Assignment 04 : Analysis of Computer Algorithms’

Ganesh Gundekarla

11700551

1. Establishing the connection between filezilla, cell machine

Process on how implementation is done for MacOS:

Required apps:

1. FileZilla.
2. Cisco any connect client. (vpn).
3. Terminal to access cell machine.

**Step one:** Using vpn to connect to the unt server:

Here, we had used vpn to connect to the vpn.unt.edu server, to access the cell machine which is the primary step in order to access the cell machine in ubuntu.

**Step two: Connection to a cell machine in the unt server .**

In terminal we gotta type :

ssh gg0640@ CELL04-CSE.ENG.UNT.EDU

This will allow us to access a cell machine in the unt server by using our Euid login credentials. After some time, the server gets initiated.

**Step 03: Connection of a ftp file transfer protocol with the server .**

This step is important in order to upload the algorithm files we had prepared to the cell machine to set up and try out with multiple inputs.

1. Here, in the File zilla app we need to set the HOST =” CELL06-CSE.ENG.UNT.EDU”
2. Username: gg0640
3. Password: my credentials

With this the connection to the server will be complete. With this we will be able to set up a file sharing connection with the server.

The below screenshot shows the successful connection.

A screenshot of a computer

Description automatically generated

1. Listing all the cities from the graphgen.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Task one :**

I am in city “A”, can I fly to city “B” with less than x connections? Give me the

route with the smallest number of connections or tell me there is no such a route.

Inputs for this task are as follows .

**First input :**

It is the total no of cities to be considered : 140

**Second input :**

This is the type of route search we should be performing and it’s a list of choices which requests us , in this case , it is gonna be one.

Inputs : after this , we need to provide the city numbers of one and two and the number of connections that we cannot exceed as being asked by the question .

A screenshot of a computer

Description automatically generated

**Task two :**

Give me the route with the smallest number of connections from city “A” to city

“D” through city “B” and “C”. (the order of “B” and “C” is not important). Or

tell me there is no such a route

**Sample of inputs :**

In the task two , the first and the second inputs are same as the above and for the second one , we should be choosing the choice 2 which implies the task 2 for performing the route search 2 function for this one .

**Inputs**

Here, the inputs for my case are cites a , b , c, d indices aka numbers all should be given for finding the route through appropriate nodes only .

Output: it gives us on how the cities are reached one by one as shown in the output .

A screenshot of a computer

Description automatically generated**Task** 3 :

I want to start from city “A”, visit all the cities that can be reached and then come back to “A” using as less connections as possible. Give me a route or tell me there is no such a route. (note: once you come back to “A”, you do not go out anymore)

Inputs :

Same for the task one and two are being repeated here .

Here , the choice = 3 is to be selected.

For this one , we need to give the start and the city end index which is the same city number . so essentially , it is telling you to start from a city and return to the same city which is not possible in my case .

A screenshot of a computer

Description automatically generated

**Task four:**

Inputs are:

Cities list: 140:

Here , the next inputs are the three cities of our friends , which then selectes a different city to meet so that a minimization of the connections occurs between us three.

Output is given below .

A screenshot of a computer

Description automatically generated

**Part two :**

1. Design issues :
2. Here, I found one design issue in the graph class , the number of vertices or nodes and the list of the adjacent vertices are being kept private requiring two functions.
3. Here , coming to the loginc , the design of routesearch() function is to handle the inputs for different tasks and a case structure is being done to choose between differencr tasks .
4. Data structures used :
5. I had used data structures like min priority queue , stack , arrays .
6. Standard c++ library implementations is used for min priority queue and stack .
7. For dijkstra’s algorithm , min priority queue is used.
8. Stack is used for printing the routes.
9. Algorithms used :
10. Dijkstra’s algo is used to find shortest paths .
11. Greedy approach in task 3 is done to visit max possible nodes.
12. For other tasks , simple logic is being done .

Running time analysis :

1. Dijkstra’s algorithm implementation will be using adjacency matrix and a priority queue in a binary heap , so it has the worst case time complexity of O(n2 logn) and pushing and popping of an element will be taking an O(logn) for a queue .
2. Route search 1 :

Dijkstra’s algo : O(n2 log n)

Path is being reconstructed of O(n2 ) for insertion of each city in the path for the beginning of the vector .

Output path is O(n)

Total time complexity. : O(n2 log n).

1. Route search 2 :
2. Here , multiple dijkstra’s calls are used, the dijkstra’s algorithm is basically upto a six times which is dependant on the existence of the paths between the cities .
3. Total time complexity would be six time the O(n^2 log n) which is O(n^2 log n)
4. Path reconstruction is done upto a 6 timers , contributing overall O(n^2)
5. Final steps would be comparing the two integers and iterating it over a list of n elements which contributes lower order terms .
6. Here , the dominating factor in the time complexity for task 2 is the mulpile calls to dijkstra’s algorithm each having a time complexity of O(n2 log n) , the overall is also same .This would be making the function inefficient for very large graphs and if the path involving multiple cities. Optimization would be using more efficient data structures and more route planning .

D. Route search 3 :

Here ,the initial dijkstra’s calls , the dijkstra’s algorithm is then called once for a each city that is visited until and then all the reachable cities have been visited , in the worst case the graph is then fully connected and every city is then getting reachable from every other algorithm could be invoked upto n times leading to O(n times n^2 log n) = O(n^3 log n) complexity.

Final call would be adding another O(n^2 log n), this will be effecting the overall complexity anyway .

When coming to loop and tour updates , the sum of operations for finding the next city and the updation across all iterations will contribute an extra O(n^2), but this is also being equally dominated by the O(n^3 log n) complexity from the repeated Dijkstra calls which are 6 total .

Dominant factor : repeated invocating of the dijkstra’s algo done upto n time leading upto time complexity of O(n^3 log n) which is inefficient for large graphs .

1. Route search 4 .

Here , in the dijkstra’s algorithm , three calls are being made , and the each call is having a time complexity of O(n^2 log n)

Meeting city cost: O(n) .

Printing routes : assume worst case where the route to the city of meeting could be travelling the entire set of cities and the total time complexity would be for printing all routes is O(N) .

Dominant factor: multiple invoactions of dijkstra’s algorithm .

Overall complexity is : O(n^2 log n).

Primarily due to adjacency matrix making it inefficient .

Scalability:

1. Here, the current implementation of this will be facing some scalability issues with a very large graphs due to its reliance on the dijkstra’s algorithm and the adjacency matrix .
2. Here , the efficiency could be improved by implementation of more efficient data structures and algorithms tailored to the basic characteristics of the graphs .

Remarks

1. Here ,the balancing simplicity and the efficiency of the algo is effectively crucial , especially when dealing with a large scale geographic data .
2. Here , continuous evaluation and optimization is being essential for improving the scalability and the performance .