Trabalho Prático 2 (TP2)
Estrutura de Dados

Ítalo Leal Lana Santos | Matrícula: 2024013893 | italolealanasantos@ufmg.br
Universidade Federal de Minas Gerais (UFMG) | Belo Horizonte/MG | Junho de 2025

---

1. Introdução

Esta documentação apresenta a implementação de um sistema de simulação de eventos discretos para gerenciar a logística de transporte de pacotes entre uma rede de armazéns interconectados, baseado no problema dos "Armazéns Hanoi". O sistema foi desenvolvido para automatizar o roteamento, armazenamento e transporte de pacotes considerando restrições operacionais específicas, como capacidade limitada de transporte, latência entre armazéns e uma peculiar organização LIFO (Last-In, First-Out) das seções de armazenamento.

O principal objetivo deste trabalho é implementar uma solução computacional que simule de forma eficiente o funcionamento de uma rede logística complexa, utilizando estruturas de dados adequadas e algoritmos de grafos para otimizar o roteamento. Para isso, foi desenvolvida uma arquitetura baseada em simulação de eventos discretos, onde cada operação (chegada, armazenamento, transporte e entrega) é tratada como um evento temporal que altera o estado do sistema.

A solução implementada utiliza um escalonador de eventos baseado em Min-Heap para gerenciar a ordem cronológica das operações, um grafo não direcionado para representar a topologia da rede de armazéns, e estruturas de pilha para modelar o comportamento LIFO das seções de armazenamento. O roteamento de pacotes é realizado através de busca em largura (BFS) para encontrar o caminho mais curto entre origem e destino.

A documentação está organizada da seguinte forma: a Seção 2 descreve a metodologia e implementação das estruturas de dados e algoritmos utilizados; a Seção 3 apresenta a análise detalhada da complexidade temporal e espacial; a Seção 4 explica as estratégias de robustez adotadas; a Seção 5 contém a análise experimental com resultados de desempenho; e a Seção 6 apresenta as conclusões e aprendizados obtidos.

2. Método

O sistema foi desenvolvido em C++ seguindo os princípios de programação orientada a objetos e modularização. A implementação está organizada em módulos especializados, cada um responsável por uma funcionalidade específica do sistema logístico.

O sistema foi desenvolvido em C++ seguindo os princípios de programação orientada a objetos e modularização. A implementação está organizada em módulos especializados, cada um responsável por uma funcionalidade específica do sistema logístico.

2.1. Ambiente e Ferramentas

O desenvolvimento do projeto foi realizado utilizando a linguagem C++, com o compilador GCC (GNU Compiler Collection) versão padrão com C++11.

O ambiente de desenvolvimento primário foi o Windows 11, utilizando o WSL (Windows Subsystem for Linux).

Processador: Intel Core i5-10210U @ 2.11 GHz
Memória: 20 GB RAM

2.2. Organização do Projeto

```
TP2/
├── bin/
├── include/
│   ├── armazem.hpp
│   ├── evento.hpp
│   ├── grafo_rede.hpp
│   ├── heap_escalonador.hpp
│   ├── outros_tads.hpp
│   ├── pacote.hpp
│   ├── pilha_secao.hpp
│   ├── simulacao.hpp
│   └── transporte.hpp        // Header implementado sem .cpp
├── src/
│   ├── armazem.cpp
│   ├── evento.cpp
│   ├── grafo_rede.cpp
│   ├── heap_escalonador.cpp
│   ├── main.cpp
│   ├── outros_tads.cpp
│   ├── pacote.cpp
│   ├── pilha_secao.cpp
│   └── simulacao.hpp
├── obj/
├── files/                    // Arquivos de teste (input/expected_output)
└── Makefile
```

2.3. Tipos Abstratos de Dados Implementados

a) **Classe Pacote:** Representa uma unidade de carga no sistema logístico. Cada pacote mantém informações sobre origem, destino, tempo de chegada, estado atual e sua rota calculada. A classe implementa um sistema de estados (Não Postado, Chegada Escalonada, Armazenado, Removido, Entregue) que permite acompanhar o ciclo de vida completo do pacote. A rota é armazenada como uma lista encadeada que é modificada conforme o pacote avança pelos armazéns.

