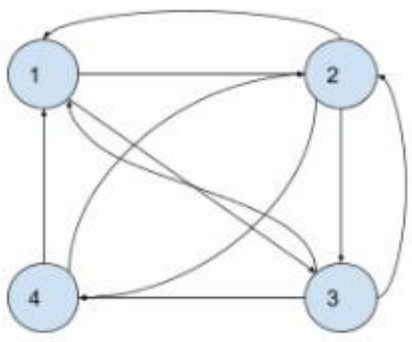
ASSIGNMENT-4

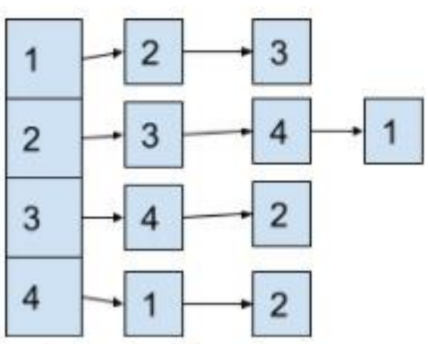
Graph

Name-Gaurav Kumar
Roll no- 214101018

Let G(V,E) is a directed Graph, where V is vertex set and E is the edge set. We represent the Graph using the adjacency list, that is, for every vertex v there is a linked list which contains the set of adjacent vertices reachable from that vertex (means out edge).



Graph G(V,E)



Adjacency list

- Vertices always start from 1 to n
- N is number if vertices

DFS () –

Let G(V, E) is a directed Graph, where V is vertex set and E is the edge set. We represent the Graph using the adjacency list, that is, for every vertex v there is a linked list which contains the set of adjacent vertices reachable from that vertex (means out edge).To do DFS we keep two information associated with each vertices num[v] and visited[v], which indicate that vertex v currently visited or not.

Steps involve in DFS ()

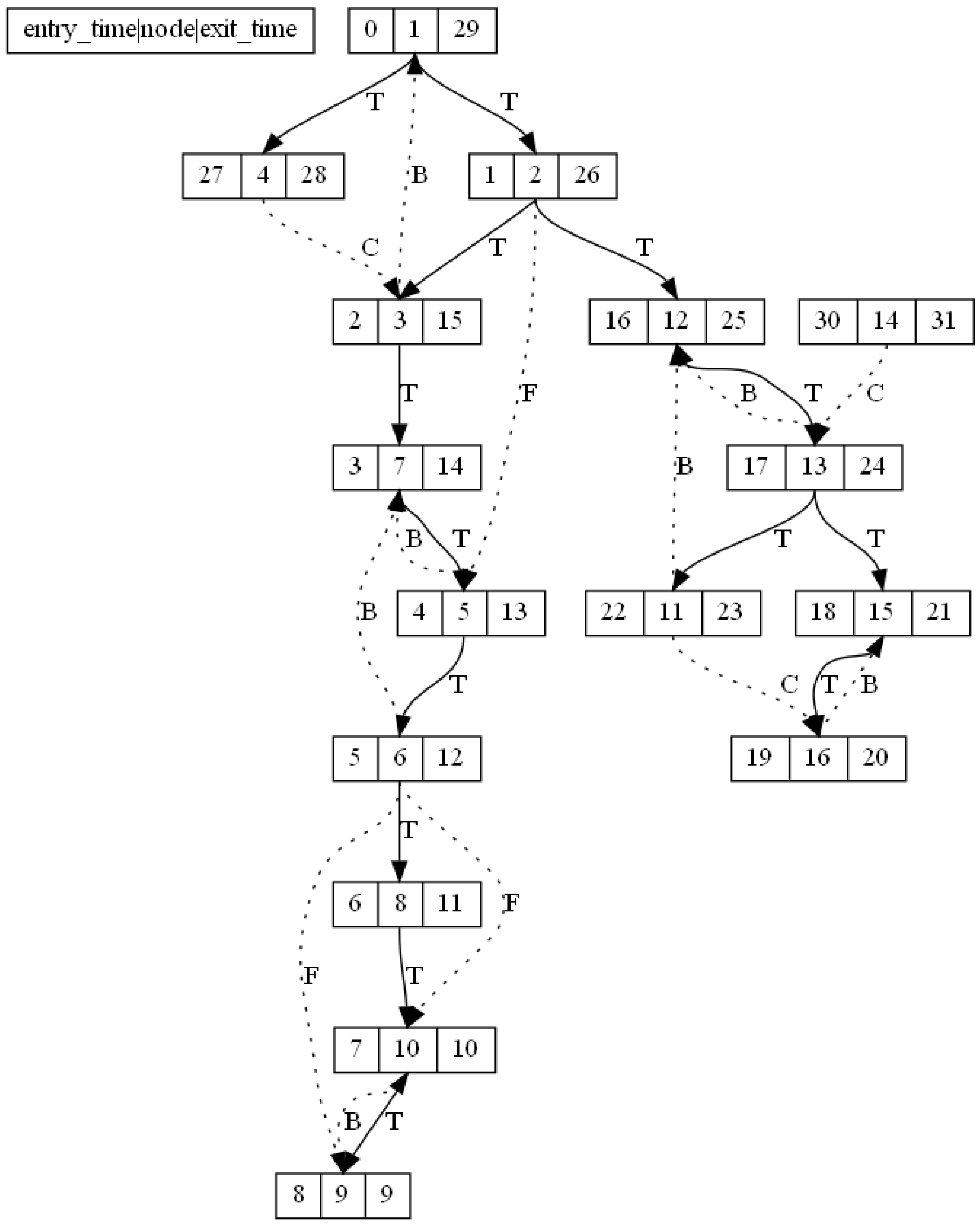
1. Initialize num[v]=0 for all vertices num[v] containing the sequence(or time) of visited nodes. And i=0, i is used for finding the sequence of dfs visited vertices.
2. Initialize visited[v]=false for all vertices
3. We start calling DFS(v) for node i to n if i-th node is unvisited then DFS will call for node i.
4. During DFS(v) we make visited[v] =true and num[v]=i+1 and find the adjacent vertex of vertex(v)
    a. For all w ε adj(v) then (v,w) is an edge
        i. if(num[w]==0) then apply DFS(w)
        ii. Else if num[w] >num[v] then it is forwarded edge
        iii. Else if mark(w)=0 then it is cross edge
        iv. Otherwise it is back edge
        v. If entry time and exit time of v is greater then the entry time and exit time of w then (v,w) edge is cross edge.
5. Every node contain the entry time and exit time of node

