

WekeAir

WekeAir

**ANTONIO CADAVID
GONZALO DE VARONA
ESTEBAN YUSUNGUAIRA**

A00354484

A00358687

A00358393

ALGORITHMS AND DATA STRUCTURES

ICESI UNIVERSITY

Phase 1. Problem identification

The young Multimillionaire Antonio Cadavinci was bored and had a bunch of time and money so he decided to create a new airline "WekeAir". He came up with this idea since many of his friends that were rich but not that much in order to buy a private airplane, sometimes struggle when it comes to travel around the world for businesses, due the volatile fares of the other airlines and their unnecessary connections in order to go from A to B.

This airline would be quite different from other airlines, this airplane would be *fair* with its customers and it would only make flights between capital cities from all around Americas. But sometime in the future it would be all around the world with WekeAir's Expansion Plan.

The idea is set, Cadavinci has around 500 airplanes all equipped and ready to fly, an elite team of crew members and pilots, but he does not have an administrative staff. So he decided to spent his spare money (a couple million dollars) in order to hire a team of engineers, who can help him solve his administrative issue. Creating a system to manage the flights booking, consulting prices and checking routes without using much menforce permanently.

Mr. Cadavince has given a list of direct flights and its respective prices to the team:

(Bogotá - Quito 300) (Bogotá - Caracas 250) (Bogotá - ciudad de Panamá 200) (Bogotá - Lima 900) (Bogotá - Asunción 1500) (Quito - Lima 300) (Quito - La Paz 250) (Quito - Ciudad de Panamá 550) (Lima - La Paz 350) (Lima -Brasilia 950) (Lima - Ciudad de Panamá 550) (La Paz - Asunción 350) (La Paz - Brasilia 300) (La Paz - Santiago 400) (La Paz - Buenos Aires 500) (Brasilia - Buenos Aires 300) (Brasilia - Ciudad de Panamá 500) (Brasilia - Montevideo 200) (Buenos Aires - Montevideo 50) (Santiago - Bueno Aires 350) (Santiago - Montevideo 250) (Caracas - Georgetown 100) (Caracas - Paramaribo 500) (Caracas - Cayena 400) (Paramaribo - Cayena 100) (Paramaribo - Georgetown 50) (Ciudad de Panamá - San José 200) (Ciudad de Panamá - Washington 4500) (Ciudad de Panamá - San José 200) (Ciudad de Panamá - La Habana 500) (Ciudad de Panamá - Managua 500) (San José - Managua 100) (San José - Belmopan 300) (San Salvador - Ciudad de México 1000) (San Salvador - Belmopán 50) (San Salvador - La Habana 200) (San Salvador - Ciudad de Guatemala 100) (Ciudad de México - Washington 800) (Ciudad de México - Ottawa 4000) (Ciudad de México - La Habana 800) (Ciudad de México - Ottawa 4000) (Washington - Ottawa 1000)

Mr Cadavinci wants the team to build a system that is able to:

- Pricing information for each selected flight
- Highlight on cheap and fast flights
- Finding the shortest route between capital cities
- Finding the cheapest route between capital cities
- Save and load all purchased flights
- Showing graphically the route between capital cities

Phase 2. Gathering information

The way common airlines set their prices for flights, is not only based on distance between cities, fuel or supplies needed for each flight, it is also based on the type of customer that purchases each trip, season of the year, even on holidays. Having this in mind, airlines know that some flights are bought mainly for business purposes rather than entertainment purposes, which in some cases those flights become more expensive than others, and all the way around, there might be some long flight in which passengers take it for other reasons, making it not as expensive as "business flights". Waiting time is a key factor for pricing flights, the shorter it is, the more expensive the flight. Time is literally *gold* for some travelers, especially in business flights.

Seasons and holidays are way more predictable reasons to influence prices, because at some point of the year, families are on vacation worldwide, or maybe it's Christmas and everyone wants to be with the loved ones, even if they are overseas, or people want to enjoy summer or winter somewhere. This obeys to the law of demand and supply, making sometime of the year having the most expensive tickets or otherwise.

Besides what has been mentioned, airlines price their flights according to taxes or airport fees. An airplane still a business which its main goal is to make the biggest amount possible money so they would not pay extra cash for fees or anything in order to lower prices, they just rather leave that to customers pocket, as a regular business would be.

It is not the same to ride a Ferrari as riding a Corolla, beside the price difference, the two of them can take you anywhere by land, however, the luxury, the commodities and performance make a huge difference, this same situation applies for airlines planes, it is not the same to travel in a 2-floor plane with sofas and TVs inside than traveling of a rough uncomfortable chair with no entertainment system on board, this makes a big difference for some airlines, for instance, look at Emirates and American Airlines planes and compare their insides.

