U.C. Berkeley — CS170 : Algorithms
Lecturers: P. Raghavendra and S. Rao

Midterm II
November 7, 2019

Midterm II

Name: Targaryen

SID: 0123456789

Name and SID of student to your left: Lannister

Name and SID of student to your right: Stark

Exam Room:

☐ Dwinelle 155	☐ Evans 10	☐ HFAX A1	☐ Dwinelle 145	☐ GPB 100	☐ Latimer 120	☐ VLSB 2040
☐ Mulford 159	☐ Kroeber 160	☐ Evans 60	☐ Soda 306	☐ Soda 320	☐ Cory 367	☐ Other

Please color the checkbox completely. Do not just tick or cross the box.

Rules and Guidelines

- **The exam is out of 135 points and will last 110 minutes.**
- Answer all questions. Read them carefully first. Not all parts of a problem are weighted equally.
- Write your student ID number in the indicated area on each page.
- Be precise and concise. **Write in the solution box provided.** You may use the blank page on the back for scratch work, but it will not be graded. Box numerical final answers.
- The problems may **not** necessarily follow the order of increasing difficulty. ***Avoid getting stuck on a problem.***
- Any algorithm covered in lecture can be used as a blackbox. Algorithms from homework need to be accompanied by a proof or justification as specified in the problem.
- Good luck!

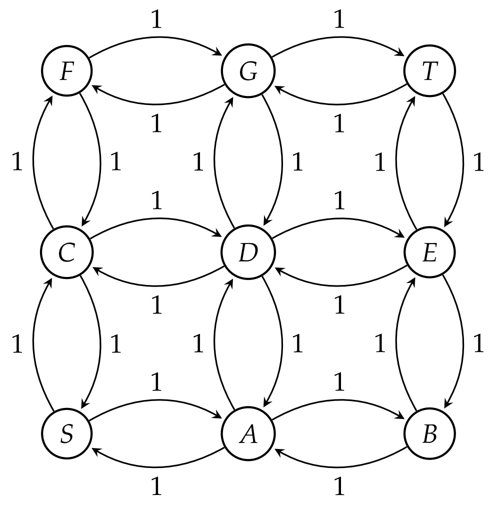
Discussion Section

Which of these do you consider to be your primary discussion section(s)? Feel free to choose multiple, or to select the last option if you do not attend a section. **Please color the checkbox completely. Do not just tick or cross the boxes.**

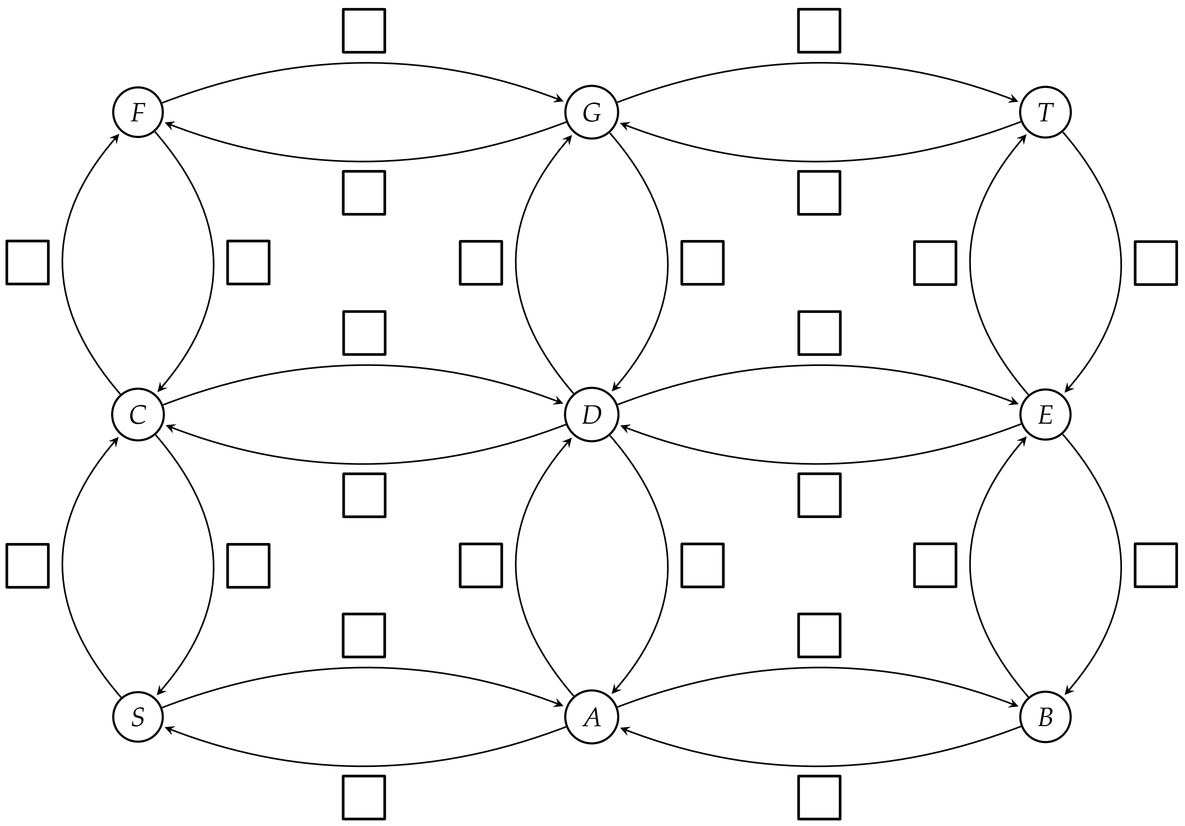
☐ Arpita, Thursday 9 - 10 am, Dwinelle 223	☐ Avni, Thursday 10 - 11 am, Dwinelle 215
☐ Emaan, Thursday 10 am - 11 pm, Etcheverry 3107	☐ Lynn, Thursday 11 am - 12 pm, Barrows 118
☐ Dee, Thursday 11 am - 12 pm, Wheeler 30	☐ Max, Thursday 12 - 1 pm, Wheeler 220
☐ Sean, Thursday 1 - 2 pm, Etcheverry 3105	☐ Neha, Thursday 2 - 3 pm, Dwinelle 242
☐ Julia, Thursday 2 - 3 pm, Etcheverry 3105	☐ Henry, Thursday 3 - 4 pm, Haviland 12
☐ Kedar, Thursday 3 - 4 pm, Barrows 104	☐ Ajay, Thursday 4-5 pm, Barrows 136
☐ Varun, Thursday 4-5 pm, Dwinelle 242	☐ Gillian, Friday 9 - 10 am, Dwinelle 79
☐ Hermish, Friday 10 - 11 am, Evans 9	☐ Vishnu, Friday 11 am - 12 pm, Wheeler 222
☐ Carlo, Friday 11 am - 12 pm, Dwinelle 109	☐ Tarun, Friday 12 - 1 pm, Hildebrand B56
☐ Noah, Friday 12 - 1 pm, Hildebrand B51	☐ Jiazheng, Friday 1 - 2 pm, Wheeler 30
☐ Claire, Friday 2 - 3 pm, Barrows 155	☐ Jialin, Friday 1 - 2 pm, Wheeler 30
☐ Teddy, Friday 1 - 2 pm, Dwinelle 105	☐ Jierui, Friday 2 - 3 pm, Wheeler 202
☐ David, Friday 2 - 3 pm, Wheeler 130	☐ Nate, Friday 2 - 3 pm, Evans 9
☐ Rishi, Friday 3 - 4 pm, Dwinelle 243	☐ Tiffany, Friday 3 - 4 pm, Barrows 104
☐ Ida, Friday 3-4 pm, Dwinelle 109	☐ Don't attend section.

