CS170 Final

Name:

SID:

GSI and section time:

Write down the names of the students on your left and right as they appear on their SID.

Name of student on your left:

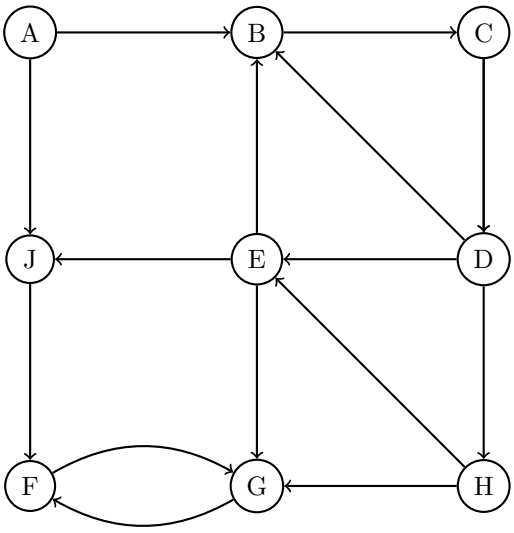
Name of student on your right:

Answer all questions. Read them carefully first. Be precise and concise. The number of points indicate the amount of time (in minutes) each problem is worth spending. Not all parts of a problem are weighted equally. Write in the space provided, and use the back of the page for scratch. By "reduction" in this exam it is always meant "polynomial-time reduction." Also, when you are asked to prove that a problem is **NP**-complete, no need to show that it is in **NP**, unless asked to do so.

Good luck!

1. **Strongly Connected Components** ***(6 points)***

For the directed graph below, list the strongly connected components **in the order in which they are output by the strongly connected components algorithm.**

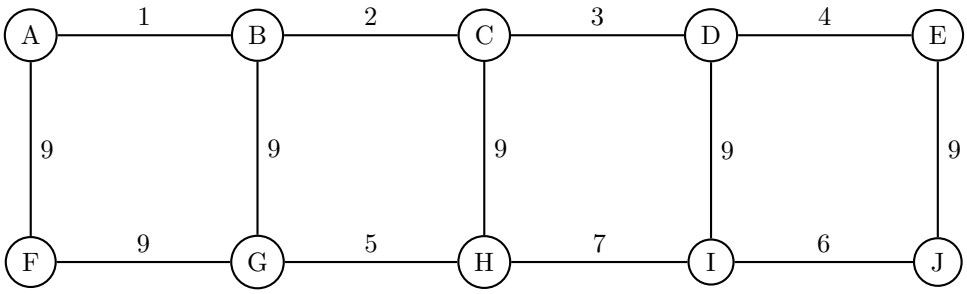


Solution:

