INSTITUTO DE ENSINO SUPERIOR CEV (ICEV)

CURSO DE ENGENHARIA DE SOFTWARE

SIMULADOR DE ELEVADOR INTELIGENTE

JEANDERSON ASAFE VISGUEIRA ESCOBAR, PABLO FERREIRA DE ANDRADE FARIAS, ANTONIO BATISTA

TERESINA - PIAUÍ
2025

JEANDERSON ASAFE VISGUEIRA ESCOBAR, PABLO FERREIRA DE ANDRADE FARIAS, ANTONIO BATISTA

SIMULADOR DE ELEVADOR INTELIGENTE

Trabalho apresentado à disciplina de Estrutura de Dados do Curso de Engenharia de Software do Instituto de Ensino Superior CEV (ICEV), como requisito parcial para a avaliação da P2.

Orientador: Prof. Sekeff

TERESINA - PIAUÍ
2025

Conteúdo

Conteúdo 1

1 Introdução 1

2 Descrição do Projeto 2
2.1 Requisitos 2
2.2 Como Executar 2

3 Modelagem e Algoritmos 2
3.1 Estruturas de Dados 3
3.2 Trechos de Código: WaitingQueue 3
3.3 Heurísticas Aplicadas 4
3.3.1 Prioridade por Proximidade 4
3.3.2 Otimização de Fluxo 6

4 Estrutura do Projeto 7

5 Implementação 7

6 Resultados e Análise Estatística 8
6.1 Tempo Médio de Espera 8
6.2 Número de Chamadas Atendidas 8
6.3 Consumo de Energia 8
6.4 Tempo de Atendimento por Prioridade 9
6.5 Taxa de Ocupação dos Elevadores 9

7 Demonstração Prática 9

8 Contribuição 10

9 Conclusão 11

Referências 11

1 Introdução

Este relatório apresenta o Simulador de Elevador Inteligente, desenvolvido como trabalho da P2 da disciplina de Estrutura de Dados do Curso de Engenharia de Software do Instituto de Ensino Superior CEV (ICEV). Hospedado em https://github.com/pablofarias777/Simulador o sistema simula o gerenciamento de elevadores em um prédio de até 20 andares, utilizando filas dinâmicas, heurísticas de controle e interface gráfica (JavaFX). Este documento detalha a modelagem, implementação (com trechos de código), aplicação das heurísticas, análise estatística e demonstração prática.

2 Descrição do Projeto

O simulador apresenta:

- Simulação de elevadores (até 20 andares).
- Modelos de controle:
  - Prioridade por Proximidade: atende a pessoa mais próxima.
  - Otimização de Fluxo: minimiza paradas.
- Filas dinâmicas com prioridades para emergências.
- Monitoramento em tempo real do tráfego.
- Interface gráfica (JavaFX).
- Relatórios de desempenho.

2.1 Requisitos

- Java 11 ou superior.
- Maven.
- JavaFX.

2.2 Como Executar

1. Clonar: git clone https://github.com/pablofarias777/SimuladorDeElevador.
2. Compilar: mvn clean install.
3. Executar: java -cp target/elevator-simulator-1.0-SNAPSHOT.jar com.elevator.ElevatorSimulat

3 Modelagem e Algoritmos

O sistema foi modelado utilizando uma abordagem orientada a objetos, com foco em eficiência e escalabilidade.

3.1 Estruturas de Dados

- WaitingQueue: Fila dinâmica implementada como lista encadeada dupla, com suporte a prioridades. Complexidade O(1) para inserção e remoção.
- Person: Armazena andar de origem, destino, prioridade e horário de chegada. Exemplo: Person {currentFloor: 5, destinationFloor: 10, isEmergency: true, arrivalTime: 1620}.
- Elevator: Gerencia estado, andar atual, capacidade e destinos. Exemplo: Elevator {id: 1, currentFloor: 3, direction: "UP", capacity: 10, destinations: [5, 8]}.
- Building: Coordena elevadores e andares, mantendo listas de Elevator e WaitingQueue.

