EXPERIMENT 7

Experiment No: 7 **Date:** 10/04/2021

Aim: Implementation of Single Source Shortest Path Algo (Dijkstra's Algo) and estimate its step count

Theory:

Dijkstra's Shortest Path Algorithm

- Given a graph and a source vertex in the graph, find shortest paths from source to all vertices in the given graph.
- Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree.
- Like Prim's MST, we generate a SPT (shortest path tree) with given source as root.
- We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree.
- At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has a minimum distance from the source.

Time Complexity

- The time complexity of the above code/algorithm looks O(V^2) as there are two nested while loops.
- If we take a closer look, we can observe that the statements in inner loop are executed O(V+E) times (similar to BFS).
- The inner loop has decreaseKey() operation which takes O(LogV) time.

EXPERIMENT 7

- So overall time complexity is O(E+V)*O(LogV) which is

$$O((E+V)*LogV) = O(ELogV)$$

- Note that the above code uses Binary Heap for Priority Queue implementation.
- Time complexity can be reduced to $O(E + VLogV)$ using Fibonacci Heap.
- The reason is, Fibonacci Heap takes O(1) time for decrease-key operation while Binary Heap takes O(Logn) time.

```
Begin Algorithm Dijkstra (G, s)
    For each vertex v ∈ V                                              } O(V) time
        d [v] ← ∞ // an estimate of the min-weight path from s to v    }
    End For                                                            }
    d [s] ← 0                                                            O(V) time to
    S ← Φ // set of nodes for which we know the min-weight path from s   Construct a
    Q ← V // set of nodes for which we know estimate of min-weight path from s   Min-heap
    While Q ≠ Φ  ← done |V| times = O(V) time
        u ← EXTRACT-MIN(Q)  ← Each extraction takes O(logV) time
        S ← S U {u}
        For each vertex v such that (u, v) ∈ E  } done O(E) times totally
            If v ∈ Q and d [v] > d [u] + w (u, v) then  } It takes O(logV) time when
                d [v] ← d [u] + w (u, v)                } done once
                Predecessor (v) = u                     }
            End If
        End For
    End While
End Dijkstra
```

Overall Complexity: O(V) + O(V) + O(VlogV) + O(ElogV)
Since |V| = Ω(|E|), the VlogV term is dominated by the ElogV term. Hence, overall complexity = O(|E|*log|V|)

EXPERIMENT 7

Algorithm

```
Algorithm ShortestPaths(v,cost,dist,n)
// dist[j], 1<jn, is set to the length of the shortest
// path from vertex v to vertex j in a digraph G with n
// vertices. dist[v] is set to zero. G is represented by its
// cost adjacency matrix cost[1: n. 1:n].
{
    for i:=1 to n do
    { // Initialize S.
        S[i]:= false; dist[i]:= cost[v, i];
    }
    S[v]:= true; dist[v] := 0.0; // Put v in S.
    for num=2 to n - 1 do
    {
    // Determine n - 1 paths from v.
    Choose u from among those vertices not
    in S such that dist[u] is minimum;
    S[u]:= true; // Put u in S.
    for (each w adjacent to u with S[w] = false) do
```