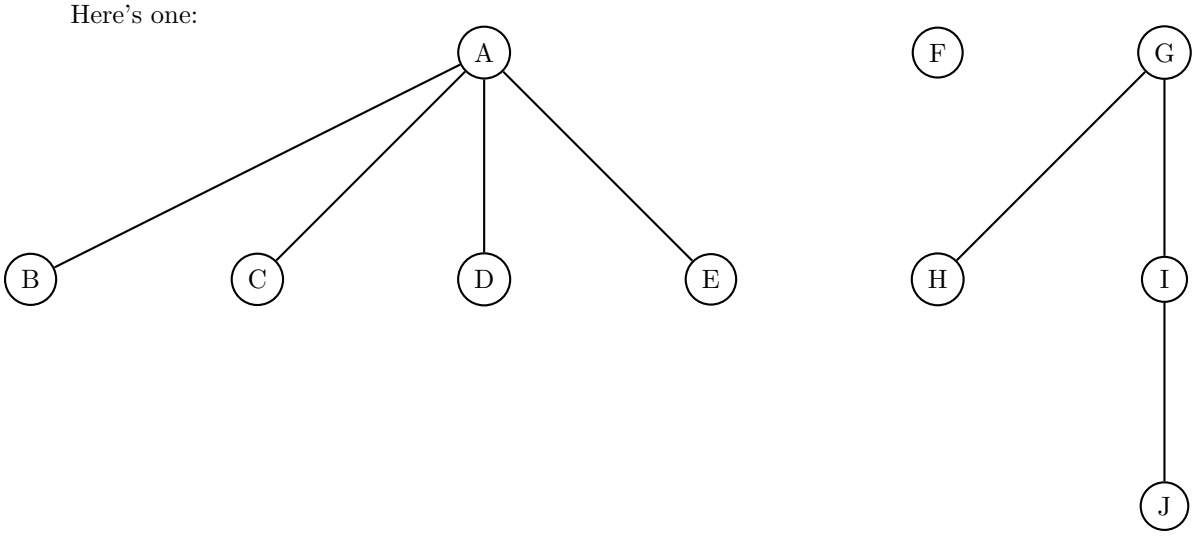
{F, G}, {J}, {B, C, D, E, H}, {A}

2. Kruskal's Algorithm *(6 points)*

Draw the state of the union-find data structure (union operations using rank, but without path compression) at the end of the $7th$ iteration of Kruskal's algorithm on the graph shown below.

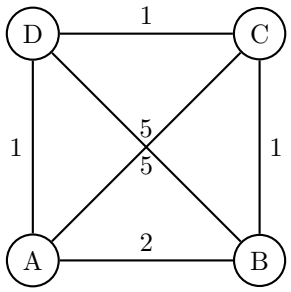


Solution: There's a couple of solutions, depending on tie-breaking in deciding the parent during the union process.
Here's one:



3. Traveling Salesman Problem *(10 points)*

In the lecture, we've learnt an approximation algorithm for traveling salesman problem based on computing MST and a depth first traversal. Suppose we run this approximation algorithm for Traveling Salesman Problem on the following graph:



The algorithm will return different tours based on the choices it makes during its depth-first traversal stage.

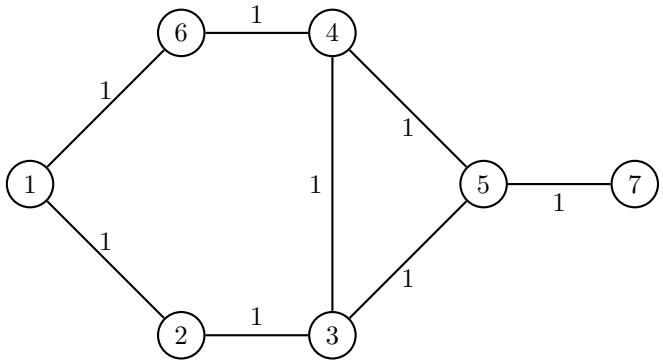
(a) *(3 points)* Which DFS traversal leads to the best possible output tour?
Solution:
A-D-C-B, **D-C-B-A**, **B-C-D-A**, or **C-D-A-B**
Explanation:
The MST is {(A,D),(D,C),(C,B)}. The optimal tour uses all of these edges.
For example, if we start traversing the tree at A, then the only traversal would be to follow the tree and go through the vertices in the order D, C, B
Another potential traversal would be to start at D, move to C, then B, and backtrack, before going to A.
Notice that if we started at D and moved to A first, then we would have to visit C next, and this would not lead to the best possible output tour (as per part (b)).
A common mistake was to answer with the optimal output tour, instead of the traversal.

(b) *(3 points)* Which DFS traversal leads to the worst possible output tour?
Solution:
C-B-D-A or **D-A-C-B**
See above explanation.
Notice that the worst possible tour overall, **D-C-A-B**, is not possible to be output by this algorithm, regardless of the DFS traversal used.

(c) *(4 points)* What is the approximation ratio given by the algorithm in the worst case for the above instance? Why is it worse than 2?
Solution:
$\frac{12}{5}$
From previous parts, the optimal tour length is 5 while the worst possible is 12. This is worse than 2 because our graph does not satisfy the triangle inequality, specifically $\ell(A, C) > \ell(A, B)+\ell(B, C)$ and$\ell(B, D) > \ell(A, B) + \ell(D, A)$, which is the necessary condition for our algorithm to guarantee an approximation ratio of 2.

4. Shortest Paths *(12 points)*

Consider the following graph on 7 vertices.



(a) *(4 points)* Suppose we execute Djikstra's algorithm to compute distances from the vertex 1. Djikstra's algorithm maintains an array $Dist[1, \ldots, n]$ where the i^{th} entry $Dist(i)$ will eventually contain the distance from vertex 1 to vertex i.

What is the order in which the following quantities "reach their final correct values"?

$$\{Dist(3), Dist(5), Dist(6), Dist(7)\}$$

Solution:

$$\{Dist(6), Dist(3), Dist(5), Dist(7)\}$$

(b) *(8 points)* The Floyd-Warshall algorithm maintains an array of pairwise distances $d(i, j)$ during its execution. During the execution, the $d(i, j)$ values keep decreasing until they reach their "correct value".

