CS 170 Final Solutions

Write in the following boxes clearly and then double check.

Name :	Oski
SID :	
Exam Room :	◯ RSF ◯ Other (Specify):
Name of student to your left :	
Name of student to your right :	

- **The exam will last 170 minutes.**
- The exam has 14 questions with a total of 179 points. You may be eligible to receive partial credit for your proof even if your algorithm is only partially correct or inefficient.
- Only your writing inside the answer boxes will be graded. **Anything outside the boxes will not be graded.** The last page is provided to you as a blank scratch page.
- Answer all questions. Read them carefully first. Not all parts of a problem are weighted equally.
- 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.
- The boxes 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. Unless the problem states otherwise, you should assume that graphs are simple.
- If you use any algorithm from lecture and textbook as a black box, 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.
- Assume the subparts of each question are **independent** unless otherwise stated.
- **Please write your SID on the top of each page; you will get 1 point for doing so.**
- **For multiple choice questions, please fill in the bubbles fully.**
- Good luck!

1 Runtime Analysis (5 points)

Consider the following snippet of code.

```
Function F(n) {

    if (n < 1) return 0

    integer count = 0

    for i = 1 to √n
    {
        count = count + 1
    }

    count = count + F(n/4)
    count = count + F(n/4)
    count = count + F(n/4)

    return count

}
```

Write a recurrence relation for $F(n)$.

Write the tightest upper bound for $F(n)$ (a bound that is asymptotically tight, up to constant factors).

Suppose we removed one of the "$count = count + F(n/4)$" lines from $F(n)$. Write the tightest upper bound for this modified $F(n)$ (a bound that is asymptotically tight, up to constant factors).

Suppose we removed *two* of the "$count = count + F(n/4)$" lines from $F(n)$ (so that only one of these lines remains). Write the tightest upper bound for this modified $F(n)$ (a bound that is asymptotically tight, up to constant factors).

Solution:

Exam continues on next page ⟹

(a) The recurrence is

$$F(n) = 3F(n/4) + \sqrt{n} = 3F(n/4) + n^{1/2}.$$

(b) By the Master Theorem, $F(n) = O(n^{\log_4 3})$

(c) The recurrence is now

$$F(n) = 2F(n/4) + \sqrt{n} = 2F(n/4) + n^{1/2}.$$

By the Master Theorem, $F(n) = O(n^{1/2} \log n) = O(\sqrt{n} \log n)$

(d) The recurrence is now

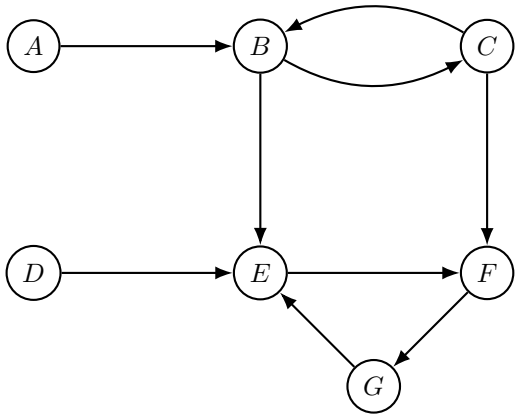
$$F(n) = F(n/4) + \sqrt{n} = F(n/4) + n^{1/2}$$

By the Master Theorem, $F(n) = O(n^{1/2}) = O(\sqrt{n})$

Exam continues on next page ⟹

2 Strongly Connected Components (8 points)

Suppose we wanted to compute the strongly connected components of the following graph.



To do so, we would run Kosaraju's algorithm, which works by computing two depth first searches. Write down the pre and post numbers computed in the first depth first search, breaking ties alphabetically.

vertex	pre	post
A		
B		
C		
D		
E		
F		
G		

Exam continues on next page ⟹

Next, write down the pre and post numbers computed in the second depth first search, breaking ties alphabetically.

vertex	pre	post
A		
B		
C		
D		
E		
F		
G		

Finally, for each number below, write down the vertices in the corresponding strongly connected component computed by Kosaraju's algorithm. Make sure that each strongly connected component is associated with the number that Kosaraju's algorithm assigns to it. (Note that the number of SCCs is possibly smaller than the number of rows provided in the table below.)