1 Max Flow

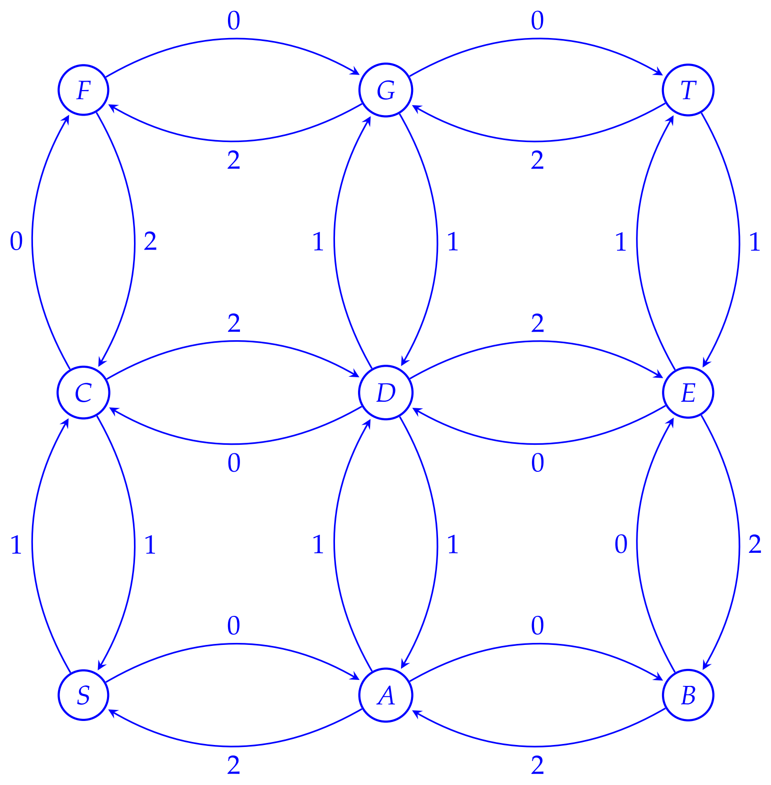
Recall that the Ford-Fulkerson algorithm iteratively uses the residual network to compute the max flow. Consider the following network:



(i) (**4 points**) Let f be a flow that assigns 1 unit of flow on the path $S \to A \to B \to E \to D \to C \to F \to G \to T$ and 0 flow elsewhere. Write down the residual capacities of the edges after routing the flow f.



Solution:



(ii) (**2 points**) What path does the algorithm use in the next iteration?

