Final Project: Open Flights

Group members:
Yanxin Jiang, Jin Fan, Xiaoyang Tian, Gezhi Zou

Content

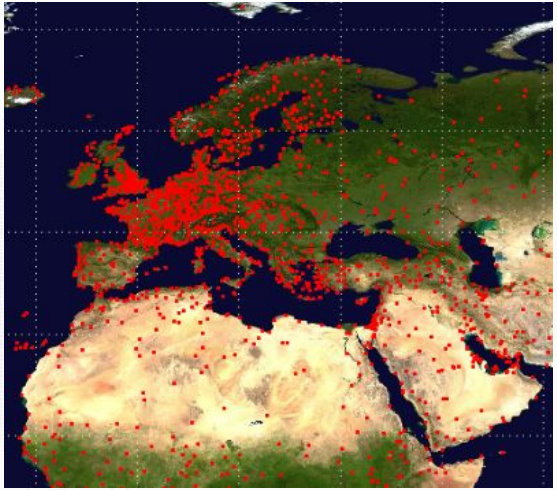
Goals

Development

Conclusion

Goals:

- Parse the data
- Construct the weighted and directed graph
- Write traversal program (BFS)
- Find the compatible path in the given dataset (Dijkstra)
- Find the important airport (Page rank)
- Tests



Development: Data Parsing and Graph Data Structure

Data Parsing:

Download the OpenFlights data set of airports and routes
Parse the character one by one and split the words based on the pattern
Disregard the invalid data

Create the Graph:

Insert airports as vertices and flights as edges
Create the adjacency matrix for the graph

TEST

1. Test #insert vertices
2. Test adjacency matrix for sample data
3. Parse the data set in OpenFlights

```cpp
void Graph::parseEdges(const std::string& filename) {
    std::ifstream Route_File(filename);
    std::string word;
    if (Route_File.is_open()) {

        /* Reads a line from `wordsFile` into `word` until the file ends. */
        // Route(int AirlineID, std::string Airline, int srcID, int dstID, int stop){
        // BA,1355,SIN,3316,LHR,507,,0,744 777
        while (getline(Route_File, word)) {
            // split the words in lines by ","
            std::vector<std::string> v = split(word, ",");
            // if source and destination airport ID aren't found, the route isn't valid
            if(v[3].find("\\N") != std::string::npos || v[5].find("\\N") != std::string::npos){
                // j++;
                invalid++;
                continue;
            }
            // split the lines into airline, airlineID, sourceID, destinationID, and stop
            std::string Airline = v[0];
            std::string AirlineID = v[1];
            std::string srcID = v[3];
            std::string dstID = v[5];
            std::string stop = v[7];
            // if airlineID isn't found, set it to 0 since airline name still exists, the route is still valid.
            int AirlineId = 0;
            if(AirlineID.find("\\N") == std::string::npos) AirlineId = std::stoi(AirlineID);
            // convert string to integer
            int srcId = std::stoi(srcID);
            int dstId = std::stoi(dstID);
            int stop_int = std::stoi(stop);
            // set up the route
            Route route(AirlineId, Airline, srcId, dstId, stop_int);
            // check whether source and destination airport exist. If not, the route is invalid
            if(airports.find(srcId) != airports.end() && airports.find(dstId) != airports.end()){
                Airports[srcId] = airports[srcId];
                Airports[dstId] = airports[dstId];
                insertEdge(route, srcId, dstId);
            }
            else{
                invalid++;
            }

        }

    }
    Route_File.close();
```

```cpp
/**
 * insert a new route as an edge into the graph by adding it into adjacency matrix
 * @param route - the route we want to add in the graph
 * @param srcID - the source airport ID of the route
 * @param dstID - the destination airport ID of the route
 **/
void Graph::insertEdge(Route route, int srcID, int dstID){
    // if srcID not found, initialize the correspdoning value of the adjacency matrix
    if (adjacency_matrix.find(srcID) == adjacency_matrix.end()) {
        adjacency_matrix[srcID] = std::unordered_map<int, Edge>();
    }
    // if srcID found, dstID not found, initialize the edge with the given source and destination airport
    if(adjacency_matrix[srcID].find(dstID) == adjacency_matrix[srcID].end()){
        adjacency_matrix[srcID][dstID] = Edge(route);
    }
    // add route to the existed edge with the given source and destination airport
    else {
        adjacency_matrix[srcID][dstID].addRoute(route);
    }

}
```

Development: Traversal and Find Shortest Pass

For Traversal:

Two BFS functions: O(m+n)

