

# Assignment No: 3

## Aim :

There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight takes to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Justify the storage representations used.

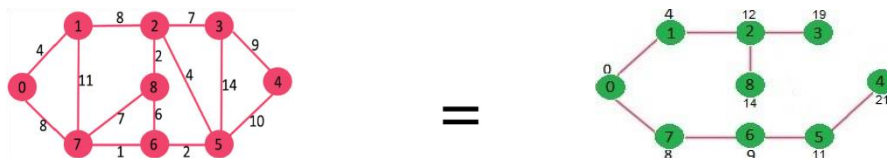
## Objectives:

\_To understand the various operations on graphs.

## Theory:

Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, we generate a shortest path tree with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, and other set includes vertices not yet included in shortest path tree.

For Example:



At every step of the algorithm, we find a vertex which is in the other set and has a minimum distance from the source. Below are the detailed steps used in Dijkstra's algorithm to find the shortest path from a single source vertex to all other vertices in the given graph.

## Algorithm:

1. Create priority queue pq
2. Enqueue(pq,s)
3. For(i=1;i<=g->v;i++)
 

Distance[i]=-1
4. Distance[s]=0

```
5. while(!isemptyqueue(pq))
{
    5.1 v=deletemin(pq);

    5.2 for all adjacent vertices w to v
    {

        Compute new distance d=distance[v]+weight[v][w];

        If(Distance[w]==-1)
        {
            Distance[w]=new distance d;

            Insert w in priorityqueue with priority d

            Path[w]=v}

        If(Distance[w]>newdistance d)
        { distance[w]=new distance d;

            Update priority of vertex w to be d;

            Path[w]=v;

        }

    }
}
```

### **Program:**

```
#include<iostream>

#define MAX 20
```

```
using namespace std;
```

```
class dijkstra
{
    int city;
    int distance[MAX][MAX];
    int d[MAX];
    int visited[MAX];
    public:
        void city_no();
        int minvertex();
        void matrix_fill();
        void dijkstra_code();
        void display();
};
```

```
void dijkstra::city_no()
{
    cout<<"\n enter the number of cities (including cities A and B) : ";
    cin>>city;
}
```

```
int dijkstra::minvertex()
{
    int mvertex=-1;
    for(int i=0;i<city;i++)
    {
```

```

        if(visited[i]==0 && (mvertex==-1 || d[i]<d[mvertex]))
            mvertex=i;
    }
    return mvertex;
}

```

```

void dijkstra::matrix_fill()
{
    cout<<"\n enter the distances between the cities : ";
    for(int i=0;i<city;i++)
    {
        cout<<"\n For city "<<i<<endl;
        for(int j=0;j<city;j++)
        {
            if(i==j)
                distance[i][j]=0;
            else
                cin>>distance[i][j];
        }
        d[i]=INT_MAX;
        visited[i]=0;
    }
}

```

```

void dijkstra::dijkstra_code()
{
    d[0]=0;
    for(int i=0;i<city-1;i++)

```

```

    {
        int mvertex=minvertex();
        visited[mvertex]=1;
        for(int j=0;j<city;j++)
        {
            if((distance[mvertex][j]!=0)&&(visited[j]==0))
            {
                int dist=d[mvertex]+distance[mvertex][j];
                if(dist<d[j])
                d[j]=dist;
            }
        }
    }
}

```

```

void dijkstra::display()
{
    cout<<"\n distance of cities from city o \n";
    cout<<"city Distance\n";
    for(int i=0;i<city;i++)
        cout<<i<<"\t"<<d[i]<<endl;
}

