Group members:
Elijah Gaohan Ye (gaohany2)
Bowei Song (boweis2)
Michael Duan (hduan5)

CS225 Final Project Report

The output and correctness of each algorithm

We have a total of three algorithms: DFS, Dijkstra, and Strongly Connected Components. DFS is depth first search. Our graph is a directed graph with vertices being the airports and edges being the routes between the airports. Since our graph is a directed graph, DFS will return all the airports that can be reached from our source airport. It is very hard to test DFS on the full dataset, so we create our own datasets: test_node.dat and test_route.dat. test_node.dat contains 10 airports, and test_route.dat contains 9 flights that connect between these airports.

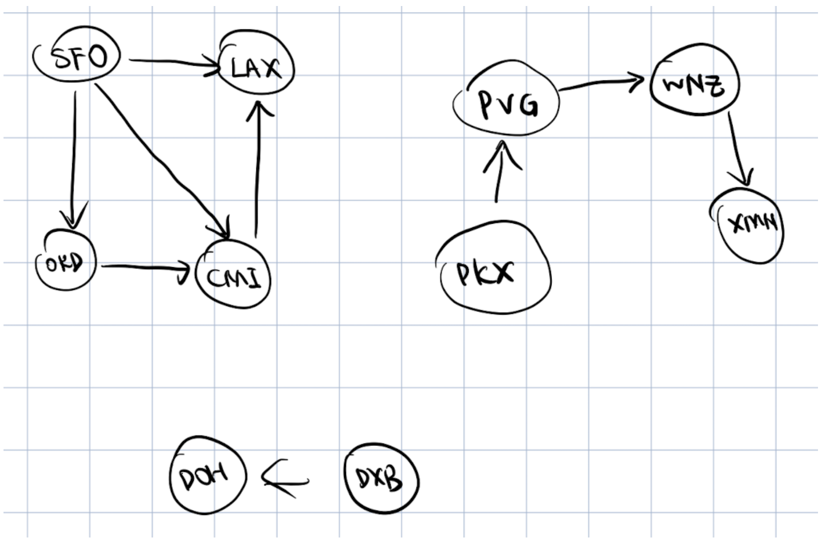
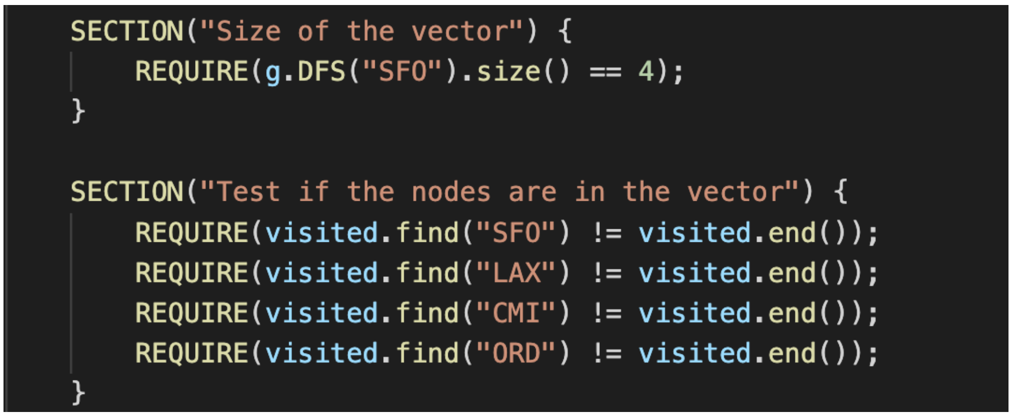


Figure 1. Graph to test DFS

Our DFS testcase utilizes these datasets and checks if DFS really does what it is supposed to do. Here we can see that our test case tests for the size of the return vector. According to the graph, the size should be 4. Also, it should contain SFO, LAX, CMI, and ORD. That's why we are checking that as well in our test case.