Time Complexity

we have to visit every edge once

so time complexity O(V+E)

DFS traverse tree -



T => Tree edge
B => Back Edge
F => forward edge
C=> Cross edge

Tarjan algorithm –

A directed Graph is strongly connected if there is a path between all pairs pf vertices. A strongly connected component(SCC) of a directed graph is a maximal strongly connected subgraph. It requires only one DFS traversal to implement this algorithm.

Tarjan algorithm –

1. DFS produces a DFS tree.
2. Strongly connected components form the subtree of the DFS tree
3. There is no back edge from one SCC to another SCC

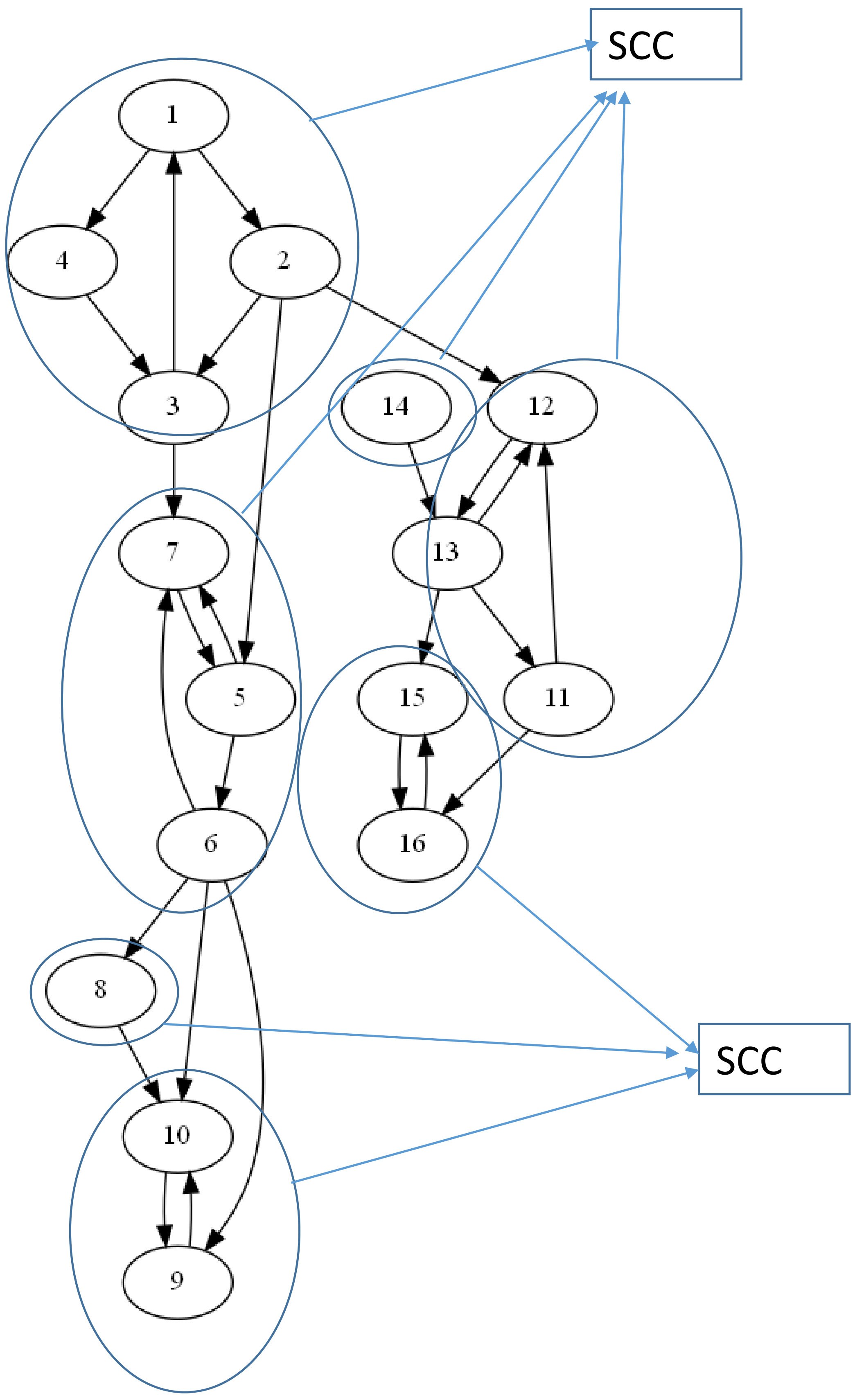
Steps for finding SCC in Graph –

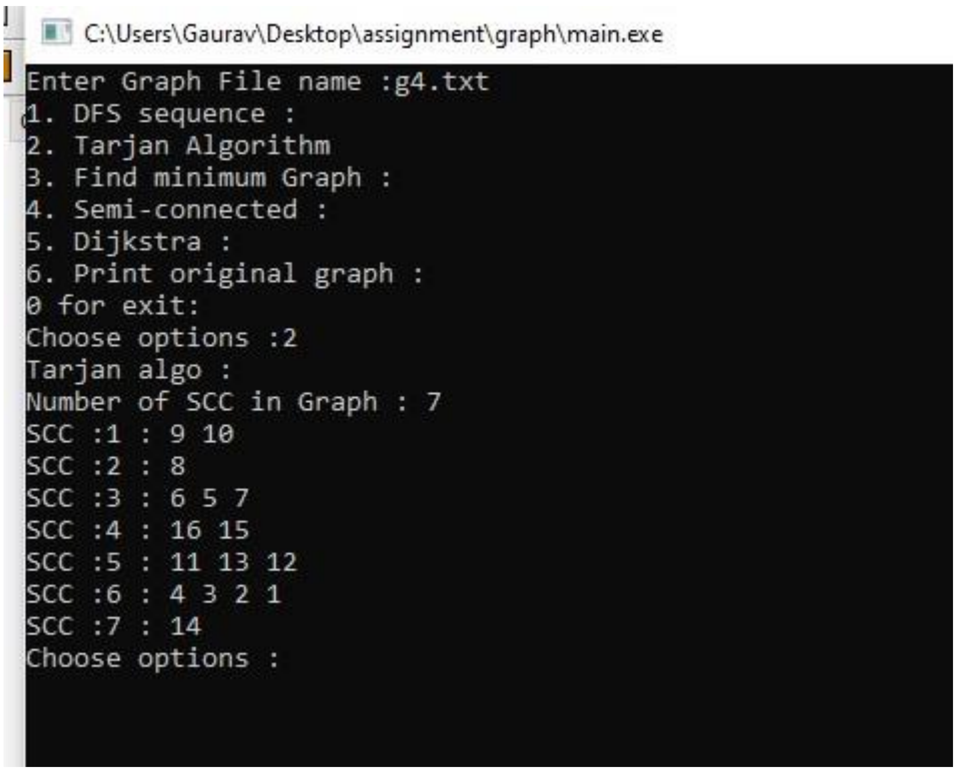
1. We calculate low[],num[] array and we use stack for containing the same component vertices.
2. Num[] array contain the time of vertices when 1st time visited
3. Low array contains the minimum vertex number of SCC
4. Stack contain the vertices which is of same SCC
5. The low value of which vertices is same they belong to the same SCC.
6. Apply DFS for all vertices on Graph
   a. Make visited [v] as true, assign the time and push vertices in stack
   b. For all vertices which is adjacent of vertex(v) check is node visited or not if node is not visited apply DFS on unvisited node
   c. If node is in stack assign minimum low number to the vertex(min(low(v), low(adjacent node))
   d. If visited order(DFS order) is same as low number means all vertices which belongs to same SCC are in stack
   e. pop vertices from stack ad print the vertices.

