# Midterm 2

- **The exam has 4 questions, is worth 100 points, and will last 120 minutes.**
- The exam has three parts. In part A solve all questions. In both parts B and C, you'll need to solve exactly one out of two options. For each part, please indicate which option you chose.
- We indicated how points are allocated to different parts of questions. Not all parts of a problem are weighted equally.
- Read the instructions and the questions carefully first.
- Begin each problem on a new page.
- Be precise and concise.
- The questions start with true/false, then short answer, then algorithm design. The problems may **not** necessarily follow the order of increasing difficulty.
- Good luck!

# Part A (55 pts) – Answer all of the following three questions:

## 0 Logistics and Self-Reflect (1 pt)

(0.5 pt) Are you recording the midterm? Did you provide a link so that staff can login to the room if needed?
(0.5 pt) How well did you do in the exam? Please write your estimate of your total score.

## 1 True/False + short justification: MSTs + Max-Flow (24 pts)

(3 pts per item) The first six items involve minimum spanning trees (MSTs). Given a weighted undirected connected graph $G = (V, E)$, are the following **true** or **false**? **If true, provide a brief justification (1-3 sentences); if false, provide a counterexample**. In the following, a **non-trivial cut** would refer to a partition $(S, V \setminus S)$ of the vertices, where both $S$ and $V \setminus S$ are **non-empty**.

(a) For any spanning tree $T$ and any non-trivial cut $(S, V \setminus S)$, $T$ contains *at least* one edge crossing this cut.

(b) For any spanning tree $T$ and any non-trivial cut $(S, V \setminus S)$, $T$ contains *at most* one edge crossing this cut.

(c) For any non-trivial cut $(S, V \setminus S)$, suppose $e$ crosses this cut and has strictly smaller weight than every other edge crossing this cut. Then any minimum spanning tree must contain $e$.

(d) For any non-trivial cut $(S, V \setminus S)$, suppose $e$ crosses this cut and there is another edge with strictly smaller weight than $e$ crossing the cut. Then no minimum spanning tree contains $e$.

(e) For any cycle in the graph, suppose $e$ is in this cycle and has strictly larger weight than every other edge in this cycle. Then no minimum spanning tree contains $e$.

(f) For any two vertices $s, t$, the shortest path from $s$ to $t$ is contained in every minimum spanning tree.

The next couple of items involve Min-Cut-Max-Flow. Again, for each statement, answer whether this statement is true or false, and provide a **short justification/counterexample**. Let $G = (V, E)$ be a network with capacities $c$ and with $s$-$t$ max-flow with value $f$. Let $(S, V \setminus S)$ be a minimal $s$-$t$ cut in the network, and let $e$ be an edge crossing this cut.

(g) If we **increase** the capacity of $e$ by 1 then we **increase** the maximal flow of the network by 1.

(h) If we **decrease** the capacity of $e$ by 1 then we **decrease** the maximal flow of the network by 1.

# 2 Max Flow, Zero-sum Games & Linear Programming (30 pts)

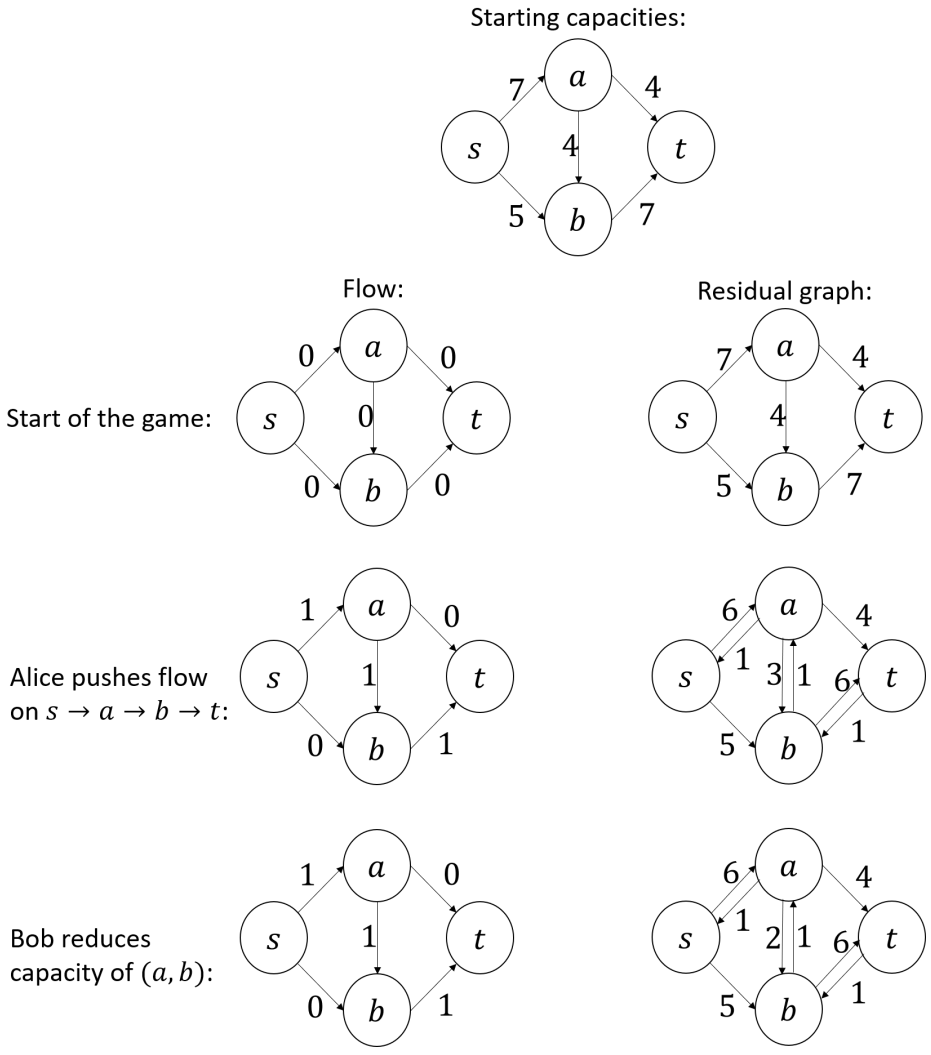
(a) Max Flow (10 points):

Alice and Bob are playing a game. They start with a directed graph $G$ with capacities $c_e$, for which $F$ is the value of the maximum $s$-$t$ flow in $G$. All capacities are integers.

