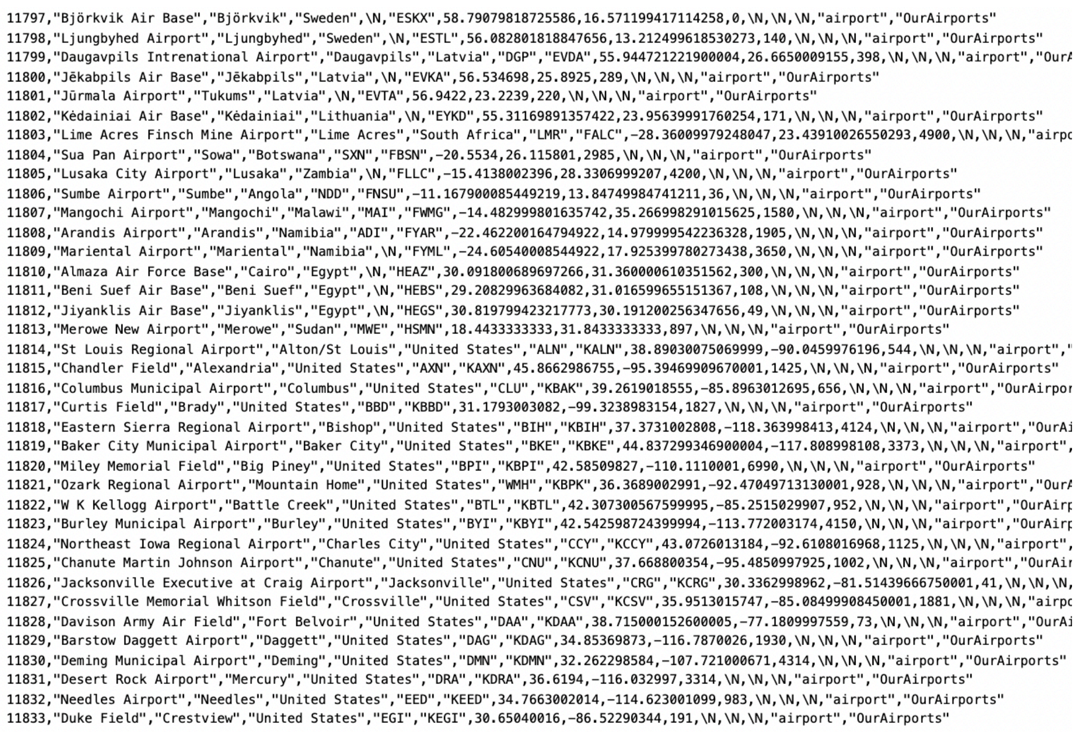
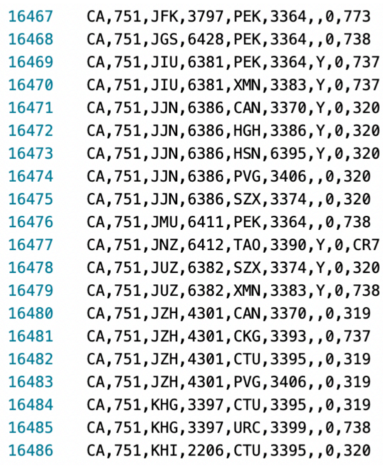
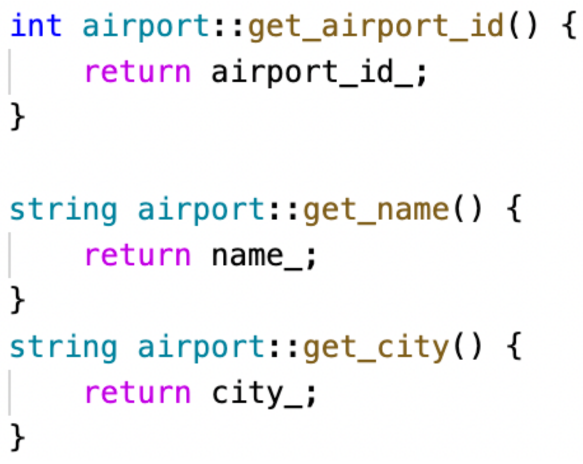
The goals for this project are finding the all the paths between two airports, figuring out the shortest distance between two airports, and finding the 10 most important airports in the world. We successfully met all the goals we set. When finding the ten most important airports in the world, we originally wanted to use the betweenness centrality algorithm. This algorithm is to find the importance of a vertex v by using the total number of shortest paths between node u and w divided by the total number of shortest paths between node u and w passing through v. Then we rank all the vertex, which is the airport in our situation, to find the ten most important airports in the world. However, during the coding process, the algorithm run time is longer than we thought, and the result cannot be output. In this case, we decided to keep the goal but use another algorithm, which is PageRank.

There are two datasets we used in this final project. The first data set we use is the airports' dataset, which contains 7698 airports. From the file, we can retrieve and get the information about airport ID, Name, latitude, longitude, etc. For example, 11817,"Curtis Field","Brady","United States”,”BBD”,"KBBD",31.1793003082,-99.3238983154,1827,\N,\N,\N,"airport","OurAirports." The second data set we use is the routes dataset, which contains 67663 routes between two airports. From the file, we can retrieve and get the information about Depart airport ID, Destination airport ID, etc.

