Implementação de Grafos e Algoritmo de Dijkstra: Análise e Discussão

Charles E. A. de Faria* João P. S. Medeiros†

Recebido em 19 de julho de 2023, aceito em 19 de julho de 2023.

Resumo

Este trabalho apresenta a implementação da estrutura de dados grafo e do algoritmo de Dijkstra, com análises e explicações sobre seu funcionamento. O objetivo principal é estudar e demonstrar o código de um grafo, além de mostrar como o algoritmo de Dijkstra pode ser aplicado para encontrar o caminho mais curto entre dois vértices em um grafo ponderado.

1 Introdução

Os grafos são estruturas de dados versáteis que permitem representar relações complexas entre objetos ou entidades. Eles são aplicados em diferentes áreas, como redes de computadores, sistemas de transporte, redes sociais e otimização de rotas. O algoritmo de Dijkstra, por sua vez, é uma solução eficiente para encontrar o caminho mais curto em um grafo ponderado. Combinar a implementação da estrutura de dados grafo e o algoritmo de Dijkstra é uma abordagem importante para solucionar problemas de otimização em diversos contextos.

Neste trabalho, exploraremos a implementação da estrutura de dados grafo e do algoritmo de Dijkstra, fornecendo análises e explicações sobre o seu funcionamento. O objetivo principal é estudar e compreender a implementação de um grafo, bem como como o algoritmo de Dijkstra pode ser aplicado para encontrar o caminho mais curto entre dois vértices em um grafo ponderado.

2 Metodologia

Este trabalho consiste na implementação de um grafo utilizando a representação de lista de adjacências. Em seguida, o algoritmo de Dijkstra é implementado para encontrar o caminho mais curto entre os nós do grafo. Os resultados, que indicam as menores distâncias entre os nós, são exibidos na saída do terminal. Todo o código foi desenvolvido em Python.

Após a execução do algoritmo de Dijkstra, os resultados obtidos no experimento serão analisados e discutidos.

3 Fundamentação

Definição 1 (Grafo Direcionado Com Pesos)**.** Um grafo direcionado ponderado é uma estrutura de dados composta por um conjunto de vértice (também chamados de nós) e um conjunto de arestas

*Aluno do Bacharelado em Sistemas de Informaçãoda Universidade Federal do Rio Grande do Norte. (e-mail: charles.araujo.701@ufrn.edu.br)

†Professor do Departamento de Computação e Tecnologiada Universidade Federal do Rio Grande do Norte. Coordenador do Laboratório de Elementos do Processamento da Informação. (e-mail: jpsm1985@gmail.com)

direcionadas que conectam os vértices. Cada aresta é associada a um peso ou valor numérico, que representa a relação ou o custo entre os vértices conectados. □

Definição 2 (Algoritmo de Dijkstra)**.** A ideia central do algoritmo de Dijkstra é explorar gradualmente os vértices de um grafo ponderado a partir de uma origem específica, atualizando os caminhos mais curtos conhecidos até cada vértice alcançado. O algoritmo mantém uma lista de vértices não visitados e atribui a cada vértice um valor de distância, que representa a menor distância conhecida entre a origem e o vértice atual.

□

Algoritmo 1 (Algoritmo de Dijkstra)**.** Determinando a menor distância entre os vértices de um grafo utilizando o algoritmo de Dijkstra.

algoritmo dijkstra(L, $origem$)
{Lista de adjacência L de um grafo direcionado $G = \langle N, E \rangle$.}
para cada $vertice \in L.vertices$ **faça**
    $dist[vertice] \leftarrow$ INFINITY
    $prev[vertice] \leftarrow$ UNDEFINED
    adicionar v **a** Q
fim para
$dist[origem] \leftarrow 0$
enquanto Q **não estiver vazio faça**
    $u \leftarrow vertice$ **em** Q **com a menor** $dist[u]$
    remover u **de** Q
    para cada vizinho de $u \in Q$ **faça**
        $alt \leftarrow dist[u] + L.Edges(u, v)$
        se $alt < dist[vizinho]$ **então**
            $dist[v] \leftarrow alt$
            $prev[v] \leftarrow u$
        fim se
    fim para
fim enquanto
retorne dist[ ], prev[ ]

□

4 Experimentos

4.1 Grafo utilizado

O grafo utilizado para a realização do experimento do algoritmo de Dijkstra pode ser visualizado na Figura 1. Podemos visualizar graficamente o mesmo grafo pela Figura 2.