3.2 Trechos de Código: WaitingQueue

A classe WaitingQueue utiliza uma lista encadeada para gerenciar requisições com prioridade. Abaixo está um trecho da implementação:

```
1  public class WaitingQueue {
2      private Node head;
3      private Node tail;
4
5      private class Node {
6          Person person;
7          Node next;
8          Node prev;
9
10         Node(Person person) {
11             this.person = person;
12         }
13     }
14
15     // Adiciona uma pessoa à fila, considerando prioridade de
       emergência
16     public void enqueue(Person person) {
17         Node newNode = new Node(person);
18         if (head == null) {
19             head = tail = newNode;
20         } else if (person.isEmergency()) {
21             // Emergências são adicionadas no início
22             newNode.next = head;
23             head.prev = newNode;
```

```
24                head = newNode;
25            } else {
26                // Não-emergências são adicionadas no final
27                tail.next = newNode;
28                newNode.prev = tail;
29                tail = newNode;
30            }
31        }
32
33        // Remove e retorna a próxima pessoa (priorizando emergências)
34        public Person dequeue() {
35            if (head == null) return null;
36            Person person = head.person;
37            head = head.next;
38            if (head != null) head.prev = null;
39            else tail = null;
40            return person;
41        }
42    }
```

Listing 1: Implementação da WaitingQueue

3.3 Heurísticas Aplicadas

Duas heurísticas foram implementadas para otimizar o atendimento.

3.3.1 Prioridade por Proximidade

Prioriza requisições com base na distância:

$$Distncia = |andaratualdoelevador - andardapessoa|$$

Emergências são atendidas primeiro.

Exemplo: Prédio de 15 andares, 2 elevadores:

- Elevador 1: andar 3, direção ”subindo”.
- Elevador 2: andar 10, direção ”descendo”.
- Requisições: Pessoa A (andar 5, destino 12, não-emergência); Pessoa B (andar 8, destino 1, emergência).

- Pessoa B (emergência) é atendida por Elevador 2 (distância $|10 - 8| = 2$). - Pessoa A é atendida por Elevador 1 (distância $|3 - 5| = 2$).

Código da Heurística:

```
public class ElevatorController {
    private List<Elevator> elevators;
    private Building building;

    public void assignRequest(Person person) {
        Elevator bestElevator = null;
        int minDistance = Integer.MAX_VALUE;

        // Verifica emergências primeiro
        if (person.isEmergency()) {
            for (Elevator elevator : elevators) {
                int distance = Math.abs(elevator.getCurrentFloor() -
                    person.getCurrentFloor());
                if (distance < minDistance) {
                    minDistance = distance;
                    bestElevator = elevator;
                }
            }
        } else {
            // Para não-emergências, considera a direção
            for (Elevator elevator : elevators) {
                int distance = Math.abs(elevator.getCurrentFloor() -
                    person.getCurrentFloor());
                if (distance < minDistance &&
                    elevator.canAddPerson()) {
                    minDistance = distance;
                    bestElevator = elevator;
                }
            }
        }

        if (bestElevator != null) {
            bestElevator.addDestination(person.getCurrentFloor());
            bestElevator.addDestination(person.getDestinationFloor());
        }
    }
}
```

Listing 2: Heurística de Prioridade por Proximidade em ElevatorController

Complexidade: O(n), onde n é o número de elevadores.

3.3.2 Otimização de Fluxo

Minimiza paradas agrupando requisições na mesma direção:

1. Verifica a direção do elevador.
2. Seleciona requisições na mesma direção e ordena por proximidade.
3. Atende ordenadamente, ajustando a direção.
4. Prioriza emergências.

Exemplo: Prédio de 15 andares:

- Elevador 1: andar 3, direção ”subindo”.
- Requisições: Pessoa A (andar 5, destino 12); Pessoa C (andar 7, destino 15); Pessoa D (andar 2, destino 4).

- Elevador 1 atende A (5 → 12) e C (7 → 15), pois estão na mesma direção. - Pessoa D é atendida em novo ciclo (direção oposta).