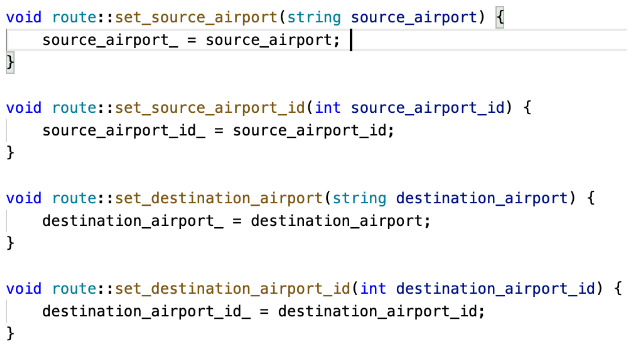

```
11797,"Björkvik Air Base","Björkvik","Sweden",\N,"ESKX",58.79079818725586,16.571199417114258,0,\N,\N,\N,"airport","OurAirports"
11798,"Ljungbyhed Airport","Ljungbyhed","Sweden",\N,"ESTL",56.082801818847656,13.212499618530273,140,\N,\N,\N,"airport","OurAirports"
11799,"Daugavpils Intrenational Airport","Daugavpils","Latvia","DGP","EVDA",55.944721221900004,26.6650009155,398,\N,\N,\N,"airport","OurA
11800,"Jēkabpils Air Base","Jēkabpils","Latvia",\N,"EVKA",56.534698,25.8925,289,\N,\N,\N,"airport","OurAirports"
11801,"Jūrmala Airport","Tukums","Latvia",\N,"EVTA",56.9422,23.2239,220,\N,\N,\N,"airport","OurAirports"
11802,"Kėdainiai Air Base","Kėdainiai","Lithuania",\N,"EYKD",55.31169891357422,23.95639991760254,171,\N,\N,\N,"airport","OurAirports"
11803,"Lime Acres Finsch Mine Airport","Lime Acres","South Africa","LMR","FALC",-28.36009979248047,23.43910026550293,4900,\N,\N,\N,"airpo
11804,"Sua Pan Airport","Sowa","Botswana","SXN","FBSN",-20.5534,26.115801,2985,\N,\N,\N,"airport","OurAirports"
11805,"Lusaka City Airport","Lusaka","Zambia",\N,"FLLC",-15.4138002396,28.3306999207,4200,\N,\N,\N,"airport","OurAirports"
11806,"Sumbe Airport","Sumbe","Angola","NDD","FNSU",-11.167900085449219,13.84749984741211,36,\N,\N,\N,"airport","OurAirports"
11807,"Mangochi Airport","Mangochi","Malawi","MAI","FWMG",-14.482999801635742,35.266998291015625,1580,\N,\N,\N,"airport","OurAirports"
11808,"Arandis Airport","Arandis","Namibia","ADI","FYAR",-22.462200164794922,14.979999542236328,1905,\N,\N,\N,"airport","OurAirports"
11809,"Mariental Airport","Mariental","Namibia",\N,"FYML",-24.60540008544922,17.925399780273438,3650,\N,\N,\N,"airport","OurAirports"
11810,"Almaza Air Force Base","Cairo","Egypt",\N,"HEAZ",30.091800689697266,31.360000610351562,300,\N,\N,\N,"airport","OurAirports"
11811,"Beni Suef Air Base","Beni Suef","Egypt",\N,"HEBS",29.20829963684082,31.016599655151367,108,\N,\N,\N,"airport","OurAirports"
11812,"Jiyanklis Air Base","Jiyanklis","Egypt",\N,"HEGS",30.819799423217773,30.191200256347656,49,\N,\N,\N,"airport","OurAirports"
11813,"Merowe New Airport","Merowe","Sudan","MWE","HSMN",18.4433333333,31.8433333333,897,\N,\N,\N,"airport","OurAirports"
11814,"St Louis Regional Airport","Alton/St Louis","United States","ALN","KALN",38.89030075069999,-90.0459976196,544,\N,\N,\N,"airport","
11815,"Chandler Field","Alexandria","United States","AXN","KAXN",45.8662986755,-95.39469909670001,1425,\N,\N,\N,"airport","OurAirports"
11816,"Columbus Municipal Airport","Columbus","United States","CLU","KBAK",39.2619018555,-85.8963012695,656,\N,\N,\N,"airport","OurAirpor
11817,"Curtis Field","Brady","United States","BBD","KBBD",31.1793003082,-99.3238983154,1827,\N,\N,\N,"airport","OurAirports"
11818,"Eastern Sierra Regional Airport","Bishop","United States","BIH","KBIH",37.3731002808,-118.363998413,4124,\N,\N,\N,"airport","OurAi
11819,"Baker City Municipal Airport","Baker City","United States","BKE","KBKE",44.837299346900004,-117.808998108,3373,\N,\N,\N,"airport",
11820,"Miley Memorial Field","Big Piney","United States","BPI","KBPI",42.58509827,-110.1110001,6990,\N,\N,\N,"airport","OurAirports"
11821,"Ozark Regional Airport","Mountain Home","United States","WMH","KBPK",36.3689002991,-92.47049713130001,928,\N,\N,\N,"airport","OurA
11822,"W K Kellogg Airport","Battle Creek","United States","BTL","KBTL",42.307300567599995,-85.2515029907,952,\N,\N,\N,"airport","OurAirp
11823,"Burley Municipal Airport","Burley","United States","BYI","KBYI",42.542598724399994,-113.772003174,4150,\N,\N,\N,"airport","OurAirp
11824,"Northeast Iowa Regional Airport","Charles City","United States","CCY","KCCY",43.0726013184,-92.6108016968,1125,\N,\N,\N,"airport",
11825,"Chanute Martin Johnson Airport","Chanute","United States","CNU","KCNU",37.668800354,-95.4850997925,1002,\N,\N,\N,"airport","OurAir
11826,"Jacksonville Executive at Craig Airport","Jacksonville","United States","CRG","KCRG",30.3362998962,-81.51439666750001,41,\N,\N,\N,
11827,"Crossville Memorial Whitson Field","Crossville","United States","CSV","KCSV",35.9513015747,-85.08499908450001,1881,\N,\N,\N,"airpo
11828,"Davison Army Air Field","Fort Belvoir","United States","DAA","KDAA",38.715000152600005,-77.1809997559,73,\N,\N,\N,"airport","OurAi
11829,"Barstow Daggett Airport","Daggett","United States","DAG","KDAG",34.85369873,-116.7870026,1930,\N,\N,\N,"airport","OurAirports"
11830,"Deming Municipal Airport","Deming","United States","DMN","KDMN",32.262298584,-107.721000671,4314,\N,\N,\N,"airport","OurAirports"
11831,"Desert Rock Airport","Mercury","United States","DRA","KDRA",36.6194,-116.032997,3314,\N,\N,\N,"airport","OurAirports"
11832,"Needles Airport","Needles","United States","EED","KEED",34.7663002014,-114.623001099,983,\N,\N,\N,"airport","OurAirports"
11833,"Duke Field","Crestview","United States","EGI","KEGI",30.65040016,-86.52290344,191,\N,\N,\N,"airport","OurAirports"
```
Airports' dataset


```
CA,751,JFK,3797,PEK,3364,,0,773
CA,751,JGS,6428,PEK,3364,,0,738
CA,751,JIU,6381,PEK,3364,Y,0,737
CA,751,JIU,6381,XMN,3383,Y,0,737
CA,751,JJN,6386,CAN,3370,Y,0,320
CA,751,JJN,6386,HGH,3386,Y,0,320
CA,751,JJN,6386,HSN,6395,Y,0,320
CA,751,JJN,6386,PVG,3406,,0,320
CA,751,JJN,6386,SZX,3374,,0,320
CA,751,JMU,6411,PEK,3364,,0,738
CA,751,JNZ,6412,TAO,3390,Y,0,CR7
CA,751,JUZ,6382,SZX,3374,Y,0,320
CA,751,JUZ,6382,XMN,3383,Y,0,738
CA,751,JZH,4301,CAN,3370,,0,319
CA,751,JZH,4301,CKG,3393,,0,737
CA,751,JZH,4301,CTU,3395,,0,319
CA,751,JZH,4301,PVG,3406,,0,319
CA,751,KHG,3397,CTU,3395,,0,319
CA,751,KHG,3397,URC,3399,,0,738
CA,751,KHI,2206,CTU,3395,,0,320
```
Routes' dataset

