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:

☐ Evans 10 ☐ Wheeler 150 ☐ North Gate 105 ☐ Hearst Field Annex A1 ☐ VLSB 2060
☐ Cory 540AB ☐ Other

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

Rules and Guidelines

- **The exam is out of 110 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.**

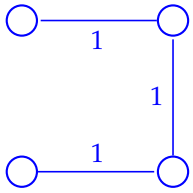
- ☐ Antares, Tuesday 5 - 6 pm, Mulford 240
- ☐ Kush, Tuesday 5 - 6 pm, Wheeler 224
- ☐ Arpita, Wednesday 9 - 10 am, Evans 3
- ☐ Dee, Wednesday 9 - 10 am, Wheeler 200
- ☐ Gillian, Wednesday 9 - 10 am, Wheeler 220
- ☐ Jiazheng, Wednesday 11 - 12 am, Cory 241
- ☐ Sean, Wednesday 11 - 12 am, Wurster 101
- ☐ Tarun, Wednesday 12 - 1 pm, Soda 310
- ☐ Jerry, Wednesday 1 - 2 pm, Wurster 101
- ☐ Jierui, Wednesday 1 - 2 pm, Etcheverry 3113
- ☐ Max, Wednesday 1 - 2 pm, Etcheverry 3105
- ☐ James, Wednesday 2 - 4 pm, Dwinelle 79
- ☐ David, Wednesday 2 - 3 pm, Barrows 140
- ☐ Vinay, Wednesday 2 - 3 pm, Wheeler 120
- ☐ Julia, Wednesday 3 - 4 pm, Wheeler 24
- ☐ Nate , Wednesday 3 - 4 pm, Evans 9
- ☐ Vishnu, Wednesday 3 - 4 pm, Moffitt 106
- ☐ Ajay, Wednesday 4 - 5 pm, Hearst Mining 310
- ☐ Zheng, Wednesday 5 - 6 pm, Wheeler 200
- ☐ Neha, Thursday 11 - 12 am, Barrows 140
- ☐ Fotis, Thursday 12 - 1 pm, Dwinelle 259
- ☐ Yeshwanth, Thursday 1 - 2 pm, Soda 310
- ☐ Matthew, Thursday 2 - 3 pm, Dwinelle 283
- ☐ Don't attend Section.

1 True/False. (22 pts)

(a) In every connected graph in which there is more than one edge of minimum cost, there is more than one minimum spanning tree

◯ True ◯ False

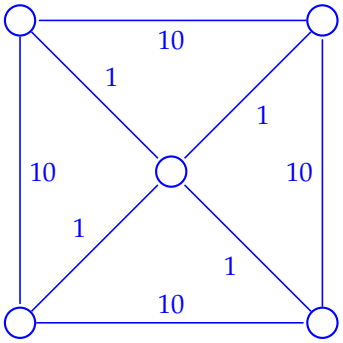
Solution: False. See counterexample:



(b) If a connected graph has a cycle in which all the edge costs are the same, then the graph has more than one minimum spanning tree

◯ True ◯ False

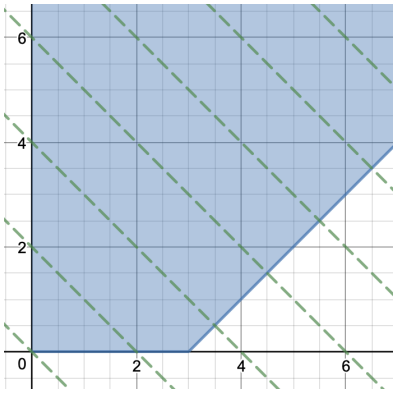
Solution: False. See counterexample:



(c) (**8 pts**) Consider the following LP:

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

Solution: Pictured is the feasible region and lines of equal objective value:



(i) The point $(x_1, x_2) = (0, 5)$ is feasible for the LP

◯ True ◯ False

Solution: True. All coordinates are non-negative and $0 - 5 \leq 3$

(ii) The point $(x_1, x_2) = (3, 0)$ is a vertex for the feasible region of the LP

◯ True ◯ False

Solution: True. This makes the inequality $x_1 - x_2 \leq 3$ tight, and also makes $x_2 \geq 0$ tight, so it defines a vertex

