Network Delay Time

Aastha Paudel

July 12, 2024

Problem Statement

You are given a network of n nodes labeled from 1 to n. You are also given times, a list of travel times as directed edges times[i] = (u, v, w), where u is the source node, v is the target node, and w is the time it takes for a signal to travel from source to target. This problem is related to finding the shortest path and delay in a network.

Examples

Example 1

- **Input**: $n = 4$, `times = [[2,1,1],[2,3,1],[3,4,1]]`, `K = 2`
- **Output**: 2

Example 2

- **Input**: $n = 4$, `times = [[2,1,1],[2,3,1],[3,4,1]]`, `K = 1`
- **Output**: -1

Algorithm and Approach

Approach

To solve this problem, we can use Dijkstra's algorithm, which is suitable for finding the shortest path in a graph with non-negative weights.

Approach

To find the minimum time required for a signal to travel from a source node K to all other nodes in the network, we use Dijkstra's algorithm:

1. **Initialization**: Initialize an array `dist` of size $n + 1$ to store the minimum time from K to each node. Initialize `dist`$[K] = 0$ and all other entries to ∞.
2. **Priority Queue (Min-Heap)**: Use a priority queue to keep track of nodes to be explored, starting with the source node K with a distance of 0.
3. **Dijkstra's Algorithm Steps**:
   - Dequeue the node with the smallest distance from the priority queue.
   - For each neighboring node v of the current node u, if traveling to v via u offers a shorter distance than currently known, update `dist`$[v]$ and enqueue v with the updated distance.
   - Continue until all reachable nodes are processed or the priority queue is empty.
4. **Result**: After processing all nodes, if any node still has distance ∞, it means it's unreachable from K; otherwise, return the maximum value in `dist` array excluding `dist`$[K]$.

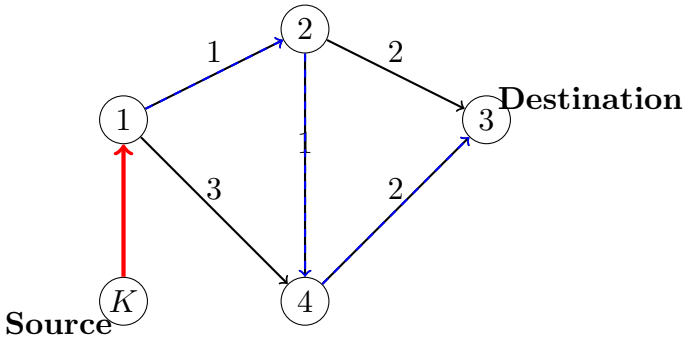
Algorithm

Input: Number of nodes n, list of times `times`, source node K
Output: Minimum time for a signal to reach all other nodes from K, or -1 if not all nodes are reachable

Function `FindMinTime(`*n, times, K*`)`**:**
- Initialize `dist` array of size $n + 1$ with ∞;
- Initialize `dist[K]` $= 0$;
- Initialize a priority queue `pq` to store tuples (time, node);
- Enqueue (0, K) into `pq`;
- **while** *`pq` is not empty* **do**
  - Dequeue (time, u) from `pq`;
  - **if** *time ¿ `dist[u]`* **then**
    - **continue**;
  - **end**
  - **for** *each (u, v, w) in `times`* **do**
    - **if** *`dist[u]` + w ¡ `dist[v]`* **then**
      - `dist`[v] = `dist`[u] + w;
      - Enqueue (`dist`[v], v) into `pq`;
    - **end**
  - **end**
- **end**
- **if** *any `dist[v]` == ∞* **then**
  - **return**-1;
- **end**
- **return** maximum value in `dist` array excluding `dist`[K];

