TRABALHO PRÁTICO 3
PROBLEMA DAS ROTAS

Mateus Ryan de Castro Lima - 2021036752
Mateusryan31@ufmg.br

Universidade Federal de Minas Gerais - Belo Horizonte, MG

1. INTRODUÇÃO

Este trabalho prático aborda uma variação do clássico Problema do Caixeiro Viajante, no qual a entrada consiste em um conjunto de cidades (vértices de um grafo), suas conexões diretas (arestas) e os respectivos pesos dessas conexões (custos das arestas). Para resolver o problema, foram implementadas três abordagens distintas: Força Bruta, Programação Dinâmica e Algoritmo Guloso. A abordagem por Força Bruta garante a solução exata, porém com alto custo computacional. A Programação Dinâmica oferece um resultado ótimo com menor tempo de execução em relação à Força Bruta. Já o Algoritmo Guloso constrói uma solução viável de forma eficiente, embora nem sempre garanta o melhor caminho.

2. MODELAGEM

A modelagem do problema foi realizada utilizando grafos ponderados para representar as cidades e estradas. O código pode ser facilmente dividido entre as 3 instâncias iniciais do problema, onde fiz uma classe Grafo para servir de apoio para as três e depois uma classe para cada uma das soluções pedidas.

2.1. Estruturas de Dados

- **Grafo:** Estrutura de Dados base, onde implementei as operações fundamentais para auxiliar em todos os três principais métodos.
- **Vetor (std::vector):** Usados em diversas situações, dentes armazenamento de inteiros, caminhos, distâncias e permutações, booleanos e até mesmo para as matrizes.
- **Mapas (std::map):** Utilizei para armazenas nomes de cidades para índices numéricos com o intuito de facilitar sua manipulação.
- **Pares (std::pair):** Não explicitamente declarado, porém o uso de std::map implica a pares.

- **Matrizes (vector de vector):** Como dito anteriormente, implementada como um vetor de vetores de inteiros em distancias, por exemplo.
- **Bitmasking:** Usei para representar subconjuntos de cidades em algoritmos de Programação Dinâmica.
- **Pilha de chamadas:** Usada implicitamente também na Programação Dinâmica, em funções como tsp e reconstroiCaminho.
- **Heap (Alocação Dinâmica):** Usada para alocar memória implicitamente para vectors e maps.
- **String:** Utilizada para armazenamento de nomes de cidades.

2.2. Principais Algoritmos

- **Força Bruta (Exaustiva)**: Explora literalmente todas as possíveis permutações e as compara, para assim encontrar a menor rota, oque torna o código inviável como veremos na parte de complexidade.
- **Programação Dinâmica com Bitmasking:** Utiliza Programação Dinâmica e bitmasking para armazenar resultados intermediários e evitar recomputações.
- **Algoritmo Guloso (Vizinho Mais Próximo):** Constrói uma solução incrementalmente escolhendo sempre a cidade mais próxima não visitada.
- **Manipulação de Grafos:** Implementa um grafo completo representado por uma matriz de adjacência.

3. SOLUÇÃO

Após um bom resumo feito acima sobre as funcionalidades dos algoritmos utilizados, deixei abaixo um pseudo-código dos mais importantes do meu código no intuito de explicar melhor suas respectivas lógicas.

- **Força Bruta:**

```
Função geraPermutação(caminhoAtual, cidadesRestantes)
        Se todas as cidades foram visitadas:
                Calcular distância total e atualizar menor rota se necessário
        Para cada cidade não visitada:
                Adicionar ao caminho, marcar como visitada e chamar `geraPermutacao`
                Remover do caminho e desmarcar
Função findMenorCaminho:
        Inicializar lista de cidades e chamar `geraPermutacao`
```

Retornar melhor caminho e menor distância.

Em resumo, o Algoritmo de Força Bruta analisa todos os casos possíveis, os armazena e só depois de compara-los ele retorna o ideal, sua resposta é ótima, porém muito cara em tempo e espaço.

- **Programação Dinâmica:**

Função `tsp(mascara, cidadeAtual)`:

Se todas as cidades foram visitadas:

Retornar custo para voltar à cidade inicial

Se já calculado, retornar valor armazenado

Para cada cidade não visitada:

Calcular custo e armazenar menor valor encontrado

Salvar resultado e retornar

Função `findMenorCaminho()`:

Chamar `tsp(1, 0)`, reconstruir rota e retornar melhor caminho e custo

Já a programação dinâmica ela subdivide o problema em problemas menores e, de certa forma, reutiiliza informações e até mesmo código já antes utilizado, com isso se torna mais viável tanto em tempo quanto em espaço.

- **Guloso (Vizinho mais Próximo):**

Função `findMenorCaminho()`:

Começar na cidade inicial, marcar como visitada

Para cada cidade restante:

Escolher a mais próxima e adicionar ao caminho

Retornar caminho e custo total e imprimir resultados

Já o Guloso sempre toma a melhor decisão momentânea, sempre pega o caminho mais barato inicialmente, não tendo garantia de chegar no menor caminho total.

