MIDTERM REPORT

On

SHORTEST PATH ALGORITHMS FOR OPTIMIZATION OF FLIGHT ROUTES

Submitted by

Name	Roll No	Branch
Aniruddh Saxena	R110218023	CSE CCVT
Dishant Bisht	R110218051	CSE CCVT
Harshit Joshi	R110218061	CSE CCVT
Manya Aeron	R110218091	CSE CCVT

Under the guidance of

Mr. Sandeep Pratap Singh
Department of Virtualization

SCHOOL OF COMPUTER SCIENCE
UNIVERSITY OF PETROLEUM & ENERGY STUDIES
Bidholi Campus, Energy Acres, Dehradun – 248007

School of Computer Science

University of Petroleum & Energy Studies, Dehradun

Minor I

PROJECT TITLE

Shortest path algorithms for optimization of flight routes

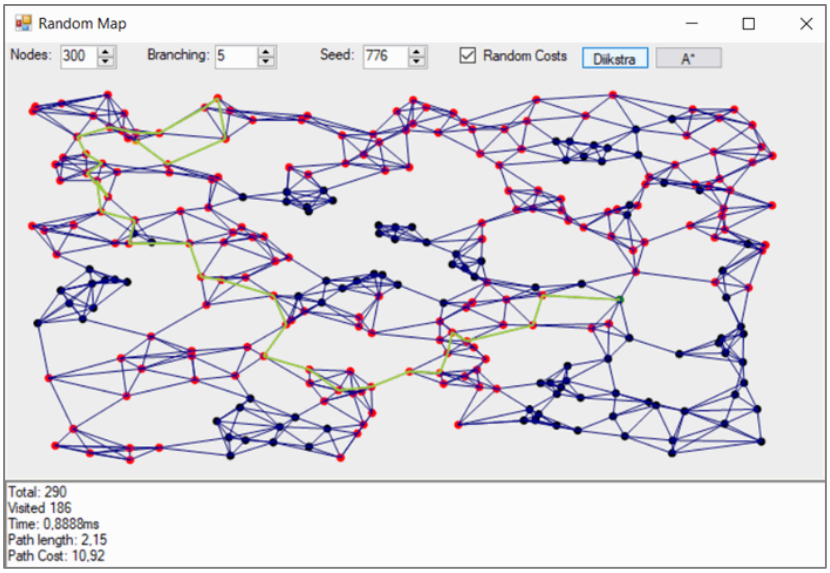
ABSTRACT

This project aims at the implementation, calculation and comparison of the shortest path algorithms to optimize the air route network system. When given a flight network, which is defined by an origin and destination (OD) pair, the problem lies in searching routes that connect the two ends of the pair and provide an efficient, profitable path and satisfy the constraint of time, distance or cost. Some of the algorithms developed to solve this problem are Dijkstra's algorithm, bellman-ford algorithm, A* algorithm and Floyd-Warshall algorithm. With these graph-based approaches, we are able to efficiently compute a set of best connections over space to provide the best suitable route for airplanes to travel. Throughout this project, we will be able to compare different algorithms and evaluate the best optimized air route which will meet the real-time constraints of dealing with the airline route management system.

INTRODUCTION

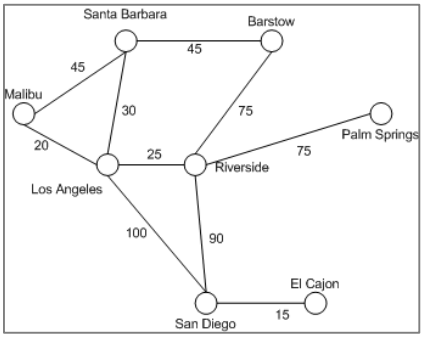
Over the last few years, the aviation sector has been evolving rapidly. Flights being the fastest mode of travelling, has become the prime choice of people to travel across the globe. This comes along with a lot of challenges for the airplane management system to overcome and many expectations to fulfill. In order to achieve all this, it should focus on the route network management system. A route is a designed path for different flights that begins at the origin airport and ends at the destination airport. Flights choice are governed by factors like- arrival or departure time and airplane type. A non-stop flight is a single flight with no stops in between. A connecting flight is a flight where passengers are required to change the plane in between the origin and destination route at a particular stop.