Os vértices 9, 12, 13, 14 e 17 não apresentam ligações com os outros vértices, dessa forma atuando como "obstáculos" ou dificultadores para certos trajetos.

4.2 Implementação do grafo

Segue na Figura 3 o código em Python do grafo utilizando a implementação da lista de adjacência.

4.3 Implementação do algoritmo de Dijkstra

Na Figura 4 podemos visualizar a implementação do algoritmo de Dijkstra.

```
1 -> 6 (weight: 71) -> 2 (weight: 33)
2 -> 1 (weight: 45) -> 7 (weight: 42) -> 3 (weight: 65)
3 -> 2 (weight: 100) -> 8 (weight: 80) -> 4 (weight: 17)
4 -> 3 (weight: 91) -> 5 (weight: 22)
5 -> 4 (weight: 90) -> 10 (weight: 24)
6 -> 1 (weight: 50) -> 11 (weight: 89) -> 7 (weight: 23)
7 -> 2 (weight: 71) -> 6 (weight: 81) -> 8 (weight: 58)
8 -> 3 (weight: 43) -> 7 (weight: 28)
9 ->
10 -> 5 (weight: 29) -> 15 (weight: 48)
11 -> 6 (weight: 13) -> 16 (weight: 52)
12 ->
13 ->
14 ->
15 -> 10 (weight: 20) -> 20 (weight: 52)
16 -> 11 (weight: 75) -> 21 (weight: 50)
17 ->
18 -> 23 (weight: 46) -> 19 (weight: 35)
19 -> 18 (weight: 73) -> 24 (weight: 16) -> 20 (weight: 2)
20 -> 15 (weight: 0) -> 19 (weight: 17) -> 25 (weight: 13)
21 -> 16 (weight: 16) -> 22 (weight: 24)
22 -> 21 (weight: 4) -> 23 (weight: 19)
23 -> 18 (weight: 21) -> 22 (weight: 43) -> 24 (weight: 78)
24 -> 19 (weight: 4) -> 23 (weight: 58) -> 25 (weight: 36)
25 -> 20 (weight: 63) -> 24 (weight: 39)
```

Figura 1: Grafo utilizado para testes.

4.4 Resultados

4.4.1 Sobre o algoritmo de Dijkstra

O algoritmo inicia a partir do vértice de origem que é passado para a função. A partir desse nó, o algoritmo consegue traçar o caminho mais curto para todos os outros nós do grafo. Dentro da função, utilizamos uma lista que guarda as distâncias acumuladas entre a origem e cada outro vértice.

A cada iteração, o algoritmo seleciona o nó com a menor distância acumulada ainda não visitado. O algoritmo compara a distância acumulada atual com a distância que seria obtida ao passar por uma aresta e, se a distância atual for maior, ela é atualizada para a nova distância menor. O processo é repetido até que o vértice de destino seja alcançado.

Ao final da execução, pode-se recuperar o menor caminho entre o nó de origem e qualquer outro vértice do grafo, além de obter a distância acumulada mínima para cada vértice.

4.4.2 Executando código

Na Figura 5, temos a visualização do resultado obtido ao executar o algoritmo de Dijkstra para o grafo da Figura 1.

Essa saída nos fornece uma visualização da trajetória percorrida no grafo da origem até o nó de número 25, imprimindo na tela a soma da distância mínima entre os nós conectados direcionalmente.

