Trabalho Prático 2 – Sistema de Escalonamento Hospitalar

RAFAEL NEUBANER DE ALMEIDA – 2023001638

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

Belo Horizonte – MG – Brazil

neubanerrafael@gmail.com

1-Introdução

Este codigo foi implementado de forma a fazer a leitura de um arquivo de pacientes e então fazer uma simulação de eventos discretos baseada no funcionamento de uma fila de atendimento hospitalar.
Os pacientes são admitidos e tem 6 diferentes tipos procedimentos para serem realizados desde triagem e atendimento ate exames hospitalares que irão admitir pacientes baseados na prioridade que vai de 2-0 onde 2 é a maior e 0 a menor.
Ao final o programa retorna um relatório com o id, data e hora de admissão, data e hora de saída e tempo total gasto dentro do hospital, tempo de atendimento e tempo de espera de cada paciente tratado.

2-Método

O programa foi desenvolvido na linguagem C++, compilado pelo compilador GCC da GNU Compiler Collection. O Computador utilizado tem as seguintes especificaç̃oes:

• Sistema Operacional: Linux Ubuntu 22.04

• Processador: 12th Gen Intel(R) Core(TM) i7-12650h @ 2.30GHz -

• RAM: 32,00 GB (utilizavel: 31,80 GB)

2.1 Organização do código

O código segue os princípios da programação orientada a objetos, utilizando varias classes para implementação das estruturas de dados. Essas classes são utilizadas pelo arquivo main do programa em cima de um arquivo csv lido que pega os dados dos procedimentos e atendimentos e os salva dentro dos seus

respectivos tads e dos pacientes que são armazenados em um minheap ordenado com base no ano, mês, dia, hora e id de cada paciente respectivamente.

2.2 Funcionamento

Após ler e armazenar todos os dados do arquivo o programa chama a função escalona para que todos os pacientes sejam triados e após isso entra em um loop que e executado 6 vezes chamando a função reescalona em cada uma delas para realizar os exames faltantes de cada paciente e joga-los para a fila final caso tenham realizados todos os seus exames. Ao fim, a função imprimir e chamada para cada paciente para expor seus dados de tempos de entrada saída atendimento e espera no hospital.

3-instruções de compilação

O programa pode ser executado utilizando o comando make run após passar como parâmetro para o comando “all” o nome do arquivo xcsv que sera lido e deverá estar localizado na pasta raiz juntamente com o makefile. Dessa forma ao rodar o comando make run o programa já irá printar na tela todos os cadastros ordenados.

4-Análise de complexidade

4.1 Funções do main:

1- Inicializar os pacientes: o arquivo é lido por um ponteiro e armazenado diretamente em minheap, tendo complexidade de tempo e espaço e iguais a O(nlogn), onde n é o tamanho do arquivo, visto que ira adicionar um paciente e chamar a função heapify (O(logn)n vezes e O(1) visto que a cada execução ele aloca um espaço na memoria o utiliza e depois o apaga novamente.
2- Laço escalonador: um laço for é executado n vezes, onde n é o numero de pacientes, chamando a função escalona para colocar o paciente na fila. Tem complexidade de tempo e espaço igual a O(n) já que a função escalona tem ambas as complexidades igual a O(1).
3- Laço reescalonador: um for que vai de 1 a 6 e para cada iteração entra em outro for que chama a função reescalona n vezes (uma para cada paciente que esta nas filas daquele procedimento). Tem complexidade de tempo e espaço iguais a O(6n), ou seja O(n), e O(1) ,visto que a função reescalona tem ambas as complexidades iguais a O(1)
4- Imprimir: um laço for que é executado n vezes, onde n é o numero de pacientes cadastrados, e chama a função imprime para cada paciente.

Possui ordem de complexidade de tempo igual a O(nlogn) e de espaço igual a O(1)visto que para imprimir eu primeiro removo a raiz do minheap e chamo a função heapify para reordena-lo.

