COS 226 Algorithms and Data Structures Spring 2025

Final

This exam has 12 questions worth a total of 100 points. You have 180 minutes.

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 are allowed to 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: ◯ McCosh 50 ◯ McCosh 64 ◯ Other

Precept:

P01	P01A	P01B	P02	P02A	P02B	P03	P03A	P04
◯	◯	◯	◯	◯	◯	◯	◯	◯

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

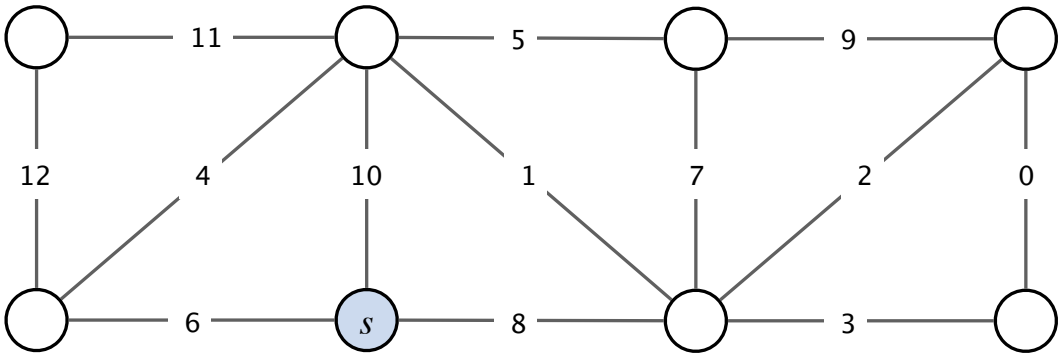
Signature

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

   In the spaces provided on the front of the exam, write your name and NetID; fill in the bubble for your exam room and the precept in which you are officially registered; write and sign the Honor Code pledge.

2. **Minimum spanning trees. (8 points)**

   Consider the following edge-weighted graph.

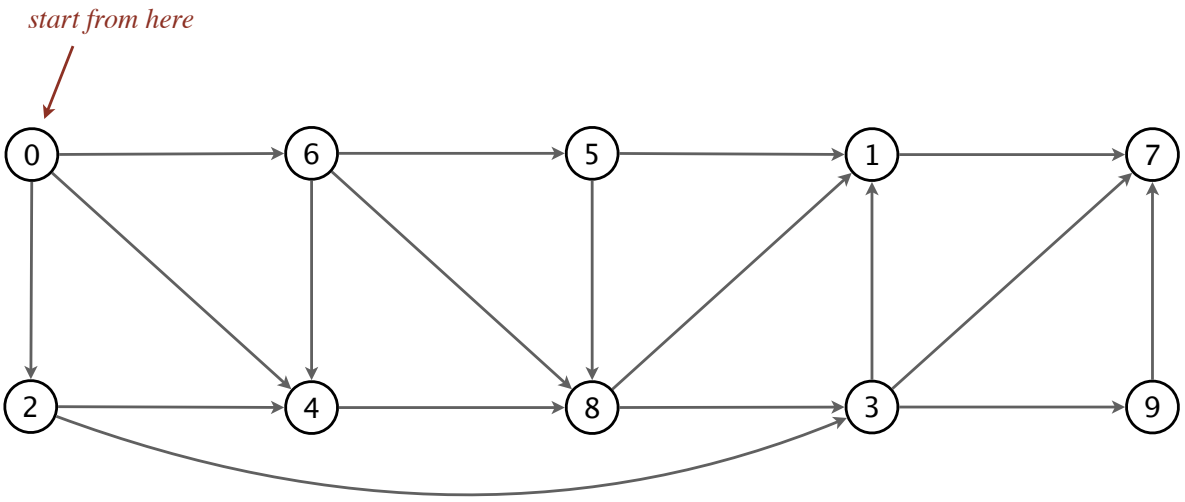


   (a) List the weights of the MST edges in the order that *Kruskal's algorithm* adds them to the MST.

   (b) List the weights of the MST edges in the order that *Prim's algorithm* adds them to the MST. Start Prim's algorithm from vertex s.