Solution: $S \to C \to D \to E \to T$

(iii) (**3 points**) Write down the total incoming flow to all these vertices in the graph after these two iterations.

Node	Total Incoming Flow
A	
B	
C	
D	
E	
F	
G	

Solution:

Node	Total Incoming Flow
A	1
B	1
C	1
D	0
E	1
F	1
G	1

(iv) (**1 point**) What is the value of the maximum flow on this network?

Solution: 2

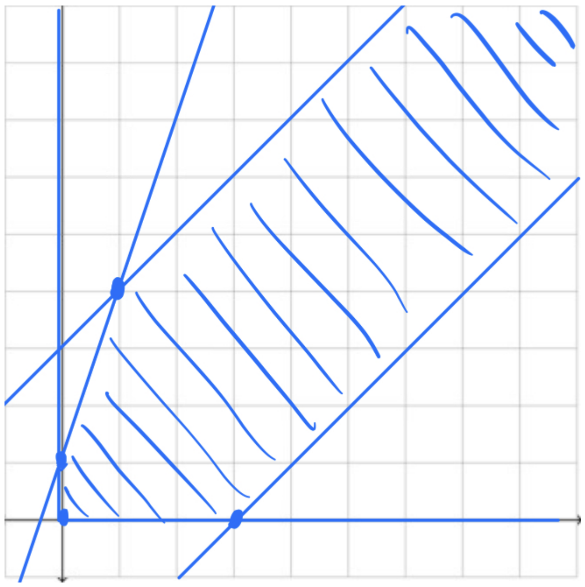
2 Linear Programming

Consider the following linear program.

$$\begin{aligned} &\text{Maximize } x + 2y \\ &\text{Subject to } -3x + y \leq 1 \\ &y - x \leq 3 \\ &y - x \geq -3 \\ &x, y \geq 0 \end{aligned}$$

(i) (**6 points**) Draw out the feasible region for the LP.

Solution:



(ii) (**3 points**) Write the dual linear program (use variables a, b, c).

Solution:

$$\text{Minimize } a + 3b + 3c \tag{1}$$
$$\text{Subject to} \tag{2}$$
$$-3a - b + c \geq 1 \tag{3}$$
$$a + b - c \geq 2 \tag{4}$$
$$a, b, c \geq 0 \tag{5}$$

(iii) (**2 points**) The number of vertices of the feasible region of the dual linear program is

Solution: 0 (The dual is infeasible)

3 Dynamic Programming: Alternate Subproblems

Choosing the right sub-problems is critical in the design of dynamic programming algorithms. Here we will consider alternate subproblems for DP algorithms we studied in the class.

In each case, specify whether it is possible to write a recurrence relation for the subproblem defined. If Yes, then write the correct recurrence relation (Base case not needed). If you believe that it is not possible, select No (no justification needed).

You may not use any external datastructures/alternate subproblems, only the ones provided.

The scoring on the question is as follows:

- Correctly answer Yes, and the correct recurrence relation: 5 points (partial credit for recurrence relation, but no credit for just "Yes").
- Correctly answer No: 3 points
- Incorrectly answer No: -3 points
- Leave it blank: 0 points

(i) **Longest increasing subsequence in the array** $a[1], \ldots, a[n]$
Subproblem:

$$LIS[i] = \text{ length of longest increasing subsequence contained within } a[1], \ldots, a[i],$$

for all $i \in \{1, \ldots, n\}$

○ Yes, it is possible ○ No, there is no recurrence relation

Solution: Not Possible. You need to know where the subsequence ends, otherwise you don't know if you can include the current element. In particular, you don't know if adding the current element keeps the increasing property of the subsequence.

(ii) **Knapsack with repetition with** n **items with values** v_i **and weight** w_i**.**
Subproblem:

$$K[v] = \text{ weight of the lightest collection with total value at least } v \text{ (repetitions allowed)}$$

for all $v \in \{1, \ldots, V\}$

◯ Yes, it is possible ◯ No, there is no recurrence relation

Solution: Possible. The idea is to consider all elements and find the lightest collection of other elements that, upon adding a new element, will reach the value v. More formally:

$$K[v] = \min_i (K[v - v_i] + w_i)$$

(iii) **Edit Distance between strings $x[1, \ldots, n]$ and $y[1, \ldots, m]$.**
Subproblems:

$$ED[i, j] = \text{edit distance between the prefix } x[1, \ldots, i] \text{ and suffix } y[j \ldots m]$$

for all $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$

◯ Yes, it is possible ◯ No, there is no recurrence relation

Solution: Not Possible. The edit distance can be defined using $x[1, \ldots, i]$ and $y[1, \ldots, j]$, as $x[i]$ needs to be edited into $y[j]$. There is no analogous way to define the relation in this case.

(iv) **Knapsack without repetition with n items with values v_i and weight w_i.**
Subproblems:

$$\begin{aligned} K[w, i, 1] &= \text{ maximum value of a collection of items with total weight } w \text{ that contains item } i \\ K[w, i, 0] &= \text{ maximum value of a collection of items with total weight } w \text{ that does not contain item } i \end{aligned}$$

for all $w \in \{1, \ldots, W\}$ and $i \in \{1, \ldots, n\}$

◯ Yes, it is possible ◯ No, there is no recurrence relation

Solution: Not Possible. You need more information about the contained items in the subproblems in order to compute $K[w, i, \star]$, as the knapsack is without repetition. This is why the knapsack problem uses the first i items rather than inclusion/exclusion of a particular item.