4.2 Funções do Paciente

1- Construtor: inicializa todos os atributos do paciente com os dados lidos do arquivo. Ordem de tempo e espaço iguais a O(1).
2- Convertehoras: função responsável por formatar as horas no formato de um relógio. Possui ambas as complexidades iguais a O(1).
3- Diadasemana: função responsável por encontrar qual era o dia da semana referente à data informada. Possui ambas as complexidades iguais a O(1).
4- Convertedia: função responsável por analisar a quantidade de dias que o paciente ficou no hospital e alterar o mês e ano que ele irá sair caso seja necessário. Possui ambas as ordens de complexidade iguais a O(1).
5- Convertemes: recebe o mês em inteiro e retorna o nome dele em string. Possui ordem de complexidade igual a O(1) para espaço e tempo.
6- Contabiliza: função responsável por fazer a contagem das horas do paciente após ele passar por cada procedimento. Ordens de complexidade iguais a O(1).
7- Contahora: função que analisa qual atendimento o paciente esta realizando para poder chamar a função contabiliza. Ordens de complexidade iguais às da função contabiliza, O(1).
8- Printpaciente: função responsável por imprimir os dados de cada paciente, chama as demais funções acima para formatar os dados da maneira que precisam ser escritos. Ordens de grandeza associadas iguais a O(1).

4.3 Funções do Procedimento

1- Construtor: inicializa todos os atributos do tad de acordo com os dados obtidos do arquivo. Complexidade de tempo igual a O(1) e de espaço igual a O(n) com N sendo o numero total de salas que precisarão ser criadas para os atendimentos.

4.4 Funções Fila

1- Troca: função que troca o conteúdo de dois elementos, ordem de complexidade de tempo e espaço iguais a O(1).
2- Subir: função heapify para subir um elemento inserido ate a sua posição correta. Complexidades de tempo e espaço iguais a O(logn) e O(1).
3- Descer: função heapify após deletar um elemento e reordenar o heap descendo a nova raiz para sua posição correta. Complexidades de tempo e espaço iguais a O(logn) e O(1).
4- Comparar: função responsável por retornar se um elemnto é maior ou menor. Ordens de complexidade e tempo iguais a O(1).
5- Inserir: função responsável por inserir elementos no heap, após inserir chama a função subir. Ordem de tempo e espaço iguais a Ologn) (função subir) e O (1).
6- Remover: função responsável por remover um paciente do heap e chamar a função descer para reorganizar o mesmo. Ordens de complexidade iguais a O(logn) e O(1).

4.5 Funções escalonador

1- Escalona: função responsável por checar a prioridade e chamar a função insere fila para a respectiva prioridade do paciente. Ordens dee complexidades iguais a O(1).
2- Reescalona: função responsável por verificar qual fila permanece com pacientes e executa os atendimentos, contabiliza as horas e chama a função escalona para realocar o paciente em sua próxima fila. Possui complexidades de tempo e espaço iguais a O(n) e O(1) para N sendo o numero total de salas de cada atendimento individual.
3- Inserefila: analisa o estado do paciente e o insere na fila que ele devera entrar. Possui ordem de complexidade igual a O(1) para tempo e espaço.

5-Estrategias de robustez

Foi feita uma condição para testar se o arquivo pode ser aberto para evitar que fossem criadas variáveis sem os parâmetros passados pelo arquivo e ocasionar consequentemente em possíveis erros.

O código foi comentado e identado para melhor leitura e entendimento.

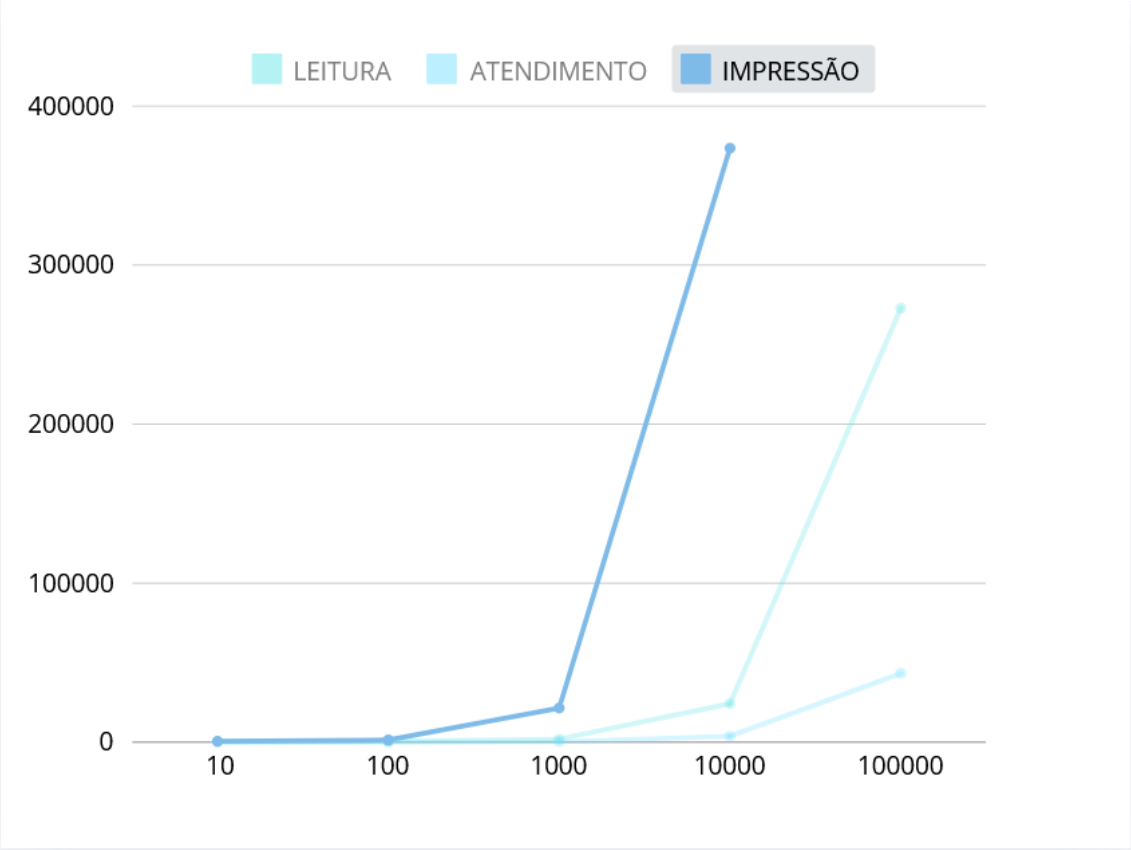
Tambem foi feito uso da modularização do código para melhor organização das implementações.

