Final Solutions

- **The exam has 8 questions, is worth 110 points, and will last 170 minutes.**
- The point distribution is 3/15/9/8/15/15/15/30. You may be eligible to receive partial credit for your proof even if your algorithm is only partially correct or inefficient.
- 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! (Destress question) Recommend us an album/movie you like:

1 DP Warmup

(3pts) Write a dynamic programming recurrence to count $f(n,m)$, the number of distinct shortest paths in a 2D integer grid from $(0,0)$ to (n,m), where n and m are positive integers. You can only travel along the x and y directions (no diagonals). A counting argument will receive no credit. Only the recurrence relation is needed.

$f(i,j) = f(i-1,j) + f(i,j-1)$. Note that any shortest path will only go upwards or right, so any and all paths approaching a point i,j must come from below or to the left of it. A base case is not necessary for full credit, but $f(i,j) = \begin{cases} 1 & i = j = 0 \\ 0 & i < 0 \text{ or } j < 0 \end{cases}$.

2 Misc True/False + Justification

(3pts each) State clearly whether the given statement is true or false and give a brief justification for your answer.

(a) There exists a graph with 3 vertices such that the lowest postorder number of *any* DFS on the graph is not in a sink SCC.

(b) Let $f: \mathbb{N} \to \{0,1,2,3,4\} = 3x+4 \pmod 5$. Then $\mathcal{H} = \{f\}$, where $\mathcal{H}$ constitutes of *that single function* f, is a universal hash family of functions: $\mathbb{N} \to \{0,1,2,3,4\}$.

(c) Let the output of the Morris algorithm for an approximate counting of a stream of n numbers be $\tilde{n}$. Then the difference between n and $\tilde{n}$ (*i.e.* $|n - \tilde{n}|$) is bounded by a constant.

(d) We don't need to calculate the 7th roots of unity to calculate the `FFT` of a length 7 vector.

(e) $\sqrt{n} = O(2^{\sqrt{\log n}})$

(a) False, starting a DFS from any vertex in a sink SCC will have the lowest postorder number in that SCC. Note that all graphs must have a sink SCC. Common misconceptions: the question asks existence, so giving one counterexample on a certain 3-vertex graph is invalid; to use counterargument one must enumerate all 3-vertex graphs. While "DFS being forced to go back as there's no child" is the reason for traversing in sink SCC achieves our goal, such scenario can also happen in vertices in non-sink SCC, so it is not a valid explanation.

(b) False. Pick $x = 1$ and $y = 6$. $f(1) = f(6) = 2 \pmod 5$, for this x, y we have collision probability of 1, which makes $\mathcal{H}$ non-universal. Picking any x, y that collide serves as a counterexample.

(c) False. It's bounded by $[0, \frac{2^n-1}{2}]$ (resulting from experiencing no update every time or experiencing update every time), which grows as n grows. The only practical bound is the probability of exceeding relative error ($|n - \tilde{n}| \leq \epsilon N$).

(d) True. We pad the vector to length-8 and we use 1, 2, 4, 8th roots of unity for the calculation. We need to round up the length and pad these vectors to the nearest power of 2, 8, for FFT on length 7 vectors. (Evaluating at 7 points also works for DFT, but we can't divide-and-conquer.)

(e) False, through L'hopital's

Exam continues on next page ⟹

3 reducecuder

(3pts each) State clearly whether the given statements are True, True iff $P = NP$, True iff $P \neq NP$, or False and give a brief justification for your answer.

(a) There exists a polynomial time reduction from 3SAT to Palindrome Checking. Assume that Palindrome Checking is a program that verifies if a given word is a palindrome or not.

Note: A palindrome is a word that reads the same forwards as well as backwards. For example, "CS17071SC" is a palindrome, but "CS170" is not.

(b) There exists a polynomial time reduction from any NP hard problem to any other NP hard problem.

(c) There exists a polynomial time reduction from Palindrome Checking to 3SAT.