(iii) The linear program is bounded

◯ True ◯ False

Solution: False. You can see this because the line $x_2 = x_1 - 3$ lies entirely within the feasible region (it fulfills the constraint $x_2 \geq x_1 - 3$). The objective values of points on this line are $x_1 + x_2 = 2x_1 - 3$, which is unbounded as x_1 grows

(iv) The dual of the linear program is feasible

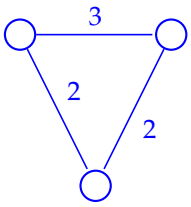
◯ True ◯ False

Solution: False. If the dual was feasible, we'd have an upper bound on the primal, but the primal is unbounded

(d) Given an undirected graph, the shortest path between any two nodes will belong to some minimum spanning tree of the graph.

◯ True ◯ False

Solution: False. See counterexample:

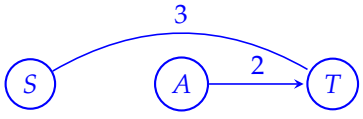


(e) (**6 pts**) For the following, consider a network G with an $s - t$ flow f. We say an edge in G is *saturated* when the flow across the edge is equal to the capacity.

(i) If f is a maximum flow, then f saturates all the edges going into t

◯ True ◯ False

Solution: False. The maximum flow on the following network doesn't saturate all edges going into T:



(ii) Let f be a maximum flow, and let G' be the result of removing all edges in G that are saturated by f. There is no path from s to t in G'.

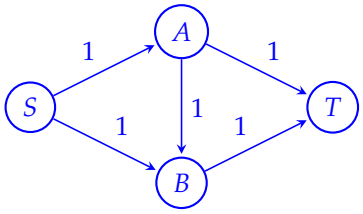
◯ True ◯ False

Solution: True. If there was still a path, that would mean we could further push flow, so f would not be a maximum flow.

(iii) Let f be any flow (not necessarily a maximum flow), and let G' be the result of removing all edges in G that are saturated by f. If there is no path from s to t in G', then f is a maximum flow in G.

◯ True ◯ False

Solution: False. See counterexample:



Pushing one flow along $S \to A \to B \to T$ disconnects S from T, but the maximum flow is 2.

(f) Let $G = (V, E)$ be an undirected complete graph with edge weights given by w_{uv}. Define the subproblem $d[i, j]$ to be the length of the shortest path from vertex i to vertex j, with the following recurrence relation:

$$d[i,j] = \min\left(w_{ij}, \min_{v \in V}\left(d[i,v] + d[v,j]\right)\right), \quad \text{if } i \neq j$$
$$d[i,i] = 0$$

Given only G, there exists a dynamic programming algorithm that uses this recurrence relation to compute $d[i, j]$ for all $i, j \in V$.

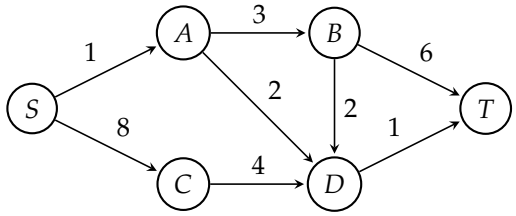
◯ True ◯ False

Solution: False, the recurrence is cyclic, so there is no way to write a dynamic programming to compute the answers from scratch. For example, on K_3, the complete graph with 3 vertices, $d[1,2]$ depends on $d[1,3]$ and $d[3,2]$, $d[1,3]$ depends on $d[1,2]$ and $d[2,3]$. $d[1,2]$ and $d[1,3]$ depend on each other. In fact, any pair of subproblems depend on each other.

