

CS 561: Homework 8

Prof. Jared Saia, University of New Mexico

Remember: You are encouraged to work on the homework in groups, but please observe the “Star Trek” rule from the syllabus.

Problems 1 and 2 revisit Problems 4 and 5 from the midterm.

1. **Weighted Activity Selection.** You are given n activities. Activity a_i has start time s_i , finish time f_i , and positive weight w_i , where $s_i < f_i$. Two activities are compatible if one finishes at or before the other starts. Your goal is to compute the maximum total weight of a set of pairwise compatible activities.

For example, for the activities $(1, 3, 5)$, $(1, 2, 3)$, and $(2, 3, 3)$, written as (start, finish, weight), an optimal solution chooses the final two activities and has total weight 6.

- (a) Give a counterexample showing that the greedy algorithm that always selects the remaining activity with earliest finish time need not be optimal for the weighted problem.
- (b) Sort the activities by nondecreasing finish time as $a_1, a_2, \dots, a_n$. For $1 \leq i \leq n$, let

$$\text{prior}(i) = \max(\{j < i : f_j \leq s_i\} \cup \{0\}).$$

Let $W(i)$ be the maximum total weight obtainable using only activities $a_1, \dots, a_i$. Give a recurrence for $W(i)$, including the base case.

- (c) Describe a dynamic program based on your recurrence. State the table dimensions, the order in which entries are filled, the value returned, and the total running time, including sorting and computing all $\text{prior}(i)$ values.
2. **Target or Bust.** Let n and B be positive integers with $1 \leq B \leq 3n$. You play a game lasting n rounds and start with score 0. In round i , you choose one of two actions:
- **SAFE:** gain exactly 1 point;
 - **GAMBLE:** gain 3 points with probability p_i , and gain 0 points otherwise.

You know all $p_i \in [0, 1]$ in advance, and the gamble outcomes in different rounds are independent. You observe each outcome before choosing your action in the next round, so your choices may depend on your current score. Your goal is to maximize the probability that your final score is at least B .

- (a) Consider the greedy strategy that, in each round, chooses the action with larger expected point gain. Give a counterexample showing that this strategy need not maximize the probability of reaching the target. Your counterexample may use only one round.
- (b) Define, in words, a function for a recurrence that gives the maximum probability of reaching the target. Give the recurrence, including all necessary base cases.
- (c) Describe a dynamic program based on your recurrence. State the table dimensions, the order in which entries are filled, the value returned, and the running time and space usage.

3. **Binary Counter with Reset.** A nonnegative binary counter is stored in an unbounded bit array $B[0], B[1], \dots$, with the least significant bit first. The counter also stores m , the largest index whose bit is 1; when the counter is zero, $m = 0$. Initially all bits are 0 and $m = 0$.

INCREMENT(B, m)

```

 $i \leftarrow 0$ 
while  $B[i] = 1$  do
     $B[i] \leftarrow 0$ 
     $i \leftarrow i + 1$ 
 $B[i] \leftarrow 1$ 
if  $i > m$  then  $m \leftarrow i$ 

```

RESET(B, m)

```

for  $i \leftarrow 0$  to  $m$  do
     $B[i] \leftarrow 0$ 
 $m \leftarrow 0$ 

```

Assume that accessing or writing one bit costs one unit and that all other work shown above costs $O(1)$. Consider an arbitrary sequence of $N \geq 1$ calls to INCREMENT and RESET.

- Give tight asymptotic worst-case bounds, as functions of N , for one call to INCREMENT and for one call to RESET. Briefly justify both bounds.
 - Use the accounting method to prove that the total cost of any such sequence is $O(N)$. Your proof should specify the charge for each operation, where saved credit is stored, and why a RESET always has enough credit to pay for every bit it clears.
4. **A Two-Ended Extendable Array.** Design a data structure that stores a sequence and supports the following operations:

- **ADDLEFT**(x) inserts x at the beginning;
- **ADDRIGHT**(x) inserts x at the end;
- **LOOKUP**(j) returns the j th item, using one-based indexing, or NULL if fewer than j items are stored.

Describe your representation and algorithms. Prove that LOOKUP takes $O(1)$ worst-case time, that both insertion operations take $O(1)$ amortized time, and that the data structure uses $O(n + 1)$ space when it currently stores n items.