This project is designed in order to provide the best feasible flight route for passengers as per their choice of flight type, source-destination preferences. This would be achieved by comparing different graph based shortest path algorithms and choose the best out of them suited according to the provided case on the basis of complexity, duration and distance. The graph's nodes act as the airports, edges as flights between them, and the path connecting our source to our destination will be calculated.



Dijkstra's algorithm:

Dijkstra's algorithm is used for finding the shortest path from a starting node to a target node in a weighted graph and creates a tree of shortest paths from the source node, to all the remaining nodes in the graph. It finds the minimum value of all the vertices at once and estimates the optimal trajectory in minimal time. It is based on the principle of solving the shortest path problem between the initial point A and the destination point B. This approach evaluates and discards undesirable sub-trajectories until it finds the optimal path. The time complexity of Dijkstra is O(VLogV).

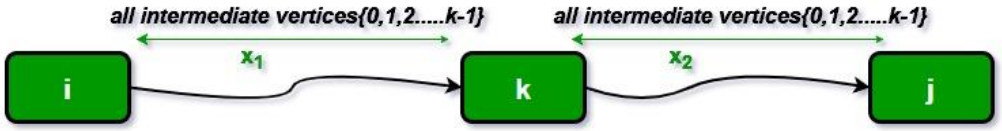


Bellman-Ford algorithm:

In Bellman-Ford algorithm also known as correcting shortest path algorithm, edges are considered one by one, unlike Dijkstra's algorithm. It is slower than Dijkstra's but even more flexible. It is also applicable to negative edges. For e.g. If there exist different ways to reach from one chemical A to another chemical B, each process will have sub-reactions involving both heat dissipation and absorption. So the release of energy is what comes under negative edges. It iteratively relaxes the earlier estimates by finding new paths that are shorter than the previously calculated paths. Time complexity of Bellman Ford is O(VE).

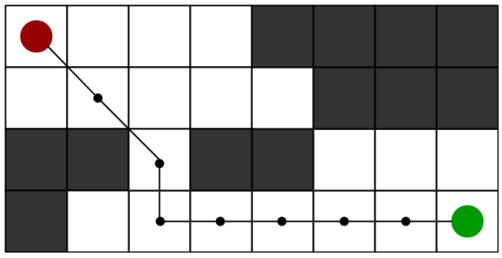
Floyd-Warshall's algorithm:

The Floyd-Warshall's algorithm is used for solving the All Pairs Shortest Path problem. It finds the shortest distances between every pair of vertices in the given weighted directed Graph. The logic is to one by one choose all vertices and update all shortest paths which include the chosen vertex as an intermediate vertex in the shortest path. As a result of this algorithm, a matrix is generated, which will show the least distance from any node to all other nodes in the graph. The time complexity of this algorithm is O (V3), where V is the number of vertices in the graph.



A* Search algorithm:

A* Search algorithm is one of the best approaches used in path-finding and graph traversal. It achieve optimality and completeness, two important properties required for algorithms. It is used to calculate the shortest distance between the source and the destination. Suppose there is a square grid which has many obstacles, scattered. The initial and the final position is provided. Now in order to reach the final position in the shortest amount of time, A* Search Algorithm comes into play. But A* is slow which makes other faster algorithms have an upper hand over it but it is still, one of the best algorithms. It's worse case time complexity is O(E), where E is the number of edges in the graph.



LITERATURE OVERVIEW

Title	Link	Author	Remarks
Understanding the aspect of Algorithm	https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/	Omar Khaleed	This page provides us with a detailed description of the Shortest Path Algorithm i.e. finding the path between two vertices in a graph such that the weight of the graph plotted is minimum.
Working on Algorithms in Detail	https://brilliant.org/wiki/shortest-path-algorithms/	Alex Chumbley	This page explains the idea of the need of the Shortest path Algorithm, comparison between the Algorithms in terms of time-space complexity and effectiveness. Also we referred to the pseudo-codes of different algorithms to structure our code.
Design and implementation of Shortest path to find Air Routes	https://dl.acm.org/doi/pdf/10.5555/1791129.1791184?download=true	James D. Teresco	This research paper discusses the implementation of Dijkstra's and A* algorithm on Google maps to find the shortest Air routes between two destinations. Ideally the graph data used needs to be small enough to be manageable and large enough to be interesting. This paper helps us in touching every aspect in theoretical model.
How to implement Shortest Path algorithm into programmable codes	https://www.researchgate.net/publication/310594546_A_Review_and_Evaluations_of_Shortest_Path_Algorithms.	Magzhan Kairanbay	This publication from Research Gate helps giving insight on Shortest Path Algorithms so that we can understand the implementation of algorithm and mould it into our needs

Theoretical Aspect of Air Route management system	https://www.tutorialspoint.com/aviation_management/aviation_management_airline_route_planning.htm	Akshat Sinha	This Tutorial point article continues to widens the research on Airline route management, discussing in detail every aspect of route management along with precautionary measures.