Phase 3. Seek for creative solutions

Idea 1: Use Java libraries to implement built-in data structures like Union Type, Graph or Node in order to model the solution.

Idea 2: Using a Dijkstra based algorithm to find the shortest path between 2 capital cities.

Idea 3: Create our own data structures like Union Type, Graph and Vertex by making them from scratch from their ideal methods for each data structure.

Idea 4: Implement a verification system (login) prior to access the application in order to make security stronger.

Idea 5: Use weighted graphs to model capital cities and the distance between each city.

Idea 6: Using a Kruskal based algorithm to find the shortest path between 2 capital cities.

Idea 7: Use BFS algorithm in order to find a route between 2 capital cities with the least number of stops.

Phase 4. Transition from ideas to preliminary designs

Idea 1: *Discarded.* Java libraries can implement what we are looking for, with *Graph*, *Node* or *Union Type* class however it can not be seen how they specifically work, so the whole point of analyzing and manipulating Graphs will make no sense.

Idea 2: Since it is needed to find the shortest path between 2 cities, then Dijkstra can really help us out by getting the shortest path between an origin city and any other city, however if we need just an specific pair of cities, we will have to make a method that uses Dijkstra but also rebuilds the path between the given cities.

Idea 3: Create our data structures from scratch is a very simple and effective way to analyze Graphs, Edges, Vertexes, and Union Types; besides we have total control of how they work, in case we need to do something else with them that the Java libraries do not do. In the other hand, we know how they work and it makes more easy the implementation and it reduces the working time, because we can modify the methods and logic easier.

Idea 4: *Discarded.* Making security stronger is an important topic for applications with sensitive information, however the customer is asking for a solution for a current issue, which does not implies much security, for the expansion plan in a long term, it will be needed.

Idea 5: If one uses weighted graphs to model the problem, the weight of an edge between cities would represent the actual distance between those cities in the “real

world”. That's why this idea will be really useful in the implementation of the solution, due its capability of representing and allowing to model “real world” attributes.

Idea 6: *Discarded.* Using a Kruskal based algorithm to find the shortest path between 2 capital cities is not wrong at all, it can even work with some adjustments and get the job done, however, this is neither the most efficient nor ideal way to find the shortest path, it does not fit perfectly for getting the job done.

Idea 7: Since it is important for customers to take the least number of stops in a flight, then it is important to find direct flights between cities, or with the least number of cities between the origin city and the destination, in that case BFS sounds about right to get this job done since it handles the number of vertexes between an origin an every other vertex in a connected Graph, so it is easy to get the number of stops between to cities using this algorithm.

Phase 5. Evaluation and selection of the best solutions

Item 1: Expansion plan for the airline

[0] The solution won't work if the airline starts its worldwide expansion plan.

[1] The solution slightly works if the airline starts its worldwide expansion plan.

[2] The solution completely works if the airline starts its worldwide expansion plan and does not need a big rework.

Item 2: Ease of implementation

[0] The solution uses operations that are difficult to understand or implement due we've not seen it or worked with it before much.

[1] The solution uses basic operations to solve the problem.

[2] The solution uses basic operations to solve the problem and can be implemented in a compact method.

Item 3: Course objectives

[0] The solution does not focus on developing and analyzing Graphs.

[1] The solution focuses on developing and analyzing Graphs.

[2] Is easy to manipulate, develop and analyze Graphs, in order to solve the problem.

Item 4: Exact solution

[0] The idea gives an approximated solution.

[1] The idea gives an exact solution.

Ideas feasibility table

Idea	Item 1	Item 2	Item 3	Item 4	Total
Idea 2	2	2	2	1	7
Idea 3	1	1	2	1	5
Idea 5	1	1	2	1	5
Idea 7	0	1	3	0	4

Functional Requirements

F.R.1	Show flying fare information
-------	------------------------------

Abstract	The application is able to show the fare of the requested flight. The application will show the different fares for a certain flight according to the route.
Input	-City of departure. -City of destination.
Output	The flight fare information will be shown on the screen.

FR.2	Finding the shortest route between two cities
Abstract	The application is able to find the shortest, therefore the fastest possible route between two different cities.
Input	-City of departure. -City of destination.
Output	Shortest route between two cities will be displayed on the screen.

FR.3	Displaying flying route
Abstract	The application is able to display on the screen the flying route between the city of origin and the city of destination.
Input	-City of departure. -City of destination.
Output	the flying route will be displayed on the screen.

FR.4	Save all purchased flights
Abstract	The application is able to save a purchased flight in a txt file.
Input	
Output	The flight it's now saved on a txt file.

FR.5	Load all purchased flights
Abstract	The application is able to load the purchased flights from a txt file everytime it starts.
Input	
Output	The list of flights is loaded from a txt file.

