CS 513

DS Lab

Assignment 4

Graph

Abhijeet Padhy

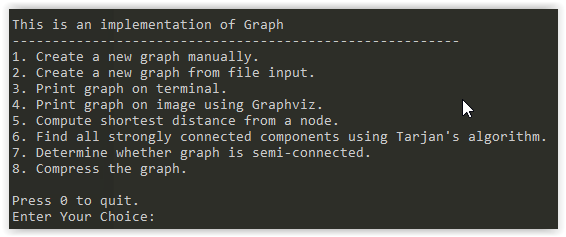
Roll: 214101001

# Index

1. Basic Information
2. Class Implementation
3. DFS Traversal and Classification of edges
4. Strongly connected components using Tarjan’s Algorithm
5. Removal of Edges from a graph
6. Semi connected Component
7. Dijsktra’s shortest path algorithm

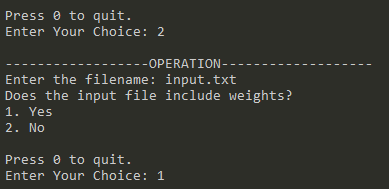
# Basic Information

The program starts with the following menu:

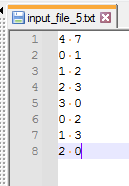


Option 1: Use this to enter the edges of the graph manually.

Option 2: Use this to take file input to initialise the edges of the graph.



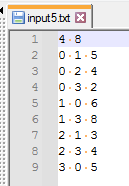
Enter the name of the input file. It then asks if the input file also contains weights. Press 1 if input file contains weights. Press 0 if input file does not contain weights. The format of the input file is shown below:



Input file without weights

The first line contains two numbers (4 and 7 in this example). The first number of first line is the number of vertices in the graph. The second number is the number of edges. The lines following the first line contain the edge information. Each row consists of two numbers, source vertex and destination vertex of the edge. In the above example, there are 4 vertices and 7 edges. Since edge weight is not specified, for each edge, the weight is 1.

However, if we choose to provide the edge weights, the option 2 should be used in the menu and the input file format corresponding to this is:

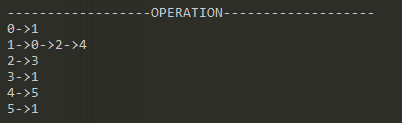


Input file with weights.

Here everything is same as the previous case except the fact that each row now contains a weight value. Thus each row consists of <src vertex> <dest vertex> <weight of the edge>.

Once file input is completed, the graph is initialised. We can then perform various operations on it.

Option 3: If we want to print the adjacency list representation of the graph, use this option. The output produced is as such:



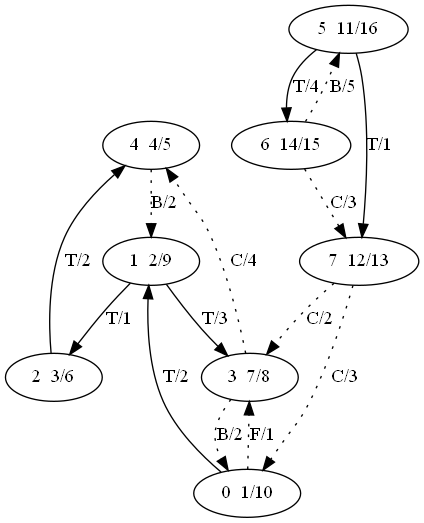
Each row corresponds to a vertex. The first number of a row is the source vertex number to all the edges going out from it. The numbers following the first number are the destination vertices of the edges going out from the source vertex. Thus vertex 1 has an edge to vertex 0, vertex 2 and vertex 4.

Options 4 to 6 will be explained in detailed in the following sections.

**Note: If number of vertices in the graph is N, vertices can only have names from 0 to N-1. No other vertex number is allowed.**

# DFS Traversal

Use option 4 to run DFS Traversal on the graph. The output produced by DFS Traversal is as shown below:



Each node consists of three values:

**<Vertex Number> <Start Time>/ <End Time>**

Vertex number starts from 0 to N-1 where N is the number of nodes.

Time starts from 1 and then increases at each visit of a vertex. The time spent on a vertex is 1.

Each edge has a label of this form:

**<Type of edge>/<Edge Weight>**

An edge can be of the following four types:

|  |  |
| --- | --- |
| **Edge Type** | **Denoted By** |
| Tree | Black, solid, T |
| Forward | Black, dotted, F |
| Back | Black, dotted, B |
| Cross | Black, dotted, C |

The vertex from which we start DFS Traversal will have start time as 1. All the tree edges are denoted by solid edges. Thus we can understand the DFS Tree just by following the solid tree edges.

Name of method is: **dfs\_traversal**

Steps of Implementation: