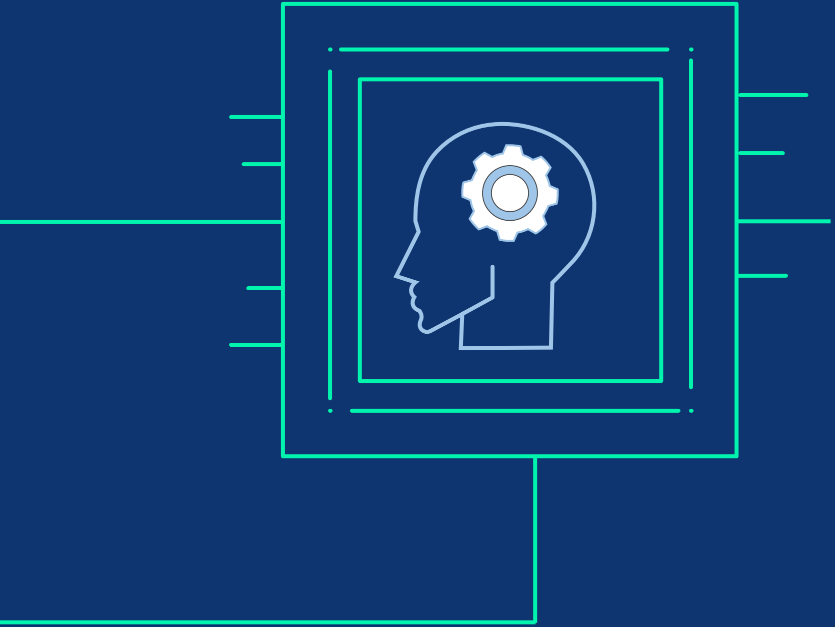
OPEN FLIGHTS FINAL PRESENTATION

Samiksha Arora, Aarya Bhatia, Ananyaa Tanwar



LEADING QUESTION

In this project, we wanted to explore the Open Flights dataset to gain insight into how air travel across nations is optimized. Our objective was to discover and analyze different factors that influence optimal flight paths.

GOALS

- Find the **optimal flight path** between two airports.
  - **BFS** used to find the optimal path based on least connecting flights
  - **Dijkstra's algorithm** used to find the optimal path based on the shortest distance

- **Build a condensed graph** by **discovering the strongly connected components** in the graph and then compressing them into a single component.

- Find the biggest and most popular airports using the **PageRank algorithm**. The PageRank algorithm provides a probability distribution of the likeliness of a person to visit each airport thereby providing insights into the most important airports.

DEVELOPMENT PROCESS

01

DATA PROCESSING

Gathering and parsing data

02

OPTIMAL PATH ALGORITHMS

Dijkstra's algorithm and BFS

03

GRAPH STRUCTURING AND RANKING AIRPORTS

Strongly Connected Components and PageRank algorithm

DATA PROCESSING

DATASET OVERVIEW

- Used the data on Airports and Routes from the OpenFlight dataset.
- Downloaded data in .csv format and saved it in the res directory in our repository

PROCESSING

- Airports dataset is used to create objects of type Airport, which are further used to create vertices of type Vertex<std::string, Airport>
- Routes data is used to create Edges of type Edge<std::string>
- Graph of type Graph<std::string,Airport> is built with the defined vertices and edges
- GraphBuilder class uses parsing functions (defined in util namespace) to construct a graph from input files
- The GraphBuilder initialises the three things- the vertices, the edges, and the distances

OPTIMAL PATH ALGORITHM - BFS

- BFS finds the shortest path between two airports based on least connecting flights
- Shortest path is the one with the least number of vertices.
- We consider all costs of the edges to be equal to 1
- In the bfs function, the parent of the vertices visited in the path are updated to point to the vertex prior to the current vertex in the path.
- Our path function in the graph class can trace the path up to the source with this information.
- The complexity of this algorithm is O(V + E).

INPUT

- source vertex key
- destination vertex key

OUTPUT

- Void function - only manipulates the data inside the vertices.
- Path function returns a vector of keys from the source to destination vertex.
- Returns an empty vector for invalid path

OPTIMAL PATH ALGORITHM - DIJKSTRA'S ALGORITHM

- Dijkstra's Algorithm finds the shortest path between two airports based on the distance between the airports.
- Haversine formula computes the distance between the airports based on latitude and longitude of airport.
- We maintain a priority queue to improve the efficiency of the algorithm.
- The complexity of this algorithm is O(V + ElogV)
- The parent pointers of the vertices visited are updated to point to the vertex prior to the current vertex in the path.
- Our path function in the graph class can trace the path up to the source with this information.

