COMP 3711H Design and Analysis of Algorithms
2015 Fall Semester
Homework 4
Handed out: Nov 13
Due: Nov 25

Assignments should be submitted to the collection box located outside room 4210A in the lab area.

Problem 1. Given an array $A[1, \ldots, n]$ of distinct integers, consider the problem of finding the longest monontonically increasing subsequence of *not necessarily contiguous* entries in your array. For example, if the input array is $10, 3, 9, 5, 8, 13, 11, 14$, then the longest monotonically increasing subsequence is $3, 5, 8, 11, 14$. The goal of this problem is to find an efficient dynamic programming algorithm for solving this problem.

(a) Let $L[i]$ be the length of the longest monotonically increasing sequence that ends at and uses element $A[i]$. Write down a recurrence for $L[i]$.

(b) Based on your recurrence in part (a), design an $O(n^2)$ time dynamic programming algorithm for finding the length of the longest monotonically increasing subsequence. Justify the running time of your algorithm.

(c) Modify your algorithm to actually find the longest increasing subsequence (not just its length).

Problem 2. Suppose you are given three strings of characters: $X = x_1 x_2 \cdots x_n$, $Y = y_1 y_2 \cdots y_m$, $Z = z_1 z_2 \cdots z_p$, where $p = n + m$. Z is said to be a *shuffle* of X and Y iff Z can be formed by interleaving the characters from X and Y in a way that maintains the left-to-right ordering of the characters from each string. The goal of this problem is to design an efficient dynamic-programming algorithm that determines whether Z is a shuffle of X and Y.

(a) Establish that *cchocohilaptes* is a shuffle of *chocolate* and *chips*, but that *chocochilatspe* is not.

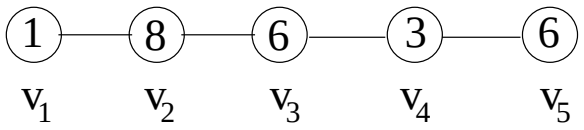
(b) For any string $A = a_1 a_2 \cdots a_r$, let $A_i = a_1 a_2 \cdots a_i$ be the substring of A consisting of the first i characters of A. For example, if A is *chocolate*, then A_3 is *cho* and A_6 is *chocol*.

Define $f(i, j)$ to be 1 if Z_{i+j} is a shuffle of X_i and Y_j, and 0 otherwise. Derive a recursive formula for $f(i, j)$. Remember to include the basis cases. Briefly explain your derivation.

(c) Design an efficient algorithm for determining whether Z is a shuffle of X and Y. Analyze the running time of your algorithm.

Problem 3. Let $G = (V, E)$ be an undirected graph with n nodes. A subset of the set of nodes V is said to be an *independent set* if no two of them are joined by an edge. Finding large independent sets is difficult in general; but here we'll see that it can be done efficiently if the graph is "simple" enough.

Call a graph $G = (V, E)$ a *path* if its nodes can be written as $v_1, v_2, \ldots, v_n$, with an edge between v_i and v_j if and only if the numbers i and j differ by exactly 1. With each node v_i, we associate a positive integer *weight* w_i. (Consider, for example, the five-node path drawn in the figure. The *weights* are the numbers drawn inside the nodes.)



The goal in this question is to solve the following problem: Find an independent set in a path G whose total weight is as large as possible. (For example, for the path shown in the figure, the maximum weight of an independent set is 14 and it is obtained by choosing the vertices v_2 and v_5.)

(a) Give an example to show that the following algorithm *does not* always find an independent set of maximum total weight.

```
The "heaviest-first" greedy algorithm
  Start with S equal to the empty set
  While some node remains in G
    Pick a node v_i of maximum weight
    Add v_i to S
    Delete v_i and its neighbors from G
  Endwhile
  Return S
```

(b) Give an example to show that the following algorithm also *does not* always find an independent set of maximum total weight.

```
Let S_1 be the set of all v_i where i is an odd number
Let S_2 be the set of all v_i where i is an even number
(Note that S_1 and S_2 are both independent sets)
Determine which of S_1 or S_2 has greater total weight,
  and return this one
```

(c) Give an algorithm based on dynamic programming that takes an n-node path G with weights and returns an independent set of maximum total weight. The running time should be polynomial in n, independent of the values of the weights. Justify the correctness and running time of your algorithm.

Problem 4. In this question, you are required to solve the 0-1 Knapsack problem for *two* knapsacks. You are given a set of n objects. For $1 \leq i \leq n$, the i-th object has value v_i and weight w_i. You are given two knapsacks each of weight capacity C. If an object is taken, it may be placed in one knapsack or the other, but not both. All weights and values are positive integers. Design an $O(nC^2)$ dynamic programming algorithm that determines the maximum value of objects that can be placed into the two knapsacks. Justify the correctness and running time of your algorithm.