

NOTE TO STUDENTS: This file contains the solutions to term test 2 together with the marking scheme and comments for each question. Please read the solutions and the marking schemes and comments carefully. Make sure that you understand why the solutions given here are correct, that you understand the mistakes that you made (if any), and that you understand *why* your mistakes were mistakes.

Also, remember that although you may not agree completely with the marking scheme given here, it was followed the same way for all students. We will remark your test only if you can clearly demonstrate that the marking scheme was not followed correctly, and only if your test was written in pen. We will make *no* exception to the marking scheme.

For all remarking requests, please submit your request **in writing** directly to your instructor. For all other questions, please don't hesitate to ask your instructor during office hours or by e-mail.

Question 1. [8 MARKS]

Recall the *Task Scheduling Problem* from the previous test.

Input: A set of tasks $T = \{T_1 = (s_1, f_1, g_1), T_2 = (s_2, f_2, g_2), \dots, T_n = (s_n, f_n, g_n)\}$, where $s_i \in \mathbb{N}$ is the start time, $f_i \in \mathbb{N}$ is the finish time, and $g_i \in \mathbb{R}^+$ is the gain for task T_i (each task is well-defined, *i.e.*, $s_i < f_i$ for all i).

Output: A subset of tasks $S \subseteq T$ such that every task in S can be scheduled on a single processor and the total gain of S is maximum. (For each i , task T_i can only be scheduled to start at time s_i and finish at time f_i , and no two tasks in S can overlap, although it is allowed to start one task at the same time that another task finishes.)

To use a Dynamic Programming algorithm to solve this problem, we first sort the tasks into non-decreasing finish time and then create a 2-dimensional table A , where $A[i, j]$ stores the maximum gain that can be achieved by scheduling tasks T_1, T_2, \dots, T_i such that all tasks complete no later than time j . On completion of the Dynamic Programming algorithm, the maximum gain possible is stored in the bottom right entry of A , namely $A[n, f_n]$.

We now want to use this table to output a schedule that achieves the maximum gain.

Part (a) [6 MARKS]

Assume that the values in array A have already been computed correctly. Write an algorithm in pseudo-code that takes A as input and uses it to output a schedule with the maximum gain.

SOLUTION:

```

OUTPUTSCHEDULE( $T, A$ )    ( $T$  is "Input" as described above)
   $S \leftarrow \{\}$         (subset of tasks as described in "Output" above)
   $j \leftarrow f_n$       (finish time for schedule, initially latest possible)
  for  $i \leftarrow n$  downto 1 do
    if  $A[i, j] > A[i-1, j]$  then
       $S \leftarrow S \cup \{T_i\}$     (schedule task  $T_i$ )
       $j \leftarrow s_i$           (update finish time to avoid overlap)
      (else do nothing: gain  $A[i, j]$  can be achieved without task  $T_i$ )
  return  $S$ 

```

MARKING SCHEME:

- 1 mark for trying to find and return (or print) a subset of tasks (notice that the problem description states *clearly* that the output is "a subset of tasks $S \subseteq T$ ").
- 1 mark for starting at the bottom-left corner n, f_n .
- 2 marks for comparing $A[i, j]$ with $A[i-1, j]$ to determine whether or not T_i should be scheduled.
- 1 mark for updating i and j correctly at the end of each iteration.
- 1 mark for general presentation (in particular, using pseudo-code).

Question 1. (CONTINUED)**Part (a)** (CONTINUED)

EXPECTED ERRORS:

- F. At most 1 mark for answering the wrong question and only trying to compute the values of A .
- G. At most 2 marks for trying to write a Greedy algorithm instead of using Dynamic Programming.

MARKING COMMENTS:

- “I don’t know how to answer” for part (a) automatically implied the same for part (b).
- (minimum penalty: -1 B) Many students expressed their answer in terms of i, j , as if these were input values. Answers must be expressed in terms of the input as described above (*i.e.*, in terms of n).
- (minimum penalty: -2 CD) Some students compared $A[i, f_i]$ with $A[i-1, s_i]$ to determine whether or not to schedule task T_i . However, you really need a separate time index j to avoid overlap (*i.e.*, in case $s_i < f_{i-1}$ so that your algorithm can skip overlapping tasks).
- (minimum penalty: -2 DH) Some students had simple loops over the index j (from f_i down to 0), which does not work: time must be updated appropriately depending on the tasks scheduled, to avoid overlap.
- (minimum penalty: -3 CD) For algorithms whose main idea to determine whether or not to schedule a task was: “move left in the array until the value changes”. This does not work: you have to move up.
- (minimum penalty: -1 E) Some students did not use pseudo-code, even though the question explicitly asked for it. Also, there was a lot of bad use of notation (*e.g.*, referring to i or j without any indication of what value they represent, writing things like “ $f_i = s_i$ ” which change the input, or even worse, using specific C/C++ constructs).
- Note that there was no excuse for missing this question! The similar problem of Job Scheduling with Durations and Profits was covered in lectures, and the basic Dynamic Programming algorithm was posted in the solutions for Test 1. It is *your* responsibility to make sure that you understand the material: when we take the time to put together solutions and post them, we expect you to take the time to read and understand them (especially if you could not solve the problem on your own)! You cannot expect that marks will be “magically” adjusted at the end of the course: adjustments will be made only if we feel that some of the questions we asked were really too difficult, but not if we feel that people got low marks mainly because they missed easy questions (like this one).

Part (b) [2 MARKS]

Briefly justify that your algorithm is correct.

SOLUTION:

From the recurrence relation satisfied by the values in A , we know that task T_i should be scheduled iff $A[i, j] > A[i-1, j]$ (the only other possibility is that $A[i, j] = A[i-1, j]$). Also, the maximum gain is stored in $A[n, f_n]$. Hence, we initialize the finish time j to the last possible deadline and for each task (starting from the last one and going back), we compare the current entry with the one immediately above to decide whether or not task T_i should be scheduled, moving j back appropriately (to the start time s_i) for every task T_i that is scheduled, to ensure that no tasks overlap.

MARKING SCHEME:

- H. If the algorithm is mostly correct, 2 marks for mentioning the fact that $A[i, j] > A[i-1, j]$ means T_i should be scheduled.
- J. If the algorithm is mostly incorrect, or there are many small mistakes, or it is badly described and hard to understand, at most 1 mark for trying to explain how decisions are made about which tasks are scheduled and why this yields a schedule with maximum gain.

Question 2. [7 MARKS]

Consider the problem of assigning tutors to serve as invigilators for course examinations.

Each tutor, on reading the examination schedule, determines which courses he/she is able to invigilate. (Obviously, no tutor would try to invigilate two examinations held at the same time!) The set of tutors and courses may be considered as a bipartite graph with vertex set $T = \{t_1, t_2, \dots, t_m\} \cup C = \{c_1, c_2, \dots, c_n\}$ together with edges from T to C such that edge (t_i, c_j) indicates that tutor i can invigilate course j . Furthermore, we restrict the problem so that each tutor can invigilate at most 2 examinations and course c_j requires at most x_j invigilators (for $1 \leq j \leq n$).

The problem is to construct an assignment of tutors to courses that maximizes the number of tutors assigned.

Part (a) [5 MARKS]

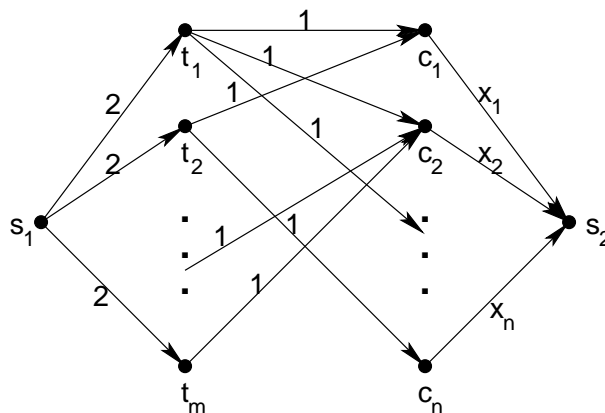
Show how to solve this problem by translating it into a Network Flow problem, *i.e.*, describe clearly how to construct a network from an input to the problem given above.

SOLUTION:

Create a network as follows:

- Vertices: $\{s_1, s_2\} \cup T \cup C$.
- Edges: use input edges, directed from T to C , each with capacity 1; plus edges (s_1, t_i) for each tutor $t_i \in T$, each with capacity 2; plus edges (c_j, s_2) for each course $c_j \in C$, where capacity of edge (c_j, s_2) is x_j .

This is illustrated in the picture below:



After finding a maximum flow in the network, assign each tutor t_i to invigilate the course(s) c_j for which the flow of edge (t_i, c_j) is equal to 1.