```

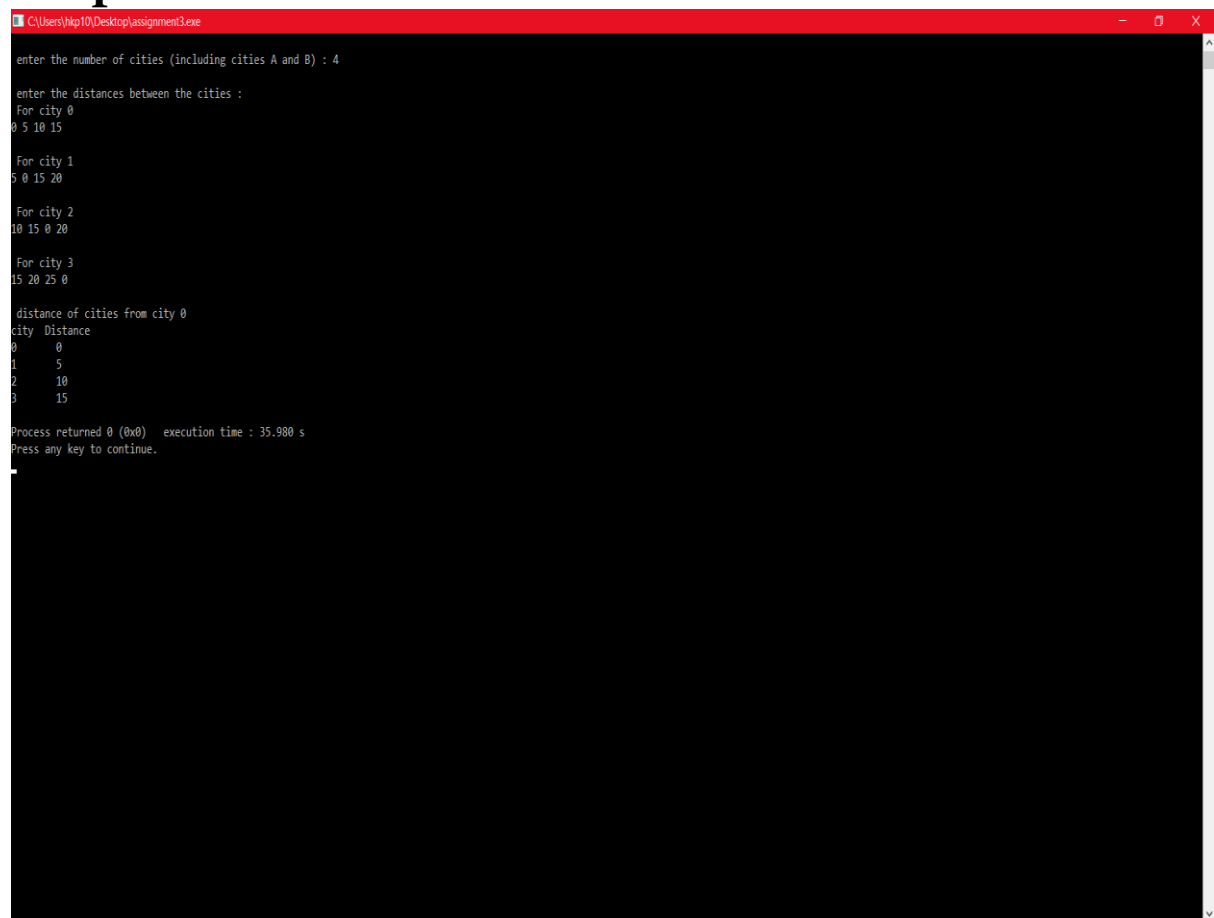
```

int main()
{
    dijkstra sp;
    sp.city_no();
}

```

```
    sp.matrix_fill();  
    sp.dijkstra_code();  
    sp.display();  
    return 0;  
}
```

## Output:



```
C:\Users\hp10\Desktop\assignment3.exe  
enter the number of cities (including cities A and B) : 4  
enter the distances between the cities :  
For city 0  
0 5 10 15  
For city 1  
5 0 15 20  
For city 2  
10 15 0 20  
For city 3  
15 20 25 0  
distance of cities from city 0  
city Distance  
0 0  
1 5  
2 10  
3 15  
Process returned 0 (0x0)   execution time : 35.980 s  
Press any key to continue.
```

## Conclusion:

From above experiment we learnt how to use shortest path algorithm using graph operation.