Trabalho Prático 2: Simulação de Eventos Discretos

Vinícius
Engenharia de Sistemas

DCC – UFMG / Belo Horizonte, Minas Gerais

21 de junho de 2025

1 Introdução

Esta documentação lida com o problema da simulação de um sistema logístico para entrega de pacotes entre armazéns conectados por rotas. Cada pacote possui uma origem, um destino e uma rota previamente definida que deve ser percorrida até a entrega final.

O objetivo do trabalho é implementar um simulador capaz de armazenar pacotes, gerenciar eventos de transporte/chegada e calcular o tempo total gasto em operações logísticas. A simulação respeita algumas restrições como capacidade de transporte, tempo de viagem e intervalo entre saídas de veículos.

Para resolver o problema citado, foi seguida uma abordagem baseada em um grafo para representar os armazéns e suas conexões, além de estruturas auxiliares como listas encadeadas, filas de prioridade (heap) e pilhas para manipulação dos pacotes

A documentação está organizada da seguinte forma:

- Seção 1: Introdução;
- Seção 2: Descrição da implementação;
- Seção 3: Instruções de compilação e execução;
- Seção 4: Análise de complexidade;
- Seção 5: Estratégias de Robustez;
- Seção 6: Análise Experimental;
- Seção 7: Conclusão;
- Referências Bibliograficas;

2 Implementação

O código foi modularizado em diferentes arquivos, separando as principais classes do sistema em arquivos de implementação ‘src‘ e cabeçalhos ‘include‘. O sistema utiliza as seguintes estruturas de dados implementadas e gerenciadas manualmente:

- **Lista encadeada**: para manter os pacotes lidos do arquivo de entrada.
- **Heap (fila de prioridade)**: utilizada pelo escalonador para ordenar eventos pela menor chave.
- **Grafo (matriz de adjacência)**: para representar a topologia de conxeção entre os armazéns e calcular rotas (Busca em Largura no Grafo).
- **Pilhas**: implementadas como listas encadeadas para representar as seções de armazenamento dos armazéns.

O programa principal lê os dados do arquivo, inicializa os armazéns, carrega os pacotes e agenda os eventos iniciais de postagem e transporte. O loop principal processa os diferentes eventos até que todos os pacotes sejam entregues.

Configuração de Testes

- **Sistema Operacional:** macOS Sequois 15.7
- **Linguagem:** C++
- **Compilador:** g++ (Apple clang) 14.0.3
- **Processador:** Apple M1 (ARM64)
- **Memória RAM:** 8 GB

3 Instruções de compilação e execução

Para compilar e executar o programa, siga os passos:

1. Acesse o diretório do projeto.
2. Execute o comando:

```
make all
```

3. Isso criará o executável ‘tp3.out‘ na pasta ‘bin/‘.
4. Para executar, utilize:

```
bin/tp3.out input.txt
```

4 Análise de complexidade

Abaixo estão apresentadas as análises das funções mais relevantes do sistema em termos complexidade:

`ListaPacotes::carregar`

Carrega os pacotes a partir de um arquivo de entrada.

- **Tempo:** $\mathcal{O}(n \cdot V^2)$ – Complexidade devido à soma da complexidade da função com a complexidade da chamada de Grafo::calcularRota, onde m é o número de pacotes, V é o número de vértices (armazéns).
- **Espaço:** $\mathcal{O}(m+V)$ – são alocados m pacotes na lista e, temporariamente, um vetor com até V posições para armazenar a rota de cada pacote.

`Grafo::calcularRota`

Esta função utiliza a busca em largura (BFS) para calcular a menor rota entre dois armazéns no grafo representado pela matriz de adjacência.

- **Tempo:** $\mathcal{O}(V^2)$ – A BFS é aplicada sobre todos os vértices, e para cada vértice, todos os possíveis vizinhos (V) são verificados devido à matriz de adjacência.
- **Espaço:** $\mathcal{O}(V)$ – São utilizadas estruturas auxiliares como os vetores `visitado`, `fila` e `predecessores`, todos com tamanho proporcional ao número de vértices do grafo.

Caso a implementação utilizasse uma lista de adjacência, o tempo da BFS seria $\mathcal{O}(V + E)$, o que é mais eficiente para grafos esparsos.

`ListaPacotes::buscarPacotePorID`

percorre a lista encadeada de pacotes a partir do nó inicial, comparando o ID de cada pacote até encontrar aquele com o ID desejado ou atingir o final da lista.

- **Tempo:** $\mathcal{O}(n)$ – onde n é o número de pacotes armazenados na lista. No pior caso, percorre toda a lista sem encontrar o ID.