SCC number	Vertices in SCC
1	
2	
3	
4	
5	
6	

Solution: First round:

Exam continues on next page ⟹

vertex	pre	post
A	1	2
B	3	6
C	4	5
D	7	8
E	9	14
F	11	12
G	10	13

Second round:

vertex	pre	post
A	13	14
B	9	12
C	10	11
D	7	8
E	1	6
F	2	5
G	3	4

SCCs:

SCC number	Vertices in SCC
1	E, F, G
2	D
3	B, C
4	A
5	
6	

Exam continues on next page ⟹

3 Max Flow Potpourri (10 points)

Suppose $V = L \cup R$ is a minimum s-t cut in a directed graph $G = (V, E)$. Recall that in such a cut, s is in L and t is in R. In the following 4 questions, we will modify the capacity of an edge in the graph and consider how it affects the maximum flow from s to t. For each one, specify whether this modification will (1) necessarily increase the maximum flow, (2) necessarily decrease the maximum flow, (3) necessarily keep the maximum flow the same, or (4) it is not clear what effect it will have on the maximum flow. (Note that parts (c) and (d) below are not typos; they should indeed say "R to R".)

(a) We increase the capacity of an edge that goes from L to R.

◯ Increase ◯ Decrease ◯ Stay the same ◯ Not clear

(b) We decrease the capacity of an edge that goes from L to R.

◯ Increase ◯ Decrease ◯ Stay the same ◯ Not clear

(c) We increase the capacity of an edge that goes from R to R.

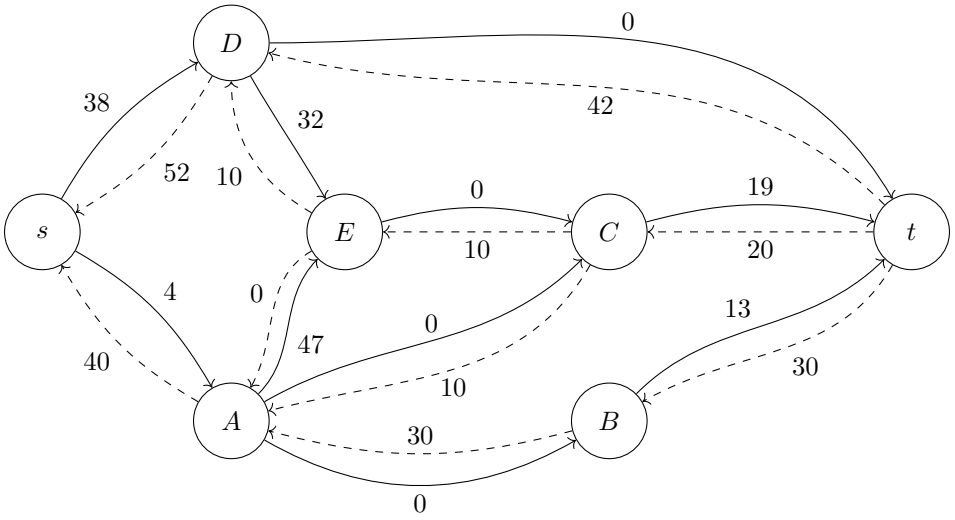
◯ Increase ◯ Decrease ◯ Stay the same ◯ Not clear

(d) We decrease the capacity of an edge that goes from R to R.

◯ Increase ◯ Decrease ◯ Stay the same ◯ Not clear

Exam continues on next page ⟹

The following 2 subparts are independent of the previous 4 subparts. Suppose we run the Ford-Fulkerson algorithm on a directed graph $G = (V, E)$ to compute the maximum flow from s to t, and it terminates with the following residual graph. The solid edges represent the edges in the original graph G and dashed edges represent the residual edges.



(e) Compute the value of the maximum flow from s to t.

(f) Compute a minimum s-t cut $V = L \cup R$ of the graph G. For each vertex $v \in V \setminus \{s, t\}$, write down whether it is contained in L or R.