3. **Depth-first search. (10 points)**

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



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

0 ___ ___ ___ ___ ___ ___ ___ ___ ___ ___

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

___ ___ ___ ___ ___ ___ ___ ___ ___ 0

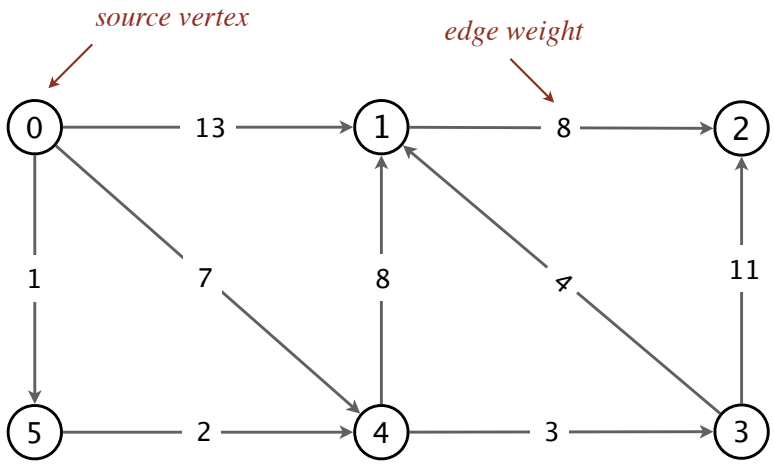
(c) Is the *reverse* of the DFS postorder in (b) a *topological order* for this digraph?

◯ *yes* ◯ *no*

4. **Shortest paths. (10 points)**

Run *Dijsktra's algorithm* in the following edge-weighted digraph, with source vertex $s = 0$.

Use the standard version from lecture/textbook with an `IndexMinPQ`. Assume the adjacency lists are in sorted order: for example, when iterating over the edges leaving vertex 0, consider the edge $0 \to 2$ before either $0 \to 4$ or $0 \to 6$.



(a) List the vertices in the order they are *added to* the priority queue.

0 ___ ___ ___ ___ ___ ___

(b) List the vertices in the order they are *removed from* the priority queue.

0 ___ ___ ___ ___ ___ ___

(c) For which vertices is the DECREASE-KEY operation called? *Fill in all checkboxes that apply.*

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

5. **Data structures. (12 points)**

(a) Suppose that the following 10 keys are inserted into an initially empty *linear-probing hash table*, in the order given:

`M O T H E R S D A Y`

Assume that all letters between `A` and `K` (`A`, `D`, `E`, and `H`) hash to index 2 and all letters between `L` and `Z` (`M`, `O`, `R`, `S`, `T`, and `Y`) hash to index 14.

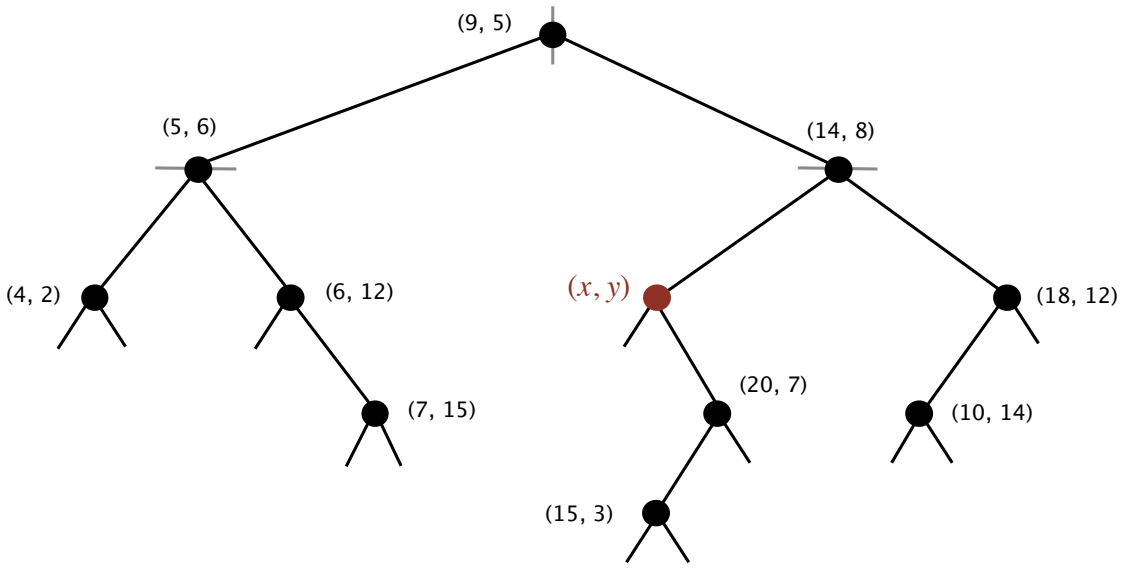
Fill in the contents of the underlying array. Assume that the length of the array is 16 and that it neither grows nor shrinks.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

