COS 226 **Algorithms and Data Structures** Fall 2025

Final

This exam has 15 questions worth a total of 75 points. You have 180 minutes to complete it.

Instructions. This exam is preprocessed by computer. Write neatly, legibly, and darkly. Put all answers (and nothing else) inside the designated answer spaces. *Fill in* bubbles and checkboxes completely: ● and ■. To change an answer, erase it completely and redo.

Resources. The exam is closed book, except that you may use a one-page reference sheet (8.5-by-11 paper, both sides, in your own handwriting). No electronic devices are permitted.

Honor Code. This exam is governed by Princeton's Honor Code. Discussing the contents of this exam before the solutions are posted is a violation of the Honor Code.

Please complete the following information now.

Name:

NetID:

Exam room: ○ Friend 101 ○ CS 104 ○ CS 105 ○ Other

Precept:

P01	P01A	P02	P02A	P02B	P03A	P03B	P04
○	○	○	○	○	○	○	○

"I pledge my honor that I will not violate the Honor Code during this examination."

Signature

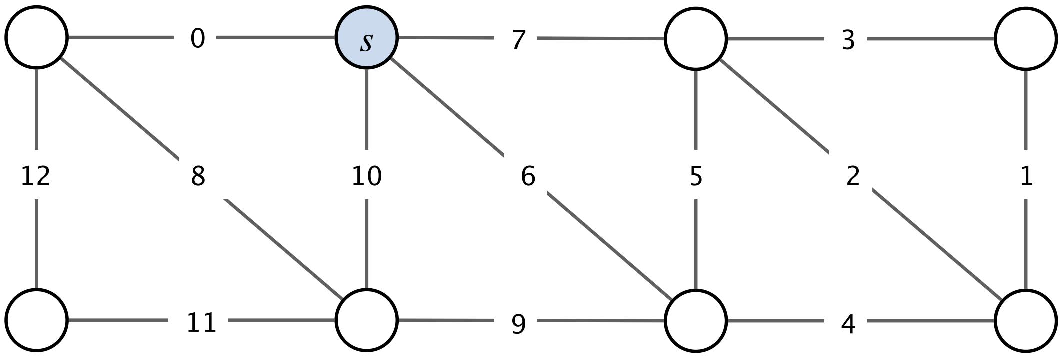
1. **Initialization. (1 point)**

   On the front page of the exam, in the spaces provided,

   - write your name and NetID;
   - fill in the bubbles for your exam room and official precept; and
   - write out and sign the Honor Code pledge.

2. **Minimum spanning trees. (5 points)**

   Consider the following edge-weighted graph (with distinct edge weights):

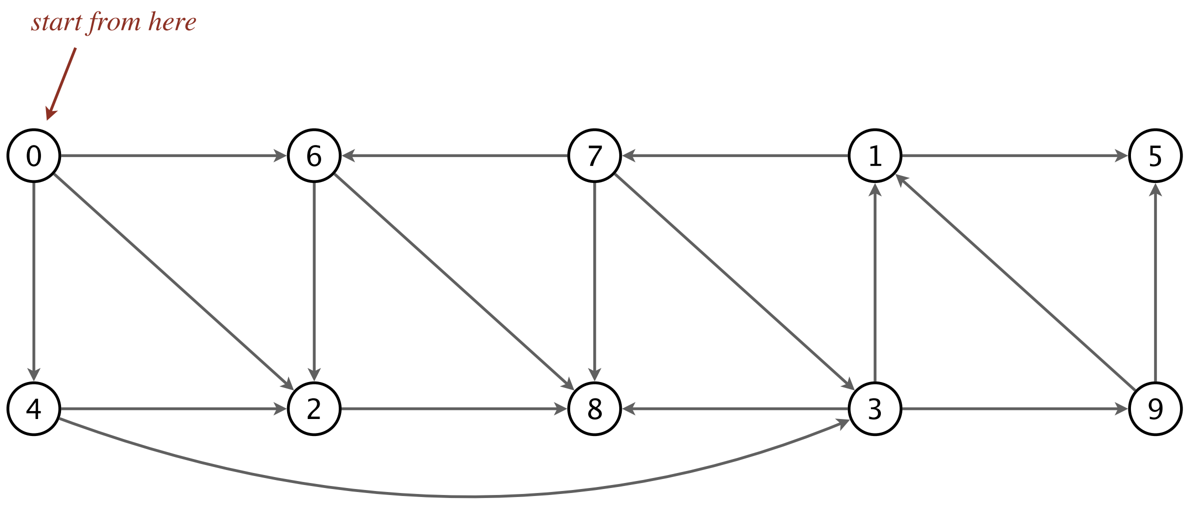


   (a) List the weights of the edges that *Kruskal's algorithm* adds to form an MST, in the order they are added.

   (b) Start *Prim's algorithm* from vertex s. List the weights of the edges that Prim's algorithm adds to form an MST, in the order they are added.

3. **Depth-first search. (5 points)**

Run *depth-first search* on the following digraph, starting from vertex 0. Assume adjacency lists are in sorted order: for example, when exploring edges leaving vertex 0, consider 0 → 2 before 0 → 4 or 0 → 6.



(a) List the 10 vertices in the order they appear in *DFS preorder*.

0 ___ ___ ___ ___ ___ ___ ___ ___ ___

(b) List the 10 vertices in the order they appear in *DFS postorder*.

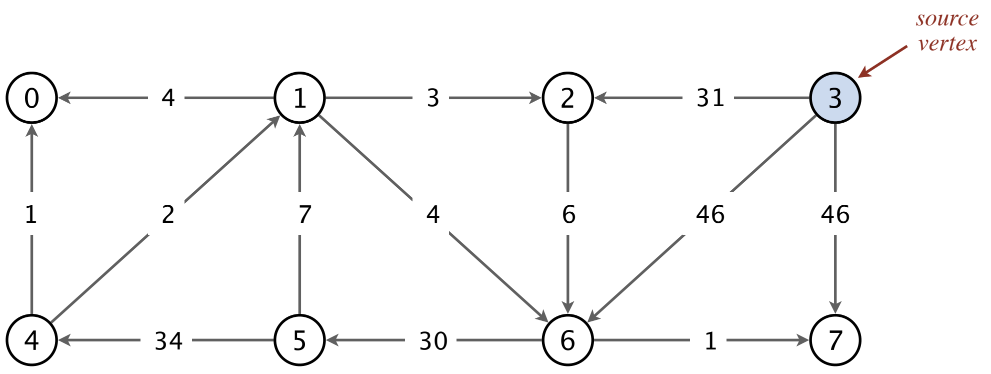
___ ___ ___ ___ ___ ___ ___ ___ ___ 0

(c) At the moment when `dfs(G, 6)` is called, for how many vertices (including vertex 6) have calls to `dfs(G, v)` been made but not yet finished?

○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8

4. **Shortest paths. (5 points)**

Consider running the *Bellman–Ford algorithm* on the edge-weighted digraph below, with source vertex $s = 3$. Assume that in each pass, the edges are processed in the order shown in the table at right (lexicographic order by endpoints).



edges	
$1 \to 0$	
$1 \to 2$	$4 \to 0$
$1 \to 6$	$4 \to 1$
$2 \to 6$	$5 \to 1$
$3 \to 2$	$5 \to 4$
$3 \to 6$	$6 \to 5$
$3 \to 7$	$6 \to 7$

After one pass over all the edges, the `distTo[]` values are as follows:

v	0	1	2	3	4	5	6	7
`distTo[v]`	∞	∞	**31**	**0**	∞	**76**	**46**	**46**

(a) What are the `distTo[]` values at the end of the next (*second*) pass?
Write the eight values in the table below.

v	0	1	2	3	4	5	6	7
`distTo[v]`								

(b) Which vertices will have their `distTo[]` values decreased during the *third* pass?
Fill in all checkboxes that apply.

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7

5. **Hash tables. (4 points)**

   Consider a *linear-probing hash table* with capacity 10 that neither grows nor shrinks. Answer parts (a)–(c) *independently*. For each part, assume the hash table initially contains the 6 keys in the positions shown below.

index	0	1	2	3	4	5	6	7	8	9
key	S			E	N	M			C	O

   (a) Assume $hash(X) = 5$. If you insert X into the hash table, at which *index* will it end up?

   ○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9

   (b) Assume $hash(Y) = 8$. If you insert Y into the hash table, at which *index* will it end up?

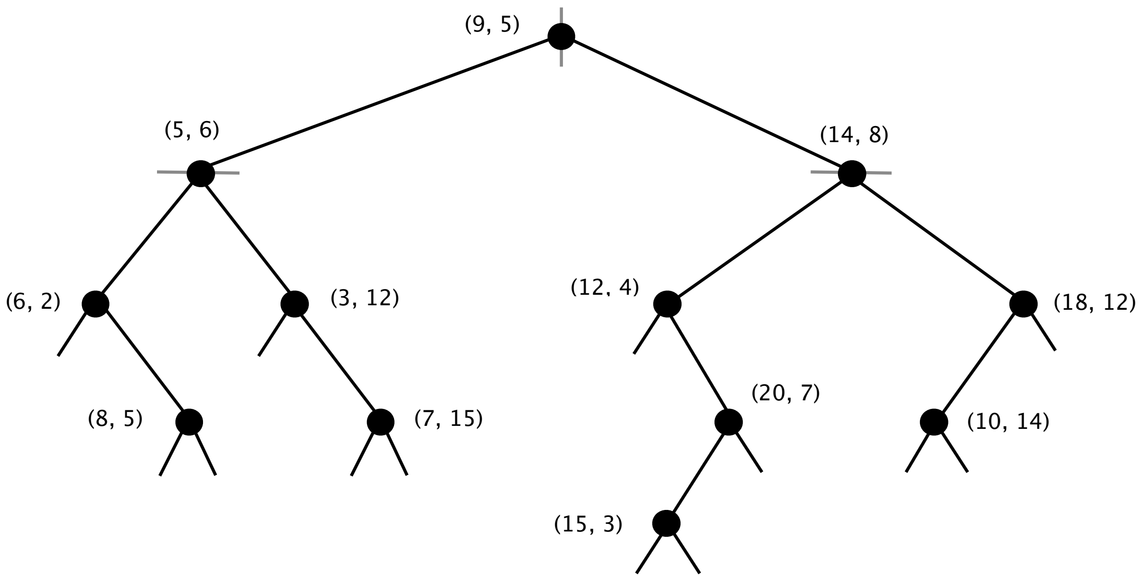
   ○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9

   (c) Assume $hash(Z)$ is equally likely to be any integer from 0 to 9, inclusive. If you insert Z into the hash table, what is the *probability* that it will end up at index 6?

   ○ 0 ○ $\frac{1}{10}$ ○ $\frac{2}{10}$ ○ $\frac{3}{10}$ ○ $\frac{4}{10}$ ○ $\frac{5}{10}$ ○ $\frac{6}{10}$ ○ $\frac{7}{10}$ ○ $\frac{8}{10}$ ○ $\frac{9}{10}$ ○ 1

6. **Kd-trees. (4 points)**

   Consider the following *2d-tree*:



(a) Suppose you insert the point $(10, 1)$. Where will it be inserted in the tree?

○ left child of $(3, 12)$ ○ left child of $(12, 4)$ ○ left child of $(15, 3)$ ○ right child of $(15, 3)$ ○ right child of $(20, 7)$

(b) Suppose you perform a *range search* for all points with $4 \leq x \leq 8$ and $10 \leq y \leq 20$. Which 2d-tree *nodes* are explored during the range search?

Fill in all checkboxes that apply.

☐ (3, 12) ☐ (5, 6) ☐ (7, 15) ☐ (8, 5) ☐ (9, 5) ☐ (12, 4) ☐ (18, 12)

7. **Analysis of algorithms. (5 points)**

Consider the following graph-processing code:

```
double[] distTo = new double[digraph.V()];
DirectedEdge[] edgeTo = new DirectedEdge[digraph.V()];
for (int v = 0; v < digraph.V(); v++) {
    distTo[v] = Double.INFINITY;
}
distTo[s] = 0.0;

for (int i = 1; i < digraph.V(); i++) {
    for (int v = 0; v < digraph.V(); v++) {
        for (DirectedEdge e : digraph.adj(v)) {
            relax(e);    // constant time
        }
    }
}
```

For this question, assume that

- `digraph` is of type `EdgeWeightedDigraph`,
- all vertices are reachable from `s`, and
- the digraph is *simple* (it has no parallel edges and no self-loops).

(a) Suppose that the `EdgeWeightedDigraph` data type is implemented using the *adjacency-lists representation.* What is the order of growth of the worst-case running time of the code as a function of the number of vertices V and edges E?

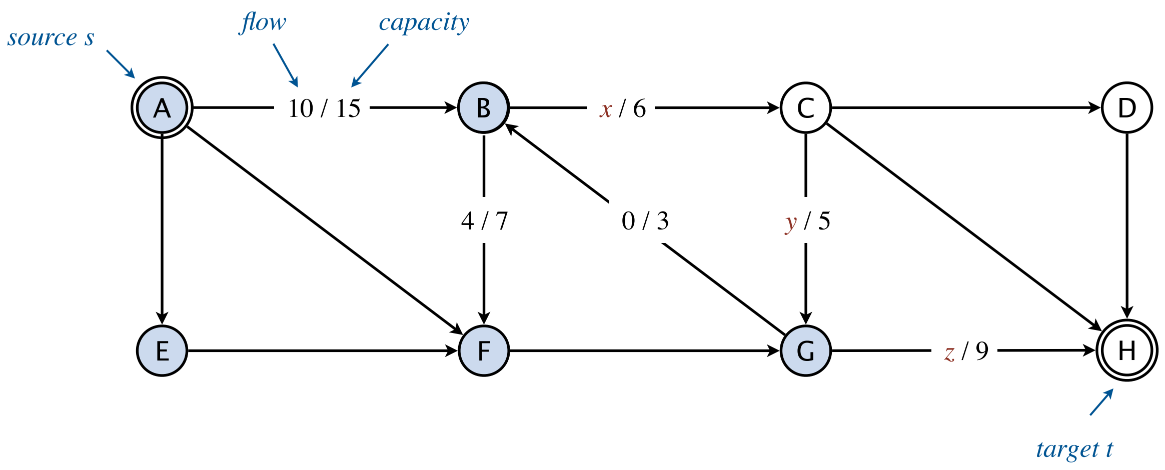
○ $\Theta(E+V)$ ○ $\Theta(V^2)$ ○ $\Theta(EV)$ ○ $\Theta(E^2)$ ○ $\Theta(V^3)$

(b) Repeat part (a), but now assume that `EdgeWeightedDigraph` is implemented using the *adjacency-matrix representation.*

○ $\Theta(E+V)$ ○ $\Theta(V^2)$ ○ $\Theta(EV)$ ○ $\Theta(E^2)$ ○ $\Theta(V^3)$

8. **Maxflows and mincuts. (5 points)**

Consider the following flow network and a *maxflow* f^*. On some edges, both the flow and the capacity are shown; on others, only the capacity is shown; on the remaining edges, neither is shown. A *mincut* with $S^* = \{A, B, E, F, G\}$ is highlighted in the diagram.



(a) What is the *capacity* of the mincut S^*?

○ 10 ○ 11 ○ 14 ○ 15 ○ 16 ○ 19 ○ 20

(b) What is the *value* of the maxflow f^*?

○ 10 ○ 11 ○ 14 ○ 15 ○ 16 ○ 19 ○ 20 ○ *can't be determined*

(c) What is x, the amount of flow on edge $B \to C$?

○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ *can't be determined*

(d) What is y, the amount of flow on edge $C \to G$?

○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ *can't be determined*

(e) What is z, the amount of flow on edge $G \to H$?

○ 0 ○ 1 ○ 2 ○ 3 ○ 5 ○ 7 ○ 9 ○ *can't be determined*

9. **Dynamic programming. (6 points)**

You have n gold bars with weights $w_1, w_2, \ldots, w_n$, all positive integers. Determine whether the bars can be partitioned into two groups so that each group has total weight exactly $\frac{1}{2}W$, where $W = w_1 + w_2 + \cdots + w_n$. Assume W is even.

For example, if the weights are $\{3, 6, 4, 12, 23\}$, then the answer is *no*. If the weights are $\{3, 6, 4, 12, 17\}$, then the answer is *yes*, because $w_1 + w_2 + w_4 = w_3 + w_5 = \frac{1}{2}W = 21$.