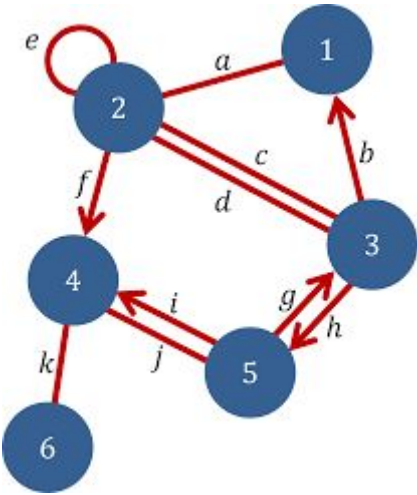
FR.6	Highlighting cheap flights
Abstract	The application is able to highlight the cheapest possible route between two cities.
Input	-City of departure. -City of destination.
Output	the cheapest route is displayed and highlighted on the screen

Non-Functional Requirements

NFR.1	The app must have a friendly user interface
Abstract	The application must be intuitive and easy to use.
Output	

NFR.2	Change graph implementation
Abstract	The application lets the user to change the graph implementation at any moment while using it..
Output	The graph implementation changes from adjacency list to adjacency matrix and viceversa.

ADT

Name	Graph
Abstract representation	 <p>1, 2, 3, 4, 5, 6 = vertex a, b, c, ... , k = edges</p>
Invariant	$\{size \geq 0\} \wedge \{\forall i, j / a \in \text{vertex}, a_1 \neq a_2 \ i \geq 0, j \geq 0, i \neq j\} \wedge$ there is not an order for graphs
Operations	Graph \rightarrow Graph addVertex:vertex addEdge:vertex ₁ , vertex ₂ , weight erase:vertex size \rightarrow integer clear isEmpty \rightarrow boolean dijkstra:vertex bfs:vertex dfs:vertex prim: vertex kruskal floydWarshal:vertex

addVertex:vertex

“Add a vertex to the Graph with no edges”

Pre: vertex \neq NIL

Pos: Add a vertex to the Graph with no edges

addEdge:vertex₁, vertex₂, weight

“Add a weighted edge between two given vertexes in the Graph, in case weight = NIL, then it is not a weighted edge, only a simple edge”

Pre: vertex₁ ≠ NIL , vertex₂ ≠ NIL

Pos: Add an edge between two given vertexes in the Graph

erase: vertex

“Erase a given vertex of the Graph and all of the edges it is connected with ”

Pre: vertex ≠ NIL

Pos: Remove a given vertex and all of the edges it is connected with

size → integer

“Return the size of the Graph”

Pre: Graph.Size ≥ 0

Pos: Returns the number of vertexes of the Graph

isEmpty → boolean

“Return true if the Graph is empty and false otherwise ”

Pre: none

Pos: Returns true if the Graph is empty, otherwise returns false.

clear

“Delete all of the vertexes and edges in the Graph”

Pre: none

Pos: Graph.size = 0

dijkstra:vertex

“Find the shortest path between a given vertex and every other vertex in a weighted and directed Graph”

Pre: vertex \neq NIL, Graph.isDirected = true, Graph.isWeighted = true

Pos: Shortest path between a given vertex and every other vertex in a Graph

bfs:vertex

“Find the shortest path between a given vertex and every other vertex in a non-directed Graph by making a breadth first search”

Pre: vertex \neq NIL, Graph.isDirected = false, Graph.isWeighted = false

Pos: BF Tree which represents the shortest path between a given vertex and every other vertex in a non-directed Graph

dfs:vertex

“Makes a depth first search based of an origin vertex ”

Pre: vertex \neq NIL , Graph.isWeighted = false

Pos: Timestamp for each vertex

prim: vertex

“Find a Minimum Spanning Tree based of an origin vertex”

Pre: Graph.isDirected = false, Graph.isWeighted = true, vertex \neq NIL

Pos: Minimum Spanning Tree based of an origin vertex

kruskal

“Find the Minimum Spanning Tree of the Graph”

Pre: Graph.isDirected = false, Graph.isWeighted = true

Pos: Minimum Spanning Tree

floydWarshall

“Find the minimum distance between each possible pair of vertexes ”

Pre: Graph.isWeighted = true

Pos: Minimum distance between each possible pair of vertexes.

Bibliography

Morris, H. (2016). How airlines set their ticket prices – plus tips to beat them at their own game. Retrieved 2 May 2020, from <https://www.telegraph.co.uk/travel/advice/how-airlines-set-the-price-of-flight-tickets-and-how-to-beat-them/>

Schlick, C. (2013). How Do Airlines Set Prices?. Retrieved 2 May 2020, from <https://flightfox.com/tradecraft/how-do-airlines-set-prices>