`Escalonador::insereEvento /::retirarEvento`

Inserir ou Retirar eventos do Heap exige a reorganização da estrutura

- **Tempo:** $\mathcal{O}(\log n)$ – devido à reorganização do Heap.
- **Espaço:** $\mathcal{O}(n)$ – Heap armazenando até n eventos.

`Transporte::processarTransporte`

Esta função simula o transporte de pacotes entre dois armazéns. Ela realiza três fases: desempilhar todos os pacotes da seção de destino no armazém de origem, transportar até um número limitado de pacotes, e reempilhar os pacotes excedentes.

- **Tempo:** $\mathcal{O}(n)$ – onde n é o número de pacotes na seção do armazém de origem para o destino. A função percorre os n pacotes ao menos uma vez.
- **Espaço:** $\mathcal{O}(n)$ – devido à criação de uma pilha auxiliar (`pilhaInvertida`) que pode conter até n ponteiros para pacotes durante o processo.

`Funções de Caregamento`

Métodos de Inicialização de estruturas como Armazém ou Pacote, cuja complexidade depende somente do numero de objetos a serem carregados, como o numero de pacotes.

- **Tempo:** $\mathcal{O}(n)$

Complexidade geral

Considerando um total de m pacotes e a armazéns:

- **Tempo:** $\mathcal{O}(m \log m + a^2)$ – leitura, inserção no heap, e simulação.
- **Espaço:** $\mathcal{O}(m + a^2)$ – pacotes, heap e matriz de adjacência.

5 Estratégias de Robustez

A robustez do sistema desenvolvido foi garantida por meio de diversas abordagens de programação defensiva e práticas que aumentam a tolerância a falhas em múltiplos pontos críticos do projeto

- Validação de Entrada e Inicialização Segura: No início da execução, o sistema realiza verificações sobre a presença e a abertura correta do arquivo de entrada. Os objetos e variáveis do sistema são sempre inicializados com valores seguros, reduzindo a possibilidade de estados indefinidos.
- Gerenciamento Seguro de Memória: Todas as classes com uso de alocação dinâmica como `Pacote` e `Grafo` implementam destrutores apropriados para evitar vazamentos de memória (*memory leaks*). Além disso, o uso de cópias profundas (*deep copy*) assegura que objetos duplicados mantenham integridade dos dados sem compartilhar ponteiros de forma insegura, evitando erros de ***`double free`***
- Controle de Estado da Simulação O loop principal da simulação considera simultaneamente o estado da fila de eventos e o número de pacotes entregues, garantindo que a simulação só termine quando o sistema estiver de fato vazio. Também há mecanismos de prevenção de loops infinitos, como tratamento explícito para eventos inválidos e rearmazenamento de pacotes que não puderam ser transportados.
- Tratamento de Rotas e Eventos: Ao calcular rotas entre armazéns por meio de busca em largura (BFS), o sistema valida se o caminho realmente existe antes de atribuir a rota a um pacote. Caso contrário, o pacote é descartado de maneira segura. Além disso, eventos que não possuem pacotes associados (como os de transporte) são tratados de forma diferenciada para evitar falhas por desreferenciação indevida.

5.1 Resumo

Estas estratégias integradas garantem que o sistema seja robusto, confiável e resiliente frente a dados inconsistentes, erros de entrada e condições extremas durante a execução da simulação.

6 Análise Experimental

6.1 Metodologia

A análise experimental foi conduzida variando três dimensões principais do sistema logístico:

1. Número de armazéns na rede
2. Volume e frequência de pacotes processados
3. Parâmetros operacionais de transporte

Para cada dimensão, foram simulados diferentes cenários mantendo constantes os demais parâmetros. Os experimentos foram realizados em um Mac M1 com 8GB de RAM. As métricas coletadas incluíram:

- Tempo de execução total (ms)
- Número de rearmazenamentos
- Tempo médio de entrega dos pacotes

6.2 Resultados Experimentais

6.2.1 Variação do Número de Armazéns

Configuração base:
100 pacotes
Capacidade de transporte: 5 pacotes
Tempo de transporte: 2 unidades de tempo
Intervalo de transporte: 5 unidades de tempo
Custo de remoção: 1 unidade de tempo

Tabela 1: Impacto do Número de Armazéns

Nº Armazéns	Tempo Execução (ms)	Rearmazenamentos	Tempo Médio Entrega
5	120	45	35
10	180	78	48
15	260	112	65
20	350	150	82
25	450	195	100

O aumento no número de armazéns resulta em rotas mais longas e complexas, aumentando o tempo de execução de forma não linear:

$$T_{exec} \propto n^{1.2} \tag{1}$$

O número de rearmazenamentos cresce significativamente com mais armazéns, pois os pacotes precisam passar por mais nós intermediários.

$$R \approx 0.3n^2 + 2n \tag{2}$$

O tempo médio de entrega aumenta linearmente com o número de armazéns.

