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

Sample answers

The exercises

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

$$\begin{tikzcd} a \arrow[r, bend left, "4"] & b \arrow[l, bend left, "-3"] \end{tikzcd} \qquad \begin{tikzcd} a \arrow[r, bend left, "3"] & b \arrow[l, bend left, "-4"] \end{tikzcd}$$

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.

80. 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.

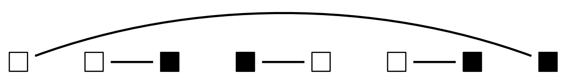
The points need to be connected, so that each white point is connected to exactly one black point and vice versa. However, the total length of wire used must be kept as small as possible.

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.

81. Recall the definition of the knapsack problem. Given a set of items $S = \{i_1, i_2, \dots i_n\}$ with

- weights: $w(i_1), w(i_2), \dots, w(i_n)$
- values: $v(i_1), v(i_2), \dots, v(i_n)$

and a knapsack of capacity W, find the most valuable selection of items that will fit in the knapsack. That is, find a set $I \subseteq S$ such that $\sum_{i \in I} w(i) \leq W$ and so that $\sum_{i \in I} v(i)$ is maximised.

Define the *benefit* of an item i to be the rational number $v(i)/w(i)$. Consider the following greedy approach to the problem:

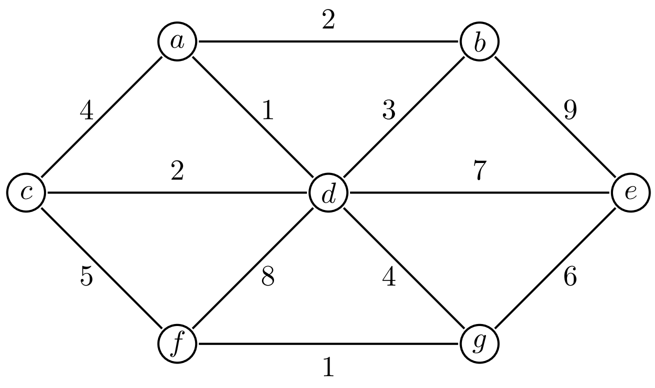
```
Let $A[1]...A[n]$ hold the items from $S$, in decreasing order of benefit
$val \leftarrow 0$
$weight \leftarrow 0$
$k \leftarrow 1$
**while** $k \leq n \wedge weight + w(A[k]) \leq W$ **do**
    select $A[k]$
    $val \leftarrow val + v(A[k])$
    $weight \leftarrow weight + w(A[k])$
    $k \leftarrow k + 1$
```

That is, at each step, from the remaining items we simply pick the one that has the greatest benefit. Give a simple example to show that this greedy algorithm does not solve the knapsack problem.

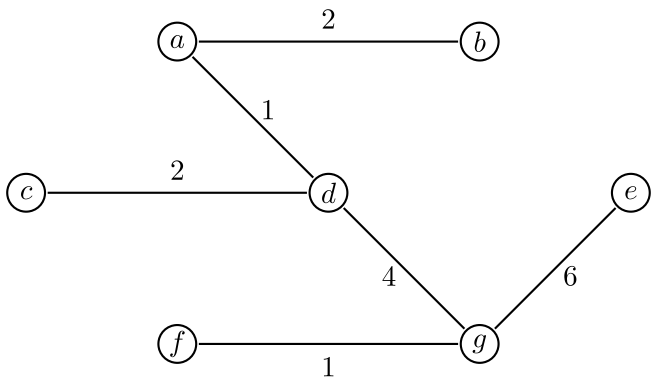
Answer: Assume capacity $W = 8$ and take, for example, three items with (value,weight) pairs as follows: i_1 : (6,5), i_2 : (4,4), i_3 : (3,4).

The three are already in decreasing order of benefit. If we pick i_1 greedily (since it has the highest benefit), there is room for no other item, so the value we achieve is 6. Of course, the better solution is to pick i_2 and i_3, for a total value of 7.

82. Work through Prim's algorithm for the graph below. Assume the algorithm starts by selecting node a. Which edges are selected for the minimum spanning tree, and in which order?



Answer: The first edge found is a–d. After that, a–b and c–d are added in either order. Last come d–g, g–f, and e–g. This gives the following minimum spanning tree, of cost 16:

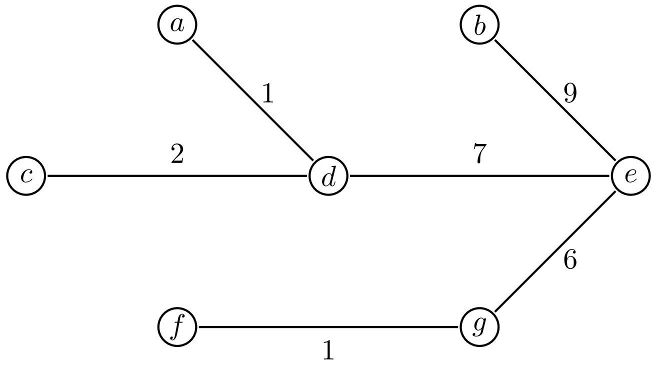


83. Use Dijkstra's algorithm to find the shortest paths for node e in the previous question's graph. That is, run the algorithm to determine the length of the shortest path from e to v, for all seven nodes v. Is the shortest path from e to b part of the graph's minimum spanning tree?

Answer: The shortest paths from e are found to be:

a :	8
b :	9
c :	9
d :	7
e :	0
f :	7
g :	6

We already saw in the previous question that the e–b edge is not part of the graph's minimum spanning tree. These are the edges captured by the 'prev' relation:

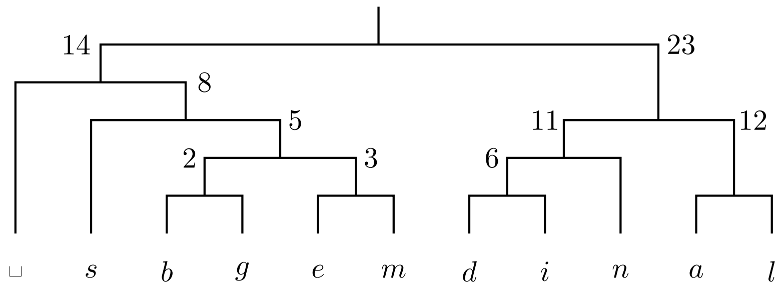


84. Lemuel Gulliver wishes to compress the string “all␣big␣endians␣and␣all␣small␣endians”. Help him by building a Huffman tree for the string (there may be several valid trees) and assign a binary code accordingly, to each of the eleven characters involved (we have used ␣ to make each space character visible). The frequencies are:

a	b	d	e	g	i	l	m	n	s	␣
6	1	3	2	1	3	6	1	5	3	6

How many bits are required for the encoded string?

Answer: Different trees are possible. Here is one:



Based on this Huffman tree, we assign codes as follows:

a	b	d	e	g	i	l	m	n	s	␣
110	01100	1000	01110	01101	1001	111	01111	101	010	00

The encoded string is 1101111110001100100101101000111010110001001110101010001101011000 001101111110001001111110111111000111010110001001110101010 and it consists of 121 bits.