EXPERIMENT 7

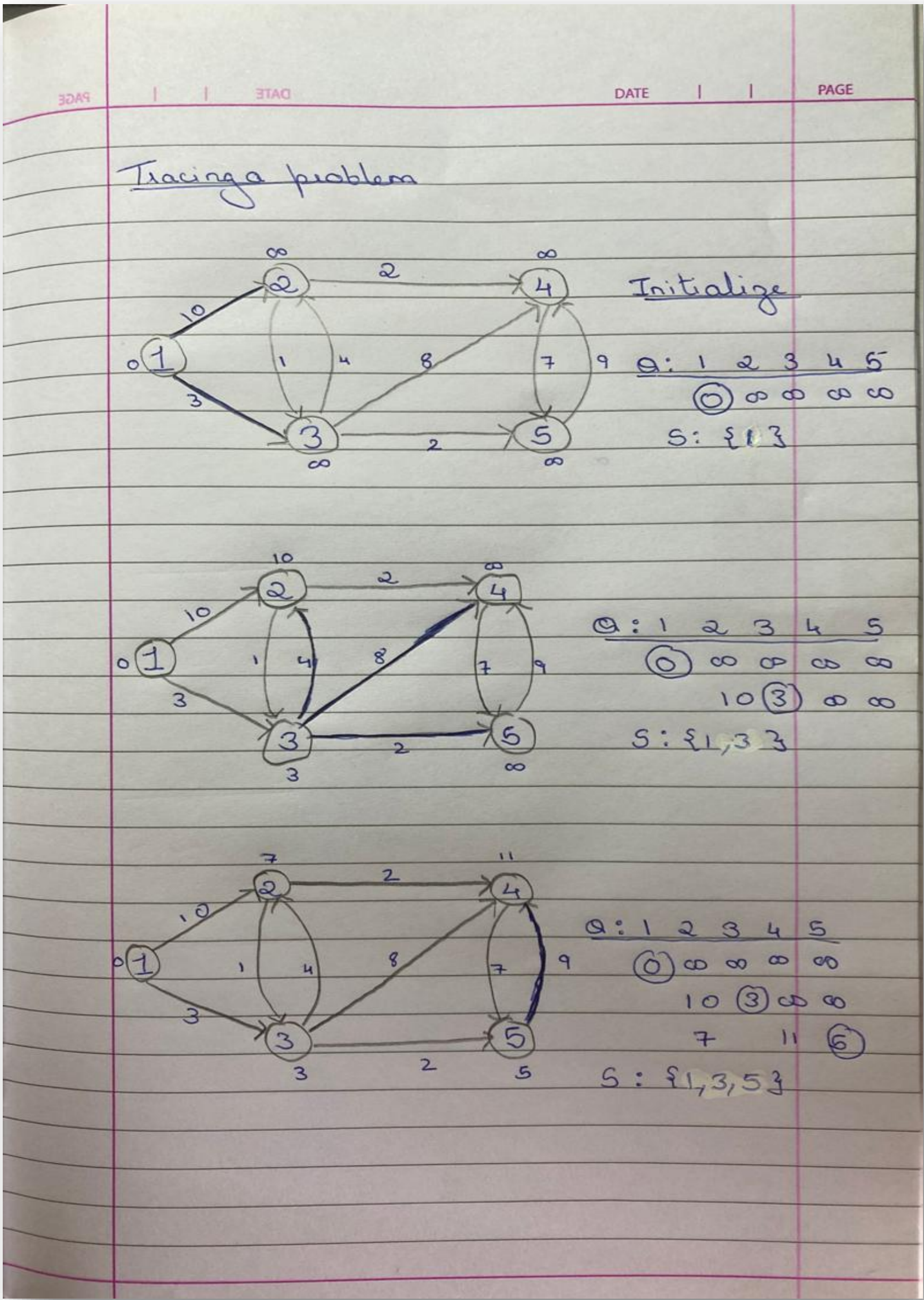
```
                // Update distances.
                if (dist[w]> dist[u] + cost[u, w])) then
                        dist[w]: dist[u] + cost[u, w];
        }
}
```

Algorithm writing

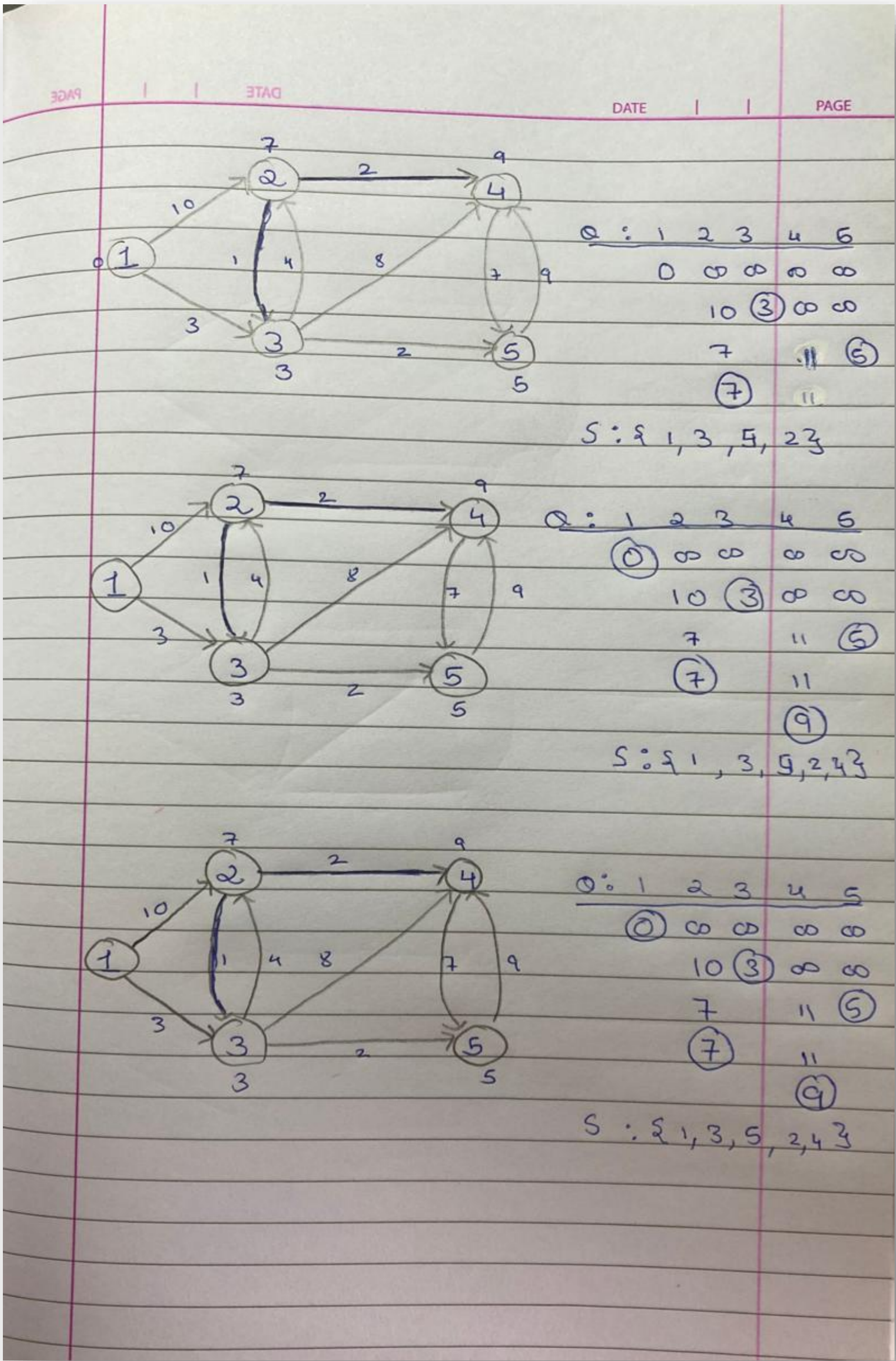
1. Mark your selected initial node with a current distance of 0 and the rest with infinity.
2. Set the non-visited node with the smallest current distance as the current node C.
3. For each neighbour N of your current node C:
    - add the current distance of C with the weight of the edge connecting C-N.
    - If it's smaller than the current distance of N, set it as the new current distance of N.
4. Mark the current node C as visited.
5. If there are non-visited nodes, go to step 2.

