Assignment 4 (Breadth First Search)

(60 marks)

Problem Statement:

Our software to monitor road traffic has been deployed at all intersections in the city. We now gather information about all vehicles travelling inside our city. In the future, we would like to run routing algorithms on the road network.

The city hall has asked us to help them understand the reach of emergency services. Emergency vehicles can travel along roads ignoring any direction requirements. So although the road network is a directed graph, the access of emergency vehicles needs to be computed on the *underlying undirected graph* defined as follows:

The underlying undirected graph $G' = (V, E')$ of a directed graph $G = (V, E)$ is defined as follows: $\forall (u, v) \in E$, both (u, v) and (v, u) are in E'. In other words, an undirected graph is also a directed graph where, if there is an edge, then there is an edge in both directions.

We would like to find all the vertices on the road network that emergency vehicles can reach starting from a given vertex. We shall do this using a *breadth first search*.

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` $u\, v_1\ v_2\ v_3 \cdots v_k$
    - `B` v.
- End of input is indicated by EOF character.

Every line ends with `\n`. The quantities `n` and `v` will fit inside the `int` data type.

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` $u\ v_1\ v_2\ v_3\ \cdots v_k$, no output. This line means that the edges $(u, v_1), (u, v_2), \ldots, (u, v_k)$ exist in the directed graph.

If input line read was `B` v , then output the vertices visited in a breadth first search starting from vertex v in the order visited.

Implementation Rules:

- For the input graph read, create an adjacency list for the underlying undirected graph:
    - Each vertex will be the head of a linked list containing its neigbours.
    - The heads of all these linked list will be an array. See figure.
- Implement a queue by using either a linked list with head and tail pointers, or a doubly linked list.

- When inserting the neighbours of a vertex into the queue, you should insert them in ascending order of their vertex number. For eg: If the neighbours of a vertex are vertices 13, 1, 5, then you should enqueue them in the order 1, 5, 13. Hence your BFS should also explore neighbours of any particular vertex in ascending order of vertex number.

Remarks (not mandatory)

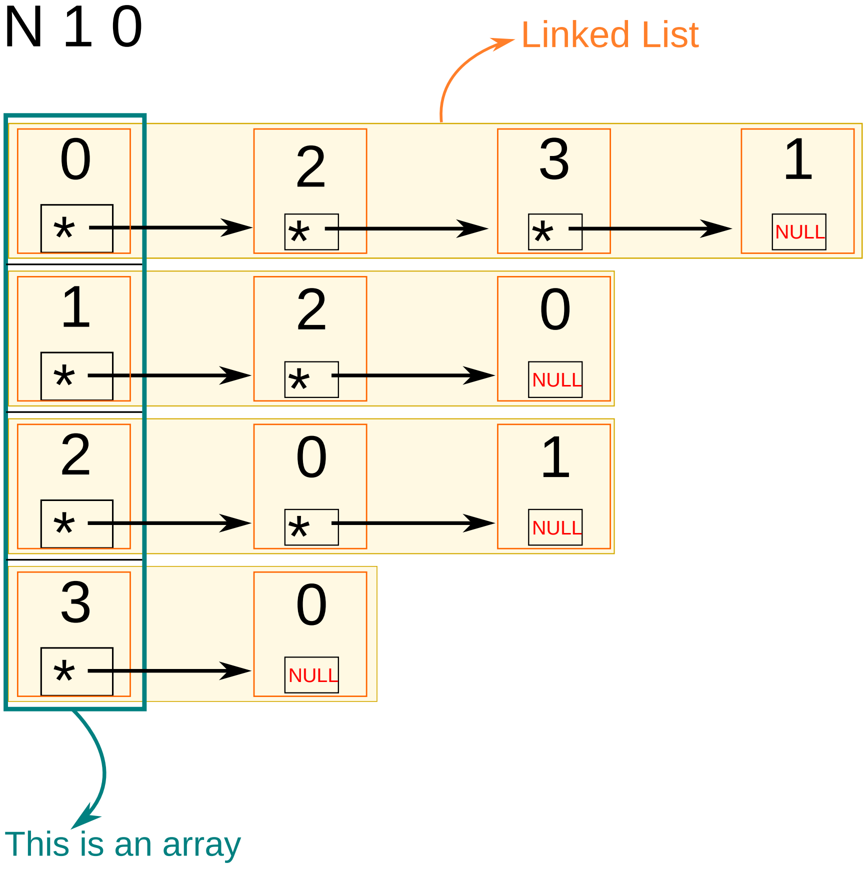
- Keep in mind that we would like to implement Dijkstra's algorithm in the near future. So write your program in a way that will allow you to extend it easily to weighted directed graphs.
- You could maintain an adjacency list with neighbours sorted during insertion itself, or you could maintain the adjacency list in its usual form, and sort the neighbours just before inserting them into your queue.
- The figure shows an example input directed graph, and the adjacency list of the corresponding undirected graph without any sorting.

4

N 0 2 3

N 2 1 2

N 1 0


Linked List
0
*
2
*
3
*
1
NULL
1
*
2
*
0
NULL
2
*
0
*
1
NULL
3
*
0
NULL
This is an array