Time complexity –

We have to traverse the graph once we apply DFS on graph

So time complexity O(V+E)





```
Enter Graph File name :g4.txt
1. DFS sequence :
2. Tarjan Algorithm
3. Find minimum Graph :
4. Semi-connected :
5. Dijkstra :
6. Print original graph :
0 for exit:
Choose options :2
Tarjan algo :
Number of SCC in Graph : 7
SCC :1 : 9 10
SCC :2 : 8
SCC :3 : 6 5 7
SCC :4 : 16 15
SCC :5 : 11 13 12
SCC :6 : 4 3 2 1
SCC :7 : 14
Choose options :
```

Minimized Graph() –

For finding minimized Graph we have to

1. Remove the parallel edges between SCC.
2. Keep the minimum number of edges which is required for create the same SCC component.
3. SCC component graph should be same
4. If two SCC connected in original graph, then in minimized Graph SCC should be connected with one edge.

Steps for finding the minimum Graph

a. Find the SCC of whole graph
b. Map vertices of same SCC with a number.
c. Create a matrix of (size of matrix=number of SCC)
d. Initialize matrix[i][j]=false

e. Traverse graph
    a. If(u,v) is an edge then
        i. If vertex u and v belongs to same SCC apply dfs on u and v is v is reachable without this edge then remove the edge and vice-versa otherwise keep this edge
        ii. If u and v are from different SCC then check an edge exist or not if(matrix[u][v]==true means edge exist so remove the redge
        iii. Otherwise make matrix[u][v]=true

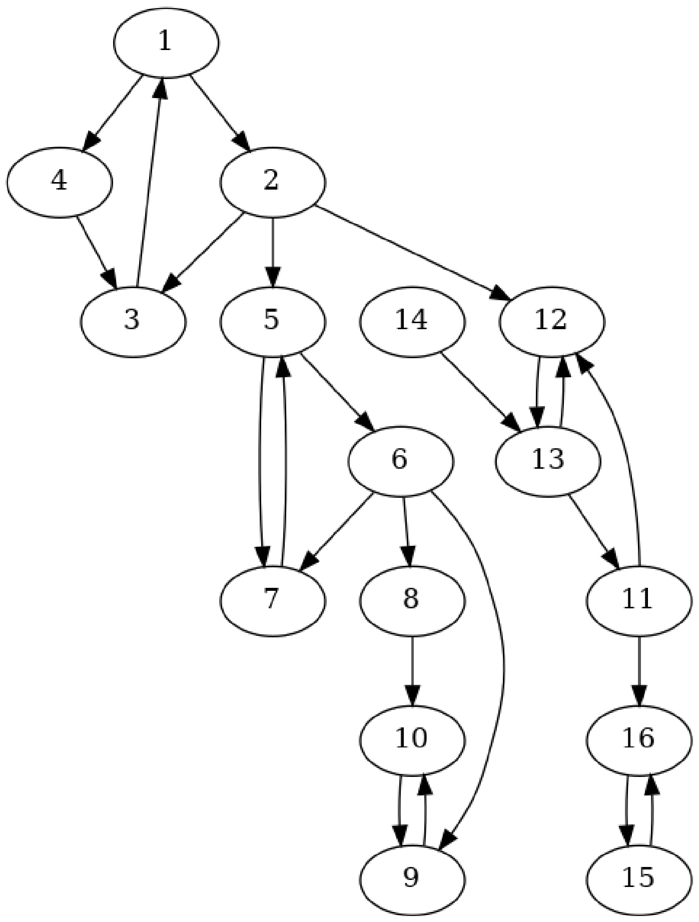
Time complexity –

SCC – O(V+E)

Traverse graph (E)

Delete edge O(E)

Total time O(V+E)



Semi-connected () –

A directed graph G = (V, E) is semi connected if, for all pairs of vertices u, v in V, we have either a path u ---> v or a path v ---> u

Steps –

1. Find the strongly connected component of entire Graph.
2. Shrink the graph as a DAG graph by taking one SCC as a vertex
3. Apply the DFS on new graph on the vertex which have in degree 0
4. If all the vertices of new graph is not reachable (reachability checked by applying DFS) from the 0th degree vertex then Graph is not semi-connected
5. If all the vertices of new Graph is reachable from the 0th degree vertex then graph is semi-connected.

