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.

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. [9 MARKS]

Consider the following "shipping truck" problem.

Input: A positive integer *weight limit* W and positive integer *package weights* $w_1, w_2, \dots, w_n$ with $w_i \leqslant W$.

Output: A positive integer *truck number* t_i for each package i so that every truck has total weight less than or equal to W, packages are placed in the order given ($1 = t_1 \leqslant t_2 \leqslant \dots \leqslant t_n$), and the total number of trucks used (the value of t_n) is minimum.

For example, with weight limit 8 and package weights $5, 3, 3, 2, 4, 6, 1$, one possible solution assigns truck numbers $1, 2, 2, 2, 3, 4, 4$. There is a fairly obvious greedy algorithm to solve this problem: assign the smallest possible truck number to each package.

```
$t \leftarrow 1$   # current truck number
$w \leftarrow 0$   # weight of current truck
for $i \leftarrow 1, 2, \dots, n$:
    if $w + w_i > W$:   # package $i$ does not fit on current truck
        $t \leftarrow t + 1$   # use one more truck (because it is necessary)
        $w \leftarrow 0$
    $t_i \leftarrow t$   # add package $i$ to current truck
    $w \leftarrow w + w_i$
return $t_1, t_2, \dots, t_n$
```

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

MARKING SCHEME:

- One mark for each definition.

MARKING CODES AND COMMENTS:

- **common error** [−0.5]: defining partial solution in terms of the number of trucks used, rather than the sequence of truck numbers
- **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 $t_1, t_2, \dots, t_i$ is promising."

Part (c) [5 MARKS]

Prove the following *exchange lemma* (that could be used in the proof of your loop invariant).

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

SAMPLE SOLUTION:

Suppose $T^* = t_1^*, t_2^*, \dots, t_n^*$ is an optimum solution (t_n^* is minimum) and T^* extends $T_i = t_1, t_2, \dots, t_i$ and $t_{i+1} \neq t_{i+1}^*$. This means $t_{i+1} < t_{i+1}^*$ because the greedy algorithm never uses a truck number larger than necessary. This implies $i + 1 < n$ because t_n^* is minimum.

Let $T^{**} = t_1^{**}, t_2^{**}, \dots, t_n^{**}$ be defined as follows: $t_j^{**} = t_j^*$ for $j = 1, 2, \dots, i, i+2, \dots, n$ and $t_{i+1}^{**} = t_{i+1}$. Then, T^{**} is a valid solution: the only difference from T^* is that package $i + 1$ has been removed from truck number t_{i+1}^* and placed on truck number t_{i+1}. This cannot cause overflow on truck t_{i+1}^* or on truck t_{i+1} (the truck selected by the algorithm). Moreover, T^{**} is an optimum solution (because $t_n^{**} = t_n^*$ is minimum) and T^{**} extends $T_{i+1} = t_1, t_2, \dots, t_i, t_{i+1}$ (because $t_1^{**} = t_1^* = t_1, \dots, t_i^{**} = t^* i = t_i$ and $t_{i+1}^{**} = t_{i+1}$).

MARKING SCHEME:

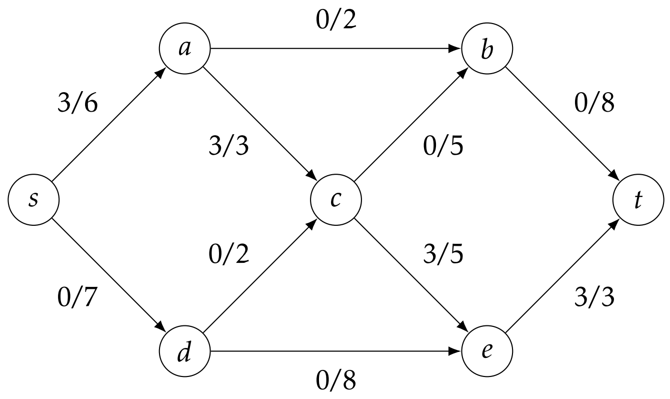
- **Idea** [2 marks]: correct main idea—reassign package $i + 1$ to the truck picked by the algorithm
- **Details** [3 marks]: correct arguments that T^{**} is valid and optimum and extends T_{i+1}

MARKING CODES AND COMMENTS:

- **common error** [−5]: copy-and-paste some other problem's proof
- **common error**: assuming there is only one optimum solution

Question 2. [7 MARKS]

Consider the following network N and initial flow f.