4. ANÁLISE DE COMPLEXIDADE

- **Força Bruta:**

Tempo: $\boldsymbol{O(n!) * O(n) = O(n! * n)}$, por conta das matrizes.
Espaço: $\boldsymbol{O(n)}$**.**

- **Programação Dinâmica:**

Tempo: $\boldsymbol{O(n^2) * O(2^n) = O(n^2 2^n)}$**.**

Espaço: $\boldsymbol{O(n^2) * O(2^n) = O(n^2 2^n)}$**.**

- **Guloso:**

  Tempo: $O(n^2)$.

  Espaço: $O(n)$.

- **Estrutura de Grafo:**

  Tempo: $O(1) + O(1) + O(n^2) = O(n^2)$.

  Espaço: $O(n^2) + O(n) = O(n^2)$.

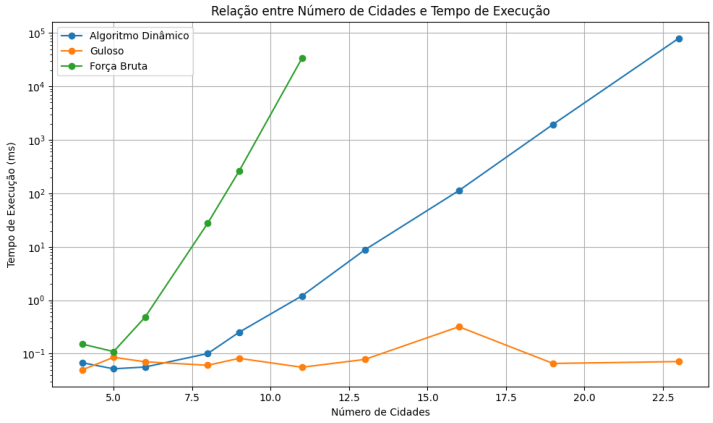

Gráfico 1 – Relação entre o número de cidades e tempo

Aqui podemos observar que Força Bruta só consegue rodar até o testCase06, Programação Dinâmica até o testCase09 e o Algoritmo Guloso é quase linear.

Valor Ótimo	Algoritmo Guloso	Erro (%)
283.0	283.0	0.0
684.0	755.0	10.38
510.0	650.0	27.45
611.0	645.0	5.56
569.0	644.0	13.18
595.0	595.0	0.0
636.0	723.0	13.68
682.0	983.0	44.13
743.0	1179.0	58.68
806.0	1268.0	57.32

Tabela 1 – Erro (%) do Algoritmo Guloso em relação ao Ótimo (Dinâmico)

Apesar de ser quase linear, o Algoritmo Guloso apresenta uma considerável margem de erro.

Número da Entrada	Tempo	Espaço
1.0	0.064262	1.90234
2.0	0.060114	1.83984
3.0	0.065134	1.96484
4.0	0.076746	2.00781
5.0	0.05258	1.95703
6.0	0.066227	1.83984
7.0	0.058401	1.96094
8.0	0.067909	1.96094
9.0	0.119898	1.89844
10.0	0.120339	1.83984

Tabela 2 – Relação entre tempo e Espaço dadas entradas pré-definidas do Algoritmo Guloso

Número da Entrada	Tempo	Espaço
1.0	0.052971	1.83984
2.0	0.049114	1.83984
3.0	0.054894	1.90234
4.0	0.177548	1.96094
5.0	0.222023	1.89844
6.0	0.990408	3.31641
7.0	6.77244	4.40234
8.0	103.377	15.9492
9.0	1966.56	123.191
10.0	77305.5	2179.07

Tabela 3 - Relação entre tempo e Espaço dadas entradas pré-definidas da Programação Dinâmica

Já sobre as tabelas 2 e 3 podemos ver que o Algoritmo Guloso é menos custoso não só em Tempo como em Espaço quando comparado a Programação Dinâmica, que apresar de começar quase linear, a partir de um certo valor ela cresce consideravelmente.

5. CONSIDERAÇÕES FINAIS

Achei o trabalho bem prático, apesar de corrido, uma ótima forma de revisar três dos importantes conteúdos vistos durante o semestre e principalmente de Programação Dinâmica, onde identifiquei algumas dúvidas ao colocar a teoria em prática. Já a Força Bruta e o Algoritmo Guloso foram mais tranquilos.

6. REFERÊNCIAS

- CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. **Algoritmos: Teoria e Prática**. 3. ed. Rio de Janeiro: Elsevier, 2011.
- TARJAN, Robert E. **Algoritmos em Grafos**. Rio de Janeiro: LTC, 1996.
- CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. **Introduction to Algorithms**. 2. ed. Cambridge: The MIT Press, 2001.
- GEEKSFORGEEKS. **Geeks for Geeks: Algoritmos e Estruturas de Dados**. Available at: https://www.geeksforgeeks.org/. Accessed on: 2 feb. 2025.