*Advanced Programming*

*Report*

*By Idrees Malik*

*(201603033)*

Introduction

The purpose of this coursework is to construct a network that performs each given task in an efficient and accurate manner. This is so that we can then asses the performance of the data structures and evaluate the efficiency based of the data structures we used.

This report will outline the results I obtained for each process command and explain how I achieved each given result by explaining my logic and method utilized within my code. I will outline and explain the efficiency of my code, and suggest improvements that I could implement into my work.

MaxDistance

|  |  |  |
| --- | --- | --- |
| **Process Operated** | **Timing**  **(microseconds)** | **Results** |
| MaxDistance | 5364.0 | Malton Rail, Zeebrugge Harbour,411.279 |

MaxDistance is a process command that will find the furthest separated places, then calculate the distance between them.

The approach I used to calculate the furthest separated locations was to find the first reference within the nodes vector then calculate the location of northings and eastings of that given node then I would go through every other node and I would calculate the distance of those given nodes, finding their northings and eastings locations. I would then use Pythagoras to calculate the actual distance of the nodes between one another then store that value into a float distance value inside of a conditional. If the current distance is less than the new calculated distance between the two nodes then update the distance to the new calculated distance, therefore allowing the distance to always be updated with the largest distance value. This would continue until the distance between every node stored within the vector has been calculated.

I believe my maxDistance method is O(N2) this is because the first for loop running through my nodes is dependant on the size of the nodes file and how many nodes are stored within in causing it to become O(N). Within the for loop there are multiple if statements which each take O(1), but as there are multiple if statements it will affect the running time causing it to become O(N). This causes the O(N) \* O(N) causes the method time complexity to become O(N2). Within the algorithm the if statements would not massively affect the timing for the algorithm to complete as the main reason for the time is how many time the for loop is iterated through. But as we have to compare every node reference to every other node reference this causes the for loop to be iterated until this has been completed.

MaxLink

|  |  |  |
| --- | --- | --- |
| **Process Operated** | **Timing**  **(microseconds)** | **Results** |
| MaxLink | 298.0 | 17191741,51889340, 356.310 |

MaxLink is a process command method that will find the longest single arc between two nodes and then calculate the distance between the two nodes.

The approach I used to calculate the MaxLink was to use a for loop to go through all my arcs, I then go through all my nodes is a separate for loop in order to find the start reference number in the index of the given arc. Once my nested for loop has found the start reference number it then calculates the northings and eastings of that given node. I then run through the nested for loop again which then finds the end reference number of the given arc. Once I’ve found the end reference number in the nodes I then calculate the northings and eastings of the given node and then calculate the distance between the given references. I continue this process until I have calculated the distance of every start and end reference within the arcs for all arcs. I have a conditional within the nested for loops which stores the largest distance between nodes.

I believe my maxLink method be O(N2). This is because the first for loop has to loop through every arc within the arc vector causing the for loop to have a notation of O(N), but also the second for loop nested within the first for loop has to loop through all the possible node references within the node vector causing it to have a notation of O(N). This causes the maxLink algorithm to have a big O notation of O(N2) due to O(N) \* O(N). The conditional within the nested for loop does not significantly affect the time taken to execute the algorithm therefore they’re not taken into account.

FindDistance

|  |  |  |
| --- | --- | --- |
| **Process Operated** | **Timing**  **(microseconds)** | **Results** |
| FindDistance | 1082.0 | Selby Rail, Howden Rail, 13.531 |

The FindDistance function is a command method that will find the distance between two specified places.

The approach I used to calculate the FindDistance between the specified places was very similar to the MaxDistance function, as I essentially was doing the same logic but this time between two specified places. Within the findDistance method I used a for loops, the for loop ran through my nodes to find the first reference and once that reference had been found I would calculate the northings and eastings of it. The for loop again would run through the nodes to find the second reference and once found would calculate the northings and eastings of the given node, then calculate the given distance between the nodes by calling an inline function to calculate the distance between the nodes. The FindDistance is similar to the MaxDistance structurally as they both contain a for loops with the only difference being the FindDistance doesn’t need to calculate the largest distance between nodes.

I believe the FindDistance method has a big O notation of O(N). This is because the method doesn’t have to fully iterate through the whole for loop as it has to find the two specified locations only allowing it to be O(N). Within the for loop the if statements do not significantly affect the time it take to go through the method preventing them from having a significant influence on the timing.

FindNeighbour

|  |  |  |
| --- | --- | --- |
| **Process Operated** | **Timing**  **(microseconds)** | **Results** |
| FindNeighbour | 14.0 | 8611522 points to:   * 8631524 * 11251704 * 9361783 * 12321385 * 13491586 |

The FindNeighbour command method will find all the neighbours linked to a given node. Then output the given neighbours to the outputs files.

The approach I used to find the neighbours of the given node reference was to create a for loop that would go through all the nodes in the node vector, and find the reference allowing me to access the arcs linked to the given reference. Once I’ve located the start reference I create a vector of arcs that stores all the arcs that node reference is linked to in a vector. I then run through the vector in a nested for loop and output each reference linked to the start reference. This method allows me to find all the neighbours of the start reference.

