CSCI 2300 — Introduction to Algorithms
Homework 6 (document version 1.1) — DUE April 21, 2020
Dynamic Programming and Linear Programming

- This homework is due by 11:59PM on the due date above and must be submitted in Submitty as a single PDF called `upload.pdf`.
- Just like the labs, please avoid using Google to find suggestions or solutions. You are allowed to use the textbooks and class notes as references. We strongly encourage you to use the office hours, labs/recitations, and Discussion Forum to successfully complete this assignment.
- Also, you are allowed to consult with your classmates, collaborating in line with the policies set forth in the syllabus. You must write up your own separate solution; do not copy-and-paste anything.
- Remember the goal is to use your own brain to work these problems out, which will help you develop the skills to do well on the exams and, more importantly, become a substantially better computer scientist.
- For your solutions and proofs, please write clearly and concisely, and use rigorous formal arguments and pseudocode that mimics the style used in the Dasgupta textbook. Be consistent.
- In general, if you are asked to provide an algorithm, the best thing to do is to provide a clear description in English (e.g., "Use BFS, but with the following change..." or "Run BFS two times as follows...").
- Be as concise as you can in your answers. Long answers are difficult to grade.

Instructions for all Problems

The Chapter 6 homework problems are dynamic programming problems. Therefore, as with the previous homework, for each problem, be sure to explicitly state the following:

1. What your subproblems are and what they mean, e.g., "$L[i]$ is the length of the shortest path ending at i" is fine, whereas "$L[i]$ is optimum at i" is insufficient.
2. Your recurrence and algorithm, for which a few lines of pseudocode should suffice.
3. One or two sentences explaining why your recurrence is correct (no need for a formal proof).
4. The runtime of your algorithm.

Warm-Up Problems (do not hand in for grading)

Work out the problems below as practice problems before going on to the next section. Note that warm-up problems might sometimes show up on an exam!

1. DPV Problem 6.17 (as a hint, reduce to a Knapsack problem) – **SOLUTIONS:**

   This problem reduces to the Knapsack problem with repetitions. For the reduction, the total capacity is v and, for each coin denomination, there is an item i of value x_i and weight x_i. It is then possible to make change for value v if and only if the maximum value we can fit in v is exactly v. From the Knapsack problem with repetitions, the runtime is $O(nv)$.

2. DPV Problem 7.1 – **SOLUTIONS:**

   The optimal solution is achieved in the upper-right corner of the convex feasible region, i.e., at point $(5, 2)$, and has the value $5x + 3y = 31$.

3. DPV Problem 7.2 – **SOLUTIONS:**

   Start by simplifying the notation used in this problem, i.e., abbreviate the four states/countries to c, k, m, and n. Concatenating two of these together indicates the number of shnupells of duckwheat transported from one city to another (e.g., mn indicates the number of shnupells of duckwheat transported from Mexico to New York).

   The linear program is then:

Objective function	$\min 4mn + mc + 2kn + 3kc$
Constraints	$kn + kc = 15$
	$mn + mc = 8$
	$mn + kn = 10$
	$mc + kc = 13$
	$mn \geq 0, mc \geq 0, kn \geq 0, kc \geq 0$

4. DPV Problem 7.3 – **SOLUTIONS:**

   Let c_i denote the quantity in cubic meters of material i. The linear program is then:

Objective function	$\max 1000c_1 + 1200c_2 + 12000c_3$
Constraints	$2c_1 + c_2 + 3c_3 \leq 100$
	$c_1 + c_2 + c_3 \leq 60$
	$c_1 \leq 40$
	$c_2 \leq 30$
	$c_3 \leq 20$
	$c_1 \geq 0, c_2 \geq 0, c_3 \geq 0$

Required Problems (hand in for grading)

Work out the problems below and submit for grading before the deadline.

