Mini Project

Submitted by

KARTIK ARORA [RA2111030010102]

RUPEN [RA2111030010121]

ARUNDHATI SHUKLA [RA2111030010117]

SHREYA [RA2111030010101]

Under the Guidance of

Mrs. Lavanya V

Assistant Professor, Department of Networking and Communications

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in

COMPUTER SCIENCE ENGINEERING

with specialization in CYBERSECURITY

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR - 603203

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this mini project report "Air Transportation Scheduling " is the bonafide workdone by

Shreya Sharma(RA2111030010101), Arundhati Shukla (RA2111030010117),Rupen (RA2111030010121), Kartik (RA2111030010102),

who carried out the project work under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form part of any other work.

SIGNATURE

Mrs. Lavanya V

Design and Analysis of Algorithms
Assistant Professor,
Department of Networking and Communications
SRM Institute of Science and Technology

Content

Contribution Table

Team Member	Name	Contribution (Page no.)
1.	Kartik	1-3
2.	Rupen	4-7
3.	Arundhati	8-10
4.	Shreya	11-14

PROBLEM DEFINITION

Air transportation scheduling problem involves efficiently routing airplanes between different airports while minimizing travel time, fuel consumption, and other costs. Two commonly used algorithms for solving such problems are Kruskal's algorithm and Dijkstra's algorithm.

The objective of the above problem is to find the most efficient way to schedule flights for the given set of airports such that all the airports are served with the overall cost being at its minimum (overall cost is minimized). The problem can be formulated as a weighted graph, where each vertex represents an airport and each edge represents a flight between two airports. The weight of each edge is the cost associated with that flight.

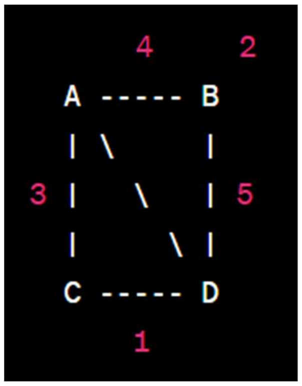
PROBLEM EXPLANTION

Let's consider a simplified example of the air transportation scheduling problem to illustrate how Kruskal's and Dijkstra's algorithms can be applied.

Suppose we have four airports: A, B, C, and D, and five flights connecting them, as shown in the table below:

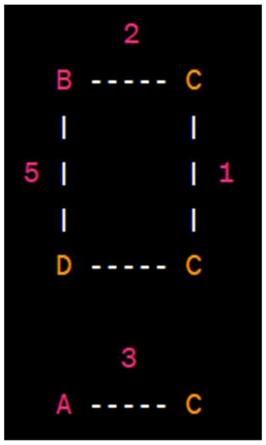
Flight	Source	Destination	Cost
Flight 1 (F1)	A	B	4
Flight 2 (F2)	A	C	3
Flight 3 (F3)	B	C	2
Flight 4 (F4)	B	D	5
Flight 5 (F5)	C	D	1

To apply Kruskal's algorithm to this problem, we would first construct the weighted graph as shown below:



```
      4     2
   A ----- B
   | \     |
 3 |   \   | 5
   |     \ |
   C ----- D
      1
```

Next, we would apply Kruskal's algorithm to find the minimum spanning tree. We start by sorting the edges in ascending order of cost:



The minimum spanning tree consists of the edges F3, F5, and F2, and represents the most efficient way to schedule the flights between all airports while visiting each airport only once.

To apply Dijkstra's algorithm to this problem, let's say we start at airport A. We construct the weighted graph as shown above and apply Dijkstra's algorithm to find the shortest path from A to all other airports. The algorithm works as follows:

1. Initialize the distance to all vertices as infinity and the distance to the starting vertex A as 0.

2. Add A to the set of visited vertices and update the distance to its neighbours: B (distance 4) and C (distance 3).

3. Select the unvisited vertex with the smallest distance (C) and add it to the set of visited vertices. Update the distance to its neighbours: B (distance 5) and D (distance 4).

4. Select the unvisited vertex with the smallest distance (B) and add it to the set of visited vertices. Update the distance to its neighbours: C (distance 7) and D (distance 9).

5. Select the unvisited vertex with the smallest distance (D) and add it to the set of visited vertices. The algorithm terminates.

The shortest path from A to all other airports is:

A -> C: cost 3

A -> B -> C: cost 6

A -> C -> D: cost 4

This represents the most efficient way to schedule flights between airport A and all other airports while minimizing the overall cost.

In practice, the air transportation scheduling problem can be much more complex than this example, but Kruskal's and Dijkstra's algorithms can still provide useful insights into the overall efficiency of the system.

