CrowdStrike Hackathon
Every Second Counts

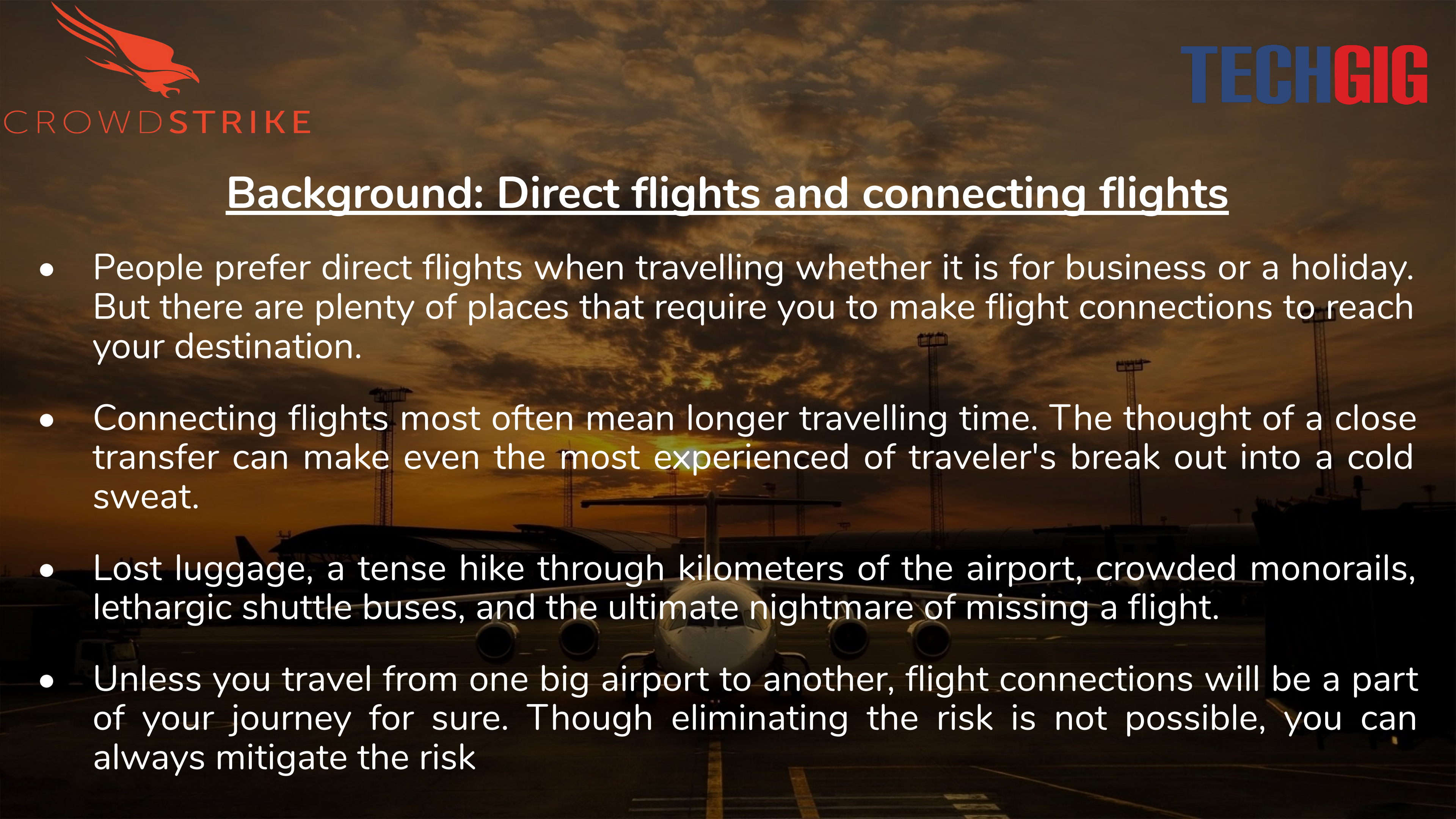
TECHGIG

CROWDSTRIKE

TECHGIG

Team Name : AutoCrats
Theme Name : Hop Count

Team Members : Akash Patel