(b) Is there an order in which the same 10 keys could be inserted into the linear-probing hash table that would lead to a *different set of empty array cells*?

◯ *yes* ◯ *no*

(c) Consider the following *2d-tree*, with the coordinates of one point (x, y) suppressed:



Which of the following points could be (x, y)?

Fill in all checkboxes that apply.

☐ (8, 8) ☐ (10, 1) ☐ (11, 7) ☐ (12, 12) ☐ (17, 4) ☐ (20, 6)

(d) Consider the following function, which returns the number of edges in a graph.

```
public static int numberOfEdges(Graph G) {
    int count = 0;
    for (int v = 0; v < G.V(); v++) {
        for (int w : G.adj(v)) {
            count++;
        }
    }
    return count / 2;
}
```

The `Graph` data type is implemented using the *adjacency-lists representation*.

What is the order-of-growth of the *worst-case* running time of `numberOfEdges()` as a function of the number of vertices V and the number of edges E?

◯ $\Theta(V)$ ◯ $\Theta(E)$ ◯ $\Theta(E+V)$ ◯ $\Theta(EV)$ ◯ $\Theta(V^2)$

(e) Assume that every vertex in G has degree 5. Which of the following expressions describes the running time of `numberOfEdges()` as a function of V?

Fill in all checkboxes that apply.

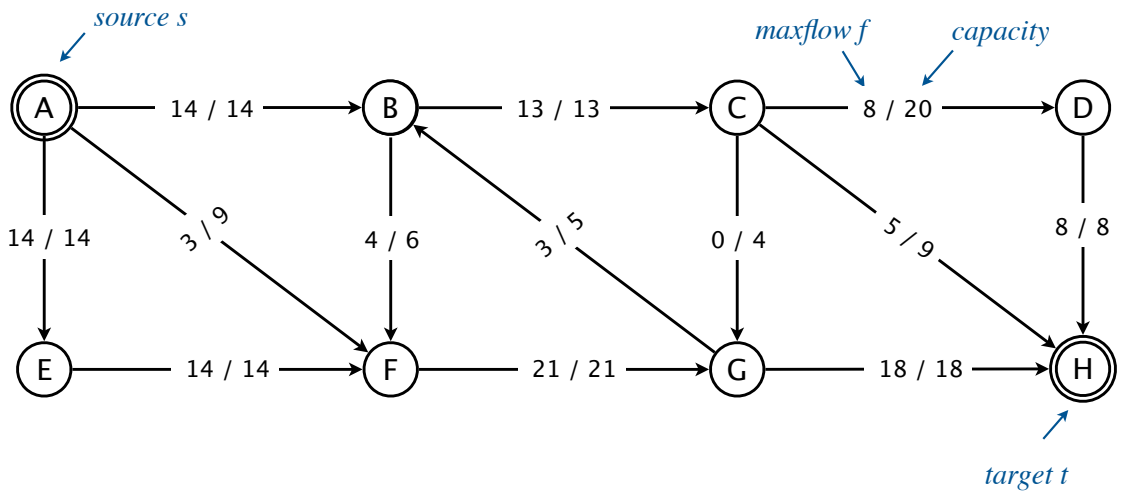
☐ $O(1)$ ☐ $O(V)$ ☐ $O(V^2)$

☐ $\Theta(1)$ ☐ $\Theta(V)$ ☐ $\Theta(V^2)$

☐ $\Omega(1)$ ☐ $\Omega(V)$ ☐ $\Omega(V^2)$

6. **Maxflows and mincuts. (10 points)**

Consider the following flow network G and a *maxflow* f.