Vertex	Which side of the cut?	Vertex	Which side of the cut?
A	◯ in L ◯ in R	D	◯ in L ◯ in R
B	◯ in L ◯ in R	E	◯ in L ◯ in R
C	◯ in L ◯ in R		

Solution:

(a) Not clear. There could be edges outside of this particular minimum cut $L \cup R$ that prevent the maximum flow from increasing. For example, there might be a second minimum cut in the graph.

Exam continues on next page ⟹

(b) Decrease

(c) Stay the same

(d) Not clear. Decreasing the capacity might cause the minimum cut to change, lowering the maximum flow, or it might not.

(e) The capacity of the residual edges entering s in the residual graph is $52 + 40 = 92$, which means the net flow existing s is equal to 92.

(f) A, E, D are in L; B and C are in R.

Exam continues on next page ⟹

4 Sat (5 points)

Consider an instance of the Sat problem which consists of the 10 Boolean variables $a, b, c, d, e, f, g, h, i, j$ and the following 10 clauses. (Note that not every variable necessarily appears in a clause.)

$$\begin{aligned}
&(a \vee \overline{c}) \\
\wedge\ &(\overline{a} \vee \overline{f} \vee \overline{g} \vee i) \\
\wedge\ &(\overline{b} \vee \overline{c} \vee \overline{d}) \\
\wedge\ &(\overline{b} \vee d \vee \overline{e} \vee \overline{f}) \\
\wedge\ &(c) \\
\wedge\ &(\overline{c} \vee e \vee \overline{g} \vee \overline{i}) \\
\wedge\ &(\overline{c} \vee \overline{f} \vee g) \\
\wedge\ &(\overline{d} \vee \overline{e} \vee \overline{f}) \\
\wedge\ &(\overline{b} \vee j) \\
\wedge\ &(f).
\end{aligned}$$

This instance is indeed satisfiable. Find a satisfying assignment with the minimum number of variables set to "T" (for True). For each variable, mark below whether this variable is assigned "T" or "F" in this assignment.

Variable	Assignment
a	
b	
c	
d	
e	
f	
g	
h	
i	
j	

Solution:

Exam continues on next page ⟹

Variable	Assignment
a	T
b	F
c	T
d	F
e	T
f	T
g	T
h	F
i	T
j	F

Exam continues on next page ⟹

5 An N-body problem (15 points)

Consider a scenario in astrophysics where we have N stars aligned along a 1-D grid. Specifically, for $i \in [N]$, the i-th star is at position i on this grid and has mass q_i. For each star in this system, we want to compute E_i, which denotes the net gravitational force exerted on star i by all the other stars in the system. Note that this force is directional and can be positive or negative.

The formula to calculate E_i for star i is as follows:

$$E_i = \sum_{j=1}^{i-1} \frac{q_i \cdot q_j}{(i-j)^2} - \sum_{j=i+1}^{N} \frac{q_i \cdot q_j}{(j-i)^2}.$$

In this formula, the first sum accounts for the gravitational attraction from stars to the left of star i, and the second sum considers the gravitational attraction from stars to the right of star i.

Your task is to design an algorithm that efficiently determines the E_i value for each $i \in [N]$. The algorithm should run in $O(N \log N)$ time. A proof of correctness and runtime analysis is not required.

Solution: Define polynomials $P(x) = \sum_{i=1}^{N-1} \frac{1}{i^2} x^i$, $Q(x) = \sum_{i=1}^{N} q_i x^i$ and the reversed polynomial

Exam continues on next page ⟹

$Q'(x) = \sum_{i=1}^{N} q_{N+1-i}x^i$. Use FFT to calculate the product polynomials $R(x) = P(x)Q(x)$ and $R'(x) = P(x)Q'(x)$. This takes $O(N \log N)$. Then we have

$$r_i = \sum_{j=0}^{i} q_j p_{i-j} = \sum_{j=1}^{i-1} \frac{q_j}{(i-j)^2}; \tag{1}$$

$$r'_i = \sum_{j=0}^{i} q'_j p_{i-j} = \sum_{j=1}^{i-1} \frac{q_{N+1-j}}{(i-j)^2} \Rightarrow r'_{N+1-i} = \sum_{j=i+1}^{N} \frac{q_j}{(j-i)^2}. \tag{2}$$

Therefore, we calculate $E_i = q_i(r_i - r'_{N+1-i})$ for $i \in [N]$. This takes $O(N)$. Overall, the runtime is $O(N \log N + N) = O(N \log N)$.