Parth Shingala
Vishal Makwana

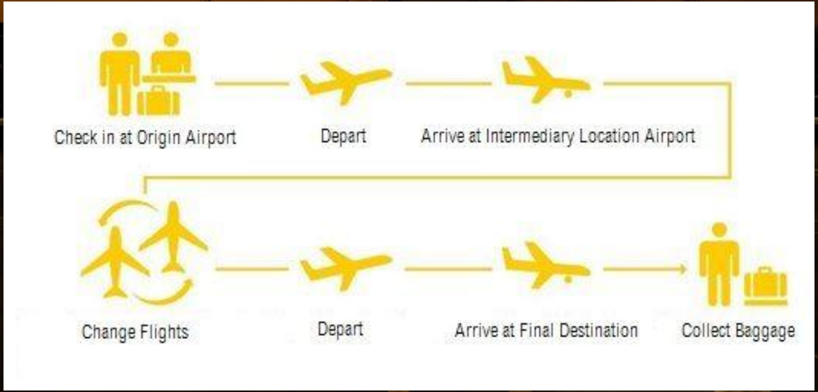


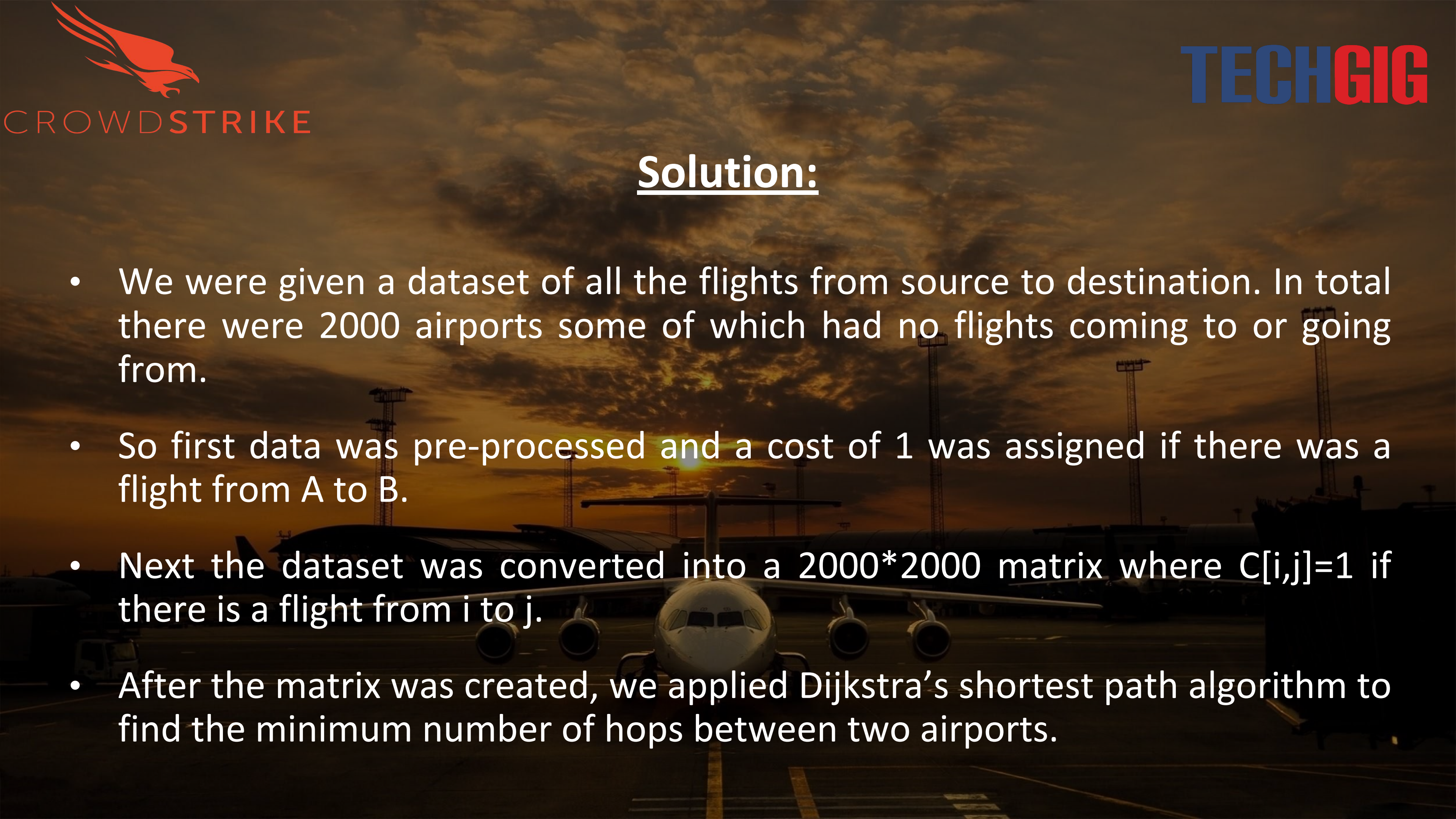
Background: Direct flights and connecting flights

