NOTE TO STUDENTS: This file contains sample solutions to the term test 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.

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 clearly demonstrate that the marking scheme was not followed correctly or that the marker misunderstood your solution.

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

Question 1. [9 MARKS]

Consider the following *Task Scheduling* problem.

Input: Tasks $T_1 = (r_1, d_1), T_2 = (r_2, d_2), \ldots, T_n = (r_n, d_n)$, where integers $r_1, r_2, \ldots, r_n$ denote the *reward* of each task and positive integers $d_1, d_2, \ldots, d_n$ denote the *deadline* of each task. The time required to complete a task is always one unit of time.

Output: Non-negative integers $t_1, t_2, \ldots, t_n$ to denote the *time slot* for scheduling each task, with maximum total reward for all tasks scheduled by their deadline $\left(\sum_{0<t_i\leqslant d_i} r_i\right)$. To avoid task conflicts, we require that $t_i = 0$ (when T_i is not scheduled) or $t_i \neq t_j$ for all $1 \leqslant i, j \leqslant n$.

For example, suppose we have five tasks $T_1 = (10, 3)$, $T_2 = (5, 1)$, $T_3 = (3, 1)$, $T_4 = (2, 2)$, and $T_5 = (1, 2)$. One optimal schedule is $3, 1, 0, 2, 0$. This enables us to finish T_1, T_2, T_4 with total reward 17.
Now consider the following greedy algorithm.

```
sort tasks such that r_1 ⩾ r_2 ⩾ ··· ⩾ r_n
for i ← 1, 2, ..., n:
    # Schedule T_i as late as possible.
    t_i ← d_i
    while t_i > 0 and t_i = t_j for some j ∈ {1, 2, ..., i − 1}:
        t_i ← t_i − 1
```

Part (a) [3 MARKS]

Give a precise definition of *promising partial solution* for the algorithm above. (This involves giving a precise definition for three separate terms: *partial solution*, *extends* and *promising*.)

SAMPLE SOLUTION:

- For $0 \leqslant i \leqslant n$, $t_1, t_2, \ldots, t_i$ is a *partial solution*.
- Optimum solution $t_1^*, t_2^*, \ldots, t_n^*$ *extends* partial solution $t_1, t_2, \ldots, t_i$ iff $t_1^* = t_1, t_2^* = t_2, \ldots, t_i^* = t_i$.
- Partial solution $t_1, t_2, \ldots, t_i$ is *promising* iff *some* optimum solution $t_1^*, t_2^*, \ldots, t_n^*$ extends $t_1, t_2, \ldots, t_i$.

MARKING SCHEME:

- One mark for each definition.
- **common error** [−1]: using sets to describe "extension" (works only when solutions are sets)

Part (b) [1 MARK]

Give a precise statement of the loop invariant used to prove that the algorithm above always finds an optimum solution. NOTE: *There is **nothing** to prove for this question!*

SAMPLE SOLUTION:

"Partial solution $t_1, t_2, \ldots, t_i$ is promising."

MARKING SCHEME:

- all-or-nothing

Part (c) [5 MARKS]

Prove the following *exchange lemma* that could be used in the proof of your loop invariant—**do *not* prove your loop invariant; only the exchange lemma stated here**.

> If $t_1^*, t_2^*, \ldots, t_n^*$ is an optimum solution that extends $t_1, t_2, \ldots, t_i$ and $t_{i+1}^* \neq t_{i+1}$, then there is another optimum solution $t_1^{**}, t_2^{**}, \ldots, t_n^{**}$ that extends $t_1, t_2, \ldots, t_i, t_{i+1}$.

SAMPLE SOLUTION:

Suppose $t_1^*, t_2^*, \ldots, t_n^*$ is an optimum solution that extends $t_1, t_2, \ldots, t_i$ and $t_{i+1}^* \neq t_{i+1}$. Then $t_{i+1}^* < t_{i+1}$ since t_{i+1} is the latest time slot possible for task T_{i+1}. Consider the value of t_{i+1}: either $t_{i+1} = t_k^*$ for some $k \in \{i+2, i+3, \ldots, n\}$, or $t_{i+1} \neq t_k^*$ for all $k \in \{i+2, i+3, \ldots, n\}$.

If $t_{i+1} = t_k^*$ for some $k \in \{i+2, i+3, \ldots, n\}$, let $t_j^{**} = t_j^*$ for $j = 1, 2, \ldots, n$, except $t_{i+1}^{**} = t_{i+1}$ and $t_k^{**} = t_{i+1}^*$. Then, $t_1^{**}, \ldots, t_n^{**}$ extends $t_1, \ldots, t_{i+1}$ and has the same total reward as $t*_1, \ldots, t_n^*$ since $t_{i+1}^{**} = t_{i+1}$, $t_k^{**} = t_{i+1}^* < t_{i+1} = t_k^*$, and $t_j^{**} = t_j^*$ for all other indices j.