(a) True iff P = NP. If P = NP, we can solve 3-SAT in polynomial time by reducing it in polynomial time to some problem in P. Solve 3-SAT, then create a trivial word which is a palindrome iff the 3-SAT instance is satisfiable. This gives a polynomial time reduction from 3-SAT to Palindrome Checking. A polynomial-time reduction from 3-SAT to Palindrome Checking together with the polynomial time algorithm for Palindrome Checking implies a polynomial-time algorithm for 3-SAT and thus implies P=NP.

(b) False. The Halting problem which is NP hard cannot be reduced to 3 SAT which is also NP hard.

(c) True, you can solve Palindrome Checking in polynomial time and then create a trivial 3 SAT instance that evaluates to True if Palindrome Checking evaluated to True and False otherwise.

Giving an example of trivial 3 SAT instances is not required but for reference a trivial True 3 SAT instance could look like (a) and (b), and a trivial False 3 SAT instance could look like (a) and ($\neg$ a).

4 Param's Picnic Planning Fiasco

(8pts) The CS 170 course staff is planning a road trip to Guinland. Originally, Param was in charge of seating assignments in cars for all members of the course staff. He arranged p people in n cars where car i has capacity $c_i \forall i \in 1, \ldots, n$.

However, given Param's poor planning skills, he assigned people in cars terribly inefficiently leaving a lot of empty space in many cars. Given the number of people p, and the capacities of each of the n cars $c_1, \ldots c_n$, come up with an efficient greedy algorithm to fit people in cars such that the CS 170 course staff can take the least number of cars possible on the road trip.

Prove the optimality of your algorithm using an exchange argument. No runtime analysis required.

Main Idea: Sort the cars in descending order of capacity and while there's unassigned people $(p > 0)$ choose the next biggest car and assign min(c_i, p) to that car. Then update $p = p - c_i$.

Proof of Correctness: Assume the optimal solution chooses cars represented by the set $O = \{o_1, o_2, ..., o_k\}$ and our greedy solution chooses cars represented by the set $G = \{g_1, g_2, ..., g_m\}$. If $\exists$ some car in G that does not already exist in O we are guaranteed that the capacity of the car in G is at least the capacity of the smallest car in O (by construction of our greedy algorithm). Therefore, we can exchange the smallest car in O to be the corresponding car in G to produce a solution that is at least as optimal as the solution O. We repeat this for all elements in G that do not appear in O until $G = O$.

Note: Some students may choose to contradict that O is a optimal solution by showing G is a better solution which is correct too.

Exam continues on next page ⟹

5 Mo' Money Mo' Problems

(15 pts) Thanks to your efforts on the project, CS 170's igloo polishing venture is flourishing! In an attempt to cut costs and further maximize profits, CS 170 now needs your help figuring out the cheapest way to get from one igloo polishing task to the next. Our clients are spread out in the vast kingdom of Guinland which consists of $|V|$ cities. Assume that no two consecutive polishing jobs are in the same city.

The $|V|$ cities of Guinland are connected by $|E|$ directed edges which can be used to travel from one city to another, i.e. we can use an edge (u, v) to travel from city u to another city v. To travel from one city u to another city v we can use either cable car, snowmobiles, or trains which have strictly positive costs $C_{c(u,v)}$, $C_{s(u,v)}$, and $C_{t(u,v)}$ respectively. If you can travel from city u to city v, you can use any of the three modes of transport, i.e. if the directed edge (u, v) exists, all 3 modes of transport will be available between u and v each with their own respective costs.

Unfortunately, for some bizarre reason, we cannot take any same mode of transportation twice in a row, i.e., we cannot enter city u on a snowmobile and leave city u on a snowmobile, and the same applies for cable cars and trains. Given a directed graph representation of Guinland $G = (V, E)$ and cities s and t, come up with an algorithm that helps us find the cheapest path between city s and city t in G and analyze its runtime. No need for a proof of correctness.