Then we have airports.cpp and routes.cpp to get information from the dataset. The main purpose of these two files is to set and get the information retrieved from the Airports.txt and Routes.txt.

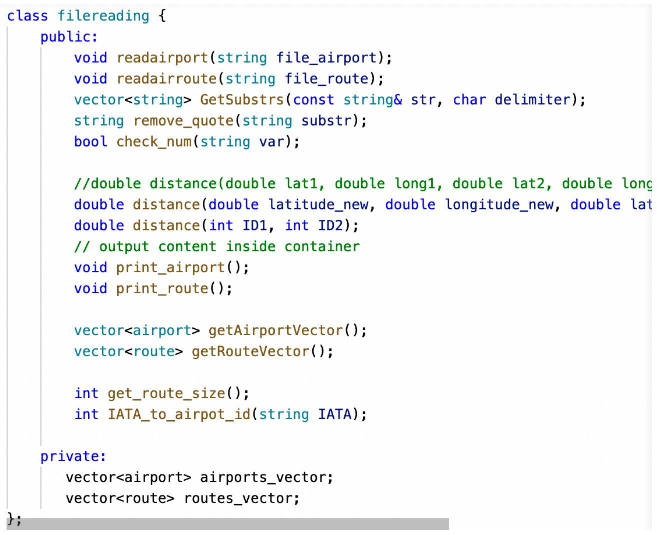


```cpp
int airport::get_airport_id() {
    return airport_id_;
}

string airport::get_name() {
    return name_;
}
string airport::get_city() {
    return city_;
}
```

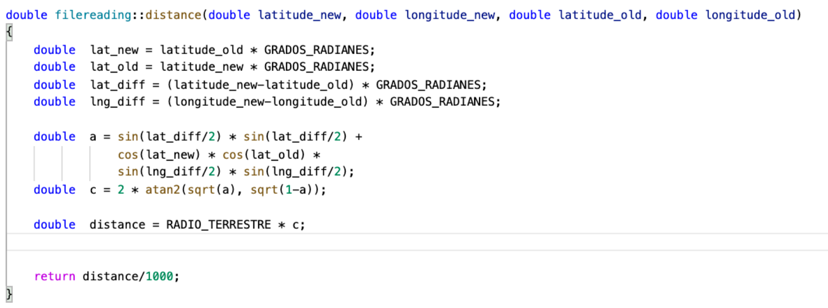


```cpp
void route::set_source_airport(string source_airport) {
    source_airport_ = source_airport;
}

void route::set_source_airport_id(int source_airport_id) {
    source_airport_id_ = source_airport_id;
}

void route::set_destination_airport(string destination_airport) {
    destination_airport_ = destination_airport;
}

void route::set_destination_airport_id(int destination_airport_id) {
    destination_airport_id_ = destination_airport_id;
}
```

The filereading files are designed to read the file from txt into the data that can be used later. One of the functions in the Filereading. cpp is to calculate the distance between two points' longitudes and latitudes by using trigonometry.