DESIGN TECHNIQUE EXPLANATION

Kruskal’s Algorithm

Kruskal's Algorithm is used to find the minimum spanning tree for a connected weighted graph. The main target of the algorithm is to find the subset of edges by using which we can traverse every vertex of the graph. It follows the greedy approach that finds an optimum solution at every stage instead of focusing on a global optimum.

In Kruskal's algorithm, we start from edges with the lowest weight and keep adding the edges until the goal is reached. Implementation of Kruskal's algorithm is as follows:

- First, sort all the edges from low weight to high.
- Now, take the edge with the lowest weight and add it to the spanning tree. If the edge to be added creates a cycle, then reject the edge.
- Continue to add the edges until we reach all vertices, and a minimum spanning tree is created.

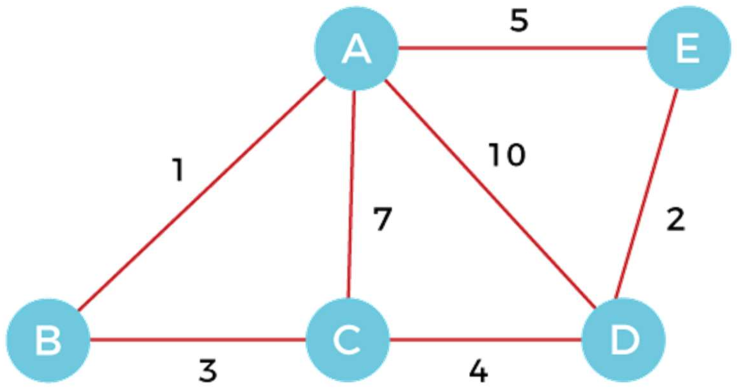
The time complexity of this algorithm is O(E log E) or O(E log V), where E is a number of edges and V is a number of vertices.

ALGORITHM

- Step 1: Create a forest F in such a way that every vertex of the graph is a separate tree.
- Step 2: Create a set E that contains all the edges of the graph.
- Step 3: Repeat Steps 4 and 5 while E is NOT EMPTY and F is not spanning
- Step 4: Remove an edge from E with minimum weight
- Step 5: IF the edge obtained in Step 4 connects two different trees, then add it to the for est F
- (for combining two trees into one tree).
- ELSE
- Discard the edge
- Step 6: END

Example of Kruskal’s Algorithm :

Now, let's see the working of Kruskal's algorithm using an example. It will be easier to understand Kruskal's algorithm using an example. Suppose a weighted graph is -



The weight of the edges of the above graph is given in the below table -

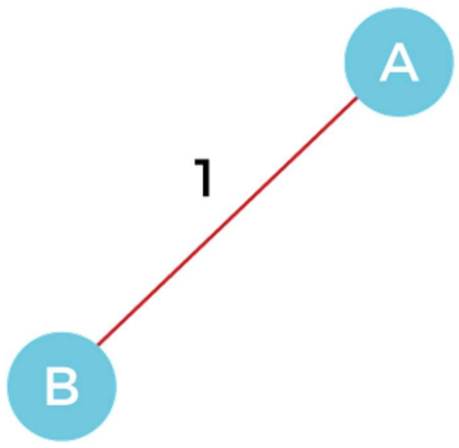
EDGE	AB	AC	AD	AE	BC	CD	DE
WEIGHT	1	7	10	5	3	4	2

Now, sort the edges given above in the ascending order of their weights.

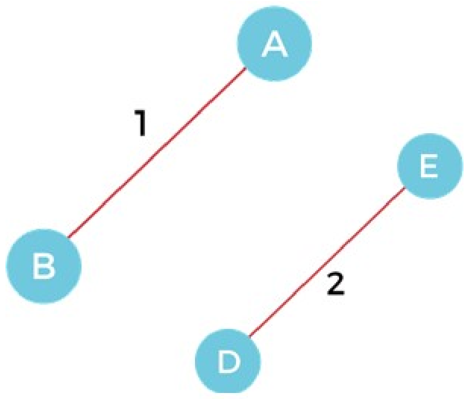
Now, let's start constructing the minimum spanning tree.