- People prefer direct flights when travelling whether it is for business or a holiday. But there are plenty of places that require you to make flight connections to reach your destination.
- Connecting flights most often mean longer travelling time. The thought of a close transfer can make even the most experienced of traveler's break out into a cold sweat.
- Lost luggage, a tense hike through kilometers of the airport, crowded monorails, lethargic shuttle buses, and the ultimate nightmare of missing a flight.
- Unless you travel from one big airport to another, flight connections will be a part of your journey for sure. Though eliminating the risk is not possible, you can always mitigate the risk

Definition:

- Sometimes it happens that there is no direct flight connecting the two cities and you have to go from source airport to some middle airport and then to the destination airport.
- The problem is to find the minimum number of connecting flights to take to reach from A to B


Check in at Origin Airport
Depart
Arrive at Intermediary Location Airport
Change Flights
Depart
Arrive at Final Destination
Collect Baggage

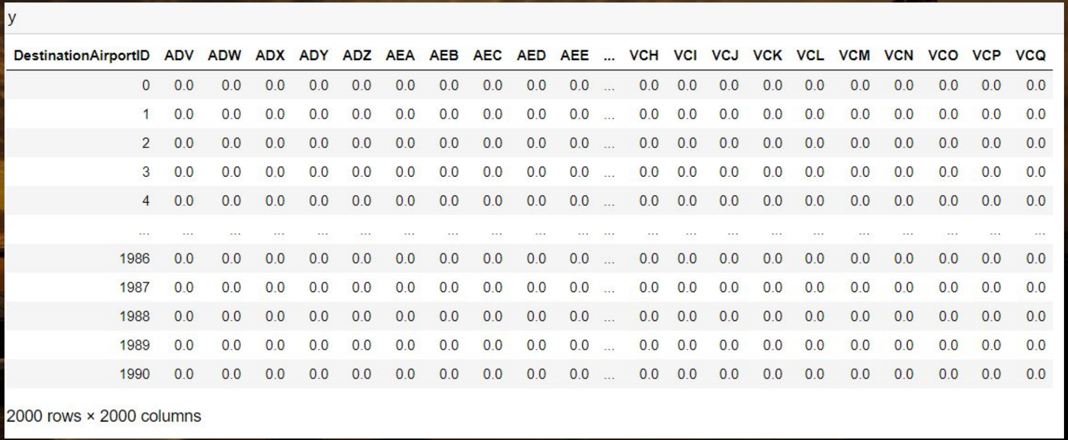


Solution:

- We were given a dataset of all the flights from source to destination. In total there were 2000 airports some of which had no flights coming to or going from.
- So first data was pre-processed and a cost of 1 was assigned if there was a flight from A to B.
- Next the dataset was converted into a 2000*2000 matrix where C[i,j]=1 if there is a flight from i to j.
- After the matrix was created, we applied Dijkstra’s shortest path algorithm to find the minimum number of hops between two airports.

