International Institute of Information Technology, Hyderabad

Data Structures and Algorithms

S21CS1.201

Prof. Ravi Kiran S & Prof. Sujit P. Gujar

COVID-Tracing Mini-Project

Akshat Goyal

Team 6

Aryan Gupta (2020101091)

Khushi Agarwal (2020101092)

Mancharla Harish (2020102062)

Tanniru Abhinav (2020112007)

Vayur Shanbhag (2020112027)

Github Link: `https://github.com/thundermage117/COVID-Tracing`

1 Overview

For this project, we are given a some stations and there are some bidirectional roads connecting some of them such that there exists at least one path through any two stations. We have modelled this problem as a graph using Adjacency lists.

Then, we are given where each person is present, and this concludes the initial status of the graph.

People are present at various stations and user is given an option to move them around. Then on a later day, user can declare a list L of people that were Covid positive, and then we must trace back all the happenings since the day people got infected and then print all the primary and secondary contacts of the list for all the days since that day to the current day.

To do this, we maintain a day struct that stores the status of the whole graph for each day, and also stores the changes that took place on each day.

Thus, when we are asked to back trace, we can use the struct of the first day, change the status of the person and then makes changes for all the days accordingly.

We will know exactly what changes took place for each day as we have stored them in our day struct. Using this we will proceed to update the day struct, using the same functions we used to initially update to update the status according to new information provided.

We have also implemented functionality to check the status of a person or station on the current day. User can give the id of a person and then the person's status will be printed, or user can give the id of a station and the station's status will be printed.

2 Data Structures Used:

1. Min-Heap: A min-heap is used for finding the safest and shortest path as specified in query 2.

   It performs heapify operation in $\log(n)$ time.

2. Linked List: Linked lists are used at various places to list all the people present at the station.

   Linked lists are also used to describe the route taken by various people to go from one station to another.

3. Array: Arrays are used to store all elementary forms of data and are also used to implement other data structures such as queues and adjacency list.

4. Hash Table : Hash Table are used to store the status of each day for 16 days at a time. As we surpass that limit, day day struct is overwritten, starting from the lowest day.

5. Adjacency list: Adjecency list is used to store the graph connecting stations.

3 Algorithms Used:

1. Update Functions:

   - Utility: Whenever we are given query 2 (A person travels from one station to another), this algorithm is used.

     Using this algorithm, we go to each station as specified by the person's route and then update the status of all the people present there.

     We also store the route taken by the person as we will need it to backtrace once query 1 is given.
   - Time complexity: This operation takes on average $O(n \cdot l)$ time, where n is the number of stations the person visited to reach his final destination and l is the average number of people on those station.
   - Memory Complexity: $O(1)$

2. Backtrace functions:

   - Utility: When query 1 is invoked (Given list L of covid positive people, print their primary and secondary contacts), this algorithm is used.

     Using this algorithm, we first update the status of all the people on the list as well as the status of all the people present on the same station on the day people of list L became covid positive.

     Then we call the Update functions on that day onwards, using the updated status of all the people and update for all the paths taken by all the people in the past X days.

     At the end of each day, we also copy the current day struct into the next day struct to have a state table of the changes on each day and traverse through all the people to print primary and secondary contacts of the people on the list L.
   - Time complexity: This operation takes on average $O(V \cdot X + N \cdot X)$ time, where V is the number of stations, N is the number of people, and X is the given input value of X.
   - Memory Complexity: $O(1)$

3. Copy functions:

   - Utility: We must maintain the state of each day, therefore at the end of a day, we copy the struct of the current day into the next day and work with the copied struct.
   - Time complexity: This operation takes $O(V)$ time, where V is the total number of stations in the graph.
   - Memory Complexity: $O(1)$

1. Query 1:
    - Time complexity: $O((V+N)\cdot X)$ where V is the number of stations, N is number of people,X is the number of days for which query is asked.
    - Memory Complexity: O(1)
2. Query 2:
    - Time complexity: $O((V+E)(logV))$
    - Memory Complexity: $O(x)$, where x is no of stations in longest path.
3. Query 3:
    - Time complexity: $O(1)$
    - Memory Complexity: $O(1)$
4. Query 4:
    - Time complexity: $O(1)$
    - Memory Complexity: $O(1)$

3-way Dijkstra Implementation(Query 2):

- Utility: We first define a new edge weight which is defined as if we are going from A to B , then the cost for this operation is $100000 \times dangervalue(B) + dist(A,B)$.

  We do this so that while applying Dijkstra algorithm, this first priority is always given to danger value and only if the danger value is same for two paths, the distance between the stations is considered. We are able to achieve this just because of the sheer value of 100000 which ensures that whatsoever distance between two stations might be, the priority is always given to danger value.

  Now initially we apply the simple Dijkstra algorithm in $O(nlogn)$ using min heaps and we also store the parent of each node and so now we retrace the path for the shortest path using the parent array.

  For the second shortest path we iterate through all the nodes of the the shortest path and now for each of the nodes ,we look at the connecting nodes and using this find the 2^{nd} shortest path and a possible contender of the third shortest path. Meanwhile we also some conditions like there should not be any two-way cycle and some other conditions like from the neighbouring node, the target node should be reachable, etc.

  Now we trace back the second shortest path(if such a path exits else we return).

  Now we repeat the same process with the second shortest path and find the third shortest path is such a path exists and we just compare the total

cost of travel with the possibe contender we obtained in step 2, whichever is lower is considered as the third shortest path and again we retrace the path using the parent array and the second shortest path.

- Time complexity: $O(nlogn)$ (including tracing the path)
- Memory Complexity: $O(n)$

4 Division Of Work:

- Aryan:
    1. 3-way path finding Algorithm: Implemented query 2 using a modified dijkstra algorithm.
    2. main.c
    3. Error handling by including while loops, printf and scanf statements.
- Harish:
    1. main.c.
    2. general purpose functions: printing query 1, station info, station details, getting worst affected person, printing station details.
- Khushi:
    1. Update functions.
    2. Backtrace functions.
    3. Update functions: copy day function, day increment function
- Abhinav:
    1. General purpose functions: Getting the danger index for some station after making changes, printing the status of a person, deleting and adding node in the day struct
    2. Graph implementation
    3. Report writing
- Vayur:
    1. Update functions: Updating data,station,person,route,removing person,adding person,updating day,query 2 update
    2. Structs : Made all the structs in the header.h file
    3. Backtrace function

5 Error Handling

We handled various errors like:

1. Wrong input format: If the input is not given in the correct format, then we give an error message and ask him to re-enter the input.
2. Out of bounds: If the given input is less than 0 or more than the limit of that quantity then we print an error message to reenter the inputs within the bounds.
3. In Query 2, if the person is not present in the source station then also we tell the user that this movement is not possible.
4. Again in Query 2, if the person who has to move is quarantined, then again we tell the user that this movement is not possible.