Problem Set #4

Quiz, 5 questions

Congratulations! You passed!

Next Item

1 / 1 point

1.
Consider a variation of the Knapsack problem where we have two knapsacks, with integer capacities W_1 and W_2. As usual, we are given n items with positive values and positive integer weights. We want to pick subsets S_1, S_2 with maximum total value (i.e., $\sum_{i \in S_1} v_i + \sum_{i \in S_2} v_i$) such that the total weights of S_1 and S_2 are at most W_1 and W_2, respectively. Assume that every item fits in either knapsack (i.e., $w_i \leq \min\{W_1, W_2\}$ for every item i). Consider the following two algorithmic approaches. (1) Use the algorithm from lecture to pick a a max-value feasible solution S_1 for the first knapsack, and then run it again on the remaining items to pick a max-value feasible solution S_2 for the second knapsack. (2) Use the algorithm from lecture to pick a max-value feasible solution for a knapsack with capacity $W_1 + W_2$, and then split the chosen items into two sets S_1, S_2 that have size at most W_1 and W_2, respectively. Which of the following statements is true?

- Neither algorithm is guaranteed to produce an optimal feasible solution to the original problem.

Correct
Indeed. Can you devise from scratch a dynamic programming algorithm that correctly solves the problem?

- Algorithm (1) is guaranteed to produce an optimal feasible solution to the original problem but algorithm (2) is not.
- Algorithm (2) is guaranteed to produce an optimal feasible solution to the original problem but algorithm (1) is not.
- Algorithm (1) is guaranteed to produce an optimal feasible solution to the original problem provided $W_1 = W_2$.

---

1 / 1 point

2.
Recall the dynamic programming algorithms from lecture for the Knapsack and sequence alignment problems. Both fill in a two-dimensional table using a double-for loop. Suppose we reverse the order of the two for loops. (I.e., cut and paste the second for loop in front of the first for loop, without otherwise changing the text in any way.) Are the resulting algorithms still well defined and correct?

- Both algorithms remain well defined and correct after reversing the order of the for loops.

Correct
The necessary subproblem solutions are still available for constant-time lookup.

- The Knapsack algorithm remains well defined and correct after reversing the order of the for loops, but the sequence alignment algorithm does not.
- The sequence alignment algorithm remains well defined and correct after reversing the order of the for loops, but the Knapsack algorithm does not.

Neither algorithm remains well defined and correct after reversing the order of the for loops.

Problem Set #4

Quiz, 5 questions

1 / 1 point

3.
Consider an instance of the optimal binary search tree problem with 7 keys (say 1,2,3,4,5,6,7 in sorted order) and frequencies $w_1 = .05, w_2 = .4, w_3 = .08, w_4 = .04, w_5 = .1, w_6 = .1, w_7 = .23$. What is the minimum-possible average search time of a binary search tree with these keys?

- 2.18

Correct

- 2.42
- 2.9
- 2.08

1 / 1 point

4.
The following problems all take as input two strings X and Y, of length m and n, over some alphabet Σ. Which of them can be solved in $O(mn)$ time? [Check all that apply.]

- [x] Compute the length of a longest common subsequence of X and Y. (Recall a subsequence need not be consecutive. For example, the longest common subsequence of "abcdef" and "afebcd" is "abcd".)

Correct
Similar dynamic programming to sequence alignment, with one subproblem for each X_i and Y_j. Alternatively, this reduces to sequence alignment by setting the gap penalty to 1 and making the penalty of matching two different characters to be very large.

- [x] Compute the length of a longest common substring of X and Y. (A substring is a consecutive subsequence of a string. So "bcd" is a substring of "abcdef", whereas "bdf" is not.)

Correct
Similar dynamic programming to sequence alignment, with one subproblem for each X_i and Y_j.

- [x] Consider the following variation of sequence alignment. Instead of a single gap penalty α_{gap}, you're given two numbers a and b. The penalty of inserting k gaps in a row is now defined as $ak + b$, rather than $k\alpha_{gap}$. Other penalties (for matching two non-gaps) are defined as before. The goal is to compute the minimum-possible penalty of an alignment under this new cost model.

Correct
Variation on the original sequence alignment dynamic program. With each subproblem, you need to keep track of what gaps you insert, since the costs you incur in the current position depend on whether or not the previous subproblems inserted gaps. Blows up the number of subproblems and running time by a constant factor.

- [ ]

Assume that X and Y have the same length n. Does there exist a permutation f, mapping each $i \in \{1, 2, \ldots, n\}$ to a distinct $f(i) \in \{1, 2, \ldots, n\}$, such that $X_i = Y_{f(i)}$ for every $i = 1, 2, \ldots, n$?

Problem Set #4

Quiz, 5 questions

Correct

This problem can be solved in $O(n)$ time, without dynamic programming. Just count the frequency of each symbol in each string. The permutation f exists if and only if every symbol occurs exactly the same number of times in each string.

---

1 / 1 point

5.

Recall our dynamic programming algorithms for maximum-weight independent set, sequence alignment, and optimal binary search trees. The space requirements of these algorithms are proportional to the number of subproblems that get solved: $\Theta(n)$ (where n is the number of vertices), $\Theta(mn)$ (where m, n are the lengths of the two strings), and $\Theta(n^2)$ (where n is the number of keys), respectively.

Suppose we only want to compute the value of an optimal solution (the final answer of the first, forward pass) and don't care about actually reconstructing an optimal solution (i.e., we skip the second, reverse pass over the table). How much space do you then really need to run each of three algorithms?

- $\Theta(1)$, $\Theta(1)$, and $\Theta(n)$
- $\Theta(1)$, $\Theta(n)$, and $\Theta(n^2)$

Correct

For maximum-weight independent set, you only need to remember the two most recent subproblems. For sequence alignment, you only need to remember the subproblems for the current and previous values of i (and for all values of j). For optimal binary search trees, previous subproblems remain relevant throughout the entire computation.

- $\Theta(n)$, $\Theta(mn)$, and $\Theta(n^2)$
- $\Theta(1)$, $\Theta(n)$, and $\Theta(n)$
