COMP S265F Design and Analysis of Algorithms
Lab 9: Dijkstra's Algorithm

In this lab, we will apply the Dijkstra's algorithm to solve a LeetCode problem "743. Network Delay Time": https://leetcode.com/problems/network-delay-time/

1. Network delay problem

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_i`,` v_i`,` w_i`)`, where u_i is the source node, v_i is the target node, and w_i is the time it takes for a signal to travel from source to target.

We will send a signal from a given node k. Return the time it takes for all the n nodes to receive the signal. If it is impossible for all the n nodes to receive the signal, return -1.

```
class Solution:
    def networkDelayTime(self, times: List[List[int]], n: int, k: int) -> int:
```

The problem has given the following examples and constraints:

- **Example 1.**
  Input: `times = [[2,1,1],[2,3,1],[3,4,1]], n = 4, k = 2`
  Output: `2`
- **Example 2.**
  Input: `times = [[1,2,1]], n = 2, k = 1`
  Output: `1`
- **Example 3.**
  Input: `times = [[1,2,1]], n = 2, k = 2`
  Output: `-1`

Constraints:

- `1 <= k <= n <= 100`
- `1 <= times.length <= 6000`
- `times[i].length == 3`
- `1 <=` u_i`,` v_i `<= n`
- u_i `!=` v_i
- `0 <=` w_i `<= 100`
- All the pairs (u_i, v_i) are unique (i.e., no multiple edges).

2. Problem formulation

The network is a weighted directed graph, where the direction of the edge is the direction of the signal, and the weight is the travel time of the signal. As the nodes are labeled from 1 to n, we can reduce all labels by 1 to make them in $V = \{0, 1, 2, \ldots, n-1\}$.

The problem is a single-source all-destinations shortest path problem from node k (vertex $k-1$). The output is the maximum shortest path distance over all vertices, or -1 if it is ∞. We can apply the Dijkstra's algorithm, which builds a shortest path subtree P by adding vertices one by one, and maintains a list `d` with the following invariant:

- for any vertex $i \in P$, `d[`i`]`$= \delta(0, i)$;
- for any vertex $i \notin P$, `d[`i`]` equals the distance of the minimum outgoing edge of P that is incident to i.

The Dijkstra's algorithm iterates $|V|$ times to build the shortest path subtree P:

Step 1. Find the vertex $u \notin P$ with the smallest `d[`u`]`.

Step 2. For every neighbor v of vertex u, if $v \notin P$, relax the edge (u, v).

To implement Step 1 efficiently, one way is to use a min-heap. Implementing Fibonacci heap in Python is challenging. Yet the `heapq`'s min-heap does not support update of items and deletion of a non-minimum item efficiently, which are necessary to the Dijkstra's algorithm.

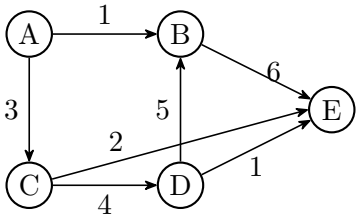
We use the following strategy as workaround, which is practically efficient:

- The min-heap initially contains tuples (`d[`u`]`, u) for all vertices u.
- When performing the relax operation, if `d[`u`]` is decreased, we add a new tuple (`d[`u`]`, u) in the min-heap without deleting the existing tuples for vertex u. As the new tuple has a smaller `d[`u`]`, it will always be popped out first.
- When vertex u is added to P, we do not delete its tuples in the min-heap. Therefore, when popping out a tuple (`d[`u`]`, u), we need to ignore the tuple if $u \in P$.

Your task. Implement the Dijkstra's algorithm to solve the network delay problem. Note that `float("inf")` can be used for infinitely large value in Python.

3. Exercises

Question 1. Given the following weighted directed graph G:



Apply the Dijkstra's algorithm on G using vertex A as the source.

(a) Show the d values of all vertices after each iteration in the Dijkstra's algorithm.

(b) Draw the shortest path tree obtained.

Question 2. Suppose that we are given a weighted directed graph $G = (V, E)$ in which edges leaving the source vertex s may have negative weights, all other edge weights are non-negative, and there are no negative-weight cycles. Argue that the Dijkstra's algorithm correctly finds shortest paths from s in this graph.

(Hint: You need to refer to the proof of correctness of the Dijkstra's algorithm in Unit 5 Slides 20-29.)