You will formulate this as a *dynamic programming* problem. Define the following subproblems, one for each i and j with $0 \le i \le n$ and $0 \le j \le \frac{1}{2}W$:

$$OPT(i,j) = \begin{cases} \textit{true} & \text{if some subset, possibly empty, of } \{w_1, w_2, \ldots, w_i\} \text{ sums to exactly } j \\ \textit{false} & \text{otherwise} \end{cases}$$

Consider the following partial bottom-up implementation:

```
boolean[][] opt = new boolean[n+1][W/2 + 1];
opt[0][0] = true;
for (int j = 1; j <= W/2; j++)
    opt[0][j] =     1     ;
for (int i = 1; i <= n; i++)
    opt[i][0] =     2     ;

for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= W/2; j++) {
        opt[i][j] =     3     ;
        if (    4     &&     5     )
            opt[i][j] =     6     ;
    }
}
StdOut.println("answer = " + opt[n][W/2]);
```

- **A.** `false`
- **B.** `true`
- **C.** `null`
- **D.** `(j >= weights[i])`
- **E.** `(j <  weights[i])`
- **F.** `(j == weights[i])`
- **G.** `opt[i-1][j-1]`
- **H.** `opt[i-1][j]`
- **I.** `opt[i][j-1]`
- **J.** `opt[i][j]`
- **K.** `opt[i][j+1]`
- **L.** `opt[i-1][j - weights[i]]`
- **M.** `opt[i][j - weights[i]]`

For each numbered oval above, write the letter of the corresponding expression on the right in the space provided. You may use each letter once, more than once, or not at all.

1	2	3	4	5	6

10. **Graph algorithms. (5 points)**

Suppose we modify a graph algorithm to terminate early. Which of the following early-termination rules still guarantee that the algorithm's output is correct?

For each rule, fill in exactly one bubble: valid or invalid.

valid	*invalid*	
○	○	Stop *Kruskal's algorithm* as soon as $V-1$ edges have been added to T (instead of continuing until all E edges have been considered).
○	○	Stop *breadth-first search* as soon as every vertex has been marked and enqueued (instead of continuing until the queue is empty).
○	○	Stop *Dijkstra's algorithm* as soon as every vertex has been inserted into the priority queue (instead of continuing until the priority queue is empty).
○	○	When using *depth-first search* to determine whether t is reachable from s, stop as soon as t is marked (instead of continuing until all reachable vertices are marked).
○	○	Assuming all edge weights are nonnegative, stop *Bellman–Ford* as soon as a full pass (relaxing all E edges) completes without decreasing any `distTo[]` values (instead of always performing $V-1$ passes).

11. **Randomness. (4 points)**

Consider the following code fragment, which uses *rejection sampling.* At each step, it selects a site in an n-by-n grid uniformly at random and opens it if it is not already open, continuing until all n^2 sites are open:

```
boolean[][] isOpen = new boolean[n][n];
int numOpened = 0;
while (numOpened < n*n) {
    int col = StdRandom.uniformInt(n);   // two calls to
    int row = StdRandom.uniformInt(n);   // uniformInt()
    if (!isOpen[col][row]) {
        StdOut.printf("open site (%d, %d)\n", col, row);
        isOpen[col][row] = true;
        numOpened++;
    }
}
```

(a) What is the *expected number* of `uniformInt()` calls performed to open the *first* site?

○ 1 [✓ 2] ○ 4 ○ 8 ○ $\Theta(n)$

(b) Once exactly one site remains closed, what is the *expected number* of `uniformInt()` calls performed to open that *last* site?

○ 2 ○ n ○ $2n$ ○ n^2 ○ $2n^2$

(c) Once exactly two sites remain closed, what is the *expected number* of `uniformInt()` calls performed to open one of them (the *second-to-last* site)?

○ 2 ○ n ○ $2n$ ○ n^2 ○ $2n^2$

(d) What is the *expected number* of `uniformInt()` calls performed to open all n^2 sites?

○ $\Theta(n^2)$ ○ $\Theta(n^2 \log n)$ ○ $\Theta(n^3)$ ○ $\Theta(n^4)$ ○ $\Omega(2^n)$

(e) What is the *worst-case* number of `uniformInt()` calls performed to open all n^2 sites?

○ $\Theta(n^2)$ ○ $\Theta(n^2 \log n)$ ○ $\Theta(n^3)$ ○ $\Theta(n^4)$ ○ $\Omega(2^n)$

12. **Expert algorithms. (5 points)**

For each of the following statements, indicate whether it is (1) always true, (2) always false, or (3) sometimes true and sometimes false, depending on the particular sequence of expert predictions and outcomes. Each statement refers to a single run of an expert algorithm on some sequence.

