COS 226	Algorithms and Data Structures	Spring 2023

Final

This exam has 14 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 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 46 ○ McCosh 50 ○ Other

Precept:

P01	P02	P02A	P03	P03A	P03B	P04	P04A	P05
○	○	○	○	○	○	○	○	○

"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. **Empirical running time. (6 points)**

Suppose that you observe the following running times (in seconds) for a program on graphs with V vertices and E edges.

		E = 100	200	400	800
V	100	0.25	0.5	1.0	2.0
	200	2.0	4.0	8.0	16.0
	400	16.0	32.0	64.0	128.0
	800	128.0	256.0	512.0	1024.0

(a) *Estimate* the running time of the program (in seconds) for a graph with $V = 1{,}600$ vertices and $E = 1{,}600$ edges.

○ 2,000 ○ 4,000 ○ 8,000 ○ 16,000 ○ 32,000

(b) What is the *order of growth* of the running time as a function of both V and E?

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

3. **Analysis of algorithms. (6 points)**

Determine the *order of growth* of the running time of each of the following code fragments as a function of V and E, where V and E are the number of vertices and edges in graph G, respectively. Assume the standard *adjacency-lists representation*.

(a)

```
int count = 0;
int V = G.V();
for (int v = 0; v < V; v++)
    for (int w = 0; w < v; w++)
        count++;
```

○ $\Theta(V)$ ○ $\Theta(E)$ ○ $\Theta(V \log V)$ ○ $\Theta(V^2)$ ○ $\Theta(V^2 \log V)$

(b)

```
int count = 0;
int V = G.V();
for (int v = 0; v < V; v++)
    for (int w : G.adj(v))
        count++;
```

○ $\Theta(V)$ ○ $\Theta(E)$ ○ $\Theta(E + V)$ ○ $\Theta(V^2)$ ○ $\Theta(VE)$

(c)

```
int count = 0;
int V = G.V();
for (int v = V; v >= 1; v = v / 2)
    for (int w = 1; w <= v; w++)
        count++;
```

○ $\Theta(V)$ ○ $\Theta(E)$ ○ $\Theta(V \log V)$ ○ $\Theta(V^2)$ ○ $\Theta(V^2 \log V)$

4. **String sorts. (5 points)**

The column on the left contains the original input of 24 strings to be sorted; the column on the right contains the strings in sorted order; the other 5 columns contain the contents at some intermediate step during one of the 3 radix-sorting algorithms listed below. Match each algorithm by writing its letter in the box under the corresponding column.

You may use each letter once, more than once, or not at all.

0	3543	1100	2346	1100	1100	1100	1100
1	2346	6501	1664	1491	1864	1491	1491
2	9397	3006	1100	1532	1491	6501	1532
3	8686	5609	1563	1563	1532	1532	1563
4	1100	5316	1719	1664	1719	7092	1664
5	3239	3117	1532	1719	1563	3543	1719
6	9458	3419	1864	1864	1664	1563	1864
7	7868	1719	1491	2346	2346	1864	2346
8	5609	5629	3239	3543	3543	7584	3006
9	5316	1532	3419	3239	3239	1664	3117
10	3006	3239	3006	3006	3006	2346	3239
11	1864	3543	3117	3419	3419	8686	3419
12	1491	2346	3543	3117	3117	5316	3543
13	3419	4149	4149	4149	4149	3006	4149
14	4149	9458	7584	5609	5609	9397	5316
15	7584	1563	5316	5316	5316	3117	5609
16	1532	1864	6501	5629	5629	9458	5629
17	6501	1664	5609	6501	6501	7868	6501
18	1719	7868	7092	7868	7868	3239	7092
19	7092	7584	7868	7584	7584	5609	7584
20	1563	8686	5629	7092	7092	3419	7868
21	5629	1491	9458	8686	8686	4149	8686
22	3117	7092	8686	9397	9397	1719	9397
23	1664	9397	9397	9458	9458	5629	9458
	A						E