What is the order in which the following quantities "reach their correct values"?

$$\{d(1,2), d(1,3), d(1,4), d(1,5), d(2,6), d(1,7)\}$$

Solution:

$$\{d(1,2), d(2,6), d(1,3), d(1,5), d(1,7), d(1,4)\}$$

5. Multiplying Numbers *(6 points)*

The following algebraic identities can be used to design a divide-and-conquer algorithm for multiplying n-bit numbers.

$$\left(2^{2n/3}a_2 + 2^{n/3}a_1 + a_0\right)\left(2^{2n/3}b_2 + 2^{n/3}b_1 + b_0\right)$$
$$= 2^{4n/3} \cdot a_2b_2 + 2^n \cdot (a_1b_2 + a_2b_1) + 2^{2n/3} \cdot (a_2b_0 + a_0b_2 + a_1b_1) + 2^{n/3} \cdot (a_1b_0 + a_0b_1) + a_0b_0$$

$$(a_0b_2 + a_2b_0) = (a_0 + a_2) \cdot (b_0 + b_2) - a_0b_0 - a_2b_2$$
$$(a_0b_1 + a_1b_0) = (a_0 + a_1) \cdot (b_0 + b_1) - a_0b_0 - a_1b_1$$
$$(a_1b_2 + a_2b_1) = (a_1 + a_2) \cdot (b_1 + b_2) - a_1b_1 - a_2b_2$$

(a) *(4 points)* Write the recurrence relation for running time $T(n)$ of the algorithm.
Solution:
$$T(n) = 6T(n/3) + O(n)$$

(b) *(2 points)* What is the running time of the algorithm?
Solution:
$$T(n) \in O\left(n^{\log_3(6)}\right)$$

6. Reduction Essentials *(8 points)*

Assume A and B are search problems, and A reduces to B in polynomial time. In each part you will be given a fact about one of the problems. Determine what, if anything, this allows you to determine about the other problem. *Answer each part in one sentence.*

(a) A is in **P**.
Solution:
Nothing.

(b) B is in **P**.
Solution:
A is in **P**.

(c) A is **NP**-hard.
Solution:
B is **NP**-complete.

(d) B is **NP**-hard.
Solution:
Nothing.

7. Breaking Encryption *(5 points)*

After years of research, Horizon Wireless released an encryption algorithm E that encrypts an n-bit message in time $O(n^2)$.

Show that if $P = NP$ then this encryption algorithm can be broken in polynomial time. More precisely, argue that if $P = NP$, then the following decryption problem can be solved in polynomial time.

DECRYPT
Input: An encrypted message M (encrypted using the algorithm E)
Goal: Decryption of M.

Solution: Since E is a polynomial-time algorithm, the decryption algorithm D must be in NP. This follows because the polynomial-time checker for D on input M with solution x is to simply check if $E(x) = M$. Now assuming $P = NP$, D must have a poly-time algorithm.

8. Zero-Sum Games *(22 points)*

Alice and Bob are playing a zero-sum game whose payoff matrix is shown below. The ij^{th} entry of the matrix shows the payoff that Alice receieves if she plays strategy i and Bob plays strategy j. Alice is the row player and is trying to maximize her payoff.

Alice \ Bob	A	B
1	4	1
2	2	5

Now we will write a linear program to find a strategy that maximizes Alice's payoff.

Note: For parts (a)-(d), we also accepted the dual LP (if you solved for Bob's strategy rather than Alice's).

(a) *(2 points)* The variables of the linear program are x_1, x_2 where x_i denotes

Solution: The probability that Alice plays strategy i

and a variable p denoting Alice's payoff.

