MINI-PROJECT: COVID TRACING
GROUP NUMBER: 28

MEMBERS
Harshit Gupta (2020114017)
N Harsha Vardhan (2020111022)
Rohan Chowdary V M (2020101130)

FILES INCLUDED

The coded ADT files in this project are-

1. extra.c and extra.h- Code related to list, stack and queue related operations.
2. graph.c and graph.h- Code related to generating a graph and performing necessary operations to find the safest and shortest path efficiently. **(Part 2 of Major Tasks)**
3. personll.c and personll.h- Code related to station and person querying and also maintaining linked lists of certain people who are possibly infected with the virus. (**Part 3 of Major Tasks)**
4. list.c- Code related to finding primary and secondary contacts of a covid positive person are included here. It also includes general code to reset values and do the essentials before going to the next day of the simulation. **(Part 1 of Major Tasks)**
5. main.c- Code which takes in input in the desired manner. It is coded in a 'Menu' like format. The format of input is mentioned in the 'README.md' file.

DATA STRUCTURES USED

The data structures used in the project are-

1. Arrays
2. Linked Lists
3. Stacks
4. Queues
5. Graphs

ALGORITHMS USED

We used linked lists, stacks and queues for the most part of the project and many functions are hence utility functions for linked lists. Several other algorithms use stacks and queues to store the data and utilize them when needed. The time complexities for insertion, deletion and access are **O(1), O(1), O(n)** respectively.
The overall time complexities of the queries regarding stations and people is **O(n)**.
The time complexity of the functions to find the primary and secondary contacts was also of the order **O(n*logn).**
We sorted the paths between source and destination (given as input) based on their safety value and road length using the 'qsort()' inbuilt 'C' function. The time complexity of this sorting function is **O(n*logn)**.

To find all the paths from the input source to the destination, depth-first traversal is used. If the vertex is unvisited, it is set to visited and pushed into the stack. If the destination is reached, the stack is printed and the destination is popped from the stack. If the destination is not reached, depth, we recursively call the depth first search and input in various needed parameters. If there are no more connected vertices to the current vertex, the current vertex is popped from the stack and the while loop is broken. On the other hand, if there is another connected vertex, then the whole process mentioned above is repeated until the stack is empty or the loop is broken in the non-recursive call. The time complexity for the above algorithm would be **O(V+E)**.

DIVISION OF WORK

Due to the absence of 2 team members, only the 3 remaining members contributed to the whole project. The 3 members and their contribution are listed below and it is to be noted that each person received sufficient help and support from the other 2 to make this project compile without any errors and give the desired output in the self-generated test cases.

1. Harshit Gupta-
    a. Contributed to the final report.
    b. Coded the functions and structures for lists, stacks and queues (as seen in extra.c and extra.h).
    c. Coded graphs, vertices and edges and created necessary functions to generate and store undirected weighted graphs and find all paths between the input source and destination.
    d. Cleaned up the code and added comments to files ‘extra.c’, ‘graph.c’ and removed useless variables and made the code understandable.
    e. Coded the ‘main.c’ to make it further interactive and appealing to the viewer.
    f. Contributed to the code for finding the primary and secondary contacts with appropriate logic and no collisions between the people.
    g. Code the safety values by deriving necessary information from other files and using the formula appropriately.
2. N Harsha Vardhan
    a. Contributed to the README and final report.
    b. Helped in coding the graph.c file by replacing the bubble sort algorithm with a more suitable(complexity wise) qsort() algorithm.
    c. Ensured that variables or functions were not being redefined while compiling the code.
    d. Coded the ‘personll.c’ file which stores all the data and variable attributes regarding the people and stations involved in the entire simulation.
    e. Added understandable and readable comments to ‘personll.c’ file. Tested out the function in the file and ensured they were working perfectly without giving errors of any kind.
    f. Contributed to the working of the functions in ‘list.c’ file by testing it out thoroughly.
3. Rohan Chowdary V M
    a. Contributed to the README and final report.
    b. Ensured the functions in ‘graph.c’ were working perfectly with proper indexing and were not segfaulting.
    c. Tested edge cases in the final code and made the necessary changes.

d. Contributed in resolving errors in finding primary and secondary contacts of the COVID positive list. Significantly contributed in brainstorming and coding in ‘list.c’ file.
e. Wrote the overall input in ‘main.c’ and made sure that the base condition for number of people and number of stations wasn't being violated in any part of the code.

GITHUB REPO LINK

https://github.com/TheAthleticCoder/IIITH-PROJ.git
