CSCI362 Analysis Homework

Instructions

The question below describes a complex computing problem. Propose *TWO* different, reasonable solutions to how you would store the data described in this problem. Analyze each solution by discussing the operations that will need to be performed on the data, how often you think those operations will need to be performed, and what the time complexity of those operations would be. Finish with a discussion of which solution you prefer and why. Consider time complexity and, if relevant, other factors in this decision.

An example problem and solution is provided for you to use as a model.

Example Problem

When people arrive at a restaurant, a host records their name and the number of people in their group. When a table becomes available, the host pages the person who has been waiting the longest among people whose group sizes would make sense at that table.

Example Solution

My first idea is to create a Reservation class with fields m_name and m_numPeople. I would store a std::list<Reservation>. Whenever someone arrives, we would make a new Reservation and push it onto the back of the list. When a table becomes available, I would iterate through the list from front to back, stopping as soon as I find a Reservation whose m_numPeople is less than or equal to the size of the available table. I would erase that Reservation from the list, and page the m_name stored within it. This list is functioning very similarly to a queue, except that I take the first-in among the small-enough elements.

Creating a new Reservation would happen in constant time, since it involves calling push_back on a list. Finding the correct Reservation to erase might take linear time because the only one with an appropriate size might be at the very end. But this seems like it would be very uncommon. Actually erasing the Reservation would be constant time, because it involves calling erase on a list.

My second idea is to create a std::map<int, std::queue<std::string>>. The keys in the map are group sizes, while the values are queues of names. When someone arrives we would get the appropriate queue for their group size and add their name to it. When a table becomes available we would first look in the queue of its size. If that queue is not empty, we would pop the front element off of it and page that person. Otherwise we would try the next smallest size, down to a minimum of 1.

Looking up the appropriate queue to add someone to would be logarithmic in the number of different group sizes, which can be assumed to be fairly small. Actually adding to the appropriate queue would be constant time. In the worst case, choosing a person to page when a table becomes available might involve searching k queues, where k is the size of that table. Both checking whether or not a queue is empty and removing something from it can be done in constant time.

Both of these approaches are reasonably fast, but the second one's runtime does not depend in any way on the number of people waiting, which seems overall faster. Some experiments with real-world data about numbers of customers and variance in group and table sizes would be needed to determine this for sure. It should be noted that the two approaches do not interpret appropriate group sizes in the same way. The first solution would be happy to put a single person at a table for 10, if they have been waiting the longest. In this way, it prioritizes fairness. The second solution would try to give that table to a larger party, even if they have not been waiting as long. It prioritizes efficient use of the restaurant's tables. This seems more sensible to me, so I would go with the second solution.

Your Problem

You are a busy person who meets lots of different people. You want to keep track of which of the people who you know also know each other. You want to be able to type in lines that contain groups of people who all know each other. Then you want the program to print one line for each person, listing first their name and then the names of all the other people they know. Both the ordering of the lines and the ordering of other people on each line should be alphabetical.

Example Input:

Alice Bob
Bob Cathy Debra
Alice Debra Elizabeth

Example Output:

Alice: Bob Debra Elizabeth
Bob: Alice Cathy Debra
Cathy: Bob Debra
Debra: Alice Bob Cathy Elizabeth
Elizabeth: Alice Debra

CSCI362 Analysis Homework Rubric

	Excellent	Good	Fair	Poor
First Solution	A clearly explained, sensible way to solve the problem. (20)	A clearly explained way to solve the problem, but one that an experienced developer would not propose. (15)	A clearly explained solution that does not correctly solve some aspects of the problem. (10)	A solution that is not explained clearly or does not obviously work for the problem. (5)
First Analysis	A clear and correct analysis that incorporates all relevant factors. (20)	A clear and correct but slightly incomplete analysis. (15)	An analysis that is significantly incomplete, or slightly incorrect. (10)	An analysis that is significantly incorrect or insufficient. (5)
Second Solution	Same description as First Solution. (20)	Same description as First Solution. (15)	Same description as First Solution. (10)	Same description as First Solution. (5)
Second Analysis	Same description as First Analysis. (20)	Same description as First Analysis. (15)	Same description as First Analysis. (15)	Same description as First Analysis. (15)
Recommendation	A clear, correct, and complete explanation of the relative merits of the two approaches. (20)	A clear, correct, but mildly incomplete explanation of the relative merits of the two approaches. (15)	A mildly unclear or incorrect or severely incomplete explanation. (10)	A severely unclear or incorrect explanation. (5)