For each statement, fill in exactly one bubble.

always true	*always false*	*depends*	
○	○	○	If a strict majority of the experts predicts 1 on day t, the *multiplicative-weights algorithm* also predicts 1 on day t.
○	○	○	In the *multiplicative-weights algorithm*, an expert who has made 6 mistakes has exactly half the weight of an expert who has made 7 mistakes.
○	○	○	If a strict majority of the experts is perfect, then the *multiplicative-weights algorithm* never makes a mistake.
○	○	○	In the *simplified AdaBoost algorithm*, the weight array on day 16 is [1/4, 1/8, 1/16, 1/8, 7/16].
○	○	○	Suppose we modify the *elimination algorithm* so that on day t we remove an expert only if both (1) the expert makes a mistake and (2) the algorithm makes a mistake. Assuming there is at least one perfect expert, this version of the elimination algorithm makes at most $\log_2 n$ mistakes, where n is the number of experts.

13. **Intractability (5 points).**

Suppose that Problem A is in **P**; Problem B is in **NP**; Problem C is *not* in **P**. Which of the following statements must be true?

Fill in all checkboxes that apply.

☐ Problem A is in **NP**.

☐ Problem C is in **NP**.

☐ Problem B can be solved in exponential time.

☐ Problem B *cannot* be solved in polynomial time.

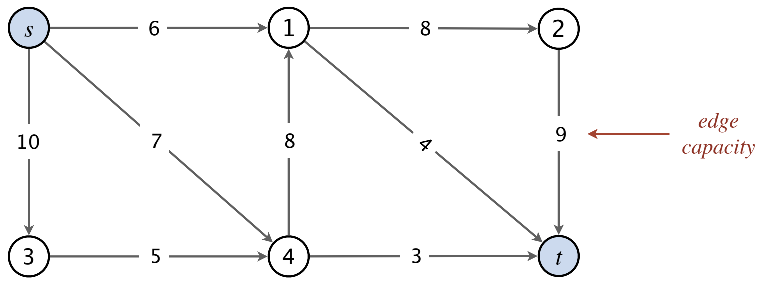
☐ If Problem C poly-time reduces to Problem B, then **P** ≠ **NP**

14. **Fattest path. (8 points)**

Given a digraph G with positive integer edge weights (capacities), the *bottleneck capacity* of a directed path is the *smallest* capacity of any edge on that path. A *fattest path* from s to t is an $s \rightsquigarrow t$ path whose bottleneck capacity is as *large* as possible.

For example, in the edge-weighted digraph G below:

- The bottleneck capacity of the path $s \to 3 \to 4 \to t$ is 3.
- The bottleneck capacity of the path $s \to 4 \to 1 \to 2 \to t$ is 7.
- The path $s \to 4 \to 1 \to 2 \to t$ is a fattest path from s to t.



(a) Design an algorithm that takes as input an edge-weighted digraph G, two vertices s and t, and an integer w, and either

- outputs an $s \rightsquigarrow t$ path whose bottleneck capacity is $\geq w$, or
- reports that no such path exists.

Your algorithm must run in $O(E + V)$ time in the worst case, where V and E are the number of vertices and edges in G.

In the box below, describe your algorithm. You may use any of the algorithms from the course as subroutines. Your answer will be graded on correctness, efficiency, and clarity.

(b) Design an algorithm that takes as input an edge-weighted digraph G and two vertices s and t, and either

- outputs a fattest path from s to t, or
- reports that no $s \rightsquigarrow t$ path exists.

Use an algorithm for the problem in part (a) as a subroutine. You may assume that this subroutine is correct, even if your solution to part (a) is not.

Performance requirement. For full credit, the running time of your algorithm must be $O((E+V)\log E)$ in the worst case, where V and E are the number of vertices and edges in G.

Partial credit. We will award 80% credit for a correct algorithm whose running time is $\Theta((E+V)\,\log U)$, where U is the maximum capacity of any edge, and 50% credit if the running time is $\Theta((E+V)\,U)$.

In the box below, describe your algorithm. Your answer will be graded on correctness, efficiency, and clarity.