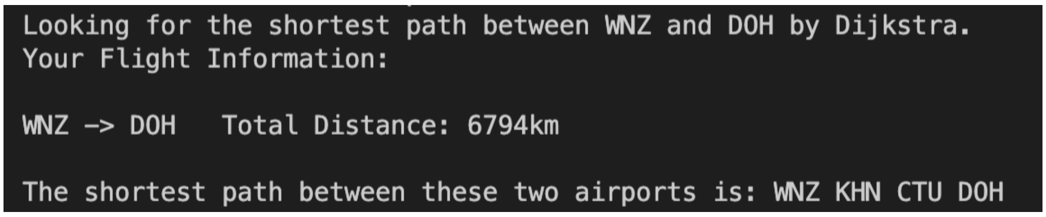


```
SECTION("Size of the vector") {
    REQUIRE(g.DFS("SFO").size() == 4);
}

SECTION("Test if the nodes are in the vector") {
    REQUIRE(visited.find("SFO") != visited.end());
    REQUIRE(visited.find("LAX") != visited.end());
    REQUIRE(visited.find("CMI") != visited.end());
    REQUIRE(visited.find("ORD") != visited.end());
}
```

Figure 2. Part of Test Case for DFS

Another algorithm is Dijkstra. This algorithm finds the shortest path between two vertices. We implemented our graph in a way that the weight of the edge is the distance between each vertex. Therefore the path this algorithm chooses will be the shortest path in terms of distances. Again, even though we are able to run the algorithm on the full scale, it is very hard to make sure it is actually right with such a big dataset. So we also wrote our test cases on our own datasets. We also have a test case to test if there is no path between two airports.



```
Looking for the shortest path between WNZ and DOH by Dijkstra.
Your Flight Information:

WNZ -> DOH   Total Distance: 6794km

The shortest path between these two airports is: WNZ KHN CTU DOH
```

Figure 3. Running Dijkstra on the Full Dataset

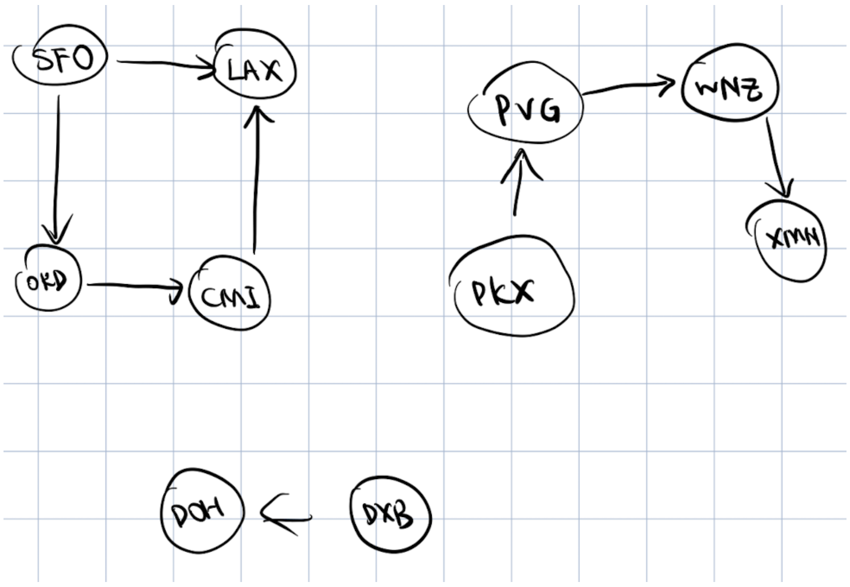
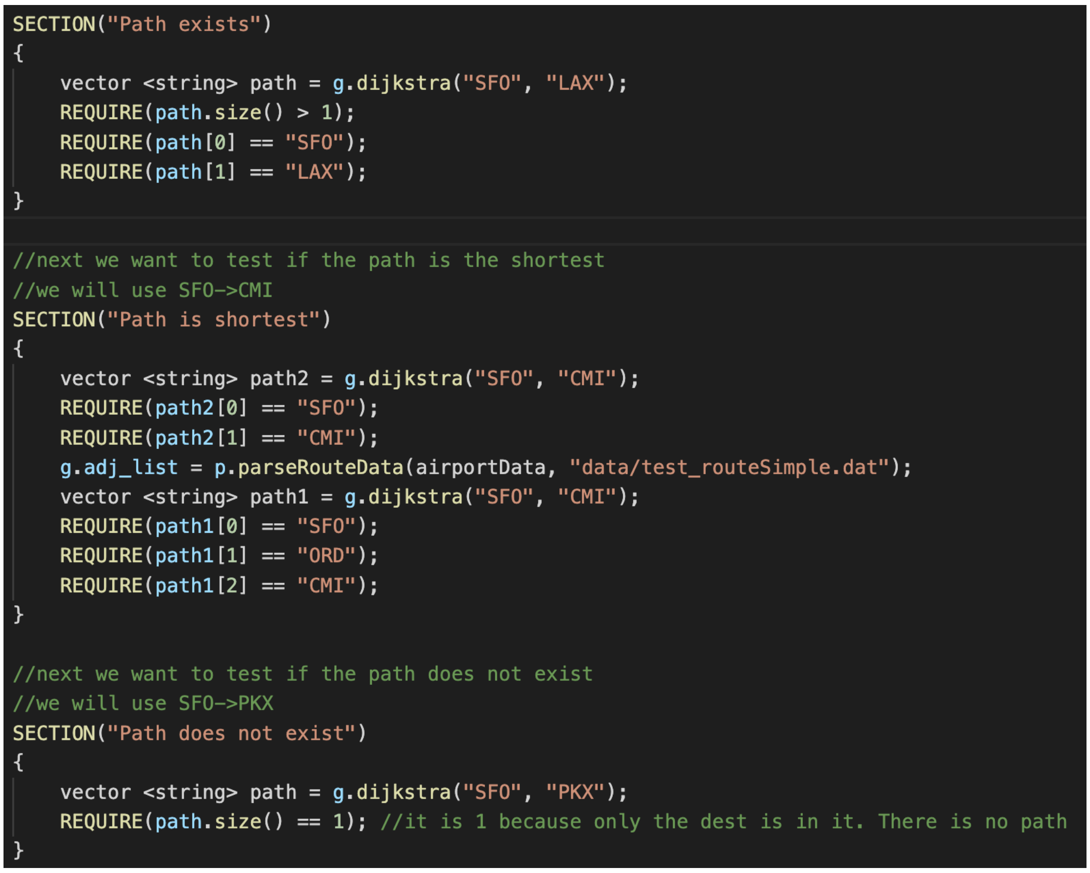