PROBLEM STATEMENT

Airline specialists have always found it difficult to evaluate the shortest path between the origin and destination. Given two points that i.e. origin and destination (OD), the problem lies in searching routes that connect the two ends of the pair and provide an efficient, profitable path and satisfy the constraint of time, distance, cost. To tackle this problem, we have come with a solution that will tell us the shortest path between two points by different shortest path algorithms and would also compare them based on time complexity and for a matter of fact financially too. This model will break the conventional system of finding routes and will be of great help when you have to find routes when it includes a single stop or multiple steps between origin and destination.

OBJECTIVES

To start with, our main objective is to make a model that tells us the shortest path between a pair of origin and destination (OD) by using different shortest path algorithms hence making it useful to optimize the air route network system.
Sub-objectives are as follows:

1. Intensive study of different shortest path algorithm to calculate distance.
2. Comparison of different algorithms based on complexity, distance ad cost.
3. To get the optimum distance between origin and distance if it contains a stop or multiple stops.

MEHODOLOGY

The entire implementation of this project can be summarized into the following steps:

1. Understanding of Shortest Path Algorithm - The shortest path problem is about finding a path between 2 vertices in a graph such that the total sum of the edges weights is minimum. The shortest Path Algorithms are defined for various types of graphs, whether directed, undirected or mixed. The most important algorithm to solve the Shortest Path problems are:

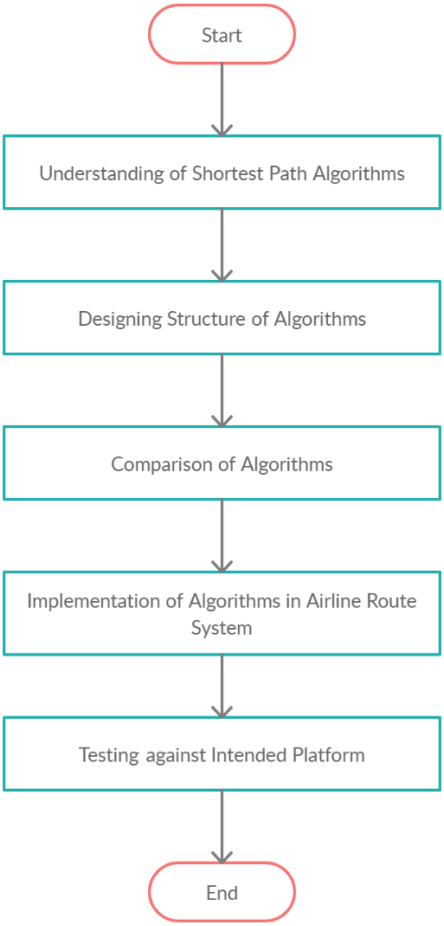
a) Dijkstra Algorithm
b) Bellman Ford Algorithm
c) A* Algorithm
d) Floyd-Warshall Algorithm

2. Designing structure of Algorithm- The working and inclusion of all the above-mentioned algorithms needs to be understood and its structure should be designed. We will be using C language to support our intended design into a working code. Every algorithm has its own individuality and space-time complexity, which brings us to our next step.

3. Comparison of Algorithms – Using the modules and library in use we will make sure to compare all the Shortest Path algorithms cleanly, analytically and present an in-depth report of the following. It will help us in understand the concept of space-time complexity even better and also help us to determine to find out the use of algorithms according to its need as every set of algorithm has its own advantages and disadvantages.

4. Airline Route Management system - Implementation of an Algorithm is a necessary step for a developer to jump from a theoretical aspect to practical knowledge. Here, we will incorporate a shortest path algorithm to build an Airline route management system. Our system will help determining the safest and shortest path (either based on duration or based on distance) between two venues.

5. Testing against intended platform- Linux terminal-based implementation shall reveal the loopholes and unidentified services in the usage and fixing of errors.