I believe the FindNeighbour method has a notation of O(N2). This is because it has 2 for loops nested within one another that with the first for loop only being able to iterate once and having to wait for the nested for loop to iterate through all the arcs linked to the given node. This causes each for loop to have a big O notation of O(N) causing the method to have a big O notation of O(N2).

Check

|  |  |  |
| --- | --- | --- |
| **Process Operated** | **Timing**  **(microseconds)** | **Results** |
| FindNeighbour | 14.0 | 8611522 points to:   * 8631524 * 11251704 * 9361783 * 12321385 * 13491586 |

The check method will verify a given route to check to see if it’s valid, taking into consideration the mode of transport.

The approach I used to get my check working was to run through all the given nodes from the command file to add them to a vector. Then run through all the nodes in a for loop and find the first node reference within the vector. Once the reference node has been found create a vector of arcs which stores all the arcs linked to the node reference within a vector. I then run through all these arcs and check to see if there is a valid path between current reference and the second reference within the vector. If there is a valid path between the two proposed nodes then it updates the first reference number to the second reference and then updates the second reference to the third reference, this process continues until each node in the path has been compared.

I believe my check method has a big O notation of O(N2). This is because of the two nested for loops within the method. The first for loop within the method loops through a single iteration of the nodes within the node vector and the second nested for loop runs through all the arcs linked to the given current node reference. This causes the first for loop to have a big O notation of O(N) and also causes the second for loop running through each arc linked to the current reference to have a big O notation of O(N). Therefore this function has an overall big O notation of O(N) \* O(N) == O(N2).

FindRoute and FindShortestRoute

|  |  |  |
| --- | --- | --- |
| **Process Operated** | **Timing**  **(microseconds)** | **Results** |
| FindNeighbour | 14.0 | 8611522 points to:   * 8631524 * 11251704 * 9361783 * 12321385 * 13491586 |

My findRoute and findShortestRoute are both the same code as they both require to do the same job, but with the only difference being that the findRoute will find a given route between the start and end reference and the findShortest finds the shortest route between two given references.

The findRoute is specified as having to find a given path between two references.There are various ways of going about finding a path between the references. A method I decided to follow to help me find a path was to use breadth first search. Breadth first search traverses the undirected node graph by searching horizontally across each reference instead of going down a linear path searching for the end node like Depth first search. Once its checked all the references across a horizontal plane it add them to a vector in order to prevent revisiting visited nodes. Then moves down a horizontal layer and now searching through all the references linked to the just searched references. This process continues until the target node has been found with references and a path has been found.

The approach I used to get my findRoute and findShortest working within my code was to run through all my node until I found the start reference, once I’ve found the start reference I find all the arcs linked to the given node and adding all these arcs to a vector. I then run through the vector checking each arc to see if they haven’t been visited before to prevent cycling through the same nodes and they have the correct mode of transport. Those arcs that are valid are added to a vector. If there are no nodes that link to the start reference which are valid then the process command returns true and fails as there’s not valid path available. Within my vector of valid nodes I order them in size of distance using a function I created that calculates distance from the current reference in the vector to the end reference and orders the vector in terms of shortest distance to the end reference.

Once I’ve ordered all the vector in size of distance I break out of the for loop, then run through all the nodes until I’ve found the first reference within the vector. I check to see if this vector has not been visited before and has the correct mode of transport. If the reference is valid I find all the arcs that the reference is connected to and add them all into a vector. I go through each reference in the vector and add valid nodes. Once I’ve added all the valid nodes to the vector I remove the current reference from the vector and add it to the visited vector. I then order the observable vector containing valid nodes into order of distance from that valid reference to the end reference.

Before I move on to the next node in the observable node I check to see if the second to last item in the visited vector and last item in the visited vector are linked to one another to check the reference I’m adding have an actual path between them. If not they’re removed from the visited node.

This whole process is repeated until the end reference is added to the observable node. If the end reference is found the path that’s been visited will be returned.

I believe my findRoute and findShortest have a big O notation of O(N5). This is because the first for loops that iterates through all the nodes has a notation of O(N). This is because it loops through each node in the vector once. The second for loop that iterates through all the observable nodes within the vector also has an notation of O(N), due to only iterating through each reference within the vector once. The for loops iterating through all the arcs linked to the current reference has a notation of O(N) and as there is a single loop it has a notation of O(N). Also the vector creation would not considerably effect the big O notation at all so it can be ignored. When checking to see if the references are linked within my visited node I run through the node this itself has a O(N) notation as it only cycles through the loop once. Finally when going through the arcs of the references within the visited node there is only a single loop meaning it is O(N). N \* N \* N \* N \* N) = N5.

Improvements

Binary Search

Throughout my code I have been using a linear search method to find items such as references. This method is long and has a time complexity of O(n). Binary search is a better approach to do the same task. While searching a sorted set of data if the value of the reference is less than the item in the middle of the data-set. Reduce the search space to the lower half. Keep repeating this process until the target reference is found. This search method is O(Log n).

Recursion

The

Conclusion