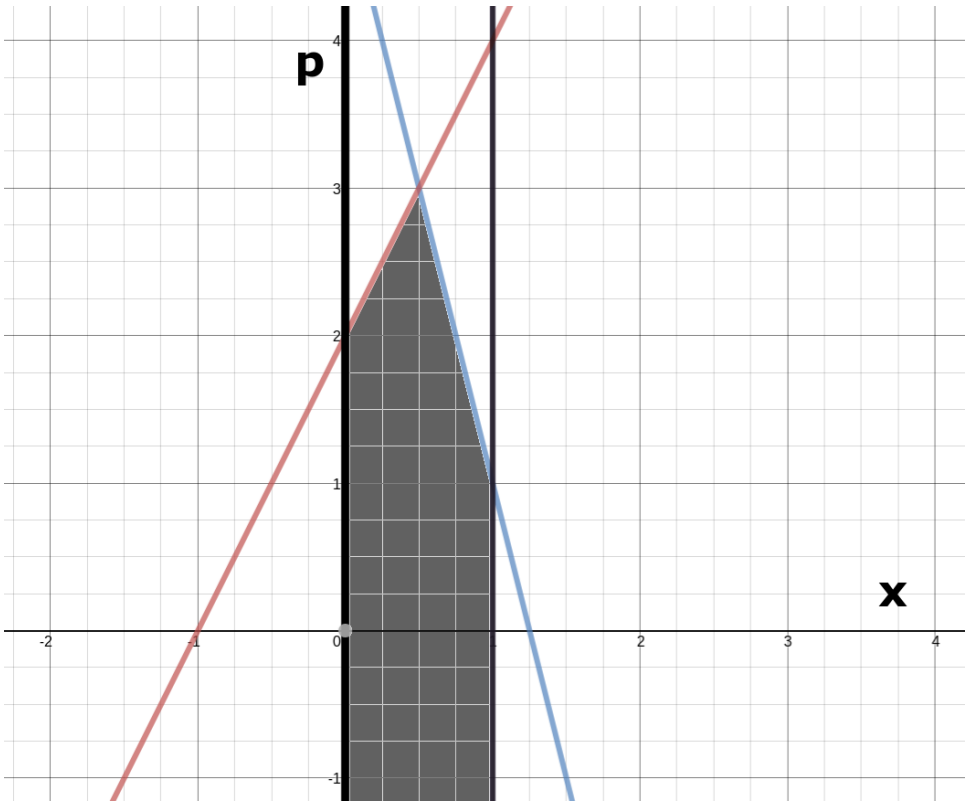
(b) *(4 points)* Write the linear program for maximizng Alice's payoff.
Solution:

$$\begin{aligned} \max\ & p \\ & p \leq 4x_1 + 2x_2 \\ & p \leq x_1 + 5x_2 \\ & x_i \geq 0 \\ & x_1 + x_2 = 1 \end{aligned}$$

(c) *(4 points)* Eliminate x_2 from the linear program and write it in terms of p and x_1 alone.
Solution:

$$\begin{aligned} \max\ & p \\ & p \leq 2x_1 + 2 \\ & p \leq -4x_1 + 5 \\ & x_1 \geq 0 \\ & x_1 \leq 1 \end{aligned}$$

(d) *(8 points)* Draw the feasible region of the above linear program in p and x_1.
Solution:



(e) *(4 points)* What is the optimal solution and what is the value of the game?
Solution:

$$x_1 = \frac{1}{2}$$
$$x_2 = \frac{1}{2}$$
$$p = 3$$

9. NP-search problems *(10 points)*

Which of the following are NP-search problems? Justify your answers.

(a) Longest Increasing Sequence
Input: A sequence of numbers $a_1, \ldots, a_n$.
Goal: Find the longest increasing subsequence of $a_1, \ldots, a_n$.

Solution:
Yes. We have discussed in class a polynomial time-algorithm for this problem; thus, the problem is in P, which is a subset of NP.
Specifically (this was not needed to be mentioned to receive full credit), we can verify any proposed solution by first checking that it indeed is an increasing subsequence, and then simply resolving the problem to find the check that it has the length of the longest possible subsequence.
Common Mistakes:

- There seems to be a prevalent misunderstanding that problems in P are not in NP. The opposite is true; even if $P \neq NP$, we would still have $P \subset NP$.
- Because of the above mistake, we did not accept answers that simply noted that the problem is in P, because it was unclear whether the student intended this as a "yes" or "no" to the question.
- The key thing we were looking for in the answer was what was stated above: the fact that the problem was solvable in polynomial-time. Actually discussing the algorithm and the details of how to solve the problem was insufficient and unneeded.
- Some students gave a verification procedure that checked if the output was an increasing subsequence, but failed to demonstrate how to check the "longest" part.

(b) Longest Path
Input: A directed graph $G = (V, E)$.
Goal: Find the longest directed path in G.

