# Trabalho Prático 2

Estudante: Sarah Menks Sperber
Matrícula: 2023001824
Email: sarahmenksperber@ufmg.com

## Introdução:

Este trabalho consiste em desenvolver e implementar um sistema de escalonamento hospitalar para o Hospital Zambs (HZ). O objetivo principal é modelar o fluxo de atendimento do hospital, que abrange triagem, atendimento médico e execução de procedimentos, utilizando a técnica de simulação de eventos discretos. Essa abordagem permite monitorar o tempo de permanência dos pacientes no hospital, bem como avaliar o uso de recursos em diferentes cenários.
Assim, no decorrer deste relatório, vamos fazer uma análise profunda dos algoritmos usados, discutindo pontos como metodologias, escolhas na implementação e complexidade assintótica.

## Métodos:

Para a construção do sistema hospitalar, foram usados quatro TADs: Patiant.hpp, Procedure.hpp, Queue.hpp e Scheduler.hpp, onde se encontram as declarações das funções do código. Nesta seção, vamos discorrer um pouco sobre o papel de cada TAD e os respetivos algoritmos usados.

### 1. Patiant.hpp

O TAD Patiant.hpp é encarregado por criar a struct Patiant, que armazena as informações do paciente, que são lidas no arquivo. Esse TAD não possui muitas funções, apenas métodos para impressão das informações do paciente e para a consulta dos dados (funções get).

### 2. Procedure.hpp

Esse é o TAD responsável por armazenar as informações dos procedimentos, como tempo de execução e quantidade de unidades, e por implementar as funções que realizam o processamento dos procedimentos.

### 3. Queue.hpp

Este TAD define a estrutura de dados Lista, projetada para armazenar e manipular os pacientes de forma sequencial. Sua implementação de deu atraves de uma lista encadeada, e aqui se encontram os métodos para a inserção e remoção dos elementos.

### 4. Scheduler.hpp

O arquivo define a interface para o Escalonador, responsável por gerenciar a fila de prioridade utilizada na simulação do sistema hospitalar. Ele organiza os eventos pendentes de execução através de um min heap. Este arquivo inclui a definição das estruturas, métodos e atributos necessários para manipular eventos, como a inserção de novos eventos, remoção do próximo evento e consulta ao próximo evento na fila de prioridade

## Análise de Complexidade

Para facilitar a análise, vamos entender a complexidade assintótica das funções de acordo com o TAD que elas se encontram.

**Patiant.hpp:** O TAD Patiants possui apenas 5 funções - ConfigDate, PrintStatistics, GetPatiantTime e GetProcedureTime - que retornam ou formatam as informações de um paciente. Como seu tempo de execução não muda de acordo com o tamanho da entrada, essas funções são O(1).

**Procedure.hpp:**

- Inicialize(): Atualiza os tempos de duração e de unidades - O(1) - e atualiza o estado de todas as unidades. Logo, a função é O(k), onde k é o numero de unidades.
- GetTime(): Retorna o tempo de duração do procedimento, O(1).
- FindEmptyUnit(): Percorre o vetor de unidades e vê qual unidade já está disponivel para o atendimento. Complexidade é O(k), onde k é o número de unidades.
- PerformProcedure(): A função apenas muda os valores do própio procedimento, mas como ele utiliza a função FindEmptyUnit(), sua complexidade é O(k)

**Patiant.hpp:**

- ConfigDate(): Apenas muda a formatação da data. Por isso, sua complexidade é O(n)
- PrintStatistics(): Imprime as estatísticas de um paciente, logo, é O(1)
- GetPatiantTime(): Apenas retorna o tempo total do paciente. O(1)
- GetProcedureTime(): Retorna o tempo do procedimento de acordo com o estado do paciente. O(1)

**Queue.hpp:**

- Enqueue(): Insere um novo elemento no final da fila. O(1)
- Remove(): Remove o primeiro elemento da lista. Logo, é O(1)
- isEmpty(): Verifica se o tamanho da lista é igual a 0, portanto, a função é O(1) Patiant* First(): Retorna o primeiro elemento da lista (no caso, o elemento apontado pelo head). Sua complexidade é O(1)

