorre novamente a lista para retornar o usuário anterior. Com isso, temos para o pior caso:

*O*(*n*)*+O*(*n*)+ *O*(*n²*) = *O*(*n²*)

Para o melhor caso da remoção, é quando o usuário está no início da lista, tendo assim um custo de *O*(*1*) para a operação de remoção, pois tanto na verificação, na procura da sua posição e no retorno do usuário anterior seu custo é *O*(*1*), tendo assim:

*O*(*1*)+ *O*(*1*)+ *O*(*1*)= *O*(*1*)

Como nas outras operações a analise tempo será feita sem o tratamento da linha do arquivo, ou seja, quando colocamos esse tratamento de linha as funções ficam no seguinte padrão:

*O(g(N)) +O(f(n)) = O(max(g(N), f(n)))*

Onde *g(N)* é o custo para tratar a linha do arquivo e f(n) é o custo da operação. Fazendo a análise da operação de enviar mensagem, começando fazendo o envio de uma mensagem, ela tem o seguinte custo: primeiro um custo *O*(*n*) para verificar se o usuário possuí cadastro no sistema, um custo *O(1)* para regatar sua caixa de mensagem, e um custo de pior caso, ou seja o caso em que a prioridade da mensagem é a última/menor, de *O(n)* e para o melhor caso séria quando a caixa de mensagem do usuário não possuí nenhuma mensagem ou então é de maior prioridade quando comparada a todas outras e nesses dois cenário o custo é O(1). Com isso temos que:

*O(n)* + *O(1)* + *O(n)* = *O(n)*

Para a operação de consulta de caixa de mensagem, também é feita a verificação do usuário no sistema de custo *O(n)* depois faz uma busca desse usuário de custo *O(n)* e imprime a primeira mensagem, pois a primeira mensagem é a de maior prioridade tendo assim o custo de *O(1)*, no fim ficamos com:

*O(n)* + *O(n)* + *O(1)* = *O(n)*

## **Espaço**

Considerando que cada usuário e cada mensagem ocupa uma unidade de espaço. Analisando o pior caso que é onde existe n usuários e cada usuário possui uma caixa de mensagem com n mensagens, temos então que a análise de espaço é dada por O(n²).

# **Conclusão**

O problema citado nesse trabalho prático foi a implementação de um novo sistema de e-mails e para isso foi utilizado a criação de uma estrutura de dados para o armazenamento dos conteúdos enviados e dos usuários do sistema. A estrutura utilizada foi a lista encadeada, com uma particularidade na inserção de conteúdo da mensagem, dando a prioridade adequada para as mensagens que cada e-mail/user recebe. Utilizando uma programação funcional para criar essa estrutura, para que na aplicação dela não ocorrer problemas no armazenamento e consulta desses dados.

Com a criação dessa lista de prioridade dentro da caixa de mensagem de cada e-mail, facilitou o custo da consulta de cada mensagem, a única dificuldade é na lógica de inserção onde possui uma regra de prioridade, a regra que utilizei foi a seguinte: a mensagem com maior prioridade fica sempre mais próxima do início da lista, ou seja, na ordem da esquerda para a direita ela fica sempre a esquerda e a de menor prioridade a direito pois será a última a ser consultada, caso exista duas mensagem de mesma prioridade a que fica mais à esquerda é a que foi inserida primeiro, ou seja, a mais antiga dentro da caixa de mensagens. Para o armazenamento de cada usuário utilizei uma lista comum sem nenhuma ordem/regra de inserção ou remoção. Teve também o tratamento das entradas e saídas de dados, de fácil implementação.

Durante esse desenvolvimento, consegui entender melhor como funciona a estrutura de uma lista, podendo manipular seus métodos para atender as particularidades de um sistema. Com a análise de complexidade pude perceber como posso melhorar o sistema para ficar ainda mais rápido e consumir menos custo computacional, dessa forma mostra a importância de fazer essa análise nos algoritmos para aumentar a eficiência e robustez dos programas.

# **Referencias**

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.