2 Go With the Flow.

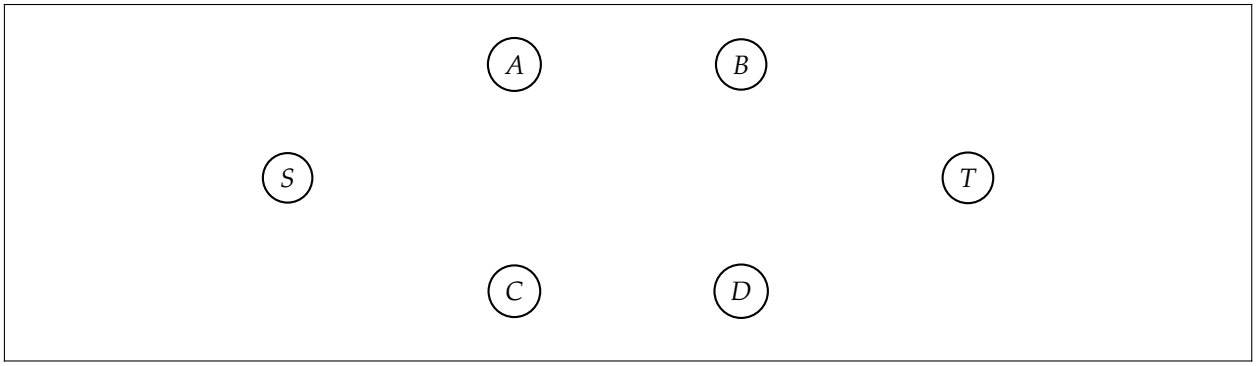
(**10pts**) Recall that the Ford-Fulkerson algorithm iteratively uses the residual network to compute the max flow.

(a) (**6 pts**) Consider the following network:

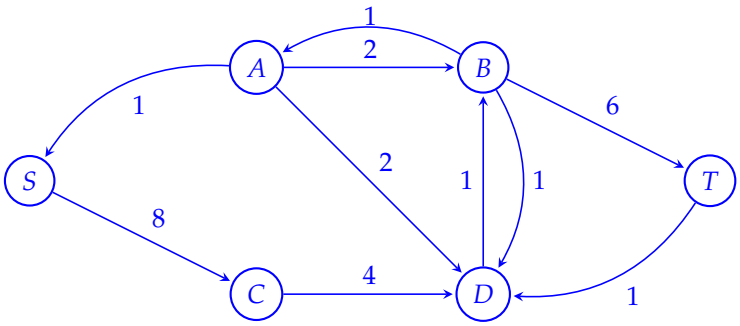


Let f be a flow that assigns 1 unit of flow on the path $S \to A \to B \to D \to T$ and 0 flow elsewhere.

(i) Draw the residual graph for f.



Solution:



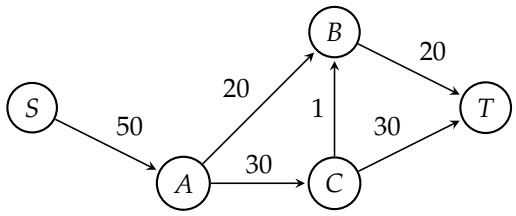
(ii) Give a path in the residual graph from S to T which can accommodate additional flow.

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

(iii) What is the value of the maximum flow on this network?

Solution: 2. The first flow had value 1, and this augmenting path has value 1. After this path S is disconnected to T, so the max flow is 2.

(b) (**4 pts**) Consider the following network:



For the next questions, consider all possible sequences of iterations of Ford-Fulkerson.

(i) What is the minimum number of iterations until there is no longer any path from S to T in the residual network?

Solution: 2. Push 20 along $S \to A \to B \to T$, then push 30 along $S \to A \to C \to T$.

(ii) What is the maximum number of iterations until there is no longer any path from S to T in the residual network?

Solution: 41. Alternate between pushing unit flow on $S \to A \to C \to B \to T$, and on $S \to A \to B \to C \to T$. This happens 20 times for each path, which is 40. Then finally push 10 flow along $S \to A \to C \to T$

Common mistakes:

i. For the max number of iterations partial credit was given to 21, 40, and 42 as these solutions demonstrate some understanding that repeatedly choosing the path S-¿A-¿C-¿B-¿T would maximize the number of iterations, and usually stemmed form a single minor error.

ii. The solution 50 was given a smaller amount of partial credit, as it comes from the general bound for the number of iterations the Ford-Fulkerson algorithm could take, despite the bound not being tight in this case.