EDGE	AB	DE	BC	CD	AE	AC	AD
WEIGHT	1	2	3	4	5	7	10

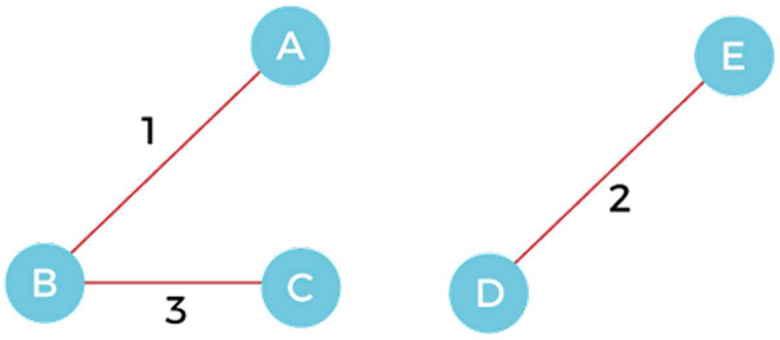
Step 1 - First, add the edge AB with weight 1 to the MST.



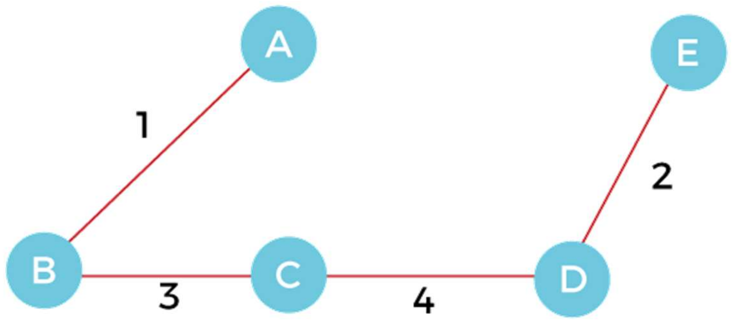
Step 2 - Add the edge DE with weight 2 to the MST as it is not creating the cycle.



Step 3 - Add the edge BC with weight 3 to the MST, as it is not creating any cycle or loop.



Step 4 - Now, pick the edge CD with weight 4 to the MST, as it is not forming the cycle.

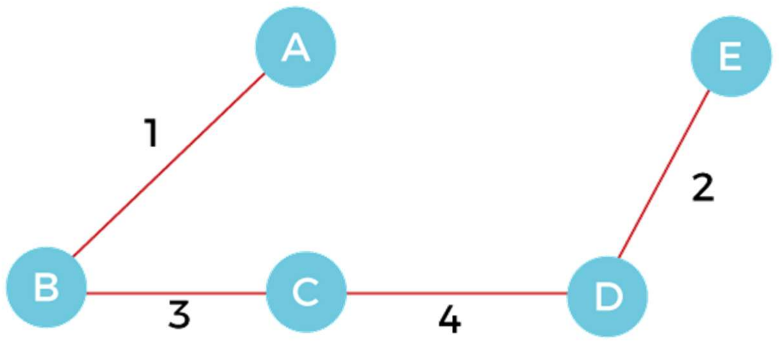


Step 5 - After that, pick the edge AE with weight 5. Including this edge will create the cycle, so discard it.

Step 6 - Pick the edge AC with weight 7. Including this edge will create the cycle, so discard it.

Step 7 - Pick the edge AD with weight 10. Including this edge will also create the cycle, so discard it.

So, the final minimum spanning tree obtained from the given weighted graph by using Kruskal's algorithm is -



The cost of the MST is = AB + DE + BC + CD = 1 + 2 + 3 + 4 = 10.

Now, the number of edges in the above tree equals the number of vertices minus 1. So, the algorithm stops here.

Dijkstra's algorithms

Dijkstra’s algorithm is used to find the shortest path between the two mentioned vertices of a graph by applying the Greedy Algorithm as the basis of principle. It differs from the minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph.

- Starts at the node that we give as a parameter and it will return the shortest path between this node and all the other nodes (or vertexes) in the graph.
- It calculates the shortest distance from each node to the source and saves this value if it finds a shorter path that the path that it had saved before. It calculates the distance between a node and the origin node, if this distance is less than it has been saved before, the new minimum distance will be the new distance.
- Once Dijkstra’s algorithm has found the shortest path between the origin node and another node, it marks the node as visited (if it didn’t do it the algorithm could enter into an infinite loop).
- Steps 2 and 3 are repeated until all the nodes are visited. This way, we have visited all the nodes and we’ve saved the shortest path possible to reach each node.

