Trabalho Prático 3
Consultas ao Sistema Logístico

Lucas Pimenta Braga - 2023034552

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

lucaspimentabraga@gmail.com

1. Introdução

Este trabalho tem como objetivo implementar e analisar um sistema de consultas ao sistema logístico dos Armazéns Hanoi, a partir do processamento cronológico de eventos extraídos de um arquivo de entrada. O sistema possibilita consultas que retornam o histórico de um pacote específico ou apresentam o levantamento dos pacotes relacionados a um determinado cliente, informando o último estado conhecido até o momento da consulta.

O modelo é fundamentado em um histórico completo de eventos que representam as operações logísticas ocorridas, como registro (RG), armazenamento (AR), remoção (RM), rearmazenamento (UR), transporte (TR) e entrega (EN). Cada evento é processado em ordem temporal, o que garante a consistência das informações utilizadas na construção dos índices de pacotes e clientes.

A implementação foi realizada em C++, empregando estruturas de dados estáticas, como arrays e matrizes, e executada em ambiente Linux, utilizando apenas bibliotecas padrão da linguagem. O projeto foi estruturado com foco em modularidade, simplicidade e eficiência, permitindo avaliar o desempenho do sistema em diferentes cenários experimentais, variando parâmetros como o número de pacotes, o número de clientes e a complexidade da malha logística.

2. Método

O sistema foi estruturado para processar o arquivo de entrada linha a linha, identificando cada linha como um evento (EV) ou uma consulta (PC ou CL). Durante essa leitura, são construídas em memória três estruturas principais:

- **Histórico Global de Eventos**:
  Todos os eventos são armazenados sequencialmente em um array fixo, controlado por um contador. Cada evento contém o timestamp, tipo, identificador do pacote e a linha completa do evento, permitindo reconstrução fiel do histórico.
- **Índice de Pacotes**:
  Cada pacote possui um identificador único e um array que armazena ponteiros para os eventos que compõem o seu histórico. Esse índice viabiliza consultas do tipo PC, permitindo recuperar rapidamente toda a trajetória do pacote até determinado instante de tempo.

- **Índice de Clientes**:
  Cada cliente identificado como remetente ou destinatário em eventos de registro (RG) mantém seu nome e um array de IDs dos pacotes nos quais esteve envolvido. Isso possibilita consultas do tipo CL, que recuperam o último estado conhecido de cada pacote do cliente até o momento consultado.

2.1 Estruturas de Dados Utilizadas

2.1.1 Histórico Global de Eventos

O sistema mantém um array fixo Evento* eventos[10000], controlado por um contador totalEventos. Cada elemento do array aponta para um struct Evento, que armazena:

- datahora: instante do evento no sistema logístico.
- tipo: string representando o tipo do evento (RG, AR, RM, UR, TR, EN).
- pacoteID: identificador único do pacote ao qual o evento pertence.
- linhaCompleta: string com o texto original do evento no arquivo de entrada, preservando o formato para saída.

Essa estrutura é o **repositório cronológico global** do sistema, garantindo que todos os eventos sejam processados e mantidos em ordem de ocorrência.

2.1.2 Índice de Pacotes

Cada pacote é representado por um struct Pacote, que contém:

- id: identificador único do pacote.
- historico[1000]: array fixo de ponteiros para eventos associados a esse pacote.
- totalEventosPacote: contador que indica quantos eventos já foram associados ao pacote.

Esse índice possibilita consultas diretas do tipo PC <id>, recuperando rapidamente o histórico completo de um pacote até determinado instante.

2.1.3 Índice de Clientes

Cada cliente é descrito por um struct Cliente, com:

- nome: string com o nome do cliente.
- pacotes[1000]: array fixo com os identificadores dos pacotes nos quais o cliente figura como remetente ou destinatário.
- totalPacotes: contador de quantos pacotes estão associados ao cliente.

Esse índice é a base para consultas CL <nome>, permitindo encontrar todos os pacotes de um cliente e retornar o último estado conhecido de cada um até o momento da consulta.