Exam continues on next page ⟹

6 Bakery (10 points)

Jon owns a bakery which produces cakes, cookies, and scones. Jon makes 10 dollars for each cake sold, 1 dollar for each cookie sold, and 2 dollars for each scone sold. However, Jon has a limited supply of ingredients: 200 units of flour, 100 units of sugar, and 50 eggs. Each of his products requires the following amounts of ingredients to make.

- One cake needs 5 units of flour, 2 units of sugar, and 5 eggs.
- One cookie needs 1 unit of flour, 2 units of sugar, and 1 egg.
- One scone needs 3 units of flour, 1 unit of sugar, and 2 eggs.

Given this information, help Jon maximize his profit, assuming that everything Jon makes gets sold.

(a) Write down the LP corresponding to this problem.

(b) Write down its dual.

Solution:

Exam continues on next page ⟹

(a) Let a be the number of cakes produced, b be the number of cookies produced, and c be the number of scones produced. Then the (Integer) LP is

$$\max 10a + b + 2c$$

subject to

$$\begin{aligned} 5a + b + 3c &\leq 200 \\ 2a + 2b + c &\leq 100 \\ 5a + b + 2c &\leq 50 \\ a, b, c, &\geq 0 \end{aligned}$$

where the first constraint is for flour, the second is for sugar, and the third is for eggs.

(b) Taking the dual (with variables d, e, and f)

$$\min 200d + 100e + 50f$$

subject to

$$\begin{aligned} 5d + 2e + 5f &\geq 10 \\ d + 2e + f &\geq 1 \\ 3d + e + 2f &\geq 2 \\ d, e, f &\geq 0 \end{aligned}$$

Exam continues on next page ⟹

7 Doom (10 points)

It is the end times. Our world is being turned into paperclips by a rogue AI, and you are coordinating the escape plan for humanity via rocket ships to Mars. What remains of the world can be represented by an undirected graph of n cities connected by m roads, with the i-th road taking h_i hours to traverse. Cities $1, 2, \ldots, k$ have rocket ships going to Mars. Devise an algorithm which computes how long it will take to travel from each city to the nearest city with a rocket ship. Your algorithm should run in time $O((n+m)\log n)$ for full credit. A proof of correctness and runtime analysis is not required.

Solution: Consider the obvious weighted graph, and add an additional vertex v_0 that has a weight 0 edge to each city with a rocket ship. Then, for any other city v, the distance of v to v_0 is exactly the amount of time it will take to travel from v to the nearest city with a rocket ship. Thus, to solve this problem, it suffices to run Dijkstra's with source v_0 to compute the distances of all vertices to v_0.

Exam continues on next page ⟹

The time complexity of this algorithm is that of Dijkstra's, which is $O((n+m)\log n)$.

Exam continues on next page ⟹

8 Knapsack (5 points)

Recall the knapsack with repetition problem. You have a knapsack with capacity W, and there are n items with weights $w_1, ..., w_n$ and values $v_1, ..., v_n$. Your goal is to determine the most valuable combination of items that can fit in the knapsack, and you are allowed to use each item multiple times. In class, we saw a dynamic programming algorithm for this problem using a 1-dimensional array K which has an entry $K[c]$ for all integers $0 \leq c \leq W$.

Suppose we are given an instance of this problem in which $n = 4$, $W = 13$, and the 4 items are given by $(1, 1), (5, 6), (8, 11), (10, 13)$, where the first coordinate of each item i specifies its weight w_i and the second coordinate specifies its value v_i. Compute the array K below.

c	$K[c]$
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	

Solution: We have the recurrence relation $K[c] = \max_{1 \leq i \leq n}\{K[c - w_i] + v_i\}$. Together with the base case $K[0] = 0$, we can solve:

Exam continues on next page ⟹

c	$K[c]$
0	0
1	1
2	2
3	3
4	4
5	6
6	7
7	8
8	11
9	12
10	13
11	14
12	15
13	17

Exam continues on next page ⟹

9 Maximum 3-Coloring (10 points)

Consider the following optimization version of the 3-coloring problem.