3 Grab bag. (14 pts)

(a) (**4 pts**) Formulate the dual of the linear program below in terms of the variables y_1 and y_2.

$$\begin{aligned} \max\ & x_1 + x_3 \\ & x_1 - x_2 \leq 5 \\ & x_1 + 3x_2 - x_3 \leq 2 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

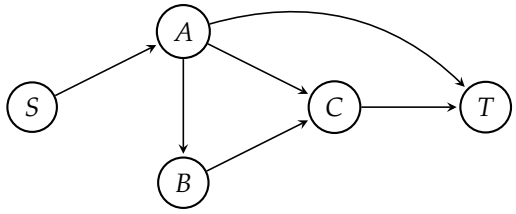
Solution: The dual LP is:

$$\begin{aligned} \min\ & 5y_1 + 2y_2 \\ & y_1 + y_2 \geq 1 \\ & -y_1 + 3y_2 \geq 0 \\ & -y_2 \geq 1 \\ & y_1, y_2 \geq 0 \end{aligned}$$

(b) (**4 pts**) Recall the dynamic programming algorithm that finds the shortest path from s to t in an unweighted DAG. Let $P(u)$ denote the length of the shortest path from u to t, and recall that:

$$\begin{aligned} P(u) &= \min_{(u,v)\in E} 1 + P(v) \\ P(t) &= 0 \end{aligned}$$

Suppose the algorithm was run on the DAG below. Give **an order** in which the subproblems will be solved from first to last (e.g. "$P(S), P(B), P(C), \ldots$")



Solution: A topological sort visits parents before children. But the subproblems here depend on chilren, so we need to compute children first. Therefore we must run this in reverse topological order:

$$P(T), P(C), P(B), P(A), P(S)$$

Common mistakes:

(a) Credit was given to P(T), P(C), P(B), P(A), P(S) OR P(C), P(B), P(A), P(S) It is slightly ambiguous whether P(T) needs to be solved or not as it is a base case

(b) Please note that a particular subproblem is solved when you have computed the the associated value for it – this is not the same as the order in which a recursive and memoized solution would add subproblems to the recursive stack. Even a recursive+memoized solution would still compute solutions in the above order.

(c) (**2 pts**) Consider the following set of clauses defining a HORN-SAT formula $\varphi(w, x, y, z, a, b)$:

$$\Rightarrow x, \quad x \Rightarrow y, \quad x \wedge y \Rightarrow w, \quad y \wedge w \Rightarrow z, \quad (x \vee \overline{w} \vee y \vee \overline{z} \vee \overline{a} \vee \overline{b})$$

How many satisfying assignments are there for φ?

Solution: We had to cancel this question because the set of clauses given was not actually a HORN-SAT formula. Apologies. 2 points were given.

The question is still valid if you ignore the fact the formula is not HORN-SAT.

(d) (**4 pts**) We have a primal LP with objective $\max x_1 + 4x_2$, and a dual LP with objective $\min y_1 - 2y_2 + 0.5y_3$. Suppose that the points $(0, 0.5)$ and $(3, 1, 2)$ are feasible in the primal and dual respectively. Argue in two or fewer sentences that they are also optimal for their respective LPs.

Solution: We can see that both points achieve a value of 2 on their objective function, and since solutions for the dual upper bound the primal and vice versa, both must be optimal

Common mistakes:

(a) Only states that both points are feasible in the dual and thus must be optimal. This is not sufficient as one could exhibit any primal feasible and dual feasible point, but they are not optimal unless their objective values are equal.
Alternatively, some solutions state that both points being feasible imply that their optimals intersect. However, this not the reason. The reason is that their objective values are equal.

(b) Some solutions state that the primal and dual optimal points intersect at only one point in the primal and dual feasible region. However, the feasible regions of the primal and dual LPs are in two different spaces. The primal is in $\mathbb{R}^2$ and the dual is in $\mathbb{R}^3$

(c) "If the primal and dual are both bounded and feasible, then they must have the same optimum value" – this is the strong duality theorem, but it is the wrong direction of it. Generally, this got the "minor mistake" one point off.

4 Bounding Codewords.

(**6 pts**) Consider the Huffman encoding for an alphabet of characters $c_1, \ldots, c_n$ such that $n = 2^k$ for integer $k \geq 1$, with respective frequencies $f_1, \ldots, f_n$. For each of the following characters, provide as a function of n the minimum and maximum possible codeword length for all sets of f_i such that $0 < f_1 \leq \ldots \leq f_n < 1$.

