Carsten Hood UIN: 922009787
CSCE 411-200 Fall 2016
Homework 3 due Wed, Oct 5, 11:30 AM

———————————————

1.

let *c* be the acres of cotton;
let *s* be the acres of soybeans;
linear program:
maximize: 500*c + 300*s // maximize profit per acre
constraints:
c + s <= 10 // can only use 10 acres
c + s >= 7 // needs to farm at least 7 acres
200*c + 100*s <= 1200 // cost of seeds/acre must fit budget
1*c + 2*s <= 12 // 12-hour planting window

———————————————

2.

(a) Which element(s) of F hold, or are needed to compute, the final answer?
You need F[i – 1, j] and F[i, j – 1], i.e., the maximum numbers of pebbles collectable at each of the cells immediately above and left of cell (i, j) if these are indeed valid cells)π

(b) Which element(s) of F hold the basis element(s) and how should they be filled in?
F[0, 0] = int(p_0_0)
The basis element is F[0, 0]; it should be set to 1 if the starting (0, 0) cell has a pebble, and 0 if it does not.

(c) What is the formula for filling in each of the other (non-basis) elements of F?
// avoid checking out-of-bounds (non-existent) cells
if i – 1 < 0 then F[i, j – 1]
else if j – 1 < 0 then F[i – 1, j]
else **F[i, j] = max(F[i –1, j], F[i – 1, j])**

(d) In what order should the elements of F be filled in?
One row at a time, moving left to right within each row. Alternatively, one column at a time, moving downwards within each column.

(e) What is the running time of the algorithm and why?
O(n*m). Each of the n*m cells in the matrix is processed exactly once. Previous subproblems' results are dynamically stored and hence are accessible in constant O(1) time.

(f) How can the actual optimal path be determined? (Just give a brief high-level description.)
First, assign maximum collectable pebble values to every cell according to the above-described method. Then perform a basic greedy algorithm starting from the destination (last) cell. At each step, move either upward or leftward, towards whichever cell is assigned a higher max pebble value (or either way if they are tied). Adding each traversed cell to the front of a sequence in turn results in the optimum path from start to finish.

———————————————

3. Problem 15-1 (p. 404) - Longest simple path in a directed acyclic graph

algorithm summary:
Associate each vertex with a distance value, initially zero for the source vertex *s* and negative infinity for the rest. Process each vertex *v* in order according to a topological sort. In each iteration considering a vertex *v*, calculate the distance to all of *v*'s subsequent vertices *u*. If this distance is greater than *u*'s current attributed distance value, update *u*'s distance. Finally, return the distance of the desired destination node *t*.

algorithm pseudocode:

```
// let d[x] denote a distance value associated with vertex x;
// let w(a, b) denote the weight of the edge between vertices a & b;
input: dag G = (V, E), starting vertex s, destination vertex t;
initialize d[s] to 0;
initialize d[v] to negative infinity for all v in V such that v != s;
let Vtop = topological-sort(G);
for every vertex v in Vtop:
   for every vertex u in V such that u is adjacent to v:
      let dist = d[u] + w(u, v);
      if d[v] < dist then:
         d[v] = dist;
return d[t]
```

algorithm complexity:
The initial topological sort takes O(V + E). The algorithm then processes all V vertices one time each, as well as each of E edges in the course of performing the distance calculations for adjacent vertices. Hence we are left with the complexity O(V + E).

———————————————

4.

(a) Which element(s) of C hold, or are needed to compute, the final answer?
C[i – 1, p] and C[i – 1, p – w_i]

(b) Which element(s) of C hold the basis element(s) and how should they be filled in?
C[0, p] = 0

(c) What is the formula for filling in each of the other (non-basis) elements of C?
if w_i > p then C[i, p] = C[i – 1, p]
else C[i, p] = max(C[i – 1, p], C[i – 1, p - w_i] + v_i)

(d) In what order should the elements of C be filled in?
Move left to right within one row at a time, in increasing order from 0 to W; and then move over the rows from top to bottom, from 0 to n.

(e) What is the running time of the algorithm and why?
O(n*W). For each of n items we perform constant time operations (due to dynamic programming) for each of W weight values, hence we have n*W total iterations.

(f) How can the actual set of items to steal be determined? (Just give a brief high-level description.)
This can be accomplished by tracking the optimally valued subset. In the course of performing the above-described process, keep track of the items constituting each subset as it forms (e.g., by adding them to a sequence). If the current subset's overall valuation is higher than the current maximum value, assign the current value to the maximum, and save the current subset as the optimal subset. At the end of the algorithm we will be left with the overall optimal subset and its valuation.