Input: An undirected graph $G = (V, E)$.

Solution: A coloring $c : V \to \{R, G, B\}$ which maximizes the number of bichromatic edges, i.e. the number of edges $(u, v) \in E$ such that $c(u) \neq c(v)$.

1. In this problem, we will devise an efficient, randomized algorithm which achieves a 2/3 approximation in expectation. In other words, if OPT is the optimal number of bichromatic edges, then your algorithm should output a coloring with at least $(2/3) \cdot OPT$ bichromatic edges in expectation. To begin, state your algorithm. (No runtime analysis necessary.)

2. Next, prove that your algorithm achieves a 2/3 approximation in expectation.

Solution: The algorithm is to color the graph with a uniformly random coloring $c : V \to \{R, GB\}$.

Exam continues on next page ⟹

In other words, independently color each vertex $v \in V$ with a color $c(v)$ chosen uniformly at random from the set $\{R, G, B\}$.

To analyze this, for a given edge $e = (u, v)$, let X_e be the indicator that c properly colors the edge e, meaning that $c(u) \neq c(v)$. Then

$$E[X_e] = Pr[X_e = 1] = Pr[c(u) \neq c(v)] = \frac{2}{3}.$$

This is because no matter what color $c(u)$ is chosen for vertex u, the odds that $c(v)$ is chosen to be a different color is $2/3$. Then the total number of bichromatic edges is $\sum_{e \in E} X_e$. We can therefore compute its expectation by linear of expectation:

$$E[\sum_{e \in E} X_e] = \sum_{e \in E} E[X_e] = \sum_{e \in E} \frac{2}{3} = \frac{2}{3}|E| \geq \frac{2}{3}OPT,$$

where the last step uses the fact that OPT is always less than $|E|$.

Exam continues on next page ⟹

10 k-wise Independent Hashing (15 points)

Let $\mathcal{H} = \{h_1, \ldots, h_m\}$ be a hash family in which each h_i maps the domain D into the range R. Recall that $\mathcal{H}$ is *k-wise independent* if for all distinct $x_1, \ldots, x_k \in D$ and all (not necessarily distinct) $a_1, \ldots, a_k \in R$,

$$\Pr_{h \sim \mathcal{H}}[h(x_1) = a_1 \wedge \cdots \wedge h(x_k) = a_k] = \frac{1}{|R|^k}.$$

In this question, we will be considering hash families with domain $D = \{x, y, z\}$ and range $R = \{0, 1, 2\}$.

1. The following table specifies a hash family $\mathcal{H} = \{h_1, \ldots, h_9\}$. For example, the i-th entry in the first row is equal to $h_i(x)$, and similarly for the other two rows. However, this table is missing some entries in the final row.

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9
x	0	0	0	1	1	1	2	2	2
y	0	1	2	0	1	2	0	1	2
z	0		2						1

Fill in the missing entries so that $\mathcal{H}$ is a pairwise (i.e. 2-wise) independent hash family.

2. True or false: the hash family you constructed in part (1) is 3-wise independent.

◯ True ◯ False

3. If $\mathcal{H} = \{h_1, \ldots, h_m\}$ is a hash family with $h_i : \{x, y, z\} \to \{0, 1, 2\}$, what is the smallest that m can be for $\mathcal{H}$ to be pairwise (i.e. 2-wise) independent?

4. If $\mathcal{H} = \{h_1, \ldots, h_m\}$ is a hash family with $h_i : \{x, y, z\} \to \{0, 1, 2\}$, what is the smallest that m can be for $\mathcal{H}$ to be 3-wise independent?

Solution:

Exam continues on next page ⟹

1.

