Third Year B. Tech., Sem V 2022-23

Design and Analysis of Algorithm Lab

Assignment / Journal submission

PRN/ Roll No: 21520010

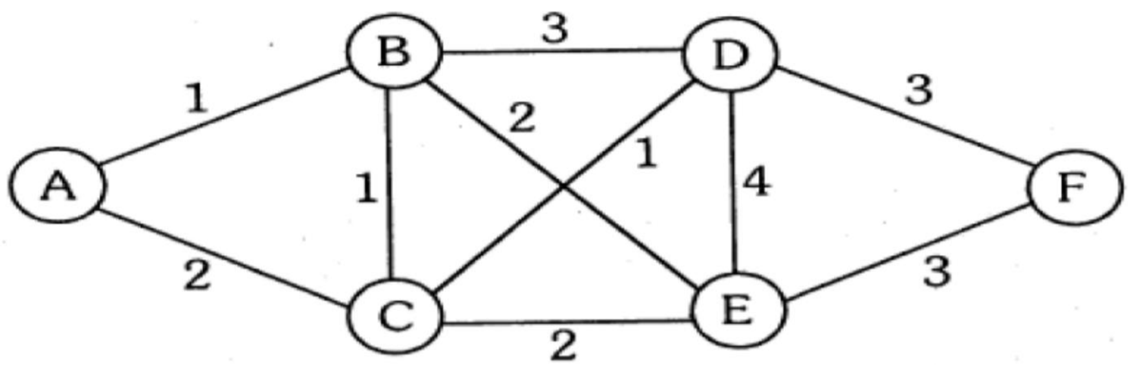
Full name: Mayur Sunil Savale

Batch: T8

Assignment: Week 8

Title of assignment: Greedy Method

1. From a given vertex in a weighted connected graph, implement shortest path finding Dijkstra's algorithm.



Ans:

a) Algorithm: (Pseudocode)

```
void djikstra(int graph[6][6], int index)
{
        // initialize a array dist[] to carry shortest distance
        // then to keep track have a visited array
        // use a queue for BFS(Breadth First Search)
        while(!g.empty())
```

```
    {
        check if current distance of each node is shortest from
        index node.
    }
    //print the graph nodes and distance to travel from given index
}
```

b) Code snapshots of implementation

```
#include<bits/stdc++.h>
using namespace std;

bool visited[6];
int dist[6];

void dijkstra(int graph[6][6], int index)
{
    for(int i=0;i<6;i++)
    {
        dist[i]=INT_MAX;
        visited[i]=false;
    }
    queue<int> q;
    q.push(index);
    dist[index]=0;
    while(!q.empty())
    {
        int r=q.front();
        q.pop();
        visited[r]=true;
        for(int i=0;i<6;i++)
        {
            if(graph[i][r]!=0)
            {
                if(visited[i])
                    continue;
```

```
					if(dist[i]>dist[r]+graph[i][r])
						dist[i]=dist[r]+graph[i][r];
					q.push(i);
				}
			}
		}
		cout<<"Node\tDistance\n";
		for(int i=0;i<6;i++)
			cout<<(char)(i+'A')<<"\t"<<dist[i]<<"\n";
}

int main()
{
		int graph[6][6]={
			{0,1,2,0,0,0},
			{1,0,1,3,2,0},
			{2,1,0,1,2,0},
			{0,3,1,0,4,3},
			{0,2,2,4,0,3},
			{0,0,0,3,3,0}
		};
		cout<<"For Node A the Shortest paths are as follows:\n";
		dijkstra(graph,0);
		return 0;
}
```

Output:

```
C:\Users\mayur\OneDrive\Desktop\5 th sem\DAA Assignment\Assignment-8\Q1.exe
For Node A the Shortest paths are as follows:
Node    Distance
A       0
B       1
C       2
D       3
E       3
F       6

--------------------------------
Process exited after 1.08 seconds with return value 0
Press any key to continue . . .
```

c) Complexity of proposed algorithm (Time & Space)

- Time Complexity: O(E*logV)
- Space Complexity: O(V+E)

d) Your comment (How your solution is optimal?)

- Depending upon type of data structures used time complexity is typically O(E*logV) which is competitive against other shortest path algorithms.
- The constrain with this algorithm is that it can’t work with undirected graphs containing -ve weights or graphs containing cycles with one -ve edge and overall edge weight of cycle be -ve as it acts in greedy manner i.e. always searching better distance(shortest).

2. Show that Dijkstra's algorithm doesn't work for graphs with negative weight edges.

Ans:

   a. For a undirected graph if a edge weight is -ve then the greedy architecture of the algorithm would make it go infinite as it always want more optimal result.
   b. For a directed graph if there is a -ve cycle(overall edge weight of cycle be -ve) then the same case takes place as it will go infinite to find more optimal solution.
   c. That's why dijkstra's algorithm can't work with graphs having -ve edge weights.