Time complexity –

For finding SCC we traverse only once the graph using BFS O(V+E)

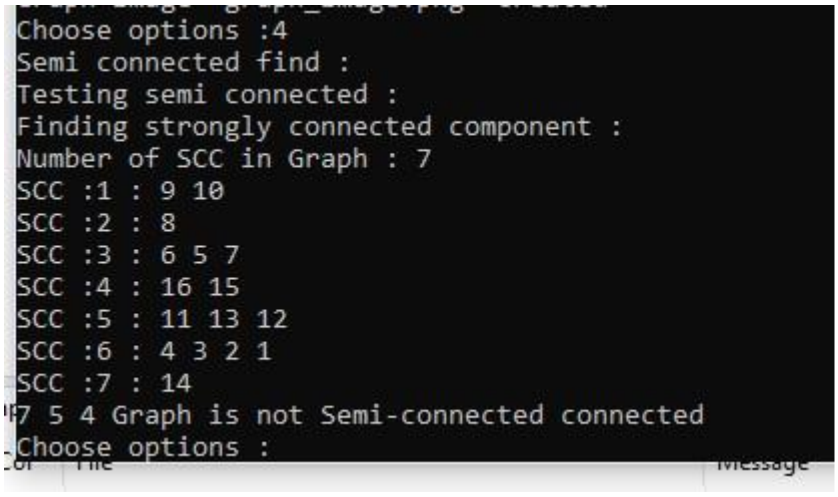
Create a component graph O(E)

Find 0th degree node O(V)

Apply DFS on shrink graph O(V+E)

Total time =O(V+E)+O(E)+O(V)+O(V+E)

Time complexity O(V+E)



```
Choose options :4
Semi connected find :
Testing semi connected :
Finding strongly connected component :
Number of SCC in Graph : 7
SCC :1 : 9 10
SCC :2 : 8
SCC :3 : 6 5 7
SCC :4 : 16 15
SCC :5 : 11 13 12
SCC :6 : 4 3 2 1
SCC :7 : 14
7 5 4 Graph is not Semi-connected connected
Choose options :
```

Dijkstra(v) –

Dijkstra algorithm is used for finding the minimum distance of all vertices from starting vertex.

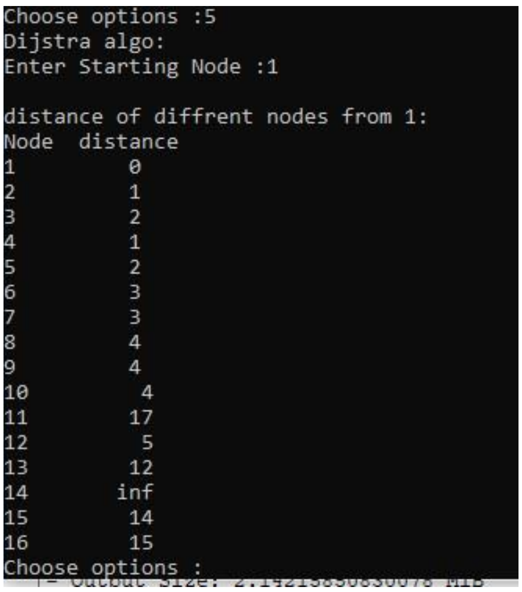
Steps –

1. Initialize distance of all vertices as infinity (dist[I] = inf)
2. Create a min-priority queue which contain the distance and vertex.
3. Initialize distance of start node(v) as 0
4. Push the start node in priority queue
5. Run the loop until queue will not empty
    a. Remove a vertex from queue which have minimum distance
    b. Find the adjacent of vertex
        i. if the sum of distance of removed vertex and weight of edge is less than the distance of adjacent vertex then update and add in queue.
6. Print distance array.

Time complexity –

Dijkstra's algorithm visits every node once (O(V)), and tries to relax all adjecent nodes via the edges. Therefore it iterates over each edge exactly twice (O(E)), each time accessing the priority queue

complexity is O(ElogV+V).