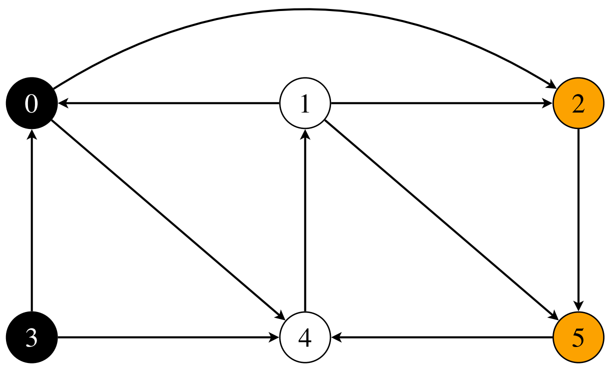
Are you attempting the full-credit or partial-credit solution? You may attempt only one.

○ *full credit* ○ *80% partial credit* ○ *50% partial credit*

15. **Tiger cut. (8 points)**

You are given a digraph G in which each vertex is colored orange, black, or white. Your goal is to recolor each white vertex either orange or black (without changing the colors of the non-white vertices) so as to minimize the number of edges that point from black vertices to orange vertices.

An example. Consider the following digraph G, where vertices 0 and 3 are black, vertices 2 and 5 are orange, and vertices 1 and 4 are white. The optimal solution colors vertex 4 black and vertex 1 orange, leaving only two edges that point from black to orange, $4 \to 1$ and $0 \to 2$. (Every other way of recoloring the white vertices yields more such edges.)



Goal. Design a *polynomial-time reduction* from the Tiger-Cut problem to the classic Min-st-Cut problem:

- Min-st-Cut: Given a *flow network* G' with positive edge capacities, a source vertex s, and a target vertex t, find an st-cut of minimum capacity.

Reduction requirements. For full credit, your reduction must construct a flow network G' with $V' = V + 2$ vertices and $E' \leq E + V$ edges, where V and E are the numbers of vertices and edges in G.

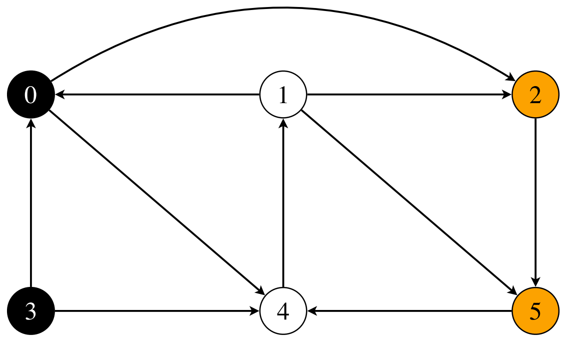
(a) Describe your reduction from TIGER-CUT to MIN-ST-CUT. Your reduction must work for any instance of TIGER-CUT, not just the one on page 16.

Describe how to transform a TIGER-CUT *instance* G *into a* MIN-ST-CUT *instance* (G', s, t). *Which vertices and edges are in the new flow network* G'*? What are their capacities? Which vertices are the source* s *and target* t*?*