6-Análise experimental

Utilizei a biblioteca chronus para medir os tempos de execução das funções do código e montar o gráfico a seguir:

Analisando o gráfico podemos ver como o tempo de execução de cada função do programa se comporta ao executarmos um arquivo com 140 mil pacientes, a leitura do arquivo acontece juntamente do preenchimento da primeira fila, executando dessa forma operações O (nlogn) para ordenar os pacientes por hora e ID (heapify). Logo após o procedimento de triagem é executado e atende por ordem, todos os pacientes, e então entra num laço de reescalonamento que irá novamente utilizar operações O(nlogn) para remover e reinserir os pacientes nas filas seguintes que foram manipuladas em formato de um minheap. Por fim entra na impressão dos pacientes que também utiliza operações O(nlogn) (remoção do minheap) e faz as manipulações dos dados dos pacientes para imprimi-los da forma que foi pedida no VPL. Podemos ver que a maior parte do tempo é gasto pela impressão dado que o cout é um comando bastante custoso.

Utilizei também os dados de 5 cargas diferentes de arquivos e calculei os tempos de cada parte do código individualmente para montar o gráfico a seguir onde no eixo x esta representando o tamanho da carga de teste e o eixo y o tempo em microssegundos gasto.
Obs: retirei o tempo de impressão do ultimo teste para não prejudicar a visibilidade do gráfico pelo ponto se localizar em um local muito acima das demais funções.



Sumario dos tempos para melhor ilustrar os números utilizados no gráfico:

N de Pacientes	LEITURA	ATENDIMENTO	IMPRESSÃO
10	274	14	687
100	602	56	1421
1000	1981	333	21540
10000	24286	3821	373544
100000	272902	43295	3271916

Utilizei também as ferramentas do valgrind para testar os acessos a memória, cache e chamadas das funções. Após rodar o memcheck não foi identificada nenhum vazamento de memória nem acesso indevido a posições de memória durante a execução do programa.

7-Conclusao

Este trabalho lidou com um problema de uma simulação de eventos discretos utilizando a situação de uma fila hospitalar na qual vários exames seriam

realizados porem seria preciso considerar critérios de prioridade no atendimento dos pacientes com prioridades entre 0 e 2 onde quanto maior o numero maior também a prioridade.

A resolução desse trabalho foi importante para se entender a importância de se saber manipular filas e os diversos usos que se pode obter da implementação de heaps e arvores durante a programação, que foi o principal desafio do programa: como relacionar as filas com os heaps. Outro ponto importante foi de poder praticar ainda mais a divisão do programa em diversos TADs diferentes para se organizar melhor o programa e facilitar as adaptações e atualizações para cobrir possíveis casos de erros.

8-Referências bibliográficas

Chaimowicks, L. and Prates, R. Slides virtuais da disciplina de estrutura de dados.
Disponibilizados via Moodle. Departamento de Ciencia da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.

Lacerda, A. and Meira, W. Slides virtuais da disciplina de estrutura de dados. Disponibilizados via Moodle. Departamento de Ciencia da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.