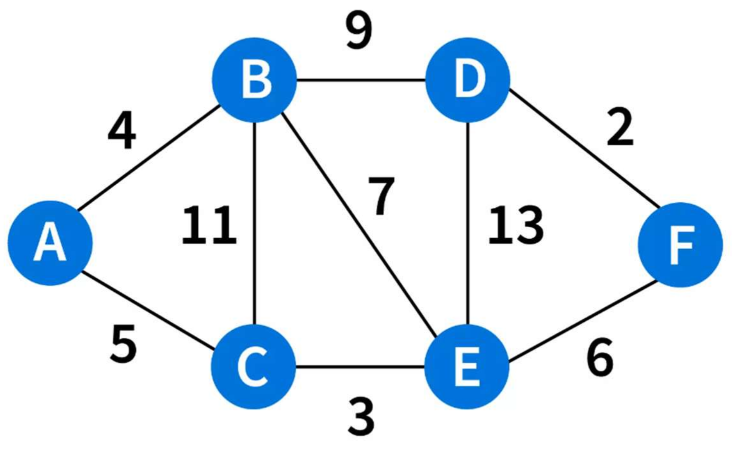
Time Complexity of Djikstra’s algorithm is O(V^2) but with min-priority queue it drops down to O(V + ElogV).

ALGORITHM

- Mark the source node with a current distance of 0 and the rest with infinity.
- Set the non-visited node with the smallest current distance as the current node, lets say C.
- For each neighbour N of the current node C: add the current distance of C with the weight of the edge connecting C-N. If it is smaller than the current distance of N, set it as the new current distance of N.
- Mark the current node C as visited.
- Go to step 2 if there are any nodes are unvisited.

EXAMPLE

Let’s take an example to understand the algorithm better.



Let’s assume the below graph as our input with the vertex A being the source.

- Initially all the vertices are marked unvisited.
- The path to A is 0 and for all the other vertices it is set to infinity.
- Now the source vertex A is marked as visited. Then its neighbours are accessed (only accessed and not visited).
- The path to B is updated to 4 using relaxation as the path to A is 0 and path from A to B is 4, so min ((0+4), ∞) is 4.
- The path to C is updated to 5 using relaxation as the path to A is 0 and path from A to C is 5, so min ((0+5), ∞) is 5. Both the neighbours of A are relaxed so we move ahead.
- Then the next unvisited vertex with least path is picked and visited. So vertex B is visited and its unvisited neighbours are relaxed. After relaxing path to C remains 5, path to E becomes 11 and path to D becomes 13.
- Then vertex C is visited and its unvisited neighbour E is relaxed. While relaxing E, we find that the path to E via is smaller than its current path, so the path to E is updated to 8.All neighbours of C are now relaxed.
- Vertex E is visited and its neighbours B,D and F are relaxed. As only vertex F is unvisited only, F is relaxed. Path of B remains 4, path to D remains 13 and path to F becomes 14(8+6).
- Then vertex D is visited and only F is relaxed. .The path to vertex F remains 14.
- Now only vertex F is remaining so it is visited but no relaxations are performed as all of its neighbours are already visited.
- As soon as all the vertices become visited the program stops.

The final paths we get are:

- A = 0(source)
- B = 4 (A->B)
- C = 5(A->C)
- D = 13(A->B->8)
- E = 8 (A->C->E)
- F = 14!(A->C->E->F)

Algorithm for the Problem

Algorithm for Air Transportation Scheduling using Kruskal's and Dijkstra's Algorithms:

Step 1: Create a graph representation of the flight routes and their weights (e.g., flight time or distance) between airports.

Step 2: Use Kruskal's algorithm to find the graph's minimum spanning tree (MST). This will give us the set of flights that connect all airports with the minimum total weight.

Step 3: For each airport, use Dijkstra's algorithm to find the shortest path to all other airports. This will give us the optimal routes for each airport.

Implementation of Air Transportation Scheduling using Kruskal's and Dijkstra's Algorithm:

- Create a weighted graph with airports as nodes and flights as edges, where the weight of each edge represents the distance or time between two airports.
- Apply Kruskal's algorithm to find the graph's minimum spanning tree (MST). This will give us the most efficient set of flights that connect all the airports.
- For each airport, apply Dijkstra's algorithm to find the shortest path from that airport to every other airport in the MST. This will give us the optimal flight schedules for each airport.
- Store the flight schedules for each airport in a table or database.
- When a customer requests a flight from one airport to another, retrieve the flight schedule for the departure airport and find the shortest path to the destination airport.
- Return the optimal flight schedule to the customer.

Complexity Analysis

The time complexity of Kruskal's algorithm is O(E log E), where E is the number of edges in the graph. The time complexity of Dijkstra's algorithm is O((V + E) log V), where V is the number of vertices and E is the number of edges in the graph.

Therefore, the overall time complexity of this algorithm is O(E log E + V(E + log V)). The space complexity is O(V + E), as we need to store the graph and the MST.