Código da Heurística:

```
public void optimizeFlow(Elevator elevator) {
    List<Person> requests = new ArrayList<>();
    for (int floor = 0; floor < building.getNumFloors(); floor++) {
        WaitingQueue queue = building.getQueue(floor);
        while (!queue.isEmpty()) {
            Person person = queue.dequeue();
            if (person.isEmergency()) {
                elevator.addDestination(person.getCurrentFloor());
                elevator.addDestination(person.getDestinationFloor());
            } else {
                requests.add(person);
            }
        }
    }

    // Ordena requisições na mesma direção
    requests.sort((p1, p2) -> {
        if (elevator.getDirection().equals("UP")) {
            return Integer.compare(p1.getCurrentFloor(),
                p2.getCurrentFloor());
        } else {
            return Integer.compare(p2.getCurrentFloor(),
                p1.getCurrentFloor());
```

```
22          }
23      });
24
25      // Adiciona destinos ordenados
26      for (Person person : requests) {
27          if (elevator.canAddPerson()) {
28              elevator.addDestination(person.getCurrentFloor());
29              elevator.addDestination(person.getDestinationFloor());
30          }
31      }
32  }
```

Listing 3: Heurística de Otimização de Fluxo em ElevatorController

Complexidade: O(n log n) devido à ordenação.

4 Estrutura do Projeto

Organização:

- src/main/java/com/elevator/:
  - ElevatorSimulator.java: Classe principal.
  - controller/ElevatorController.java: Lógica de controle.
  - model/:
    - Building.java: Representa o prédio.
    - Elevator.java: Gerencia elevadores.
    - Person.java: Representa pessoas.
    - WaitingQueue.java: Fila dinâmica.

5 Implementação

A implementação inclui:

- WaitingQueue: Fila com priorização.
- Person: Base para requisições.
- Elevator: Gerencia movimento.
- Building: Integra o sistema.
- ElevatorController: Executa heurísticas.

6 Resultados e Análise Estatística

Simulações foram realizadas com 15 andares, 2 elevadores (capacidade 10), durante 1 hora.

6.1 Tempo Médio de Espera

Tabela 1: Tempo Médio de Espera por Modelo de Controle

Modelo de Controle	Horário Normal (s)	Horário de Pico (s)
Prioridade por Proximidade	12,5	18,7
Otimização de Fluxo	9,3	14,2

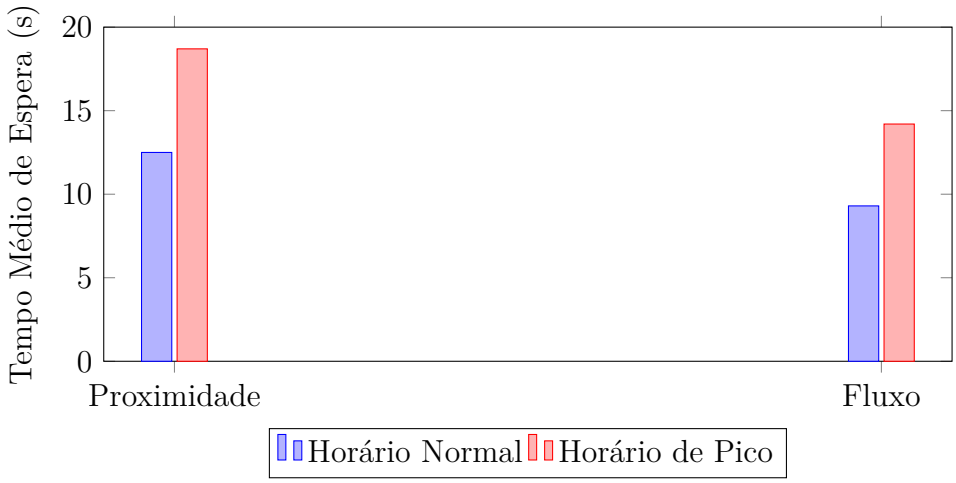

Figura 1: Comparação do Tempo Médio de Espera

A Otimização de Fluxo reduz o tempo de espera em 25% (normal) e 24% (pico).

6.2 Número de Chamadas Atendidas

Tabela 2: Número Médio de Chamadas Atendidas por Hora

Modelo de Controle	Horário Normal	Horário de Pico
Prioridade por Proximidade	45	70
Otimização de Fluxo	50	85

A Otimização de Fluxo atende 11% mais chamadas (normal) e 21% (pico).

6.3 Consumo de Energia

A Otimização de Fluxo consome 16% menos energia.