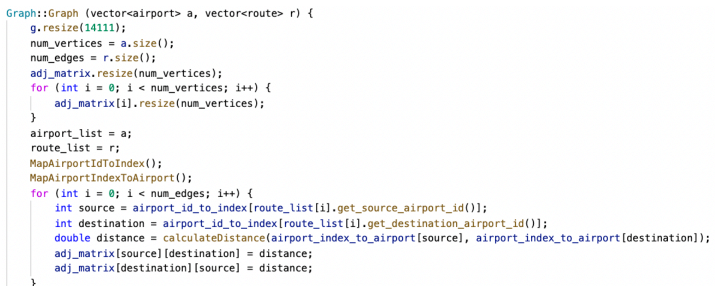


```cpp
class filereading {
    public:
        void readairport(string file_airport);
        void readairroute(string file_route);
        vector<string> GetSubstrs(const string& str, char delimiter);
        string remove_quote(string substr);
        bool check_num(string var);

        //double distance(double lat1, double long1, double lat2, double long
        double distance(double latitude_new, double longitude_new, double lat
        double distance(int ID1, int ID2);
        // output content inside container
        void print_airport();
        void print_route();

        vector<airport> getAirportVector();
        vector<route> getRouteVector();

        int get_route_size();
        int IATA_to_airpot_id(string IATA);

    private:
       vector<airport> airports_vector;
       vector<route> routes_vector;
};
```



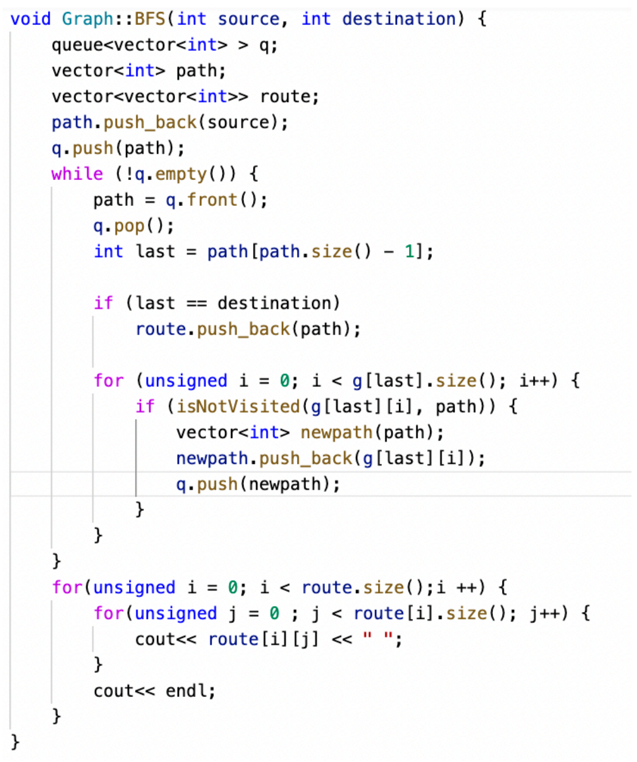
```cpp
double filereading::distance(double latitude_new, double longitude_new, double latitude_old, double longitude_old)
{
    double  lat_new = latitude_old * GRADOS_RADIANES;
    double  lat_old = latitude_new * GRADOS_RADIANES;
    double  lat_diff = (latitude_new-latitude_old) * GRADOS_RADIANES;
    double  lng_diff = (longitude_new-longitude_old) * GRADOS_RADIANES;

    double  a = sin(lat_diff/2) * sin(lat_diff/2) +
                cos(lat_new) * cos(lat_old) *
                sin(lng_diff/2) * sin(lng_diff/2);
    double  c = 2 * atan2(sqrt(a), sqrt(1-a));

    double  distance = RADIO_TERRESTRE * c;


    return distance/1000;
}
```

After all the data is inputted and the file read, it is into our central part with some crucial algorithms. The first step in the graph. cpp is to build the graph based on the vector of airports and routes. Each airport is a vertex, and each route is an edge. The weight of each edge is the distance between two airports. In this case, there are connections between airports represented by the graph.



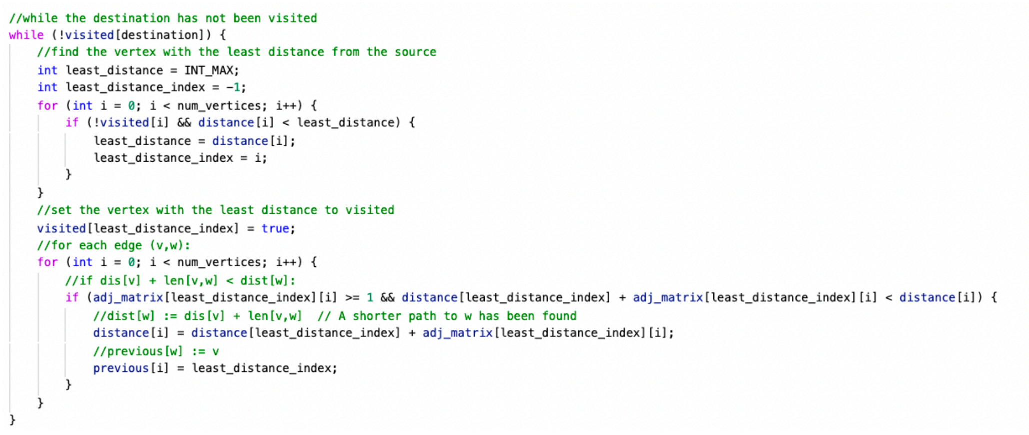
```cpp
Graph::Graph (vector<airport> a, vector<route> r) {
    g.resize(14111);
    num_vertices = a.size();
    num_edges = r.size();
    adj_matrix.resize(num_vertices);
    for (int i = 0; i < num_vertices; i++) {
        adj_matrix[i].resize(num_vertices);
    }
    airport_list = a;
    route_list = r;
    MapAirportIdToIndex();
    MapAirportIndexToAirport();
    for (int i = 0; i < num_edges; i++) {
        int source = airport_id_to_index[route_list[i].get_source_airport_id()];
        int destination = airport_id_to_index[route_list[i].get_destination_airport_id()];
        double distance = calculateDistance(airport_index_to_airport[source], airport_index_to_airport[destination]);
        adj_matrix[source][destination] = distance;
        adj_matrix[destination][source] = distance;
    }
```

The first main algorithm we use is BFS, which stands for breadth first search. It is used to find all possible routes between two airports including non-stop or transfer airlines. This algorithm used the adjacent-list.