3. Modify the Dijkstra's algorithm to find shortest path.

Ans:

a) Algorithm: (Pseudocode)

```
void Dijkstra(int graph[6][6], int index)
{
        //keep a prev array to keep previous node index in graph
        //initialize a array dist[] to carry shortest distance
        //then to keep track have a visited array
        //use a queue for BFS(Breadth First Search)
        while(!q.empty())
        {
                check if current distance of each node is shortest
                from index node. If it is shortest distance then update
                the prev array.
        }
        //print the graph nodes with help of prev array
}
```

b) Code snapshots of implementation

```
#include<bits/stdc++.h>
using namespace std;

bool visited[6];
int dist[6];

void dijkstra(int graph[6][6], int index,int tr)
{
   int prev[6];
   for(int i=0;i<6;i++)
   {
      dist[i]=INT_MAX;
      prev[i]=-1;
      visited[i]=false;
   }
   queue<int> q;
   q.push(index);
   dist[index]=0;
   while(!q.empty())
   {
      int r=q.front();
      q.pop();
      visited[r]=true;
      for(int i=0;i<6;i++)
      {
         if(graph[i][r]!=0)
         {
            if(visited[i])
               continue;
            if(dist[i]>dist[r]+graph[i][r])
            {
               dist[i]=dist[r]+graph[i][r];
               prev[i]=r;
```

```
                }
                q.push(i);
            }
        }
    }
    vector<char> path;
    if(dist[tr]==INT_MAX)
        return;
    cout<<"Shortest Path to "<<(char)(tr+'A')<<" :\n";
    for(int i=tr;i!=index;i=prev[i])
        path.push_back((char)(i+'A'));
    reverse(path.begin(), path.end());
    for(auto it: path)
        cout<<it<<" ";
    cout<<"\n";
}

int main()
{
    int graph[6][6]={
        {0,1,2,0,0,0},
        {1,0,1,3,2,0},
        {2,1,0,1,2,0},
        {0,3,1,0,4,3},
        {0,2,2,4,0,3},
        {0,0,0,3,3,0}
    };
    cout<<"For node A the Shortest paths are as follows:\n\n";
    for(int i=1;i<6;i++)
        dijkstra(graph,0,i);
    return 0;
}
```

Output:

```
PS C:\Users\mayur> cd "c:\Users\mayur\OneDrive\Desktop\5 th sem\DAA Assignment\Assignment-8\" ;
For node A the Shortest paths are as follows:

Shortest Path to B :
B
Shortest Path to C :
C
Shortest Path to D :
C D
Shortest Path to E :
B E
Shortest Path to F :
C D F
PS C:\Users\mayur\OneDrive\Desktop\5 th sem\DAA Assignment\Assignment-8>
```

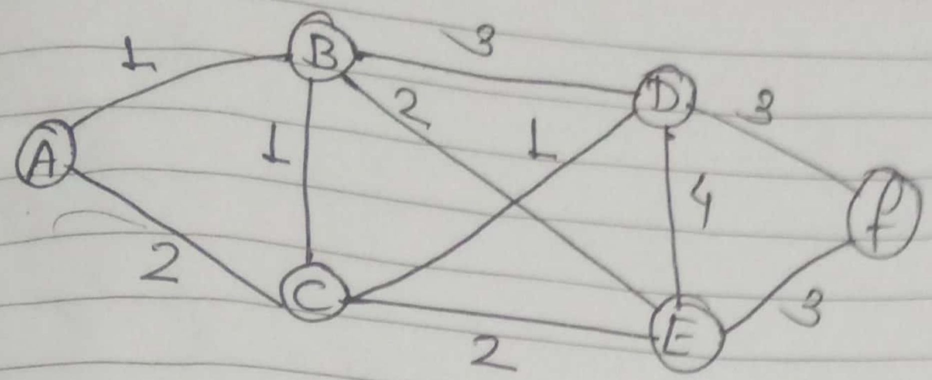
c) **Complexity of proposed algorithm (Time & Space)**

- Time Complexity: O(E*logV)
- Space Complexity: O(V+E)

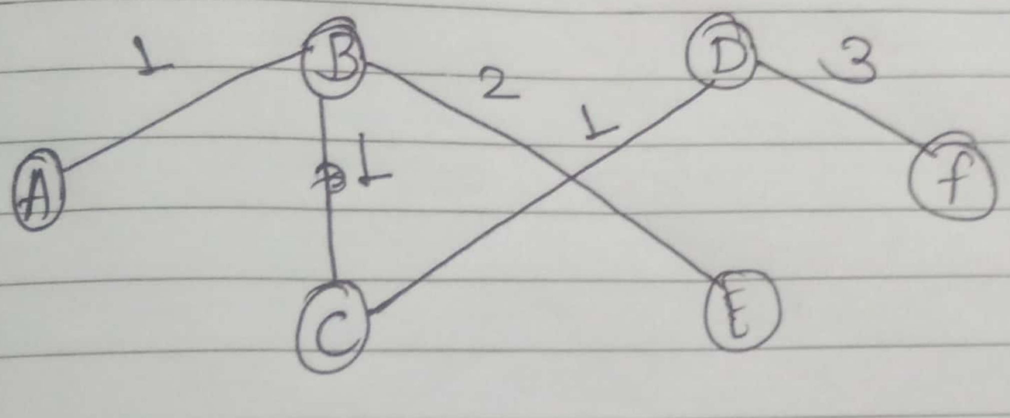
d) **Your comment (How your solution is optimal?)**

- In this modification we just kept a prev array to keep previous element in shortest path to node so it doesn't add more complexity to the current algorithm which still keeps the dijkstra's algorithm competitive.

Assignment-8.



Path	A	B	C	D	E	f
A	0	1	2	∞	∞	∞
A, B	0	1	2	3	3	∞
A, B, C	0	1	2	3	3	∞
A, B, C, D	0	1	2	3	3	6
A, B, C, D, E	0	1	2	3	3	6
A, B, C, D, E, f	0	1	2	3	3	6



cost = 1 + 1 + 1 + 2 + 3
= 8