SYSTEM REQUIREMENTS (HARDWARE/SOFTWARE)

Hardware:

- RAM: 4GB
- Disk Space: 4GB

Compiler:

- GNU Compiler Collection (GCC)

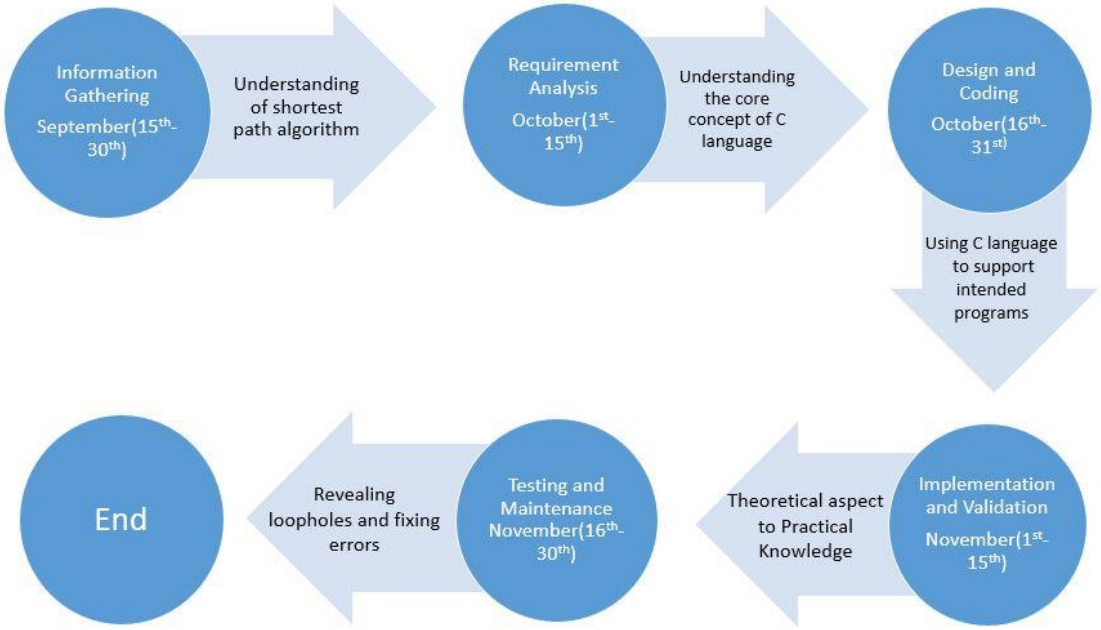
Operating System:

- Linux/Windows

IDE:

- Eclipse C++

SCHEDULE (PERT CHART)



CODE SNIPPETS

1) Dijkstra Algorithm

dijkstra.h > dijkstra(int [V][V], int, int, int)

```
void dijkstra(int G[V][V],int n,int src,int dest)
{

    int cost[V][V],distance[V],pred[V];
    int visited[V],count,mindistance,nextnode,i,j;

    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            cost[i][j]=G[i][j];

    for(i=0;i<n;i++)
    {
        distance[i]=cost[src][i];
        pred[i]=src;
        visited[i]=0;
    }

    distance[src]=0;
    visited[src]=1;
    count=1;

    while(count<n-1)
    {
        mindistance=INFINITY;

        for(i=0;i<n;i++)
            if(distance[i]<mindistance&&!visited[i])
            {
                mindistance=distance[i];
```

```c
C dijkstra.h > dijkstra(int [V][V], int, int, int)
            if(distance[i]<mindistance&&!visited[i])
            {
                mindistance=distance[i];
                nextnode=i;
            }

            visited[nextnode]=1;
            for(i=0;i<n;i++)
                if(!visited[i])
                    if(mindistance+cost[nextnode][i]<distance[i])
                    {
                        distance[i]=mindistance+cost[nextnode][i];
                        pred[i]=nextnode;
                    }
        count++;
    }


            printf("\nTotal Distance = %d",distance[dest]);
            printf("\nPath \n");
            printf("%d",dest);
            j=dest;
            do
            {
                j=pred[j];
                printf("<-%d",j);
            }while(j!=src);

}
```

