

Homework Assignment #3

Due: November 9, 2010, by 10:30 am

(in the drop box for your CSCC73 tutorial section, near room SW-626A)

Appended to this document is a cover page for your assignment. Fill it out, staple your answers to it, and deposit the resulting document into the course drop box (without putting it in an envelope).

Recall that a dynamic programming algorithm to solve a given problem P involves the following elements:

- (a) A definition of a polynomial number of subproblems that will be solved (and from whose solution we will compute the solution to P — see (d) below).
- (b) An ordering of these subproblems.
- (c) A recursive formula to compute the solution to each subproblem from the solutions to smaller subproblems (“smaller” in the sense of the ordering in (b)).
- (d) A way to compute the solution to P from the solutions to the subproblems computed in (c).

Explaining the correctness of a dynamic programming algorithm amounts to justifying (i) why the recursive formula in step (c) correctly computes the subproblems defined in step (a), and (ii) why the computation in (d) yields a solution to the given problem. Part (ii) is often immediate from the definitions.

Question 1. (10 marks) (Exercise 6.1 in DPV, slightly reworded.) A **contiguous subsequence** of a list S is a subsequence consisting of consecutive elements of S . For instance, if S is

5, 15, -30, 10, -5, 40, 10

then 15, -30, 10 is a contiguous subsequence of S , but 5, 15, 40 is not. Consider the following problem:

Input: A list of numbers $S = a_1, a_2, \dots, a_n$

Output: The maximum number m such that m is the sum of all elements of a contiguous subsequence of S . (We define the sum of an empty subsequence to be 0.)

It is trivial to design an $O(n^2)$ algorithm to solve this problem, since there are only $O(n^2)$ contiguous subsequences of a sequence of length n . Use dynamic programming to give an $O(n)$ algorithm to solve this problem. Explain why your algorithm is correct.

Hint: For each integer i , $1 \leq i \leq n$, consider the maximum number that is the sum of all elements of a contiguous subsequence of S that ends with a_i .

Question 2. (10 marks) (Exercise 6.7 in DPV, reworded.) A **palindrome** is a string that reads the same backwards as forwards.

a. (9 marks) Give an $O(n^2)$ algorithm which, given a string S , returns the maximum number m such that m is the length of a substring of S that is a palindrome. Explain why your algorithm is correct.

Hint: For all integers i, j , $1 \leq i \leq j \leq n$, consider the quantity $P[i, j]$ defined as the maximum length of a substring of $S[i..j]$ that is a palindrome.

b. (1 marks) Modify your algorithm in part (a) so that it returns a maximum-length substring of S that is a palindrome.

Question 3. (10 marks) We are given a picture of some tissue in the form of an $n \times n$ array A of bits representing pixels of the picture: A “1” in this array denotes the presence of blood in the pixel, and a “0” denotes the absence of blood in that pixel. A blood clot is a contiguous lump of blood, and (for some reason) we are only interested in square blood clots. Our goal is to find a largest blood clot in the picture.

More specifically, the input is an $n \times n$ array $A[1..n, 1..n]$ of bits, not all of which are 0. A *clot* is defined to be a 4-tuple (i_1, i_2, j_1, j_2) such that $1 \leq i_1 \leq i_2 \leq n$, $1 \leq j_1 \leq j_2 \leq n$, $i_2 - i_1 = j_2 - j_1$, and $A[i, j] = 1$ for all i, j such that $i_1 \leq i \leq i_2$ and $j_1 \leq j \leq j_2$. The goal is to find a clot so that $i_2 - i_1$ is as large as possible.

It is easy to get an $O(n^5)$ time algorithm for this problem. You should do better. Use dynamic programming, explain why your algorithm is correct, and state its running time. For full credit, you must give an $O(n^2)$ algorithm.

Hint: Consider, for each i, j , the size of a largest clot whose lower right corner is $[i, j]$.

Question 4. (10 marks) Consider the variation on the Subset Sum Problem described in KT 6.4, where we are given an additional input Δ , and we are only interested in sets S for which every two elements are at least Δ apart. As before, the elements of S must sum to at most W , and we want the sum of the elements of S to be as large as possible.

More precisely, the input is a sequence of positive integers $w_1, w_2, \dots, w_n, W, \Delta$. We wish to find $S \subseteq \{1, 2, \dots, n\}$ such that $\sum_{j \in S} w_j$ is as large as possible, subject to the following constraints:

- $\sum_{j \in S} w_j \leq W$ and
- for all $i, j \in S$, if $i \neq j$ then $|w_i - w_j| \geq \Delta$.

Give a dynamic programming algorithm for this problem. Explain why your algorithm is correct, and analyse its time complexity. Your algorithm should run in time polynomial in n and W .

Question 5. (10 marks) Consider the following *Two-Knapsack* variation of the Knapsack Problem described in KT 6.4 (page 271).

The input is a sequence of positive integers $w_1, v_1, w_2, v_2, \dots, w_n, v_n, W_1, W_2$. (Integers w_i and v_i are the weight and value of item i ; W_1 and W_2 are the capacities of the two knapsacks.) A *knapsack pair* is defined to be a pair (S_1, S_2) such that $S_1, S_2 \subseteq \{1, 2, \dots, n\}$, $S_1 \cap S_2 = \emptyset$, $\sum_{i \in S_1} w_i \leq W_1$, and $\sum_{j \in S_2} w_j \leq W_2$. The goal is to produce a knapsack pair so that $\sum_{i \in S_1} v_i + \sum_{j \in S_2} v_j$ is as large as possible. (Intuitively, S_1 and S_2 are the sets of items that go into knapsacks 1 and 2, respectively; the total weight of the items in each knapsack must not exceed the knapsack’s capacity; and we wish to maximize the total value of the items in the two knapsacks.)

Give a dynamic programming algorithm for this problem. Explain why your algorithm is correct, and analyse its time complexity. Your algorithm should run in time polynomial in n , W_1 , and W_2 .

Cover page for CSCC73 Assignment #3

Submitted by

(1) Family Name: _____

(2) Family Name: _____

Given Name: _____

Given Name: _____

Student Number: _____

Student Number: _____

By virtue of submitting this homework I/we acknowledge that I am/we are aware of the policy on homework collaboration for this course.