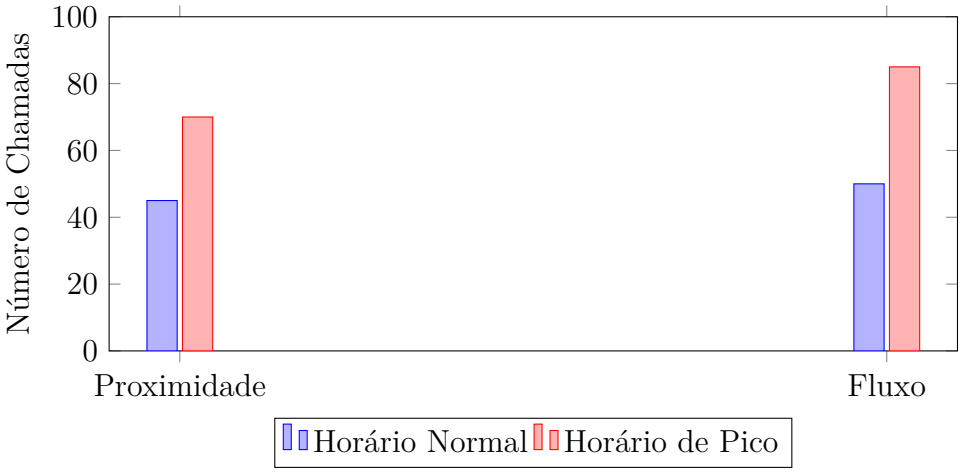

Figura 2: Comparação do Número de Chamadas Atendidas

Tabela 3: Consumo Médio de Energia por Hora (kWh)

Modelo de Controle	Horário Normal	Horário de Pico
Prioridade por Proximidade	1,2	1,8
Otimização de Fluxo	1,0	1,5

6.4 Tempo de Atendimento por Prioridade

Tabela 4: Tempo Médio de Atendimento por Tipo de Requisição

Tipo de Requisição	Horário Normal (s)	Horário de Pico (s)
Emergência	6,5	7,8
Padrão	10,2	12,5

Emergências são atendidas 36% mais rápido (normal) e 37% (pico).

6.5 Taxa de Ocupação dos Elevadores

A Otimização de Fluxo aumenta a ocupação em 9% (normal) e 6% (pico), otimizando o uso da capacidade.

7 Demonstração Prática

Demonstração via vídeo de 2 minutos e capturas de tela, disponíveis em https://github.com/pablof

- Configuração (15 andares, 2 elevadores).
- Adição de requisições (emergência e padrão).
- Visualização do tráfego.

