Midterm 1

1 True/False

3 points each.

(a) The MST of G is unique if and only if G has distinct edge weights.

Solution: False, e.g. consider when G is a tree where all edge weights are the same. Then G has only one MST, which is all edges in G.

(b) The spanning tree of maximum weight in G is the minimum spanning tree in a copy of G with all edge weights negated.

Solution: True. Minimum spanning algorithms only consider relative edge order so still work with negative edge weights to produce a minimum spanning tree. The spanning tree T that minimizes $-w(T)$ maximizes $w(T)$.

(c) Let G be a weighted undirected graph with positive edge weights where edge e has weight w_e for all $e \in E$, and G' be a copy G except that every edge e has weight w_e^2. Any MST of G' is an MST of G.

Solution: True. One way to see this is that the ordering of edges by weights remains the same, so the set of MSTs that can be output by Kruskal's is the same.

(d) In Huffman coding, assuming distinct frequencies, the character with the largest frequency has depth (distance to root) less than or equal to the depth of every other character in the Huffman tree.

Solution: True. If not, you can swap it with the character that is closer to the root and the encoding improves.

(e) If a satisfying assignment for a Horn formula exists, the greedy algorithm for Horn formulas finds the satisfying assignment that sets the fewest variables to true among all satisfying assignments.

Solution: True. We know the algorithm only sets a variable to true only if it must be true in every satisfying assignment.

(f) If a linear program has more than one optimal solution, then it has infinitely many.

Solution: True. Every point between two optimal solutions is also feasible and optimal.

(g) Recall that in a zero sum game, a player has a best defense strategy: a strategy that will achieve the optimal expected payoff even if the opponent knows the strategy beforehand. It is always best to play this strategy, *even if you know ahead of time your opponent is playing some non-optimal strategy*.

Solution: False. If you know your opponent is playing some non-optimal strategy, it may be possible to do better than the expected payoff for your optimal strategy.

(h) If we multiply all capacities by a constant c, then the max-flow of the graph will by multiplied by c.

Solution: True, the min-cut of the graph will be c * the weight of the initial cut, as this transformation affects all cuts equally. Therefore the max flow of the graph will be c * the value of the initial max flow.

(i) If all edge capacities in a max flow problem instance are distinct, the maximum flow is unique.

Solution: False. Say that there is one edge from the start vertex s to a vertex v with capacity 1. Then all flows starting from v are bottlenecked by 1, so if there are multiple different paths to t with edges that have capacity greater than 1, any of those paths can be chosen, even with different weights, with the same maximum flow.

(j) Suppose the maximum (s,t)-flow of some graph has value f. Now we increase the capacity of every edge by 1. Then the maximum (s,t)-flow in this modified graph will have value at most $f+1$.

Solution: False. An example is two parallel s-t paths where all the edges in one path have capacity 2, and all the edges in the other path have capacity 1. The maximum flow is 3, originally, and then it increases to 5, rather than 4 if one increases each edge capacity by 1.

2 Quick Fixes.

(a) Suppose you are given a MST T in a graph G. Consider further that a tree edge, $e \in T$, has its weight increased. Give a linear time method to find an MST in the graph with new weights. You must justify the correctness of your answer, but need not provide any runtime analysis. (**6 points**)

Solution: Removing edge e will break the tree into two components. Find the minimum weight edge between those components, and add it to $T - e$ to produce a new tree. The resulting set of edges is a tree as there is no cycle and it has $n-1$ edges and is minimum as we added the minimum weight edge across a cut. The runtime is linear since one can use connected components to find the connected subsets and one can find the minimum valued edge between the subsets with a scan over edges giving an $|E| + |V|$ time algorithm.

(b) Consider a knapsack instance with n items where item i has weight w_i and value v_i. Suppose you have a solution for the knapsack problem **with repetition** of weight W. Briefly describe as efficient a method as possible to compute a solution with the addition of an $(n+1)$th item with weight w_{n+1} and value v_{n+1}. State your running time. No justification is needed.

(You may assume that you still have the table where $K(w)$ for $w \in [0, W]$ corresponds to the highest value knapsack of weight w. To be clear, repetition is allowed.) (**6 points**)

Solution: We will compute a new table $K'(w)$ which corresponds to the best knapsack with repetition of all the items by $K'(w) = \max\{K(w), K'(w - w_{n+1}) + v_{n+1}\}$. The runtime is $O(W)$ since $w \in [0, W]$ and the maximum takes constant time.

