CS 577: Introduction to Algorithms — Homework 4

Out: 03/022/2022 — **Due: 04/05/2022**

Ground Rules

- Read the lecture notes.
- Read Chapters on Greedy Algorithms, Dynamic Programming, Amortized Analysis.
- You should do ***all*** problems, but hand in only the graded problems.
- Write your name and student ID clearly on your submission.
- Clearly mark the beginning and end of your solution to each problem.

1.
   - In the lecture for Union-Find we discussed an accounting method that gives some credit in a global account that paid for the per step cost when a node on a path has parent with rank that belongs to a higher bracket. Here is a problem that can be solved in a similar manner (although arguably simpler). You have a digital counter (you can assume it has an unbounded number of digits), where each digit is a symbol in $D = \{0, 1, \ldots, 9\}$. The counter holds a digital number in the usual sense. A stream of digits come in in sequence $a_1, a_2, a_3, \ldots a_n \in D$. Your digital counter, starts with 0 at time 0, and at time step $t \geq 1$ will record the prefix sum $\sum_{i=1}^{t} a_i$ in the counter. The cost per each addition from $\sum_{i=1}^{t-1} a_i$ to $\sum_{i=1}^{t-1} a_i + a_t = \sum_{i=1}^{t} a_i$ is one plus the number of "carry" steps the addition causes.
   - Show that the total cost of these n addition operations is $O(n)$.
   - What if $a_1, a_2, a_3, \ldots a_n \in \{-1, 1\}$? Show that it is possible that your digital counter is "trashing", that is, for a suitable sequence of n inputs from $\{-1, 1\}$, the digital counter costs more than $O(n)$ asymptotically. What is an upper bound of the total cost in this case?
   - Now suppose you can afford two digital counters. Design an algorithm that uses these two digital counters jointly to handle the sequence of n operations adding or subtracting digits, i.e., we receive $\pm a_1, \pm a_2, \pm a_3, \ldots \pm a_n$ where each $a_i \in D$, and we are suppose to compute their sum including the signs. Your algorithm should have total cost $O(n)$.
   - In an earlier homework (hw0) we have seen that every integer (positive, negative, or zero) can be written uniquely in the form $\sum_{i=0}^{\infty} b_i 3^i$, where each $b_i \in \{0, 1, -1\}$ and, $b_i \neq 0$ for only finitely many $i \geq 0$. Use this knowledge to build a "ternary" counter. Write a pseudocode for the operations of PLUS 1 and MINUS 1 in this "ternary" counter system.

2. In the lecture for the analysis of the $O(n \log^* n)$ upper bound for the Union-Find algorithm (Union-by-rank-and-Find-with-path-compression) we specifically separated out what we called the Zero type cost. Specifically, if a Find operation traverses up a path of length k from a vertex v to the root, there are $k + 1$ unit costs to be associated to each node along the path. We charged separately for the root, the child of the root. Explain why we separate out this case. Why can't we have included this in the Second type charge (even if the child of the root and the root belong to the same bucket)? What property of another vertex along the path that has rank in the same bucket as its parent that distinguishes the child of the root?

   Can a child of a root become a non-child of a root later?

3. **Graded Problem (Page limit: 1 sheet; 2 sides)**

   Given a set of n closed intervals $[a_1, b_1], \ldots, [a_n, b_n]$, (where for each $1 \leq i \leq n$, we have $a_i \leq b_i$), you need to output a minimum-size set of points $S = \{p_1, \ldots, p_k\}$ such that S intersects every interval. That is, for any of the n intervals $[a_i, b_i]$, there is at least one point $p_j \in S$ such that $a_i \leq p_j \leq b_i$.

   (a) Give an efficient algorithm for this problem.

(b) What is the running time of your algorithm?

(c) Prove the correctness of your algorithm, as well as the running time of your algorithm.

4. **Graded Problem (Page limit: 1 sheet; 2 sides)** A bandit has broken into a store and has found many items he wants to bring with him and then flee the scene. There are n items, of positive integer sizes $s_1, s_2, \ldots, s_n$. Each item has a positive value to him, $v_1, v_2, \ldots, v_n$. Being greedy the bandit would have liked to bring as much as he can, but his getaway car has only a limited capacity that can hold a total size at most S. Therefore his problem is to select a subset $A \subseteq \{1, 2, \ldots, n\}$ such that the sum $\sum_{i \in A} s_i \leq S$ and the total value (his total loot)

$$\sum_{i \in A} v_i$$

is maximized.

Find a Dynamic Programming algorithm to solve this problem. Your solution should have worst case running time polynomial in (the quantities, not the binary integer sizes of) $s_1, s_2, \ldots, s_n$, $v_1, v_2, \ldots, v_n$, and S.

Analyze its running time. Prove your algorithm is correct, and runs in your stated time.

5. A **supersequence** of a sequence is obtained by adding elements to the given sequence (in any locations). For example, *dynamicprogramming* is a supersequence of the sequence *damn*. A **palindrome** is a sequence which reads the same backwards and forwards. For example, *amanaplanacanalpanama* is a palindrome.

In this problem you are given a sequence A of length n and your goal is to find the shortest supersequence of A that is a palindrome.

First try to analyze the problem by asking what kind of palindrome can be the shortest supersequence of A if the first and the last symbols of A are the same. And what if they are different?

Develop a dynamic programming algorithm for this problem. Analyze its running time. (Your DP should run in $O(n^2)$ time.)