1. **traverseAll: use queue**
2. **traverse_with_dest: use queue**

Output:
vector of string which store traversed airports srcID

TEST

1. BFS # small dataset
2. BFS with dest # small dataset
3. construct graph # real data

```cpp
std::vector<int> BFS::traverseAll(const Graph &graph, int srcID)
{

    std::map<int, bool> visited; // create map to record visited vertices
    vector<int> airports; // the return vector
    std::queue<int> BFS_queue; //create queue for BFS
    matrix_ = airport_graph_.getAdjacency_matrix(); // get unordered map adjacency_matrix of the graph

    // Check whether the source airport is valid. If not, return the empty vector and warn that
    // "Airport isn't found!!!" on the terminal
    if(matrix_.find(srcID) == matrix_.end()) {
        std::cout << "Airport isn't found!!!" << std::endl;
        return airports;
    }

    visited[srcID] = true; //mark index srcId as "visited", change false to true
    BFS_queue.push(srcID);//enqueue the srcID as start
    airport_graph_ = graph; // get graph


    while (!BFS_queue.empty())
    {
        srcID = BFS_queue.front(); //start from the front of the queue
        airports.push_back(srcID); // push the vertex into return vector
        BFS_queue.pop(); // Dequeue this vertex from queue

        for (auto it : matrix_[srcID]) // Get all the adjacent vertices of that dequeued vertex.
        {
            if (visited.find(it.first) == visited.end())
            {
                visited[it.first] = true; //if there are adjacent vertices that haven't been visited, mark it as "visited"
                BFS_queue.push(it.first);//and enqueue it
            }
        }
    }
    std::cout << "the number of airports we traverse is " << airports.size() << std::endl;
    return airports; //return vector
}
```

Development: Traversal and Find Shortest Pass

Find Shortest Pass:

Overview:

1. Choose the start and end airports, find the shortest pass only by the number of airport transfer
2. The output is the airport ID through which the route passes, separated by a space

TEST

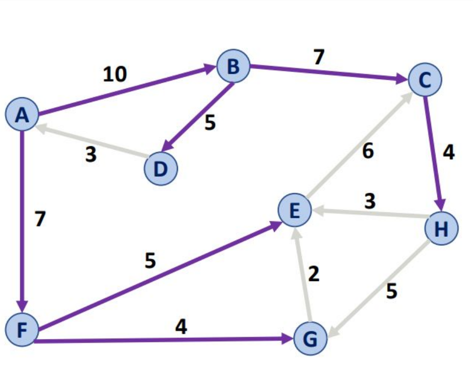
1. TEST bfsshortest step # real data
2. TEST bfsshortest step # small dataset

Development: Dijkstra’s Algorithm

Dijkstra:

Overview:

1. Choose the start and end airports, find the shortest pass by the weight and number of airport transfer
2. The output is the airport ID through which the route passes, separated by a space




```
function Dijkstra(Graph, source):

    for each vertex v in Graph.Vertices:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0

    while Q is not empty:
        u ← vertex in Q with min dist[u]
        remove u from Q

        for each neighbor v of u still in Q:
            alt ← dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```

TEST

1. TEST shortest path # small dataset
2. Find shortest path # real data

Development: Page Rank Algorithm

Goal:

Evaluate the importance of the airports.

Algorithms:

Use the formula above and the adjacency matrix to calculate the pagerank value for each airports.

Output:

Vector that indicates the rank of the importance of airports
The most important airports

Results:

Time complexity is slow while doing the iterations.

```cpp
std::unordered_map<int, double> PageRank::pageRank(const Graph & graph, int time, double damping_factor) {
    graph_ = graph;
    adj_matrix_ = graph_.getAdjacency_matrix();
    airports = graph_.getAirports();
    number_ap = graph_.getAirportNum();
    // initailize the page rank value to 1/size of airports
    for(auto it : airports){
        rank_[it.first] = 1.0 / (double) number_ap;
    }
    // calculate the damping value
    double damping_value = (1.0 - damping_factor);
    // start the iterations for PageRank
    for(int i = 0; i < time; i++){
        // calculate page rank value of airport x
        for(auto x : airports){
            double rank = 0.0;
            for(auto y : adj_matrix_){
                // y.first srcID ; y.second destID Edge
                // x.first destID(we want)
                if(y.second.find(x.first) != y.second.end()) {
                    // find the number of routes from the airport x
                    int deg = getOutDegree(y.first);
                    //          destID              srcID
                    rank += damping_factor * rank_[y.first] / (double) deg;
                }


            rank += damping_value;
            rank_[x.first] = rank;
            }
        }
    }
    return rank_;
}
```

```
elapsed time: 684.358s
3682    3830    1701    3751    3670    4029    1382    340     3364    580     3550
Hartsfield Jackson Atlanta International Airport
===============================================================================
All tests passed (33 assertions in 14 test cases)
```

Rank of the importance of airports by 10 iterations

$$PR(u) = \frac{1-d}{N} + d \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

TEST

1. TEST sample data and get the rank of the importance of airport
2. TEST the subset of data and data set itself from OpenFlights to get the most important airport

Conclusion

Parse data

Read the .csv file

Build directed weighted graph

Find Shortest path

BFS

Dijkstra

Find Important airports

Page Rank Algorithm

Thanks !

THANKS FOR YOUR ATTENTION