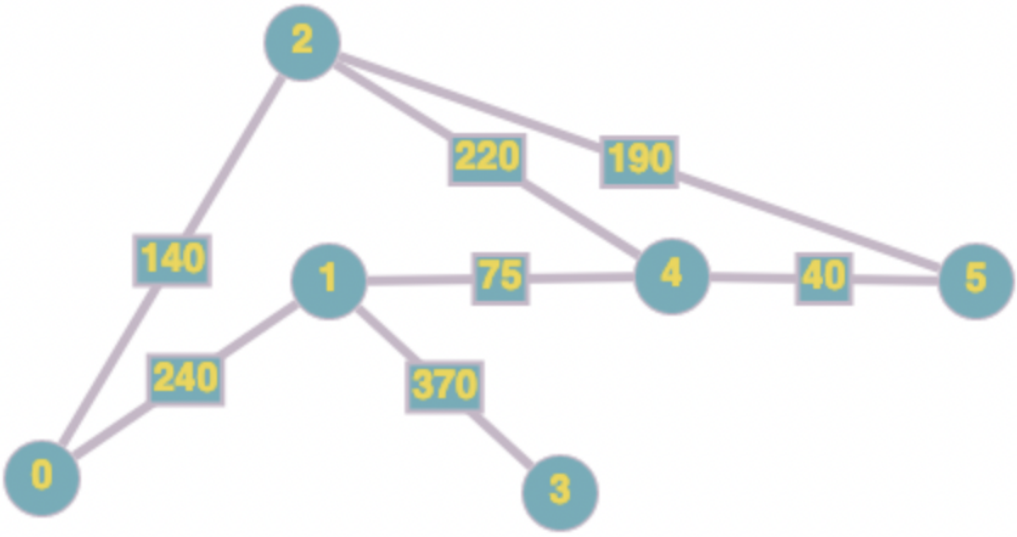
Algorithm Analysis and Reporting

Path Finder

Name: Muhanad
Surname: Tuameh

Algorithm

A program that tries to find all available paths between two points with costs (Time and Price) using the dfs algorithm.



First I used adjacency matrix logic to create the graph.

Time Adjacency Matrix (The time recorded in minutes):

	0	1	2	3	4	5
0	0	240	140	0	0	0
1	240	0	0	370	75	0
2	140	0	0	0	220	190
3	0	370	0	0	430	0
4	0	75	220	430	0	40
5	0	0	190	0	40	0

If there is a non-zero value in the matrix between two nodes, it means there is a path.

This method is followed to find paths between nodes A and B.

1. A is added to visited nodes
2. A node that is not in the list of visited nodes is selected from the neighbors of the last visited node.
3. Add selected node to the list of visited nodes
4. If the selected node is B, it means a path has been found and is added to the list of paths found.
5. If the selected node not B then return to 2
6. Stop if all nodes are visited

Complexity:

```
void dfs(char *dest, Graph *flights, size_t citiesSize, VisitedStack *visited, Path
*path, char **cities, int *counter){
   size_t destCity, i, currCity = peek(visited);
   destCity = getCityNo(cities, dest, citiesSize); O(V)
   if (flights->adjMat[currCity][destCity].total_time != 0 )
   {
       push(visited, destCity, cities);
       addToPaths(flights, path, visited, *counter); O(V)
       pop(visited);
       *counter = *counter+1;
       return;
   }
   for (i = 0; i < citiesSize; i++) O(V)
   {
       if (!isVisited(i, visited) && flights->adjMat[currCity][i].total_time != 0 )
       {
           push(visited, i, cities);
           dfs(dest, flights, citiesSize, visited, path, cities, counter); O(V!)
       }
   }
   pop(visited);
}
```

The complexity of the main function calculated as follow:
Finding city number: O(V) getCityNo();
Adding the path to the path list: O(V) addToPaths()
Looking at neighbor lists: O(V) for (i = 0; i < citiesSize; i++)
DFS: O(V!)

Time Complexity = O(v) + O(v) + O(v) + O(v!) = O(v!)

Space Complexity:

Because of using the adjacency matrix the space complexity will be O(V^2).

Output of the program:

```
Available Cities
0: Istanbul
1: Berlin
2: Atina
3: Helsinki
4: Paris
5: Londra

Price Adjacency Matrix
    0    1    2    3    4    5
0    0    200    120    0    0    0
1    200    0    0    250    100    0
2    120    0    0    0    200    175
3    0    250    0    0    300    0
4    0    100    200    300    0    100
5    0    0    175    0    100    0

Time Adjacency Matrix(In minutes)
    0    1    2    3    4    5
0    0    240    140    0    0    0
1    240    0    0    370    75    0
2    140    0    0    0    220    190
3    0    370    0    0    430    0
4    0    75    220    430    0    40
5    0    0    190    0    40    0

Enter Source: Istanbul
Enter Destination: Londra
Istanbul pushed to visited stack

currentCity: Istanbul, destCity: Londra
Path Found Between Istanbul and Berlin
Berlin pushed to visited stack

currentCity: Berlin, destCity: Londra
Path Found Between Berlin and Helsinki
Helsinki pushed to visited stack

currentCity: Helsinki, destCity: Londra
Path Found Between Helsinki and Paris
Paris pushed to visited stack

currentCity: Paris, destCity: Londra
Londra pushed to visited stack
Path Found Between Berlin and Paris
```

```
Paris pushed to visited stack

currentCity: Paris, destCity: Londra
Londra pushed to visited stack
Path Found Between Istanbul and Atina
Atina pushed to visited stack

currentCity: Atina, destCity: Londra
Londra pushed to visited stack

Paths Found:
Paths 1:
Istanbul Berlin Helsinki Paris Londra
price: 850
hours: 17
minutes: 60
transit: 4

Paths 2:
Istanbul Berlin Paris Londra
price: 400
hours: 5
minutes: 55
transit: 3

Paths 3:
Istanbul Atina Londra
price: 295
hours: 5
minutes: 30
transit: 2
```