2.2 Funções Implementadas

2.2.1 Função novoEvento

Responsável por criar um evento a partir dos dados extraídos da linha do arquivo. Suas etapas principais são:

- Aloca memória para o novo evento.
- Armazena a data-hora, tipo, ID do pacote e a linha original.
- Insere o ponteiro do evento no array eventos[totalEventos++].

2.2.2 Função addEventoPacote

Atualiza o índice do pacote:

- Recebe o ponteiro para o Pacote e o Evento.
- Insere o evento no array histórico do pacote.
- Incrementa totalEventosPacote.

2.2.3 Função addPacoteCliente

Mantém o índice do cliente:

- Recebe o ponteiro para o Cliente e o ID do pacote.
- Insere o ID no array pacotes.
- Incrementa totalPacotes.

2.2.4 Função consultaPacote

Executa a consulta PC:

- Busca o pacote pelo ID.
- Percorre o array histórico do pacote.
- Imprime todos os eventos até o tempo requisitado.

2.2.5 Função consultaCliente

Executa a consulta CL:

- Localiza o cliente pelo nome.
- Para cada pacote do cliente, identifica o último evento até o tempo da consulta.
- Imprime esses últimos eventos, formando o resumo do estado dos pacotes.

2.2.6 Funções auxiliares

- **Leitura do arquivo**: identifica linhas EV, CL ou PC e direciona para a função correta.
- **Impressão**: usa printf com formatação padronizada para garantir a saída nos moldes esperados pelo problema.

3. Análise de Complexidade

Esta seção avalia o desempenho dos algoritmos implementados no sistema de consultas do logístico dos Armazéns Hanoi, considerando tempo de execução e uso de memória. A seguir, são analisadas as principais funções do sistema.

3.1. Função de carregamento e indexação dos eventos

Responsável por ler o arquivo de entrada, criar os eventos e atualizar os índices de pacotes e clientes.

- **Complexidade de Tempo:**
  Cada linha do arquivo é processada uma única vez, com operações constantes para criação do evento, atualização do pacote e do cliente. Assim, para n linhas (entre eventos e consultas), temos complexidade O(n).
- **Complexidade de Espaço:**
  São alocados arrays fixos para armazenar eventos, pacotes e clientes. A memória cresce proporcional ao número de eventos E, pacotes P e clientes C, resultando em O(E + P + C).

3.2. Função consultaPacote

Executa a consulta do tipo PC, retornando o histórico do pacote até um tempo específico.

- **Complexidade de Tempo:**
  Percorre o array de eventos do pacote até encontrar eventos com timestamp maior. Se um pacote tem k eventos, então O(k).
- **Complexidade de Espaço:**
  Não aloca estruturas adicionais além do necessário para imprimir os resultados. Logo, O(1).

3.3. Função consultaCliente

Executa a consulta do tipo CL, retornando o último estado conhecido de cada pacote associado ao cliente até o instante consultado.

- **Complexidade de Tempo:**
  Para um cliente com q pacotes associados, a função percorre cada pacote e busca o último evento em seu histórico até o tempo requisitado. Assim, O(q × t), onde t é o tamanho médio do histórico dos pacotes.
- **Complexidade de Espaço:**
  Utiliza espaço constante para controlar o estado temporário da busca. Portanto, O(1).

3.4. Funções de impressão

Responsáveis por gerar as saídas formatadas das consultas.

- **Complexidade de Tempo:**
  Cada chamada imprime uma linha, e para m linhas de resultado temos O(m).
- **Complexidade de Espaço:**
  Não utilizam estruturas auxiliares além de variáveis locais, resultando em O(1).

3.5. Funções auxiliares e manipulação dos índices

Incluem a adição de eventos ao vetor global, associação do evento ao pacote e atualização da lista de pacotes do cliente.

- **Complexidade de Tempo:**
  Essas operações são realizadas em O(1), pois consistem em inserções diretas em arrays controlados por contadores.
- **Complexidade de Espaço:**
  A memória já está pré-alocada pelos arrays fixos, não havendo custo adicional além do controle dos contadores. Portanto, O(1).