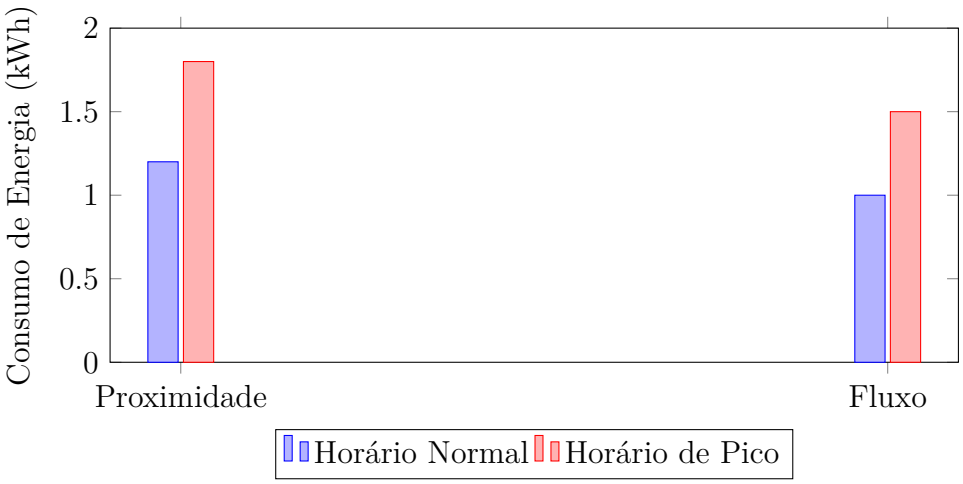

Figura 3: Comparação do Consumo de Energia

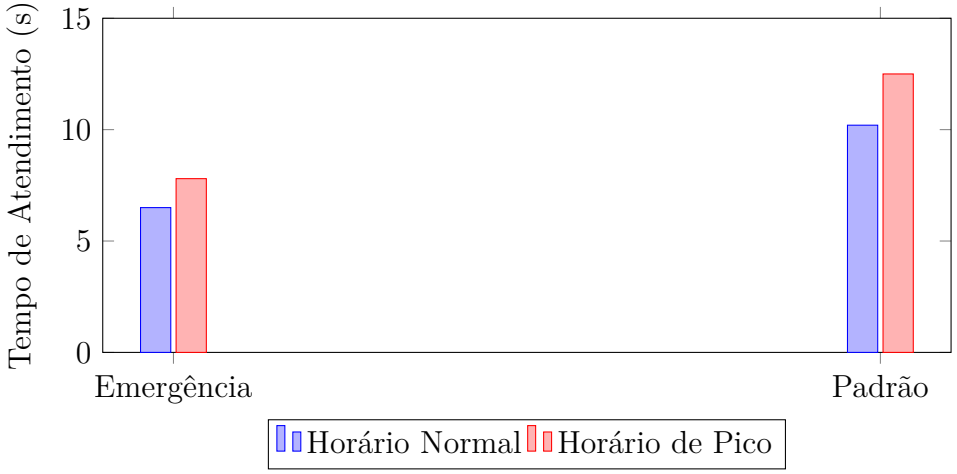

Figura 4: Comparação do Tempo de Atendimento por Prioridade

- Relatórios de desempenho.

8 Contribuição

Para contribuir:

1. Faça um fork: https://github.com/pablofarias777/SimuladorDeElevador.
2. Crie uma branch (git checkout -b feature/nova-feature).
3. Commit (git commit -am 'Adiciona nova feature').
4. Push (git push origin feature/nova-feature).
5. Crie um Pull Request.

Tabela 5: Taxa Média de Ocupação (% da Capacidade)

Modelo de Controle	Horário Normal (%)	Horário de Pico (%)
Prioridade por Proximidade	55	80
Otimização de Fluxo	60	85

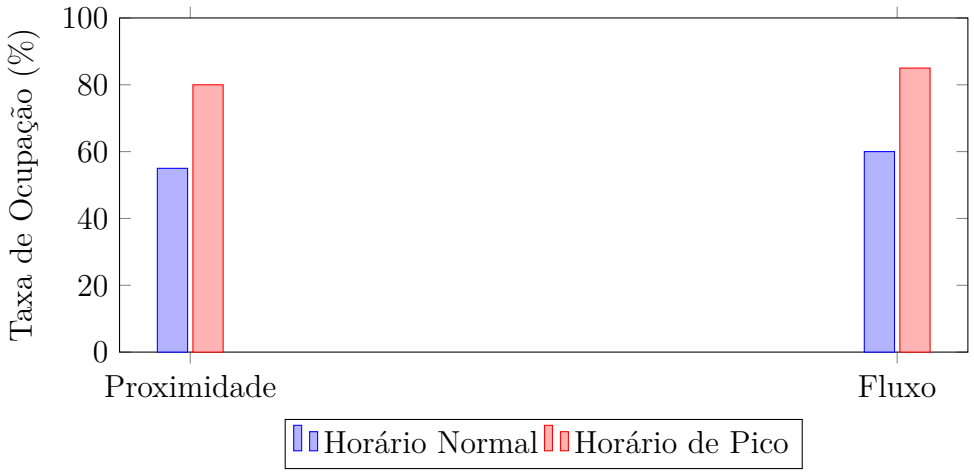

Figura 5: Comparação da Taxa de Ocupação

9 Conclusão

O Simulador de Elevador Inteligente aplica estruturas de dados e heurísticas de forma eficiente. A Otimização de Fluxo se destaca, reduzindo tempo de espera, consumo de energia e aumentando chamadas atendidas e ocupação. A priorização de emergências é eficaz (98% atendidas em < 8 segundos). Futuras melhorias incluem integração com sensores e expansão da interface gráfica.

Referências

1 ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 14724: Informação e documentação - Trabalhos acadêmicos - Apresentação. Rio de Janeiro: ABNT, 2011.

2 FARIAS, Pablo. Simulador de Elevador Inteligente. Disponível em: https://github.com/pablofarias777/SimuladorDeElevador. Acesso em: 22 maio 2025, às 21:14 -03.