

Solutions to Test 1

Problem 1

a. It is not true that one can always find a minimum spanning tree with the given property. When S contains edges that form a cycle, we cannot find a spanning tree whose edges are a subset of S , because a spanning tree is by definition acyclic.

Marking guide: 0.5 marks for simply stating the correct answer and 1.5 marks for the **correct** justification, as above. Arguing that the spanning tree that satisfies the given property is not necessarily the spanning tree of minimum cost is **not** correct, and was not awarded any marks.

b. We run Kruskal's algorithm for the graph $G' = (V, E \setminus S)$, starting with the connected components defined by the edgeset S . Let E' be the edges of the spanning tree returned by Kruskal's algorithm; the spanning tree we seek is $T = (V, E' \cup S)$.

Marking guide: In almost all cases, incorrect algorithms were given at most 2 points. A common wrong answer is to run Kruskal's algorithm without the correct initialization step shown above, and then argue that every time the algorithm encounters an edge in S it has to add it; this algorithm may introduce cycles in the output graph. If you don't mention Kruskal's algorithm but rather describe it (e.g., in pseudocode), you have to be careful to state it clearly and precisely (e.g., mention the precise ordering of the edges). As usual, partial marks were taken off if the writing style is not good, and it is hard to make sense of what the algorithm *precisely* does.

c. The complexity of the algorithm is dominated by the complexity of Kruskal's algorithm.

Marking guide: 0.5 marks for the answer and 0.5 marks for justification. The marker was lenient as long as the complexity was something that made sense; for instance, you cannot claim that the complexity is $O(n)$ or $O(m)$ when it is a well-known fact that sorting takes $\Theta(m \log m)$ time.

Problem 2

As noticed by some students, this scheduling problem is very similar to one presented in lectures where each task has a latest finishing time as well as a duration. Note that the test version is a simpler question since task T_i **MUST** start at time s_i and end at time f_i . This makes the algorithm in part (e) simpler than the one seen in class.

Part (a)

This question was quite well done. There are a number of different parameters on which the sort could be done. The most popular and obvious is "sorting by gain", where the tasks are sorted by non-increasing gain. The justification for this is that we want to optimize the total gain and thus our greedy algorithm would choose the task with the largest gain first.

Other possibilities (which all received full marks, if properly justified) are: gain per unit execution time, finishing times, and starting times.

One half mark was deducted if there was no justification.

Part (b)

Again this question was well done. If “sorting by gain” was the answer for part (a), then the following tasks show that this greedy algorithm is not optimum:

$\{1, 3, 3\}, \{1, 2, 2\}, \{2, 3, 2\}$

The greedy algorithm will examine these tasks in this order and can only schedule the first task with a total gain of 3 units. The optimum schedule however, would choose the second and third tasks with a total gain of 4 units.

For each of the other possible greedy approaches, there are similarly easy counter examples.

Some students just gave a set of tasks without explaining what their greedy algorithm would do. At least one half mark was deducted for this.

Part (c)

As with the scheduling problem discussed in class, our two dimensional array A , is designed as follows:

The rows are indexed from 0 to n , the number of tasks, and the columns are indexed from 0 to f_{max} , the latest finishing time. $A(i, j)$ stores the maximum profit that can be achieved by scheduling tasks T_1, T_2, \dots, T_i such that all tasks finish by time j .

Part (d)

As with the scheduling problem discussed in class, the maximum profit that can be achieved for the whole problem is stored in $A(n, f_{max})$, the bottom right entry.

Part (e)

This algorithm is similar to, but easier than the one shown in class.

1. Sort the tasks by non-decreasing finishing times.
2. for j from 0 to f_n
3. $A(0, j) \leftarrow 0$
4. for i from 1 to n
5. for j from 1 to f_n
6. if $f_i > j$ then $A(i, j) \leftarrow A(i-1, j)$
7. else $A(i, j) \leftarrow \text{MAX}\{A(i-1, j), A(i-1, s_i) + g_i\}$

The justification received up to 2 marks and had to touch on the decisions made in lines 6 and 7. Here is a complete justification which is more than expected in order to receive full marks.

If $f_i > j$, then it is impossible to schedule task i so that it finishes by time j , and thus the maximum profit that can be achieved by scheduling tasks T_1, T_2, \dots, T_i such that all tasks finish by time j is the maximum profit that can be achieved by scheduling tasks T_1, T_2, \dots, T_{i-1} in this time frame (i.e. the value stored in $A(i-1, j)$). Otherwise, it is possible to schedule task T_i and we have to determine whether it is worth doing so. If we don't schedule T_i , then the maximum profit is stored in $A(i-1, j)$; if we do, then the maximum profit is the sum of g_i and the maximum profit that can be achieved by scheduling tasks T_1, T_2, \dots, T_{i-1} so that they all finish before T_i starts (i.e. the value stored in $A(i-1, s_i)$).

Problem 3

We show one of the possible solutions.

$$s \xrightarrow{0/3} C \xrightarrow{0/3} B \xrightarrow{0/4} A \xrightarrow{0/2} D \xrightarrow{0/2} t \quad \text{Residual capacity: 2.}$$

$$s \xrightarrow{0/7} E \xrightarrow{0/5} D \xrightarrow{0/6} C \xrightarrow{0/4} F \xrightarrow{0/5} t \quad \text{Residual capacity: 4.}$$

$$s \xrightarrow{0/6} A \xleftarrow{2/4} B \xrightarrow{0/3} t \quad \text{Residual capacity: 2.}$$

$$s \xrightarrow{2/3} C \xleftarrow{4/6} D \xrightarrow{0/1} B \xrightarrow{2/3} t \quad \text{Residual capacity: 1.}$$

Total flow: 9.

To justify that the flow is maximum, observe that the $s-t$ cut (V_s, V_t) with $V_s = \{s, A, B, C, D, E\}$, and $V_t = \{F, t\}$ has total capacity 9. From the *max-flow min-cut* theorem we know that the maximum flow is equal to the minimum cut, hence the flow cannot exceed 9, and it has therefore to be optimal (maximum).

Marking guide: 3.5 marks for the augmenting paths and showing what the max flow precisely is. Incorrect flow computations get **at most** 2 marks for this part. One mark deduction if the flows are not shown when stating the augmenting paths, and 0.5 or 1 mark off for not using the proper direction in certain critical edges. The residual capacity for the first augmenting path counts for 0.5 marks.

Stating **explicitly** what is the relationship between the flow and the cut counts for 0.5 marks. You **must** point out that they should be equal. Showing a correct cut counts for 1.5 marks (no marks given for wrong cuts). Notice also how we formally describe a cut: as two subsets of the vertex set, and not by drawing a line in the figure, which can be very misleading (although no points were deducted in this case).