Solution: Common mistake: Off-by-one error

(a) The most frequent character, c_n

Minimum: Maximum:

Solution: The minimum is 1. Any character must be assigned some bitstring, and a zero-length bitstring is not possible.

The maximum is $\log n$. The most frequent character is the highest in the tree, and if it waas lower than $\log n$ the tree would have too many leaves.

(b) The least frequent character, c_1

Minimum: Maximum:

Solution: The minimum is $\log n$. A Huffman tree with n leaves must have at least one node at depth at least $\log n$, so the least frequent character must be at depth at least $\log n$.

The maximum is $n - 1$, by the argument given in discussion. There must be at least two symbols at the lowest levels of the Huffman tree.

5 Tilted.

(**8 pts**) Consider the following family of linear programs parameterized by some real number c:

$$\begin{aligned} \max\ & x + y \\ & x \leq 5 \\ & y \leq 5 + cx \\ & x, y \geq 0 \end{aligned}$$

For each of the questions below, describe all possible values of c, e.g. "$c \leq -3$ or $5 < c \leq 10$ or $c = 15$". If there are no possible values, write "none". If all real values are possible, write "all".

(a) For what values of c is the point $(x, y) = (5, 0)$ feasible?

Solution: $c \geq -1$. This can be done geometrically, or observing that this point always satisfies the first constraint, and the first constraint is satisfied if $c \geq -1$.

(b) For what values of c does the LP have infinitely many optimal points?

Solution: $c = -1$. This is when the edge defined by the second constraint is parallel to the lines of equal objective value.

(c) For what values of c is the LP unbounded?

Solution: None. The LP is always bounded. $x + y \leq 5 + 5 + cx \leq 10 + 5c$, which is an upper bound on the maximum for any c.

(d) For what values of c is the dual LP bounded? (*Hint:* you do not need to construct the dual to answer this question)

Solution: All. The LP is always feasible. The dual is always feasible, so it must always be bounded

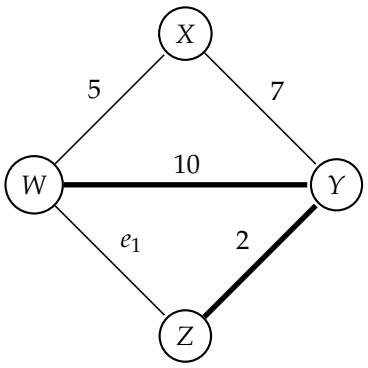
6 Happy Little Trees.

(**10 pts**) For each of the following graphs, all e_i can vary between 1 and 100, inclusive.

For each bolded edge below, mark **exactly one of** ABCD:

- Mark A if for all possible values of the e_i's, every MST contains the bolded edge
- Mark B if for all possible values of the e_i's, only some MSTs (at least one, but not all) contain the bolded edge
- Mark C if for all possible values of the e_i's, no MST contains the bolded edge
- Mark D if none of the above

(a)



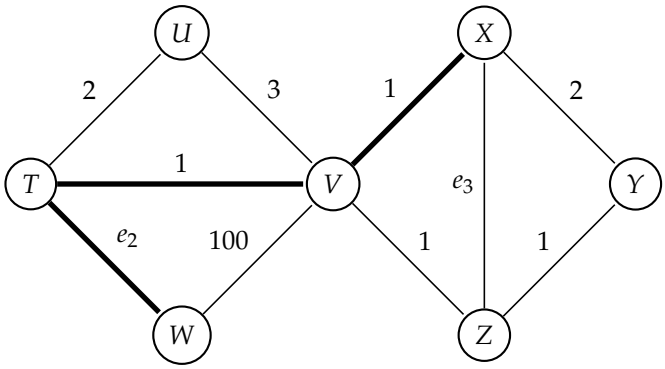
(i) The edge $\{W, Y\}$ ○A ○B ○C ○D

Solution: C: $\{W, Y\}$ is the heaviest edge in a cycle, so it is never in any MST

(ii) The edge $\{Z, Y\}$ ○A ○B ○C ○D

Solution: A. Consider any spanning tree that does not contain $\{Z, Y\}$. If you add $\{Z, Y\}$ in, that will form a cycle. The only cycles are $Z \to W \to Y \to Z$, or $Z \to W \to X \to Y \to Z$. In either case, the heaviest edge in the cycle is not $\{Z, Y\}$, so if it replaces the heaviest edge that will make a smaller spanning tree.