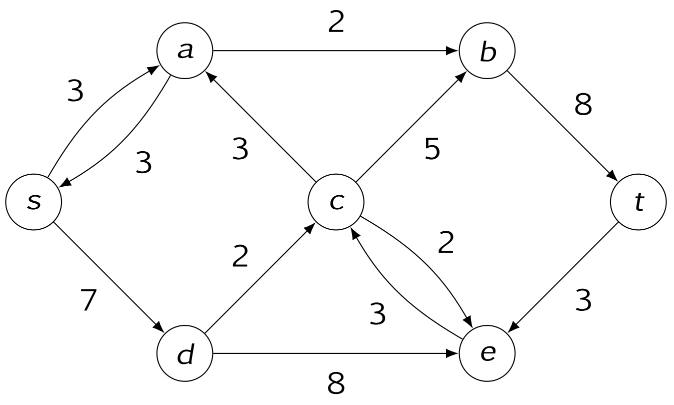


Part (a) [3 MARKS]

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

SAMPLE SOLUTION:

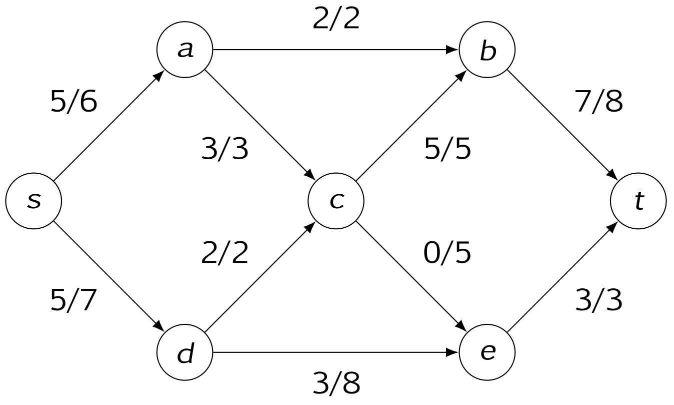
Residual network N_f:



Augmenting paths:

- $s \xrightarrow{7} d \xrightarrow{8} e \xrightarrow{3} c \xrightarrow{5} b \xrightarrow{8} t$; residual capacity $\Delta_f(P) = 3$
- $s \xrightarrow{3} a \xrightarrow{2} b \xrightarrow{5} t$; residual capacity $\Delta_f(P) = 2$
- $s \xrightarrow{4} d \xrightarrow{2} c \xrightarrow{2} b \xrightarrow{3} t$; residual capacity $\Delta_f(P) = 2$

Final flows:



MARKING SCHEME:

- **Flow** [1 mark]: correct maximum flow
- **Paths** [2 marks]: correct use of augmenting paths

MARKING CODES AND COMMENTS:

- **common error** [−1]: flow is incorrect
- **common error** [−0.5]: missed the augmenting path with a backward edge
- **common error** [−1]: missed obvious augmenting paths that did not require backward edges
- **common error** [−0.5]: used a non augmenting path (e.g., tried to reverse the flow of an entire path)
- **common error** [−0.5]: did not give the augmenting paths *but* gave a correct residual graph
- **common error** [−0.5]: did the reverse flow part in multiple steps (instead of just one augmenting path that includes the backward edge)
- **common error** [−0.5]: did not fully utilize augmenting path (e.g., some students would only push 1 flow on a path with residual capacity 2)

Part (b) [4 MARKS]

Find a minimum cut in network N *based on your maximum flow from Part (a)*. Then, list every edge that crosses your cut 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.

