ALGORITHMS TASK

Approach to the Task

Firstly, I saw the data and noticed the following :
1) There are a lot of flights and in comparison, the number of cancelled flights is quite small. This makes it easier to reallocate the affected passengers
2) Each flight is on average 75% full. Again, this helps reallocate the affected passengers. What this practically means is that we have to find on average ~4 routes to reroute the affected passengers.
3) The data set is not too large so we can run a lot of bruteforce type algorithms with reasonable run times.

I chose to use C++ instead of Python to code this task in because I am more comfortable and familiar with it, and it is also much faster.

Actually getting to the data, I immediately faced an issue when I realised that even if the number of cancelled flights are not large, there are very few flights that can actually even be used as replacements. My script found 2 direct flights, 4 routes with 1 layover and 41 routes with 2 layovers (of course some routes have common flights in between them). This can be simply found with a nested loop over all flights since the number of flights is only 416. This ignores routes with < 3600 (=1 hour?) gap in between 2 adjacent flights.

First I tried some easy ways to distribute people : like first allocate the direct flights, then the 1 layover and then the 2 layover ones. Here there are again many ways one can allocate the 2 layover ones so I tried to randomly generate permutations and see how many people I could allocate. The number of reallocations stayed in the 140 - 170 range.

Before making further improvements to the algorithm, I decided to make a ranking function which will help me decide when a strategy is actually useful. I have a total of 500 points for every passenger. Based on the number of layovers, the point system looks like this :

● 0 layovers : points will be given in range [400, 500]
● 1 layover : points will be given in range [275, 425]
● 2 layovers : points will be given in range [150, 350]
● Not allocated at all : 0 points

I thought this point function is fair as it gives importance to the number of layovers while still keeping a significant portion of points just for being able to allocate a flight (even with 2 layovers).

The next part of the point function is to decide the actual value within that range. This has equal weightage from 2 parts : flight duration and difference of arrival time from original timing. Flight duration is an inverse function of the allocated flight length (with full points if the new flight is shorter). Difference from original time is a simple linear function, with earlier arrival time gaining more points.

Having done the point function, I made a list of all possible flight combinations that are even possible (only 47 as mentioned above). Then I tried shuffling the vector or flight combinations and trying to fill that route with passengers greedily from start to end. If we shuffle quite a few times (say 1000/10000) we get a close to optimal solution. The Point values stayed around a range of 38000-50000 and we were able to get 174 passengers reallocated at maximum out of the 505.

A further improvement to the algorithm was to sort the flights in order of the points that they would give. This alone gave me 51240 points which was more than I got before. I tried to improve on this with a few heuristics like sorting and then randomly swapping 20 indices to still have random but closer to optimal vectors. This did improve the accuracy of the algorithm and the average number of points as the range became 46000-51240 instead, however it didn't improve the maximum points obtained.

Detailed Description of Final Algorithm :

We have a vector of possible routes to take, initially empty.
For every cancelled flight, we first look at all possible direct (non-cancelled) flights and see if it has the same endpoints as the cancelled flight. If so, we add it to our vector.
Then, we look at possible connecting flights by brute forcing over all pairs of flights, and checking if they satisfy all conditions (endpoints same, and the arrival time of the 1st is at least 3600 before the departure of the 2nd), if it does, we add it to our vector
Then we look at possible routes with 3 flights, again brute forcing all triplets. This runs fast because there are only 416 flights.

Thus, we have our vector of all possible routes we can take.
For every route, we also know the associated point value. So, we sort the vector in descending order of points (thus prioritising higher points).

Then, we try to greedily move the passengers from the cancelled flights to the routes in order. So for example, if a route has 3 flights x, y, z, and our cancelled flight is w, then we can move min(not_allocated_passengers[w], leftover_space[x], leftover_space[y], leftover_space[z]) to the route (and update those values as well).

For a final optimization, we run 10000 iterations where we try to shuffle the vector of possible routes. A purely random shuffling is less optimal. Instead, we take the sorted in descending order vector and swap 20 indices randomly. This greatly improves the average points.

Final Statistics :

```
Affected,Reallocated,AvgLay,TimeDiff,SolTime
505,174,1.552,47216.546,4067
```

174 is also the highest number of reallocations I could find.