(v) **All-pairs shortest paths in a graph $G = (V, E)$ with edge weights $\{w_e\}_{e \in E}$.**

Subproblem:

$d[i, j, k] =$ length of shortest path from i to j among those that do not pass through $\{1, \ldots, k\}$ in the intermediate steps.

for all $i, j, k \in \{1, \ldots, n\}$.

○ Yes, it is possible ○ No, there is no recurrence relation

Solution: Possible.

One way to think about it is $d[i, j, k]$ is also the length of the shortest path from i to j that only passes through $\{k+1, \ldots, n\}$. There can be two cases: the path either uses $k+1$ or not.

$$d[i, j, k] = \min(d[i, j, k+1], d[i, k+1, k+1] + d[k+1, j, k+1])$$

4 LP again

(i) (**3 points**) How many **vertices** does the following feasible region have?

$$\begin{aligned}-1 &\le x_1 \le 1\\ -1 &\le x_2 \le 1\\ -1 &\le x_3 \le 1\\ -1 &\le x_4 \le 1\\ -1 &\le x_5 \le 1\end{aligned}$$

Solution: Each vertex has 5 tight constraints among the 10 in the program. $x_i \le 1$ and $x_i \ge -1$ cannot both be tight. So one can pick $x_i \le 1$ or $x_i \ge -1$ for each i, among the 5 tight constraints, which is two choices for each $i = 1, \ldots, 5$. Therefore there are $2^5 = 32$ vertices in all, given explicitly by $(x_1, \ldots, x_5) = (\pm 1, \pm 1, \pm 1, \pm 1, \pm 1)$

(ii) (**5 points**) Consider the following linear program:

$$\begin{aligned}x + 2y + 3z &\le 6 && (6)\\ 2x + 5y + 7z &\le 14 && (7)\\ 3x + y + 5z &\le 9 && (8)\\ x + 4y &\le 12 && (9)\\ x &\ge 0 && (10)\\ y &\ge 0 && (11)\\ z &\ge 0 && (12)\end{aligned}$$

Two of these points are NOT vertices of the feasible region, mark them.

Solution: This is in ^{3}, which means at least 3 constraints need to be tight at a point for it to be a vertex.

(a) $(3, 0, 0)$

◯ Not a vertex

Solution: Constraints 3, 6, 7 are tight

(b) $(1, 1, 1)$.

◯ Not a vertex

Solution: Constraints 1,2,3 are tight

(c) $(0, 2, 0)$

◯ Not a vertex

Solution: Only constraints 5,7 are tight. Not a vertex

(d) $(2, 2, 0)$

◯ Not a vertex

Solution: Constraints 1,2,7 are tight

(e) $(0,0,1)$

◯ Not a vertex

Solution: Only constraints 5,6 are tight. Not a vertex

Solution: Since the *LP* is 3 dimensional, a vertex is a point that is tight (that is, equal to, not greater than) 3 of the constraints. From the observations above, we see that (a), (b), and (d) are vertices whereas (c) and (e) are not,

(iii) (**2 points**) Suppose $(x,y) = (3,-1)$ and $(x,y) = (-1,3)$ are both feasible solutions to some linear program. Write down another point that is definitely a feasible solution to the same linear program.

Solution: Any point on the line segment joining $(3,-1)$ and $(-1,3)$, for example the midpoint $(1,1)$.

(iv) (**2 points**) Consider primal(left) and dual(right) LP:

$$\begin{array}{cc} \max \mathbf{c}^T\mathbf{x} & \min \mathbf{y}^T\mathbf{b} \\ \mathbf{A}\mathbf{x} \le \mathbf{b} & \mathbf{y}^T\mathbf{A} \ge \mathbf{c}^T \\ \mathbf{x} \ge 0 & \mathbf{y} \ge 0 \end{array}$$

(a) If we change the primal objective to $\max 2c^T x$, what is the new dual objective function?

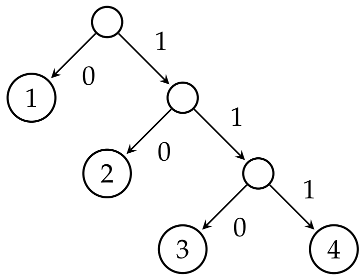
Solution: unchanged

(b) Let D^* be the value of the optimal solution to the dual. If we change the primal objective to $\max 2c^T x$, what would value of the optimal solution to the new dual be (in terms of D^*)?

Solution: twice as original

5 Greedy

(i) (**5 points**) Consider the following Huffman tree (denote T) shown below:



Let $\mathcal{F}$ denote the set of all frequency vectors $f = (f_1, f_2, f_3, f_4)$ for which the above tree is the optimal Huffman tree, i.e., Formally,

$$\mathcal{F} = \{(f_1, f_2, f_3, f_4) \in \mathbb{R}^4 | T \text{ is an optimal Huffman tree for frequencies } (f_1, f_2, f_3, f_4)\} \quad (13)$$

Write an LP whose feasible region is $\mathcal{F}$.