Ao visualizar os valores acumulados das distâncias mínimas, podemos perceber que o algoritmo chegou ao seu destino final seguindo o seguinte caminho:

$$\textbf{Shortest path} = 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 10 \rightarrow 15 \rightarrow 20 \rightarrow 25.$$

O algoritmo também tentou seguir outro caminho possível:

$$\textbf{Alternative path} = 1 \rightarrow 6 \rightarrow 11 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 23 \rightarrow 18 \rightarrow 19 \rightarrow 24 \rightarrow 25$$

.

No entanto, como podemos ver na Figura 5, mesmo percorrendo as arestas com menor peso, essa trajetória é mais longa do que a primeira solução mostrada anteriormente. Ao chegar ao nó 24 por meio dessa outra possibilidade, a distância mínima já supera a distância mínima percorrida pelo outro caminho.

5 Conclusão

Este trabalho proporcionou uma visão geral sobre a implementação da estrutura de dados grafo e do algoritmo de Dijkstra. Ao longo do texto, foram explorados os conceitos fundamentais desses tópicos e fornecidas algumas análises de seu funcionamento.

O algoritmo de Dijkstra, mostrou-se extremamente útil para encontrar o caminho mais curto entre dois vértices em um grafo ponderado. Levando em consideração os pesos das arestas, ele nos permite determinar o caminho mais eficiente em termos de distância ou custo, abrindo possibilidades para a resolução de diversos problemas do mundo real.

Portanto, este trabalho serviu como uma introdução sólida ao estudo de grafos e ao algoritmo de Dijkstra.