Figure 4. Test_routeSimple

In the test case for Dijkstra, we use two route datasets that we created. Test_route (Figure 1) has direct connection from SFO to CMI, but Test_routeSimple does not have direct connection from SFO to CMI. The ideal solution is that for Test_route, SFO will go directly to CMI. As for Test_routeSimple, SFO needs to go to ORD before going to CMI. We test this by checking each element of each path. We also check what happens if there is no path between two airports. We check that by making sure the path it returns has only one element since our algorithm will always have the destination in the path that it returns.



```
SECTION("Path exists")
{
    vector <string> path = g.dijkstra("SFO", "LAX");
    REQUIRE(path.size() > 1);
    REQUIRE(path[0] == "SFO");
    REQUIRE(path[1] == "LAX");
}

//next we want to test if the path is the shortest
//we will use SFO->CMI
SECTION("Path is shortest")
{
    vector <string> path2 = g.dijkstra("SFO", "CMI");
    REQUIRE(path2[0] == "SFO");
    REQUIRE(path2[1] == "CMI");
    g.adj_list = p.parseRouteData(airportData, "data/test_routeSimple.dat");
    vector <string> path1 = g.dijkstra("SFO", "CMI");
    REQUIRE(path1[0] == "SFO");
    REQUIRE(path1[1] == "ORD");
    REQUIRE(path1[2] == "CMI");
}

//next we want to test if the path does not exist
//we will use SFO->PKX
SECTION("Path does not exist")
{
    vector <string> path = g.dijkstra("SFO", "PKX");
    REQUIRE(path.size() == 1); //it is 1 because only the dest is in it. There is no path
}
```

Figure 5. Test cases for Dijkstra

Last algorithm is Strongly Connected Components(SCCs). A strongly connected component (SCC) means that each vertice in this SCC can reach other vertices. In the application for flights, if two airports are in the same SCC, it means they can reach each other. As a traveller, it is important to be able to arrive at your destination, but it is also important to be able to return from your destination. Again, even though we are able to run the algorithm on the full scale, it is very hard to make sure it is actually right with such a big dataset. So we also wrote our test cases on our own datasets. We wrote a special route data for our SCC algorithm.