Solution: The huffman algorithm repeatedly merges the two smallest frequencies. The constraints come to 'force' the nodes to be merged in the order given in the tree above (first merge 3 and 4, then merge 2 with that node, then merge 1 with that node).

$$\begin{aligned} f_3 &\leq f_2 \\ f_4 &\leq f_2 \\ f_2 &\leq f_1 \\ f_3 + f_4 &\leq f_1 \\ f_1 + f_2 + f_3 + f_4 &= 1 \\ f_1, f_2, f_3, f_4 &\geq 0 \end{aligned}$$

(ii) (**4 points**) The greedy algorithm for Minimum Set Cover is executed on an input consisting of a family of subsets $\{S_1, \ldots, S_m\}$ where each $S_i \subseteq \{1, \ldots, 1000\}$.

(a) In the first iteration, the greedy algorithm picks a set of size 200 (thereby covering 200 elements). What is the smallest possible value of the optimal solution?

Solution: In any iteration, if there are N elements still to cover, optimal solution covers all of them with OPT sets. So the greedy algorithm always picks a set covering at least N/OPT new elements. In this case $N = 1000$, so $200 \geq 1000/OPT$ which implies that $OPT \geq 5$.

(b) In the second iteration, the greedy algorithm picks a set of size 100 that covers 10 additional elements. What is the smallest possible value of the optimal solution?

Solution: In any iteration, if there are N elements still to cover, optimal solution covers all of them with OPT sets. So the greedy algorithm always picks a set covering at least N/OPT new elements. In this case $N = 1000 - 200 = 800$, so $10 \geq 800/OPT$ which implies that $OPT \geq 80$.

6 Zero Sum Games (10 points)

Consider a zero-sum game given by a $n \times n$ payoff matrix P. Let $\mathcal{R}_{pure}$ and $\mathcal{R}_{mixed}$ denote the sets of pure and mixed strategies for the row-player. Similarly, let $\mathcal{C}_{pure}$ and $\mathcal{C}_{mixed}$ denote the set of pure and mixed strategies for the column player.

For a row-player strategy r and a column player strategy c, let $\mathsf{P}(r, c)$ denote the payoff of the row player when the players use strategies r and c. Depending on which player goes first and what strategies they can use, one can define several different quantities associated with the game. In each of the following cases, write down whether the LHS is $\geq$, $=$, $\leq$ than the RHS. (Write the strongest identity that is true)

(i)

$$\min_{c \in \mathcal{C}_{pure}} \left(\max_{r \in \mathcal{R}_{pure}} \mathsf{P}(r, c) \right) \qquad \boxed{} \qquad \max_{r \in \mathcal{R}_{pure}} \left(\min_{c \in \mathcal{C}_{pure}} \mathsf{P}(r, c) \right)$$

Solution: $\geq$. It is better for the row player if they go second with knowledge of the column player's strategy, which happens in the LHS. So the LHS is $\geq$ than the RHS. This is not equality because strong duality holds only for mixed strategies.

(ii)

$$\min_{c \in \mathcal{C}_{mixed}} \left(\max_{r \in \mathcal{R}_{mixed}} \mathsf{P}(r, c) \right) \qquad \boxed{} \qquad \max_{r \in \mathcal{R}_{mixed}} \left(\min_{c \in \mathcal{C}_{mixed}} \mathsf{P}(r, c) \right)$$

Solution: $=$. This is exactly strong duality of mixed strategies for LPs.

(iii)

$$\min_{c \in \mathcal{C}_{pure}} \left(\max_{r \in \mathcal{R}_{mixed}} \mathsf{P}(r,c) \right) \qquad \boxed{} \qquad \min_{c \in \mathcal{C}_{mixed}} \left(\max_{r \in \mathcal{R}_{mixed}} \mathsf{P}(r,c) \right)$$

Solution: $\geq$. It can be better for the column player if they are able to used a mixed strategy when they go first, because mixed strategies include pure strategies, in addition to many non-pure strategies. Any zero-sum game where the optimal strategy for the column player is not pure is an example of this.

(iv)

$$\min_{c \in \mathcal{C}_{pure}} \left(\max_{r \in \mathcal{R}_{mixed}} \mathsf{P}(r,c) \right) \qquad \boxed{} \qquad \min_{c \in \mathcal{C}_{pure}} \left(\max_{r \in \mathcal{R}_{pure}} \mathsf{P}(r,c) \right)$$

Solution: $=$. In both cases the row player goes second. If the row player knows the column player's mixed strategies, they should just compare the expected payoffs for each of their actions and take the highest one. So it is no better if the row player can use a mixed strategy.

7 Multiplicative Weights (True/False) (8 points)

Let $w_1^{(t)}, \dots, w_n^{(t)}$ be the weights of the multiplicative weights update algorithm after t steps. If $w_i^{(t)} > w_j^{(t)}$ then which of the following statements is true.

(i) Expert i had a smaller total loss than expert j on the first t steps.