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9
x	0	0	0	1	1	1	2	2	2
y	0	1	2	0	1	2	0	1	2
z	0	1	2	1	2	0	2	0	1

2. False. For example,

$$\Pr_{h\sim\mathcal{H}}[h(0) = 0 \wedge h(1) = 0 \wedge h(2) = 0] = \frac{1}{9} \neq \frac{1}{3^3}.$$

3. 9. Part (1) shows that a pairwise independent $\mathcal{H}$ is achievable with $m = 9$. To see that you can't do better than 9, note that we need $Pr[h(x) = a \wedge h(y) = b] = 1/9$ for all $a, b \in \{0, 1, 2\}$. That means there must exist an $h \in \mathcal{H}$ such that $h(x) = a \wedge h(y) = b$ for all $a, b \in \{0, 1, 2\}$, and since there are 9 ways to pick a and b we need at least 9 h's.

4. 27. The same argument as in part (3) shows that 27 is necessary. It's also sufficient, by considering the $\mathcal{H}$ which contains every single function $h : \{x, y, z\} \to \{0, 1, 2\}$.

Exam continues on next page ⟹

11 Max Coverage (15 points)

Consider the following variant of the Set Cover problem.

Input: A universe of n elements $U = \{1, \ldots, n\}$ and subsets $S_1, \ldots, S_m \subseteq U$ such that $\cup_{i=1}^{m} S_i = U$, as well as an integer k.

Solution: A collection of k subsets covering as much of U as possible, i.e. $J \subseteq \{1, \ldots, m\}$ such that $|J| = k$ and $\cup_{i \in J} S_i$ contains the maximum number of elements possible.

This problem is NP-hard, so we will design an approximation algorithm for it. Consider the following greedy strategy.

Greedy: Repeat until k subsets have been selected: pick the subset containing the largest number of uncovered elements.

We will show that this algorithm is achieves a constant-factor approximation ratio. To do so, let us define the following three quantities.

- OPT = the number of elements covered by the optimum solution.
- For each $1 \leq i \leq k$, x_i = the number of *new* elements covered by Greedy in the i-th step.
- For each $1 \leq i \leq k$, $z_i = OPT - x_1 - \cdots - x_i$. In addition, $z_0 = OPT$.

1. First, prove that $x_{i+1} \geq z_i/k$ for all $1 \leq i \leq k$.

2. Next, use part (1) to prove that $z_i \leq (1 - \frac{1}{k})^i \cdot OPT$ for all $0 \leq i \leq k$. (Hint: induction.)

Exam continues on next page ⟹

3. Finally, use part (2) to prove that Greedy is an α-approximation algorithm, where α is a constant independent of k. You should try to prove the largest value of α that you can that holds for all values of k. You may use the following inequality from class: for any real number t, $1 - t \leq e^{-t}$.

Solution:

1. Let C be the elements covered by the optimal solution. Note $|C| = OPT$. After i steps, our algorithm has covered $x_1 + \cdots + x_i$ elements, meaning at least z_i elements in C remain uncovered. But C is completely covered by k subsets, so these remaining z_i elements must be covered by k subsets as well. Hence one of these subsets must cover at least z_i/k elements. Since Greedy will pick the subset that covers the most elements, $x_{i+1} \geq z_i/k$.

2. Proof by induction. The base case of $i = 0$ is true because $z_0 = OPT$ by definition.

   For the induction step, let us assume it is true for i, and we will show it is true for $i + 1$. By definition, we know that $z_{i+1} = z_i - x_{i+1}$. By part (1),

   $$z_{i+1} \leq z_i - z_i/k = z_i(1 - 1/k).$$

   Then, by the inductive hypothesis

   $$z_i(1 - 1/k) \leq (1 - \tfrac{1}{k})^i \cdot OPT \cdot (1 - 1/k) = (1 - \tfrac{1}{k})^{i+1} \cdot OPT.$$

3. By part (2) and the inequality provided,

   $$z_k \leq (1 - 1/k)^k OPT \leq (e^{-1/k})^k OPT = e^{-1} OPT = \frac{1}{e} OPT.$$

   Now, recalling that $z_k = OPT - x_1 - \cdots - x_k$, we get that

   $$x_1 + \cdots + x_k \geq OPT - \frac{1}{e} OPT = (1 - \frac{1}{e}) OPT.$$

   Since $x_1 + \cdots + x_k$ is the number of elements covered by our algorithm, it achieves an approximation ratio of $\alpha = 1 - \frac{1}{e} \approx 0.632$.

