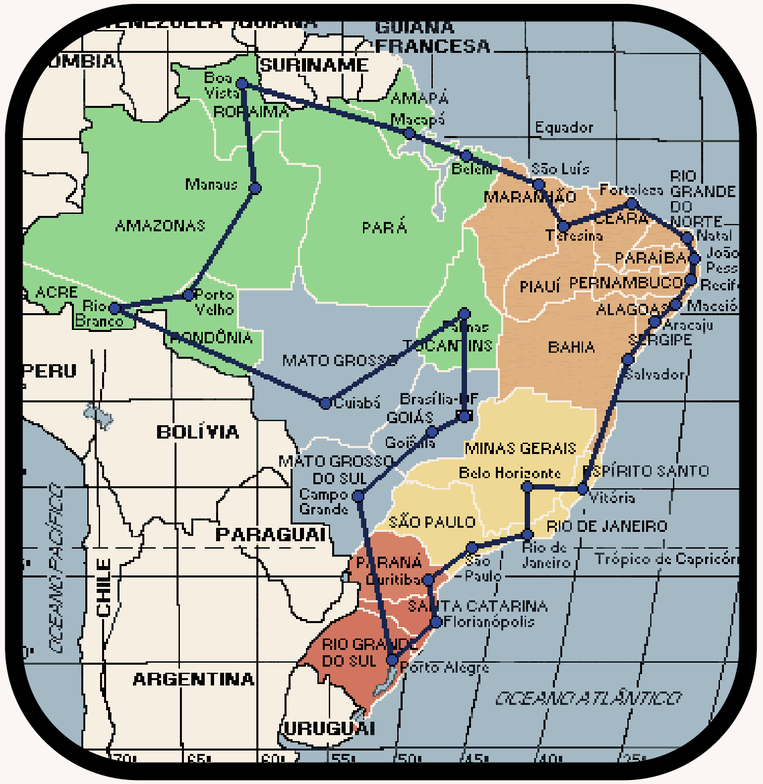
PROBLEMA DO CAIXEIRO VIAJANTE

Gilberto Alexsandro, Guilherme Miranda e Thiago Camara

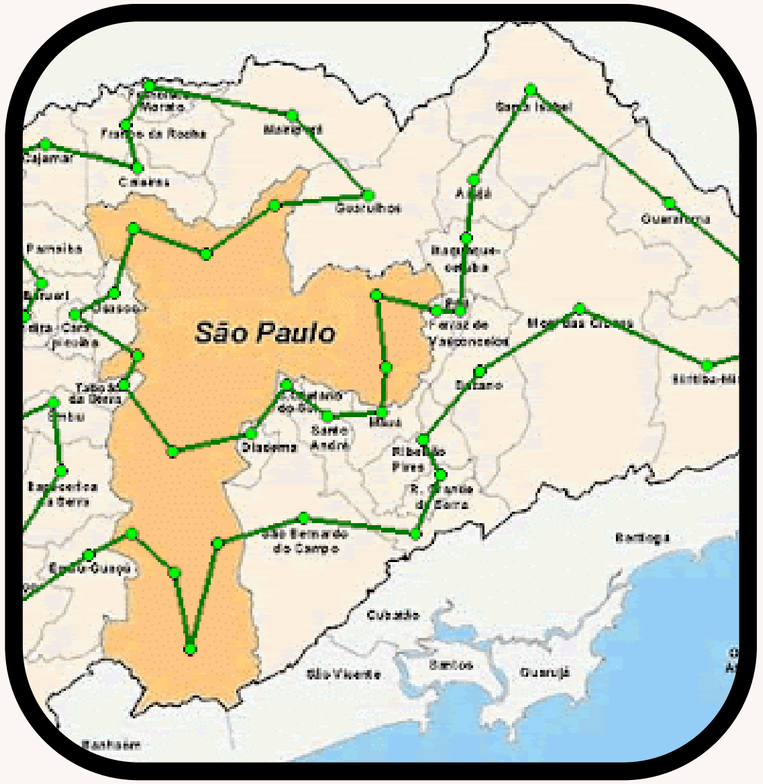
DEFININDO O PROBLEMA

Imagine um vendedor que precisa visitar várias cidades e voltar para o ponto de partida, percorrendo a menor distância possível. O Problema do Caixeiro Viajante consiste em encontrar o caminho mais curto em um grafo completo, passando por todos os vértices exatamente uma vez e retornando ao vértice de origem. A distância entre cada par de vértices é conhecida.