Solution:
No; this is an NP-Hard optimization problem. (We can show this problem to be NP-Hard by reducing the Undirected Longest Path problem, stated to be an NP-complete problem in the book, to this one).

10. Computing Diameter *(19 points)*

The diameter d of an undirected graph $G = (V, E)$ with unit lengths, is defined to be the maximum distance between any two vertices, i.e. $d = \max_{u,v \in V}(u, v)$, where $d(u, v)$ is the length of the shortest path between u and v.

Problem dropped

(a) *(5 points)* Give a simple algorithm to compute d in time $O(|V| \cdot (|V| + |E|))$.

(b) *(10 points)* Give a 2-approximate algorithm that runs in $O(|V| + |E|)$ time for approximately computing the diameter. Justify why the output of your approximation algorithm d_{approx} satisfies: $d \leq d_{approx} \leq 2d$. (*Hint: diameter is twice the radius*)

(c) *(4 points)* Give an example where: $d_{approx} = 2d$.

11. Assigning Backups *(20 points)*

Horizon Wireless is building a 5G network. The company has a set V of wireless towers; the distance $d(i, j)$ between any two of them is known, and each tower is capable of transmitting to other towers within a distance r.

- To make this network fault-tolerant, Horizon wants to assign each tower $v \in V$ to **two** other backup towers, so that if v is about to fail, it can transmit its data to them.
- Due to storage constraints, each tower can only serve as a backup for up to **three** other towers.

(a) *(10 points)* Suppose Horizon partitions its towers V into active towers A and backup towers B. Devise an algorithm which, given $V = A \cup B$, a distance function $d(i, j)$ on V, and tower radius r, assigns each active tower $a \in A$ to two backup towers in B (subject to the storage constraint), or reports that no assignment is possible. (*Hint: build a graph and use a known algorithm.*

Solution:

We want to reduce the problem to max flow. Create a directed graph with vertex set $\{s, t\} \cup V$. Create an edge from s to every vertex $a \in A$ with capacity 2. Create an edge from each $b \in B$ to t with capacity 3. Finally, for every pair of vertices (a, b) with $a \in A$, $b \in B$, create an edge from a to b with capacity 1 if and only if $d(a, b) \leq r$. The existence of an assignment is equivalent to the existence of a flow with value at least $2|A|$.

If there is a flow with value $2|A|$, then it must saturate all of the edges into each of the vertices $|V|$. We know from class that there is an integer-valued flow with this value. In particular, on every edge from a vertex in A to a vertex of B, the flow value is either 0 or 1. Therefore, this flow has exactly two outgoing edges from every vertex in A with nonzero flow. Furthermore, this flow assigns at most 3 active towers to any backup tower. Therefore, there is a valid assignment.

Conversely, suppose that there is an assignment of active towers to backup towers. Make a flow in the network we created by putting a flow of 1 on each edge corresponding to assigned pair (a, b) and 0 for all other pairs $(a, b), a \in A, b \in B$. For all incoming edges to A, assign 2 units of flow. For all outgoing edges from $b \in B$, assign the right flow to achieve flow conservation. This is an $s - t$ flow with value $2|A|$ because flow conservation is achieved at all vertices in A. This completes the proof of correctness.

Note that you didn't need to give this justification to receive full credit.

(b) *(10 points)* To use its network more efficiently, Horizon wants to use all towers in V as active towers, but still wants to assign each tower $v \in V$ to two other towers in V as backups. Again, no tower can be a backup for more than three other towers. Give an algorithm to find the assignment of backups for every tower.

Solution:

We split every vertex $v \in V$ into two vertices, v_a and v_b. We then run the algorithm from the previous part, passing in $\{v_a\}$ as A and $\{v_b\}$ as B.

12. NP-complete problems *(24 points)*

Prove that the following problems are NP-hard.

In each case, specify which problem you are reducing from, which problem you are reducing to. Briefly, but precisely describe how you transform an instance of one problem to another. *Proof of correctness of the reduction is not necessary.*

(a) *(8 points)* Directed Rudrata Cycle
Input: A directed graph $G = (V, E)$.
Goal: Find a directed cycle that visits every vertex in V exactly once.
Solution:
We reduce from Rudrata Cycle. To convert an undirected graph to a directed graph, we replace each undirected edge (u, v) with two directed edges, (u, v) and (v, u).