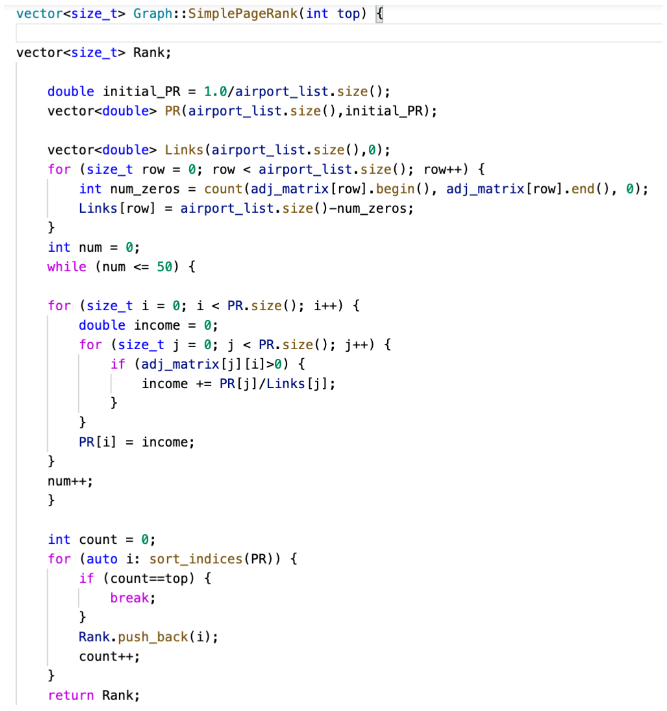
```cpp
void Graph::BFS(int source, int destination) {
    queue<vector<int> > q;
    vector<int> path;
    vector<vector<int>> route;
    path.push_back(source);
    q.push(path);
    while (!q.empty()) {
        path = q.front();
        q.pop();
        int last = path[path.size() - 1];

        if (last == destination)
            route.push_back(path);

        for (unsigned i = 0; i < g[last].size(); i++) {
            if (isNotVisited(g[last][i], path)) {
                vector<int> newpath(path);
                newpath.push_back(g[last][i]);
                q.push(newpath);
            }
        }
    }
    for(unsigned i = 0; i < route.size();i ++) {
        for(unsigned j = 0 ; j < route[i].size(); j++) {
            cout<< route[i][j] << " ";
        }
        cout<< endl;
    }
}
```

The second main algorithm we use is Dijkstra, which is used to find the shortest route between two airports. The picture is the fragment of the part of the code, which is also the highlights of this algorithm. This algorithm used the adjacent-matrix.



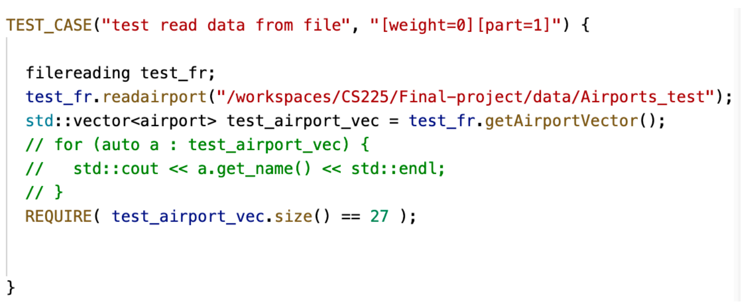
```cpp
//while the destination has not been visited
while (!visited[destination]) {
    //find the vertex with the least distance from the source
    int least_distance = INT_MAX;
    int least_distance_index = -1;
    for (int i = 0; i < num_vertices; i++) {
        if (!visited[i] && distance[i] < least_distance) {
            least_distance = distance[i];
            least_distance_index = i;
        }
    }
    //set the vertex with the least distance to visited
    visited[least_distance_index] = true;
    //for each edge (v,w):
    for (int i = 0; i < num_vertices; i++) {
        //if dis[v] + len[v,w] < dist[w]:
        if (adj_matrix[least_distance_index][i] >= 1 && distance[least_distance_index] + adj_matrix[least_distance_index][i] < distance[i]) {
            //dist[w] := dis[v] + len[v,w]  // A shorter path to w has been found
            distance[i] = distance[least_distance_index] + adj_matrix[least_distance_index][i];
            //previous[w] := v
            previous[i] = least_distance_index;
        }
    }
}
```

The third main algorithm we use is PageRank. This is an algorithm used by Google Search to rank web pages in their search engine results by counting the number and quality of links to a page to determine a rough estimate of how important the website is. In our case, it is used to find the most important airports. This algorithm used the adjacent-matrix.

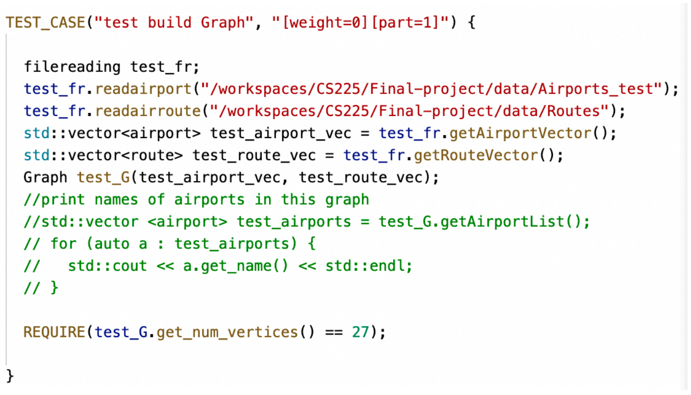


```
vector<size_t> Graph::SimplePageRank(int top) {

vector<size_t> Rank;

    double initial_PR = 1.0/airport_list.size();
    vector<double> PR(airport_list.size(),initial_PR);

    vector<double> Links(airport_list.size(),0);
    for (size_t row = 0; row < airport_list.size(); row++) {
        int num_zeros = count(adj_matrix[row].begin(), adj_matrix[row].end(), 0);
        Links[row] = airport_list.size()-num_zeros;
    }
    int num = 0;
    while (num <= 50) {

    for (size_t i = 0; i < PR.size(); i++) {
        double income = 0;
        for (size_t j = 0; j < PR.size(); j++) {
            if (adj_matrix[j][i]>0) {
                income += PR[j]/Links[j];
            }
        }
        PR[i] = income;
    }
    num++;
    }

    int count = 0;
    for (auto i: sort_indices(PR)) {
        if (count==top) {
            break;
        }
        Rank.push_back(i);
        count++;
    }
    return Rank;
```