If $t_{i+1} \neq t_k^*$ for all $k \in \{i+2, i+3, \ldots, n\}$, let $t_j^{**} = t_j^*$ for $j = 1, 2, \ldots, n$, except $t_{i+1}^{**} = t_{i+1}$. Then, $t_1^{**}, \ldots, t_n^{**}$ extends $t_1, \ldots, t_{i+1}$ and has the same total reward as $t*_1, \ldots, t_n^*$ since $t_{i+1}^{**} = t_{i+1}$, and $t_j^{**} = t_j^*$ for all other indices j.

MARKING SCHEME:

- **Idea** [2 marks]: correct main idea—schedule T_{i+1} at the greedy time t_{i+1} and move any task already scheduled there to t_{i+1}^*, which can only be earlier
- **Details** [3 marks]: correct arguments that $t_1^{**}, \ldots, t_n^{**}$ is valid and optimum and extends $t_1, \ldots, t_{i+1}$

Question 2. [8 MARKS]

Let $A = a_1, a_2, \ldots, a_n$ and $B = b_1, b_2, \ldots, b_m$ be two sequences of integers (postive, negative, or zero). Our goal is to convert A to B by performing the **minimum** number of *modification operations* on A. There are four possible modification operations:

1. Delete a number from sequence A
2. Insert a zero in any position in sequence A
3. Increase a number in A by one
4. Decrease a number in A by one

For example, $A = 1, 2, 3, 5, 8, 13, 21$ can be converted into $B = 1, 2, 4, 8, 16, -2$ with only nine operations: delete 21, insert 0 at the end, decrease 0 twice (to become -2), increase 13 three times (to become 16), decrease 5 one time, and delete 3. Note that this is not the only solution with only nine operations.

Part (a) [3 MARKS]

Define a *two-dimensional* array to solve this problem using dynamic programming. State clearly the range of valid indices for your array, the sub-problem corresponding to each array entry, and the value stored in each entry. (HINT: Consider prefix sequences of A and B. Note that you have a lot more room than necessary for this: don't feel obligated to fill it all!)

SAMPLE SOLUTION:

Let $M[i, j]$ denote the minimum number of modification operations required to convert $a_1, \ldots, a_i$ into $b_1, \ldots, b_j$, for $0 \leqslant i \leqslant n$ and $0 \leqslant j \leqslant m$.

MARKING SCHEME:

- **Indices** [1 mark]: correct index range (starting at 1 is okay but will affect the next part)
- **Sub-problem** [1 mark]: correct sub-problem (stated explicitly)
- **Value** [1 mark]: correct value
- **common error**: define an array with two indices $[i, d]$, where i represents the last sequence item a_i, b_i (restricted to be the same for both sequences) and d represents the operations allowed

Part (b) [5 MARKS]

On the next page, give a recurrence relation for your array values. Include appropriate base case(s) and explain what you are computing and why. (HINT: For any two sequences $a_1, \ldots, a_n$ and $b_1, \ldots, b_m$, try every way to use the operations to generate b_m, either directly from a_n or without it.)

SAMPLE SOLUTION:

$$
\begin{aligned}
M[0,0] &= 0 \\
M[i,0] &= M[i-1,0] + 1, \quad \text{for } 1 \leqslant i \leqslant n \\
M[0,j] &= M[0,j-1] + 1 + |b_j|, \quad \text{for } 1 \leqslant j \leqslant m \\
M[i,j] &= \min\Big(M[i-1,j] + 1, M[i,j-1] + 1 + |b_j|, M[i-1,j-1] + |b_j - a_i|\Big), \quad \text{for } 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m
\end{aligned}
$$

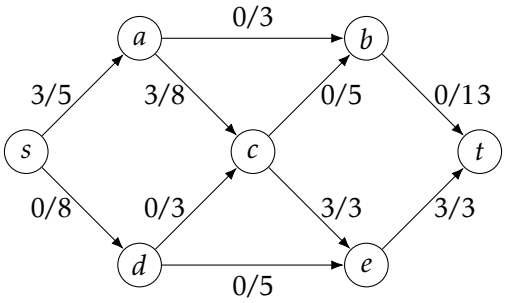
where $M[i-1,j]+1$ is the number of operations if we delete a_i; $M[i,j-1]+1+|b_j|$ is the number of operations if we insert 0 and increase/decrease it until it reaches b_j; $M[i-1,j-1]+|b_j - a_i|$ is the number of operations if we increase/decrease a_i until it reaches b_j (including the case when $a_i = b_j$ and no additional operations are required beyond $M[i-1,j-1]$).