3.6. Complexidade Total do Sistema

Em relação **a complexidade de tempo**, o tempo total do sistema é dominado por duas fases principais:

1. **Carregamento e indexação dos eventos:**
   Cada linha do arquivo é processada uma única vez para construir os índices, com custo O(n).
2. **Execução das consultas:**
   Depende do tipo e quantidade de consultas. No pior caso:
   - Uma consulta PC percorre o histórico do pacote, custo O(k), para k eventos associados.
   - Uma consulta CL percorre todos os pacotes do cliente e, para cada pacote, busca o último evento até o tempo consultado, resultando em O(q × t), onde q é o número de pacotes do cliente e t o tamanho médio do histórico.

Considerando N como o total de linhas do arquivo, E o número de eventos, P o número de pacotes e C o número de clientes, temos o custo geral dominado por: $O(N) + \sum consultas$, ou seja, linear no número total de linhas para indexação, mais o custo específico das consultas.

E em relação **a complexidade de espaço**, o espaço total utilizado pelo sistema é proporcional ao número de entidades armazenadas: $O(E + P + C)$, pois mantém o vetor global de eventos (E), o vetor de pacotes com seus históricos (P × tamanho médio do histórico), e a lista de clientes com seus pacotes

(C × pacotes por cliente).

4. Estratégias de Robustez

A robustez de um sistema se refere à sua capacidade de funcionar corretamente e manter a estabilidade mesmo em situações inesperadas ou adversas, como entradas inválidas, falhas de memória ou eventos de execução não previstos. No desenvolvimento deste sistema de simulação de eventos discretos para os Armazéns Hanoi, foram implementadas diversas estratégias de programação defensiva e tolerância a falhas para garantir que o sistema opere de maneira confiável e eficiente.

4.1. Inicialização e Gerenciamento das Estruturas

A inicialização correta das variáveis e das estruturas de dados é fundamental para evitar o uso de memória não inicializada e o acesso a posições inválidas, prevenindo falhas de segmentação (segfaults).

Como foi implementado:

- Arrays e contadores de eventos, pacotes e clientes são inicializados explicitamente antes do uso, garantindo estado consistente.
- Ao criar eventos, pacotes ou clientes, todas as variáveis internas são devidamente setadas, evitando lixo de memória.

4.2. Verificação de Limites e Acessos Seguros

Para garantir segurança no acesso às estruturas estáticas, o sistema implementa checagens cuidadosas dos limites superiores e inferiores dos arrays.

Aplicações no código:

- Antes de inserir um evento no histórico de um pacote, é validado se o contador totalEventosPacote está dentro do limite máximo.
- Na inclusão de pacotes associados a um cliente, é verificado se totalPacotes não ultrapassa o tamanho do array alocado.
- Ao percorrer históricos para consultas PC ou CL, o laço respeita sempre o contador real de elementos armazenados.

4.3. Tratamento e Validação do Arquivo de Entrada

A leitura do arquivo de entrada é cuidadosamente verificada para garantir que os dados processados sejam válidos e estejam no formato esperado.

Detalhes técnicos:

- Antes de iniciar o processamento, o sistema checa se o arquivo foi corretamente aberto.
- O programa termina com uma mensagem informativa se o arquivo não for encontrado ou estiver inacessível.
- As linhas são analisadas e separadas conforme seu tipo (EV, PC ou CL), evitando leituras ou conversões inválidas.

4.4. Garantia de Consistência Temporal

Como o sistema depende de consultas que refletem o último estado conhecido até determinado instante, é essencial manter os eventos em ordem cronológica.

Como é garantido:

- O arquivo de entrada já é fornecido ordenado cronologicamente, mas o sistema mantém a adição de eventos aos arrays de forma sequencial, preservando essa ordem no vetor global e nos históricos dos pacotes.

4.5. Tolerância a Casos Limite nas Consultas

O sistema foi projetado para continuar operando mesmo em consultas que poderiam, de outra forma, causar inconsistências ou travamentos, garantindo estabilidade na execução.

