Aula 06 - Exercício prático Complexidade de problemas

Aluno: Gian Franco Joel Condori Luna

September 24, 2024

Exercices

1 (0,5) Apresente um algoritmo para o problema que pode ser resolvido em tempo polinomial (classe P) que vc pesquisou no exercício prático. Pode usar algoritmo seu/de outros autores, citar a fonte! Se possível mostrar a execução do algoritmo (print do resultado) para um pequeno dataset.

Solução:

Problema do caminho mais curto em grafos

Explicação do Problema

Dado um grafo direcionado com pesos não negativos, o objetivo é encontrar o caminho mais curto de um nó de origem para todos os outros nós. O algoritmo de Dijkstra oferece uma solução eficiente para esse problema.

Algoritmo de Dijkstra

A seguir, apresento um pseudocódigo do algoritmo de Dijkstra:

1. Inicialize todas as distâncias do nó de origem para todos os outros nós como infinito, exceto a distância até a origem, que será 0.
2. Coloque todos os nós em um conjunto de nós não visitados.
3. Selecione o nó não visitado com a menor distância e marque esse nó como visitado.
4. Para cada vizinho do nó atual, calcule a distância tentativa do nó de origem até esse vizinho passando pelo nó atual. Se essa distância for menor que a distância armazenada anteriormente, atualize-a.

5. Repita o processo até que todos os nós tenham sido visitados ou que não haja um nó alcançável.

Código em Python:

```python
# Fonte: ChatGPT
# Python 3.12
import heapq

def dijkstra(grafo, inicio):
    # Inicializar as distancias e a fila de prioridade
    distancias = {vertice: float('inf') for vertice in grafo}
    distancias[inicio] = 0
    fila_prioridade = [(0, inicio)]

    while fila_prioridade:
        distancia_atual, vertice_atual = heapq.heappop(
            fila_prioridade)

        # Ignorar se ja encontramos uma distancia menor
        if distancia_atual > distancias[vertice_atual]:
            continue

        # Relaxamento das arestas
        for vizinho, peso in grafo[vertice_atual].items():
            distancia = distancia_atual + peso

            # Se encontrarmos um caminho mais curto
            if distancia < distancias[vizinho]:
                distancias[vizinho] = distancia
                heapq.heappush(fila_prioridade, (distancia,
                    vizinho))

    return distancias

# Grafo de exemplo
grafo = {
    'A': {'B': 1, 'C': 4},
    'B': {'A': 1, 'C': 2, 'D': 5},
    'C': {'A': 4, 'B': 2, 'D': 1},
    'D': {'B': 5, 'C': 1}
}

# Executar Dijkstra
vertice_inicial = 'A'
distancias = dijkstra(grafo, vertice_inicial)

# Imprimir resultados
print(f"Distancias desde {vertice_inicial}:")
for vertice, distancia in distancias.items():
    print(f"Distancia ate {vertice}: {distancia}")
```

Execução do algoritmo

Para o pequeno conjunto de dados representado pelo grafo de exemplo, obtemos as distâncias mínimas do vértice 'A' para todos os outros vértices:

```
Distancias desde A:
Distancia ate A: 0
Distancia ate B: 1
Distancia ate C: 3
Distancia ate D: 4
```

Fontes Consultadas

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.
- ChatGPT

2 (0,5) Apresente um algoritmo para o problema que pode ser resolvido em tempo não polinomial (classe NP) que vc pesquisou no exercício prático. Pode usar algoritmo seu/de outros autores, citar a fonte! Se possível mostrar a execução do algoritmo (print do resultado) para um pequeno dataset.

Solução:

Problema do Caixeiro Viajante (TSP - Travelling Salesman Problem)

Explicação do Problema

Dado um conjunto de cidades e as distâncias entre cada par de cidades, o objetivo do TSP é encontrar o caminho de menor custo que visita todas as cidades uma vez e retorna à cidade de origem. A solução por força bruta, que tenta todas as possíveis permutações de cidades, tem complexidade fatorial (O(n!)), onde n é o número de cidades.

Algoritmo por Força Bruta

Um algoritmo de força bruta gera todas as permutações possíveis das cidades e calcula o custo de cada caminho. O caminho com o menor custo será a solução.

Código em Python (força bruta)

```python
# Fonte: ChatGPT
# Python: 3.12
from itertools import permutations
```

```
    # Funcao para calcular o custo total de um caminho
    def calcular_custo(caminho, matriz_distancias):
        custo = 0
        for i in range(len(caminho) - 1):
            custo += matriz_distancias[caminho[i]][caminho[i+1]]
        # Adicionar a volta a cidade de origem
        custo += matriz_distancias[caminho[-1]][caminho[0]]
        return custo

    # Algoritmo de forca bruta para resolver o TSP
    def tsp_forca_bruta(matriz_distancias):
        n = len(matriz_distancias)
        cidades = list(range(n))

        # Gerar todas as permutacoes das cidades
        caminhos_possiveis = permutations(cidades)

        # Inicializar o menor custo com infinito
        menor_custo = float('inf')
        melhor_caminho = None

        # Avaliar todos os caminhos possiveis
        for caminho in caminhos_possiveis:
            custo_atual = calcular_custo(caminho, matriz_distancias
                )
            if custo_atual < menor_custo:
                menor_custo = custo_atual
                melhor_caminho = caminho

        return melhor_caminho, menor_custo

    # Exemplo de matriz de distancias entre 4 cidades
    matriz_distancias = [
        [0, 10, 15, 20],
        [10, 0, 35, 25],
        [15, 35, 0, 30],
        [20, 25, 30, 0]
    ]

    # Executar o algoritmo de forca bruta
    melhor_caminho, menor_custo = tsp_forca_bruta(matriz_distancias
        )

    # Imprimir os resultados
    print(f"Melhor caminho: {melhor_caminho}")
    print(f"Menor custo: {menor_custo}")
```

Execução do algoritmo

Para o pequeno conjunto de dados representado por 4 cidades, o algoritmo gera todas as permutações de cidades e encontra o caminho de menor custo. A saída pode ser:

```
    Melhor caminho: (0, 1, 3, 2)
    Menor custo: 80
```

Este resultado significa que o caminho mais curto para visitar todas as cidades, partindo da cidade 0, passando por todas as outras e voltando à cidade de origem, tem um custo total de 80 unidades.

Fontes Consultadas

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.
- ChatGPT