EXPERIMENT 7

Tracing with Example



EXPERIMENT 7



EXPERIMENT 7

Program

```
#include<iostream>

using namespace std;

#define V 6

int stepcount=0;

int minDistance(int dist[], bool sptSet[])

{

        int min = INT_MAX;stepcount++;

        int min_index;


        for (int v = 0; v < V; v++)

   {

      stepcount++;

      stepcount++;

      if (sptSet[v] == false && dist[v] <= min)

      {

         min = dist[v], min_index = v;stepcount++;

      }

   }
```

EXPERIMENT 7

```
        stepcount++;

        return min_index;

}

void printSolution(int dist[])

{

   cout<<"Vertex \t\t Distance from Source"<<endl;stepcount++;

        for (int i = 0; i < V; i++)

   {

       stepcount++;

       cout<<i<<"\t\t\t"<<dist[i]<<endl;stepcount++;

   }


}


void dijkstra(int graph[V][V], int src)

{

        int dist[V];

        bool sptSet[V];
```

EXPERIMENT 7

```
        for (int i = 0; i < V; i++)
    {
        stepcount++;
        dist[i] = INT_MAX, sptSet[i] = false;stepcount++;
    }
        dist[src] = 0;stepcount++;



        for (int count = 0; count < V - 1; count++)
        {
      stepcount++;
                int u = minDistance(dist, sptSet);stepcount++;
                sptSet[u] = true;stepcount++;
                for (int v = 0; v < V; v++)
      {
        stepcount++;
        stepcount++;
                        if (!sptSet[v] && graph[u][v] && dist[u] !=
INT_MAX
```

EXPERIMENT 7

```
                                   && dist[u] + graph[u][v] < dist[v])
                                   {
                                      dist[v] = dist[u] +
graph[u][v];stepcount++;
                                   }
               }
                  }
            stepcount++;
                  printSolution(dist);
         }

         int main()
         {
            int graph[V][V];
                  cout<<"Enter the vertices for a graph with 6
vetices:";stepcount++;
            for (int i=0;i<V;i++)
            {
               stepcount++;
               for(int j=0;j<V;j++)
```

EXPERIMENT 7

```
        {
            stepcount++;
                        cin>>graph[i][j];stepcount++;
        }
    }
    stepcount++;
        dijkstra(graph, 0);
        cout<<"*************"<<endl;
        cout<<"Stepcount:"<<stepcount<<endl;
        cout<<"*************"<<endl;
    return 0;
}
```

EXPERIMENT 7

Output

```
C:\Users\Vedant\OneDrive\Desktop\GEC\MADF\PRA...
Enter the vertices for a graph with 6 vetices:
8 4 6 2 5 7
7 5 8 4 2 1
3 5 7 6 1 4
8 5 1 3 4 9
7 8 5 2 1 7
9 4 7 2 5 6
Vertex           Distance from Source
0                        0
1                        4
2                        3
3                        2
4                        4
5                        5
*************
Stepcount:268
*************


--------------------------------
Process exited after 36.81 seconds with return value 0
Press any key to continue . . .
```

Conclusion

- Detailed concept of Single Source Shortest Path Algo (Dijkstra's Algo)was studied successfully.
- Program using Dijkstra's Algorithm was executed successfully.
- The step count for the Dijkstra's Algorithm was obtained.