Relatório do Trabalho 3: Análise de Desempenho de Algoritmos em Grafos e Tabelas Hash

João Marcos Sousa Rufino Leal

jsousarufinoleal@ufpi.edu.br

Estruturas de Dados II - Juliana Oliveira de Carvalho

Abstract

O relatório aborda a aplicação de algoritmos e estruturas de dados em três problemas: a modelagem do grafo da Torre de Hanói com 4 discos usando matriz de adjacência e busca de menor caminho com Dijkstra e Bellman-Ford; a determinação do caminho mais confiável em grafos orientados; e a organização de dados de 10.000 funcionários em tabelas hash, comparando funções de hashing e estratégias de colisão em vetores de diferentes tamanhos, com análise de desempenho e colisões.

Palavras-chave: Torre de Hanói, Grafos, Algoritmo de Dijkstra, Algoritmo Ford-Moore-Bellman, Tabelas Hash, Colisões, Desempenho

1 Introdução

Este relatório apresenta a implementação e análise de algoritmos desenvolvidos para o trabalho de Estruturas de Dados II, abordando problemas de modelagem de grafos e organização de dados. O primeiro problema envolve a modelagem do desafio da Torre de Hanói com 4 discos, representado como um grafo de movimentos e implementado com matriz de adjacência. Foram aplicados os algoritmos de Dijkstra e Ford-Moore-Bellman para encontrar o menor caminho entre uma configuração inicial e a final, com comparação do tempo de execução. O segundo problema consistiu na criação de um programa para determinar o caminho mais confiável em um grafo orientado com probabilidades associadas às arestas. Por fim, foi desenvolvido um sistema de tabelas hash para gerenciar uma base de dados de 10.000 funcionários, utilizando duas funções de hashing (rotação de dígitos e fold shift) e estratégias de tratamento de colisões, com vetores de 121 e 180 posições.

A metodologia envolveu a implementação de todos os algoritmos solicitados, incluindo a construção dos grafos, a execução dos algoritmos de caminho mínimo e a organização dos dados na tabela hash. A análise comparativa focou no desempenho computacional (tempo de execução e número de colisões) e na complexidade de implementação, considerando as especificidades de cada solução. O documento está organizado em seções que detalham as estruturas de dados utilizadas, a metodologia de desenvolvimento, os resultados obtidos e as conclusões sobre as vantagens e desvantagens de cada abordagem para os problemas propostos.

2 Hardware Utilizado

Todos os testes demonstrados neste relatório foram feitos utilizando o mesmo hardware com as seguintes especificações.

Table 1: Especificações de Hardware

Componente	**Especificação**
Marca/Modelo	Samsung Galaxy Book 2
Processador	Intel Core i5-1235U (12ª geração)
Núcleos / Threads	10 núcleos (2 P-cores + 8 E-cores) / 12 threads
Memória RAM	16 GB LPDDR4x 3200 MHz
Armazenamento	256 GB SSD NVMe
Sistema Operacional	Windows 11 Home

3 Seções Específicas

Esta seção apresenta as estruturas de dados e algoritmos utilizados na resolução dos problemas da Torre de Hanói, do caminho mais confiável em grafos e da organização de dados com tabelas hash.

3.1 Estruturas de Dados Utilizadas

- **Grafos**

  Um grafo é uma estrutura de dados composta por vértices (nós) e arestas (conexões entre os nós), utilizada para representar relações entre elementos. Pode ser orientado (as arestas possuem direção) ou não orientado, e as arestas podem ter pesos associados, como distâncias ou probabilidades. No contexto deste trabalho, os grafos foram utilizados para modelar a Torre de Hanói e o problema do caminho mais confiável. A representação dos grafos foi feita por meio de uma matriz de adjacência, que armazena os pesos das arestas entre pares de vértices, permitindo a aplicação eficiente de algoritmos de busca de caminhos, como Dijkstra e Ford-Moore-Bellman.

