"Open Flight" Project

yg17   paiz3   bangyan3   yujiem4

Goals

郭:

Hi, welcome to our final project, the Open Flight. First, let's briefly talk about the purpose of the project and our achievement.

缪:

Our project is designed to deal with some problems that might arise when facing a large set of airports and airlines. The first problem is the shortest paths problem. Namely, given an airport as the source, we want to figure out the shortest paths to some other airports. This is probably the most commonly met problem. The second problem is the strongly connected component problem. That is, the two airports are in the same component if and only if each airport can be reached from the other only by plane. This problem is becoming more important because many airports are shut down due to the pandemic, and we want to find out the regions we can reached after a set of airports is out of service. We have implemented algorithms to solve both problems successfully.

Development

And now let's talk about our development process. We can divide our project into four major parts: the classes, the algorithms and tests, the data and inputs, and the main funciton and outputs.

We have created two main classes for the project. One of them is the "Node" class that represents an airport. It contains fields such as the name of the airport, the position of the airport, whether the airport is out of service, etc. The other is the "Graph" class that represents the whole map of airports and airlines. Because of the relationship between a single airport and a map of multiple airports, we have decided to make the node class nested in the graph class.\

图:

We used Dijkstra's (发音：'die - stra) algorithm for the shortest path problem, and Kosaraju's (发音：扣萨ra菊) algorithm for the strongly connected component problem. Both of them return a map that maps to a list of airports. This is ideal because we only have to run the algorithm once to get multiple results. DFS traversal was used in both algorithms. We also included test cases to check our correctedness. We have tested the two algorithms both before and after some airports is put out of service, and the results identical to what we would have expected.

The data and inputs are contained in four files in total, two for data and two for inputs. The data files included the raw information of the airports and the airlines. More than half of them we don't need, so we have filtered the data by choosing only the desired sections that were separated by commas. Also, we have made the users write their input into files because the amount of input could be large, so allowing files as input is beneficial.

Finally is the main function that runs the algorithms and generates the outputs. We decided to always print out our output in the terminal, while letting the users to decide whether they want to save the output into another separate file.

派:

Conclusion

郭:

Let's try to run the algorithms.

Here we can see that most shortest paths from ORD are about 1 to 5 flights long. This result is expected since large airports like ORD has much more airlines than average. If we change the source airport to some small airport, we might get some shortest paths that are about 5 to 10 flights long.

And the strongly connected componet that ORD is in has a total of 1910 airports. This is the result after we have put about 1 to 2 thousand airport out of service. So I'd say it is quite a large component.

We have learned a few things throughout the process. The first is that it is important to create a so called high-level architecture of the whole program in the beginning. For example, the clear declarations of interfaces between functions. We haven't done that in the first place, and this has led us to some uncessary conflicts when we are trying to merge the different parts of our program.

Also, we need to really figure out the needs of the users and development percise solutions for them. Or else, we might end up being simply implementing the classical algrithms but ignoring some real problems.

And finally, the robustness of the program is important. It should be considered a fine practice to modify the code so that it can handle multiple edge cases, instead of just assuming that the users never make mistakes. For example, in our code, we check whether the input file names are correct. But there are still some other exceptions that might occur, and if we were given more time, we would definite improve that.

That would be all of our presentation. Thank you.