(b)



Solution: This essentially decomposes into two subproblems, one on $TUVW$ and one on $VXYZ$, for each bolded edge you only need to consider the subgraph it is a part of.

(i) The edge $\{T, V\}$ ◯A ◯B ◯C ◯D

Solution: A. Similar cycle argument to a.ii)

(ii) The edge $\{T, W\}$ ◯A ◯B ◯C ◯D

Solution: D. Any spanning tree must contain one of $\{T, W\}$ or $\{V, W\}$.
If $e_2 = 1$, then $\{T, W\}$ is in every MST. If $e_2 = 100$, then it can be swapped with $\{V, W\}$ with no change in weight, so there is an MST that does not contain it. Therefore D must apply.

(iii) The edge $\{V, X\}$ ◯A ◯B ◯C ◯D

Solution: D. If $e_3 > 1$, then the subgraph $VXYZ$ has a unique MST that contains all the weight-1 edges. If $e_3 = 1$, then by inspection there are multiple MSTs, one which contains $\{V, X\}$ and one that doesn't

7 Sparse Graph MSTs.

(**12 pts**) You are given a weighted, connected, undirected graph $G = (V, E)$ such that $|E| = |V|$. Give an algorithm that finds a minimum spanning tree in time $O(|V|)$.

(a) Main Idea

Solution: The solution relies on the fact that exactly one edge in the graph will not be in the MST, since any tree has exactly $|V| - 1$ edges. Since this graph has $|V|$ edges and is connected, there must be one cycle. Therefore, our algorithm will perform DFS to detect a cycle and then output the tree containing all edges in the graph except the largest weight edge in the cycle.

We know the graph is still connected because we deleted an edge in a cycle. Furthermore, the graph can have at most one cycle, so if we delete any edge not in this cycle, the graph will become disconnected. Thus deleting the maximum edge in this cycle removes the most weight from the graph, giving the smallest weight spanning tree.

Common mistakes:

(a) Some students tried to run Prims or Kruskals algorithms, which are not linear time.

(b) Many submissions tried deleting each edge and checking if that disconnected the graph; this will take quadratic time.

(c) Some solutions just deleted the heaviest edge without making sure it was in the cycle / would leave the graph connected

(d) Some solutions run DFS by always following the lightest edge. This does not work in the case of a cycle since this would simply remove the last edge traversed by DFS (which may not be the heaviest edge).

(b) Runtime analysis

Solution: The DFS cycle detection procedure takes time $O(|V| + |E|)$ and finding the largest edge in the cycle takes $O(|V|)$ time. So the algorithm takes time $O(|V| + |E|)$. However, we have that $|E| = |V|$ for this graph, so the runtime is indeed $O(|V|)$.

8 Points on a Line.

(**14 pts**) You are given a set S of n points on a line. The points are given to you in sorted order. You want to find a set C of points of minimum size such that that every point in S is at distance at most 1 from at least one point in C. (Note that the points in C need not belong to S.) You would like a greedy algorithm that runs in time polynomial in n and finds an optimal solution.

For example, given the points $S = \{2.7, 3.2, 3.6, 4, 4.9, 5.2\}$, a possible solution is $C = \{3.5, 5\}$, and there is no smaller solution.

(a) Describe your greedy algorithm (3 sentences or less)

Solution: Pick the smallest point $s \in S$, and add $s + 1$ to C. Remove any point x in S such that $s \leq x \leq s + 2$, as it has been covered by $s + 1$. Repeat this process until S is empty.

(b) Argue for its correctness using an exchange argument.

Solution: Let C^* be the greedy solution and C be some other solution with $C \neq C^*$, with both sets in sorted order. Consider the first point where they differ, i.e. i such that $c_i^* \neq c_i$, and disregard any points in S covered by any of c_1 to c_{i-1}. It cannot be the case that $c_i > c_i^*$, as otherwise $c_i^* - 1 \in S$ won't be covered. Then $c_i \leq c_i^*$, in which case swapping c_i with c_i^* can only cover more points since $c_i^* - 1$ is the smallest remaining uncovered point.