A. Original input

B. LSD radix sort

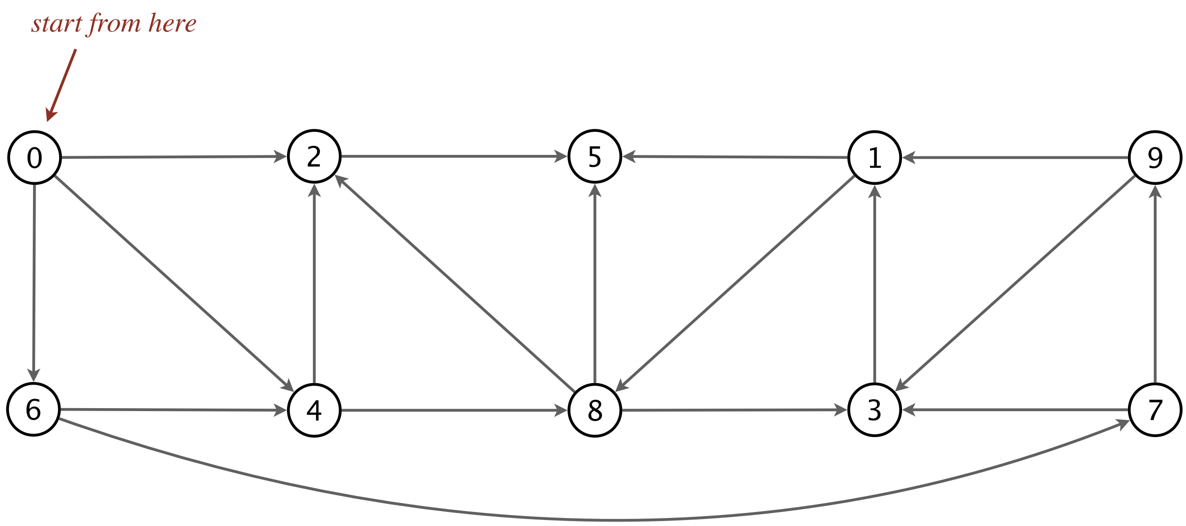
C. MSD radix sort

D. 3-way radix quicksort (*no shuffle*)

E. Sorted

5. **Depth-first search. (8 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*.

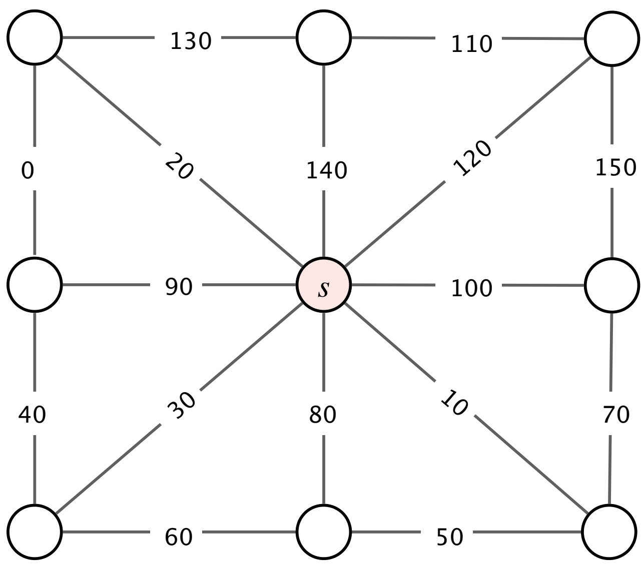
___ ___ ___ ___ ___ ___ ___ ___ ___ ___ 0

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

◯ *yes* ◯ *no*

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

   Consider the following edge-weighted graph.

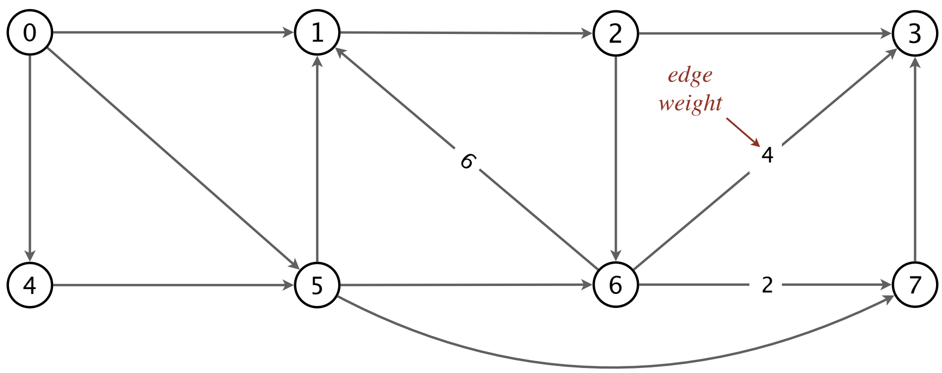


(a) List the weights of the MST edges in the order that *Kruskal's algorithm* adds them to the MST.