DEFININDO O PROBLEMA

O Problema do Caixeiro Viajante é um problema NP-completo: ele não tem nenhum algoritmo eficiente conhecido. Contudo, adotando-se certas premissas, há algoritmos eficientes que fornecem uma distância total não muito acima da menor possível.



Algoritmo de Força Bruta

```c
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define V 14
#define INF 1000000000 // Valor de custo inviável

// Função para calcular o custo total de um caminho
long long calcularCusto(int path[], long long grafo[V][V]) {
  long long custoTotal = 0;
  for (int i = 0; i < V - 1; i++) {
    custoTotal += grafo[path[i]][path[i + 1]];
  }
  custoTotal += grafo[path[V - 1]][path[0]]; // Retorno à cidade inicial
  return custoTotal;
}

// Função para gerar permutações e encontrar o menor custo
void tsp(int pos, int path[], int visited[], long long grafo[V][V],
         long long *minCost) {
  if (pos == V) {
    long long custoAtual = calcularCusto(path, grafo);
    if (custoAtual < *minCost) {
      *minCost = custoAtual;
    }
    return;
  }

  for (int i = 0; i < V; i++) {
    if (!visited[i]) {
      visited[i] = 1;
      path[pos] = i;
      tsp(pos + 1, path, visited, grafo, minCost);
      visited[i] = 0;
    }
  }
}

int main() { // Definindo um grafo de adjacência estático para 12 cidades
  long long grafo[V][V] = {
      {0, 20, 18, 25, 12, 22, 15, 28, 19, 24, 11, 26, 17, 21},
      {20, 0, 23, 17, 21, 19, 26, 13, 25, 18, 22, 15, 24, 27},
      {18, 23, 0, 20, 16, 24, 11, 27, 22, 19, 25, 17, 21, 26},
      {25, 17, 20, 0, 19, 22, 28, 15, 24, 21, 18, 23, 12, 26},
      {12, 21, 16, 19, 0, 14, 25, 20, 18, 24, 22, 17, 26, 19},
      {22, 19, 24, 22, 14, 0, 17, 26, 21, 20, 18, 25, 13, 24},
      {15, 26, 11, 28, 25, 17, 0, 20, 22, 19, 24, 21, 18, 23},
      {28, 13, 27, 15, 20, 26, 20, 0, 24, 22, 19, 25, 17, 21},
      {19, 25, 22, 24, 18, 21, 22, 24, 0, 20, 17, 26, 15, 23},
      {24, 18, 19, 21, 24, 20, 24, 22, 20, 0, 23, 17, 26, 19},
      {11, 22, 25, 18, 22, 18, 24, 19, 17, 23, 0, 21, 26, 20},
      {26, 15, 17, 23, 26, 25, 21, 25, 26, 17, 21, 0, 19, 24},
      {17, 24, 21, 12, 26, 19, 18, 17, 26, 26, 21, 19, 0, 23},
      {21, 27, 26, 26, 19, 24, 23, 21, 23, 19, 20, 24, 23, 0}};

  int path[V];             // Array para armazenar a sequência das cidades
  int visited[V] = {0};    // Array para controlar as cidades visitadas
  long long minCost = INF; // Inicializando o custo mínimo como INF

  visited[0] = 1; // Começando pela cidade 0
  path[0] = 0;

  struct timespec start_time, end_time;

  clock_gettime(CLOCK_MONOTONIC, &start_time);

  tsp(1, path, visited, grafo, &minCost);

  clock_gettime(CLOCK_MONOTONIC, &end_time);

  double execution_time = (end_time.tv_sec - start_time.tv_sec) +
                          (end_time.tv_nsec - start_time.tv_nsec) / 1e9;

  printf("Custo mínimo encontrado: %lld\n", minCost);
  printf("Tempo de execução: %f segundos\n", execution_time);

  return 0;
}
```

COMPLEXIDADE E CUSTO

O método do força bruta consiste em testar todas as permutações possíveis das cidades e fornece ao final uma solução exata, porém é um método não muito bom para um grande conjunto de cidades.

A complexidade é O(V-1!), pois na função “tsp” do código a variável “v” é chamada recursivamente, onde “v” é o número de cidades. Isso quer dizer que com o aumento no numero de cidades, o tempo de execução cresce exponencialmente.