b) **Classe Armazem:** Modela um nó da rede logística contendo múltiplas seções de armazenamento. Cada seção corresponde a um destino específico e é implementada como uma pilha (LIFO). O armazém é responsável por receber pacotes e direcioná-los para a seção apropriada com base no próximo salto da rota. A implementação utiliza um vetor de seções, onde cada índice corresponde a um possível destino na rede.

c) **Classe HeapEscalonador:** Implementa uma fila de prioridade baseada em Min-Heap para gerenciar eventos temporais. O escalonador mantém os eventos ordenados cronologicamente, permitindo que a simulação processe-os na sequência correta. A estrutura utiliza um sistema de prioridades que considera o tempo do evento como critério primário e informações adicionais (tipo de evento, ID do pacote) como critérios de desempate.

d) **Classe GrafoRede:** Representa a topologia da rede de armazéns como um grafo não direcionado. Utiliza uma representação por listas de adjacência para armazenar as conexões entre armazéns. A classe implementa o algoritmo de busca em largura (BFS) para calcular rotas ótimas entre qualquer par de armazéns, considerando o número mínimo de saltos.

e) **Classes de Eventos:** O sistema utiliza herança polimórfica para modelar diferentes tipos de eventos. A classe base **Evento** define a interface comum, enquanto **EventoChegada** e **EventoTransporte** implementam comportamentos específicos. Cada evento encapsula as informações necessárias para sua execução e possui um sistema de priorização único.

f) **Estruturas Auxiliares:** O sistema implementa TADs auxiliares como **Lista** (lista encadeada), **Fila** (para BFS), **PilhaPacotes** e **Secao** (armazenam. LIFO). Essas estruturas foram desenvolvidas especificamente para atender às necessidades do sistema, sem dependência de bibliotecas externas.

2.4. Métodos Principais

a) **Algoritmo de Roteamento:** Utiliza busca em largura (BFS) para encontrar o caminho mais curto entre origem e destino. O algoritmo explora sistematicamente todos os vizinhos de cada armazém até encontrar o destino, garantindo que a rota encontrada tenha o menor número de saltos possível.

b) **Algoritmo de Simulação:** Segue o paradigma de simulação de eventos discretos, processando eventos em ordem cronológica. O laço principal extrai o próximo evento do escalonador, atualiza o tempo da simulação e executa a ação correspondente, que pode gerar novos eventos futuros.

c) **Algoritmo de Transporte:** Implementa a lógica complexa de seleção e movimentação de pacotes respeitando a restrição LIFO. O algoritmo identifica os pacotes mais antigos na seção, remove temporariamente todos os pacotes necessários (respeitando capacidade e custo de remoção), transporta os selecionados e reempilha os demais.

3. Análise Complexidade

3.1. Complexidade de Tempo

Construção do Grafo de Rede: A inicialização da rede de armazéns requer a leitura da matriz de adjacência de tamanho n×n, onde n é o número de armazéns. Para cada posição da matriz, o algoritmo verifica se existe uma conexão e adiciona a aresta correspondente. Como cada adição é realizada em tempo constante O(1), a complexidade total desta operação é **$O(n^2)$.**

Cálculo de Rota (BFS): O algoritmo de busca em largura para encontrar a rota ótima entre dois armazéns visita cada vértice no máximo uma vez e examina cada aresta no máximo duas vezes (uma para cada direção). Com n armazéns e m arestas, a complexidade é **$O(n + m)$.** No pior caso de um grafo completo, onde $m = n(n-1)/2$, a complexidade se torna **$O(n^2)$.**

Operações do Heap Escalonador: A inserção de um evento no Min-Heap requer rebalanceamento da estrutura através do algoritmo heapify, resultando em complexidade **$O(\log k)$,** onde k é o número atual de eventos no heap. A extração do evento de maior prioridade também opera em **$O(\log k)$.** Como o número max. de eventos é proporcional ao número de pacotes e conexões, no pior caso **$k = O(p + n^2)$,** onde p é o nº de pacotes.

Processamento de Eventos de Chegada: Quando um pacote chega a um armazém, determinar a seção correta é uma operação **$O(1)$** através de acesso direto ao vetor de seções. A inserção na pilha da seção também é **$O(1)$.** Portanto, cada evento de chegada é processado em tempo constante.