Common mistakes:

(a) Considering another solution that is strictly smaller than C^* and saying that the greedy algorithm

in 8a) is "mathematically optimal" in some sense, so it "covers more points" than the other solution. This is an intuitive argument but not a rigorous one

(b) Doing the argument above but only for the first points in C and C^*, and either being vague about how to extend the argument or not saying anything at all for other points. We usually gave these 2 points

(c) What's the runtime of your algorithm?

Solution: $\mathcal{O}(n)$. The points are already sorted, and we can continuously pick the smallest point, add $s+1$ to C, and remove the points $s+1$ covers. All these constant-time operations happen at most once for each point, meaning linear runtime.

9 I am the Machine(s).

(**14 pts**) You have a mission-critical production system that consists of n stages that must be completed in sequence. Each stage i can only be completed by a machine of type M_i. Unfortunately, the machines are faulty, and machines of type M_i fail with probability f_i (you may assume all failures are independent of each other).

A system with one copy of each machine type will succeed with probability $(1 - f_1) \cdot (1 - f_2) \cdots (1 - f_n)$. By adding redundant machines of type M_i, stage i is not completed only if all machines of type M_i fail. Therefore, if we have m_i machines of type M_i, the probability that stage i is completed is $1 - f_i^{m_i}$ and the probability that the whole system succeeds is now $\prod_{i=1}^{n}(1 - f_i^{m_i})$. Notice that if **any** of the $m_i = 0$, the success probability will be zero.

Unfortunately, you only have B dollars to spend on machines, and it costs c_i dollars to purchase each machine of type M_i. You may assume both B and the c_i's are positive integers.

Give a dynamic programming algorithm that finds the maximum achievable success probability while staying under budget. *Your algorithm should compute a probability, not specific values for each m_i.*

More formally:

You are given probabilities $f_1, \cdots, f_n$, costs $c_1, \ldots, c_n$, and nonnegative integer budget B. Give a dynamic programming algorithm to compute the maximum success probability $\prod_{i=1}^{n}(1 - f_i^{m_i})$ where each m_i is a nonnegative integer and $\sum_i c_i m_i \leq B$.

(a) Define your subproblems.

(b) What are the base cases?

(c) Write the recurrence relation for the subproblems.

Solution:

We have an $n \times B$ table $P[\cdot, \cdot]$, with the intended meaning that $P[k, b]$ is the highest reliability probability that you can get for the sub-instance consisting of only of the first k tasks, given a budget b, that is, it is the maximum of $\prod_{i=1}^{k}(1 - f_i^{m_i})$ over choices of $m_1, \ldots, m_k$ such that $\sum_{i=1}^{k} c_i m_i \leq b$. We have

$$P[k+1, b] = \max_m P[k, b - c_{k+1} \cdot m] \cdot (1 - f_{k+1}^m)$$

where the maximum is taken over all $m = 0, 1, \ldots, \lfloor b/c_{k+1} \rfloor$. The base case is

$$P[0, b] = 1$$

for every $b = 0, \ldots, B$. The solution of the problem is the entry $P[n, B]$. We will fill up the table with an outer loop over $k = 1, \ldots, n$ and an inner loop over $b = 1, \ldots, B$.

Alternatively, we can define our subproblem to be $P(k, b, i)$, where $P(k, b, i)$ represents the highest reliability you can get for the sub-instance consisting of only of the first k tasks, given a budget b, where we purchase at least i machines for the kth task. In this case our recurrence is as follows

$$P[k, b, i] = \max(P[k-1, b, 0] \cdot (1 - f_k^i), P[k, b - c_k, i+1])$$

In this case our base cases are:

$$P[0, b] = 1$$

$$P[k, b] = 0 for b < 0$$

Full credit was given to this solution, as it still achieves the same runtime as the first solution.

You can also define $P(k, b, i)$ to represent the highest reliability you can get for the sub-instance consisting of only of the first k tasks, given a budget b, where we purchase at least i machines for the kth task

Common mistakes:

1. Many students over-parameterized their subproblems in ways that would increase the time to compute all subproblems.

   These solutions were often brute-force and tried compute the success probability for each combination of m_is.

2. Several students tried to have subproblems parameterized only by the remaining budget.

   These approaches typically failed to correctly compute the probability of success.