(b) *(8 points)* Californian Cycle

Input: A directed graph $G = (V, E)$ with each vertex colored *blue* or *gold*, i.e., $V = V_{blue} \cup V_{gold}$.
Goal Find a *Californian cycle* which is a directed cycle through all vertices in G that alternates between blue and gold vertices. (*Hint: Directed Rudrata Cycle*)

Not many responses received full credit.

Solution:
We reduce Directed Rudrata Cycle to Californian Cycle, thus proving the NP-hardness of Californian Cycle.

Given a directed graph $G = (V, E)$, we construct a new graph $G' = (V', E')$ as follows:

- For each $v \in V$, create a blue node v_b with an edge to a gold node v_g (in G').
- For each $(u, v) \in E$, add edge (u_g, v_b) to E'. Another way to view this is that for each node $v \in V$, we are redirecting all its incoming nodes to v_g, and all its outgoing nodes originate from v_b (in G').

Here are some common attempts that may look reasonable, but are incorrect:

i. Color the graph (does not matter how), then find a Californian cycle.
ii. For all $v \in V$, add a blue node $v' \in V'$. For every edge $(u, v) \in E$, add $(u, w) \in E'$ and $(w, v) \in E'$, where w is a new gold node.
iii. For all $v \in V$, create blue node v_b and gold node v_g in G'. For each edge $(u, v) \in E$, create edges (u_b, v_g) and (u_g, v_b) in G'.
iv. Same as the previous construction, but also add edges (v_g, v_b) and/or (v_b, v_g).
v. Make two copies of G: one blue, one gold, each one with the same edges as in G. Then add edges between corresponding nodes of opposite color.

(c) *(8 points)* 4-SAT

Input: n boolean variables $\{x_1, \ldots, x_n\}$ and clauses $\{C_1, \ldots, C_m\}$ with each having exactly *four distinct* literals. For example, the following is an instance of $4 - SAT$.

$$(x_1 \vee x_2 \vee \overline{x_4} \vee \overline{x_5}) \wedge (x_3 \vee \overline{x_4} \vee \overline{x_1} \vee x_2) \wedge (x_1 \vee x_3 \vee x_4 \vee x_5)$$

Note that all the 4 literals within a clause have to be distinct.

Goal: Find an assignment to the variables $x_1, \ldots, x_n$ that satisfies all the clauses.

Solution:

We reduce from 3-SAT to 4-SAT.

For the reduction, we create a new variable, y. We duplicate every clause in the 3-SAT input, adding y as a literal to one set and $\overline{y}$ as a literal the the other set. For example, for the 3-SAT clause, $(x_1 \vee x_2 \vee \overline{x_4})$, we create the two 4-SAT clauses $(x_1 \vee x_2 \vee \overline{x_4} \vee y) \wedge (x_1 \vee x_2 \vee \overline{x_4} \vee \overline{y})$. We pass these clauses to the 4-SAT solver.

If we had to prove this, we would note that if the solver set y to be true, then half of the clauses are immediately satisfied, and the other half become the input to the 3-SAT instance, so the assignment of x_i values must satisfy that input; this also happens if the solver set y to be false. Thus, in either case, the assignment of the x_i values must be a valid solution to the 3-SAT instance.

Alternate solution:

It was also possible to give a reasonable reduction from SAT to 4-SAT, in a parallel manner to the reduction of SAT to 3-SAT discussed in class.

However, this was much more tricky, both in writing the transformation of SAT clauses that had more than four literals, but also in correctly transforming clauses with fewer than four literals (in particular, something similar to the other solution was needed, to ensure that any newly added variables would not change the solution). No student gave a fully correct reduction, with this approach.

Common Mistakes:

- The most common mistake was thinking that the reduction could mandate the value of a variable in the 4-SAT instance (e.g. they create a new variable y and force the value to be false).
- Similarly, a common mistake was trying to put "False" directly into a clause, as a literal. That is not how clauses work.
- Conversely, some students tried adding a new variable to each clause and mandating the new variable to be *true*. They seem to have misunderstood how SAT works, as this would simply trivially satisfy all clauses, and any assignment of values to the x_i variables would result in a valid solution.
- The above reason is also why it was insufficient to simply add a new variable, and leave it at that; the 4-SAT solver could then simply set the new variable to be true, thus immediately satisfying all clauses.
- Some students indicated that they have not fully understood how generalizations work. The method of reducing by generalization involves taking an existing NP-Complete problem A, and showing that B is a generalization of A. Thus, we can write a reduction from A to B, utilizing this generalization. In this problem, although it is true that SAT is a generalization of 4-SAT, this does not grant us a way of reducing SAT to 4-SAT, only the other way around (which does not prove anything).