Exam continues on next page ⟹

12 NP-Completeness Reductions (30 points)

You may assume that the following problems are NP-complete: Rudrata (Hamiltonian) Path, Rudrata (Hamiltonian) Cycle, Vertex Cover, Independent Set, 3-Coloring, and 3-SAT. (We only saw the optimization version of Vertex Cover in class. For this problem, you are allowed to use the variant of Vertex Cover in which you are given an integer k as part of the input and asked to find a vertex cover of size at most k.)

For both of the following problems, fill in the details of the proof that they are NP-complete.

1. **Feedback Vertex Set:** Given a directed graph $G = (V, E)$, a subset $S \subseteq V$ of the vertices is called a *feedback vertex set* if removing the vertices in S from G as well as the edges adjacent to them results in a graph with no directed cycles.

   Here is the **Feedback Vertex Set** problem.

   Input: A directed graph $G = (V, E)$ and an integer k.

   Solution: A feedback vertex set $S \subseteq V$ of size $|S| \leq k$.

   Proof. It is clear that the Feedback Vertex Set problem is in NP. Now we will use a polynomial time reduction to show that the problem is indeed NP-complete.

   Given an instance Φ of the problem [ ] *we will construct an instance Ψ of the problem* [ ] *as follows.*

   [ ]

   The proof that this is a valid reduction is as follows:

   [ ]

Exam continues on next page ⟹

2. **Bipartite Subgraph:** Given an undirected graph $G = (V, E)$, a subset of the vertices $S \subseteq V$ is called a *bipartite subgraph* if removing the vertices in $V \setminus S$ from G as well as the edges adjacent to them results in a graph which is bipartite.

   Here is the **Bipartite Subgraph** problem.

   Input: An undirected graph $G = (V, E)$ and an integer k.

   Solution: A bipartite subgraph $S \subseteq V$ of size $|S| \geq k$.

   Proof. It is clear that the Bipartite Subgraph problem is in NP. Now we will use a polynomial time reduction to show that the problem is indeed NP-complete.

   Given an instance Φ *of the problem* [ ] *we will construct an instance* Ψ *of the problem* [ ] *as follows.*

   [ ]

   The proof that this is a valid reduction is as follows:

   [ ]

Solution:

1. Given an instance $G = (V, E)$ and k of the problem **Vertex Cover**, we will construct an instance $G' = (V, E')$ and k of the problem **Feedback Vertex Set** as follows. Gor each edge $(u, v) \in E$ in G, we will add the two directed edges $u \to v$ and $v \to u$ to E'. This constructs G'.

   Proof that this works. For every edge $e = (u, v)$ in the original graph, there is a length 2 cycle between u and v in G'. This means that at least one of its endpoints must be in the feedback

Exam continues on next page ⟹

vertex set. Thus, a vertex cover in G of size $\leq k$ corresponds to a feedback vertex set in G' of size $\leq k$, and vice versa.

2. Given an instance $G = (V, E)$ and k of the problem **Independent Set**, we will construct an instance $G' = (V \cup U, E')$ of the problem **Bipartite Subgraph** as follows. We take the graph $G = (V, E)$, add n new vertices U, and then connect each of these new vertices to each vertex in V.

   Proof that this works. The largest bipartite subgraph in G' will always consist of the largest independent set in G as well as all the vertices in U. Hence, G will contain an independent set of size at least k if and only if G' contains a bipartite subgraph of size at least $n + k$. And any bipartite subgraph of G' of size at least $n + k$ can be converted into an independent set in G of size k by removing the vertices in U.

Exam continues on next page ⟹

13 Feedback Edge Set (20 points)

In the last problem, we saw that the Feedback Vertex Set problem is NP-complete on directed graphs. In this problem, we will see that the Feedback *Edge* Set problem is in P on undirected graphs.