We will create a new graph G'. For each city v (except s and t), create three vertices in G': v_c, v_s, v_t. v_c corresponds to coming to v via cable car, v_s corresponds to coming to v via snowmobile, and v_t corresponds to coming to v via train. If we can travel between cities v and u, then add the following six edges to G': $(v_c, u_s), (v_c, u_t), (v_s, u_c), (v_s, u_t), (v_t, u_c), (v_t, u_s)$. The weights of edges are the positive costs of transportation. This construction forces the edges to satisfy the requirement that we cannot travel via the same means of transportation subsequently. Then, for each city v such that we can travel from s to v, add edges $(s, v_c), (s, v_s), (s, v_t)$ with corresponding weights. For each vertex v_x from which we can travel to t, add two edges that correspond to two means of transportation other than x, again with the corresponding weights. Finally, run Dijkstra's Algorithm on G' from s and return the shortest distance from s to t.

The construction of G' takes $O(|V| + |E|)$ time and it has $O(|V|)$ vertices and $O(|E|)$ edges, so running Dijkstra's on it takes $O((|V| + |E|) \log |V|)$ or $O(|V| \log |V| + |E|)$ depending on what data structure we use for priority queue (we consider both runtimes correct).

An alternative construction would be to not make an exception for s and t and create three copies for all vertices, including s and t. Then, add a dummy vertex S such that S is connected to s_c, s_s, s_t with edges of weight 0. Then run Dijkstra's from S and return the minimum of distances from S to t_c, t_s, t_t. Note that these are not the only correct ways to account for these two vertices, any correct way works.

Incorrect alternative approaches:

1. Dynamic programming approach

   Many submissions tried to use a dynamic programming approach for this problem. This does not work because for a valid DP, we would need a DAG structure on this graph, however, it is not guaranteed to be a DAG.

2. "Turn it into a DAG"

   Some submissions tried to turn the graph into a DAG and then use something similar to a DP solution. This fails because there is no good way to cut the back edges to turn this into a DAG.

3. Modification of Dijkstra's

   Some submissions tried to modify Dijkstra's algorithm to accommodate for the three different means of transport, i.e. choose the best means of transport on each relaxation of edges. This does not lead to a valid solution because it greedily chooses the cheapest way of transport for the current timestep; however, it is possible that you have to first choose an expensive transport so that on the next step you can choose something much cheaper.

Exam continues on next page ⟹

6 Happy Birthday, Adnaan

(15pts) It's Adnaan's birthday today and the CS 170 staff are shopping presents for him in a shop. There are n items on the shopping list. The shop is doing a peculiar holiday sale: there are m "works well together" unordered pairs $e_k = (i, j)$, $k \in \{1, \ldots, m\}; i, j \in \{1, \ldots, n\}$. For the k-th pair ($k \in \{1, \ldots, m\}$), if you have already bought one item in the pair from the store, you are allowed to buy the other item in the pair for the price of d_k dollars instead of its original price. Given $(e_k)_{k=1}^m$, $(d_k)_{k=1}^m$, and the original price of the n items $(p_i)_{i=1}^n$, design an algorithm that outputs

- the least amount of money the staff could spend to buy all n items and
- a possible order of buying the n items that achieves this least amount of money.

Describe your algorithm and analyze its runtime. Proof of correctness is not needed.

Example:

$$n = 3, m = 3$$
$$(e_k)_{k=1}^m = (1,2), (2,3), (3,1)$$
$$(d_k)_{k=1}^m = 4, 2, 6$$
$$(p_i)_{i=1}^n = 10, 5, 3$$

The minimum amount of money is \$9, and it could be achieved as follows.

1. First buy item 3 with its original price $p_3 = 3$.
2. Since we have bought item 3, we can buy item 2 with \$2 or its original price of \$5. Of course we pay \$2 for item 2 to minimize expenditure.
3. Since we have item 3, we can buy item 1 with 6 dollars. We also have item 2, which allows us to buy item 1 with 4 dollars. Its original price is \$10. Therefore we pay \$4 for item 1.

Therefore if we buy the items in the order of $3, 2, 1$, we spent a total of \$9 to obtain all items.

Hint: Minimum Spanning Tree.