(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.

7. **Shortest paths. (8 points)**

Suppose that you are running Dijkstra's algorithm in the following edge-weighted digraph, with source vertex $s = 0$. Just prior to relaxing vertex 6, the `distTo[]` array is as follows:



v	distTo[]
0	0.0
1	50.0
2	
3	43.0
4	41.0
5	35.0
6	38.0
7	36.0

(a) Which vertices (including vertex 6) are currently in the priority queue?
Mark all that apply.

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ■ 6 ☐ 7

(b) Which vertex will Dijkstra's algorithm relax immediately after vertex 6?

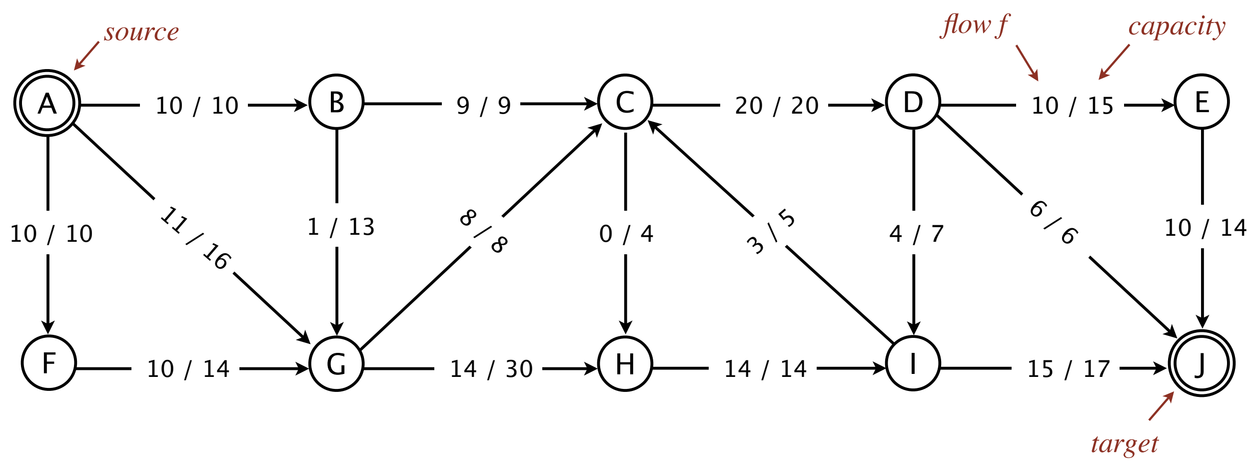
○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ *cannot be determined*

(c) Which is the weight of edge 7→3?

○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ *cannot be determined*

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

Consider the following flow network and *maximum flow* f.



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

○ 20 ○ 31 ○ 32 ○ 36 ○ 37

(b) What is the *capacity* of the cut $\{A, B, C\}$?

○ 31 ○ 42 ○ 50 ○ 63 ○ 76

(c) Which vertices are on the source side of a *minimum cut*? *Mark all that apply.*

A	B	C	D	E	F	G	H	I	J
■	□	□	□	□	□	□	□	□	□

(d) Suppose that the capacity of edge $B \to C$ is increased from 9 to 10. Which of the following paths would become *augmenting paths* with respect to flow f? *Mark all that apply.*

□ $A \to G \to B \to C \to I \to D \to E \to J$

□ $A \to B \to C \to D \to E \to J$

□ $A \to G \to B \to C \to I \to J$

□ $A \to G \to H \to C \to I \to J$

□ *none of the above*

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

(a) Suppose that the following keys are inserted into an initially empty *linear-probing hash table*, but not necessarily in the order given,

key	*hash*
A	3
B	4
C	4
D	0
E	1

Which of the following hash tables could arise? Assume that the initial size of the hash table is 5 and that it neither grows nor shrinks.

Fill in all checkboxes that apply.