Before the first round of the game, no flow has been pushed. In each round: Alice chooses an $s$-$t$ path in the residual graph, pushes one unit of flow on this path, and updates the residual graph. Then, Bob chooses an edge in the original graph whose capacity is larger than the current flow through it, decreases its capacity by 1, and updates the residual graph.

The game ends when Alice cannot choose a path in the residual graph, or Bob cannot choose an edge that is not at capacity.

For example, Alice and Bob's first turns might look like this:




If Alice plays perfectly, what is the minimum flow she is guaranteed to push? If Bob plays perfectly, what is the maximum flow he can limit Alice to pushing? Justify your answer.

**Note:** If you cannot solve this problem, for **half** of the points solve the following problem: Suppose someone presents you with a solution to a $s$-$t$ max-flow problem on some network. Give a linear time ($O(|V| + |E|)$) algorithm to determine whether the solution does indeed give a maximum flow.

(b) Zero-sum games + LP (20 points):

Alice and Bob are playing a two-person zero-sum game described by the payoff matrix:

$$\begin{bmatrix} +1 & -3 \\ -2 & +4 \end{bmatrix}$$

Here Alice is the row player and Bob the column player, and the entry of the matrix corresponds to the payoff for Alice. For this problem, you may denote Alice and Bob's strategies as $(p, 1-p)$ and $(q, 1-q)$, respectively, where $p$ denotes the probability that Alice plays the first row, etc. You are encouraged to justify your solution to both parts to maximize your chances of getting partial credit.

i) Write down an LP for finding Alice's optimal strategy. Hint: you might find it helpful to first write down the max-min condition satisfied by Alice's optimal strategy.

ii) Write down the dual to the LP from part i).

# Part B (20 pts): Greedy Algorithms – Solve one of the following two problems:

## 3 Greedy Option A: Fractional Knapsack

A thief has broken into a vault and has a knapsack in which to carry the loot. There are $n$ items in the vault, where the $i$-th item weighs $w_i$ and has value $v_i$. Assume that the knapsack can hold a total weight of at most $W$, and that for each $i$ the thief can choose to carry any fraction $0 \leq a_i \leq 1$ of the $i$-th item, in which case it contributes weight $a_i w_i$ and value $a_i v_i$ to the knapsack. Help the thief design an efficient *greedy* algorithm to get away with the largest value in loot. Use an exchange argument to justify the correctness of your algorithm, and bound the running time.

## 3 Greedy Option B: Combining Ropes

You have $n$ ropes with lengths $f_1, f_2 \ldots f_n$. You need to connect all these ropes into one rope. At each step, you may connect two ropes, at the cost of the sum of their lengths. Your total cost is the sum of the costs of all the steps resulting in a single rope.

Give a greedy algorithm that finds the lowest-cost strategy for combining ropes. Prove the optimality of your strategy using the optimality of Huffman coding (alternatively, you can prove optimality directly, but we recommend relying on Huffman's optimality). No runtime analysis needed.

**An example:** if we are given three ropes of lengths 1, 2, 4, then we can connect the three ropes as follows:

- First, connect ropes with length 1 and 2. This costs 3, and leaves you with two ropes of lengths 3, 4.
- Then, connect ropes with length 3 and 4. This costs 7, and leaves you with one rope of length 7.
- The overall cost of the above solution is 3+7 = 10.

# Part C (25 pts): Dynamic Programming – Solve one of the following two problems:

For whichever option you choose in this part, your solution should include the following:

- Identifying the subproblems (Option A only).
- A recurrence relation.
- Base cases for your recurrence relation.
- The order in which to solve the subproblems.
- The final output of the algorithm.
- A **justification** for your recurrence relation (A full proof by induction is not needed).
- Runtime analysis.

## 4 DP Option A: Counting Paths

You are given an unweighted **DAG** (Directed Acyclic Graph) $G$, along with a start node $s$ and a target node $t$. Design a linear time (i.e., runtime $O(|V| + |E|)$) dynamic programming algorithm for computing the number of all paths (not necessarily shortest) from $s$ to $t$.

## 4 DP Option B: All Pairs Shortest Paths

We have a directed weighted graph $G(V, E)$. The weight $w(e)$ of each edge can be any integer, including negative integers, but there are no negative-weight cycles. Our goal is to find the shortest path length between every pair of vertices. One option is to run Bellman-Ford from every vertex. However, this could take $O(|V|^4)$ time if the graph is a complete graph. In this problem we will design a **faster** algorithm.

Denote by $d(u, v, i)$ the length of the shortest $u$-$v$ path using at most $2^i$ edges, for $i = 0, 1, \ldots, \lceil \log |V| \rceil$ (if no path using at most $2^i$ edges exists then $d(u, v, i) = \infty$). Design an efficient dynamic programming algorithm for computing all shortest path lengths between every pair of vertices, using the subproblems $d(u, v, i)$.