Assignment 5 (Dijkstra's Shortest Path)

(60 marks)

Problem Statement:

As a final addition to our software package, we would like to provide routing services to vehicles on the road network. The city road network can be seen as a weighted directed graph quite naturally - every intersection is a vertex, and a road from an intersection to another is an edge from the vertex associated with the first intersection to the vertex associated with the second one. The weights on the edges could correspond to various different parameters based on the scenario. In our scenario, the weight of the edges indicates the length of the road.

Given a weighted directed graph $G = (V, E)$ and two vertices $s, t \in V$ as input, we would like to compute the shortest path from s to t. We will use the classic Dijkstra's algorithm to find the length of the shortest path, and the shortest path itself. To implement this algorithm, we shall build a min-heap to serve as a priority queue.

Input Format:

- The first line will give you the number of vertices n.
- Each of the next lines look like one of the following:
  - `N` $u\ v_1\ w_1\ v_2\ w_2\ v_3\ w_3 \cdots v_k$

    This line means that the edges $(u, v_1), (u, v_2), \ldots, (u, v_k)$ exist in the directed graph with weights $w_1, w_2, \ldots, w_k \in \mathbb{N}$ respectively. The numbers N,u, and all v_i, w_i will fit inside the `int` data type.
  - `D` $s\ t$

    where $s, t \in V$.
- Every line ends with `\n`. End of input is indicated by EOF character.

Output:

First line of input: No output. Create an adjacency list for n vertices numbered $0, 1, \ldots, n-1$.

For each line that reads `N` $u\ v_1\ w_1\ v_2\ w_2\ v_3\ w_3 \cdots v_k$, no output.

If input line read was `D` $s\ t$, then:

- Output the length of the shortest path from s to t followed by the vertices along a shortest path from s to t. Use space as delimiter. End the line with `\n`.
- If no path exists from s to t, output `-1` followed by `\n`.

Implementation Rules:

- Create an adjacency list for the input directed graph.
  - Each vertex will be the head of a linked list containing its neighbours.
  - The heads of all these linked list will be an array. See figure.
  - For each vertex, store and maintain its index in the priority queue.
- Implement a priority queue using a min-heap. The min-heap should be implemented using an array.

- In the priority queue, break ties using the vertex number: If there are two vertices with the same priority, then the vertex with the smaller vertex number gets a higher priority.
- Use $<$ in checking if an edge can be relaxed in Dijkstra's algorithm. If you use $\leq$, your program's output might not match the test cases we use.
- Give your priority queue access the vertex's nodes in the adjacency list. You could either maintain a pointer to the vertex node (as shown figure), or simply the index of the vertex in the adjacency head list.

Remarks (not mandatory):

- When breaking ties in the min-heap, you don't really have to change the key (priority). Keep the same key and make the smaller vertex number is the parent of the larger vertex number. This way, Extract-min will correctly break ties without any extra effort.
- The figure shows an example input, the adjacency list for it, and the initial min-priority queue using a heap implemented as an array. Observe that the heads of linked lists maintain their indices in the heap, and the heap maintains pointers to the head nodes, thus allowing for access in both directions. You could alternatively just maintain the index of the vertex in the priority queue rather than an explicit pointer to the head node.

4
N 0 2 4 3 8
N 1 0 4
N 2 3 3 1 1 2 1
N 3 0 2