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

Edges (a, b), (a, c), (d, c), (e, t) are forward edges; edge (c, e) is a backward edge. X is a minimum cut because $c(X) = c(a, b) + c(a, c) + c(d, c) + c(e, t) = 2 + 3 + 2 + 3 = 10 = f(s, a) + f(s, d) = |f|$.

MARKING SCHEME:

- **Cut** [1 mark]: correct minimum cut
- **Work** [2 marks]: correct method for finding minimum cut
- **Edges** [1 mark]: correct list of edges across the cut

MARKING CODES AND COMMENTS:

- **common error** [−0.5]: missing edges across the cut
- **common error** [−0.5]: 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

Question 3. [8 MARKS]

Let $S = a_1, a_2, \ldots, a_n$ be any sequence of real numbers. A *subsequence* of S is a sequence $a_{i_1}, a_{i_2}, \ldots, a_{i_m}$ for some $m \geqslant 1$ and $1 \leqslant i_1 < i_2 < \cdots < i_m \leqslant n$ (intuitively, a subsequence "picks" some of the numbers, in order, but can also "skip" some numbers). For example, $3, 1, 4$ is a subsequence of $S = 2^1/_9, 3, 1, -2.5, \sqrt{15}, 4, 1.2$ (with $i_1 = 2, i_2 = 3, i_3 = 6$).

The Longest Decreasing Subsequence ("LDS") problem is defined as follows.

Input: A sequence of real numbers $S = a_1, a_2, \ldots, a_n$.

Output: A subsequence $a_{i_1}, a_{i_2}, \ldots, a_{i_m}$ of S such that $a_{i_1} > a_{i_2} > \cdots > a_{i_m}$ (the subsequence is *decreasing*) and m is maximum (the subsequence is as long as possible).

Part (a) [3 MARKS]

Define a *one-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: To make the next part easier, define each sub-problem so that one particular member of the subsequence is known. Note that you have much more space than you need for your final answer.)

SAMPLE SOLUTION:

Define $L[i]$ = length of a LDS of $a_1, a_2, \ldots, a_i$ whose last element is exactly a_i, for $1 \leqslant i \leqslant n$.

MARKING SCHEME:

- **Indices** [1 mark]: correct index range (starting at 0 is OK)
- **Sub-problem** [1 mark]: correct sub-problem (stated explicitly)
- **Value** [1 mark]: correct value
- **error code V_1** [−0.5]: array stores actual LDS for sub-problem instead of only its length
- **common error**: Full marks for solutions without condition that subsequence ends with a_i (next part will be more difficult then).

Part (b) [5 MARKS]

Give a recurrence relation for your array values. Include appropriate base case(s). Justify your answer by discussing the recursive structure of the problem. (You also have more room than necessary for this part.)

SAMPLE SOLUTION:

$$L[i] = \begin{cases} 1 & \text{if } \forall j < i, a_j \leqslant a_i, \\ 1 + \max_{j<i, a_j > a_i} L[j] & \text{if } \exists j < i, a_j > a_i. \end{cases}$$

(A LDS of $a_1, \ldots, a_i$ that ends with a_i consists of a LDS of $a_1, \ldots, a_j$ followed by a_i, for some $j < i$ with $a_j > a_i$.)

MARKING SCHEME:

- **Base cases** [1 mark]: appropriate base case(s)
- **Recursive case** [2 marks]: correct recursive case
- **Justification** [2 marks]: reasonable justification based on recursive structure
- **Structure** [1 mark]: any attempt to give a reasonably structured recurrence relation (in case of mostly incorrect answer)

(Bonus) [3 MARKS]

Explain (in detail) how to use your array values to output a LDS of $a_1, a_2, \ldots, a_n$ as efficiently as possible.

WARNING! This question is long and will be marked *harshly*: credit will be given only for making *significant* progress toward a correct answer. Please attempt this only *after* you have completed the rest of the midterm test.

SAMPLE SOLUTION:

(No solution for bonus problem. If you're curious, ask during office hours or on Piazza!)

MARKING SCHEME:

- **Idea** [1 mark]: correct high-level idea
- **Details** [2 marks]: idea implemented correctly