MARKING SCHEME:

- **Recurrence** [4 marks]: appropriate base case(s), correct possibilities included in the recursive case, and correct expression for each possibility
- **Explanation** [1 mark]: reasonable explanation of recursive cases
- **Structure**: 1 mark for any attempt to give a reasonably structured recurrence relation (in case of mostly incorrect answer)

Question 3. [8 MARKS]

Consider the following network N and initial flow f.

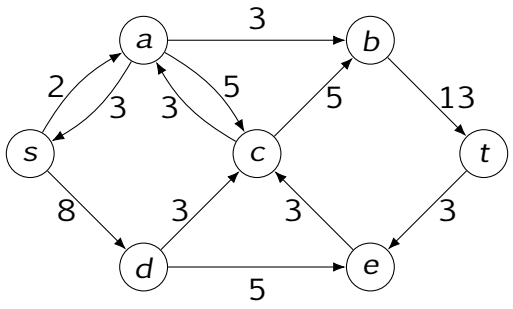


Part (a) [4 MARKS]

Find a maximum flow f^* in network N *using augmenting paths* ***and*** *starting from initial flow* f. Show your work—we want to see *how* you obtain your answer.

SAMPLE SOLUTION:

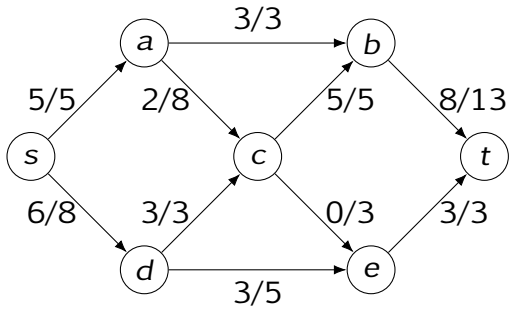
Initial residual network N_f:



Augmenting paths in residual network (with updated residual capacities indicated):

- $s \xrightarrow{2} a \xrightarrow{3} b \xrightarrow{13} t$; residual capacity $\Delta_f(P) = 2$
- $s \xrightarrow{8} d \xrightarrow{3} c \xrightarrow{5} b \xrightarrow{11} t$; residual capacity $\Delta_f(P) = 3$
- $s \xrightarrow{5} d \xrightarrow{5} e \xrightarrow{3} c \xrightarrow{2} b \xrightarrow{8} t$; residual capacity $\Delta_f(P) = 2$
- $s \xrightarrow{3} d \xrightarrow{3} e \xrightarrow{1} c \xrightarrow{4} a \xrightarrow{1} b \xrightarrow{6} t$; residual capacity $\Delta_f(P) = 1$

Final flows:



MARKING SCHEME:

- **Flow** [1 mark]: correct maximum flow
- **Paths** [3 marks]: correct use of augmenting paths—it's okay NOT to draw the residual network; it's okay to specify paths on the original network (with forward and backward edges) instead of using the residual network; other paths than the ones above are possible
- **common error** [−1]: flow is incorrect
- **common error** [−1 to −2]: missed augmenting paths with a backward edge
- **common error** [−1 to −2]: missed obvious augmenting paths that did not require backward edges
- **common error** [−0.5]: did not give the augmenting paths *but* gave a correct residual graph

Part (b) [4 MARKS]

Find a minimum cut $X^* = (V_s, V_t)$ in network N *based on your maximum flow* f^* *from Part (a).* Then, list every edge that crosses your cut X^* and indicate clearly whether it is a *forward* or a *backward* edge. Show your work—we want to see *how* you obtain your answer.

SAMPLE SOLUTION:

Start with $X^* = (\{s\}, \{a, b, c, d, e, t\})$ and follow the algorithm in the proof of the Ford-Fulkerson Theorem.

- Edge (s, d) has unused capacity so move d: $X^* = (\{s, d\}, \{a, b, c, e, t\})$.
- Edge (d, e) has unused capacity so move e: $X^* = (\{s, d, e\}, \{a, b, c, t\})$.
- No more moves possible.

Edges (s, a), (d, c), (e, t) are forward edges; edge (c, e) is a backward edge. X^* is a minimum cut because $c(X^*) = c(s, a) + c(d, c) + c(e, t) = 5 + 3 + 3 = 11 = f(s, a) + f(s, d) = |f|$.

MARKING SCHEME:

- **Cut** [1 mark]: correct minimum cut (relative to flow and residual network from part (a))
- **Work** [2 marks]: correct method for finding minimum cut
- **Edges** [1 mark]: correct list of edges across the cut, with correct directions
- **common error** [−1]: correct justification ("max. flow = min. cut") but no indication of how cut was found
- **common error** [−2]: mixing up capacity of cut and flow across cut
- **common error** [−1]: calculation error
- **common error**: using *all* edges with non-zero residual capacity to generate the cut (instead of only those reachable from s)