Deliver:

- The 2000 * 2000 matrix which we obtain from the dataset where c[i,j]=1 if there is a flight from i to j.
- This matrix is sent to the Dijkstra Algorithm



y

DestinationAirportID	ADV	ADW	ADX	ADY	ADZ	AEA	AEB	AEC	AED	AEE	...	VCH	VCI	VCJ	VCK	VCL	VCM	VCN	VCO	VCP	VCQ
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1986	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1987	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1988	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1989	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1990	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

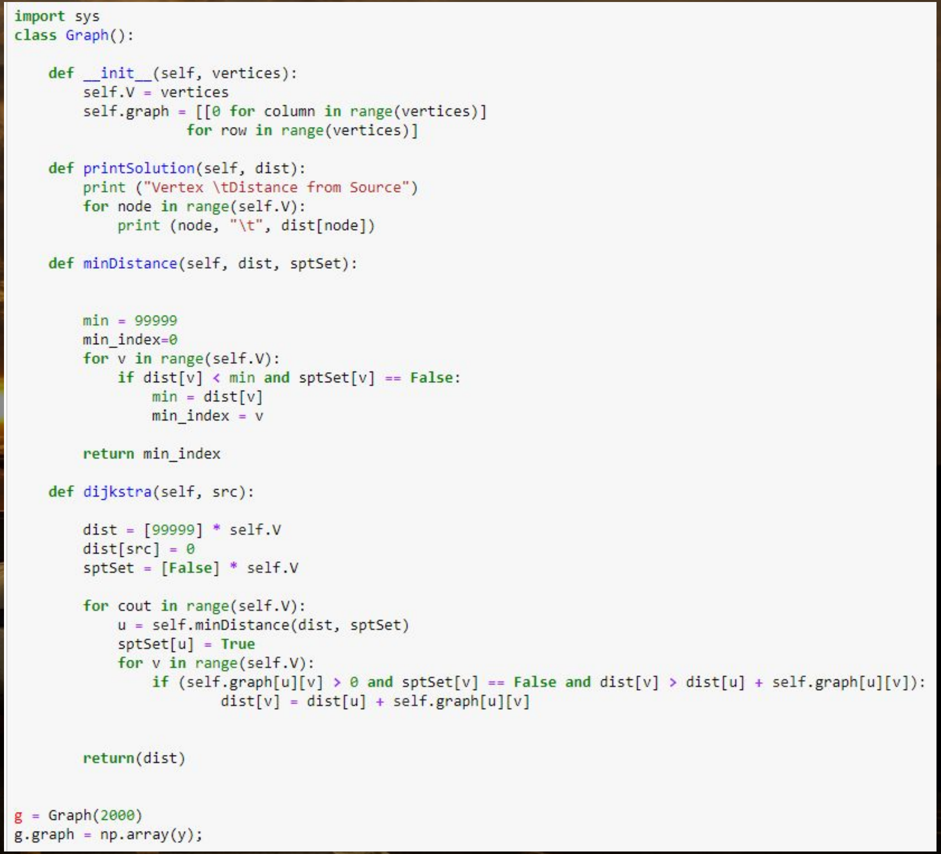
2000 rows × 2000 columns

CROWDSTRIKE

TECHGIG

Dijkstra's Algorithm

- The 2000*2000 matrix obtained from above is passed to the Dijkstra Algorithm and it returns shortest path from all the source to all the destinations



```
import sys
class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    def printSolution(self, dist):
        print ("Vertex \tDistance from Source")
        for node in range(self.V):
            print (node, "\t", dist[node])

    def minDistance(self, dist, sptSet):


        min = 99999
        min_index=0
        for v in range(self.V):
            if dist[v] < min and sptSet[v] == False:
                min = dist[v]
                min_index = v

        return min_index

    def dijkstra(self, src):

        dist = [99999] * self.V
        dist[src] = 0
        sptSet = [False] * self.V

        for cout in range(self.V):
            u = self.minDistance(dist, sptSet)
            sptSet[u] = True
            for v in range(self.V):
                if (self.graph[u][v] > 0 and sptSet[v] == False and dist[v] > dist[u] + self.graph[u][v]):
                        dist[v] = dist[u] + self.graph[u][v]


        return(dist)


g = Graph(2000)
g.graph = np.array(y);
```