3 A little bit of greed.

(a) Suppose you have an undirected graph $G = (V, E)$, with average degree d (or equivalently $|E| = d|V|/2$).

Assume d is even for simplicity. We want to find some $V' \subseteq V$, such that each $v \in V'$ has at least $d/2$ neighbors in V'.

Consider the algorithm that begins with $V' = V$, and repeatedly removes any vertex from V' which has strictly less than $d/2$ neighbors in V' until no such vertices exist in V'.

1 This algorithm terminates with a non-empty set V' where every vertex $v \in V'$ has at least $d/2$ neighbors in V'. (**2 points**)

Solution: True. See next part.

2 If you answered **True**, provide a proof. If you answered **False**, give an example graph where the algorithm removes all vertices. (**6 points**)

Solution: Let r be the number of vertices removed. The total number of edges removed is strictly less than $rd/2$.

If all vertices are removed we have $r = |V|$, the total number of edges removed is strictly less than $d|V|/2$. The total number of edges is $d|V|/2$, so they all weren't removed which contradicts the assumption that all vertices were removed.

(b) In homework, you saw the following problem: You have G, with weights $\ell(v) \geq 0$ and one proceeds by marking vertices until all vertices are marked. Furthermore, when marking a vertex u the player receives $\sum_{v \in M(u)} \ell(v)$ points where $M(u)$ is the set of marked neighbors of u.

1 The number of points for the strategy of marking the vertices in decreasing order of $\ell(v)$ is $\sum_{e=(u,v) \in E}$ ____. Fill in the blank (put your answer in the box). (**2 points**)

Solution: $\sum_{e=(u,v)} \max\{\ell(u), \ell(v)\}$. Since the order is to mark from largest to smallest, for each edge one receives the larger point value of its endpoints.

2 Give a one line argument based on the previous part that this order is optimal. (**2 points**)

Solution: This order maximizes the expression in the sum over edges. That is, no order can get more than $\max(\ell(u), \ell(v))$ for any edge $e = (u, v)$.

(c) Recall the greedy set cover algorithm which repeatedly adds the set with the most uncovered elements to its solution until all elements are covered.

Suppose we run the greedy set cover algorithm on an instance where there are 9 elements in the union of all the sets, and the optimal solution uses 3 sets to cover them. What is the largest possible number of sets in the solution output by the greedy set cover algorithm? Note: You do not need to compute or estimate any logarithms to answer this question. (**3 points**)

Solution: 5. Since the best solution uses 3 sets, in each iteration the greedy set cover algorithm adds a set that includes at least a third of the remaining elements. So the maximum number of uncovered elements after each iteration starts at 9 and then goes: $\lfloor 9 * \frac{2}{3} \rfloor = 6$, $\lfloor 6 * \frac{2}{3} \rfloor = 4$, $\lfloor 4 * \frac{2}{3} \rfloor = 2$, $\lfloor 2 * \frac{2}{3} \rfloor = 1$, $\lfloor 1 * \frac{2}{3} \rfloor = 0$.

This is tight, e.g. consider an input with sets $A = \{1,2,3\}, B = \{4,5,6\}, C = \{7,8,9\}, D = \{1,4,7\}, E = \{5,8\}$. The optimal solution is A, B, C, but greedy might buy D, A, E, B, C in that order.

4 Now for dynamism.

(a) In the **min weight** knapsack problem **without repetition**, one is given n items where item i has weight w_i and value v_i and one wishes to find the minimum weight set of items with value at least V. We define a subproblem $K(v, i)$ to be the minimum weight of any knapsack of value at least v using the first i items.

1 Give the recurrence for $K(v, i)$. (No justification needed) (**4 points**)

Solution: $K(v, i) = \min(K(v - v_i, i - 1) + w_i, K(v, i - 1))$. An optimal knapsack either uses item i or doesn't, and then uses an optimal knapsack on the previous $i - 1$ items. The recurrence follows.