Além disso, possui uma complexidade temporal de O(V), pois itera sobre as V cidades para calcular o custo total do caminho.

RESULTADOS

5 Cidades

- Custo mínimo encontrado: 102
- Tempo de execução: 0.000003 segundos

8 Cidades

- Custo mínimo encontrado: 104
- Tempo de execução: 0.000169 segundos

12 Cidades

- Custo mínimo encontrado: 164
- Tempo de execução: 2.392850 segundos

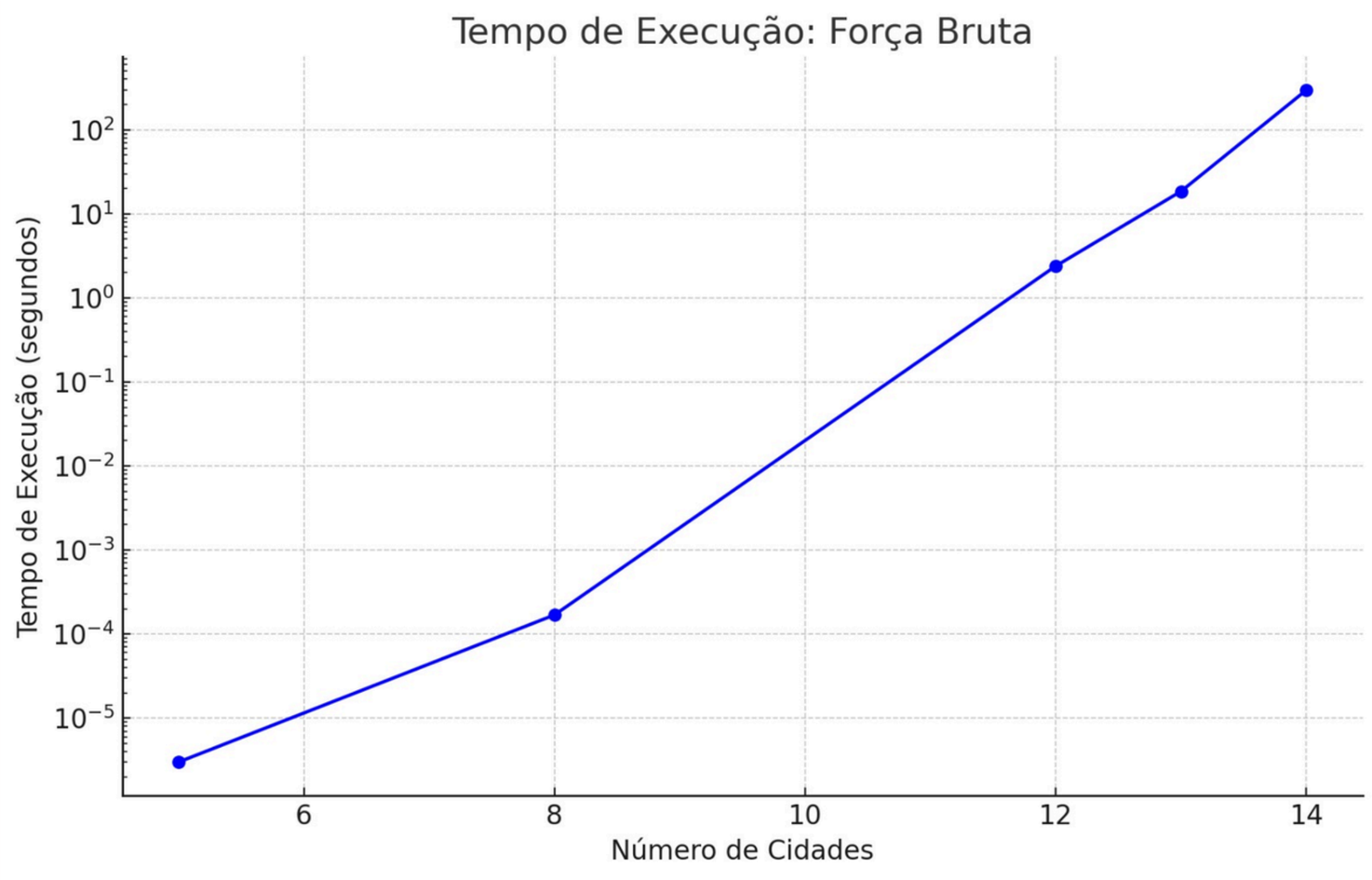
13 Cidades

- Custo mínimo encontrado: 188
- Tempo de execução: 18.539735 segundos

14 Cidades

- Custo mínimo encontrado: 211
- Tempo de execução: 296.402887 segundos

RESULTADOS


Tempo de Execução: Força Bruta
Tempo de Execução (segundos)
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
6
8
10
12
14
Número de Cidades