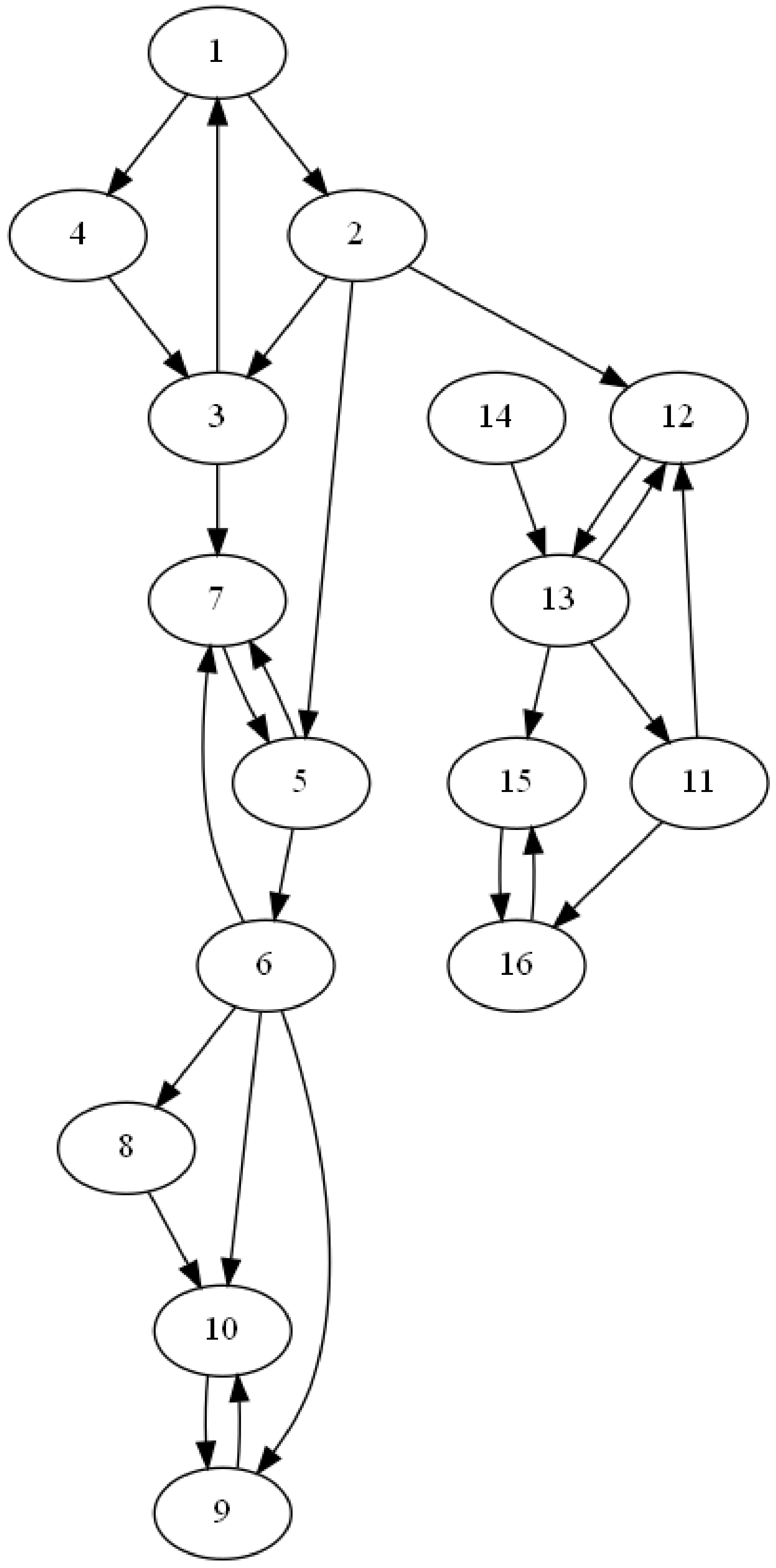


```
Choose options :5
Dijstra algo:
Enter Starting Node :1

distance of diffrent nodes from 1:
Node  distance
1         0
2         1
3         2
4         1
5         2
6         3
7         3
8         4
9         4
10         4
11        17
12         5
13        12
14       inf
15        14
16        15
Choose options :
```

Print_Graph()

I am creating the graph png of name “graph_image.png


1
4
2
3
14
12
7
13
5
15
11
6
16
8
10
9