Cenários tratados:

- Se um cliente não possuir pacotes até o instante consultado, a consulta CL retorna apenas a indicação do número de resultados igual a zero, sem comprometer o restante da execução.
- Caso um pacote não possua eventos até o tempo consultado em PC, o histórico retornado será vazio de forma controlada, evitando acesso a posições não inicializadas.

Essas estratégias tornam o sistema mais confiável e preparado para lidar com diferentes cenários, inclusive aqueles com dados escassos ou situações-limite nas consultas. Ao combinar verificações preventivas, tratamento explícito de erros e consistência temporal rigorosa, o modelo garante que as respostas às consultas sejam corretas e que o sistema permaneça estável durante toda a execução.

5. Análise experimental

Esta seção apresenta os experimentos realizados para avaliar o desempenho do sistema de consultas logísticas dos Armazéns Hanoi, com foco no impacto dos principais parâmetros no tempo de execução.

5.1. Plano Experimental

Os testes foram conduzidos variando um parâmetro por vez, mantendo os demais fixos, para isolar o efeito de cada variável sobre o desempenho. Os parâmetros controlados foram:

- **Número de pacotes:** variou-se de 1.000 até 100.000 pacotes.
- **Número de clientes:** variou-se de 100 até 100.000 clientes.
- **Número de armazéns:** variou-se de 5 até 1.000.000, simulando o aumento indireto do número total de eventos no sistema.

Os experimentos foram conduzidos em ambiente Linux Ubuntu, utilizando scripts Bash para geração automática dos arquivos de entrada (.in), alteração dos parâmetros no gerador de workload, compilação e execução do sistema. O tempo total foi medido pelo comando externo time, enquanto o tempo interno de processamento foi registrado pelo próprio programa ao término da execução.

Os parâmetros fixos adotados foram:

- Número de clientes: 100 (exceto nos testes que o variaram).
- Número de pacotes: 10.000 (exceto nos testes que o variaram).
- Número de armazéns: 30 (exceto nos testes que o variaram).

5.2. Procedimento Experimental

1. **Geração dos arquivos de entrada:**
   Scripts Bash controlaram as alterações nos parâmetros e compilaram automaticamente o gerador de workload (genwkl3.c), produzindo arquivos .in para cada combinação testada.
2. **Execução do sistema:**
   Para cada arquivo, o sistema foi executado processando todos os eventos e consultas definidos, simulando a carga total sobre os índices e as estruturas de dados.

3. **Medição dos tempos:**

   O sistema foi projetado para **desmembrar o tempo total** em:

   - **Tempo de leitura:** englobando a carga inicial do arquivo em memória.
   - **Tempo de processamento:** medido a partir da execução efetiva das consultas em memória, após a leitura completa, isolando o custo algorítmico do sistema.

4. **Coleta dos dados:**
   Os resultados foram armazenados em arquivos CSV (resultados_pacotes.csv, resultados_clientes.csv, resultados_armazens.csv).

5. **Geração dos gráficos:**
   Utilizou-se Python com Matplotlib para plotar gráficos comparativos do tempo de processamento em função dos parâmetros.

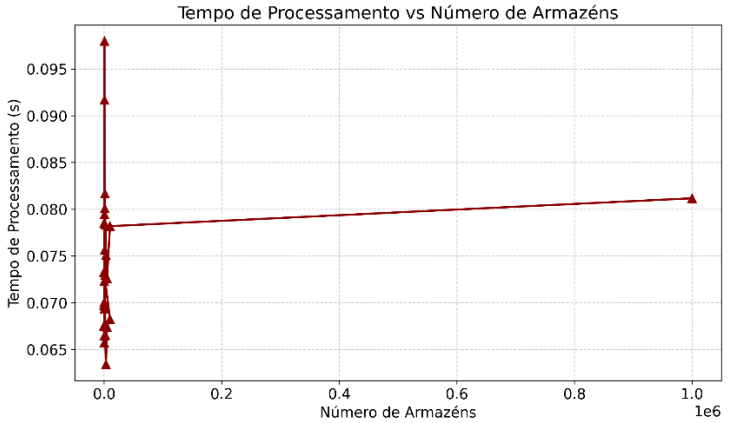
5.3. Resultados e Discussão

Os experimentos revelaram como cada parâmetro influencia o desempenho do sistema.