(a) What is the *value* of the maxflow f?

○ 29 ○ 31 ○ 34 ○ 37 ○ 39

(b) Find a min s–t cut. *Fill in the checkboxes of all vertices on the s side of the cut.*

■ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H

(c) What is the *capacity* of the cut $\{A, B, E, F\}$?

○ 29 ○ 31 ○ 34 ○ 37 ○ 39

(d) What is the *net flow* across the cut $\{A, C, F\}$?

○ 29 ○ 31 ○ 34 ○ 37 ○ 39

(e) Suppose that an edge $B \rightarrow D$ of capacity 10 (and flow 0) is added to the flow network G. Find an *augmenting path* with respect to f in the modified flow network.

In the box below, write the sequence of vertices in the path.

$A \rightarrow$

(f) What is the *bottleneck capacity* of the augmenting path found in part (e)?

○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6

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

Recall the *egg drop problem* from lecture, where there is an n-story building and the goal is to drop eggs to identify the threshold floor T (for which an egg breaks on floor T or higher, and does not break on floor $T-1$ or lower).

In this version of the problem, you have exactly $m > 0$ eggs and want to determine the exact minimum number of tosses needed to determine T. For example, if $m = 1$, the answer is n because, depending on the value of T, the egg might break on floor 1 or survive on floor n.

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

$$OPT(i, j) = \textit{min number of drops with } i \textit{ eggs and } j \textit{ floors}$$

Consider the following partial bottom-up implementation:

```
int[][] opt = new int[m+1][n+1];
for (int j = 1; j <= n; j++)
   opt[1][j] = [1] ;
// for each number of eggs i
for (int i = 2; i <= m; i++) {
   // for each floor j
   for (int j = 1; j <= n; j++) {
      opt[i][j] = [2] ;
      // for each floor x on which to drop next egg
      for (int x = 1; x <= j; x++) {
         int temp = [3] + Math.max( [4] , [5] );
         opt[i][j] = Math.min(temp, [6] );
      }
   }
}
```

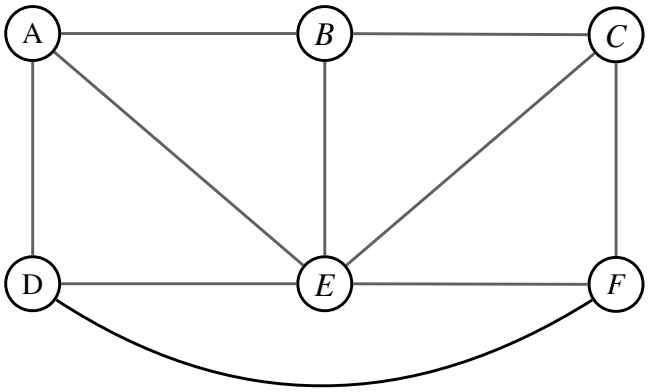
A. `0`
B. `1`
C. `Integer.MAX_VALUE`
D. `i`
E. `j`
F. `opt[i-1][j-x]`
G. `opt[i-1][j-1]`
H. `opt[i-1][j]`
I. `opt[i-1][x-1]`
J. `opt[i-1][x]`
K. `opt[i][j-x]`
L. `opt[i][j-1]`
M. `opt[i][j]`
N. `opt[i][x-1]`
O. `opt[i][x]`

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

8. **Karger's algorithm. (5 points)**

Run one execution of *Karger's algorithm* for finding a *global mincut* in the following graph. The table at right gives the uniformly random weights that this execution of Karger's algorithm assigns to the edges.



edge	*random weight*
A–B	0.8
A–D	0.1
A–E	0.3
B–C	0.2
B–E	0.7
C–E	0.5
C–F	0.6
D–E	0.0
D–F	0.9
E–F	0.4

(a) Which cut does this execution of Karger's algorithm find?