Algoritmo de Vizinho Mais Próximo

```c
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define V 5
#define INF 1000000000 // Valor de custo inviável

// Função para encontrar o vizinho mais próximo
int vizinhoMaisProximo(int grafo[V][V], int start, int *caminho) {
    int visitado[V] = {0};
    int custoTotal = 0;
    int atual = start;
    visitado[start] = 1;
    caminho[0] = start;

    for (int i = 1; i < V; i++) {
        int proxCidade = -1;
        int menorDist = INF;  // Inicializa com INF para garantir que encontramos um mínimo válido

        for (int j = 0; j < V; j++) {
            // Verifica se a cidade não foi visitada e se o caminho não é inviável (INF)
            if (!visitado[j] && grafo[atual][j] < menorDist && grafo[atual][j] != INF) {
                menorDist = grafo[atual][j];
                proxCidade = j;
            }
        }

        // Se não encontrou um caminho viável, o grafo pode estar desconectado
        if (proxCidade == -1) {
            fprintf(stderr,
                    "Erro: Não foi possível encontrar uma cidade vizinha não visitada a partir da cidade %d.\n",
                    atual);
            exit(EXIT_FAILURE);
        }
        visitado[proxCidade] = 1;
        caminho[i] = proxCidade;
        custoTotal += menorDist;
        atual = proxCidade;
    }
```

```c
    }

    caminho[V] = start;  // Retorno à cidade inicial
    custoTotal += grafo[atual][start];  // Adiciona o custo de voltar à cidade inicial

    return custoTotal;
}
int main() {
    int grafo[V][V] = {
        {0, 10, 20, 30, 25},
        {10, 0, 15, 35, 20},
        {20, 15, 0, 30, 28},
        {30, 35, 30, 0, 22},
        {25, 20, 28, 22, 0}
    };
    // Captura o tempo de início
    clock_t start_time = clock();

    int caminho[V + 1];
    int custoInicial = vizinhoMaisProximo(grafo, 0, caminho);

    // Captura o tempo de término
    clock_t end_time = clock();

    // Calcula o tempo de execução em segundos
    double execution_time = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;

    printf("Custo inicial (Vizinho Mais Próximo): %d\n", custoInicial);
    printf("Caminho inicial: ");
    for (int i = 0; i <= V; i++) {
        printf("%d ", caminho[i]);
    }
    printf("\n");

    // Exibe o tempo de execução
    printf("Tempo de execução: %f segundos\n", execution_time);
    return 0;
}
```

COMPLEXIDADE E CUSTO

O método do vizinho mais próximo (nearest neighbour) consiste em sempre se movimentar para a cidade mais próxima que ainda não foi visitada. E mesmo que não entregue uma solução exata, ele consegue entregar uma resposta aproximada e também pode trabalhar com um maior conjunto de cidades.

A complexidade é O(V^2), onde V é o número de cidades. A complexidade total reflete o crescimento quadrático do número de operações à medida que o número de cidades aumenta. O tempo de execução do algoritmo cresce rapidamente, tornando-o menos eficiente para um grande número de cidades.

RESULTADOS

5 Cidades

- Custo mínimo encontrado: 105
- Caminho inicial: 0, 1, 2, 4, 3, 0
- Tempo de execução: 0.000002 segundos

8 Cidades

- Custo mínimo encontrado: 109
- Caminho inicial: 0, 7, 6, 1, 4, 5, 2, 3, 0
- Tempo de execução: 0.000002 segundos

12 Cidades

- Custo mínimo encontrado: 170
- Caminho inicial: 0, 8, 1, 10, 4, 5, 2, 7, 6, 11, 3, 9, 0
- Tempo de execução: 0.000003 segundos