○ True ○ False

Solution: True. The weight of an expert at a given timestep is given by $(1-\epsilon)$ raised to the their total loss up until that timestep. Thus, if $w_i^{(t)} > w_j^{(t)}$ then expert j has incurred a higher total loss.

(ii) Expert i is more likely to have a smaller loss than Expert j in the step $(t+1)$.

○ True ○ False

Solution: False. The loss assigned to an expert on a given round can be completely arbitrary and independent of losses assigned in previous rounds.

(iii) Expert i is more likely to have a smaller loss than Expert j in the step $(t+1)$, only if the step t is sufficiently large.

○ True ○ False

Solution: False. For the same reason as (ii): losses can be arbitrary, and long-term behavior has no impact.

(iv) Multiplicative weights algorithm is more likely to pick expert i than expert j in the step $t+1$.

○ True ○ False

Solution: True. The probability of picking an expert on a given round is directly proportional to that expert's weight at that round.

8 True/False (10 points)

(i) Permuting the order of the clauses in a Horn-SAT formula can change the solution produced by the greedy algorithm for it.

◯ True ◯ False

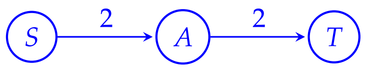
Solution: False. The greedy Horn-SAT algorithm sets the minimum number of variables to be true. Furthermore, the variables set to be true in a certain execution of the algorithm must be true in any satisfying assignment, so permuting the order of the clauses will still give the same minimal set of true literals.

(ii) Suppose that decreasing the capacity of edge $u \to v$ decreases the maximum flow from s to t in a network G. Then u and v must be on different sides of every minimum cut of the graph G.

◯ True ◯ False

Solution: False.

Consider the following graph:



The edge $A \to T$ is not part of the minimum cut consisting of $S \to A$ as the crossing edge. However, decreasing the capacity of $A \to T$ by 1 decreases the max flow from 2 to 1. Thus this graph is a counterexample.

(iii) Suppose that increasing the capacity of edge $u \to v$ increases the maximum flow from s to t in a network G. Then u and v must be different sides of every minimum cut of the graph.

◯ True ◯ False

Solution: True. Assume towards a contradiction that there was some minimum cut across which $u \to v$ was not a crossing edge. Then increasing $u \to v$ does not increase the minimum cut of the graph. However, the problem says that it increases the max flow. Since the max flow and the min cut are equal, this is a contradiction. Therefore, $u \to v$ crosses every minimum cut.

(iv) Simplex runs in polynomial time $O(n^c)$ for some fixed $c \in \mathbb{N}$, over all inputs of size n.

◯ True ◯ False

Solution: False. Although simplex is known to be empirically fast, the worst-case runtime is exponential in the size of the LP.

(v) Suppose that $S_1, S_2 \subset \mathbb{R}^n$ denote the feasible regions of two linear programs LP_1 and LP_2, then $S_1 \setminus S_2$ is also a feasible region of a different LP. (Recall that for two sets A and B, $A \setminus B$ denotes those elements that are in A but not in B).

◯ True ◯ False

Solution: False. The feasible region of an LP must be a polytope of some kind. One can imagine a counterexample where S_1 is a rectangle and S_2 is a rectangle contained within S_1. The resulting region $S_1 \setminus S_2$ has a rectangular hole in it and thus is not a polytope.

9 Fault-Tolerant Road Network (16 points)

There are n cities in the country Pessimia and you are given positive distances $\{d_{ij}\}_{i,j \in \{1,\dots,n\}}$ between them.

The government of Pessimia would like to build a road network $G = (\{1, \dots, n\}, E)$ between the cities. The government would like to build as many roads between cities as possible. But the road network G needs to be fault-tolerant in the following sense:

If any road $(i, j) \in E$ is deleted from G, then the network G still contains an alternate path from i to j of length at most $2d_{ij}$.

1. Describe an algorithm to find the fault-tolerant road network $G = (V, E)$ with the maximum number of roads. Your algorithm can use shortest path algorithms as a black box, and would need to run in time at most $O(|V|^6 + |E|^3)$.

   (Hint: Greedy)

2. Briefly justify the correctness of your algorithm.

Solution:

Algorithm: Begin with the complete graph. Then repeatedly remove any edge $e = (i, j)$, d_{ij}, for which there is no alternate path in the remaining graph with length at most $2d_{ij}$. Note that this algorithm may check an edge over and over.

The time to compute the shortest alternate path for a single edge is at most $|E| + |V| \log |V|$ (run Dijkstra's algorithm), and to do this for all edges is at most $|E|^2 + |E| \cdot |V| \log |V|$. Each iteration removes at least one edge (otherwise the process terminates), giving an upper bound on runtime of $|E|^3 + |E|^2 \cdot |V| \log |V| = \mathcal{O}(|V|^6 + |E|^3)$. Also note this argument works if the runtime for Dijkstra's is $\mathcal{O}(|V|^2)$, as then this is done $E = \mathcal{O}(|V|^2)$ times per iteration for a total of $E = \mathcal{O}(|V|^2)$ iterations. Dijkstra does $\mathcal{O}(|V|)$ remove-min and $\mathcal{O}(|V|^2)$ decrease-key, which can be implemented in $\mathcal{O}(|V|)$ and $\mathcal{O}(1)$ time respectively by using an unsorted array.