Fill in the checkboxes of all vertices that are on the same side of the cut as vertex A.

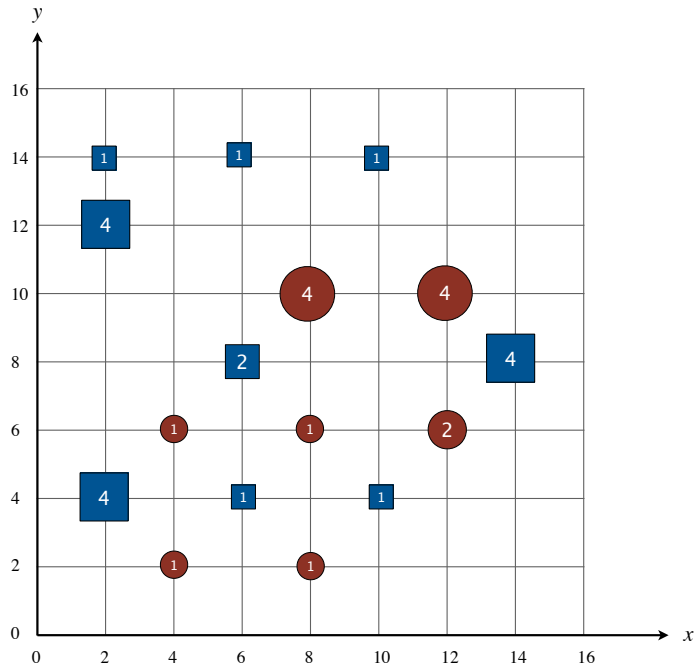
A ■ *B* □ *C* □ *D* □ *E* □ *F* □

(b) How many edges cross the cut found by this execution of Karger's algorithm?

○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6

9. **Multiplicative weights (8 points).**

Consider the following set of 16 points in the plane. Each point is labeled with a square (0) or circle (1) and has an associated weight. Compute a *decision stump* (d_p, v_p, s_p) that maximizes the sum of the weights of the correctly classified points.



(a) What is the *dimension* d_p?

◯ 0 ◯ 1

(b) What is the *threshold value* v_p?

◯ 1 ◯ 3 ◯ 5 ◯ 7 ◯ 9 ◯ 11 ◯ 13 ◯ 15

(c) What is the *sign* s_p?

◯ 0 ◯ 1

(d) What is the *total weight* of points correctly classified by the decision stump?

[ ]

10. **Intractability (10 points).**

Suppose that Problem A is in **P**; Problem B is in **NP**; Problem C is **NP**-complete; and Problem D is *not* in **P**. Which of the following can you infer?

Fill in all checkboxes that apply.

☐ Problem C is in **NP**.

☐ Problem D is **NP**-complete.

☐ Problem B can be solved in exponential time.

☐ If Problem B can be solved in poly-time, then **P** = **NP**.

☐ If Problem B *cannot* be solved in poly-time, then neither can Problem C.

☐ Problem A poly-time reduces to Problem B.

☐ Problem B poly-time reduces to Problem C.

☐ 3-SATISFIABILITY poly-time reduces to Problem C.

☐ If Problem C poly-time reduces to Problem B, then B is **NP**-complete.

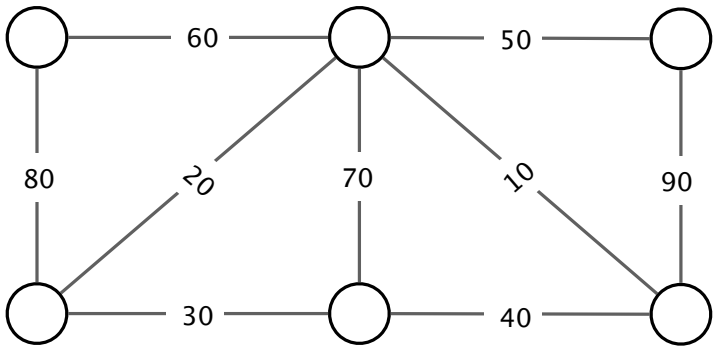
☐ If Problem C poly-time reduces to Problem A, then **P** = **NP**.