2 What is the **minimum space complexity** of solving the recurrence above? (**2 points**)

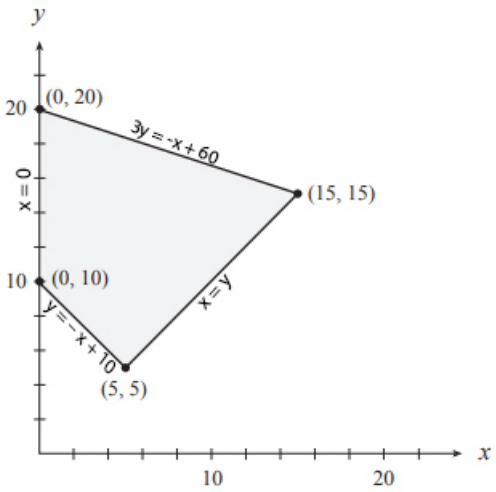
Solution: V. To fill in the values of $K(v, i)$ one only needs the values for $K(v, i - 1)$ for V possible values. Thus, one can compute the recurrence in $\Theta(|V|)$ space.

5 Linear Programming.

(a) What is the optimal value of the linear program max $x_1 + x_2$, where $x_1 + 2x_2 \leq 3$, and $x_1, x_2 \geq 0$? (**3 points**)

Solution: 3. There are a number of ways to see that $x_1 = 3$ and $x_2 = 0$ is optimal here. Either check the three vertices of the feasible region, or notice that x_2 provides less value than x_1.

(b) Suppose that you plot the feasible region from a linear program and get the following graph (the shaded region is feasible). Assuming that your objective is min $x + 2y$, what is the value of the optimal solution? (**3 points**)



Solution:

Note that we only need to check the 4 vertices of the polytope for the value of the objective function.

x	y	Value = x + 2y
0	20	40
15	15	45
0	10	20
5	5	15

Clearly, the point 5, 5 is the smallest, and so is the optimal solution, with transportation cost equaling \$15.

(c) Consider a linear program with n variables and m inequalities (including non-negativity constraints) where all variables are unrestricted.

1 A vertex of the linear program is a point where at least ____ inequalities hold with equality. (**2 points**)

Solution: n. This is the definition of a vertex.

2 For two points which are neighboring vertices at least ____ inequalities hold with equality at both points. (**2 points**)

Solution: $n - 1$. This is the definition of neighboring vertices of a polytope.

(d) Express the constraint $|x| \leq 4$ as a set of linear inequalities. (Do not introduce new variables in your solution.) **(3 points)**

Solution: $x \leq 4, x \geq -4$.

(e) We wish to find $x, y \geq 0$ that minimizes $|ax + by - 1|$. Write a linear program for this problem. **(4 points)**

Solution: $\min z$,
$ax + by - 1 \leq z$.
$ax + by - 1 \geq -z$.
$x, y \geq 0$.

For each of the LPs in parts (f), (g), (h), decide if it is infeasible, unbounded, has a unique solution, or has multiple solutions.

(f) Objective function: $\max x_1 + 2x_2$; constraints: $x_1 + x_2 \geq 10; x_1 \leq 0$ **(2 points)**

◯ infeasible	◯ unbounded	◯ unique solution	◯ multiple solutions

Solution: Unbounded. Take $x_1 = 0$ and x_2 can be arbitrarily large.

(g) Objective function: $\max x_1 + x_2$; constraints: $x_1 + x_2 \leq 3; x_1, x_2 \geq 0$ **(2 points)**

◯ infeasible	◯ unbounded	◯ unique solution	◯ multiple solutions

Solution: Multiple solutions: $(0,3)$ and $(3,0)$ and all points on the line connecting them maximize the objective function.

(h) Objective function: $\min 3x_1 - 2x_2$; constraints: $x_1 + x_2 \leq 10; 2x_1 + 2x_2 \geq 30$ **(2 points)**

◯ infeasible	◯ unbounded	◯ unique solution	◯ multiple solutions

Solution: Infeasible. The second equation is $x_1 + x_2 \geq 15$, which contradicts the first equation that $x_1 + x_2 \leq 10$.

(i) Write the dual of the following LP: **(4 points)**

$$\begin{aligned} \max\ & 2x_1 + 3x_2 \\ x_1 + 4x_2 &\leq 5 \\ 3x_1 + 2x_2 &\leq 4 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Solution:

$$\begin{aligned} \min\ & 5y_1 + 4y_2 \\ y_1 + 3y_2 &\geq 2 \\ 4y_1 + 2y_2 &\geq 3 \\ y_1, y_2 &\geq 0 \end{aligned}$$

(j) Consider the **zero sum game** represented by the matrix, where each entry in the matrix corresponds to the row player's payoff.