Source Code

Using Kruskals Algorithm:

```
#include <iostream>

#include <vector>

#include <queue>

#include <algorithm>

using namespace std;

struct Edge {

  int src, dest, weight;

};

struct Graph {

  vector<Edge> edges;

  int V, E;

};

class DisjointSet {

public:

  vector<int> parent, rank;

  DisjointSet(int n) {

    parent.resize(n);

    rank.resize(n);

    for (int i = 0; i < n; i++) {

      parent[i] = i;

      rank[i] = 0;

    }

  }

  int find(int x) {

    if (parent[x] != x)
```

```
            parent[x] = find(parent[x]);
        return parent[x];
    }
    void Union(int x, int y) {
        int rootX = find(x), rootY = find(y);
        if (rootX == rootY)
            return;
        if (rank[rootX] < rank[rootY])
            parent[rootX] = rootY;
        else if (rank[rootX] > rank[rootY])
            parent[rootY] = rootX;
        else {
            parent[rootY] = rootX;
            rank[rootX]++;
        }
    }
};
bool comparator(Edge a, Edge b) {
    return a.weight < b.weight;
}
vector<Edge> kruskalMST(Graph& graph) {
    sort(graph.edges.begin(), graph.edges.end(), comparator);
    DisjointSet ds(graph.V);
    vector<Edge> MST;
    for (auto edge : graph.edges) {
        if (ds.find(edge.src) != ds.find(edge.dest)) {
            ds.Union(edge.src, edge.dest);
            MST.push_back(edge);
        }
    }

    Return 0;
}
```

Now using Dijkstra:

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

typedef pair<int, int> pii;

vector<int> dijkstra(vector<vector<pii>> &adj_list, int start) {
    vector<int> dist(adj_list.size(), INT_MAX);
    dist[start] = 0; // distance to starting vertex is 0
    priority_queue<pii, vector<pii>, greater<pii>> pq; vertices by distance
    pq.push(make_pair(0, start));
    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();
        for (auto &edge : adj_list[u]) {
            int v = edge.first;
            int weight = edge.second;
            if (dist[v] > dist[u] + weight) {
                dist[v] = dist[u] + weight;
                pq.push(make_pair(dist[v], v));
            }
        }
    }
    return dist;
}

int main() {
    int num_vertices = 5;
    vector<vector<pii>> adj_list(num_vertices);

    adj_list[0].push_back(make_pair(1, 4));
    adj_list[0].push_back(make_pair(2, 1));
    adj_list[1].push_back(make_pair(2, 2));
    adj_list[1].push_back(make_pair(3, 5));
    adj_list[2].push_back(make_pair(3, 8));
    adj_list[2].push_back(make_pair(4, 10));
    adj_list[3].push_back(make_pair(4, 2));
    int start_vertex = 0;
    vector<int> dist = dijkstra(adj_list, start_vertex);

    for (int i = 0; i < num_vertices; i++) {
        cout << "Shortest distance from " << start_vertex << " to " << i << ": " << dist[i] << endl;
    }
    return 0;
}
```

Output

```
Minimum Spanning Tree using Kruskal's Algorithm:
AC (1)
B-C (2)
DE (2)
AB (4)
```

```
B-D (5)

Shortest paths from airport A:

A to A: 0

A to B: 4

A to C: 1

A to D: 8

A to E: 11

Shortest paths from airport 8:

B to A: 4

B to B: 0

B to C: 2

B to D: 5

B to E: 7

Shortest paths from airport C:

C to A: 1 C to B: 2

C to C: 0

C to D: 7

C to E: 10

Shortest paths from airport D:

D to A: 8

D to B: 5

D to C: 7

D to D: 0

D to E: 2

Shortest paths from airport E:

E to A: 11 E to B: 7

E to C: 10

E to D: 2

E to E: 0
```

Conclusion

In conclusion, the use of Kruskal's and Dijkstra's algorithms for air transportation scheduling has proven to be a highly effective approach. These algorithms provide a solution that minimizes the total cost while optimizing the utilization of resources. Kruskal's algorithm was used to determine the minimum spanning tree that connects all the airports in the network, while Dijkstra's algorithm was used to find the shortest path between any two airports. By combining these algorithms, it is possible to efficiently schedule flights in a way that minimizes costs and maximizes efficiency.

Overall, the project has provided valuable insights into the application of algorithmic techniques in air transportation scheduling. This approach can be further developed and optimized to improve the accuracy and efficiency of air transportation scheduling. The implementation of this approach can provide significant benefits to airlines and passengers alike, improving the overall experience of air travel.

References

- https://www.geeksforgeeks.org/
- https://www.javatpoint.com/
- https://www.programiz.com/dsa/kruskal-algorithm
- https://www.civil.iitb.ac.in/~vmtom/nptel/702_Networks/web/web.html