Let's say an edge is 'not fault-tolerant' if removing it increases the distance from i to j to more than $2d_{ij}$. For proving correctness, there is an important property: the distances can only get larger the more edges are removed, which means if e is not fault tolerant any $G = (\{1, \ldots, n\}, E)$, it cannot be fault tolerant in any subgraph $G' = (\{1, \ldots, n\}, E')$ where $E' \subseteq G'$.

Informally, this means the algorithm 'only removes what it needs to, but no more' (much like Horn-SAT only sets true what it needs to). Answers that argued something like this were given full credit.

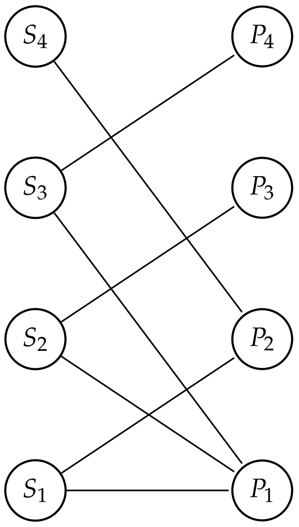
Formally, we can show by induction that for any subgraph G, the algorithm returns the largest fault-tolerant road network using only edges from E. The algorithm will remove any edges that are not fault-tolerant, call these edges F. Then the algorithm will return a fault-tolerant road network using edges from $E \setminus F$ (which is the largest possible by induction), which is by necessity also the largest fault-tolerant road network using only edges from E.

10 Matching Pets (12 points)

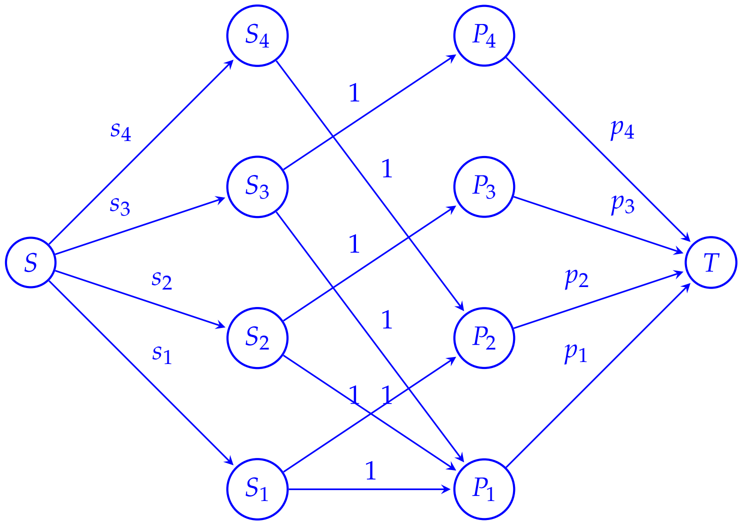
An animal shelter has profiles of students and pets in the shelter, as well as a list of compatible pairs among students and pets. The shelter is trying to setup meetings between students and the pets in the shelter.

1. (**4 points**) A compatibility graph between 4 students and 4 pets is shown below. Here are the constraints on the meetings:
   - The i^{th} student S_i has indicated a preference of meeting exactly s_i pets.
   - The j^{th} pet P_j can meet at most p_j students.
   - All meetings must be between compatible pairs.

   You would like to use one Max-Flow computation to decide whether one can setup the meetings so that the preferences of all students and pets are satisfied. Starting with the compatibilities provided, draw the network on which one would run the Max-flow algorithm. Indicate the source of the flow by S, the sink by T and mark the capacities of all edges.



Solution:

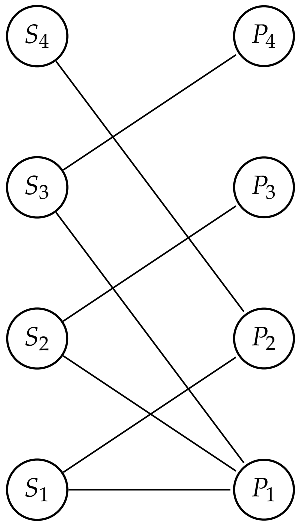


This works very similarly to matching, although it constrains each of the students to be matched with at most s_i pets, and similarly with the pets. There exists a matching that satisfies the constraints iff all the edges leaving s are saturated by the max flow (so each student meets with exactly the number of pets desired).