Utilise the hint. We note that MST algorithms pick $|V| - 1$ edges in a graph with $|V|$ vertices. Since we wish to pick n items, we create a graph with $n + 1$ vertices, one for each item and one "dummy" vertex. Then, we connect all item vertex to the dummy vertex with edge weight equal to the price of the item. We also connect "works well together" vertices with edge weight equal to "work well together" price. Note that if you possess one item in the pair, you may purchase the other item with price d_k, regardless of which item you bought first.

Now we find MST in this graph. Starting from the dummy node, we perform any search to get some order to traverse the tree. We purchase in the order which we visit the vertices that represent corresponding items.

Creating the graph takes $\mathcal{O}(m + n)$ time as we have $n + 1$ vertices and $m + n$ edges; traversing through the tree must be able to be upper-bounded by $\mathcal{O}(m + n)$ since that is the time complexity to traverse the whole graph. Running MST takes $\mathcal{O}[(m + n) \log(m + n)]$ time, which is the bottleneck and the overall time complexity of our algorithm. One, of course, may employ an optimised MST algorithm and achieve faster runtime, but for the purpose of this exam, we are satisfied with such time complexity.

Now for the proof of correctness which we do not require. First, we show MST algorithm indeed produce a purchasing plan that is optimal. We note that in our graph, each edge represent a possible transaction, and our total cost is the sum of edge weights. Suppose there is a better transaction represent by another edge that replaces one edge in the MST. If a tree is not formed after replacement and it's disconnected, then we may assert we are unable to buy each item once. If a cycle is formed, them we purchased some item possibly twice. If it is still a tree, we notify the sum of its edge weights must be at most as optimal as the MST. In all 3 cases, the MST we found is superior.

We then turn to show the traversal produce an legal purchase plan. Since we start from the dummy vertex, we are forced to traverse with an edge that requires us to use original price. Then, every time we use

Exam continues on next page ⟹

an edge that has prior purchase requirement, note that the properties of search forces us to visit either of the vertices that the edge is connecting to, and that indicates we have purchased (visited) the other item (vertex) to use "works well together" price. This concludes our solution.

Exam continues on next page ⟹

7 FALSY-SAT

(15 pts) Define the FALSY-SAT problem as follows:

Given a 3-CNF boolean formula ϕ, **decide** if there's an assignment to the variables of ϕ that satisfies ϕ without containing three true literals in any clause.

In other words, a FALSY-SAT instance is satisfied if and only if exactly 1 or 2 literal(s) in each clause is/are assigned to true.

For example, $(\overline{x} \vee \overline{x} \vee \overline{x})$ is not satisfiable under the conditions of FALSY-SAT because all three literals are the same, so to make the clause true, false must be assigned to x, and three literals in the clause, namely three $\overline{x}$, must all be true.

(a) Give a polynomial time reduction from 3SAT (*i.e.* **decide** if the given 3-CNF is satisfiable) to FALSY-SAT by replacing each clause c_i

$$(a_1 \vee a_2 \vee a_3)$$

with the two clauses

$$(a_1 \vee a_2 \vee b_i) \text{ and } (\overline{b_i} \vee a_3 \vee t)$$

where b_i is a new variable for each clause c_i and t is a single new additional new variable.

Hint: For one direction for the proof, show that you could always make $t = 0$.

(b) Suppose you have a correct answer to the previous part, conclude that FALSY-SAT is NP-Complete.