☐

0	1	2	3	4
A	B	C	D	E

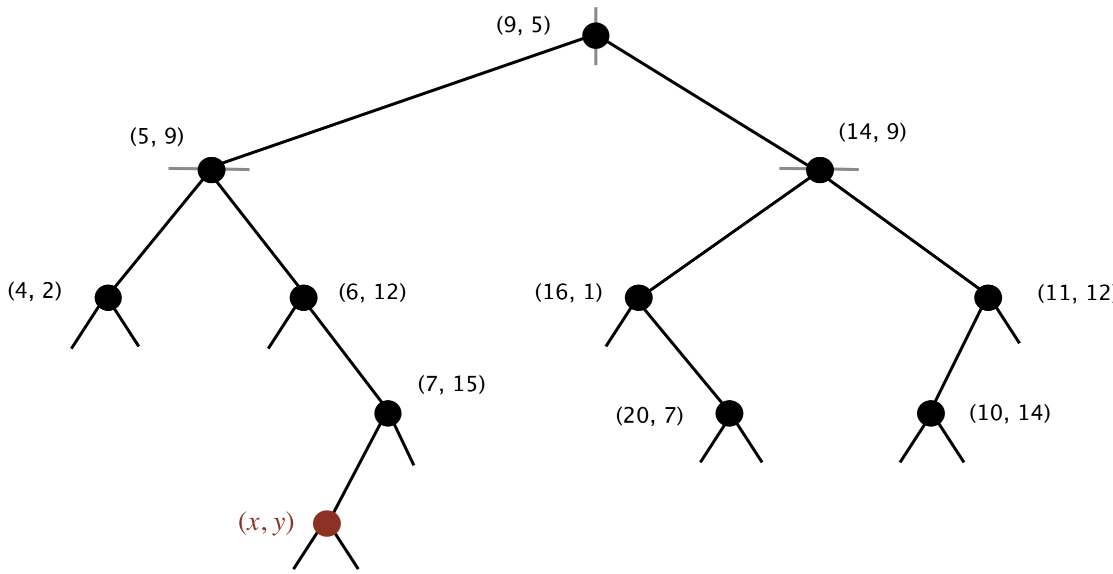
☐

0	1	2	3	4
D	B	C	E	A

☐

0	1	2	3	4
C	D	E	A	B

(b) Consider the following *2d-tree*:

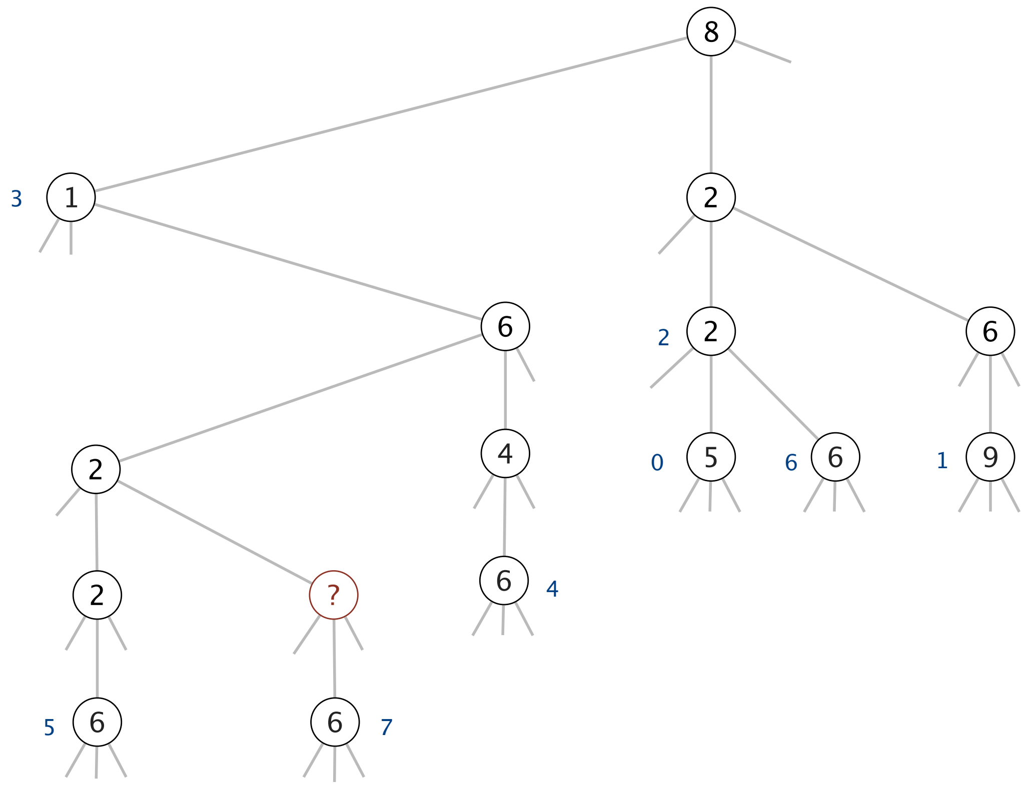


Which of the following points could correspond to (x, y)?

Fill in all checkboxes that apply.

☐ (5, 10) ☐ (7, 10) ☐ (7, 16) ☐ (8, 8) ☐ (8, 14) ☐ (10, 10)

(c) Consider the following *ternary search trie*, where the question mark represents an unknown digit:



Which of the following string keys are (or could possibly be) in the TST?

Fill-in all checkboxes that apply.

1	226	236	36	56	646	76
■	□	□	□	□	□	□

81	822	8225	826	8269	869	96
□	□	■	□	□	□	□

10. **Data compression. (8 points)**

For each of the following data compression algorithms, identify the *worst-case compression ratio*. Recall that the compression ratio is the number of bits in the encoded message divided by the number of bits in the original message.

For each algorithm on the left, write the letter of the best-matching term on the right. You may use each letter once, more than once, or not at all.

☐	*Run-length coding* with 8-bit counts.	**A.** ~ 1
		B. ~ 3/2
☐	*Huffman coding* over the extended ASCII alphabet ($R = 256$).	**C.** ~ 2
☐	*LZW compression* over the extended ASCII alphabet ($R = 256$), with 12-bit codeword.	**D.** ~ 4
		E. ~ 8
☐	*Burrows–Wheeler compression* over the extended ASCII alphabet ($R = 256$). This includes the Burrows–Wheeler transform, move-to-front encoding, and Huffman coding.	**F.** ~ 12
		G. ~ 16
		H. ~ 256

11. **Burrows–Wheeler transform. (5 points)**

(a) What is the Burrows–Wheeler transform of the following string?

`A N A B E L L A`

	⟵ *integer index*							
								⟵ *core*

Feel free to use this grid for scratch work.

(b) Consider all strings whose *core* Burrows–Wheeler transform (i.e., the Burrows–Wheeler transform excluding the integer index) is the same as the core Burrows–Wheeler transform of

`A N A B E L L A`

In the space below, write the lexicographically smallest such string (i.e., the first one that would appear alphabetically).

12. **DFS postorder. (5 points)**

Consider the following partial implementation for computing the *DFS postorder* in a digraph:

```
public PostorderDFS(Digraph G) {
    marked = new boolean[G.V()];
    postorder = new Queue<Integer>();
    for (int v = 0; v < G.V(); v++)
        if (!marked[v])
            dfs(G, v);
}

private void dfs(Digraph G, int v) {
      1
      2
      3
      4
      5
}
```

A. `dfs(G, v);`

B. `dfs(G, w);`

C. `marked[v] = true;`

D. `marked[w] = true;`

E. `postorder.enqueue(v);`

F. `postorder.enqueue(w);`

G. `if (!marked[v])`

H. `if (!marked[w])`

I. `for (int w : G.adj(v))`

J. `for (int v = 0; v < G.V(); v++)`

For each numbered oval above, write the letter of the corresponding code fragment on the right in the space provided. Use each letter at most once.

1	2	3	4	5

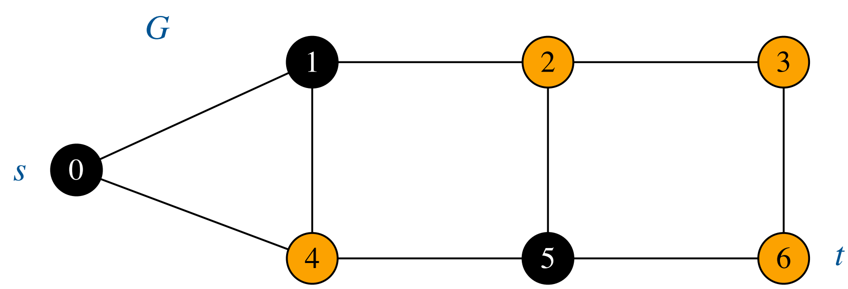
13. **Shortest tiger path. (10 points)**

Consider a graph in which each vertex is colored black or orange. A *tiger path* is a path that contains exactly one edge whose endpoints have opposite colors.

