School of Computing and Information Systems
COMP90038 Algorithms and Complexity Tutorial Week 11

Sample Answers

The exercises

74. Use the dynamic-programming algorithm developed in Lecture 18 to solve this instance of the coin-row problem: 20, 50, 20, 5, 10, 20, 5.

    Answer: We build the table S of optimal values as follows:

i :	0	1	2	3	4	5	6	7
$C[i]$:	–	20	50	20	5	10	20	5
$S[i]$:	0	20	50	50	55	60	75	75

    The optimal selection uses the coins at indices 2, 4, and 6.

75. In Week 12 we will meet the concept of *problem reduction*. This question prepares you for that. First, when we talk about the length of a path in an un-weighted directed acyclic graph (dag), we mean the number of edges in the path. (You could also consider the un-weighted graph weighted, will all edges having weight 1.)

    Show how to reduce the coin-row problem to the problem of finding a longest path in a dag. That is, give an algorithm that transforms any coin-row instance into a longest-path-in-dag instance in such as way that a solution to the latter provides a solution to the former.

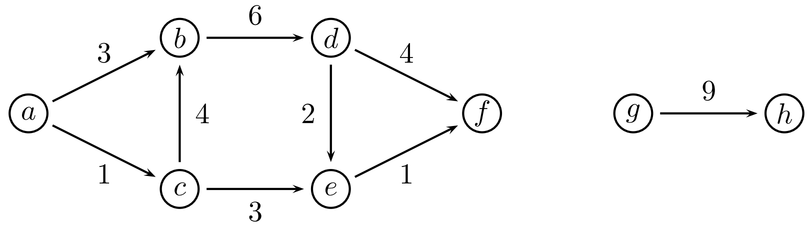
    Hint: If there are n coins, use $n+1$ nodes; let an edge with weight i correspond to picking a coin with value i.

    Answer: Assume we have n coins $c_1, \ldots, c_n$. We generate a weighted dag with $n+1$ nodes $C_0, C_1, \ldots, C_n$. The dag has edges as follows:

    - $n-1$ edges $(C_0, C_n), (C_1, C_n), \ldots, (C_{n-2}, C_n)$, each with weight c_n.
    - $n-2$ edges $(C_0, C_{n-1}), (C_1, C_{n-1}), \ldots, (C_{n-3}, C_{n-1})$, each with weight c_{n-1}.
    - and so on, down to two edges (C_0, C_3) and (C_1, C_3), each with weight c_3.
    - one edge (C_0, C_2) with weight c_2, and
    - one edge (C_0, C_1) with weight c_1.

    Any path in the generated dag corresponds to a legal selection of coins, and the sum of the weights along a given path is exactly the sum of the coins chosen.

76. Consider the problem of finding the length of a "longest" path in a *weighted*, not necessarily connected, dag. We assume that all weights are positive, and that a "longest" path is a path whose edge weights add up to the maximal possible value. For example, for the following graph, the longest path is of length 15:



Use a dynamic programming approach to the problem of finding longest path in a weighted dag.

Answer: This is easy if we process the nodes in topologically sorted order. For each node t we want to find its longest distance from a source, and to store these distances in an array L. That is, for each t we want to calculate

$$max(\{0\} \cup \{L[u] + weight[u,t] \mid (u,t) \in E\})$$

So:

$T \leftarrow$ TopSort($\langle V, E\rangle$) — List of nodes sorted topologically
for each $t \in T$ (in topological order) **do**
  $L[t] \leftarrow 0$
  for each $u \in V$ **do**
    if $(u,t) \in E$ **then**
      if $L[u] + weight[u,t] > L[t]$ **then**
        $L[t] \leftarrow L[u] + weight[u,t]$
$max \leftarrow 0$
for each $u \in V$ **do**
  if $L[u] > max$ **then**
    $max \leftarrow L[u]$
return max

For the sample graph, DFS-based topsort yields the sequence g, h, a, c, b, d, e, f. The "longest path" table L gets filled as follows:

t :	a	b	c	d	e	f	g	h
$L[t]$:							0	
								9
	0							
			1					
		5						
				11				
					13			
						15		

77. Design a dynamic programming algorithm for the version of the knapsack problem in which there are unlimited numbers of copies of each item. That is, we are given items $I_1, \ldots, I_n$ that have values $v_1, \ldots, v_n$ and weights $w_1, \ldots, w_n$ as usual, but each item I_i can be selected several times. Hint: This actually makes the knapsack problem a bit easier, as there is only one parameter (namely the remaining capacity w) in the recurrence relation.

Answer: Assume the items $I_1, \ldots, I_n$ have values $v_1, \ldots, v_n$ and weights $w_1, \ldots, w_n$. Let $V(w)$ denote the optimal value we can achieve given capacity w. With capacity w we are in a position to select any item I_i which weighs no more than w. And if we pick item I_i then the best value we can achieve is $v_i + V(w - w_i)$. As we want to maximise the value for capacity w, we have the recurrence

$$V(w) = max\{v_i + V(w - w_i) \mid 1 \leq i \leq n \wedge w_i \leq w\}$$

That leads to this table-filling approach:

for $w \leftarrow 1$ to W **do**
  $V[w] \leftarrow max(\{0\} \cup \{v_i + V(w - w_i) \mid 1 \leq i \leq n \wedge w_i \leq w\})$
return $V[W]$

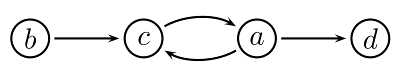
As an example, consider the case $W = 10$, and three items I_1, I_2, and I_3, with weights 4, 5 and 3, respectively, and values 11, 12, and 7, respectively. The table V is filled from left to right, as follows:

w :	1	2	3	4	5	6	7	8	9	10
$V[w]$:	0	0	7	11	12	14	18	22	23	25

Hence the optimal bag is $[I_1, I_3, I_3]$ for a total value of 25.

78. Work through Warshall's algorithm to find the transitive closure of the binary relation given by this table (or directed graph):

$$\begin{array}{c|cccc|} & a & b & c & d \\ \hline a & 0 & 0 & 1 & 1 \\ b & 0 & 0 & 1 & 0 \\ c & 1 & 0 & 0 & 0 \\ d & 0 & 0 & 0 & 0 \\ \hline \end{array}$$



Answer: We run down the columns from left to right, stopping when we meet a 1. This first happens when we are in row 3, column 1. At that point, 'or' row 1 onto row 3 (and so on):

$$\begin{array}{c|cccc|} & a & b & c & d \\ \hline a & 0 & 0 & 1 & 1 \\ b & 0 & 0 & 1 & 0 \\ c & 1 & 0 & 1 & 1 \\ d & 0 & 0 & 0 & 0 \\ \hline \end{array} \Rightarrow \begin{array}{c|cccc|} & a & b & c & d \\ \hline a & 1 & 0 & 1 & 1 \\ b & 0 & 0 & 1 & 0 \\ c & 1 & 0 & 1 & 1 \\ d & 0 & 0 & 0 & 0 \\ \hline \end{array} \Rightarrow \begin{array}{c|cccc|} & a & b & c & d \\ \hline a & 1 & 0 & 1 & 1 \\ b & 1 & 0 & 1 & 1 \\ c & 1 & 0 & 1 & 1 \\ d & 0 & 0 & 0 & 0 \\ \hline \end{array}$$