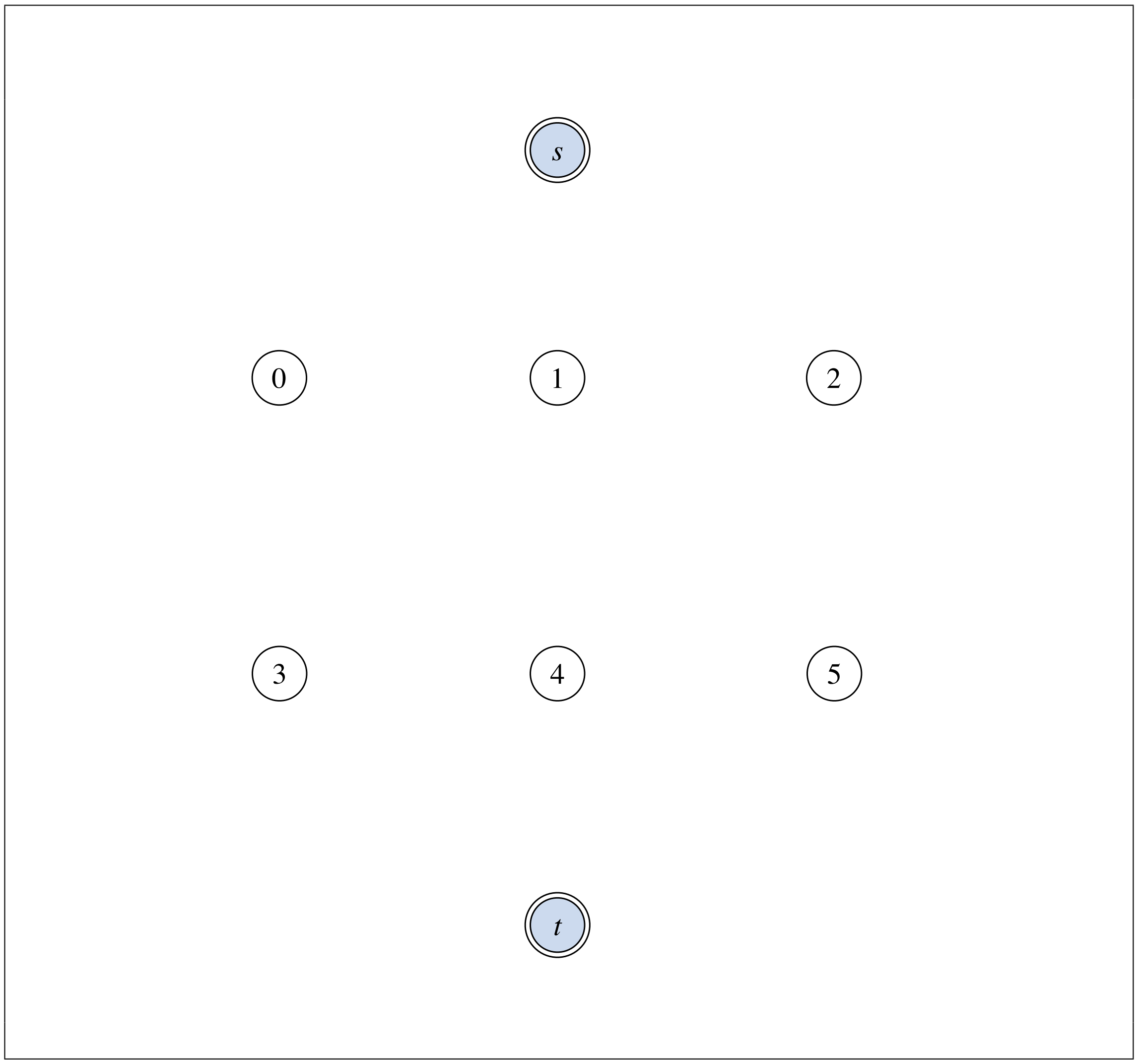
Now describe how to transform a solution to the MIN-ST-CUT *instance back into a solution to the original* TIGER-CUT *instance. How do you use a minimum st-cut* (S^*, T^*) *of* G' *to determine which color to assign to each white vertex?*

Your answers will be graded on correctness, efficiency, and clarity.

(b) *Draw* the MIN-ST-CUT instance (G', s, t) that would be constructed for the TIGER-CUT instance from page 16:



We have already provided the $V' = V + 2$ vertices in G' and labeled the source s and target t. Be sure to draw all edges, including their directions and capacities.



This page is intentionally blank. You may use this page for scratch work but do not remove it from the exam.

You may use this page as a reference during the exam.

Discrete Sums

Triangular:	$1 + 2 + 3 + 4 + \cdots + n \sim \frac{1}{2} n^2$
Logarithmic:	$\log_2 1 + \log_2 2 + \log_2 3 + \log_2 4 + \cdots + \log_2 n \sim n \log_2 n$
Harmonic:	$1 + \dfrac{1}{2} + \dfrac{1}{3} + \dfrac{1}{4} + \cdots + \dfrac{1}{n} \sim \ln n$
Geometric:	$1 + 2 + 4 + 8 + \cdots + n \sim 2n$ (for n a power of 2)

Logarithm Identities

Product rule:	$\log(xy) = \log x + \log y$
Quotient rule:	$\log\left(\dfrac{x}{y}\right) = \log x - \log y$
Power rule:	$\log(x^k) = k \log x$
Change of base:	$\log_a x = \dfrac{\log_b x}{\log_b a}$

Unless otherwise noted, log *denotes the logarithm function in some fixed base* $b > 1$.