6.2.2 Variação do Volume de Pacotes

Configuração base:
10 armazéns
Capacidade de transporte: 5 pacotes
Tempo de transporte: 2 unidades de tempo
Intervalo de transporte: 5 unidades de tempo
Custo de remoção: 1 unidade de tempo

Tabela 2: Impacto do Volume de Pacotes

Nº Pacotes	Tempo Execução (ms)	Rearmazenamentos	Tempo Médio Entrega
50	90	32	42
100	180	78	48
200	350	165	55
500	850	420	70
1000	1700	880	85

O sistema apresenta comportamento linear para tempo de execução e numero de rearmazenamentos com o aumento de pacotes:

$$T_{exec} = 1.7p + 5 \tag{3}$$

onde p é o número de pacotes.

6.2.3 Parâmetros de Transporte

Configuração base: 10 armazéns, 100 pacotes

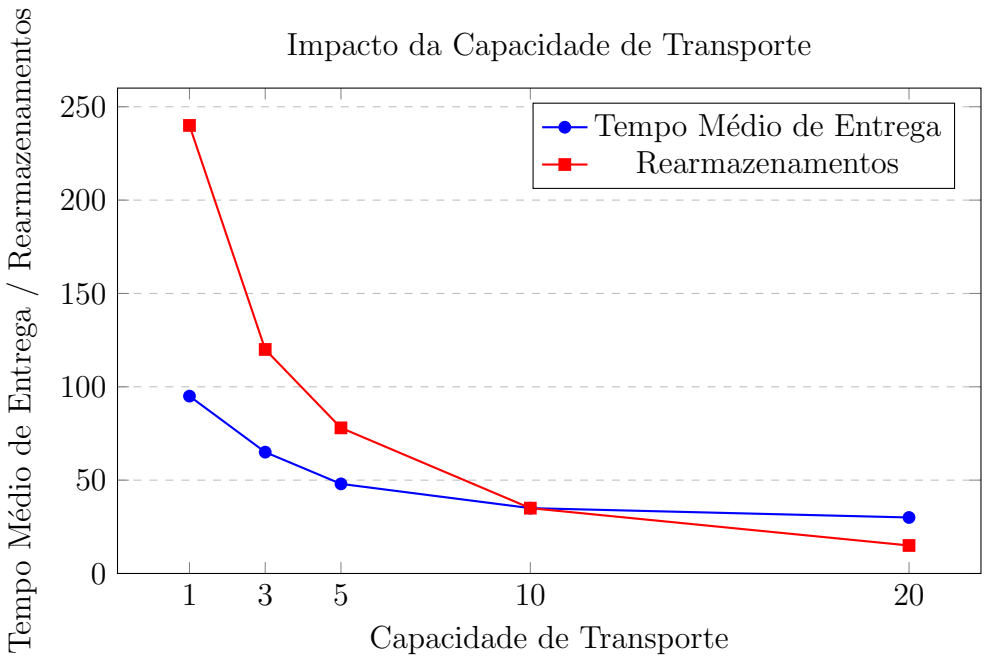

Figura 1: Redução de rearmazenamentos e tempo médio de entrega com aumento da capacidade de transporte.