Algorithm 1: Network Delay Time using Dijkstra's Algorithm

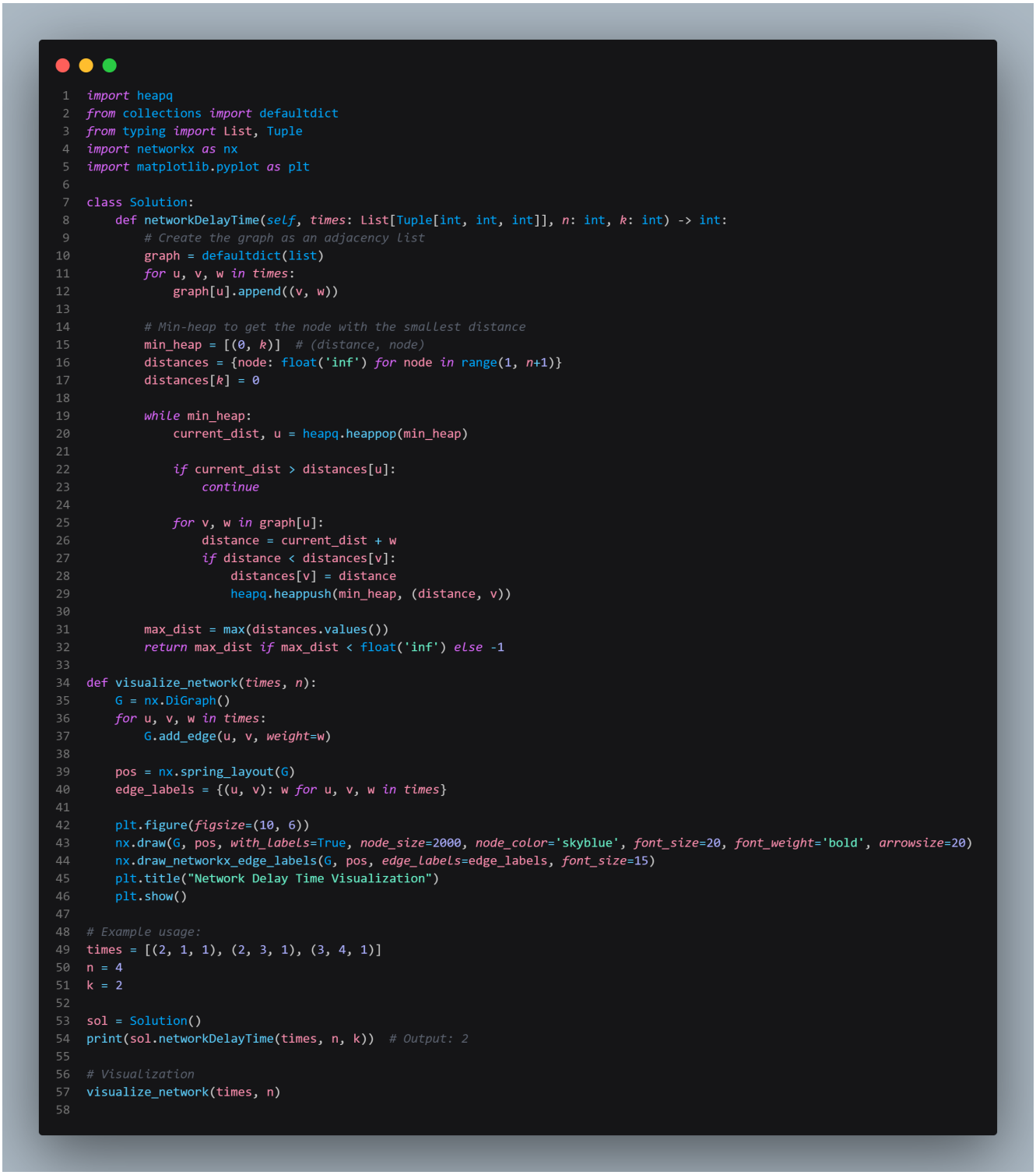
Visualization



Iteration Table for Example 1

Iteration	State of `dist`	PQ Contents	Popped	Updated Distances
Initial	[inf, 0, inf, inf, inf]	(0, 2)	-	-
1	[inf, 0, 1, inf, inf]	(0, 1), (1, 3)	(0, 2)	(1, 1), (1, 3)
2	[inf, 0, 1, 2, inf]	(1, 3), (2, 4)	(1, 1)	(1, 3), (2, 4)
3	[inf, 0, 1, 2, 3]	(2, 4)	(1, 3)	(2, 4)

Solution Code



```python
import heapq
from collections import defaultdict
from typing import List, Tuple
import networkx as nx
import matplotlib.pyplot as plt

class Solution:
    def networkDelayTime(self, times: List[Tuple[int, int, int]], n: int, k: int) -> int:
        # Create the graph as an adjacency list
        graph = defaultdict(list)
        for u, v, w in times:
            graph[u].append((v, w))

        # Min-heap to get the node with the smallest distance
        min_heap = [(0, k)]  # (distance, node)
        distances = {node: float('inf') for node in range(1, n+1)}
        distances[k] = 0

        while min_heap:
            current_dist, u = heapq.heappop(min_heap)

            if current_dist > distances[u]:
                continue

            for v, w in graph[u]:
                distance = current_dist + w
                if distance < distances[v]:
                    distances[v] = distance
                    heapq.heappush(min_heap, (distance, v))

        max_dist = max(distances.values())
        return max_dist if max_dist < float('inf') else -1

def visualize_network(times, n):
    G = nx.DiGraph()
    for u, v, w in times:
        G.add_edge(u, v, weight=w)

    pos = nx.spring_layout(G)
    edge_labels = {(u, v): w for u, v, w in times}

    plt.figure(figsize=(10, 6))
    nx.draw(G, pos, with_labels=True, node_size=2000, node_color='skyblue', font_size=20, font_weight='bold', arrowsize=20)
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=15)
    plt.title("Network Delay Time Visualization")
    plt.show()

# Example usage:
times = [(2, 1, 1), (2, 3, 1), (3, 4, 1)]
n = 4
k = 2

sol = Solution()
print(sol.networkDelayTime(times, n, k))  # Output: 2

# Visualization
visualize_network(times, n)
```

Figure 1: network delay using Dijkstra's Algorithm

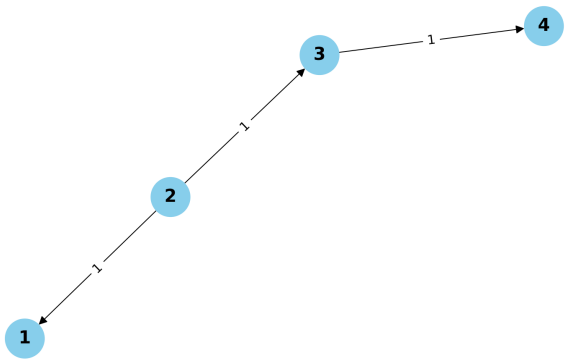


Figure 2: Visiualize