CS301 HOMEWORK 2

Sabanci University

Mert Ziya

27800

QUESTION 1:

Every year, we send some of CS301 students to the "Center of Advanced Algorithms" in Westeros for a scientific visit, where all expenses are paid by the university. We send the most successful students in CS301 to this visit. We measure the success based on overall numeric grade in CS301. The number of students selected for this visit depends on the budget available, changes from one year to another. Since the number of students taking CS301 is increasing, we want to decide the students that we will send for this visit automatically by using an algorithm. This algorithm should use the student information (e.g. student id and CS301 overall numeric grade of the student) and the number of students that will be sent to the visit. Let's say there are n students in CS301, and we will send m of these students to the visit. The ties that we might have between the students with the same grade will be broken randomly.

a) Please design an algorithm, as efficient as possible, which we can use for this purpose. (Do not write any code (pseudo or actual). Please only explain how you would solve this problem in a couple of sentences.)

1. According to question, sorting needs to be done with respect to student's "overall numeric grade". I assumed that the lowest grade can be 0.00 and highest grade can be 100.00 because this is the format for the "overall numeric grade" used in the courses of Sabancı University (in the decimal part there can be 2 numbers).
2. for sorting the array i would use the logic of "Radix sort". First, i would multiply the each student's grade by 100 in the Array 'A'. Then, i would create an array of size 9(call it 'bin'). Each element in this array should be a 'Queue'. (Multiplying by 100 may be unnecessary but it does not change the complexity) (Important part is going from least significant digit to most significant digit.)
3. Highest grade became 10000 after the previous part that's why we need to iterate 5 times. For each iteration i would divide the grade by (iterationNumber-1)*10 [lsd to msd] and take the mod(10) of the grade. Then, i would place it to the corresponding index inside the 'bin'. When each iteration is done i would replace the elements inside the 'bin' in the 'FIFO' format. (in the final(5th) iteration, if there are equal numbers in the queue i would randomly-instead of FIFO- give priority to one of the equal numbers).
4. Finally, i would iterate through the array 'A', m times and the first 'm' element gives us the desired output (students that needs to go "Center of Advanced Algorithms")

(b) Give an upper bound for the run time complexity of your algorithm (as tight as possible).

2nd part of question a) tooks 'n+9' time (iterate through an array of size n for 1 time and create an array of size '9'). 3rd part took "around" '20n' time ('iterate', 'divide', 'take mod', 'place it to bin' for each element in the array for 5 times). 4th part tooks 'm' time (iterate the first m element of the array). So the total complexity of this algorithm is (n+9)+(20n)+(m)= (21N+M+9). Since $m < n$ and 20 and 9 is constant, the upper bound is O(n).

QUESTION 2:

Suppose that you have n friends $F = f_1, f_2, \ldots, f_n$. Some of your friends know each other. If your friends f_i and f_j know each other, they follow each other on social media, and they can see the messages posted for each other. So, when you send a message to your friend f_i, if f_j is a friend of f_i, f_j will also see this message coming from you to f_i.
On April 23 (National Sovereignty and Children's Day in Turkey), you want to send a message to all your friends. However, if you send the same message to all your friends, those friends of yours f_i and f_j who are also friends of each other will see that you are sending the same message to them. Hence, they would not feel special, getting such a general message from you.
However, sending each friend a different message would mean writing n different messages, which is not easy.
Then you consider the following. If you write the same message to two of your friends f_i and f_j who don't know each other, then since they would not see this same message, they would still feel special. Using this idea, you can reduce the number of different messages that you need to write.
We can state this problem as an optimization problem as follows:

Problem Definition (optimization version):

Given your friends $F = \{f_1, f_2, \ldots, f_n\}$ and for each pair of your friends $f_i, f_j \in F$, whether f_i and f_j know each other or not, what is the minimum number of different messages that you need to write, so that no two of your friends who know each other will get the same message

Please state the same problem as a decision problem:

Given a set of friends $F = f_1, f_2, \ldots, f_n$, a graph $G = (V, E)$ where $V = F$ and E consists of edges (f_i, f_j) representing that friends f_i and f_j know each other, and an integer k, is it possible to assign messages $M = m_1, m_2, \ldots, m_k$ to the vertices in V, with k or fewer different messages, such that no two adjacent vertices in G receive the same message?

QUESTION 3:

a) A red-black tree insertion requires O(1) rotations in the worst case.

The statement is TRUE. Because, The worst-case scenario for a red-black tree insertion takes at most 2 rotations and 2 is constant(O(1)). When a new node is inserted, it is initially colored red, which might violate the property that no red node can have a red parent or red child. To restore the tree properties, we consider the relationship between the inserted node (N), its parent (P), its uncle (U), and its grandparent (G). There are two scenarios for worst case(2 rotations) to happen and they are:

1. N is the right child of P, and P is the left child of G (Right-Left case): Perform a left rotation on P. Now, N becomes the left child of G, and the problem becomes the Left-Left case.
2. N is the left child of P, and P is the left child of G (Left-Left case): Perform a right rotation on G. Then, swap the colors of P and G.

b) A red-black tree insertion requires O(1) node recoloring in the worst case.

The statement is FALSE. A red-black tree insertion may require O(log n) node recoloring in the worst case, not O(1). This is because after an insertion, the tree needs to maintain its properties, including the equal number of black nodes on all paths from the root to a null leaf. Restoring these properties can involve recoloring nodes, and in the worst case, a chain of violations may need to be resolved, which can involve recoloring nodes up to the height of the tree, which is O(log n).

c) Walking a red-black tree with n nodes in pre-order takes (n lg n).

The statement is FALSE. Walking a red-black tree with n nodes in pre-order takes $\Theta(n)$ time, not $\Theta(n \lg n)$. Pre-order traversal visits each node once, and the time complexity of a traversal is determined by the number of nodes in the tree. Since there are n nodes, the time complexity is $\Theta(n)$ not $\Theta(nlgn)$.

QUESTION 4:

a) What does NP stand for?

NP stands for "Nondeterministically Polynomial". It is a complexity class used in computational complexity theory to classify decision problems that can be verified by a deterministic Turing machine in polynomial time.

b) When do we say a problem is in NP?

We say a problem is in NP, if given a guess (solution) 'g' for a problem 'P', and the solution 'g' can be checked for being a valid solution to 'P' in time $O(n^k)$, where n is the size of the input and k is a constant, then problem 'P' is said to be an NP problem. This definition emphasizes the polynomial-time verification of the solution's correctness.