Projeto e Análise de Algoritmos

O Problema do Caixeiro Viajante

Gabriel Leão

O Problema do Caixeiro Viajante

Um caixeiro viajante precisa visitar uma lista de cidades, passando por cada uma delas exatamente uma vez, e retornar à cidade de origem. O objetivo é minimizar a distância total percorrida (ou o custo, tempo, etc.).

O Problema do Caixeiro Viajante

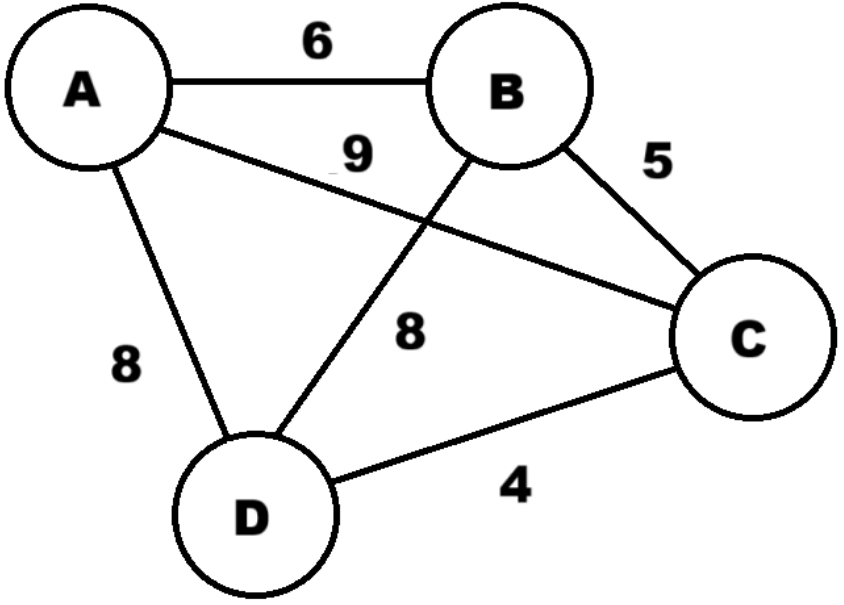
Dado:

1. Um conjunto de n cidades;
2. A distância (ou custo) entre cada par de cidades;

Objetivo:

Encontrar a menor rota possível que passe por todas as cidades exatamente uma vez e retorne ao ponto de partida.

O Problema do Caixeiro Viajante



Força Bruta

```java
import java.util.*;

public class Main {

  static final int[][] dist = {
    // A  B  C  D
    { 0, 6, 9, 8 }, // A
    { 6, 0, 5, 8 }, // B
    { 9, 5, 0, 4 }, // C
    { 8, 8, 4, 0 }  // D
  };

  static final String[] cidades = { "A", "B", "C", "D" };


  public static void main(String[] args) {
  int[] perm = {1, 2, 3}; // Começando sempre por A (índice 0)
  int minCusto = Integer.MAX_VALUE;

  String[] melhoresRotas = new String[6];
  int countMelhores = 0;

  System.out.println("Rotas e seus custos:\n");

  do {
    int custo = 0;
    int atual = 0; // começa em A
```

Input for the program (Optional)

Output:

```
Rotas e seus custos:

A -> B -> C -> D -> A | Custo: 23
A -> B -> D -> C -> A | Custo: 27
A -> C -> B -> D -> A | Custo: 30
A -> C -> D -> B -> A | Custo: 27
A -> D -> B -> C -> A | Custo: 30
A -> D -> C -> B -> A | Custo: 23

Melhor rota(s) com custo 23:
A -> B -> C -> D -> A
A -> D -> C -> B -> A
```

Força Bruta

```
Rotas e seus custos:

A -> B -> C -> D -> A | Custo: 23
A -> B -> D -> C -> A | Custo: 27
A -> C -> B -> D -> A | Custo: 30
A -> C -> D -> B -> A | Custo: 27
A -> D -> B -> C -> A | Custo: 30
A -> D -> C -> B -> A | Custo: 23

Melhor rota(s) com custo 23:
A -> B -> C -> D -> A
A -> D -> C -> B -> A
```

Força Bruta

Número de cidades	Permutações
4	6
5	24
6	120
10	362.880
11	3.628.800
15	87.178.291.200
20	121.645.100.408.832.000