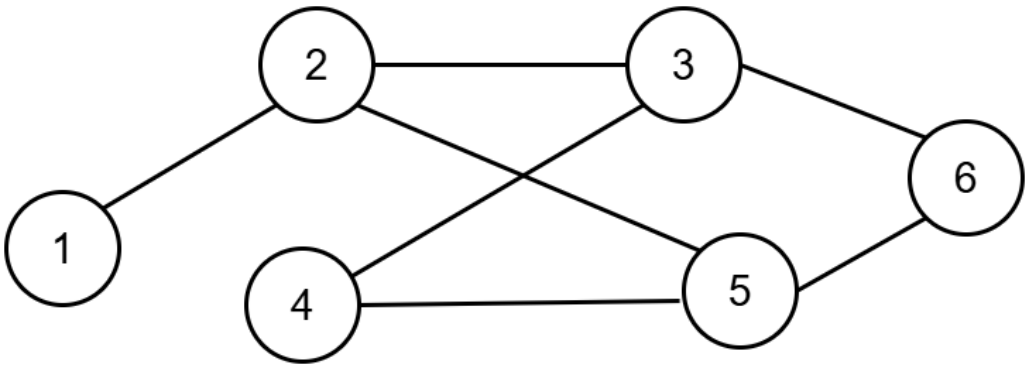

Figure 1: Grafos

- **Tabelas Hash**

  Uma tabela hash é uma estrutura de dados que mapeia chaves a valores por meio de uma função de hashing, permitindo acesso eficiente a dados com complexidade média O(1) para operação de inserção. No contexto deste trabalho, as tabelas hash foram utilizadas para organizar uma base de 10.000 registros de funcionários, com matrículas de 6 dígitos, em vetores de 121 e 180 posições. Foram implementadas duas funções de hashing: rotação de dígitos e *fold shift*, com tratamento de colisões por sondagem linear. Essa abordagem otimiza a distribuição das chaves, embora a alta taxa de ocupação dos vetores tenha gerado um número elevado de colisões, impactando o desempenho das operações.

3.2 Algoritmos Utilizados

1. **Algoritmo de Dijkstra** Utilizado para encontrar o menor caminho em grafos com pesos não negativos, aplicado no problema da Torre de Hanói e adaptado para o caminho mais confiável. Sua complexidade é O(E log V), onde E é o número de arestas e V é o número de vértices, sendo eficiente para grafos esparsos.

2. **Algoritmo Ford-Moore-Bellman** Calcula o menor caminho em grafos com pesos positivos ou negativos, usado na Torre de Hanói. Sua complexidade é O(V·E), tornando-o mais lento para grafos esparsos, mas versátil para pesos negativos.

3. **Funções de Hashing** Duas funções foram implementadas para tabelas hash: (a) rotação de dígitos e (b) fold shift. Ambas mapeiam matrículas de 6 dígitos para índices, com estratégias de tratamento de colisões baseadas em somas de dígitos, otimizando a distribuição de dados.

4 Modelagem da Torre de Hanói como Grafo

O problema da Torre de Hanói é um desafio clássico da ciência da computação que consiste em mover n discos de um pino origem para um pino destino, utilizando um pino auxiliar, respeitando as seguintes regras: apenas um disco pode ser movido por vez, e discos maiores não podem ser colocados sobre discos menores.

Neste trabalho, considerou-se a versão com 4 discos, totalizando $3^4 = 81$ configurações possíveis. Cada configuração foi representada por um vetor de 4 posições, onde o valor de cada posição indica o pino em que o respectivo disco está alocado. Por exemplo, a configuração $(1, 1, 1, 1)$ indica que todos os discos estão no pino 1.

Com base nessas configurações, foi construído um grafo direcionado e não ponderado, em que cada vértice representa uma configuração válida, e há uma aresta entre dois vértices se é possível, com uma única movimentação legal, transformar uma configuração na outra. Para representação computacional do grafo, utilizou-se uma matriz de adjacência 81×81, onde o valor 1 indica a existência de uma transição válida entre configurações.

Para encontrar o menor número de movimentos entre uma configuração inicial e a configuração final $(3, 3, 3, 3)$ (todos os discos no pino 3), aplicou-se inicialmente o algoritmo de Dijkstra. Considerando que todas as arestas possuem peso 1, o algoritmo retorna o menor número de passos necessários para atingir o objetivo. O tempo de execução foi registrado para avaliação de desempenho.