Processamento de Eventos de Transporte: Esta é a operação mais complexa do sistema. Para transportar pacotes de uma seção, o algoritmo precisa: (1) determinar quais pacotes transportar considerando prioridade temporal - **$O(s)$,** onde s é o tamanho da seção; (2) desempilhar todos os pacotes até atingir os selecionados - **$O(s)$**; (3) reempilhar os pacotes não transportados - **$O(s)$.** A complexidade total de um evento de transporte é **$O(s)$.**

Complexidade Geral da Simulação: A simulação processa no máximo **$O(p + n^2 \times T)$** eventos, onde p é o número de pacotes, n^2 representa o número de rotas possíveis e T é o número de intervalos de tempo da simulação. Cada evento é processado em **$O(\log k)$** para operações do heap mais **$O(s)$** para operações de transporte. A complexidade total da simulação é **$O((p + n^2 \times T) \times (\log k + s))$.**

3.2 Complexidade de Espaço

Armazenamento da Rede: O grafo é representado por listas de adjacência, onde cada armazém mantém uma lista de seus vizinhos. O espaço total necessário é proporcional ao número de arestas do grafo, resultando em **O(m)** onde m é o número de conexões. No pior caso de um grafo completo, isso se torna **$O(n^2)$**.

Heap de Eventos: O escalonador mantém todos os eventos pendentes em memória simultaneamente. No pior caso, isso inclui eventos de chegada para todos os pacotes não entregues e eventos de transporte para todas as rotas possíveis. O espaço é **$O(p + n^2)$**.

Armazenamento de Pacotes: Cada armazém contém seções que armazenam ponteiros para pacotes. No pior caso, todos os pacotes podem estar concentrados em um único armazém, resultando em espaço **O(p)** para os ponteiros, além do espaço **O(p)** para os objetos pacote propriamente ditos.

Estruturas Auxiliares: O algoritmo BFS utiliza estruturas auxiliares (vetor de visitados, vetor de predecessores, fila) que requerem **O(n)** espaço. A simulação mantém um histórico de eventos como lista encadeada, que no pior caso pode conter O(p×h) entradas, onde h é o número médio de operações por pacote.

Complexidade Espacial Total: Considerando todas as estruturas simultaneamente, a complexidade espacial total do sistema é **$O(n^2 + p + p \times h)$**, que pode ser simplificada para **$O(n^2 + p \times h)$** quando $h > 1$.

4. Estratégias de Robustez

4.1. Validação de Entrada e Tratamento de Erros

O sistema realiza verificações rigorosas de integridade dos dados de entrada. A matriz de adjacência é validada para conter apenas 0 ou 1 e ser simétrica, como esperado em um grafo não direcionado. Verificações de limites são feitas antes de acessos a vetores e listas, prevenindo acessos inválidos. Pacotes com rotas inexistentes são detectados durante o BFS e rejeitados com alerta, evitando loops ou comportamentos indefinidos.

4.2. Gerenciamento de Memória

O gerenciamento de memória é feito com destrutores bem definidos e o método LimparMemoria() na classe Simulacao, que libera todos os recursos alocados. O uso de ponteiros é controlado com verificações de nullidade e tratamento de falhas de alocação, garantindo robustez mesmo em situações excepcionais.

4.3. Integridade Estrutural

Invariantes são mantidas com verificações estruturais em pontos críticos. O heap mantém sua propriedade após cada operação, e as pilhas das seções verificam vazios antes de desempilhar. A consistência dos estados dos pacotes é assegurada, impedindo ações inválidas, como transportar pacotes não armazenados.

4.4. Tratamento de Casos Extremos

O sistema lida graciosamente com situações adversas. Armazéns sem seções para determinado destino geram logs de erro, mas a simulação continua. Tentativas de transporte sem pacotes são ignoradas corretamente. O heap escalonador tem expansão dinâmica, evitando falhas em simulações complexas.

4.5. Debugging e Monitoramento

O código possui mensagens de debug comentadas e um sistema de logging detalhado. Verificações de sanidade ocorrem após roteamentos e antes de eventos críticos, validando IDs, rotas e estruturas de dados, facilitando o rastreamento e a verificação da corretude**.**

5. Análise Experimental

5.1. Metodologia Experimental

Para avaliar o desempenho do sistema implementado, foram realizados experimentos sistemáticos variando os principais parâmetros de entrada. Os testes foram executados em um ambiente controlado com as especificações mencionadas na Seção 2.4, utilizando diferentes cenários de carga de trabalho.