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

(1) What is the expected payoff if the column player plays $(2/5, 3/5)$ and the row player picks the best response to this strategy? (Recall that the row player is trying to maximize the payoff.) **(2 points)**

Solution: -1/5. The max of $4 \times (2/5) + -3 \times (3/5)$ and $-2 \times (2/5) + 1 \times (3/5)$.

(2) What is the expected payoff if the row player plays $(1/2, 1/2)$ and the column player plays the best response to this strategy? **(2 points)**

Solution: -1. The min of $4 \times 1/2 + -2 \times 1/2$ and $-3 \times 1/2 + 1 \times 1/2$.

(3) Is the expected payoff positive, negative, or zero when both players play optimally? **(1 point)**

◯ positive ◯ negative ◯ zero

Solution: Negative. The first solution shows that it must be negative.

6 In the flow.

For all parts of this problem, let G be a directed graph with source s and sink t, where **all edge capacities are** 1**.**

(a) (True/False). Consider running the Ford-Fulkerson algorithm to find a max s-t flow in G. If G is a DAG, then during the course of the algorithm, every edge capacity or residual reverse edge capacity in the residual graph will be 0 or 1. **(2 points)**

(Ford-Fulkerson is the name of the algorithm which repeatedly finds an s-t path in the residual graph and pushes as much flow as on possible on this path.)

Solution: True. So either the directed edge (u, v) has the original capacity or was saturated and the reverse arc has the capacity 1 since each augmentation is of value 1.Note that if (u, v) is an edge, then (v, u) is not an edge by the DAG assumption, so we do not have to account for (u, v) being saturated and (v, u) being unsaturated.

Let $\delta(v, G) = \text{indegree}(v, G) - \text{outdegree}(v, G)$, where $\text{indegree}(v, G)$ is the total capacity of edges pointing to v in G, and $\text{outdegree}(v, G)$ is the total capacity of edges pointing out of v in G. **For the following 3 parts your answer should be in terms of some subset of $\delta(v, G)$, indegree(v, G), outdegree(v, G) (for any vertex v) and f.**

(b) If one has routed f units of flow from s to t and the resulting residual graph is G', what is $\delta(v, G')$ when $v \notin s, t$? **(3 points)**

Solution: $\delta(v, G)$. Since for any intermediate vertex, we have flow conservation, so any incoming flow also leaves, so the new residual graph adds the same amount to both the in-degree and the out-degree.

(c) If one has routed f units of flow from s to t and the resulting residual graph is G', what is $\delta(s, G')$? (**3 points**)

Solution: $\delta(s, G) + 2f$. The flow out of the source removes outdegree and adds indegree.

(d) If one has routed f units of flow from s to t and the resulting residual graph is G', what is $\delta(t, G')$? (**3 points**)

Solution: $\delta(t, G) - 2f$. The flow into the sink removes indegree and adds outdegree.

7 Merging Piles

Given n sandpiles of weights $w_1, \ldots, w_n$ on a line, we want to merge all the sand piles together. At each step we can only merge **adjacent** sand piles, with cost equal to the weight of the merged sandpile.

In particular, merging sandpiles i and $i+1$ has cost $w_i + w_{i+1}$. Furthermore, one gets a single sandpile of weight $w_i + w_{i+1}$ in its place which is now adjacent to sandpiles $i-1$ and $i+2$.

Note that different merging orders will result in different final costs, and we wish to find the order that minimizes that cost.

(a) What's the minimum cost of merging the four piles $(100, 1, 1, 100)$? (**2 points**)

Solution: 1+1+2+100+102+100 = 306

(b) You will give a dynamic programming algorithm for this problem. (**12 points**)

(1) Define your subproblems.

Solution: We define a subproblem $f(i, j)$ to be the minimum possible cost of merging all the piles from i to j.

(2) Describe and justify the recurrence for computing the solution to a subproblem.

Solution:

$$f(i, j) = \min_{i \le k < j} \{f(i, k) + f(k+1, j)\} + w_i + \cdots + w_j$$

for all $i < j$.

For any solution that merges the sandpiles from i to j must have a final merge which merges piles from i to k and from $k+1$ to j. Then those sub-intervals of sandpiles should be merged at minimum cost which is defined recursively. The recurrence chooses the minimum cost solution of this form.

