# Midterm 2

- **The exam will last 110 minutes.**
- ***The answer boxes are for in-person exams. You do not need to print out the exam and answer in "boxes".*** Make sure you clearly mark your answers.
- The exam has 8 questions with a total of 100 points. You may be eligible to receive partial credit for your proof even if your algorithm is only partially correct or suboptimal.
- Answer all questions. Read them carefully first. Not all parts of a problem are weighted equally.
- Be precise and concise.
- The problems may **not** necessarily follow the order of increasing difficulty.
- The points assigned to each problem are by no means an indication of the problem's difficulty.
- Unless the problem states otherwise, you should assume constant time arithmetic on real numbers.
- If you use any algorithm from lecture and textbook as a blackbox, you can rely on the correctness and time/space complexity of the quoted algorithm. If you modify an algorithm from textbook or lecture, you must explain the modifications precisely and clearly, and if asked for a proof of correctness, give one from scratch or give a modified version of the textbook proof of correctness.
- Good luck!

## 1 Huffman Warmup

(4pts) Given a Huffman encoding of a file $F$, how would you produce a Huffman encoding of the file $F||F$ (the concatenation of the file and itself)? Justify your answer.

There are two ways to interpret the question.
1. How is the encoding of each character in the file affect? The encoding of each character stays the same since the frequencies are the same.
2. How is the encoded version of the file affected? Since the encoding for each character remains the same (as shown above), the encoding of the file will be the original encoding of the file concatenated to itself.
Both answers are equally correct.

## 2 Flow Computation

(8pts) Find the maximum $s$-$t$ flow and minimum $s$-$t$ cut on the graph below. (Fill in the appropriate data in the lines below.)

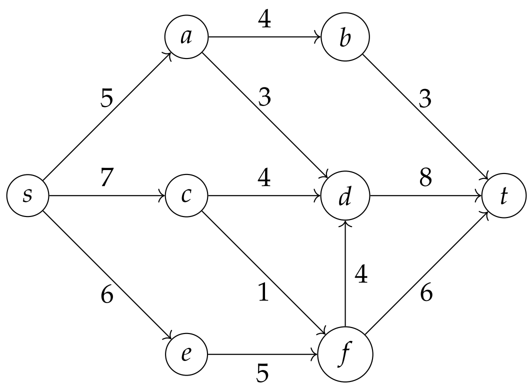


Figure 1: All parts of this problem are with respect to the above graph.

(a) Final max flow and min cut values.

**Max flow value:** $\boxed{\phantom{00}}$ **Min cut value:** $\boxed{\phantom{00}}$

(b) Final flow across each edge (each blank should be a number):

| | |
|---|---|
| $(s,a)$ | |
| $(s,c)$ | |
| $(s,e)$ | |
| $(a,b)$ | |
| $(a,d)$ | |
| $(c,d)$ | |

| | |
|---|---|
| $(c,f)$ | |
| $(e,f)$ | |
| $(b,t)$ | |
| $(d,t)$ | |
| $(f,d)$ | |
| $(f,t)$ | |

(c) Edges that cross the min-cut: $\boxed{\phantom{000000}}$

(a) Max flow = min cut, and both are 15

(b) The following is a max $s$-$t$ flow:

| | |
|---|---|
| $(s,a)$ | 5 |
| $(s,c)$ | 5 |
| $(s,e)$ | 5 |
| $(a,b)$ | 3 |
| $(a,d)$ | 2 |
| $(c,d)$ | 4 |

| | |
|---|---|
| $(c,f)$ | 1 |
| $(e,f)$ | 5 |
| $(b,t)$ | 3 |
| $(d,t)$ | 6 |
| $(f,d)$ | 0 |
| $(f,t)$ | 6 |

(c) $(\mathbf{s},\mathbf{a}),(\mathbf{c},\mathbf{d}),(\mathbf{e},\mathbf{f}),(\mathbf{c},\mathbf{f})$

## 3 Flow Proof

Let $s$, $v$, $t$ be vertices in $G$. Prove that if there's a flow of value[1] $k$ from $s$ to $v$ and a flow of value $k$ from $v$ to $t$ in $G$, then there's a flow of value $k$ from $s$ to $t$ in $G$.