**Scheduler.hpp:**

- CreateEvent(): Cria um novo evento e, depois, o insere no escalonador. Para manter as propriedades do min heap, o método chama a função LowHeapfy, que é O(log n)
- InsertEvent(): Insere um evento no escalonador. Como ele também chama a função LowHeapfy, sua complexidade assintótica é O(log n)
- RemoveNext(): Remove o nó da árvore e chama a função HighHeapfy para manter as propriedades do min heap. Assim, sua complexidade é O(log n)
- GetParent(), GetLeftSucessor() e GetRightSucessor(): Como o min heap foi implementado com um vetor, essas funções são O(1)
- isEmpty(): erifica se o tamanho da lista é igual a 0, portanto, a função é O(1)
- GetNextTime(): Retorna a data do evento que está no nó da árvore. Assim, a função é O(1)

## Estratégias de Robustez:

Para garantir um código seguro e sem falhas, algumas estratégias de segurança foram implementadas, tais como:

**1. Uso de try catch:** Para tratar as excessões que podem ser lançadas, foram implementados blocos com instruções try, throw e catch.

**2. Verificação de arquivo válido:** A função main possui um teste de checagem do arquivo. Se o usuário não passar a entrada desejada por parâmetro (um arquivo válido), o código é encerrado e uma mensagem de texto é exibida.

**3. Validações de entrada:** Além das funções mencionadas acima, o código também valida o arquivo de entrada e confere os parâmetros passados entre as funções.

## Análise Experimental:

A análise do uso temporal do código visa avaliar o desempenho do programa em relação ao tempo de execução das suas operações. Para isso, foram realizados testes experimentais,

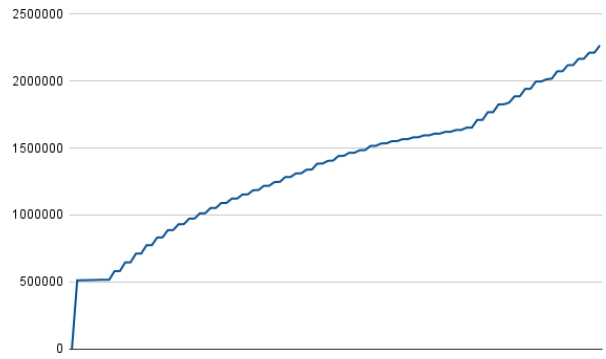


Figura 1: Tempo gasto para processar 100 pacientes

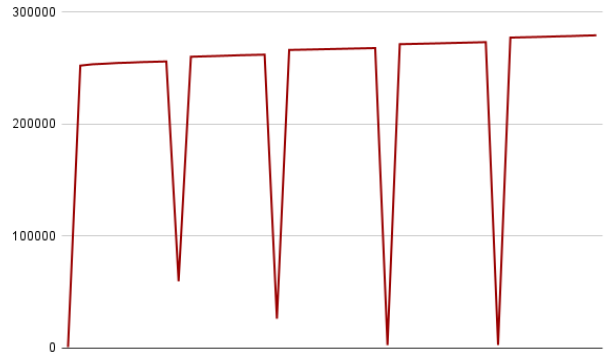


Figura 2: Tempo geral para processar 100 pacientes

medindo o tempo gasto por funções e processos principais, considerando diferentes tamanhos de entrada.

As Figuras 1 e 2 mostram o desempenho temporal do código, em nanossegundos.

## Conclusão

Por meio deste trabalho, foram consolidados conhecimentos teóricos sobre simulações de eventos discretos e sobre o impacto da implementação de soluções computacionais no gerenciamento de sistemas. O aprendizado obtido reforça a habilidade de lidar com problemas reais, criando sistemas robustos e eficientes. Além disso, foi possível praticar os conceitos relacionados a árvores binárias (min heap) e filas de prioridade, assuntos discutidos em aula.

Durante a implementação da solução para o problema, houveram alguns desafios que

dificultaram a elaboração do trabalho, como o cálculo preciso dos tempos em fila.

Toda a análise revelou a importância de decisões eficientes no escalonamento e alocação de recursos para minimizar tempos de espera e melhorar a utilização de recursos. Por isso, mesmo com os erros cometidos, a tarefa executada foi de grande valor para o meu aprendizado enquanto aluna.

## Bibliografia:

LACERDA, Anísio; MEIRA JR., Wagner. Slides virtuais da disciplina de estruturas de dados. Belo Horizonte: Universidade Federal de Minas Gerais, 2024.

VALGRIND. Valgrind User Manual. Disponível em: https://valgrind.org/docs/manual/ms-manual.html. Acesso em: 2 dez. 2024