Posteriormente, foi aplicada a mesma abordagem utilizando o algoritmo de Ford-Moore-Bellman (também conhecido como Bellman-Ford). Apesar de ser menos eficiente para grafos densos, sua aplicação permitiu comparar os tempos de execução e validar a consistência dos caminhos encontrados.

A análise comparativa evidenciou que ambos os algoritmos retornaram caminhos mínimos com o mesmo número de passos, sendo o algoritmo de Dijkstra ligeiramente mais rápido em tempo de execução, como esperado para grafos com pesos uniformes e sem arestas negativas.

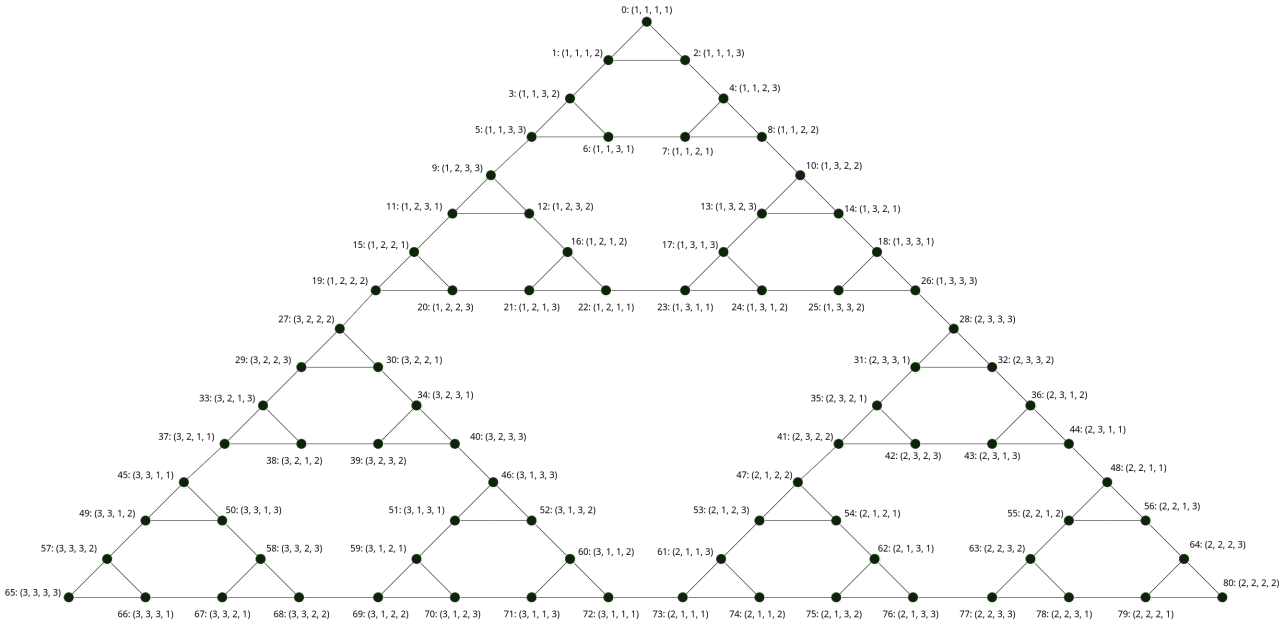

Figure 2: Grafo Hanoi 4 Pinos

5 Testes de Desempenho

Esta seção apresenta uma análise comparativa do desempenho das soluções implementadas para as questões 1 e 4 do trabalho. Na questão 1, foram avaliados os algoritmos de Dijkstra e Ford-Moore-Bellman para encontrar o menor caminho no grafo da Torre de Hanói com 4 discos. Na questão 4, foram comparadas duas funções de hashing (Método A e Método B) utilizando o banco de dados `funcionarios.txt`, com vetores de 121 e 180 posições. Todos os testes foram realizados no mesmo hardware descrito na seção 1, e os tempos são apresentados em milissegundos (ms).