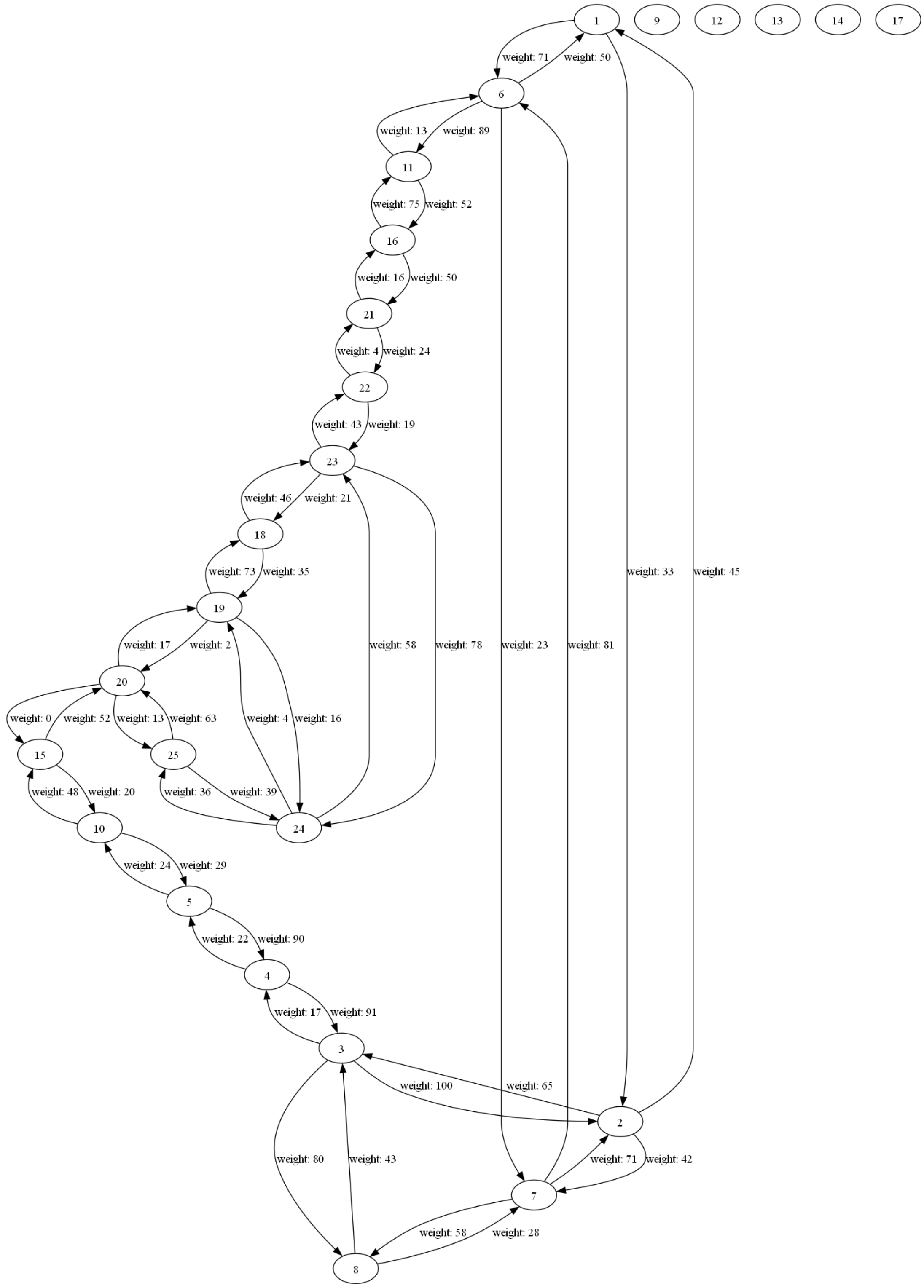

Figura 2: Visualização gráfica do grafo.

```
class Graph:
    def __init__(self):
        self.adjacency_list = {}

    def add_node(self, node):
        if node not in self.adjacency_list:
            self.adjacency_list[node] = {'neighbors': []}

    def add_edge(self, node1, node2, weight):
        if node1 in self.adjacency_list and node2 in self.adjacency_list:
            self.adjacency_list[node1]['neighbors'].append((node2, weight))
```

Figura 3: Exemplo de implementação do grafo.

```
def dijkstra(self, source):
        dist = {}
        prev = {}
        Q = set()

        for node in self.adjacency_list:
            dist[node] = float('inf')
            prev[node] = None

            Q.add(node)

        dist[source] = 0

        while Q:
            current = min(Q, key=lambda node: dist[node])
            Q.remove(current)

            for neighbor, weight in self.adjacency_list[current]['neighbors']:
                alt = dist[current] + weight

                if alt < dist[neighbor]:
                    dist[neighbor] = alt
                    prev[neighbor] = current

        return dist, prev
```

Figura 4: Exemplo de implementação do algoritmo de Dijkstra.

```
Distances:
Node 1: Min Distance = 0, Previous = None
Node 2: Min Distance = 33, Previous = 1
Node 3: Min Distance = 98, Previous = 2
Node 4: Min Distance = 115, Previous = 3
Node 5: Min Distance = 137, Previous = 4
Node 6: Min Distance = 71, Previous = 1
Node 7: Min Distance = 75, Previous = 2
Node 8: Min Distance = 133, Previous = 7
Node 9: Min Distance = inf, Previous = None
Node 10: Min Distance = 161, Previous = 5
Node 11: Min Distance = 160, Previous = 6
Node 12: Min Distance = inf, Previous = None
Node 13: Min Distance = inf, Previous = None
Node 14: Min Distance = inf, Previous = None
Node 15: Min Distance = 209, Previous = 10
Node 16: Min Distance = 212, Previous = 11
Node 17: Min Distance = inf, Previous = None
Node 18: Min Distance = 326, Previous = 23
Node 19: Min Distance = 278, Previous = 20
Node 20: Min Distance = 261, Previous = 15
Node 21: Min Distance = 262, Previous = 16
Node 22: Min Distance = 286, Previous = 21
Node 23: Min Distance = 305, Previous = 22
Node 24: Min Distance = 294, Previous = 19
Node 25: Min Distance = 274, Previous = 20
```

Figura 5: Visualização do resultado do algoritmo de Dijkstra.