COP3530 Project 3 (VaccinApp) Report
Group #54
Samer Khatib
GitHub Repository
Video Demonstration

Final Project Proposal

Problem: Lack of public awareness regarding growing access to COVID vaccinations in Gainesville (for individuals not related to the University of Florida).

Motivation: As vaccines are distributed throughout Gainesville, specific data regarding the scheduling of a vaccination is becoming more involved to achieve. Finding a source where all the information needed about scheduling a vaccination in an easy, straightforward way would help people know where and how to get vaccinated in Gainesville.

Features: I will solve the problem by developing an open-sourced command-line based application that will:

- Show a non-interactive chart of vaccination sites in a given area
- Give direct links to scheduling a vaccine
- Who can get the vaccine at the given location

Data: I will be utilizing several data sets to create my application. These datasets will be regarding information about the COVID vaccination sites around Gainesville. These numbers and locations will help me build a command-line application in which a graph is drawn about all the vaccination sites in Gainesville. Additionally, since there are not 100,000 vaccination sites around Gainesville, I will also automatically generate randomized data based on vaccination sites around the United States.

Datasets

- https://www.kaggle.com/padmajabuggaveeti/covid-vaccination-dataset-2021
- https://www.kaggle.com/gpreda/covid-world-vaccination-progress
- https://www.kaggle.com/gpreda/covid-19-vaccination-progress

Tools: The programming language that I will be using to develop the application is C++. I also utilized a library to download and convert .json files to .csv files for easier access using standard C++ operations. I used no other external libraries throughout the project.

Strategy: I implemented a **min-heap** with pair values indicating the closest vaccination site to an individual. When the code is run, a graph is built location by location by popping off the values in the heap to load in locations based on proximity to the user. I compared these results to the implementation of an **ordered map,** where I remove and delete the first value to build the graph. I will use these two methods to determine which data structure will provide a lower time complexity and execution time to construct the graph.

Data Structures Used: When developing the min-heap, I used a vector that I filled with float, string pairs to save the distance of each location from the user and the information about the location. These pairs were useful for allowing me to easily manipulate and change the information stored for each location when I needed to do so. For developing the ordered map, I used a red-black tree with rotations that helped me keep the map balanced and allowed me to store float, string pairs in the same manner the information was stored in the min-heap. By using an inorder traversal, I was able to print the values in the same order as with the minheap.

Distribution of Responsibility and Roles: Samer will be responsible for all of it.

Project Analysis

Changes in the Project: Initially, I wanted to utilize Dart to create a fully functional application that drew a graph of the closest vaccine locations using the Google Maps API. After my group left, I quickly realized that developing a fully-fledged application alone given the 1-month timeline was unreasonable, so I switched my attention to using a more familiar language. Moving forward, I wanted to use C++ to create a simplified version of the original concept. I settled on making a Command-Line-based chart that would list the closest vaccination locations, giving the user the distance, the name of the location, the address, and the website. This allowed me to devote more time to refining the data structures I was creating for the project, instead of primarily focusing on the implementation of the data structures within an app.

Time Complexity of Major Functions:

MinHeap (where n is the number of float, string pairs)

pop(): The time complexity of popping off the minimum value has a time complexity of O(logn) in the worst case. This is because in the worst case, the heap has to rotate a log(n) number of nodes to maintain the heap properties by heapifying the nodes.

insert(): The time complexity of inserting an element has the worst time complexity of O(logn). This is because much like popping the nodes, you have to heapify a log(n) number of nodes to maintain the heap.

HeapifyUp()/HeapifyDown(): The time complexity running heapifying on the heap is log(n) in the worst case since you start at the top/bottom of the heap and work your way up/down the heap, dividing the number of nodes that need to be viewed in half every iteration. This means that you end up viewing a log(n) number of float, string pairs in the worst case to correct the heap (and maintain the heap properties).

Ordered Map (Red-Black Tree) (Where n is the number of nodes)

insert(): For the ordered map, inserting a new element has a time complexity of O(logn) in the worst instance. This is because when you insert a new element, you start at the root node, and worst your way down the tree by using the Binary Search Tree properties. This means that in every instance, you divide the number of observable nodes by 2 until you either find the node that has the same value as the inserted node, or you find the appropriate place in the tree to place the node.

traverseInorder(): Since you have to traverse the entire tree, the time complexity of traversing inorder would be O(n).

TreeRecolor(): The time complexity of the tree recolor function is O(logn) in the worst case. This function is called when the uncle of a newly inserted red node is red. This causes me to recursively recolor the nodes in the tree (starting from the newly inserted red node) until I reach the root. Therefore, in the worst case, you end up accessing O(logn/2) nodes, but since you drop the constant, the time complexity becomes O(logn).

Project Reflection

Group Experience: Since my group left me, I did not have a great group experience. From the beginning, there was minimal communication until my group members completely stopped responding to my attempts to contact them. From there, I contacted Professor Kapoor and he informed me that it was likely that they were dropping the class. To this day, I have no idea what the real reason why stopped responding to me was, and I can only assume that it was for the reasons Professor Kapoor informed me of.

Individual Experience: I found modeling the data structures that we have discussed in class harder than I originally expected. I assume this is due to the sheer amount of data that I ended up inserting into my heap and map. The large data amounts caused difficulty keeping track of all the rotations that occurred and overall made the project significantly more difficult.

Challenges: The largest challenge I faced when doing this project was the lack of communication between my group members. Due to the limited communication, the initiation of the project was delayed several weeks, and a few days after we completed the proposal, my group members stopped responding to me entirely. Pertaining to the actual development of the project, I encountered a stack overflow error for some time when developing the Ordered Map. After significant debugging, I discovered that it was due to an infinite loop that occurred when I attempted to receive the uncle of a node that had the wrong parent. This resulted in the search for the uncle to recur indefinitely due to the fact that my getUncle() function verified the parentage in relation to the grandparent, and if the parent was not set correctly, the search would continue forever.

If I were to start once again as a group: I would contact my group members earlier and complete the proposal and early development of the project before they left. Therefore, I would

at least have a head start on the concepts of the project itself, and finishing the original idea (full-fledged flutter app) would have been more accessible towards the end of the project.
What Samer learned throughout the Process: I learned that the usage of git version control is incredibly handy when working on a group project of this scale. I feel as if git should be taught with the curriculum of either COP3530 or COP3503, since it is not very complicated, and offers users great benefits in terms of branching so that each group member can work on their own section of the project which can be merged into the master branch once the work is complete.

References

- Stepik Heaps – Step 1 (https://stepik.org/lesson/390629/step/1?unit=379729)
- Red-Black Tree (https://www.geeksforgeeks.org/red-black-tree-set-2-insert/)
- Red-Black Tree (https://www.geeksforgeeks.org/c-program-red-black-tree-insertion/)
- Vaccine Data API (https://www.vaccinespotter.org/api/)
- Function used to get Distance between two Coordinate Points (Haversine formula)
- Project 1: Gator AVL (https://ufl.instructure.com/courses/419043/assignments/4548335)
- Extra credit: Hashmap vs Treemap [used to find time differences between data structures] (https://ufl.instructure.com/courses/419043/assignments/4548325)
- Opening a .pdf file from within the code (https://stackoverflow.com/questions/2745812/opening-a-document-programmatically-in-c)
- Formatting Visually Appealing Table (https://stackoverflow.com/questions/14765155/how-can-i-easily-format-my-data-table-in-c)