These are all of the main functions and algorithm we used in the final project. Now, let's talk about the test function to these algorithm. These are the two test cases for file-reading and graph-building. They mainly use the function we wrote before to check whether the file read and graph builded successfully. The most direct and easiest way is to check the vertex number is correct from the testing file.

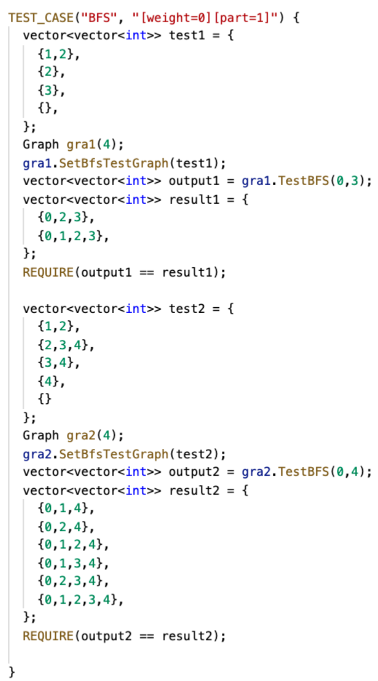


```
TEST_CASE("test read data from file", "[weight=0][part=1]") {

  filereading test_fr;
  test_fr.readairport("/workspaces/CS225/Final-project/data/Airports_test");
  std::vector<airport> test_airport_vec = test_fr.getAirportVector();
  // for (auto a : test_airport_vec) {
  //   std::cout << a.get_name() << std::endl;
  // }
  REQUIRE( test_airport_vec.size() == 27 );


}
```

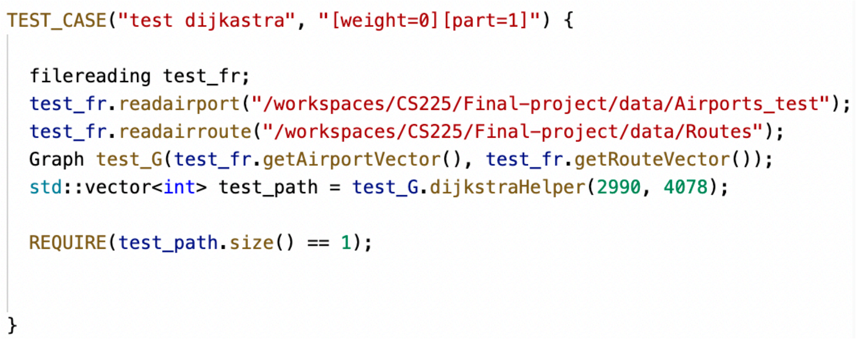


```
TEST_CASE("test build Graph", "[weight=0][part=1]") {

  filereading test_fr;
  test_fr.readairport("/workspaces/CS225/Final-project/data/Airports_test");
  test_fr.readairroute("/workspaces/CS225/Final-project/data/Routes");
  std::vector<airport> test_airport_vec = test_fr.getAirportVector();
  std::vector<route> test_route_vec = test_fr.getRouteVector();
  Graph test_G(test_airport_vec, test_route_vec);
  //print names of airports in this graph
  //std::vector <airport> test_airports = test_G.getAirportList();
  // for (auto a : test_airports) {
  //   std::cout << a.get_name() << std::endl;
  // }

  REQUIRE(test_G.get_num_vertices() == 27);

}
```

These are the test cases for the main algorithm.
By checking whether can get the correct number of the paths from two vertex and whether getting the only shortest path from BFS and Dijkstra separately.

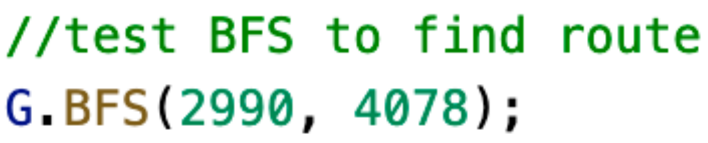


```
TEST_CASE("BFS", "[weight=0][part=1]") {
  vector<vector<int>> test1 = {
    {1,2},
    {2},
    {3},
    {},
  };
  Graph gra1(4);
  gra1.SetBfsTestGraph(test1);
  vector<vector<int>> output1 = gra1.TestBFS(0,3);
  vector<vector<int>> result1 = {
    {0,2,3},
    {0,1,2,3},
  };
  REQUIRE(output1 == result1);

  vector<vector<int>> test2 = {
    {1,2},
    {2,3,4},
    {3,4},
    {4},
    {}
  };
  Graph gra2(4);
  gra2.SetBfsTestGraph(test2);
  vector<vector<int>> output2 = gra2.TestBFS(0,4);
  vector<vector<int>> result2 = {
    {0,1,4},
    {0,2,4},
    {0,1,2,4},
    {0,1,3,4},
    {0,2,3,4},
    {0,1,2,3,4},
  };
  REQUIRE(output2 == result2);
}
```