(8pts) Please complete this elegant five-sentence proof.

*Proof.* Suppose, for the sake of contradiction, that ___(1)___.
Then, by the ___(2)___ theorem, we know there exist a $(s,t)$-cut[2] $(S,T)$ of the graph, such that ___(3)___.
Without loss of generality, let $v \in S$.[3]
Since ___(4)___, then by the same theorem, we know that ___(5)___.
Contradiction. □

*Proof.* Suppose, for the sake of contradiction, that $G$ does not have a size-$k$ flow from $s$ to $t$.
By the max-flow min-cut theorem, we know that there exist a $(s,t)$-cut $(S,T)$ of the graph, $s \in S, t \in T$, such that the capacity of this cut is strictly less than $k$.
Without loss of generality, let $v \in S$.
Since we know there is a size-$k$ flow from $v$ to $t$, then by the same theorem, we know that the capacity of the cut $(S,T)$ is at least $k$.
Contradiction. □

## 4 Polytope Park Rangers

(5pts) Consider the following LP.

$$\begin{aligned} &\max 3x+y \\ \text{subject to } & -2x+y \geq -15 \\ & x \geq 2y \\ & y \geq 0 \end{aligned}$$

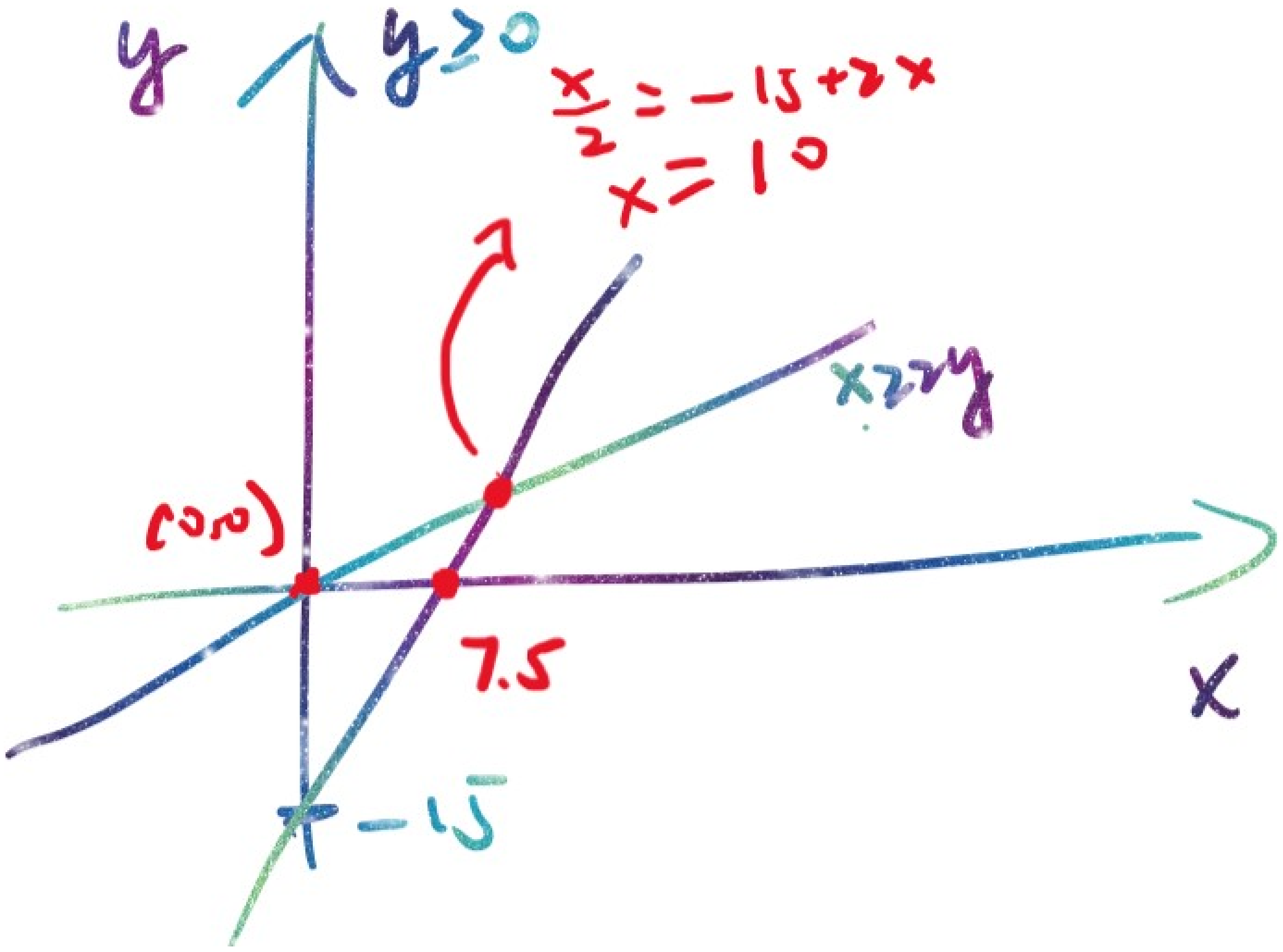
What is the optimal $x, y$ and objective? Answer in the boxes below.

