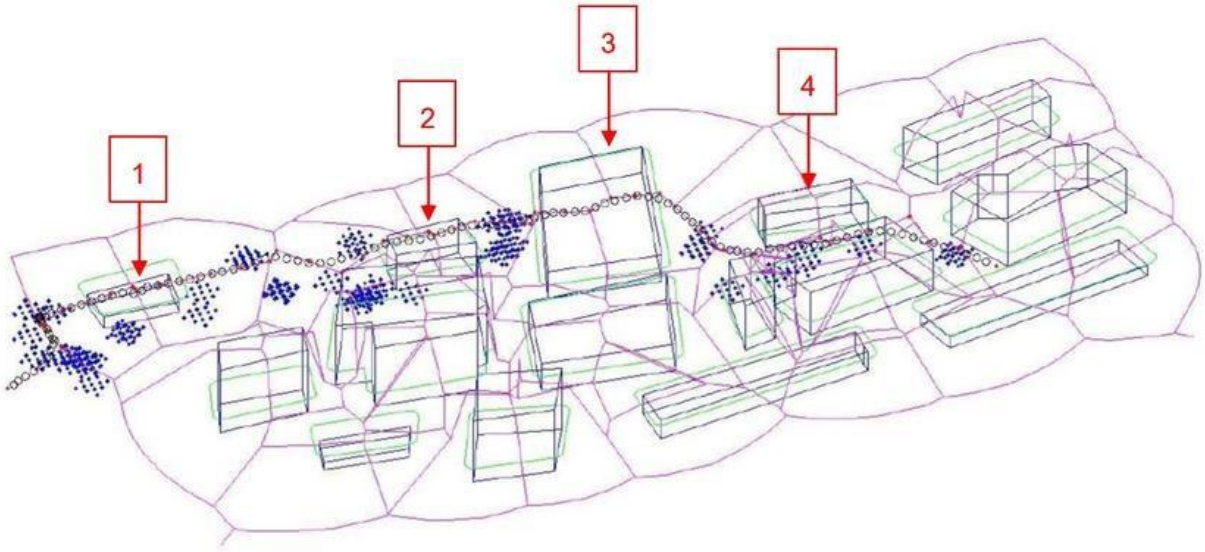
Problem Statement:

The problem of organizing inter-city flights is one of the most important challenges facing airplanes and how to transport passengers and commercial goods between large cities in less time and at a lower cost. In this report, we have tried to implement the Dijkstra algorithm to solve this complex problem and also to update it to see the shortest-route from the origin node (city) to the destination node (other cities) in less time and cost for flights using simulation environment. Such as, when graph nodes describe cities and edge route costs represent driving distances between cities that are linked with the direct road.



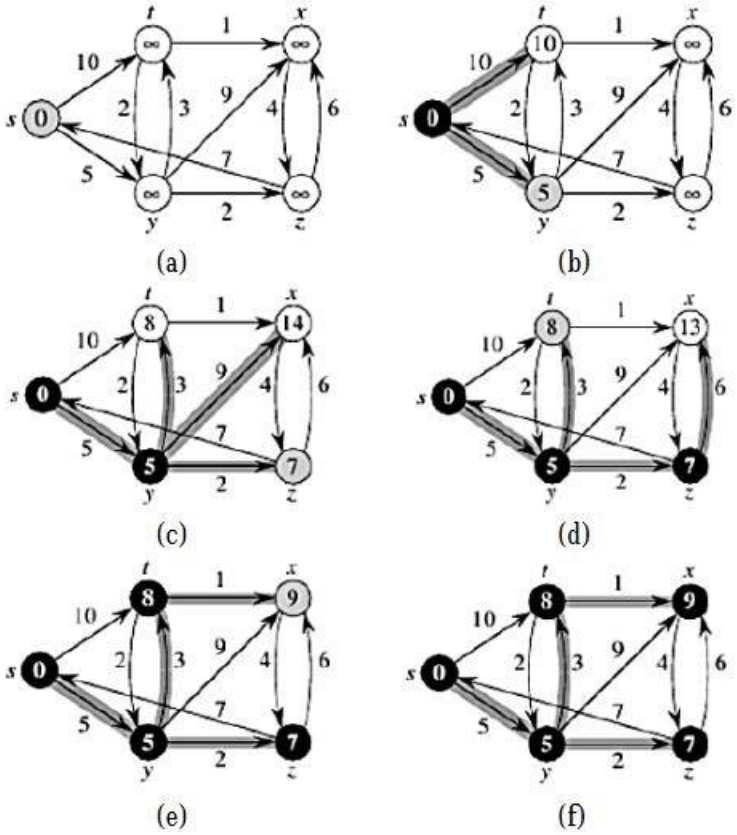
Problem Description:

We are given information about a set of flights between different cities. You want to go from a city A to a city B and would like to find out the earliest time by which you can reach B. The cities have names which can be strings, for example, “Delhi”, “Mumbai”,etc.