INPUT

- source vertex key
- destination vertex key

OUTPUT

- Also modifies the data of the vertices involved in the path.
- Path function returns a vector of keys with the actual optimal path.
- Returns an empty vector for invalid path

PAGE RANK

- Ranks vertices by their importance.
- Iteration Method
- Initial rank for all vertices = 1/N
- N= number of vertices, PR = Page Rank, d = damping factor (0.85), B(u) = { v | Edge (v,u) exists }.
- Graph must be connected. Let all disconnected vertices have degree N, because there is an equal probability of any vertex to visit vertex v.
- We run this algorithm till all ranks reach a steady state.

$$PR(u) = 1 - d + d \times \Sigma_{v \in B_u} \frac{PR(v)}{degree(v)}$$

$$PR(t+1) - PR(t) < \epsilon \approx 0.05$$

INPUT

- Graph

OUTPUT

- Void function
- Page Rank updates the rank values of the vertices in the graph.
- The *get_ranks* function returns a vector of (rank, key) pairs in the order of greatest to least rank.

STRONGLY CONNECTED COMPONENTS

- Kosaraju’s algorithm used to discover the strongly connected components of the graph.
- Runs two depth first search traversals, once on the given graph and then on the transpose.
- After finding the SCCs, we compress one vertex from each SCC into a new vertex of a condensed graph and insert all the edges accordingly.
- The SCC partitions the graph under the relation that if vertices (u,v) belong to the same SCC v must be reachable by u and vice versa.
- Complexity: O(V+E)

INPUT

- Graph

OUTPUT

- Void function
- Stores the SCCs in a vector called components.
- The *build_condensed_graph* function returns a Graph pointer of the same type as the original graph.

TESTING PROCESS

01

BFS AND DIJKSTRA'S

Checks for both invalid and valid airports and paths

Checks for invalid formatting

Tests edge cases like path outgoing and incoming to the same airport

02

PAGERANK ALGORITHM

Test a small graph with 4 vertices to check for desired output

Test page rank on complete graph to output all airports by rank

03

STRONGLY CONNECTED COMPONENTS

Test if SCC can create a simple graph with char keys from A to F

Test creation of SCC on entire airports graph and print results to file

OVERALL RESULTS

- Successfully loaded data files into a graph, which was then used for algorithm implementation
- Efficiently implemented a traversal (BFS), covered algorithm (Dijkstra’s algorithm) and 2 complex algorithms (PageRank and Strongly Connected components)
- Codebase allows us to find the most optimal path between two airports based on both least connecting flights and shortest distance.
    - Example: for Hangzhou Xiaoshan International Airport (HGH) to Shanghai Hongqiao International Airport (SHA), shortest path is HGH -> LYG -> SHA (via Lianyungang Baitabu Airport - LYG) whereas the least connecting flight path is HGH -> CGO -> SHA (via Zhengzhou Xinzheng International Airport - CGO)
- Similarly, we learned that our graph had 9 strongly connected components with multiple vertices while the other 4470 were singular airports. The biggest cluster of interconnected airports has 3354 airports in it.
- Based on PageRank, we found the most important airport to be Hartsfield–Jackson Atlanta International Airport (ATL) followed by O'Hare International Airport (ORD)

TAKEAWAY

Learned about the real world applications of various diverse algorithms and their impact on air travel. We can use such algorithms to better understand how flight routes are planned and optimized. Additionally, we can also compare how algorithms differ from each other in terms of complexities and efficiencies.

Moreover, we can also think of some use cases where the algorithms we used would need to be modified to better fit the expectations.

IMPROVEMENTS

- We would like to make our project more visual and user friendly in the future.
- A more flexible command line interface would be fun for the users.
- A visualisation of the paths would give us a more clear idea about how our algorithms compare to each other and the real world.
- We would observe if page rank and SCC have similarities because the airports which have more connections should be more reachable, thereby increasing their rank.

CONCLUSION

This project gave us the opportunity to apply skills and concepts gained throughout the semester. We were also able to answer our leading question by to discovering and analyzing different factors that influence optimal flight paths.

THANKS!

TEAM MEMBERS

Aarya Bhatia @aaryab2
Samiksha Arora @arora21
Ananyaa Tanwar @atanwar2