Given an undirected graph $G = (V, E)$, a subset $S \subseteq E$ of the edges is called a *feedback edge set* if removing the edges in S from G results in a graph with no undirected cycles. Here is the **Minimum Feedback Edge Set** problem.

Input: An undirected graph $G = (V, E)$ and weights on the edges $\{w_e \mid e \in E\}$.

Solution: A feedback edge set $S \subseteq E$ of minimum total weight $\sum_{e \in S} w_e$.

Design an algorithm for the Minimum Feedback Edge Set problem. Your algorithm should run in time polynomial in the number of vertices $n = |V|$. You may assume for simplicity that the graph G is connected. No proof of correctness or runtime analysis is required. (Hint: think about what type of graph is left after the edges in S are removed.)

Exam continues on next page ⟹

Solution: After all of the edges in a feedback edge set are removed, G becomes an acyclic undirected graph. This means that it is a collection of trees. Because we want to minimize the total weight of edges removed, we will want to maximize the total weight of the edges in this collection of trees. Therefore, we will want this collection of trees to be connected, which means it should be a spanning tree. So, our algorithm is to compute the *maximum* (not minimum!) spanning tree T, and return the edges that are *not* in T as our set S.

To find the maximum spanning tree, one can negate all of the edges in the graph and run an MST algorithm such as Kruskal's algorithm.

Exam continues on next page ⟹

14 Dinner Party (20 points)

You are planning a dinner party for n of your friends. You have a long table with n seats from left to right, and you need to decide what order to seat your friends in. Some pairs of friends want to talk to each other, and this creates an awkward situation for anyone sitting between them. The awkwardness that a particular friend experiences is equal to the total number of pairs who want to talk to each other that they are sitting between. You want to devise an ordering of your friends so that the maximum awkwardness felt by any one of your friends is as small as possible.

Formally, let $G = (V, E)$ be an undirected graph with $n = |V|$ vertices. Given an ordering of the vertices $v_1, \ldots, v_n$, the *awkwardness* that vertex v_j feels is given by the following quantity:

$$\mathsf{Awk}(v_j) = |\{(v_i, v_k) \in E \mid i < j < k\}|.$$

Then the *maximum awkwardness* is given by $\max_{1 \le j \le n} \mathsf{Awk}(v_j)$. Your goal is to compute the smallest possible maximum awkwardness over all possible orderings $v_1, \ldots, v_n$ of the vertices. There is a trivial algorithm which enumerates over all of the $n!$ possible orderings; your goal will be to outperform this algorithm. Describe a dynamic programming algorithm to solve this problem which runs in time $O(\text{poly}(n) \cdot 2^n)$. (Here, poly$(n)$ refers to any function which is polynomial in n.)

1. What are the subproblems? (precise and succinct definition needed)

2. What is the recurrence relation?

3. What is the runtime?

Exam continues on next page

Solution: Subproblems: There is a subproblem $C[S]$ for every subset of the vertices $S \subseteq V$. Define an S-first ordering to be one which puts the vertices in S first and then the vertices in $V \setminus S$ second. Then

$$C[S] = \min_{S\text{-first orderings}} \{\max_{v \in S} \{\mathsf{Awk}(v)\}\}.$$

In other words, $C[S]$ is the minimum maximum awkwardness over all orderings which put the vertices in S first and then the vertices in $V \setminus S$ second, except you only compute the awkwardness of the vertices in S.

The recurrence relation for $C[S]$ involves choosing which vertex in S is the rightmost under the ordering.

$$C[S] = \min_{v \in S} \max(\#\text{ of edges that go from } S \setminus \{v\} \text{ to } V \setminus S, C[S \setminus v])$$

The number of subproblems is 2^n. For each subproblem, we have to loop over all vertices $v \in S$, and then count the number of edges that cross v. Thus, each subproblem takes at most time $O(|V||E|) = O(n^3)$, for a total runtime of $O(n^3 2^n)$.

This page left intentionally blank
for scratch purposes.
This page **will not be graded**.

DO NOT DETACH THIS PAGE.