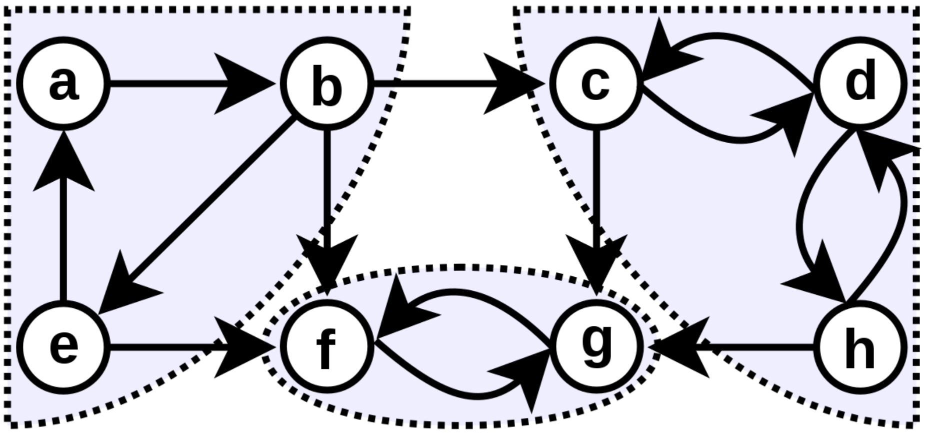


Figure 6. Strongly Connected Components from Wikipedia

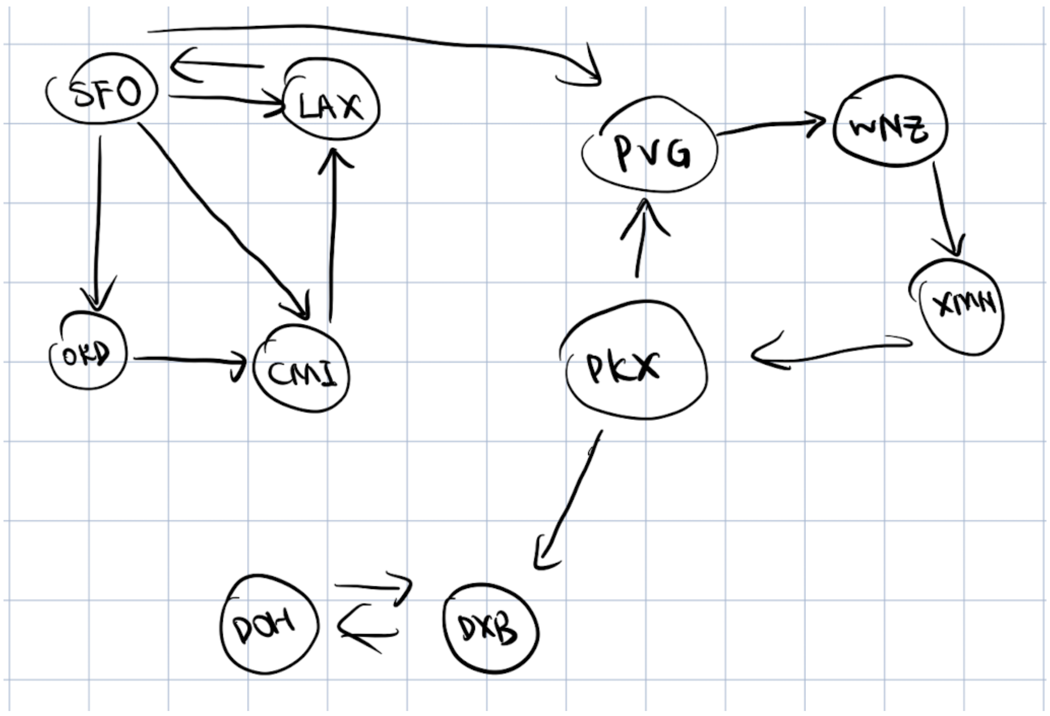
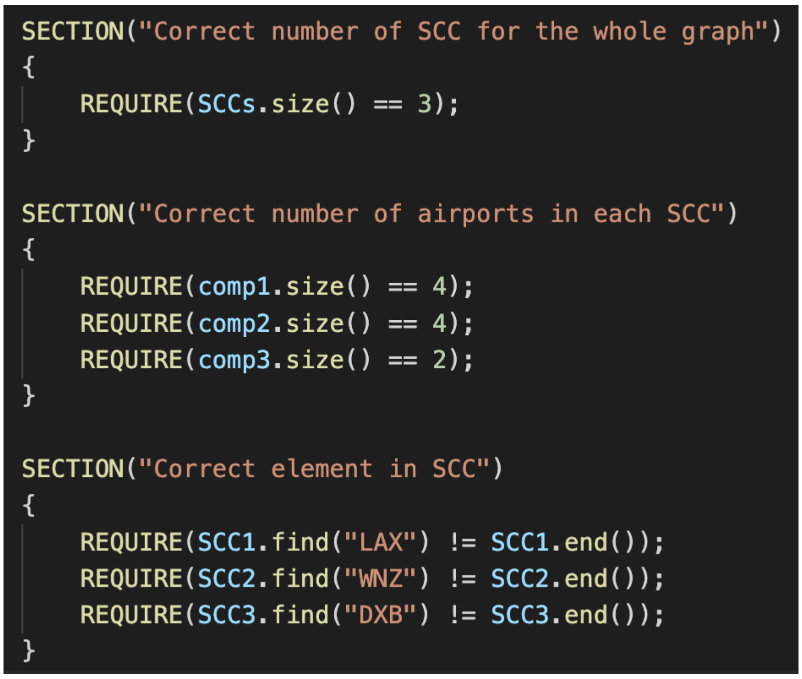