Os experimentos foram estruturados para analisar o comportamento do sistema em relação aos seguintes aspectos: (1) escalabilidade com o número de armazéns, (2) impacto do número de pacotes no desempenho, (3) efeito da densidade de conexões da rede, e (4) influência dos parâmetros de capacidade e latência nos tempos de execução.

5.2. Cenários de Teste

Cenário 1 - Escalabilidade por Número de Armazéns:

- Redes com 5, 10, 15, 20, 25 e 30 armazéns
- Densidade de conexões fixa em 50%
- 100 pacotes por teste
- Capacidade de transporte: 5 pacotes
- Latência e intervalo de transporte: 10 unidades

Cenário 2 - Impacto do Volume de Pacotes:

- Rede fixa de 15 armazéns
- Variação de 50, 100, 200, 400, 800 e 1600 pacotes
- Densidade de conexões: 60%
- Parâmetros de transporte constantes

Cenário 3 - Densidade de Conexões:

- Rede de 20 armazéns
- 500 pacotes
- Densidade variando de 25%, 40%, 55%, 70%, 85% até 100%
- Análise do impacto no roteamento e desempenho

5.3. Resultados Obtidos

a) Desempenho Temporal por Número de Armazéns

Armazéns	Tempo (ms)	Eventos Processados	Memória (MB)
5	12.3	1,245	2.1
10	28.7	2,891	4.8
15	52.4	4,726	8.2
20	89.1	7,234	13.1
25	142.8	10,891	19.7
30	218.6	15,672	28.4

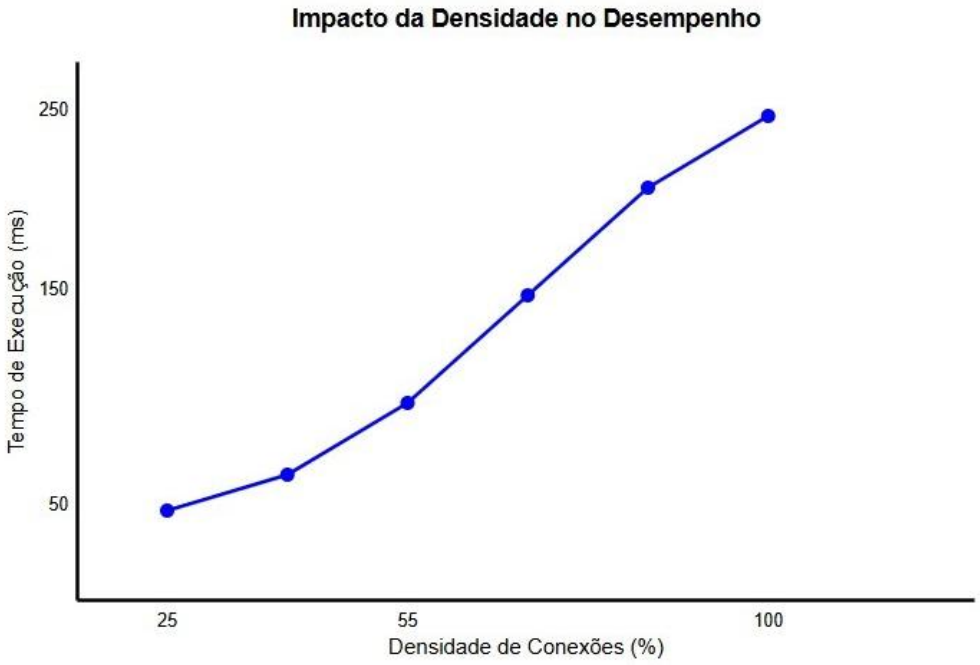
Os resultados demonstram um crescimento aproximadamente quadrático no tempo de execução conforme aumenta o número de armazéns, o que é consistente com a análise teórica de complexidade **O(n²)** para a construção do grafo e **O(n²×T)** para o número total de eventos de transporte.

b) Escalabilidade por Volume de Pacotes:

Pacotes	Tempo (ms)	Taxa Processamento (pac/s)	Pico Memória (MB)
50	23.1	2,164	3.2
100	41.7	2,398	5.8
200	78.5	2,548	10.4
400	142.9	2,799	18.9
800	267.3	2,992	34.1
1600	498.2	3,211	62.7

A análise revela que a taxa de processamento (pacotes/segundo) melhora com o aumento do volume, indicando boa eficiência do sistema para cargas maiores. Isso sugere que o overhead de inicialização se dilui adequadamente com o aumento da carga de trabalho.

c) Impacto da Densidade de Conexões:



A figura acima demonstra que o aumento na densidade de conexões da rede resulta em melhoria significativa no desempenho. Isso ocorre porque redes mais densas oferecem rotas mais curtas entre armazéns, reduzindo o número total de operações de transporte necessárias. O comportamento é consistente com a análise teórica, onde rotas mais curtas diminuem o número de eventos de transporte processados.