1. DPV Problem 6.18

   SOLUTIONS AND GRADING GUIDELINES:

   $\Longrightarrow$ 2 points for clearly described and correct subproblems
   $\Longrightarrow$ 2 points for clearly described and correct algorithm and recurrence
   $\Longrightarrow$ 2 points for showing correctness (no need for formal proof)
   $\Longrightarrow$ 2 points for correct runtime

   We can reduce this problem to the Knapsack problem *without* repetition. The total capacity is v and there is an item i of value x_i and weight x_i for each coin denomination. In this case, it is possible to make change for value v if and only if the maximum value we can fit in v is exactly v. Given the above reduction, the runtime is $O(nv)$.

   Here is a more detailed solution, essentially lining up with the Knapsack problem without repetition (which also applies to Problems 6.19 below).

   Subproblems: Given maximum capacity v, define subproblem $K(w, q)$ for $1 \leq w \leq v$ and $1 \leq q \leq v$ to represent the maximum value achievable (without repetition of items) using a knapsack of capacity w and items $1 \ldots q$ from which to choose the next item.

   Algorithm and Recursion: Initialize $K(1, 1) = 0$. The other subproblems can be solved incrementally using the recurrence relation:

$$K(w, q) = \max \begin{cases} K(w - w_q, q - 1) + v_q & \text{(we choose item } q\text{)} \\ K(w, q - 1) & \text{(we do not choose item } q\text{)} \end{cases}$$

   The recursion is such that for each subproblem, we select either the case in which we choose item q or do not choose item q, respectively.

   Correctness and Runtime: We want to find $K(v, n)$, where v is the maximum capacity and n is the number of items (i.e., number of coin denominations). The total runtime is $O(nv)$ since to find solutions to all subproblems, we have $O(nv)$ subproblems, each requiring $O(1)$ time.

2. DPV Problem 6.19 – your algorithm should run in time at most $O(nvk)$ to receive full credit.

   SOLUTIONS AND GRADING GUIDELINES:

   $\Longrightarrow$ 2 points for clearly described and correct subproblems
   $\Longrightarrow$ 2 points for clearly described and correct algorithm and recurrence
   $\Longrightarrow$ 2 points for showing correctness (no need for formal proof)
   $\Longrightarrow$ 2 points for correct runtime

   We can reduce this problem to the Knapsack problem *without* repetition. The total capacity is v, but unlike the Problems 6.17 and 6.18, we have k items of value x_i and weight x_i for each coin denomination x_i, i.e., a total of $k \times n$ items. It is possible to make change for value v if and only if the maximum value we can fit in v is exactly v.

   Given the above reduction, the runtime is $O(nkv)$.

3. DPV Problem 7.18 (parts (a) and (c) only) – for part (a), show how to use the original max-flow problem to solve this variation; and for part (c), show how to use linear programming to solve this.

SOLUTIONS AND GRADING GUIDELINES:

$\Longrightarrow$ 2 points in part (a) for addressing many sources
$\Longrightarrow$ 2 points in part (a) for addressing many sinks
$\Longrightarrow$ 2 points in part (c) for identifying lower bound l_e
$\Longrightarrow$ 2 points in part (c) for reduction using l_e

(a) Given many sources and many sinks, we can first state that $s_1, s_2, \ldots, s_n$ are the sources and $t_1, t_2, \ldots, t_n$ are the sinks. Next, create new master source node S and new master sink node T. Connect S to each s_i by an edge with infinite capacity. Similarly, connect each t_i to T with an edge with infinite capacity.

Given the above, we have successfully reduced the proposed problem into an instance suitable for the traditional max-flow algorithm.

(c) With each edge having not only a capacity, but also a lower bound on the flow it must carry, we can first state that l_e is the lower bound on the flow for edge e. In the original max-flow linear program, there is the constraint that requires the flow sent along edge e, which is denoted f_e, to be between 0 and c_e, where c_e is the capacity for edge e.

To reduce the proposed problem to the original max-flow linear program, we can change the above constraint to be $l_e \leq f_e \leq c_e$, adding this constraint to each edge e.