1. **Variação do Número de Armazéns**

Com 10.000 pacotes e 100 clientes fixos, variou-se o número de armazéns de forma massiva para gerar diferentes quantidades de eventos. O tempo de processamento manteve-se estável, em torno de 0,08 segundos, mesmo para cenários simulando até 1.000.000 de armazéns (com efeitos indiretos sobre o total de eventos), evidenciando que o volume dos eventos em si não foi o fator limitante. Pequenas flutuações ocorreram devido à variação do arquivo gerado e ao custo marginal de manter a lista cronológica.

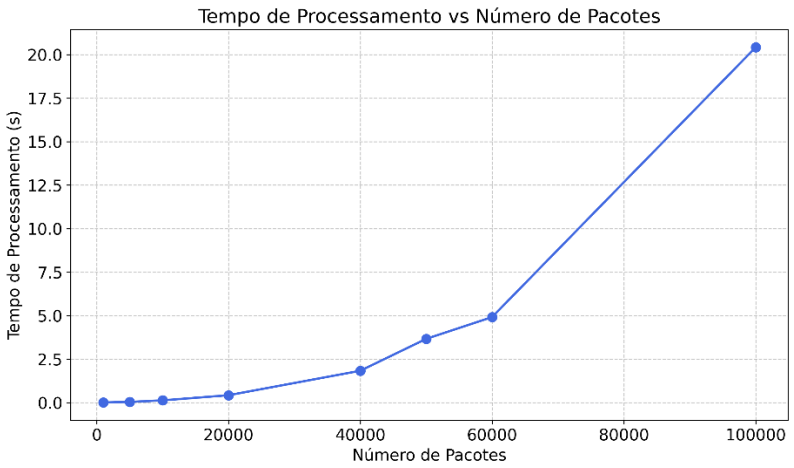
Gráfico 1 – Tempo de Processamento vs. Número de Armazéns



2. **Variação do Número de Pacotes**

Com 30 armazéns e 100 clientes fixos, variou-se o número de pacotes entre 1.000 e 100.000. Observou-se um aumento exponencial no tempo de processamento, atingindo cerca de 20 segundos para 100.000 pacotes. Este resultado está alinhado com a análise de complexidade, já que o volume de pacotes afeta diretamente o tamanho dos históricos mantidos nos índices, e consequentemente, o tempo gasto nas consultas PC e CL.

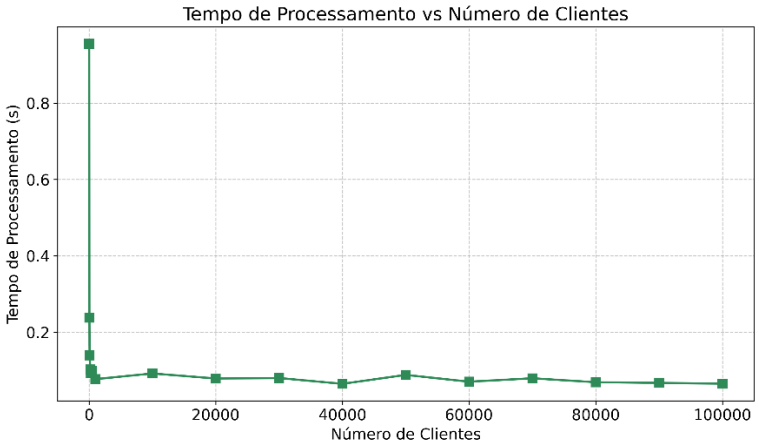
Gráfico 2 – Tempo de Processamento vs. Número de Pacotes



3. Variação do Número de Clientes

Com 30 armazéns e 10.000 pacotes fixos, variou-se o número de clientes de 100 até 100.000. O tempo de processamento apresentou um pico inicial devido à formação e expansão do índice de clientes, mas estabilizou em valores muito baixos, próximos a 0,08 segundos. Isso evidencia que o sistema é altamente eficiente no gerenciamento da lista de pacotes por cliente, mesmo para bases extremamente grandes.