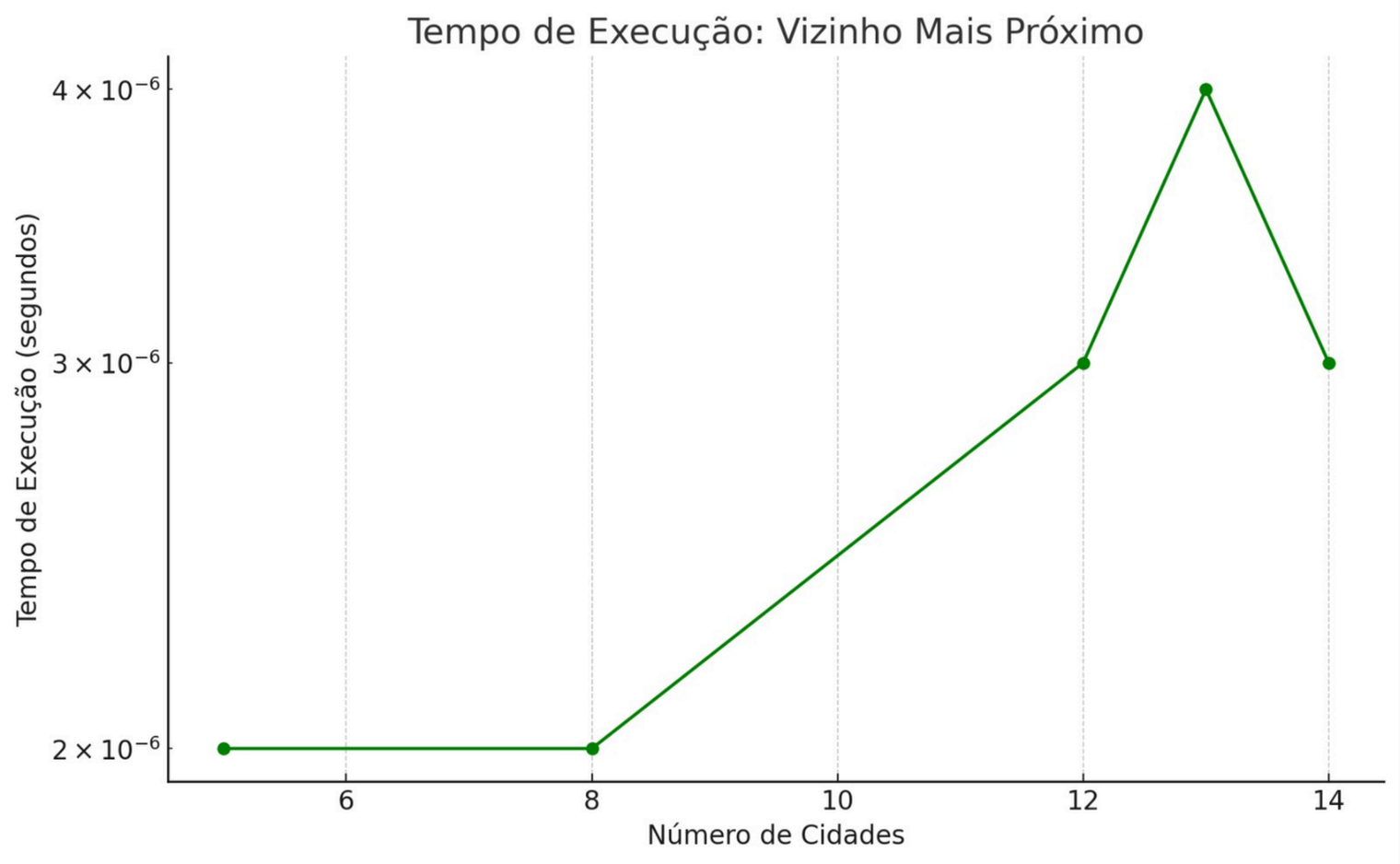
13 Cidades

- Custo mínimo encontrado: 205
- Caminho inicial: 0, 10, 8, 12, 3, 7, 1, 11, 2, 6, 5, 4, 9, 0
- Tempo de execução: 0.000004 segundos