Each flight is described by a flight number, which is again a string (e.g., “AI 102”), source city, destination city, departure time, arrival time. Assume that no flight crosses midnight, i.e., departure time is after midnight and arrival time is before

midnight. A query will specify a source city, a destination city and a departure time. For example, it can specify source city as “Delhi”, destination city as “Mumbai” and departure time as 10:00 (assume that all times are specified in 24 Hour format, i.e. 00:00 to 23:59). There will be no two cities with the same name. Your goal is to find a sequence of flights starting from the source city (“Delhi” in this example), with the first flight starting at or after 10:00 and reaching Mumbai as early as possible.
Note that the trip could involve multiple flights. Also assume that you always want to reach the destination the same day (i.e., before midnight), if it is not possible to do so, should print an error message.

Method : DIJKSTRA’s Algorithm
Dijkstra is an algorithm applied for search and it provides a shorter-route from single-source for a graph with nonnegative edge route costs. It depends on the close road to explain the problem of the single source-shortest route. It keeps the nearest way to source on the graph form source (s) to destination (v).
Dijkstra's algorithm aims to define the shortest routes for a given starting node. When choosing the node for the shortest-route, the algorithm follows a so-called greedy strategy. The node that is closest to the starting node is always chosen. With the Dijkstra algorithm, all edge weightings must therefore be non-negative.



Constraints :

There is one main complication. You must make sure that the arrival time of a flight in this trip is before the departure time of the next flight in the trip, plus a layover time. Ideal layover time is 2 hours or more. But, if cannot find an itinerary with 2+ hours layover then you may relax the problem to have a layover time 1.5 hours or more. If you still cannot find an itinerary with 1.5 hours layover, then you may relax the problem further with 1 hour or more layover. No further relaxations are permitted.

Note that for our problem (not for real world) "two 1.5+ hour but less than 2 hour layovers" and a combination of "one 1.5+ hour layover and one 2 hour layover" will be considered equivalent with respect to the layover constraint. Of these two itineraries should output the one that reaches the destination the soonest.

Dijkstra's algorithm works on the principle of relaxation where an approximation of the accurate distance is steadily displaced by more suitable values until the shortest distance is achieved. Also, the estimated distance to every node is always an overvalue of the true distance and is generally substituted by the least of its previous value with the distance of a recently determined path.

Algorithm:

1) Create a set sptSet (shortest path tree set) that keeps track of vertices included in the shortest- path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty.

2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.
3) While sptSet doesn't include all vertices
    a) Pick a vertex u which is not there in sptSet and has a minimum distance value.
    b) Include u to spt Set.
4) Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if the sum of distance value of u(from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

Example:

Suppose there are four cities: S, T, U, and V.

There is a flight named A1 from S to T with departure time = 1:00 and arrival time = 6:00

There is a flight named A2 from S to U with departure time = 1:00 and arrival time = 2:00

There is a flight named A3 from T to V with departure time = 7:00 and arrival time = 12:00

There is a flight named A4 from T to V with departure time = 8:00 and arrival time = 13:00

There is a flight named A5 from U to T with departure time = 4:00 and arrival time = 5:00

Suppose we want the least time consuming trip from S to V with departure time 1:00

There are four possible routes:

1. A1 to A3
2. A1 to A4
3.A2 to A5 to A3
4. A2 to A5 to A4

Of these routes, the first route is invalid since the layover time between two flights is only 1 hour.

Among the other routes, the least time consuming route is A2 to A5 to A3. Now, Suppose we want the least time consuming trip from U to V with departure time 4:30

The possible routes are: "A5 to A4" and "A5 to A3".

However, the departure time of flight A5 is earlier at 4:00. Therefore, both the routes are invalid.

Code:

```
#include<iostream>
#include<climits> using namespace
std;

int miniDist(int distance[], bool Tset[]) // finding minimum distance
{
	int minimum=INT_MAX,ind;

	for(int k=0;k<6;k++)
	{ if(Tset[k]==false && distance[k]<=minimum)
		{ minimum=distance[k];
			ind=k;
		}
	} return
	ind;
}

void DijkstraAlgo(int graph[6][6],int src) // adjacency matrix
{ int distance[6]; // // array to calculate the minimum distance
	bool Tset[6];// boolean array to mark visited and unvisited for each node

	for(int k = 0; k<6; k++)
	{ distance[k] = INT_MAX;
		Tset[k] = false;
	}

distance[src] = 0; // Source vertex distance is set 0

	for(int k = 0; k<6; k++)
	{ int m=miniDist(distance,Tset);
		Tset[m]=true;

		for(int k = 0; k<6; k++)
```

```
            {
                // updating the distance of neighbouring vertex if(!Tset[k] &&
                graph[m][k] && distance[m]!=INT_MAX &&
                distance[m]+graph[m][k]<distance[k])
            distance[k]=distance[m]+graph[m][k]; }
        }
        cout<<"Vertex\t\tDistance from source vertex"<<endl;

        for(int k = 0; k<6; k++)
        { char str=65+k;
            cout<<str<<"\t\t\t"<<distance[k]<<endl; }
}

int main()
{ int graph[6][6]={
            {0, 10, 20, 0, 0, 0},
            {10, 0, 0, 50, 10, 0},
            {23, 0, 0, 20, 30, 0},
            {0, 50,20, 0, 20, 20},
            {0, 10, 30, 20, 0, 10},
            {0, 0, 0, 20, 10, 0}
            };

        DijkstraAlgo(graph,0);
        return 0;
}
```

Output:

Vertex	Distance from source
A	0
B	10
C	20
D	40
E	20
F	30