Gráfico 3 – Tempo de Processamento vs. Número de Clientes



5.4. Discussão e Conclusões Experimentais

A comparação entre os resultados experimentais e a análise de complexidade teórica confirmou a consistência do modelo implementado, evidenciando os seguintes pontos técnicos:

- **Número de Pacotes:**
  O crescimento do tempo de processamento com o número de pacotes foi o mais expressivo, pois está diretamente associado ao aumento dos históricos mantidos para cada pacote. Isso se alinha à complexidade **O(k)** das consultas PC, onde k é o tamanho do histórico do pacote, e ao custo cumulativo dos acessos em CL.

- **Número de Clientes:**
  Apesar do número total de clientes ter aumentado significativamente nos testes, o impacto sobre o tempo de processamento foi baixo, como previsto pela complexidade **O(q × t)** das consultas CL, onde q é o número de pacotes do cliente e t o tamanho médio dos históricos. Como os pacotes se distribuíram bem entre os clientes, o crescimento foi contido.
- **Número de Armazéns (indiretamente, o número de eventos):**
  Ao variar o número de armazéns, simulou-se um aumento na topologia e na quantidade total de eventos processados. O tempo de execução manteve-se estável, evidenciando que a varredura sequencial do vetor global de eventos (O(E)) não se tornou gargalo, e confirmando a eficiência do armazenamento cronológico linear.
- **Separação do tempo de leitura e processamento:**
  Foi observada uma diferença clara entre o tempo total (dominado pela leitura do arquivo em disco, I/O) e o tempo interno do processamento em memória, que seguiu precisamente as complexidades esperadas para a manipulação das estruturas já carregadas.

Esses resultados demonstram que o sistema apresenta desempenho consistente e alinhado à análise de complexidade realizada, confirmando que as estratégias adotadas para indexação e processamento cronológico dos eventos são eficazes, mesmo sob diferentes escalas de carga.

6. Conclusões

Este trabalho apresentou o desenvolvimento de um sistema para consultas logísticas no contexto dos Armazéns Hanoi, implementado em C++ e projetado para processar cronologicamente eventos de movimentação e responder a consultas sobre pacotes e clientes.
A construção de índices específicos para pacotes e clientes, associados a um vetor global cronológico de eventos, mostrou-se uma estratégia eficiente para organizar grandes volumes de dados logísticos e garantir respostas rápidas às consultas solicitadas. A análise experimental evidenciou a escalabilidade do sistema em diferentes cenários, confirmando a coerência entre os resultados práticos e a complexidade teórica esperada para as operações implementadas.
Além de cumprir o propósito técnico proposto, o projeto proporcionou aprendizado significativo sobre o gerenciamento manual de estruturas de dados, a importância do controle rigoroso de limites em arrays fixos e a relevância do planejamento experimental para a avaliação de desempenho. Esses aspectos consolidaram conhecimentos essenciais de algoritmos e estruturas.

7. Bibliografia

1. ANISIO MENDES LACERDA; WAGNER MEIRA JÚNIOR. *Slides virtuais da disciplina de Estruturas de Dados – 2024*. Disponibilizado via Moodle. Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte, 2024.

2. GEEKSFORGEEKS. Array Data Structure. GeeksforGeeks, 2023. Disponível em: https://www.geeksforgeeks.org/array-data-structure/. Acesso em: 2 jul. 2025.

3. MEHLHORN, Kurt; SANDERS, Peter. Algorithms and Data Structures: The Basic Toolbox. Springer, 2008. Disponível em: https://link.springer.com/book/10.1007/978-3-540-77978-0. Acesso em: 2 jul. 2025.