Trabalho Prático 1
Servidor de e-mails

Lucas Victor da Silva Costa

Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, Minas Gerais, Brazil

1. Introdução

O trabalho prático consiste em criar um programa que simule um servidor de e-mails, suportando operações básicas de criação de usuários e envios de e-mails. Para isso, tipos abstratos de dados devem ser implementados para manusear todas as informações da simulação. Além disso, será feita uma análise do custo assintótico das funcionalidades do servidor e uma bateria de testes de desempenho e memória. O objetivo é comparar a análise de custo e os testes práticos de desempenho, para inferir se o algoritmo se comportou da forma esperada, e entender como nosso programa manipula a memória durante sua execução.

2. Método

O programa foi desenvolvido inteiramente em C++, os TAD's foram implementados através de Classes, sendo elas:

- **Mail:** representa um e-mail, contém uma mensagem e a prioridade dela;
- **Priority Queue:** é uma fila de e-mails (**Mail**), seu funcionamento é similar a de uma fila comum, porém, e-mails com maior prioridade passam na frente dos de menor prioridade;
- **User:** representa uma conta de um usuário, cada conta possui um ID único e um InBox, que é uma fila de e-mails (**Priority Queue**);
- **Account List:** é uma lista que contém todos os usuários registrados no servidor.

Métodos das Classes:

- **Mail**
  - void setPriority(int p): muda a prioridade do e-mail para o valor de p;
  - void setMessage(string m): muda a mensagem do e-mail para m;
  - int getPiority(): retorna o valor da prioridade do e-mail;
  - string getMessage(): retorna a mensagem do e-mail.
- **Priority Queue**
  - void insert(ItemType item): insere um novo item(e-mail) na fila, de acordo com a prioridade dele;
  - ItemType getItem(): retorna o primeiro item da fila;
  - void clean(): retira todos os elementos da fila;
  - int length(): retorna a quantidade de elementos na fila.
- **User**
  - int getId(): retorna o ID do usuário;
  - Mail getMail(): retorna o primeiro e-mail do InBox do usuário;
  - void setMail(Mail m): insere o e-mail m no InBox do usuário;

    - int checkInBox(): retorna o número de e-mails no InBox do usuário;
- **Account List**
    - int insert(User u): insere o usuário u na lista de usuários. Retorna 1 se a inserção foi feita com sucesso e 0 caso contrário;
    - int remove(int id): remove o usuário com o respectivo id da lista de usuários. Retorna 1 se a remoção foi feita com sucesso e 0 caso contrário.
    - int length(): retorna o número de usuários cadastrados na lista.
    - User *getUser(int id): retorna o endereço de memória do usuário com o respectivo ID .

3. Análise de Complexidade

A seguir estão as análises de complexidade de tempo das funções mais relevantes do programa.

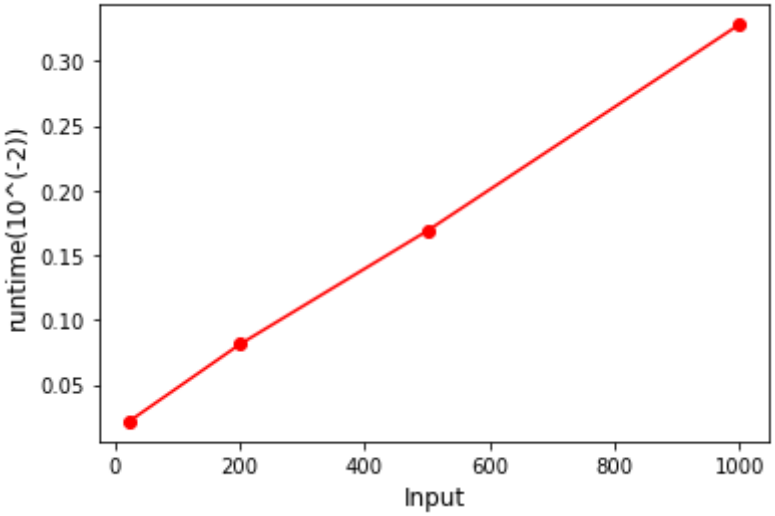
- **void priority_queue::insert(ItemType item):** essa função percorre a fila, começando pelo primeiro elemento, até encontrar a posição que o novo item deve entrar, considerando sua prioridade. No melhor caso, quando o novo item tem prioridade igual ou maior ao elemento que tem a maior prioridade da fila, o novo elemento é colocado na primeira posição, nesse caso a função é *O(1).* O pior caso é quando o novo elemento tem prioridade menor do que todos os elementos da fila, tendo que ser colocado no final dela, sendo assim, nesse caso a função é *O(n).* Considerando que na minoria das vezes a função caia no melhor caso, o caso médio tende mais para o pior caso, podemos considerá-lo *O(n)* também.
- **ItemType priority_queue::getItem():** essa função retorna o primeiro elemento da fila, sendo assim ela é sempre *O(1);*
- **void priority_queue::clean():** essa função percorre a fila deletando cada um dos elementos, sendo assim ela é sempre *O(n);*
- **int account_list::insert(User u):** essa função insere o novo usuário sempre no endereço de memória após o ultimo elemento inserido, sendo assim ela é sempre *O(1);*
- **int account_list::remove(User u):** essa função percorre a lista deletando cada um dos usuários nela, sendo assim ela é sempre *O(n);*
- **User *account_list::getUser(int id):** essa função percorre a lista até encontrar o usuário com o respectivo ID, no melhor caso o usuário está na primeira posição, nesse caso a função é *O(1).* No pior caso o usuário está na última posição, nesse caso a função é *O(n).* Considerando que na minoria das vezes a função caia no melhor caso, o caso médio tende mais para o pior caso, podemos considerá-lo *O(n)* também.

O custo da simulação não só depende da quantidade de instruções, adicionar, remover etc., mas também de quais são as instruções e em qual ordem elas serão feitas.

Já para o custo de memória, percebe-se que ele cresce linearmente com a quantidade de usuários cadastrados e com a quantidade de e-mails na simulação. Ou seja, *O(n).*

4. Análise Experimental

Para os testes de desempenho utilizei entradas com 22, 200, 500 e 1000 instruções diferentes, como adicionar usuários, enviar mensagem, consultar mensagens de um usuário e remover usuários. Com os resultados dos testes o gráfico abaixo foi gerado utilizando Pyplot.



Os resultados agiram de acordo com o previsto na análise de complexidade, podemos concluir, assim, que a simulação do servidor de e-mails possui um custo computacional linear.

5. Conclusão

Nesse trabalho conseguimos simular em pequena escala, e em menor complexidade, um sistema de e-mails. Foram implementadas diversos TAD's que mapeiam as estruturas da simulação e juntos compõem todas as funcionalidades que o sistema de e-mails deve ter. Com isso, tivemos a oportunidade de aprender a lidar com TAD's que possuem relações entre si, e que podem ser formados com a união de outras estruturas abstratas.

6. 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.

7. **Execução**
   Para compilar o programa basta estar na pasta TP01 e executar o seguinte comando no terminal: **$ make all**
8. **Execução**
   Para executar o programa basta executar o arquivo bin/tp com os seguintes parâmetros: **$ bin/tp “nome_do_input.txt”**