Shortest tiger path problem. Given an undirected graph G and two vertices s and t, find a tiger path between s and t that uses the fewest edges (or report that no such path exists).

An example. Consider the graph G below with $s = 0$ and $t = 6$.

- The shortest path between s and t is 0–4–5–6, but it is not a tiger path.
- The shortest *tiger path* between s and t is 0–1–2–3–6.



Goal. Formulate the shortest tiger path problem as a traditional (unweighted) shortest path problem in a *directed* graph. Specifically, define a digraph G', source s', and destination t' such that the length of the shortest path from s' to t' in G' is always equal to the length of the shortest tiger path between s and t in G. For simplicity, you may assume that s is black and t is orange.

Performance requirements. For full credit, the number of vertices in G' must be $\Theta(V)$ and the number of edges must be $\Theta(E)$, where V and E are the number of vertices and edges in G, respectively.

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

Briefly describe how to construct the *digraph* G', s', and t' from G, s, and t.
Your description should work for any graph G, not just the one on the facing page.

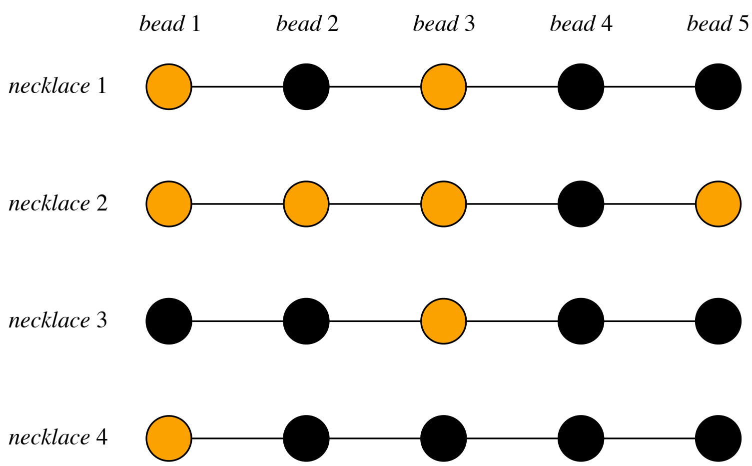
Draw the *digraph* G' corresponding to graph G on the facing page. Label s' and t'.

14. **Necklaces. (10 points)**

A *necklace* consists of a sequence n beads, each of which is either orange (0) or black (1). In this question, we have a set of m necklaces and are interested in identifying a common sequence of beads that appears at the end of many necklaces.

The problem. Given a set of m necklaces, each containing n beads, and a positive integer $k \leq n$, design an algorithm to find the most *popular* ending sequence of length k.

An example. Consider the following $m = 4$ necklaces, each containing $n = 5$ beads.



The most popular ending sequences for various values of k are as follows:

- $k = 1$: *black* (3 necklaces end with a black bead).
- $k = 2$: *black–black* (3 necklaces end with two black beads).
- $k = 3$: *orange–black–black* (2 necklaces end with this sequence).
- $k = 4$: *black–orange–black–black* (2 necklaces end with this sequence).
- $k = 5$: *orange–orange–orange–black–orange* (1 necklace ends with this sequence). There are three alternative answers.

Performance requirements. For full credit, the running time of your algorithm must be be $\Theta(mn)$ in the worst case.

Your answer will be graded for correctness, efficiency, and clarity (but not Java syntax). If your solution relies upon an algorithm or data structure from the course, do not reinvent it; simply describe how you are applying it.

(a) Describe your algorithm for identifying a most popular ending sequence of length k.

(b) *Draw* a diagram of the underlying data structures (such as arrays, linked lists, or binary trees) that your algorithm uses for the example input on the facing page. Show all relevant information, including any links and auxiliary data.

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