11. **Threshold connectivity. (10 points)**

Given a connected graph G with positive integer edge weights, determine the smallest possible integer w^* such that removing all edges with weight strictly greater than w^* leaves the graph connected.

Example. In the edge-weighted graph G below, the smallest possible integer $w^* = 60$.

- G remains connected after removing the three edges of weight > 60.
- G becomes disconnected after removing the four edges of weight ≥ 60.



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

Partial credit. We will award 90% credit if the running time is $\Theta(E \log U)$ in the worst case, where U is the maximum weight of an edge; and 50% credit if the running time is $\Theta(E\,U)$ in the worst case.

In the box below, describe your algorithm for solving the THRESHOLD-CONNECTIVITY problem. Your algorithm should work for any valid instance, not just the one on the facing page.

Your answer will be graded for correctness, efficiency, and clarity.

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

◯ *full credit* $O(E \log E)$

◯ *90% partial credit* $\Theta(E \log U)$

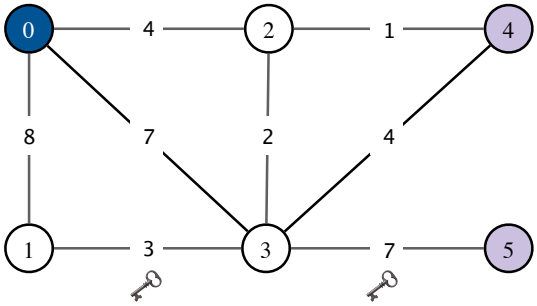
◯ *50% partial credit* $\Theta(E U)$

12. **Key-and-portal shortest path. (10 points)**

Let G be an undirected graph with non-negative edge weights and let s be a *source* vertex. Some of the edges contain *keys* and some of the vertices are *portals*. The goal is to find a shortest path from s to a portal vertex that includes one (or more) edges with keys. (The path is permitted to revisit a vertex or reuse an edge.)

Example. Consider the following edge-weighted graph G with $s = 0$, keys on the edges 1–3 and 3–5, and portals at vertices 4 and 5:

- The key-and-portal shortest path is 0–2–3–5. Its length is 13 (4 + 2 + 7).
- The path 0–1–3–5 is a valid path (of length 18), but not a shortest one.
- The path 0–2–4 is an invalid path because none of its edges contains a key.
- The path 0–2–4–3–1 is an invalid path because it does not end at a portal vertex.



Goal. Design a reduction from the KEY-AND-PORTAL-SHORTEST-PATH problem to the SOURCE-SINK-SHORTEST-PATH problem.

- KEY-AND-PORTAL-SHORTEST-PATH: Given an undirected graph G with non-negative edge weights, a source vertex s, some edges labeled with keys, and some vertices labeled as portals, find the length of a shortest path from s to a portal vertex that includes at least one edge labeled with a key.

- SOURCE-SINK-SHORTEST-PATH: Given a digraph G' with non-negative edge weights and two distinguished vertices s' and t', find the length of a shortest path from s' to t'.

That is, given any instance of KEY-AND-PORTAL-SHORTEST-PATH*, describe how to construct an instance* (G', s', t') *of* SOURCE-SINK-SHORTEST-PATH *whose solution has the same value.*