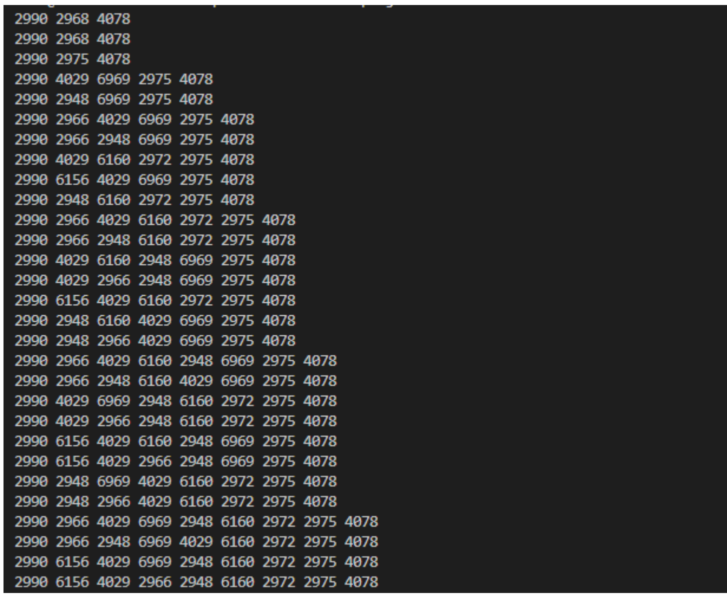


```
TEST_CASE("test dijkastra", "[weight=0][part=1]") {

  filereading test_fr;
  test_fr.readairport("/workspaces/CS225/Final-project/data/Airports_test");
  test_fr.readairroute("/workspaces/CS225/Final-project/data/Routes");
  Graph test_G(test_fr.getAirportVector(), test_fr.getRouteVector());
  std::vector<int> test_path = test_G.dijkstraHelper(2990, 4078);

  REQUIRE(test_path.size() == 1);


}
```

From the result outputted, we can conclude that all of our algorithms are worked correctly. I will discuss the result now.
When testing BFS, we just input the two numbers randomly, which represent two airports. Then, the BFS result print all the path between these two airports successfully, including all the paths that pass through other airports. The perfect result from BFS is indirectly showing that our file-reading and graph-building is successful. These are all the routes from airport ID 2990 to 4078 ranked by the distance.

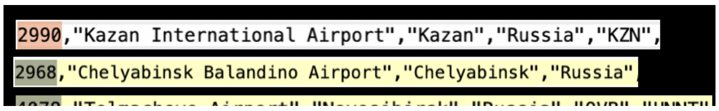


```
//test BFS to find route
G.BFS(2990, 4078);
```

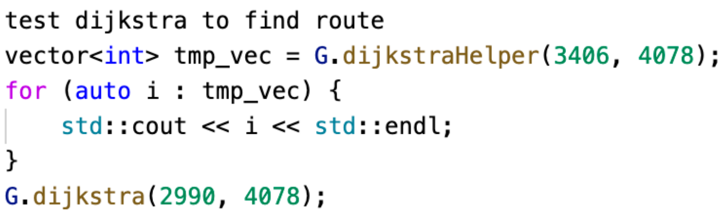


```
2990 2968 4078
2990 2968 4078
2990 2975 4078
2990 4029 6969 2975 4078
2990 2948 6969 2975 4078
2990 2966 4029 6969 2975 4078
2990 2966 2948 6969 2975 4078
2990 4029 6160 2972 2975 4078
2990 6156 4029 6969 2975 4078
2990 2948 6160 2972 2975 4078
2990 2966 4029 6160 2972 2975 4078
2990 2966 2948 6160 2972 2975 4078
2990 4029 6160 2948 6969 2975 4078
2990 4029 2966 2948 6969 2975 4078
2990 6156 4029 6160 2972 2975 4078
2990 2948 6160 4029 6969 2975 4078
2990 2948 2966 4029 6969 2975 4078
2990 2966 4029 6160 2948 6969 2975 4078
2990 2966 2948 6160 4029 6969 2975 4078
2990 4029 6969 2948 6160 2972 2975 4078
2990 4029 2966 2948 6160 2972 2975 4078
2990 6156 4029 6160 2948 6969 2975 4078
2990 6156 4029 2966 2948 6969 2975 4078
2990 2948 6969 4029 6160 2972 2975 4078
2990 2948 2966 4029 6160 2972 2975 4078
2990 2966 4029 6969 2948 6160 2972 2975 4078
2990 2966 2948 6969 4029 6160 2972 2975 4078
2990 6156 4029 6969 2948 6160 2972 2975 4078
2990 6156 4029 2966 2948 6160 2972 2975 4078
```

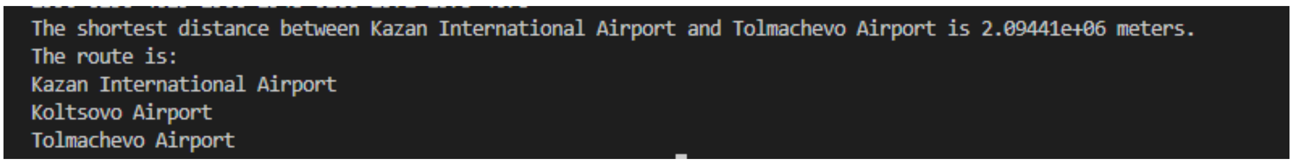
Similarly, when testing Dijkstra, we just input the two numbers randomly, which represent two airports. The output exactly match the airport name from the ID number. Also the shortest path from Kazan(ID 2990) airport to Tolmachevo airport(ID 4078) is same to the first result from BFS, which means this is the shortest path. The output will show the distance and the transfer station if there have.



```
2990,"Kazan International Airport","Kazan","Russia","KZN",
2968,"Chelyabinsk Balandino Airport","Chelyabinsk","Russia"
```