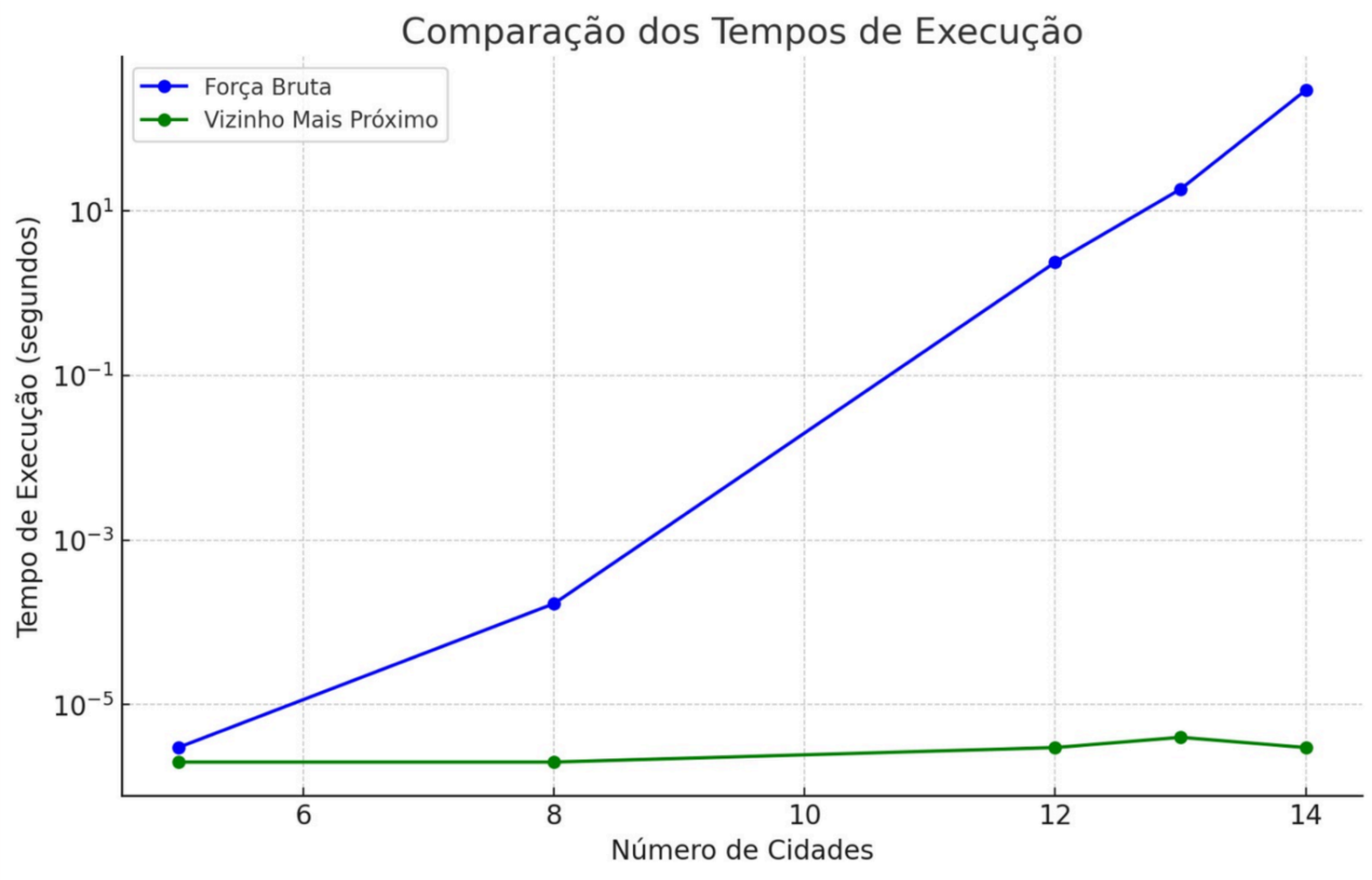
14 Cidades

- Custo mínimo encontrado: 219
- Caminho inicial: 0, 10, 8, 12, 3, 7, 1, 11, 2, 6, 5, 4, 13, 9, 0
- Tempo de execução: 0.000003 segundos

RESULTADOS


Tempo de Execução: Vizinho Mais Próximo
Tempo de Execução (segundos)
4×10^{-6}
3×10^{-6}
2×10^{-6}
6
8
10
12
14
Número de Cidades

RESULTADOS


Comparação dos Tempos de Execução
Força Bruta
Vizinho Mais Próximo
Tempo de Execução (segundos)
10^{1}
10^{-1}
10^{-3}
10^{-5}
6
8
10
12
14
Número de Cidades

OBRIGADO PELA ATENÇÃO!