(3) Describe the base cases or base subproblems.

Solution: The base case is $f(i, i) = 0$ and $f(i, i+1) = w_i + w_{i+1}$.

(4) What is the time and space complexity of computing your solution?

Solution: Since there are at most n^2 $f(i, j)$'s the space is $O(n^2)$. The runtime is $O(n^3)$ since the recurrence for a subproblem takes the minimum over minimum over $O(n)$ terms.

8 Setting up (Integer) Linear Programs.

Integer linear programming is very general in that we can formulate a lot of problems using it.

A matching in a graph $G = (V, E)$ is a subset of edges, $M \subseteq E$, where no vertex in V is incident to more than one edge in M.

Formulate the problem of finding the maximum sized matching for an undirected graph $G = (V, E)$ as an **integer linear program**.

In addition to linear constraints, you may include constraining values of variables to come from a specific set, e.g., the constraint $x \in \{0, 1, 2, 3, 4\}$ indicates that x is an integer in $[0, 4]$. For each part below, you must state what is asked **both mathematically and in words**. (Only the description of each part is needed, no justification required) (**8 points**)

(1) Variables:

Solution: $x_e \ \forall e \in E$. This variable takes on values 1 or 0, representing whether the corresponding edge is in M or not respectively.

(2) Objective:

Solution: $\max \sum_{e \in E} x_e$. Maximize the number of edges picked.

(3) Constraints:

Solution:

- $x_e \in \{0, 1\}$: x_e should be either 1 or 0 representing whether e is in M or not.
- $\sum_{e=(u,v)} x_e \leq 1 \ \forall v \in V$. Represents the fact that at most one edge incident to v is in M.

9 Interval Covers.

Given an ordered set of points $a_1, \ldots, a_n$, we wish to cover them with **exactly** K intervals. For example, the points $\{1, 3, 4, 7\}$ are covered by the 2 intervals $\{[1, 3], [4, 7]\}$, or by intervals $\{[1, 1], [3, 7]\}$ (i.e., zero length intervals are fine.)

(a) The ℓ_1-cost of a set of intervals $\{[x_1, y_1], \ldots, [x_k, y_k]\}$ is $\sum_k |x_k - y_k|$. Give an efficient algorithm that given an ordered set of n points produces the minimum ℓ_1-cost K interval cover. Briefly justify correctness and state the time and space complexity. (**8 points**)

Algorithm:

Solution: Choose the largest $K - 1$ ℓ_1-gaps between adjacent points to divide the interval $[a_1, a_n]$ into K intervals. The ℓ_1-gap size between a_i and a_{i+1} is $|a_{i+1} - a_i|$.

Justification. Solution: The total cost is $|a_n - a_1|$− sum of gap sizes. This is minimized when the gap sizes are maximized. Choosing $K - 1$ numbers greedily from $n - 1$ yields the set of maximum sum.

Time/Space complexity.

Solution: $O(n + K \log n)$ since one can use a heap to find the largest element in the set of gap sizes. The space used is $O(n)$.

(b) The ℓ_2-cost of a set of intervals $\{[x_1, y_1], \ldots, [x_k, y_k]\}$ is $\sum_k |x_k - y_k|^2$.

Give an efficient algorithm that given an ordered set of n points produces the minimum ℓ_2-cost K interval cover. Briefly justify correctness and state the time and space complexity. (**8 points**)

Algorithm:

Solution: Let $D(i,k)$ be the minimum ℓ_2 cost of the set of points $\{a_1, \ldots, a_i\}$ with k intervals, where the last interval ends at k.

This can be recursively computed using the recurrence:

$$D(i,k) = \min_{j<i}(D(j,k-1) + |a_i - a_{j+1}|^2)$$

The base cases are $D(0,0) = 0$, $D(i,0) = \infty$.

Justification.

Solution: The last interval in a solution with value $D(i,k)$ is of the form $[a_j, a_i]$, and a solution of value $D(j,k-1)$. The recurrence chooses the best of those two form. The base cases indicate that covering nothing cost nothing and that covering with nothing is infeasible. **Time/Space complexity.**

Solution: Time is $O(n^2K)$ since there are $O(nK)$ subproblems and each can be filled in, in time $O(n)$. If one fills in the table in increasing order of k, one can use $O(n)$ space.