First, observe an important property of FALSY-SAT: if an assignment/witnesss X satisfies a given FALSY-SAT instance ϕ, then $\bar{X}$, the assignment with all literals negated, satisfies it as well. Call this Property 1.
Proof: Suppose not. Then with $\bar{X}$, ϕ must have at least one clause that looks like $(T \vee T \vee T)$ or $(F \vee F \vee F)$. With X, they are $(F \vee F \vee F)$ or $(T \vee T \vee T)$, which are still unsatisfiable. $\perp$.
We can demonstrate the correctness of such Cook reduction by showing that a FALSY-SAT instance obtained by executing such replacement is satisfiable if and only if the original 3-SAT instance is satisfiable.
If: If the original clause is satisfiable because of a_1 or a_2 but not a_3, FALSY-SAT instance can set b_i to be false, making the first clause satisfied; the second clause is also satisfied with a true $\bar{b_i}$ and a false a_3.
If the original clause is satisfiable because of a_3 but not a_1 and a_2, FALSY-SAT instance can set b_i to be true, making the second clause satisfied with a false $\bar{b_i}$ and a true a_3 and making the first clause satisfied with a true b_i and two false literals.
The above two cases have no t involved. Nevertheless, if the original clause is satisfiable by all three literals, set b_i to be false and t to be false, we observe the new clauses are satisfiable.
Only If: First, ensure t is assigned false (use property 1 if t is assigned true). It is impossible to get all a_1, a_2 and a_3 to be false as it would imply either the first clause or the second clause is not satisfied.
The reduction is evidently polynomial time: each clause takes $\mathcal{O}(1)$ to process (creating one b_i and splitting).

For (b), first behold that (a) demonstrate that FALSY-SAT is $\mathcal{NP}$-Hard, as it can be reduced from a $\mathcal{NP}$-Complete problem. Then, observe noticeably that FALSY-SAT has an $\mathcal{O}(n)$ verifier. This fact displays that FALSY-SAT is in $\mathcal{NP}$.

Exam continues on next page ⟹

8 Set Cover Symphony

(30pts) In class we saw that there is a $\ln n$ greedy approximation algorithm for set cover. In this problem, we will first see another approximation algorithm for set cover (part (a)), then we will discover how randomness could give us a much better algorithm (part (b) through (d)).

Note: all subparts are doable without completing the other subparts of this question. So if you're stuck, feel free to try out other subparts.

Recall in set cover we have a universe $\mathcal{U} = \{1, \ldots, n\}$ of items that we would like to cover by taking some subcollection of given sets $S_1, \ldots, S_m \subseteq \mathcal{U}$. We would like to take as few sets as possible. One can express set cover as an Integer Linear Program in the following way:

$$\min \sum_{i=1}^{m} x_i$$

s.t.

$\forall u \in \mathcal{U}$, $\sum_{i:\, S_i \ni u} x_i \geq 1$
$\forall 1 \leq i \leq m$, $0 \leq x_i \leq 1$
$\forall 1 \leq i \leq m$, x_i is an integer

Figure 1: Integer linear program representing set cover.

In the Integer Linear Program (ILP), we have one variable x_i per set S_i. We interpret an ILP solution with $x_i = 1$ as indicating that S_i should be taken in the subcollection, and $x_i = 0$ indicates that it should not be taken. Also, the notation "$\sum_{i:\, S_i \ni v}$" denotes a summation over all i such that v is in S_i.

(a) Show that if no element is contained in more than k sets, then there is a polynomial-time k-approximation algorithm for set cover *by using the linear programming relaxation* obtained by removing the integrality constraints. Specifically, you should describe how to obtain a solution from the relaxed linear program without integrality constraints, show your algorithm works (*i.e.* all items will be covered), and prove the approximation factor is k.

(b) Consider taking the linear programming relaxation of the above ILP and solving it to obtain a vector $x^* \in [0,1]^m$ (*i.e.* m real numbers between 0 and 1, inclusive). Now define $p \in [0,1]^m$ by

$$p_i := \min\{1, \beta \cdot x_i\}$$

We interpret the p_i as probabilities and randomly take set S_i with probability p_i and do not take it otherwise. Show that the *expected* number of sets we take is at most $\beta \cdot \mathsf{OPT}$, where OPT is the number of sets taken in an optimal set cover.

(c) Consider the same process as in part (b). How do we know that the sets we take actually form a set cover?! Show that if for any particular item, the probability that it is not covered by at least one of the sets we randomly take is at most $e^{-\beta}$.

Hint: for any $z \in \mathbb{R}$, recall from class that $1 + z \leq e^z$ (you may use this fact without proof).

$$\min \sum_{i=1}^{m} x_i$$

s.t.