2) Bellman Ford Algorithm

```c
C bellmanFord.h > bellmanFord(int [V][V], int, int, int)
void bellmanFord(int G[V][V],int m, int src, int dest)
{
    int i,j,e=0,n,u,v,k,distance[500],parent[500],p=0,flag=1,edge[500][500];

    for(i=0;i<m;i++)
    {
       for(j=0;j<m;j++)
        {
            if(G[i][j]!=0)
                edge[p][0]=i,edge[p++][1]=j;
        }
    }

    for(i=0;i<m;i++)
    {
        distance[i] = INFINITY;
        parent[i] = -1 ;
        distance[src]=0 ;
    }
    for(i=0;i<m-1;i++)
    {
        for(k=0;k<p;k++)
        {
            u = edge[k][0] , v = edge[k][1] ;
            if(distance[u]+G[u][v] < distance[v])
                distance[v] = distance[u] + G[u][v] , parent[v]=u ;
        }
    }
```

```
    for(k=0;k<p;k++)
     {
         u = edge[k][0] , v = edge[k][1] ;
         if(distance[u]+G[u][v] < distance[v])
             flag = 0 ;
     }
    if(flag)
    {
       for(i=0;i<m;i++)
       {
           if (i == dest)
           printf("Vertex %d -> cost = %d Parent = %d \n",i,distance[i],parent[i]+1);
       }
    }
}
```

3) Floyd Warshall Algorithm

```
floydWarshall.h > floydWarshall(int [V][V], int, int, int)
void floydWarshall (int G[V][V],int n,int src,int dest)
{
    int dist[V][V], i, j, k;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            dist[i][j] = G[i][j];

    for (k = 0; k < n; k++)
    {
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < n; j++)
            {
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }

    printf ("\nTotal Distance  = %d", dist[src][dest]);
}
```

4) A* Algorithm

```
aStar.h > aStar(int, int, int)
void aStar(int n,int src,int dest)
{
    VERTICE *curr = (VERTICE*)calloc(1,sizeof(VERTICE));
    curr = headers[src];
    VERTICE *temp = (VERTICE*)calloc(1,sizeof(VERTICE));
    int visited[n];
    int ch[n];

    for(int i = 0 ; i<n ; i++)
    {
        visited[i]=0;
        headers[i]->distance = INFINITY;
        ch[i] = INFINITY;
    }

    printf("Enter Heuristics \n");
    for(int i = 0 ; i<n ; i++)
    {
       scanf("%d", &headers[i]->heuristic);
    }


    visited[curr->name] = 1;
    curr-> distance = 0;
    ch[curr->name] = curr->heuristic;
    int cname = curr->name;
    int cd ;

    while( (cname != dest) && curr != NULL )
```

```c
aStar.h > aStar(int, int, int)
14      }
15
16      printf("Enter Heuristics \n");
17      for(int i = 0 ; i<n ; i++)
18      {
19         scanf("%d", &headers[i]->heuristic);
20      }
21
22
23      visited[curr->name] = 1;
24      curr-> distance = 0;
25      ch[curr->name] = curr->heuristic;
26      int cname = curr->name;
27      int cd ;
28
29      while( (cname != dest) && curr != NULL )
30      {
31          cd = curr->distance;
32
33          for (int i = 0; i < curr->c; ++i)
34          {
35              temp = curr->links[i];
36
37              if (visited[temp->name] == 0 )
38              {
39                  if(ch[temp->name] > (temp->heuristic + curr->distance + curr->weights[i]))
40                  {
41                      ch[temp->name] = temp->heuristic + curr->distance + curr->weights[i];
42                      temp->distance = curr->distance + curr->weights[i];
```

```c
aStar.h > aStar(int, int, int)
46
47              visited[cname] = 1;
48              ch[cname] = INFINITY;
49
50              int min = 0;
51              for(int i = 0 ; i<n ; i++)
52              {
53                  if(ch[i]<ch[min])
54                      min = i;
55              }
56              if (ch[min] == INFINITY)
57                  curr = NULL;
58              else{
59                  curr->next = headers[min];
60                  curr = headers[min];
61                  cname = curr->name;
62              }
63          }
64
65          if(cname = dest){
66              cd = curr->distance;
67              temp = headers[src];
68              printf("\n Total Distance = %d \n", cd );
69              while(temp != NULL){
70                  cname = temp->name;
71                  printf("-> %d " , cname);
72                  temp = temp->next;
73              }
74          }
```