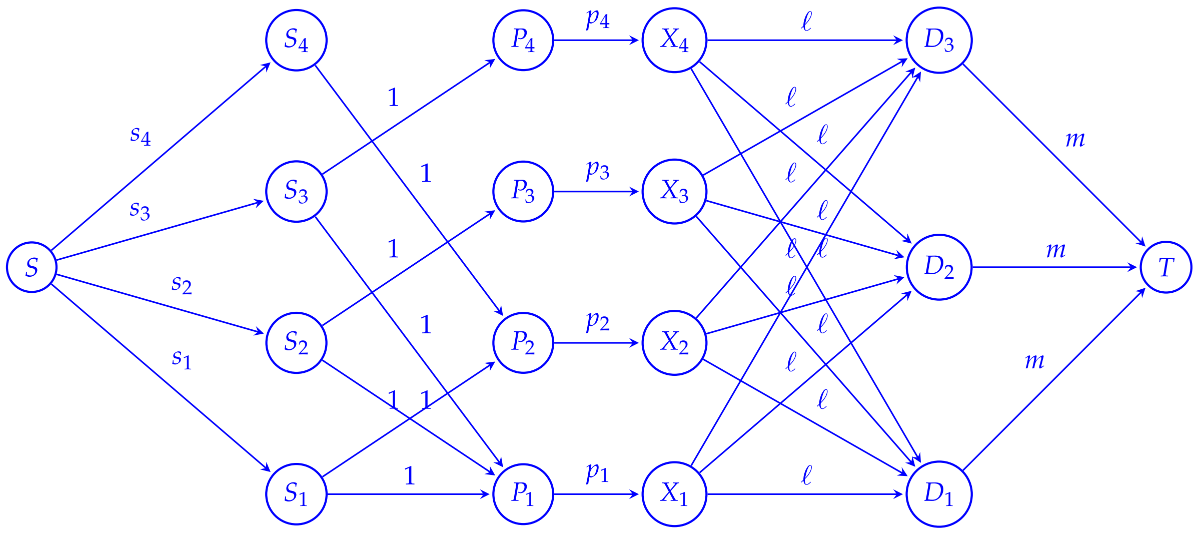
2. (**8 points**) The compatibility graph between 4 students and pets is shown below. Here are the constraints on the meetings:

   - The meetings need to be scheduled over 3 days.
   - The animal shelter cannot host more than m meetings in total on the same day.
   - No pets wants to meet more than ℓ students on the same day, but the students don't mind meeting any number of pets on the same day.
   - The i^{th} student S_i has indicated a preference of meeting exactly s_i pets.
   - The j^{th} pet P_j can meet at most p_j students.
   - All meetings must be between compatible pairs.

   You would like to use a single Max-Flow computation to decide which pairs of students and pets should meet on each day. Starting with the compatibilities provided, draw the graph on which one would run the Max-flow algorithm. Indicate the source by S, the sink by T and mark the capacities of all edges.



Solution:



11 Maximum Connected Subgraph (18 points)

You are given a tree $T = (V, E)$ with vertex weights $w(v)$. The weights can be any real number (positive or negative). You wish to find the largest possible weight of any connected subgraph $S \subseteq V$, where the weight of S is defined as $\sum_{s \in S} w(s)$. Note that if S is empty, that is a valid subgraph with total weight 0.

(a) (**4 points**) In *maximum contiguous subarray*, you are given an array $[a_1, \ldots, a_n]$ where a_n are arbitrary real numbers, and you want to find the maximum-sum contiguous subarray. Describe how to use an algorithm for maximum connected subgraph as a black-box to solve maximum contiguous subarray.

Solution: Construct a graph that is a path of length n, and give the vertices weights $[a_1, \ldots, a_n]$ in that order. Then running the algorithm for maximum connected subgraph will solve maximum contiguous subarray. The root can be any arbitrary vertex.

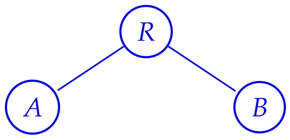
This works because the connected subgraphs of a path are exactly the contiguous subarrays for the array.

(b) (**4 points**) Henry fixes an arbitrary vertex r to be the root of the tree, and proposes the following dynamic programming algorithm to find the weight of maximum connected subgraph:

- *Subproblems*: $N[v]$ - the weight of the largest connected subgraph in the subtree of v
- *Base cases*: if v is a leaf, $N[v] = \max(0, w(v))$
- *Recurrence*: $N[v] = \max(0, w(v)) + \sum\{N[c] \mid c \text{ child of } v, N[c] > 0\}$
- *Return*: $N[r]$ where r is the root

Give an example of a tree on 3 vertices along with weights for which Henry's algorithm fails.

Solution: Consider the following example, with R weight -10, A and B weight 20:



Henry's algorithm will return 40 as the weight of the largest connected subgraph, but this corresponds to the subgraph A and B (excluding R), which is not connected. The issue is that the subproblem does not consider if the subgraph is building is connected.

(c) (**5 points**) Now, design a dynamic programming algorithm to *correctly* find the weight of the maximum-weight connected subgraph. First, define your subproblem(s).

Solution:

- $N[v]$ - the maximum connected subgraph in the subtree of v that must include v

(d) (**5 points**) Write the recurrence relation for the subproblem(s).

Solution:

$$N[v] = w(v) + \sum\{N[c] \mid c \text{ child of } v, N[c] > 0\}$$

If you are forced to include the root v, then to maximize weight you should also include the maximum subgraphs starting at the root of each child, so long as they are positive.

To finish this off, you would do $\max_v N[v]$ to find the maximum connected subgraph. The total runtime is $|V|$, which by part (a) also gives a linear-time algorithm for maximum contiguous subarray sum.