13. 3-Set Cover *(32 points)*

The 3-Set Cover problem is a special case of the Minimum Set Cover problem where every element appears in at most 3 sets. The formal definition of the problem is as follows.

3-Set Cover
Input: A universe of elements $\mathcal{U}$, a family of subsets $\{S_1, S_2, \ldots, S_m\}$ of $\mathcal{U}$ such that each element $u \in \mathcal{U}$ belongs to at most 3 subsets in $\{S_1, \ldots, S_m\}$.

Solution: Find the smallest collection of subsets that covers every element in the universe $\mathcal{U}$.

(a) *(10 points)* We will now show that 3-Set Cover is NP-hard. using the fact that Vertex Cover is NP-complete.

We will reduce the Vertex Cover problem to the 3-Set Cover problem.

Given an instance of the Vertex Cover problem, we construct an instance

3-Set Cover as follows.

The universe $\mathcal{U}$ consists of all the edges.

There is one set corresponding to every vertex.

Every element appears in at most 3-sets because each edge is connected two vertices.

The rest of the proof is straight-forward, and we skip it.

(b) *(6 points)* Now we will try to develop an approximation algorithm for 3-Set Cover using linear programming. First, let us try to model the problem using a linear program (our model will eventually only give us an approximation to the problem).

- There is one variable x_i for every set S_i, which denotes whether we pick S_i in the collection or not or the probability with which we pick S_i in the collection.

  Note: In order for the linear program to make sense and get credit for the rest of the questions your definition for the x_i's should be correct. Unfortunatelly, more than half of the class was not able to get any credit for parts: b, c, d, and e.

- What are the constraints?
  Solution:

  - for each $u \in \mathcal{U}$ such that $u \in S_i, S_j, S_k$: $\quad x_i + x_j + x_k \geq 1$
  - for $i = 1, \ldots, m$: $\quad 0 \leq x_i \leq 1$

- What is the objective function?
  Solution:
  We want to minimize the linear objective function: $\quad f(x_1, \ldots, x_m) = \sum_{i=1}^{m} x_i$

(c) *(3 points)* Suppose *LPOpt* denotes the optimal value for the linear program. Suppose *Opt* is the size of the smallest 3-Set Cover. Can $LPOpt > Opt$? Justify your answer.
Solution:
No. Because the optimal collection for the 3-Set Cover problem will be a feasible point of the linear program.

(d) *(3 points)* Can $LPOpt < OPT$? Justify your answer.

Solution:

Yes. For example let $S_1 = \{1, 2\}, S_2 = \{2, 3\}, S_3 = \{1, 3\}$ and $\mathcal{U} = \{1, 2, 3\}$. Then $OPT = 2$, while $LPOpt = 1.5$ for $x_1 = x_2 = x_3 = 0.5$.

(e) We will now prove that the following algorithm yields a 3-approximation for the problem.

`Solve the linear program described earlier.`
`For each set` S_i `do,`

- `Pick` S_i `if` $x_i \geq \frac{1}{3}$

The proof of correctness of the algorithm has two parts.

i. *(5 points)* Argue that every element in the universe $\mathcal{U}$ is covered.

Solution:

For each $u \in \mathcal{U}$ such that $u \in S_i, S_j, S_k$, either x_i or x_j or x_k will be greater or equal than $1/3$ because of the constraint $x_i + x_j + x_k \geq 1$. Hence we will pick either S_i or S_j or S_k and so we will cover u.

ii. *(5 points)* Argue that the number of sets picked by the algorithm is at most $3 \cdot LPOpt$.

Solution:

Let $\overline{x_i} = \begin{cases} 1 & \text{if } x_i \geq 1/3 \\ 0 & \text{otherwise} \end{cases}$. Then $\overline{x_i} \leq 3 \cdot x_i$, hence $\sum_{i=1}^{m} \overline{x_i} \leq 3 \cdot \sum_{i=1}^{m} x_i = 3 \cdot LPOpt$ and $\sum_{i=1}^{m} \overline{x_i}$ is the number of the sets picked by the approximation algorithm.

Note: Since $LPOpt \leq OPT$, we also have that $\sum_{i=1}^{m} \overline{x_i} \leq 3 \cdot OPT$.