$x^*$: $\boxed{\phantom{xxxx}}$ , $y^*$: $\boxed{\phantom{xxxx}}$

Optimal objective value: $\boxed{\phantom{xxxx}}$

There are 3 vertices on the polytope: $(0,0)$, $(7.5,0)$, $(10,5)$. Objectives are, respectively, $0, 22.5, 35$. Therefore, 35 is optimal for objective and $(x^*,y^*) = (10,5)$.

[1] *i.e.* a size-$k$ flow.
[2] Recall the definition of an $(s,t)$-cut: a partition of the graph into two sets $(S,T)$ such that $s \in S$, $t \in T$.
[3] In other words, a similar argument holds when $v \in T$.



## 5 Memory Usage

(5pts) Tianchen is running a multiplicative weight update algorithm on an old computer and he notices his memory usage is at capacity from having too much data while running his MWU program. Which action(s) will definitely decrease memory usage by keeping track of less data when running the MWU program?
A. Decrease number of timesteps.
B. Decrease number of experts.
C. Increase $\epsilon$.
D. Decrease $\epsilon$.
**Justify your answer.** *i.e.* Explain why the action(s) you chose works and then briefly explain (in one or two sentences) why the other action(s) won't work.

B. It's an online algorithm, and we need to store a constant number of variables per expert. Decreasing the number of experts reduces the number of variables stored.
The number of timesteps only affects the runtime of the algorithm, while $\epsilon$ only affects the calculations of weights.

## 6 Game

(20pts) Consider the following matrix game. A positive payoff goes to the row player. $a, b, c, d > 0$.

$$\begin{bmatrix} a & -b \\ -c & d \end{bmatrix}$$

(a) What is the optimal strategy for the row player? For the column player? Show your work. Remember to justify why your answers are the optimal strategies.

(b) Under what conditions is the game fair; that is, the expected value to the optimal players are 0? Show your work.