Figure 7. Test_route_SCC

We have two functions for our SCC algorithm. One will run SCC on the whole graph, and will return a 2D vector. Each element of this 2D vector is a vector itself. The size of this 2D vector means the number of SCCs in our graph. In our graph, we can see that there are 3 SCCs, so we have a test case for it. The other function is to return a vector just associated with that single airport. So, if I run the second function on SFO, the associated SCC will have SFO, LAX, ORD, and CMI, which will result in size 4. That's what the middle part of the test case does for us. Lastly, I also check if it has the correct elements in it.



```
SECTION("Correct number of SCC for the whole graph")
{
    REQUIRE(SCCs.size() == 3);
}

SECTION("Correct number of airports in each SCC")
{
    REQUIRE(comp1.size() == 4);
    REQUIRE(comp2.size() == 4);
    REQUIRE(comp3.size() == 2);
}

SECTION("Correct element in SCC")
{
    REQUIRE(SCC1.find("LAX") != SCC1.end());
    REQUIRE(SCC2.find("WNZ") != SCC2.end());
    REQUIRE(SCC3.find("DXB") != SCC3.end());
}
```

Figure 5. Test cases for SCC algorithm

Answer to our leading question

Our leading question is “We want to build a search tool that helps people to find the best way for them to travel by using airline route data from Open Flights… a shortest path from one airport to another.” We did answer our leading question by finding the shortest path between two airports. During the process of writing and testing our algorithms, I found that most of the airports in the dataset are in one big SCC, which makes sense since in the real world, most of the airports should be connected to one another. However, there also are a lot of airports that seem to be isolated from the big SCC. By carefully examining the dataset, many of these isolated airports are military bases. Therefore, there are no regular flights from or to it. Also, in figure 2, we find that the shortest path from WNZ to DOH is WNZ -> KHN -> CTU -> DOH. The path might be the shortest path in distance but in real life, no one will take this path since it might take days to reach the destination. The better way to do it is to make the weight be the time for each path or even the number of stops it needs to take.