MARKING SCHEME:

- 1 mark for layout (bipartite graph plus vertices s_1, s_2)
- 1 mark for capacities of edges (s_1, t_i)
- 1 mark for capacities of edges (c_j, s_2)
- 1 mark for indicating the direction of edges
- 1 mark for describing correctly the edges in the “bipartite graph”

NOTE: You must be precise when describing the graph G . A figure suffices only if *all* important information can be seen in the figure, otherwise (partial) marks are deducted.

Question 2. (CONTINUED)**Part (b)** [2 MARKS]

Give a brief justification that the maximum flow in your network yields an assignment of tutors to courses that obeys the constraints of the problem and maximizes the number of tutors assigned.

SOLUTION:

Any flow in the network must obey the constraints of the problem: no tutor can be assigned to invigilate more than 2 courses because the flow going into each vertex t_i is at most 2; no course c_j will be assigned more than x_j invigilators because the flow coming out of each vertex c_j is at most x_j .

Moreover, the total flow in the network represents the total number of tutors assigned to invigilate exams, so any maximum flow assigns a maximum number of tutors.

MARKING SCHEME:

- 1 mark for feasibility, *i.e.*, stating *precisely* why the assignment of capacities enforces the constraints of the problem
- 1 mark for explaining why the maximum flow implies an optimal solution

Question 3. [6 MARKS]**Part (a)** [3 MARKS]

Prove the transitivity of $\xrightarrow{\text{poly}}$, *i.e.*, show that for all languages X, Y, Z , if $X \xrightarrow{\text{poly}} Y$ and $Y \xrightarrow{\text{poly}} Z$, then $X \xrightarrow{\text{poly}} Z$.

SOLUTION:

$X \xrightarrow{\text{poly}} Y$ means that there is a polytime computable function f such that $x \in X$ iff $f(x) \in Y$ for all strings x ; $Y \xrightarrow{\text{poly}} Z$ means that there is a polytime computable function g such that $y \in Y$ iff $g(y) \in Z$ for all strings y .

Define a function $h(x) = g(f(x))$. h is computable in polytime since each of f and g is computable in polytime. Also, $x \in X$ iff $f(x) \in Y$ iff $g(f(x)) \in Z$ for all strings x , *i.e.*, $x \in X$ iff $h(x) \in Z$ for all strings x . By definition, this means that $X \xrightarrow{\text{poly}} Z$.

MARKING SCHEME:

- 1 mark for describing function h
- 1 mark for arguing about the “if and only if” part of the proof (0.5 marks for each part, if you argued separately or did only one direction)
- 1 mark for explaining why h is polytime (there is not much to explain, but you must give *some* justification; saying something as simple as shown above suffices)

COMMON ERRORS:

- Missing the “only if” part of the proof. This has been a very common error so far.
- Interpreting the poly-time reduction as a “hardness” operator (*i.e.*, trying to argue about the relative difficulty of X, Y, Z). While in a certain context this is true, it is not what the definition of reduction is. This is supposed to be a proof: *you must use the definition*, and it is not acceptable to give arguments based on the vague, intuitive meaning of the concepts. (Doing this will get you 0.5 marks only.)
- Several solutions marked so far are not clear and it is hard to see which parts of a correct solution were shown and which were ignored.

Question 3. (CONTINUED)**Part (b)** [3 MARKS]

Suppose that we have shown a language X to be NP -complete, and that for part of our proof we showed $Y \xrightarrow{\text{poly}} X$ for some language Y known to be NP -complete. Does $X \xrightarrow{\text{poly}} Y$? Justify.

SOLUTION:

Yes.

Step 1: Since X is NP -complete, then in particular, $X \in NP$.

Step 2: Since Y is NP -complete, then in particular, Y is NP -hard. By definition, this means that $L \xrightarrow{\text{poly}} Y$ for all $L \in NP$.

Step 3: By steps 1 and 2, we conclude that $X \xrightarrow{\text{poly}} Y$.

MARKING SCHEME:

- 1 mark for every step. This is provided that the overall solution makes sense. If it does not, only partial marks are given. (For instance, just stating Steps 1 and 2 does not mean an automatic 2 marks, if the context is not right.)
- If you answered “no”, then you get at most 1 mark, depending on how reasonable the justification seems.

Question 4. [9 MARKS]

Consider the following WEIGHTED VERTEX COVER problem.

Input: A graph $G = (V, E)$ with weighted vertices (i.e., each vertex $v \in V$ is assigned an integer weight $w(v) \in \mathbb{N}$) together with an integer target $K \in \mathbb{N}$.

Output: Is there a vertex cover of G with total weight at most K , i.e., is there a set of vertices $C \subseteq V$ such that every edge has at least one endpoint in C ($\forall (u, v) \in E, u \in C$ or $v \in C$) and the total weight of C is at most K ($\sum_{v \in C} w(v) \leq K$)?