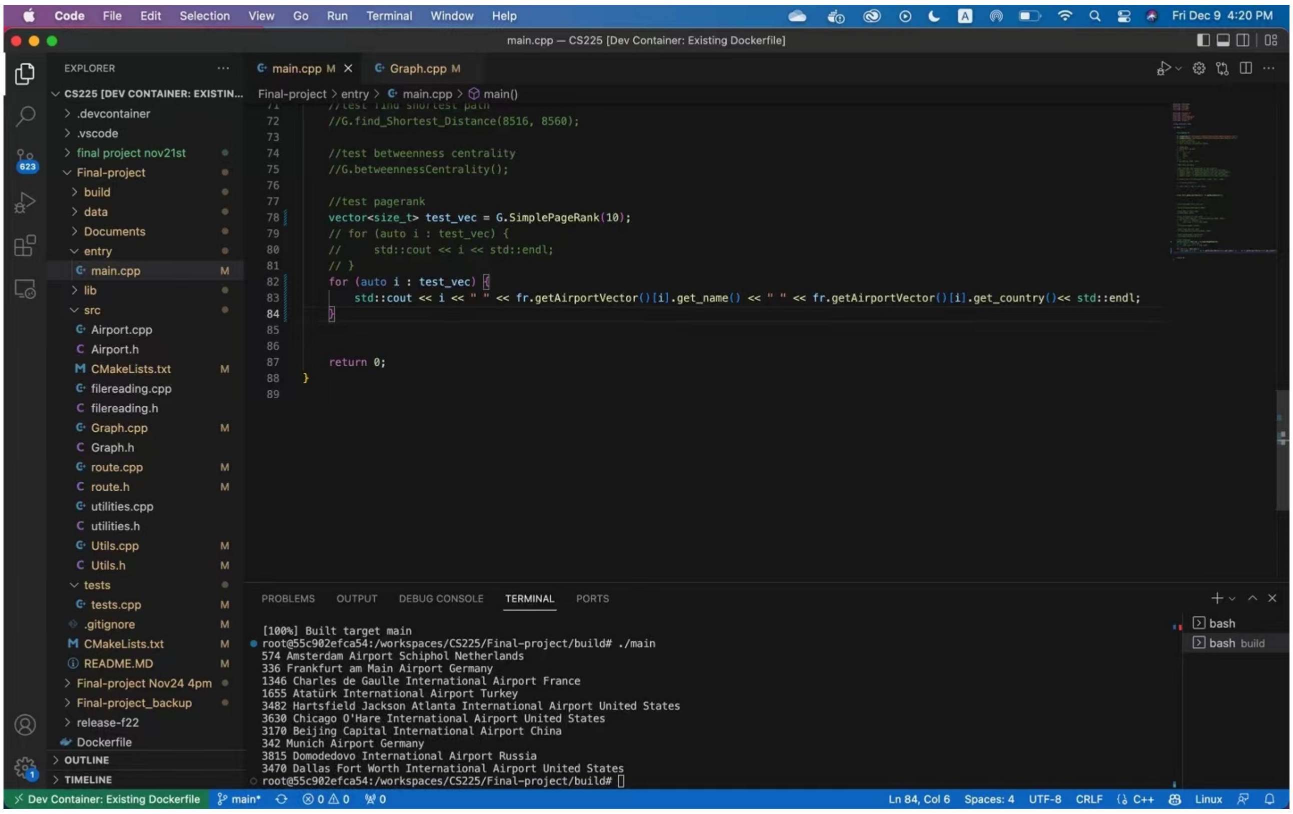
```
test dijkstra to find route
vector<int> tmp_vec = G.dijkstraHelper(3406, 4078);
for (auto i : tmp_vec) {
    std::cout << i << std::endl;
}
G.dijkstra(2990, 4078);
```



```
The shortest distance between Kazan International Airport and Tolmachevo Airport is 2.09441e+06 meters.
The route is:
Kazan International Airport
Koltsovo Airport
Tolmachevo Airport
```

Lastly, is the PageRank algorithm. Inputing how many airport we want to output, which is 10 here. Then the output shows the ten most important airports by rank due to their transit abilities. The bigO of PageRank algorithm is O(E·k) where E is the number of edges and k is the number of iterations. We iterate for 50 times to increase the credibility and accuracy.

However, the O(50E) is larger than we expect and the result need to be output in pretty long time period. Still, we can print out the result.


main.cpp — CS225 [Dev Container: Existing Dockerfile]
main.cpp M
Graph.cpp M
Final-project > entry > main.cpp > main()
EXPLORER
CS225 [DEV CONTAINER: EXISTIN...
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Dev Container: Existing Dockerfile
main*
Ln 84, Col 6
Spaces: 4
UTF-8
CRLF
C++
Linux

```cpp
    //G.find_Shortest_Distance(8516, 8560);

    //test betweenness centrality
    //G.betweennessCentrality();

    //test pagerank
    vector<size_t> test_vec = G.SimplePageRank(10);
    // for (auto i : test_vec) {
    //     std::cout << i << std::endl;
    // }
    for (auto i : test_vec) {
        std::cout << i << " " << fr.getAirportVector()[i].get_name() << " " << fr.getAirportVector()[i].get_country()<< std::endl;
    }


    return 0;
}
```

```
[100%] Built target main
root@55c902efca54:/workspaces/CS225/Final-project/build# ./main
574 Amsterdam Airport Schiphol Netherlands
336 Frankfurt am Main Airport Germany
1346 Charles de Gaulle International Airport France
1655 Atatürk International Airport Turkey
3482 Hartsfield Jackson Atlanta International Airport United States
3630 Chicago O'Hare International Airport United States
3170 Beijing Capital International Airport China
342 Munich Airport Germany
3815 Domodedovo International Airport Russia
3470 Dallas Fort Worth International Airport United States
root@55c902efca54:/workspaces/CS225/Final-project/build# 
```

Overall, all the algorithm worked successfully. The result shows the correct output as we expected. If there is any chance, we will keep revise the PageRank algorithm to reduce its' runtime.