CS211 ALGORITHMS & DATA STRUCTURES II

Final Project

Alex Maxwell

18457412

The overall strategy of my approach: To tackle the Equipment challenge I used a greedy nearest neighbour Algorithm, where I iterated over all of the coordinates and found the distance between the current position and each i of the iteration. Per each iteration I found the smallest distance and position which was at least 100km and an unvisited airport. This then became the next current position/next airport on the path. For the most part this was fine but there was a case where the path became longer and there was less and less options to choose from, eventually this alone wasn't sustainable as not every airport had an unvisited airport at least 100km away from it. So I also had to use another greedy nearest neighbour algorithm which would determine the airport which was nearest to the current position airport and nearest to the unvisited airport we wanted to reach, finding the nearest intermediary between both airports. This intermediary would be an already visited airport. This added extra cost in terms of time and distance to the solution by adding extra airports which have already visited. I used two data structures within the solution, an array called check, of type Boolean. As well as an array list type string called currentPath. The check array held the purpose of holding the Boolean values (true or false) of whether an airport at a slot was visited(true) or unvisited(false). I chose an array for this function as indexing was key and the data structure had a defined and definitive size (1001, number of airports). I used an array list to hold the string values of the airports from 0-1000, the reason I used an array list instead of a more basic data structure is because the size of the data structure was not definitive and was dependent on the path taken(whether more or less intermediaries are needed) . I wrote this program in java because it is my strongest language and it is robust and flexible. I did not use any statistical or artificial intelligence techniques within my solution. I did use an optimization technique, I used a minimal 2-opt optimization technique to try get rid of any crossovers and shorten distance and time. For this technique I checked if the sum of 3 edges between 4 vertices (i-1,i,i+1,i+2) was bigger than the sum of 3 edges between 4 vertices (i-1,i+1,i,i+2) and iterated over the whole path to get rid of any crossovers. This adaption of the 2-opt technique swaps pairs rather than chunks that is why I would consider it to be minimal. The overall structure of the code was a basic Object Orientated Programming approach where I had a class(equipment) which made an object of another class(brain) passing in the co-ordinates to be used in the class brain. The brain held the solving portion in terms of finding the solution string, it had public operations getDistance() and getSolution(). The equipment class calls the brainobject.getSolution() to return the solution string and prints out the cost of the string/ output. The overall Space complexity of the program is O(n), let's say we double the coordinates inputted to the program, the check array would double in size and the currentPath array list would more or less be double its current size. The Time complexity of the program is O(n^2). The code was pretty efficient and wasn't taxing in regard to time, It could run within the HackerRank time limit so approx. 5 secs to generate a solution.