Prove that WEIGHTED VERTEX COVER is NP -complete. You may refer to any of the problems listed on the last page of this test and use the fact that all of them are known to be NP -complete.

(HINT: The point of this question is to test that you understand and remember the steps for showing that a problem is NP -complete, by doing it on a specific problem. Therefore, make sure that you explain what you are doing; in particular, you may find it helpful to start by writing an outline of the steps that you will follow. Also note that you may not need all of the space below and on the next page.)

SOLUTION:

In this solution, we will use the abbreviations “WVC” to stand for WEIGHTED VERTEX COVER and “VC” to stand for VERTEX COVER (it was fine to do this in your solution as well).

First, we show that $WVC \in NP$. Given an input for WVC ($G = (V, E)$ with weights $w(v)$, and $K \in \mathbb{N}$), a certificate consists of a set of vertices C . The verification algorithm checks that $C \subseteq V$, that every edge $e \in E$ has at least one endpoint in C , and that $\sum_{v \in C} w(v) \leq K$. Clearly, this can be done in time polynomial in the size of the input. Moreover, from the definition of the problem, it is clear that G contains a vertex cover of weight no more than K iff there is a value of the certificate that makes the verification algorithm accept.

Second, we show that WVC is NP -hard. This will be achieved by proving that $VC \xrightarrow{\text{poly}} WVC$, since VC is known to be NP -complete (hence NP -hard). Intuitively, VC is just a special case of WVC where the weight of each vertex is 1. However, for the sake of completeness, we will describe the construction in detail.

Question 4. (CONTINUED)

SOLUTION (CONTINUED):

Given an input $G = (V, E)$ and $K \in \mathbb{N}$ for VC, construct an input $G' = (V', E')$ and $K' \in \mathbb{N}$ for WVC as follows:

$$\text{let } G' \leftarrow G;$$

$$\text{let } w(v) \leftarrow 1 \text{ for all } v \in V';$$

$$\text{let } K' \leftarrow K.$$

Clearly, this construction can be carried out in polytime (measured as a function of the size of G, K). Moreover, G contains a vertex cover of size no more than K iff G' contains a vertex cover of total weight no more than K' : by construction, the weight of any vertex cover in G' is equal to its size, and G and G' contain the same vertices and edges, so any vertex cover in one of them is also a vertex cover in the other.

MARKING SCHEME—STRUCTURE: These marks are awarded purely for the structure of the proof, even if the contents are incorrect or missing (as long as there is enough information to be able to find the structure).

- A. 1 mark for stating “we show $WVC \in NP$ ” (or *clearly* trying to do this).
- B. 1 mark for trying to describe an appropriate certificate and verification algorithm.
- C. 1 mark for stating “we show WVC is NP -hard” (or *clearly* trying to do this).
- D. 1 mark for trying to reduce a known NP -hard problem D to WVC by constructing an input to WVC given an input to D .
- E. 1 mark for trying to argue that the answers are the same for both inputs.

MARKING SCHEME—CONTENT: These marks are awarded purely for the contents of the proof, even if the structure is missing or incorrect (as long as there is enough information to be able to find the contents).

- F. 1 mark for describing a correct certificate and verification algorithm to show that $WVC \in NP$.
- G. 1 mark for having a correct and clearly described polytime construction to show reduction to WVC .
- H. 2 marks for correctly stating that the construction can be done in polytime and correctly arguing that the answers for both inputs are the same.

EXPECTED ERRORS: The penalties listed here are *minimum* penalties: other errors can combine with these to reduce the mark further.

- (−2 BF) Vague or missing arguments that $WVC \in NP$, e.g., no description of certificate or verification algorithm, no argument that algorithm accepts for some value of certificate iff input is a yes-instance.
- (−2 DG) Doing reduction in the wrong direction.
- (−3 DGH) Describing reduction in terms of answer for the inputs, e.g., “take the vertex cover for input G and do the following with it”, or doing two constructions (one in each direction) instead of one construction together with an argument that the answers are the same.
- (−1 F) Including K in the certificate (you need to take value from input) or forgetting to verify that certificate C satisfies $C \subseteq V$.
- (−1 H) Having a reduction that consists only of something like: “Let weight of every vertex be 1; by restriction, VC is a special case of WVC .” Your proof needed at least a one sentence argument that this works.
- (−1 H) Saying only “answers are the same” or something equivalent, but with no attempt at justifying why this is the case.