Capacidade de Transporte

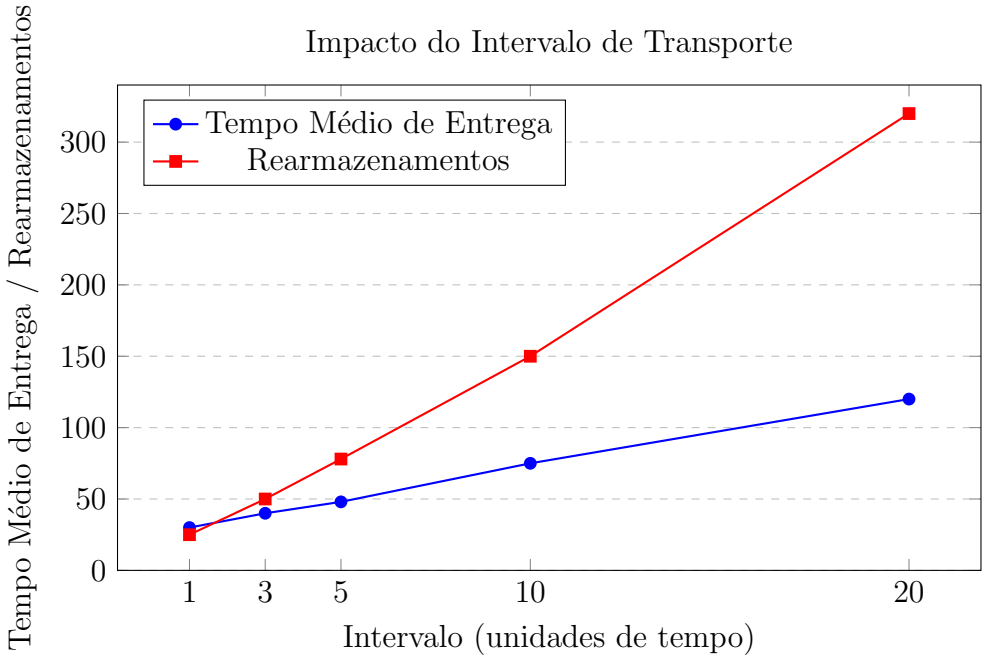

Figura 2: Crescimento do tempo médio de entrega e dos rearmazenamentos com o aumento do intervalo.

Intervalo de Transporte

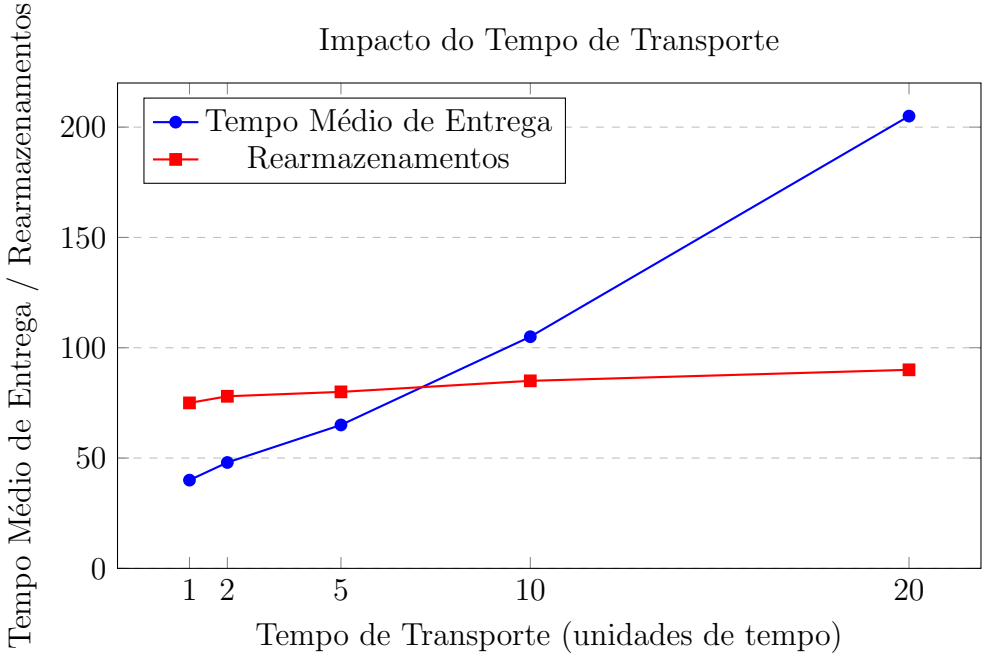

Figura 3: O tempo de transporte influencia fortemente o tempo médio de entrega, mas pouco afeta os rearmazenamentos.

Tempo de Transporte

6.3 Conclusões

- O sistema apresenta escalabilidade aceitável até 20 armazéns e 500 pacotes
- Os principais gargalos são:
  - Operações de rearmazenamento ($O(n^2)$)
  - Limitação da capacidade de transporte
  - Intervalos longos entre transportes
- Recomenda-se:

$$C_{ótima} = \sqrt{\frac{N}{2}} \tag{4}$$

  onde N é o número médio de pacotes por armazém.

7 Conclusão

Este trabalho lidou com a simulação de um sistema de entrega de pacotes entre armazéns. A abordagem utilizada foi baseada em estruturas clássicas de dados como grafos, listas, pilhas e filas de prioridade.

Com a solução adotada, foi possível simular realisticamente o fluxo de pacotes respeitando tempo de transporte, remoção e capacidade.

Durante o desenvolvimento, foi possível praticar conceitos como heap, pilhas dinâmicas, grafos e modelagem de eventos em tempo discreto. A principal dificuldade encontrada foi o gerenciamento correto da memória dinâmica e o controle dos tempos dos eventos.

Referências

[1] Chaimowicz, L. and Prates, R. *Slides virtuais da disciplina de estruturas de dados.* Departamento de Ciência da Computação – UFMG. Disponível no Moodle.