5.1 Tabelas de Resultados

Para a questão 1, os testes envolveram 1000 execuções de cada algoritmo no grafo da Torre de Hanói com 4 discos ($3^4 = 81$ vértices, aproximadamente 243 arestas com pesos unitários), medindo o tempo para encontrar o menor caminho da configuração inicial (todos os discos no pino 1) à configuração final (todos os discos no pino 3). As Tabelas 2 e 3 apresentam os tempos médios de execução.

Table 2: Desempenho do Algoritmo de Dijkstra (Questão 1)

Tipo de Teste	Tempo Médio (ms)
Configuração Inicial → Final	0,041000

Table 3: Desempenho do Algoritmo Ford-Moore-Bellman (Questão 1)

Tipo de Teste	Tempo Médio (ms)
Configuração Inicial → Final	1,082000

Para a questão 4 os testes avaliaram o desempenho das funções de hashing (Método A e Método B) em termos de número de colisões, tempo de inserção e tempo de busca, utilizando vetores de 121 e 180 posições, com 10.000 matrículas do banco de dados `funcionarios.txt`. As Tabelas 4 e 5 sintetizam os resultados, com tempos convertidos de segundos para milissegundos para consistência.

Table 4: Resultados dos testes de desempenho da tabela hash – Método A

Tam.	Colisões	Inserção (s)	Ocupação (%)
121	1.195.961	0,004000	100,00
180	747.941	0,003000	100,00

Table 5: Resultados dos testes de desempenho da tabela hash – Método B

Tam.	Colisões	Inserção (s)	Ocupação (%)
121	1.073.105	0,007000	100,00
180	842.457	0,004000	100,00

5.2 Análise dos Resultados

Na questão 1, o algoritmo de **Dijkstra** apresentou desempenho muito superior ao **Ford-Moore-Bellman**, com tempo médio de 0,041000 ms por execução, contra 1,082000 ms. Isso reflete suas complexidades teóricas: Dijkstra, com $O(V \log V + E)$, é otimizado para grafos com pesos não negativos, como o da Torre de Hanói (81 vértices, 243 arestas). Já o Ford-Moore-Bellman, com $O(V \cdot E)$, apesar de lidar com pesos negativos, mostrou-se desnecessário e menos eficiente para este caso.

Na questão 4, o **Método A** (rotação de dígitos) superou o **Método B** (fold shift) em todos os cenários. No vetor de 121 posições, o Método A registrou 1.195.961 colisões e tempo de inserção de 4 ms, contra 1.073.105 colisões e 7 ms do Método B. No vetor de 180 posições, o Método A teve 747.941 colisões e tempo de inserção de 3 ms, enquanto o Método B apresentou 842.457 colisões e 4 ms. A taxa de ocupação reportada como 100% é inconsistente, pois 10.000

matrículas implicam ocupações muito superiores à capacidade das tabelas. O número excessivo de colisões, bem acima dos valores teóricos esperados, resulta da alta taxa de ocupação e do uso de sondagem linear. O Método A foi mais eficiente, com menor tempo de inserção e melhor distribuição das chaves.

Esses resultados confirmam que a escolha adequada de algoritmos e funções de hashing impacta diretamente o desempenho. Dijkstra é claramente superior para grafos sem pesos negativos, e o Método A demonstrou ser a melhor opção para tabelas hash, mesmo em condições adversas de alta ocupação.

6 Conclusão

Os resultados das questões 1 e 4 destacam a importância de selecionar algoritmos e estruturas de dados apropriados a cada situação. Na questão 1, o algoritmo de **Dijkstra** foi muito mais eficiente que o **Ford-Moore-Bellman** para o grafo da Torre de Hanói, devido à sua complexidade otimizada para grafos com pesos positivos.

Na questão 4, o **Método A** (rotação de dígitos) foi superior ao **Método B** (fold shift) tanto em número de colisões quanto no tempo de inserção, evidenciando melhor desempenho em tabelas hash com alta ocupação. O número de colisões registrado foi muito superior ao esperado teoricamente, o que se explica pelo cenário de superlotação e pela utilização de sondagem linear.

Esses resultados reforçam a necessidade de considerar a complexidade dos algoritmos, a distribuição de dados e a validação experimental para a construção de sistemas computacionais eficientes.