CS225 Final Project

kmajid2, ebaadss2, adnann2, ymohid2

Objectives:

- ➔ Create a graph that is made up of airports and routes
- ➔ Implement a BFS traversal on the graph
- ➔ Use Dijkstra's shortest path algorithm on the graph
- ➔ Find the importance of airports using the Page Rank algorithm

Leading Question:

What is the real-world application of the algorithms that we've learned? For example, how should we map the data to the graph so that our implementation of the Dijkstra's algorithm can help find out the shortest path of two airports? How do we optimize airline routes given different goals?

Data Parsing and Graph Creation

Data collection and parsing

➔ Gathered airport and route info from OpenFlights dataset
➔ Parsed CSV files and stored data in vectors of strings

Creating the graph

➔ Inserted Airport and Flight objects
➔ Used unordered_map to keep track of adjacencies

Testing the graph

➔ Printed out selected airports and their adjacencies

BFS Traversal

- ➔ Used three BFS functions using algorithms discussed in class
- ➔ Used a queue data structure
- ➔ Included in the PageRank algorithm

Testing

- ➔ Used a segment of data; smaller dataset
- ➔ BFS_moves
- ➔ BFS_dest

Results

- ➔ Time Complexity: O(m + n)

```
BFSTraversal(start_node):
  visited := a set to store references to all visited nodes

  queue := a queue to store references to nodes we should visit later
  queue.enqueue(start_node)
  visited.add(start_node)

  while queue is not empty:
    current_node := queue.dequeue()

    process current_node
    # for example, print(current_node.value)

    for neighbor in current_node.neighbors:
      if neighbor is not in visited:
        queue.enqueue(neighbor)
        visited.add(neighbor)
```

Pseudocode for BFS

Dijkstra’s Algorithm

- ➔ Used the process discussed in class
- ➔ Choose a starting point and an endpoint to find the shortest path between them
- ➔ Returns a vector of airports along the path and the distance from start->end

Testing

- ➔ Started with paths between neighbors
- ➔ Compared with public flight distances

Results

- ➔ Time Complexity: O(|E| + |V|log(|V|))
- ➔ Confirmed shortest paths

```
Dijkstra(Graph, source, destination):

  initialize distances  // initialize tentative distance value
  initialize previous   // initialize a map that maps current node
  initialize priority_queue   // initialize the priority queue
  initialize visited

  while the top of priority_queue is not destination:
      get the current_node from priority_queue
      for neighbor in current_node's neighbors and not in visited:
          if update its neighbor's distances:
              previous[neighbor] = current_node
      save current_node into visited

  extract path from previous
  return path and distance
```

Pseudocode for Dijkstra’s

Page Rank Algorithm

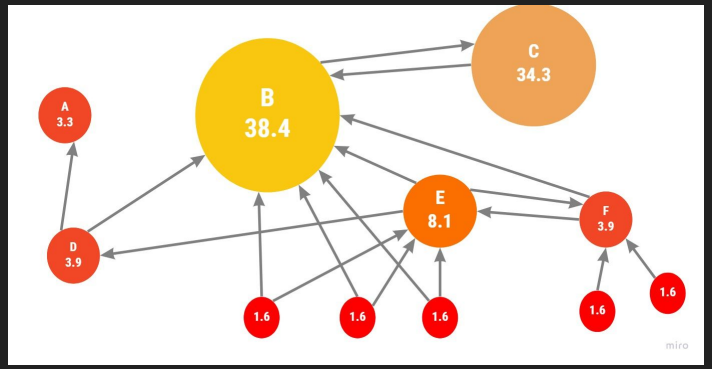
- ➔ Get initial adj matrix by processing a graph object
- ➔ Adjust the matrix with normalization and damping factor
- ➔ Generate a starting vector
- ➔ Output
  - ◆ Vector with the importance of Airports
  - ◆ Most important airports are found with a helper function

Testing

- ➔ Run the algorithm on a small subset of the data
- ➔ Run the algorithm on the full dataset

Results

- ➔ Confirmed with subset of data
- ➔ Normalization increases runtime significantly



Visual example of PageRank

Conclusion and Future Implementations

- ➔ Through this project, we were able to successfully implement our traversal and algorithms
- ➔ Our tests revealed that our graph creation, traversals, and algorithm implementations are accurate
- ➔ We all learned a lot about algorithms like PageRank that we did not have much exposure to beforehand
- ➔ In the future, we could make a user-interactive visual representation of the flights, including connections and distances.
- ➔ Overall, we learned a lot about data structures, writing tests, and the software development process through this project.

Thanks for watching