Implementação do Algoritmo de Dijkstra para Determinar a Rota Mais Curta entre Cidades do Piauí

Resumo

Este artigo apresenta a implementação do algoritmo de Dijkstra para a determinação da rota mais curta entre cidades do estado do Piauí. O algoritmo foi desenvolvido utilizando a linguagem Python, com o apoio das bibliotecas `networkx` e `matplotlib` para modelagem e visualização dos grafos. O estudo exemplifica a aplicação prática de estruturas de dados e algoritmos de grafos em problemas reais, como a determinação de rotas eficientes.

1. Introdução

Os algoritmos de grafos são amplamente utilizados em problemas de otimização e busca, sendo aplicáveis em áreas como redes de comunicação, transporte e logística. O algoritmo de Dijkstra, em particular, é uma solução eficaz para encontrar o caminho mais curto entre dois pontos em um grafo, onde as arestas possuem pesos não negativos. Este artigo explora a aplicação do algoritmo de Dijkstra na determinação da menor rota entre cidades do Piauí, utilizando Python para sua implementação.

2. Fundamentação Teórica

2.1 Grafos

Um grafo é uma estrutura de dados composta por vértices (ou nós) conectados por arestas (ou arcos). As arestas podem ter pesos associados que representam custos, distâncias ou qualquer outra métrica relevante. Grafos são representações poderosas em problemas onde é necessário modelar conexões ou relações entre entidades.

2.2 Algoritmo de Dijkstra

Desenvolvido por Edsger Dijkstra em 1956, o algoritmo de Dijkstra resolve o problema do caminho mais curto em grafos ponderados com arestas de peso não negativo. O algoritmo opera por meio de uma busca em largura modificada, priorizando a expansão de nós com menores distâncias acumuladas desde o ponto de origem.

3. Implementação

3.1 Modelagem do Problema

O problema foi modelado utilizando um grafo onde as cidades do Piauí representam os vértices, e as estradas entre elas representam as arestas, com os pesos correspondendo às distâncias entre as cidades.

```
graph = {
    'Santa Cruz do Piauí': {'Oeiras': 70, 'Paquetá': 50},
    'Picos': {'Oeiras': 85, 'Simplicio Mendes': 110, 'Paquetá': 55},
    'Oeiras': {'Santa Cruz do Piauí': 70, 'Picos': 85, 'Floriano': 120, 'Simplicio Mendes'
    'Floriano': {'Oeiras': 120},
    'Simplicio Mendes': {'Oeiras': 50, 'Picos': 110},
    'Paquetá': {'Santa Cruz do Piauí': 50, 'Picos': 55}
}
```

3.2 Algoritmo de Dijkstra

O algoritmo foi implementado para calcular a rota mais curta entre uma cidade de origem (`Santa Cruz do Piauí`) e uma cidade de destino fornecida pelo usuário. A função a seguir exemplifica a lógica do algoritmo:

```
import heapq

def dijkstra(graph, start, end):
    queue = [(0, start)]
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    path = {}

    while queue:
        current_distance, current_node = heapq.heappop(queue)
        if current_node == end:
            break
        for neighbor, weight in graph[current_node].items():
            distance = current_distance + weight
            if distance < distances.get(neighbor, float('inf')):
                distances[neighbor] = distance
                path[neighbor] = current_node
                heapq.heappush(queue, (distance, neighbor))

    route, step = [], end
    while step:
        route.append(step)
        step = path.get(step)
    return distances[end], route[::-1]
```

3.3 Visualização do Grafo

A visualização do grafo foi realizada utilizando a biblioteca `networkx`, com destaque para a rota mais curta determinada pelo algoritmo de Dijkstra:

```
import networkx as nx
import matplotlib.pyplot as plt

def draw_graph(graph, path=None):
    G = nx.Graph([(u, v, {'weight': w}) for u, neighbors in graph.items() for v, w in neigl
    pos = nx.spring_layout(G)
    nx.draw(G, pos, with_labels=True, node_size=500, node_color='skyblue', font_size=15, f
    nx.draw_networkx_edge_labels(G, pos, edge_labels={(u, v): d['weight'] for u, v, d in G
    if path:
        nx.draw_networkx_edges(G, pos, edgelist=list(zip(path, path[1:])), edge_color='r',
    plt.show()
```

3.4 Execução e Resultados

Ao executar o código, o usuário é solicitado a fornecer a cidade de destino, e o algoritmo calcula e exibe a menor distância e o caminho mais curto. Em seguida, o grafo é visualizado com a rota destacada:

```
if __name__ == "__main__":
    end_city = input("Digite a cidade de destino a partir de Santa Cruz do Piauí: ")
    try:
        distance, route = dijkstra(graph, 'Santa Cruz do Piauí', end_city)
        print(f"Menor distância de Santa Cruz do Piauí para {end_city}: {distance}")
        print(f"Caminho mais curto: {' -> '.join(route)}")
        draw_graph(graph, route)
    except KeyError:
        print(f"A cidade '{end_city}' não está no grafo.")
```

4. Discussão

A implementação demonstrou ser eficaz para resolver o problema proposto, mostrando-se aplicável em cenários reais de planejamento de rotas. O uso de bibliotecas como `networkx` e `matplotlib` facilitou tanto a manipulação do grafo quanto a visualização dos resultados, oferecendo uma abordagem prática para o ensino e compreensão de algoritmos de grafos.

5. Conclusão

Este artigo apresentou a implementação do algoritmo de Dijkstra para a determinação da rota mais curta entre cidades do Piauí, destacando a importância dos algoritmos de grafos em problemas de otimização. O código desenvolvido pode

ser expandido para incluir novas cidades ou adaptado para outros contextos que requeiram a análise de caminhos mínimos.

6. Referências

- Dijkstra, E. W. (1959). *A Note on Two Problems in Connexion with Graphs*. Numerische Mathematik, 1, 269-271.
- **Algoritmo de Dijkstra**: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
- **NetworkX Documentation**: https://networkx.github.io/
- **Matplotlib Documentation:** https://matplotlib.org/