National Institute of Technology, Calicut
Department of Computer Science and Engineering
CS2094 – Data Structures Lab
Assignment 5 Advanced

Submission deadline (on or before):
30th March 2016, 10:00:00 PM

Naming Conventions for submission:
Submit a single ZIP (.zip) file (do not submit in any other archived formats like .rar or .tar.gz). The name of this file must be ASSG<Number>A_<ROLLNO>_<FIRSTNAME>.zip
(For example: ASSG5A_BxxyyyyCS_LAXMAN.zip). DO NOT add any other files (like temporary files, input files, etc.) except your source code, into the zip archive.
The source codes must be named as ASSG<Number>A_<ROLLNO>_<FIRST NAME>_<PROGRAM-NUMBER>.<extension> (For example: ASSG5A_BxxyyyyCS_ LAXMAN_1.c)

Assignment Questions
General Instructions for all the questions:

- Invalid input should be detected and suitable error messages should be generated.
- Sample inputs are just indicative.

GRAPH ALGORITHMS

Data structure conventions:

In a graph with **n** vertices, the vertices are labeled from **0** to **n-1**. Use adjacency lists to store the graphs, with the vertices sorted in ascending order. The adjacency list of each node is a singly linked list that contains its adjacent nodes sorted in ascending order from left to right. The nodes in this list contain two fields, namely, the label of the adjacent node and the weight of the edge, if provided. Unless specified otherwise, the adjacency lists must be processed iteratively from left to right.

1. **SHORTEST PATH**

   Write a program that implements Dijkstra's algorithm for computing shortest paths in a directed graph with positive edge weights. For a graph with **n** vertices and **m** edges, your program must run in O((m+n) log n) time, by using a binary min-heap to maintain the distances to the vertices in the graph. Then, the extract_min and decrease_key operations can be used to determine the next node and to update distances, respectively, in O(log n) time per operation.

   Input/output format:

   The first line of the input contains a positive integer **n**, the number of vertices in the graph, in the range 1 to 1000.

   The subsequent **n** lines contain the labels of the nodes adjacent to the respective nodes, sorted in ascending order from left to right. If a node has no adjacent nodes, then the line corresponding to its adjacency list will be empty.

   The subsequent **n** lines contain the weights of the edges corresponding to the adjacency list. The edge weights are positive real numbers in the range [0, 10000]. If a node has no adjacent nodes, then the line corresponding to its adjacent edge weights will be empty.

   The rest of the input consists of multiple lines, each one containing a four-letter string followed by zero, one or two integers. The integers, if given, will be in the range **0** to **n-1**.

- The string “apsp” will be followed by a single integer, the label of the source vertex. Output the shortest path distance from the source vertex to all the **n** vertices in the graph, sorted in the order of their labels, in a space separated format. Print “INF” for nodes that are unreachable from the source vertex.
- The string “sssp” will be followed by two integers, respectively, labels of the source and destination nodes. Output the shortest path from the source node to the destination node, if such a path exists. Print “UNREACHABLE”, otherwise.
- The string “stop” means terminate the program.

The output, if any, of each command should be printed on a separate line.

Sample Input

```
9
1 4
5
3
6

2 7 8
2
4
5 7
2 20
3
7
5

1 6 4
0
2
2 1
apsp 0
apsp 2
sssp 2 8
sssp 0 6
sssp 0 7
sssp 5 6
ssss 8 7
stop
```

Sample Output

```
0 2 6 13 12 5 18 10 9
INF INF 0 7 INF INF 12 INF INF
UNREACHABLE
18
10
13
1
```

2. Write a program that implements the Floyd-Warshall algorithm for computing all pairs shortest paths in an undirected graph with positive or negative edge weights, but with no negative weight cycles.

Input format:

The first line of the input contains a positive integer **n**, the number of vertices in the graph, in the range 1 to 1000.

The subsequent **n** lines contain the labels of the nodes adjacent to the respective nodes, sorted in ascending order from left to right. If a node has no adjacent nodes, then the line corresponding to its adjacency list will be empty.

The subsequent **n** lines contain the weights of the edges corresponding to the adjacency list. The edge weights are real numbers in the range [-10000, 10000]. If a node has no adjacent nodes, then the line corresponding to its adjacent edge weights will be empty.

Output format:

The output consists of **n** lines, each containing data in **n** space separated columns. The value in the **ith** row and **jth** column is the shortest distance from the vertex with label **i-1** to the vertex with label **j-1**. If there exists no path between any two nodes, print "INF" as their shortest distance.

Sample Input

```
9
1 4
0 5
3 5 6
2 6
0 7
1 2 7 8
2 3
4 5 8
5 7
2 20
2 3
7 1 0
7 5
20 2
3 1 6 2
0 5
2 6 1
2 1
```

Sample output

```
0 2 6 11 10 5 6 8 7
2 0 4 9 8 3 4 6 5
6 4 0 5 6 1 0 4 3
11 9 5 0 11 6 5 9 8
10 8 6 11 0 5 6 2 3
5 3 1 6 5 0 1 3 2
6 4 0 5 6 1 0 4 3
8 6 4 9 2 3 4 0 1
7 5 3 8 3 2 3 1 0
```

3. **MINIMUM SPANNING TREE**

Write a program that computes the minimum spanning tree of a connected undirected graph using Kruskal's algorithm. Use the disjoint set data structure, implemented using rooted forests with the ranked union and path compression heuristics applied, to build the minimum spanning tree. Every disjoint set in the data structure would correspond to a single connected component in the tree and adding an edge between two connected components is equivalent to computing the union of the sets corresponding to them.

Input format:

The first line of the input contains a positive integer **n**, the number of vertices in the graph, in the range 1 to 1000.

The subsequent **n** lines contain the labels of the nodes adjacent to the respective nodes, sorted in ascending order from left to right.

The subsequent **n** lines contain the weights of the edges corresponding to the adjacency list. The edge weights are real numbers in the range [-10000, 10000]. Further, no two edges have the same weight.

Output format:

The output consists of **n** lines, with the **ith** line containing the adjacency list of the vertex with label **i-1**, sorted in ascending order.

Sample Input

```
12
8 9
2 3 4
1 3 5 6
1 2 4
1 3 5 7 8 9
2 4 6
2 5 7 10 11
4 6 8
0 4 7 9
0 4 8
6 11
6 10
27 41
10 11 17
10 7 33 44
11 7 26
17 26 5 8 15 16
33 5 21
44 21 31 18 29
8 31 20
27 15 20 13
41 16 13
18 23
29 23
```

Sample Output

```
8
2 4
1 3
2
1 5 7 8
4 6
5 10
4
0 4 9
8
6 11
10
```

****