(a) For the row player, we can write down a linear program. Let $x_1$ be the probability to play row 1, and $x_2$ for row 2. Then the goal is to maximize $z$ such that $z \leq ax_1 - cx_2$ and $z \leq dx_2 - bx_1$, and furthermore $x_1 + x_2 = 1$ and $x_1, x_2 \geq 0$. We make a guess that at equilibrium, both pure strategies of the opponent are equally good against the row player's strategy (which we verify later). This corresponds to $ax_1 - cx_2 = dx_2 - bx_1$. Also using the fact that $x_1 + x_2 = 1$, we can solve for $x$ to obtain

$$x_1 = \frac{c+d}{a+b+c+d},\ x_2 = \frac{a+b}{a+b+c+d}$$

and for $z = ax_1 - cx_2$, this implies

$$z = \frac{ad - bc}{a+b+c+d}$$

The LP for the column player is to minimize $z$ such that $z \geq ay_1 - by_2$ and $z \geq dy_2 - cy_1$, and furthermore $y_1 + y_2 = 1$ and $y_1, y_2 \geq 0$. Here $y_1$ is the probability of playing column 1, and $y_2$ the probability of playing column 2. A similar solution approach yields

$$y_1 = \frac{b+d}{a+b+c+d},\ y_2 = \frac{a+c}{a+b+c+d}$$

To get full credit, one should justify that these are actually optimal solutions to the LPs. This follows by weak duality, which states that the value of any primal solution is at most the value of any dual solution. Since we found primal and dual solutions of equal value, they must be optimal (the column player's LP is the dual of the row player's).

(b) The game is fair when the optimal $z$ comes out to 0, which means $ad = bc$.

## 7 Nice numbers

(20pts) The problem below is a dynamic programming problem and should be viewed as having 3 parts. You should find a function $f$ which can be computed recursively so that evaluation of $f$ on a certain input (or combining its evaluations on a few inputs) gives the answer to the stated problem.

- Part (1) is to define $f$ *in words* (without mention of how to compute it recursively). You should clearly state how many parameters $f$ has, what those parameters represent, what $f$ evaluated on those parameters represents, and how you should use $f$ to get the answer to the stated problem.
- Part (2) is to give a recurrence relation showing how to compute $f$ recursively, including a description of the base cases.
- In part (3) you should give the running time **and space** for solving the original problem using computation of $f$ via memoization or bottom-up dynamic programming. If you need to use certain data structures, compute in a certain order, preprocess the data in a certain way, *etc.*, to optimize computation of $f$, you should say so.

**Note:** if there are multiple solutions to solve the stated dynamic programming problem, you should describe the most time-efficient one you know. If there are multiple solutions with the same asymptotic time complexity, you should describe the implementation that gives the best asymptotic space complexity.

**The problem:** A number is "nice" if each pair of adjacent digits in its decimal representation differ by at least 2. For example, "13" is nice, but "28549" isn't (the 5 and 4 only differ by one). Given $k$, calculate the number of nice integers $x$ that are $k$ digits long. That is, calculate the number of nice $x$ such that $10^{k-1} \leq x \leq 10^k - 1$. You should output this number modulo $P = 1000003$.

(a) Let $f(d, r)$ be the number of nice $r$ digit numbers mod $P$ that start with the digit $d$ for some $0 \leq d \leq 9$. Then we will want to output $(\sum_{d=1}^{9} f(d,k)) \mod P$.

(b) To compute $f$,

$$f(d,r) = \begin{cases} 1, & \text{if } r = 1 \\ (\sum_{d':|d-d'|\geq 2} f(d', r-1)) \mod P, & \text{if } r > 1 \end{cases}$$

(c) The running time is $O(k)$. The space can be made $O(1)$ by doing bottom up dynamic programming since $f(\cdot, r)$ values only depend on $f(\cdot, r-1)$ values.

## 8 We're Going To Be Friends

(30pts) Jack lives in Yaweno, where all the expressways between cities are one-way (*i.e.* directed edge). There are $n$ cities and $m$ expressways. Jack has $c_i$ friends living in city $i$, $i \in \{1, \dots, n\}$. Jack lives in city $s$. Jack wants to go to city $t$. $s, t \in \{1, \dots, n\}$. Given $n, s, t$ and the $m$ expressways between the $n$ cities, Jack would like to calculate the maximum number of friends he can visit driving from $s$ to $t$ using only expressways.