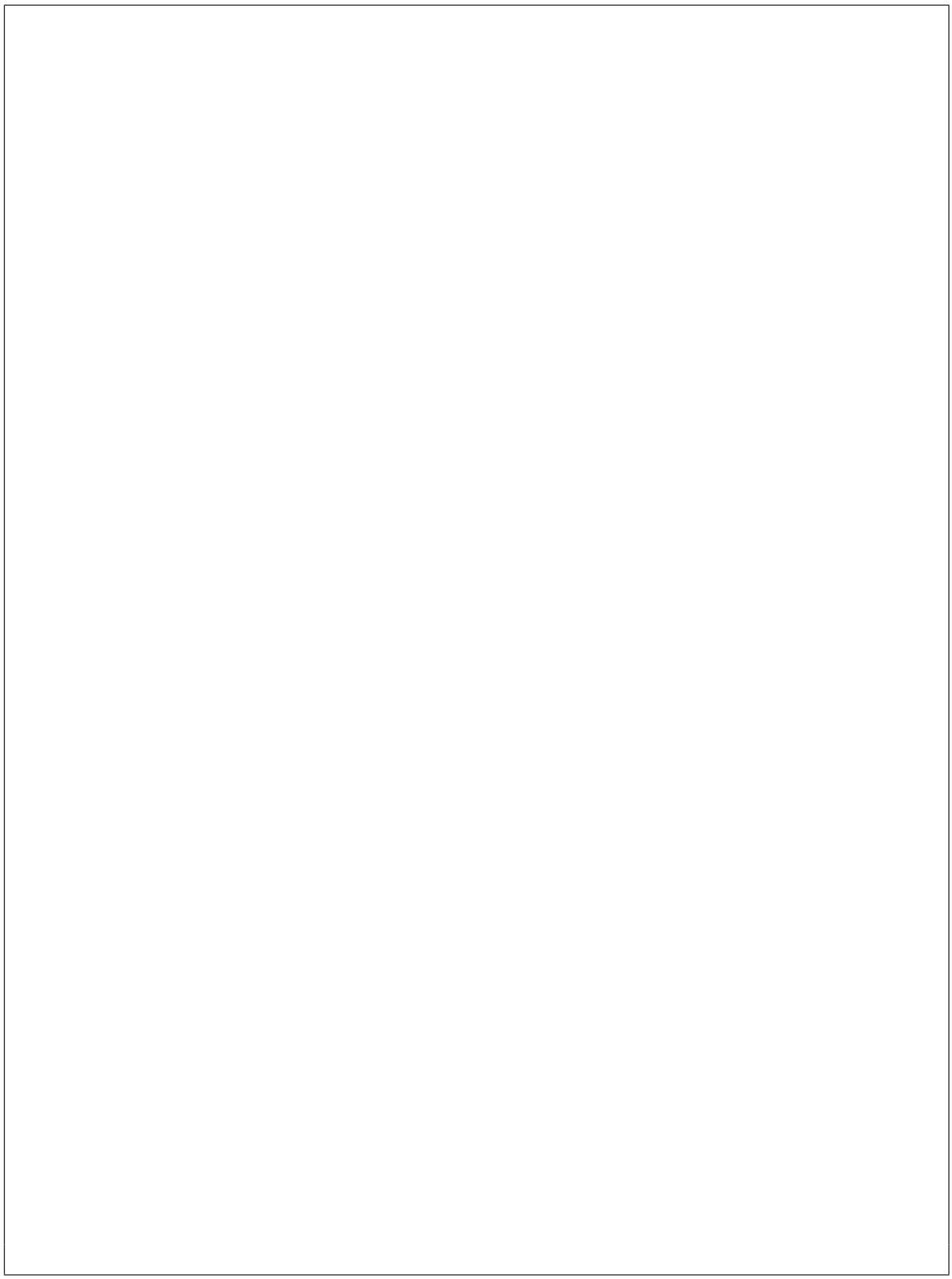
Reduction requirements. For full credit, the number of vertices V' and edges E' in G' must each be $O(E + V)$, where V and E are the number of vertices and edges in G.

(a) Describe your reduction for solving the KEY-AND-PORTAL-SHORTEST-PATH problem. Your description should work for any instance of KEY-AND-PORTAL-SHORTEST-PATH, not just the one on page 16.

To receive credit, your answer must be a reduction, not an algorithm that solves KEY-AND-PORTAL-SHORTEST-PATH *from scratch. Partial credit will be awarded for reductions that correctly model different aspects of the problem (e.g., the keys, the portals, and the undirected edges).*

Your answer will be graded for correctness, efficiency, and clarity.

(b) *Draw* the SOURCE-SINK-SHORTEST-PATH instance G' that would be constructed to solve the KEY-AND-PORTAL-SHORTEST-PATH instance G on page 16. Be sure to draw all vertices (labeling the source s' and destination t') and all edges (including arrows and edge weights).



This page is intentionally blank. You may use this page for scratch work.