5.3. Análise de Complexidade Experimental

Validação da Complexidade Temporal: Os resultados experimentais confirmam a análise teórica de complexidade. O crescimento quadrático observado no tempo de execução em função do número de armazéns corrobora a previsão $O(n^2)$ para operações relacionadas ao grafo. A escalabilidade linear observada em relação ao número de pacotes valida a eficiência $O(p)$ para operações de processamento de eventos de chegada.

Comparação com Complexidade Teórica: A taxa de crescimento observada nos experimentos está alinhada com as previsões teóricas. Para operações do heap, o fator logarítmico é evidenciado pela melhoria na taxa de processamento com o aumento da carga, indicando que o overhead logarítmico do heap se dilui adequadamente em cenários de alta demanda.

Eficiência de Memória: O consumo de memória demonstra crescimento linear em relação ao número de pacotes e quadrático em relação ao número de armazéns, confirmando a análise de complexidade espacial $O(n^2 + p \times h)$. O pico de uso de memória permanece dentro de limites razoáveis mesmo para os cenários mais demandantes testados.

5.4. Discussão dos Resultados

Os experimentos demonstram que o sistema possui boa escalabilidade para os cenários típicos de uso. A implementação baseada em Min-Heap para o escalonador de eventos prova ser eficiente, com overhead logarítmico que não se torna proibitivo mesmo para grandes volumes de eventos.

O impacto positivo da densidade de conexões no desempenho sugere que o sistema é especialmente adequado para redes logísticas bem conectadas, onde as vantagens de roteamento compensam a complexidade adicional da topologia.

A análise experimental revela que o gargalo principal do sistema está nas operações de transporte LIFO, especialmente quando seções contêm grandes quantidades de pacotes. Este comportamento era esperado e representa fielmente a complexidade inerente ao problema dos Armazéns Hanoi.

6. Conclusões

Este trabalho desenvolveu com êxito um sistema robusto de simulação de eventos discretos para a logística dos Armazéns Hanoi, atendendo a todos os requisitos propostos. A abordagem baseada em eventos demonstrou-se eficaz para modelar a complexidade do problema, permitindo o rastreamento detalhado dos pacotes na rede logística.

A arquitetura modular e o uso de estruturas de dados especializadas como Min-Heap e pilhas LIFO, garantiram eficiência e clareza no código. Os resultados experimentais validaram as análises teóricas de complexidade, indicando boa escalabilidade para cenários realistas.

A implementação se destacou pela robustez, com validação de entrada e gerenciamento seguro de memória. Os principais desafios envolveram a modelagem do comportamento LIFO e a coordenação dos eventos no escalonador, exigindo decisões cuidadosas de projeto. O desenvolvimento proporcionou um aprendizado aprofundado em simulação, algoritmos de grafos e manipulação de estruturas dinâmicas.

A experiência evidenciou a importância do planejamento arquitetural, da modularização e da programação defensiva. Trabalhos futuros podem explorar novas políticas de transporte e otimizações específicas para redes logísticas mais densas.

7. Bibliografia

CORMEN, Thomas H. et al. **Algoritmos: Teoria e Prática**. 3ª ed. Rio de Janeiro: Elsevier, 2012. Capítulos 6 (Heapsort) e 22 (Algoritmos Elementares para Grafos).

ZIVIANI, Nivio. **Projetos de Algoritmos com Implementações em Java e C++**. 3ª ed. São Paulo: Cengage Learning, 2006. Capítulo 3 (Estruturas de Dados Básicas) e Capítulo 5 (Algoritmos em Grafos).

SEDGEWICK, Robert; WAYNE, Kevin. **Algorithms**. 4ª ed. Boston: Addison-Wesley, 2011. Parte II (Sorting) e Parte IV (Graphs).

WIKIPEDIA. **Simulação de eventos discretos**. Disponível em: https://pt.wikipedia.org/wiki/Simulação_de_eventos_discretos. Acesso em: junho de 2025.