(a) **For this subpart only**, suppose the expressways of Yaweno **do not form any cycle** (*i.e.* they form a directed acyclic graph). Give a **three-part dynamic programming solution**. The template is copied below for your convenience.

This problem is a dynamic programming problem and should be viewed as having 3 parts. You should find a function $f$ which can be computed recursively so that evaluation of $f$ on a certain input (or combining its evaluations on a few inputs) gives the answer to the stated problem.

- Part (1) is to define $f$ *in words* (without mention of how to compute it recursively). You should clearly state how many parameters $f$ has, what those parameters represent, what $f$ evaluated on those parameters represents, and how you should use $f$ to get the answer to the stated problem.
- Part (2) is to give a recurrence relation showing how to compute $f$ recursively, including a description of the base cases.
- In part (3) you should give the running time **and space** for solving the original problem using computation of $f$ via memoization or bottom-up dynamic programming. If you need to use certain data structures, compute in a certain order, preprocess the data in a certain way, *etc.*, to optimize computation of $f$, you should say so.

**Note:** if there are multiple solutions to solve the stated dynamic programming problem, you should describe the most time-efficient one you know. If there are multiple solutions with the same asymptotic time complexity, you should describe the implementation that gives the best asymptotic space complexity.

(b) Suppose you have access to a correct and time-optimal implementation of the special-case version of the problem in part (a) that runs in $O(f(n, m))$. Design an efficient algorithm that solves the general-case version. Describe your algorithm, justify its correctness, and then analyze its runtime in terms of $n$, $m$, and $f(n, m)$.

(a) (1) Let $f(v)$ be the maximum number of friends Jack can visit starting from $s$ and ending in $v$. The answer is then $f(t)$. Alternatively, you could define $f(v)$ to be the maximum number of friends Jack can visit starting from $v$ and ending at $t$, and return $f(s)$.

(2)
- Base Case: $f(s) = c_s$, $f(v) = -\infty \ \forall v \neq s$.
- Recurrence relation:

$$f(v) = \max_{(u,v) \in E} f(u) + c_v$$

.

(3)
- Time Complexity: Making a copy of the reversed edges takes $O(m + n)$ time (assuming an adjacency list representation). Since we reversed all the edges of the graph, each vertex does $O(\deg^- v)$ work to find all the incoming edges, where $\deg^- v$ denotes the indegree of $v$. Therefore the total runtime is $O(m + n)$. **Note:** Even though each node has an indegree of $O(n)$, a bound of $O(n^2)$ is too loose by the amortized analysis above.
- Space Complexity: Assuming a graph representation is given to us already, space complexity is $O(n)$ because we need to store $f(v)$ for all $v \in V$.
- Subproblem ordering: Traverse the vertices in topological order. **Note:** This level of detail was not necessary for points, but in order to implement this, first topological sort the vertices with a DFS. Next, either make a copy of the reversed edges of the graph to get the incoming edges of each vertex, or you can use the outgoing edges by iteratively updating $f(u) = max(f(u), f(v) + c_u)$ for each $(v, u) \in E$ . Then we can compute $f(\cdot)$ in topological order.

(b) **Algorithm Description**: Decompose the graph into a directed acyclic graph of strongly connected components. Create an instance of part (a) that resembles this DAG of SCCs - *i.e.* every city correspond with a SCC; the number of friends in a city in this new graph equals the total number of friends in every city inside the corresponding SCC. Run the algorithm on this instance.

**Proof of correctness**: Once we have reached a node in a SCC, we can visit every other node in the SCC and come back to the node from which we started (informally, we visited all other vertices in the SCC *for free*, visiting all friends in that SCC). Therefore, maximizing the total number of friends we visit on a path is equivalent to maximizing the total value of all SCCs we visit (where the value of a SCC = the total number of friends in cities inside the SCC). Thus, this problem has been reduced to an instance of part a.

**Runtime Analysis**: Decomposing the graph takes $O(n+m)$. Total runtime is $O(n+m+f(n,m))$.