P9. Estructuras de Datos en Python III:
Grafos

Alumna: Daniela Mendez Ramirez

Número de Cuenta: 258331-9

Aplicaciones Móviles

Profesor: Omar Vázquez González

Fecha de Entrega: 28 de enero de 2025

Abstract

Este documento presenta la implementación del algoritmo de Dijkstra en Python, aplicado a tres grafos de prueba. Se describe la estructura del código, la representación de los grafos mediante diccionarios de adyacencia y los resultados obtenidos al ejecutar el algoritmo en distintos escenarios.

Introducción

El problema del camino más corto es fundamental en la teoría de grafos y tiene múltiples aplicaciones en redes de comunicación, planificación de rutas y optimización. El algoritmo de Dijkstra, propuesto por Edsger W. Dijkstra en 1956, es una solución eficiente para encontrar la ruta más corta desde un nodo de origen a todos los demás en un grafo ponderado dirigido sin pesos negativos.

Descripción del Algoritmo de Dijkstra

El algoritmo de Dijkstra emplea una cola de prioridad para seleccionar el nodo con la menor distancia acumulada, expandiendo iterativamente los caminos hasta que se determinan las rutas más cortas a todos los nodos alcanzables. Su complejidad es de $O((V + E) \log V)$ utilizando un montículo binario (heap). El algoritmo funciona de la siguiente manera:

1. Se inicializa una estructura de datos que almacena las distancias mínimas desde el nodo origen a cada nodo, estableciendo infinito para todos los nodos excepto el de inicio.
2. Se utiliza una cola de prioridad para gestionar los nodos pendientes de exploración.
3. En cada iteración, se extrae el nodo con la distancia más corta y se actualizan las distancias de sus vecinos si se encuentra un camino más corto.
4. Se repite este proceso hasta que se hayan visitado todos los nodos o no queden nodos accesibles.

La implementación del algoritmo se realizará en Python y se probará con tres grafos distintos, correspondientes a los grafos IV, V y VII del conjunto de prueba.

```python
import heapq

def dijkstra(graph, start):
    # Diccionario de distancias inicializado con infinito
    distances = {node: float('inf') for node in graph}
    distances[start] = 0

    # Cola de prioridad con el nodo de inicio
    pq = [(0, start)]  # (distancia, nodo)

    while pq:
        current_distance, current_node = heapq.heappop(pq)

        if current_distance > distances[current_node]:
            continue  # Si ya encontramos un camino más corto, ignoramos

        for neighbor, weight in graph[current_node].items():
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(pq, (distance, neighbor))

    return distances

# Representación de los grafos como diccionarios de adyacencia

graph_IV = {
    1: {2: 3, 3: 1},
    2: {4: 3},
    3: {2: 1, 5: 2},
    4: {},
    5: {7: 3},
    7: {6: 1},
    6: {}
}

graph_V = {
    1: {2: 1, 5: 4},
    2: {3: 3},
    3: {4: 2},
    4: {},
    5: {3: 5, 6: 7},
    6: {}
}
```

```python
graph_VII = {
    'A': {'B': 9, 'C': 2},
    'B': {},
    'C': {'D': 3, 'E': 6},
    'D': {},
    'E': {'F': 3},
    'F': {}
}

# Pruebas con los grafos
print("Distancias desde el nodo 1 en el grafo IV:", dijkstra(graph_IV, 1))
print("Distancias desde el nodo 1 en el grafo V:", dijkstra(graph_V, 1))
print("Distancias desde el nodo A en el grafo VII:", dijkstra(graph_VII, 'A'))
```

```
PS Clase_17> python3 .\djisktrak.py
Distancias desde el nodo 1 en el grafo IV: {1: 0, 2: 2, 3: 1, 4: 5, 5: 3, 7: 6, 6: 7}
Distancias desde el nodo 1 en el grafo V: {1: 0, 2: 1, 3: 4, 4: 6, 5: 4, 6: 11}
Distancias desde el nodo A en el grafo VII: {'A': 0, 'B': 9, 'C': 2, 'D': 5, 'E': 8, 'F': 11}
PS Clase_17>
```

Dados los siguientes grafos: