CSCE 411-200, Fall 2016
Homework 3 Solutions

Problem 1: Transform the following word problem into a linear program. Farmer Fred has 10 acres of land, on which he can grow cotton and soybean. For tax purposes, he needs to farm at least 7 acres. He has \$1200 to spend on seeds. Cotton seeds cost \$200 per acre while soybean seeds cost \$100 per acre. Farmer Fred has to plant all the seeds in a 12-hour period. It takes one hour per acre to plant cotton seeds and two hours per acre to plant soybean seeds. The profit per acre of cotton is \$500, while the profit per acre of soybean is \$300. How many acres of cotton and how many acres of soybean should he plant in order to maximize his profit?

Solution: Variables are x_c = number of acres of cotton and x_s = number of acres of soybean. Objective function is to maximize $500 \cdot x_c + 300 \cdot x_s$ subject to the following constraints:

- $x_c \geq 0$
- $x_s \geq 0$
- $200 \cdot x_c + 100 \cdot x_s \leq 1200$
- $1 \cdot x_c + 2 \cdot x_s \leq 12$
- $x_c + x_s \leq 10$
- $x_c + x_s \geq 7$

Problem 2: Design a dynamic programming solution for the following problem. You have a board consisting of n rows and m columns of cells. Some, but not necessarily all, of the cells have pebbles in them (at most one pebble per cell). A robot starts at the top left cell and is supposed to move to the bottom right cell. At each step, the robot can either move down by one cell or to the right by one cell. When the robot is in a cell with a pebble, it collects the pebble. What is the maximum number of pebbles the robot can collect? (The locations of the pebbles are known.)

Hint: For arbitrary i and j, let $F[i, j]$ be the maximum number of pebbles the robot can collect when traveling between its starting location and cell (i, j). The robot can get to cell (i, j) in one of two ways: either from the cell above or from the cell to the left. Use boolean variable pij to indicate whether or not there is a pebble in cell (i, j).

Solution:

(a) Which element(s) of F hold, or are needed to compute, the final answer?

$$F[n, m]$$

(b) Which element(s) of F hold the basis element(s) and how should they be filled in?

$$F[1, 1] = p_{11}$$

(c) What is the formula for filling in each of the other (non-basis) elements of F?

$$F[i, j] = \begin{cases} p_{ij} + F[i-1, j] & \text{if } j = 1, \\ p_{ij} + F[i, j-1] & \text{if } i = 1, \\ p_{ij} + \max\{F[i-1, j], F[i, j-1]\} & \text{if } j \geq 2 \text{ and } i \geq 2 \end{cases}$$

(Or row 1 and column 1 could be included as part of the basis.)

(d) In what order should the elements of F be filled in?

Either row-major or column-major order.

(e) What is the running time of the algorithm and why?

$\Theta(n \cdot m)$ since there are $n \cdot m$ table entries and each one takes constant time to compute.

(f) How can the actual optimal path be determined? (Just give a brief high-level description.)

Keep track of which choice is made when computing the maximum of coming into the current cell either from above or from the left.

Problem 3: Problem 15-1 (p. 404). Describe a dynamic-programming approach to find a longest weighted simple path from vertex s to vertex t in a DAG. What does the subproblem graph look like? What is the efficiency of the algorithm?

Solution: Similar to Bellman-Ford. Suppose $G = (V, E, w)$ is a weighted DAG. Let $D[i, k]$ be the maximum weight of all paths from s to vertex i that use at most k edges.

The goal is $D[i, |V| - 1]$ (since G has no cycles, every path from s to any other node has at most $|V| - 1$ edges in it).

The basis is $D[i, 1] = w(s, i)$ for all vertices i (if there is no edge from s to i, then this is ∞).

Recursion: $D[i, k]$ is the maximum, over all incoming neighbors j of i, of $D[j, k-1] + w(j, i)$; however, if this maximum is smaller than $D[i, k-1]$, then set $D[i, k]$ to $D[i, k-1]$.

This can be optimized as with Bellman-Ford for a DAG: we don't need to keep entries in D for past values of k (i.e., we can fill in D in place) and we consider the vertices in topological order:

```
topologically sort G
d[v] := infinity for all v in V
d[s] = 0
for each u in V in topological sort order to
   for each neighbor v of u do
      d[v] := max{d[v], d[u]+w(u,v)}
return d[t]
```

Running time is $O(V + E)$: $O(V + E)$ time for the topological sort, $O(V)$ time for the initializations, $O(V)$ iterations of the outer for loop, and $O(E)$ iterations *in total* of the inner for loop, with inner-most body taking constant time.

To compute the actual path corresponding to the final value of $d[t]$, keep track of which choice is chosen for the maximum.

Problem 4: Design a dynamic programming algorithm to solve the 0-1 knapsack problem. As a reminder, in this problem there are n items, the i-th item has value v_i and weight w_i, where v_i and w_i are positive integers ($i = 1, ..., n$). A thief wants to maximize the value of what he steals, but he cannot carry more than W pounds in his knapsack, where W is an integer. He must take all of an item or none of it. What is the maximum value that the thief can steal?

Hint: Consider any fixed order of the items. For arbitrary i and p, let $C[i, p]$ be the maximum total value of any subset of the first i items whose total weight is at most p. To compute $C[i, p]$, the thief has two choices when considering item i: either he takes or he leaves it behind.

- If he takes the i-th item, then he gets the value of the item but the item also takes up some of the capacity of the knapsack for the previous items to use.
- If he doesn't take the i-th item, then he doesn't get the value of the item, so all the capacity of the knapsack is available for previous items.

(a) Which element(s) of C hold, or are needed to compute, the final answer?

$$C[n, W]$$

(b) Which element(s) of C hold the basis element(s) and how should they be filled in?

To make our life easier, let's start enumerating the rows with 0 (empty subset of items) and start enumerating the columns with 0 (no capacity).

Row 1: For $p = 0, 1, \ldots, W$, $C[0, p] = 0$ (i.e., if we are not allowed to take any items, then the total value we can steal is zero).

Column 1: For $i = 0, 1, \ldots, n$, $C[i, 0] = 0$ (i.e., if we don't have the capacity to take anything, then the total value we can steal is zero).

(c) What is the formula for filling in each of the other (non-basis) elements of C?

$$C[i, p] = \max\{v_i + C[i-1, p-w_i], C[i-1, p]\}$$

Use the convention that if $p - w_i < 0$, then replace $C[i-1, p-w_i]$ with $-\infty$.

(d) In what order should the elements of C be filled in?

Row-major order.

(e) What is the running time of the algorithm and why?

$O(nW)$ since there are $(n+1)\cdot(W+1)$ entries in the table and each one takes constant time to compute.

(f) How can the actual set of items to steal be determined? (Just give a brief high-level description.)

Keep track of which choice is made when doing the "max" computation for each table entry.