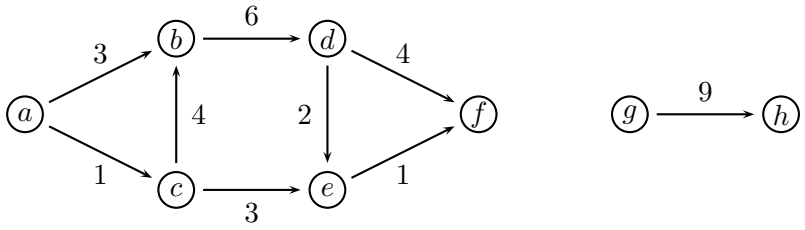
Department of Computing and Information Systems
COMP90038 Algorithms and Complexity Tutorial Week 12

Sample answers

The exercises

1. 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:

$T \leftarrow$ TopSort($\langle V, E \rangle$) — List of nodes sorted topologically
for $t \in T$ (in topological order) **do**
  $L[t] \leftarrow max(\{0\} \cup \{L[u] + weight[u,t] \mid (u,t) \in E\})$
return $max\{L[v] \mid v \in V\}$

For the sample graph, topsort yields the sequence A, C, B, D, E, F, G, H (or, alternatively, 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							
	0		1					
	0	5	1					
	0	5	1	11				
	0	5	1	11	13			
	0	5	1	11	13	15		
	0	5	1	11	13	15	0	
	0	5	1	11	13	15	0	9

2. 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 $I_1, \ldots, I_n$ 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.

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

	a	b	c	d
a	0	0	1	1
b	0	0	1	0
c	1	0	0	0
d	0	0	0	0

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):

	a	b	c	d
a	0	0	1	1
b	0	0	1	0
c	1	0	1	1
d	0	0	0	0

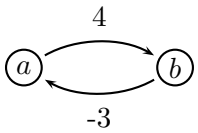
$\Rightarrow$

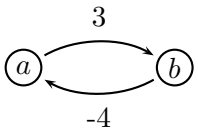
	a	b	c	d
a	1	0	1	1
b	0	0	1	0
c	1	0	1	1
d	0	0	0	0

$\Rightarrow$

	a	b	c	d
a	1	0	1	1
b	1	0	1	1
c	1	0	1	1
d	0	0	0	0

4. Floyd's algorithm sometimes works even if we allow negative weights in a dag.





For example, for the left graph above, it will produce these successive distance matrices:

$$D^0 = D^1 = D^2 = \begin{bmatrix} 0 & 4 \\ -3 & 0 \end{bmatrix}$$

What happens for the right graph above? What do D^0, D^1 and D_2 look like? Explain why D^2 ends up giving an incorrect result in this case (but not in the previous case).

Answer: We get the following distance matrices:

$$D^0 = \begin{bmatrix} 0 & 3 \\ -4 & 0 \end{bmatrix} \quad D^1 = \begin{bmatrix} 0 & 3 \\ -4 & -1 \end{bmatrix} \quad D^2 = \begin{bmatrix} -1 & 2 \\ -5 & -2 \end{bmatrix}$$

This is where the algorithm stops, and the distances seem all wrong. The problem is that there is a *negative-weight cycle* in the graph. If we kept going round that cycle, we would get smaller and smaller negative "distances"—it does not make sense to ask for the path that has the smallest accumulated sum of weights.

If negative weights are possible, we really should add this test to Floyd's algorithm: If, at any point a negative element is created in the diagonal of the matrix, halt with some error message.

5. We are given a sequence of "connection points" spaced out evenly along a straight line. There are n white, and n black points, in some (random) order.

The points are spaced out evenly, so that the distance between two adjacent points is 1.

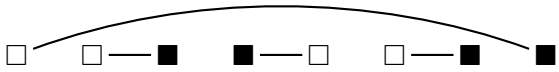
The points need to be connected, so that each white point is connected to exactly one black point and vice versa. However, the total length of wire used must be kept as small as possible.

Consider the following (greedy) algorithm to solve the problem:

$k \leftarrow 1$
while there are still unconnected points **do**
    create all possible connections of length k
    $k \leftarrow k + 1$

Argue the correctness of this algorithm, or, alternatively, devise an example that proves that it may not produce an optimal wiring.

Answer: The algorithm does not always yield the shortest wiring, witness the example:



This uses wire of length 10, but there are solutions that use only wire of length 8 (find one).

An efficient way of finding a best wiring makes use of a stack. The algorithm walks through the sequence of connection points in a linear fashion, starting with an empty stack. For each point it meets, it checks whether it has a point of the opposite colour on the top of the stack, and if so, the stack is popped, and the two points are connected. Otherwise it pushes the newly met point onto the stack.