5) Pseudo Graph and Structure for Vertices

psuedoGraph.h > psuedoGraph(int [V][V], int)

```c
void psuedoGraph(int G[V][V],int n)
{
    int i,j,k,count;
    for(i=0;i<n;i++)
    {
        headers[i] = (VERTICE*)calloc(V,sizeof(VERTICE));
        headers[i]->c = 0;
        headers[i]->name = i;
        headers[i]->next = NULL;
    }

    for(i=0;i<n;i++)
    {
        count=0;
        for (j=0;j<n;++j)
        {
            if(G[i][j]!=0 && G[i][j]!=INFINITY)
            {
                headers[i]->weights[count] = G[i][j];
                headers[i]->links[count] = headers[j] ;
                count++;
            }
        }
        headers[i]->c = count;
    }

```

structure.h > Vertice

```c
typedef struct Vertice VERTICE;

struct Vertice
{
    /*
    float longitude;
    float latitude;
    */
    int heuristic;
    int distance;
    int name;
    int c;
    VERTICE *next;
    VERTICE *links[V];
    int weights[V];
};

VERTICE *headers[V];
```

OUTPUTS

1) Input No. of Vertices and Matrix. Size of Matrix is equivalent to no. of vertices

```
Enter no. of vertices:13

Enter the adjacency matrix:
0 7 2 3 0 0 0 0 0 0 0 0 0
7 0 3 0 4 0 0 0 0 0 0 0 0
2 3 0 0 4 0 0 0 1 0 0 0 0
3 0 0 0 0 0 0 0 0 0 0 0 2
0 4 4 0 0 0 5 0 0 0 0 0 0
0 0 0 0 0 0 0 2 0 0 0 5 0
0 0 0 0 5 0 0 0 3 0 0 0 0
0 0 0 0 0 2 0 0 2 0 0 0 0
0 0 1 0 0 0 3 2 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 6 4 4
0 0 0 0 0 0 0 0 0 0 6 0 4 4
0 0 0 0 0 5 0 0 0 4 4 0 0
0 0 0 2 0 0 0 0 0 4 4 0 0

Enter the starting node:0

Enter the destination node:5

Enter Choice
 1. Dijkstra
 2. Bellman Ford
 3. floyd Warshall
 4. A star
 5. Exit
```

2) Choose any of the option given below. We chose Bellman Ford and the output is printed in the terminal

```
Enter Choice
 1. Dijkstra
 2. Bellman Ford
 3. floyd Warshall
 4. A star
 5. Exit
2
Vertex 5 -> cost = 7 Parent = 8
```

3) Output of Dijkstra Algorithm

```
Enter Choice
 1. Dijkstra
 2. Bellman Ford
 3. floyd Warshall
 4. A star
 5. Exit
1

Total Distance = 7
Path
5<-7<-8<-2<-0
```

4) Output of Floyd Warshall Algorithm

```
Enter Choice
 1. Dijkstra
 2. Bellman Ford
 3. floyd Warshall
 4. A star
 5. Exit
3

Total Distance  = 7
```

5) Output of A* Algorithm

```
Enter Choice
 1. Dijkstra
 2. Bellman Ford
 3. floyd Warshall
 4. A star
 5. Exit
4
Enter Heuristics
10 9 7 8 8 0 6 3 6 4 4 3 5

 Total Distance = 7
-> 0 -> 2 -> 8 -> 7 -> 5
```

REFERENCES

1. https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/
2. https://brilliant.org/wiki/shortest-path-algorithms/
3. https://dl.acm.org/doi/pdf/10.5555/1791129.1791184?download=true
4. https://www.researchgate.net/publication/310594546_A_Review_and_Evaluations_of_Shortest_Path_Algorithms
5. https://www.tutorialspoint.com/aviation_management/aviation_management_airline_routeplanning.htm
6. https://towardsdatascience.com/a-star-a-search-algorithm-eb495fb156bb
7. https://www.geeksforgeeks.org/
8. www.tutorialspoint.com
9. http://www.i3s.unice.fr/~kamal/publications/ICORES2018_AKI.pdf

10. https://warwick.ac.uk/fac/sci/dcs/research/em/publications/web-em/07/paperpham.pdf

11. Anany Levitin, Introduction to Design Analysis and Algorithms

12. Steven Skiena. The Algorithm Design Manual

13. Hector Ortega-Arranz, Arturo Gonzalez-Escribano, The Shortest-Path Problem