National Institute of Technology Calicut
Department of Computer Science and Engineering
Fourth Semester B. Tech.(CSE)-Winter 2023-24
CS2094D Data Structures Laboratory
Assignment Cycle#3
Part A

Submission deadline (on or before): 21.03.2024, 11:00 PM

Policies for Submission and Evaluation:

- You must submit your assignment in the Eduserver course page, on or before the submission deadline.
- Ensure that your programs will compile and execute without errors using gcc compiler.
- During the evaluation, failure to execute programs without compilation errors may lead to zero marks for that evaluation.
- Your submission will also be tested for plagiarism, by automated tools. In case your code fails to pass the test, you will be straightaway awarded zero marks for this assignment and considered by the examiner for awarding F grade in the course. Detection of ANY malpractice related to the lab course can lead to awarding an F grade in the course.

Naming Conventions for Submission

- Submit a single ZIP (.zip) file (do not submit in any other archived formats like .rar, .tar, .gz). The name of this file must be

  `ASSGC<NUMBER>_<PART>_<ROLLNO>_<BATCHNO>_<FIRST-NAME>.zip`

  (Example: $ASSGC1\_A\_BxxyyyyCS\_CS01\_LAXMAN.zip$). 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

  `ASSGC<NUMBER>_<PART>_<ROLLNO>_<BATCHNO>_<FIRST-NAME>_<PROGRAM-NUMBER>.c`

  (For example: $ASSGC1\_A\_BxxyyyyCS\_CS01\_LAXMAN\_1.c$). If you do not conform to the above naming conventions, your submission might not be recognized by our automated tools, and hence will lead to a score of 0 marks for the submission. So, make sure that you follow the naming conventions.

Standard of Conduct

- Violation of academic integrity will be severely penalized. Each student is expected to adhere to high standards of ethical conduct, especially those related to cheating and plagiarism. Any submitted work MUST BE an individual effort. Any academic dishonesty will result in zero marks in the corresponding exam or evaluation and will be reported to the department council for record keeping and for permission to assign F grade in the course. The department policy on academic integrity can be found at: https://minerva.nitc.ac.in/?q=node/650.

QUESTIONS

1. You are given an undirected graph G. Implement the following functions using DFS as a sub-routine.

   (a) noOfConnectedComponents(G): Print the total number of connected components in the graph. A connected component is a maximal subgraph in which every pair of vertices are connected by a simple path.

   (b) sizeOfComponents(G): Print the sizes of all connected components in the graph, sorted in increasing order of their sizes. The size of a component is the number of vertices it contains.

   (c) noOfBridges(G): Print the number of bridges in the graph. If there are no bridges, print -1. A bridge is an edge whose removal increases the number of connected components in the graph.

   (d) noOfArticulationPoints(G): Print the number of articulation points in the graph. If there are no articulation points, print -1. An articulation point is a vertex whose removal increases the number of connected components in the graph

   Input format:

   - The first line contains an integer (m) specifying the number of nodes in the graph.
   - The subsequent next m lines contain the label of the respective node, followed by the nodes adjacent to it in ascending order of their labels.
   - Each line of input is a character from the menu list ['n','s','b','a','t']..
   - Input 'n' calls the function noOfConnectedComponents(G).
   - Input 's' calls the function sizeOfComponents(G).
   - Input 'b' calls the function noOfBridges(G).
   - Input 'a' calls the function noOfArticulationPoints(G).
   - Input 't' terminates the execution of the program.
   - All the inputs in a line are separated by space.

   Output format:

   - A line may contain -1.
   - The output of the result of any menu is printed in a new line.

   Sample Input 1:

   ```
   5
   1 2
   2 1
   3 4
   4 3 5
   5 4
   n
   s
   b
   a
   t
   ```

   Sample Output 1:

   ```
   2
   2 3
   3
   1
   ```

Sample Input 2:

8
1 2
2 1 3
3 2 4
4 3 5 6
5 4
6 4 7
7 6 8
8 7

Sample Output 2:

1
8
7
5

2. Given a directed graph containing N nodes as an adjacency matrix,implement the following using a Breadth-First Search(BFS) traversal algorithm:

   - Is_Topological_sort_possible(): This function checks whether topological sort can be done on the graph or not. A topological sort is a linear ordering of the vertices in a Directed *A*cyclic Graph (DAG) such that for every directed edge e1 $->$ e2 , vertex e1 appears before e2 in the ordering. Topological sort is possible if and only if the graph is a DAG.This function will print 1 if a topological sort is possible, -1 otherwise.
   - Number_of_strongly_components(): This function analyzes the graph defined by the edges and calculates the number of connected components.

   Input format:

   - The first line contains an integer N specifying the number of nodes in the graph.
   - Each line of input is a character from the menu list ['t','c','x'].
   - Input 't' calls the function Is_Topological_sort_possible().
   - Input 'c' calls the function Number_of_strongly_components()
   - Input 'x' terminates the execution of the program
   - All the inputs in a line are separated by space.

   Output Format:

   - A line may contain -1 or 1.
   - The output of the result of any menu is printed in a new line.

   Sample Input 1:

   6
   0 1 1 0 0 0
   0 0 1 0 1 0
   0 0 0 1 0 1
   0 0 0 0 0 0
   0 0 0 0 0 0
   0 0 0 0 0 0
   t
   c
   x

Sample Output 1:
1
6
Sample Input 2:
6
0 1 1 1 0 0
0 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
0 0 0 1 0 1
0 0 0 0 0 0
t
c
x

Sample Output 2:
1
4

3. You are given an integer N representing the number of nodes in the graph . The graph is represented by adjacency list .The next N lines represent an adjacency list where first integer is node itself and next integers are the nodes adjacent to it in ascending order . The goal is to implement the following using a Depth-First Search (DFS) traversal algorithm:

   - **All_path(Node1,Node2)**: Print all paths from a given source to a destination vertex. Separate each path with a new line , otherwise print -1 if no path exists between nodes.

     NOTE - paths are printed in sorted order
   - **valid_tree()**:This function checks if the given graph is a tree or not. A valid tree has the following properties:
     (a) There are no cycles (loops) in the graph.
     (b) Every vertex is reachable from the root vertex.
     (c) There are no isolated vertices (vertices not connected to the rest of the graph).

     function prints 1 if the given graph is a tree else -1

   Input format:

   - The first line contains an integer N specifying the number of nodes in the graph.
   - Next "N" lines are sequence of integers where first integer is node itself and next integers in that line are representing nodes which are adjacent to that node.
   - Each line of input is a character from the menu list ['a','t','x'].
   - Input 'a' calls the function All_path(Node1,Node2).
   - Input 't' calls the function valid_tree()
   - Input 'x' terminates the execution of the program.
   - All the inputs in a line are separated by space.

   Output format:

   - A line may contain -1 or 1.
   - The output of the result of any menu is printed in a new line.

Sample Input 1:
6
0 1 3 5
1 0 2 5
2 1 3 4
3 0 2
4 2
5 0 1
t
a 1 4
a 0 1
x

Sample Output 1 :
-1
1 0 3 2 4
1 2 4
1 5 0 3 2
0 1
0 3 2 1
0 5 1

Sample Input 2:
8
0 1 7
1 0 2
2 1 3 4 5 6
3 2
4 2
5 2
6 2
7 0
a 6 7
a 0 6
t
x

Sample Output 2 :
6 2 1 0 7
0 1 2 6
1