CS 218, Winter 2024, **Due: 11:59pm, Thu 02/29, 2024**
Assignment #4 [1] **Basic Programming Due: 11:59pm, Fri 02/23, 2024**
Name: **Student ID:**

Deadline. The homework is due **11:59pm, Thu 02/29, 2024**. You must submit your solutions (in pdf format generated by LaTeX) via GradeScope. The training programming assignment is due earlier, see more details below.

Late Policy. You have up to four grace days (calendar day) for the entire quarter. You don't lose any point by using grace days. If you decide to use grace days, please specify how many grace days you would like to use and the reason at the beginning of your submission.

Collaboration Policy. You can discuss the homework solutions with your classmates. You can get help from the instructor, but only after you have thought about the problems on your own. It is OK to get inspiration (but not solutions) from books or online resources, again after you have carefully thought about the problems on your own. However, you **cannot copy anything from other source**. You **cannot** share your solution/code with anyone else. You **cannot** read other's solution/code. If you use any reference or webpage, or discussed with anyone, you must cite it fully and completely (e.g., *I used case 2 in the examples in the Wikipedia page* `https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)` *about Master theorem for Problem 3*, or *I discussed problem 2 with Alice and Bob.*). *If you use ChatGPT or similar AI tools, you have to attach the full conversation to make it clear what help you obtained from them.* **Otherwise it will be considered cheating.** We reserve the right to deduct points for using such material beyond reason. **You must write up your solution independently, and type your answers word by word on your own: close the book/notes/online resources when you are writing your final answers.**

Write-up. Please use LaTeX to prepare your solutions. For all problems, **please explain how you get the answer** instead of directly giving the final answer, except for some special cases (will be specified in the problem). For all algorithm design problems, please describe using **natural language**. You could present pseudocode if you think that helps to illustrate your idea. Please do not only give some code without explanation. **In grading, we will reward not only correctness but also clarity and simplicity. To avoid losing points for hard-to-understand solutions, you should make sure your solutions are either self-explanatory or contains good explanations.** See some more details here `https://www.cs.ucr.edu/~ygu/teaching/218/W24/assignments/index.html`

Programming Problems. You will need to submit your code on CodeForces (a readme file about submitting code is available on the course webpage). You also need to submit a short report through GradeScope along with your solutions of the written assignments. In the report, you need to **specify your submission id**, describe the algorithm you designed, and show cost analysis if necessary. Note that your code will be automatically saved by CodeForces, so you do not need to submit the code again. For each problem, there will be 10-20 test cases. **For the training programming problems, you need to finish before 11:59pm, Fri 02/23, 2024.** With reasonable implementation, using C++ or Java is guaranteed to be able to pass all tests. You can use other languages, but it's not guaranteed that the implementations can be within the time limit.

[1] Some of the problems are adapted from existing problems from online sources. Thanks to the original authors.

1 (2.5pts) Weight-balanced trees

We've learned a lot of different balance binary trees in previous algorithm courses. This time let's look at the weight-balanced tree.

A weight-balanced tree is a balanced binary tree. For each node u, we define the *weight* $w(u)$ of it to be 1+ its subtree size. Note that this means that an empty node has weight 1. A weight-balanced tree guarantees that for any node u, the weight of any of its child is at least a factor of α of $w(u)$. In particular, if we use $lc(u)$ and $rc(u)$ to denote the left and right children of node u, we have $\alpha < w(lc(u))/w(u) < 1-\alpha$ and $\alpha < w(rc(u))/w(u) < 1-\alpha$. It has been proved that for $\frac{2}{11} < \alpha \leq 1 - \frac{1}{\sqrt{2}}$, any insertion and deletion can be implemented on weight-balanced trees using just single and double rotations. In this problem, you can assume α is a constant.

In the following, we will use n to denote the size of the entire tree. We may use n' to denote the size of a certain subtree.

Questions:

1. (0.2pts) Prove that for any node u, $w(u) = w(lc(u)) + w(rc(u))$.
2. (0.5pts) Prove that for a constant α, a weight balanced tree of size n has height $O(\log n)$.
3. Now consider we don't use rotations upon imbalance, but instead re-build the tree. That is to say, after insertion/deletion, if we find out any subtree t is unbalanced, we simply flatten all elements in this subtree into an array A, then we build an (almost) perfectly-balanced tree t' to replace t.
   (a) (0.3pts) For a subtree of size n' (probably unbalanced), what is the total cost to re-build it to be (almost) perfectly balanced?
   (b) (0.4pts) An insertion/deletion may cause multiple subtrees to be unbalanced. Prove that you only need to re-build once (re-build one subtree) to rebalance the entire tree.
   (c) (0.7pts) It's easy to see that in the worst case, an insertion may cause $O(n)$ time because it may result in rebuilding the entire tree. But what about the amortized case? Prove that the amortized cost of rebuilding for each insertion is a constant.
   (d) (0.4pts) Prove that the amortized cost for each insertion is $O(\log n)$ for a tree of size n.

2 (1 pt) Union-Find

Prove that union-by-height (union-by-rank) can also guarantee that the longest length in the data structure is $O(\log n)$. Union-by-height means that, when merging two linked-trees in the linked forest, we always attach the shallower one to the deeper one.

Hint: in class we've already talked bout union-by-size, you can come up with a similar proof about union-by-height.

3 (1 pt) Spanning Trees

1. (0.2pts) What is a spanning tree of a graph?
2. (0.1pts) Given a graph $G = (V, E)$ with positive edge weights, how many edges are there in its spanning tree? Why?
3. (0.7pts) Given a graph $G = (V, E)$ with positive edge weights, show an algorithm that computes the maximum spanning tree: find the spanning tree with the highest weight sum. Your algorithm must run in $O(n^2)$ time. Please explain your algorithm, explain why it is correct and prove the running time bound. You can get partial points if your algorithm is correct and runs in polynomial time.

4 Basic Programming Problems (4pts)

Both problems are on codeforces. Each problem is 2 points. You should submit your report in a separate entry on GradeScope.

5 Bonus Problems (4pts)

The three programming problems are all on codeforces.