$\forall u \in \mathcal{U}$, $\sum_{i:\, S_i \ni u} x_i \geq 1$
$\forall 1 \leq i \leq m$, $0 \leq x_i \leq 1$
$\forall 1 \leq i \leq m$, x_i is an integer

Figure 2: The integer linear program representing set cover is copied here for your convenience.

(d) Now upper bound the probability that the subcollection we take *fails* to cover all items in the universe. Conclude that if we set $\beta = \ln(10n)$, then the probability that the subcollection of sets we take is a set cover is at least 90%. You may use the result from part (c) without proof.

Exam continues on next page ⟹

(a) **Algorithm**: Solve the relaxed LP. For each x_i, round it up to 1 if $x_i \geq \frac{1}{k}$, otherwise round it down to 0.

Correctness: For each $u \in \mathcal{U}$, there must be a $S_i \ni u$ such that $x_i \geq \frac{1}{k}$ since no element is contained in more than k sets. Therefore for all $u \in \mathcal{U}$ there is a set that covers it.

Approximation factor: Let $\mathsf{OPT}(LP)$ be the optimal value of the LP relaxation, and let OPT be the optimal number of sets for the set cover instance. Let our the number of sets we use using this algorithm be N, we have

$$\sum_{i=1}^{m} x_i = \mathsf{OPT}(LP) \leq \mathsf{OPT}$$

$$N = \sum_{i=1}^{m} \mathbb{I}_i x_i \leq \sum_{i=1}^{m} k x_i = k \sum_{i=1}^{m} x_i$$

where $\mathbb{I}_i = \begin{cases} \frac{1}{x_i} \text{ if } x_i \geq \frac{1}{k} \\ 0 \text{ otherwise} \end{cases}$. Therefore $N \leq k \cdot \mathsf{OPT}$.

Common misconception: It is not enough to note that at least one set must be used for each element in any solution, and show that at most k sets are used by our approximation. Since the same set may be used to cover many elements, number of sets per element is not a useful measure.

(b) The expected number of sets we take is

$$\sum_{i=1}^{m} p_i \leq \sum_{i=1}^{m} \beta \cdot x_i = \beta \sum_{i=1}^{m} x_i = \beta \cdot \mathsf{OPT}(LP) \leq \beta \cdot \mathsf{OPT}$$

.

(c) For an item u, let S_i be an arbitrary set that covers it. The probability that S_i is not chosen is $1 - p_i \leq e^{-p_i}$.

Therefore, the probability that no sets cover u is at most $\prod_{i:\, S_i \ni u}(1 - p_i)$. If one of the p_i is 1, then the probability is zero, and we're done. If none of the p_i is 1, that means $\beta \cdot x_i < 1$ for all $i : S_i \ni u$. Then we can bound the probability as follows.

$$\begin{aligned}
&\prod_{i:\, S_i \ni u} (1 - p_i) \\
&\leq \prod_{i:\, S_i \ni u} e^{-p_i} \\
&= \exp\left(- \sum_{i:\, S_i \ni u} p_i\right) \\
&= \exp\left(- \sum_{i:\, S_i \ni u} \min\{1, \beta \cdot x_i\}\right) \\
&= \exp\left(- \sum_{i:\, S_i \ni u} \beta \cdot x_i\right) \\
&= \exp\left(-\beta \sum_{i:\, S_i \ni u} x_i\right) \\
&\leq \exp(-\beta)
\end{aligned}$$

where the last inequality follows from the constraint $\forall u \in \mathcal{U}$, $\sum_{i:\, S_i \ni u} x_i \geq 1$ of the linear program.

(d) By union bound and our answer from part (c), the probability that there *exists* a element that is not covered by any set is $n \cdot e^{-\beta}$. (Alternatively you could use Markov's inequality since the expected number of uncovered elements is $n \cdot e^{-\beta}$).

If we set $\beta = \ln(10n)$, then the probability that an arbitrary item u is not covered by some set is at most $\frac{1}{10n}$. Hence the probability that there exists at least one uncovered item is at most $1/10$.

Blank scratch page.