Vizinho mais próximo

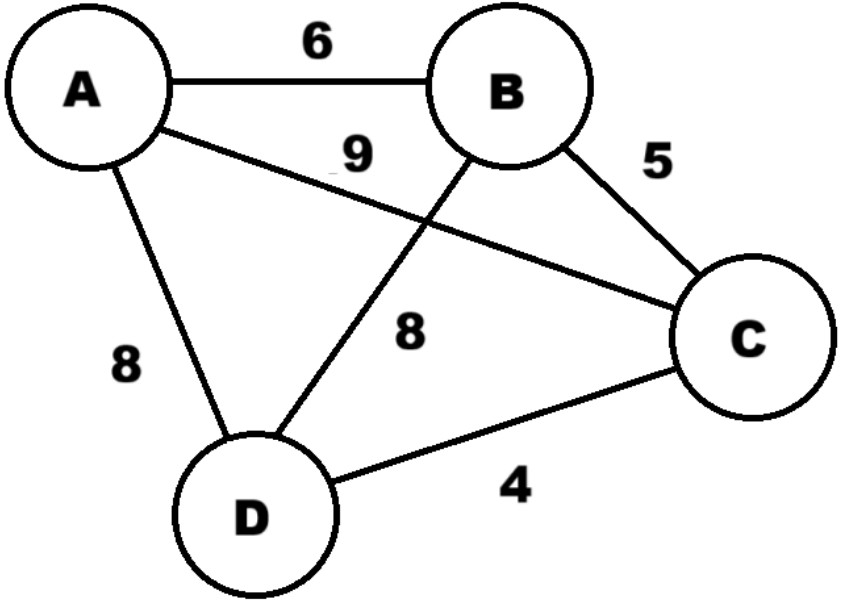
Heurística simples e rápida para resolver o Problema do Caixeiro Viajante. Ela não garante a melhor solução possível, mas pode fornecer uma solução "razoável" rapidamente, especialmente útil em instâncias grandes onde métodos exatos são inviáveis.

Vizinho mais próximo

Funcionamento:

1. Escolhe-se uma cidade inicial arbitrária.
2. A partir da cidade atual, parte-se para a cidade mais próxima (com menor distância) que ainda não foi visitada.
3. Marca-se essa cidade como visitada.
4. Repete-se o processo até que todas as cidades tenham sido visitadas.
5. Retorna-se à cidade inicial para fechar o ciclo.

Vizinho mais próximo



Código em Java

```java
import java.util.*;

public class Main {
  static final int[][] dist = {
        //  A   B   C   D
        { 0, 6, 9, 8 }, // A
        { 6, 0, 5, 8 }, // B
        { 9, 5, 0, 4 }, // C
        { 8, 8, 4, 0 }  // D
    };

    static final String[] cidades = { "A", "B", "C", "D" };
```

```java
public static void main(String[] args) {
  int numCidades = dist.length;
  boolean[] visitado = new boolean[numCidades];

  int atual = 0; // Começa em A
  int custoTotal = 0;
  String rota = cidades[atual];

  visitado[atual] = true;

  for (int i = 1; i < numCidades; i++) {
      int proximo = -1;
      int menorDist = Integer.MAX_VALUE;

      // Busca o vizinho mais próximo ainda não visitado
      for (int j = 0; j < numCidades; j++) {
        if (!visitado[j] && dist[atual][j] < menorDist) {
          menorDist = dist[atual][j];
          proximo = j;
        }
      }

      visitado[proximo] = true;
      custoTotal += menorDist;
      atual = proximo;
      rota += " -> " + cidades[atual];
  }
```

```
    // Retorna para a cidade inicial (A)
    custoTotal += dist[atual][0];
    rota += " -> " + cidades[0];

    System.out.println("Rota encontrada (vizinho mais próximo):");
    System.out.println(rota);
    System.out.println("Custo total: " + custoTotal);
  }
}
```

Output:

```
Rota encontrada (vizinho mais próximo):
A -> B -> C -> D -> A
Custo total: 23
```

Vizinho mais próximo

Vantagens:

1. Muito rápido, mesmo com muitas cidades
2. Fácil de implementar
3. Funciona bem em casos simples ou regulares

Desvantagens:

1. Não garante a melhor solução
2. Pode gerar rotas bem piores que a ideal, especialmente em mapas assimétricos ou com armadilhas locais (chamadas de "ótimos locais")

Vizinho mais próximo

A técnica do Vizinho Mais Próximo é útil quando:

1. Precisa-se de uma solução rápida
2. O número de cidades é grande
3. Aceita-se uma solução boa, mas não perfeita

Comparativo

Técnica	**Força Bruta**	**Vizinho Mais Próximo**
Abordagem	Exata	Heurística (aproximada)
Garante melhor solução?	Sim	Não
Complexidade de tempo	O(n!)	$O(n^2)$
Escalabilidade	Viável para instâncias pequenas	Pode ser usada em instâncias grandes
Velocidade de execução	Muito lenta para n > 10	Muito rápida