Algorithm Design

IIT Goa
Deadline:18-03-23 (23:00pm)

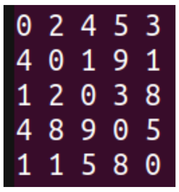
(Lab - 07)

Question

– You may form groups of two each and stick to the submission guidelines as before.

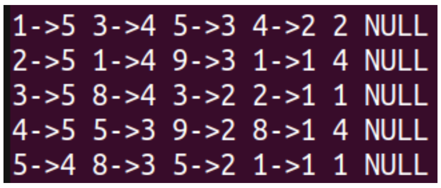
In this assignment, you are supposed to read the adjacency matrix of a graph from the input file provided with. The first line contains the number of vertices n and the next n lines correspond to the rows in the adjacency matrix (you may assume the vertices are labelled 1 to n). You are provided with a wrapper program which reads the name of the file as a command line argument and loads the details to the adjacency matrix. You are supposed to implement the following member functions in the class **graph** (*refer the cpp file provided with*).

1. **printAdjMatrix():** Prints the contents of the **weighted** adjacency matrix the following form (the vertices are labelled from 1 to 5).



```
0 2 4 5 3
4 0 1 9 1
1 2 0 3 8
4 8 9 0 5
1 1 5 8 0
```

2. **countEdge():** Counts the edges in the graph, stores it in the variable *edgeCount* and returns the same.
3. **loadAdjList():**Loads the associated adjacency list data structure using the adjacency matrix.
4. **printAdjList():**Prints the adjacency list in the template below ($->$ *neughbor weight*). The i^{th} row begins with vertex i followed by the list of it's neighbors together with the corresponding edge weights. Recall that adjacency list is an array of linked lists.



```
1->5 3->4 5->3 4->2 2 NULL
2->5 1->4 9->3 1->1 4 NULL
3->5 8->4 3->2 2->1 1 NULL
4->5 5->3 9->2 8->1 4 NULL
5->4 8->3 5->2 1->1 1 NULL
```

5. **runFW():**Populates the necessary data structures (refer *graph* class) required to reconstruct the shortest paths between all the vertices using Floyd-Warshall algorithm.
6. **runDijkstra():**Fetches the super *source* vertex read from the user, run Dijkstra's algorithm and populates the necessary data structures (refer *graph* class) required to reconstruct the shortest paths from *source* to all the remaining vertices (***special credit to implementations using binary heaps*).

7. **printDijkstraPathTo():**Prints the shortest path together with it's weight to an intended target vertex.
8. **printFWPathBetween():**Prints the shortest path together with it's weight between an intended pair of vertices.
9. Write two menu driven functions **testDijkstra()** and **testFW()** that would let the user repetitively test both the algorithms.

1. To end with, write an independent program that would read a file name *File.txt* and the number of vertices n from the user. Furthermore, the program should read a value p between 0 and 1 and generate a random graph having each edge exists with probability p and writes adjacency matrix of the graph to the file *File.txt* in the same format as before.

*******END*******