COS 226	Algorithms and Data Structures	Fall 2019

Final

This exam has 16 questions (including question 0) worth a total of 100 points. You have 180 minutes. This exam is preprocessed by a computer when grading, so please **write darkly** and **write your answers inside the designated spaces.**

Policies. The exam is closed book, except that you are allowed to use a one-page cheatsheet (8.5-by-11 paper, two sides, in your own handwriting). Electronic devices are prohibited.

Discussing this exam. Discussing the contents of this exam before solutions have been posted is a violation of the Honor Code.

This exam. Do not remove this exam from this room. In the space provided, write your name and NetID. Also, mark your exam room and the precept in which you are officially registered. Finally, write and sign the Honor Code pledge. You may fill in this information now.

Name:

NetID:

Course: COS 126 ○ COS 226 ●

Exam room: McCosh 50 ○ Other ○

Precept: P01 ○ P02 ○ P04 ○ P05 ○ P07 ○ P08 ○ P09 ○ P10 ○

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

Signature

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

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

1. **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					
		10,000	20,000	40,000	80,000	160,000	320,000
	10,000	6.25	8.84	12.50	17.68	25.00	35.36
	20,000	12.50	17.68	25.00	35.36	50.00	70.71
V	40,000	25.00	35.36	50.00	70.71	100.00	141.42
	80,000	50.00	70.71	100.00	141.42	200.00	282.84
	160,000	100.00	141.42	200.00	282.84	400.00	565.69
	320,000	200.00	282.84	400.00	565.69	800.00	1131.37

running time of the program (in seconds)

   (a) Estimate the running time of the program (in seconds) for a graph with $V = 640{,}000$ vertices and $E = 640{,}000$ edges.

   seconds

   (b) Estimate the *order of growth* of the running time of the program as a function of both V and E.

2. **Memory. (4 points)**

Suppose that you implement a symbol table (containing string keys and integer values) using an r-way trie with following data type:

```
public class RwayTrie {
    private final int r;
    private Node root;

    public RwayTrie(int r) {
        this.r = r;
        root = null;
    }

    private class Node {
        private final int value;
        private Node[] next;

        private Node(int value) {
            this.value = value;
            this.next = new Node[r];
        }
    }

    ...
}
```

Using the 64-bit memory cost model from lecture and the textbook, how much memory does each `Node` object use? Count all memory allocated when a `Node` object is constructed. Write your answer as a function of r.

	bytes

3. **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	4170	1233	3963	4170	9601	1018	1018
1	9601	1866	2145	9601	5601	1233	1233
2	8287	1018	1018	5601	4514	1866	1866
3	6853	2119	2119	5052	1018	2119	2119
4	5185	2145	3923	9152	2119	2145	2145
5	5052	3923	1866	7722	7722	3923	3923
6	9152	3963	1233	6853	3923	3963	3963
7	3923	4170	4514	3923	4528	4170	4170
8	9388	4528	4170	1233	6728	4528	4435
9	1233	4514	4435	7453	1233	4514	4514
10	4528	4435	4528	3963	4435	4435	4528
11	8587	5185	7453	4514	2145	5185	5052
12	7453	5052	6728	5185	5052	5052	5185
13	6728	5601	8587	4435	9152	5601	5601
14	1866	6853	9388	2145	6853	6853	6728
15	2119	6728	9152	1866	7453	6728	6853
16	1018	7453	5052	7056	7056	7453	7056
17	4514	7056	5185	8287	3963	7056	7453
18	4435	7722	6853	8587	1866	7722	7722
19	2145	8287	8287	9388	4170	8287	8287
20	3963	8587	7056	4528	5185	8587	8587
21	7056	9601	5601	6728	8287	9601	9152
22	5601	9152	7722	1018	8587	9152	9388
23	7722	9388	9601	2119	9388	9388	9601
	A						E

A. Original input

B. LSD radix sort

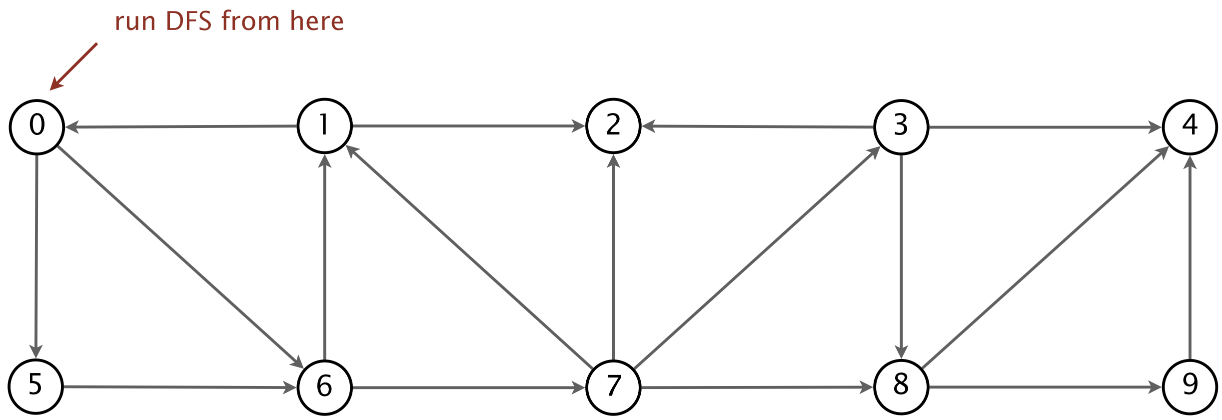
C. MSD radix sort

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

E. Sorted

4. **Depth-first search. (6 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 3, consider the edge $3 \to 2$ before either $3 \to 4$ or $3 \to 8$.



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

   0 ___ ___ ___ ___ ___ ___ ___ ___ ___

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

   ___ ___ ___ ___ ___ ___ ___ ___ ___ 0

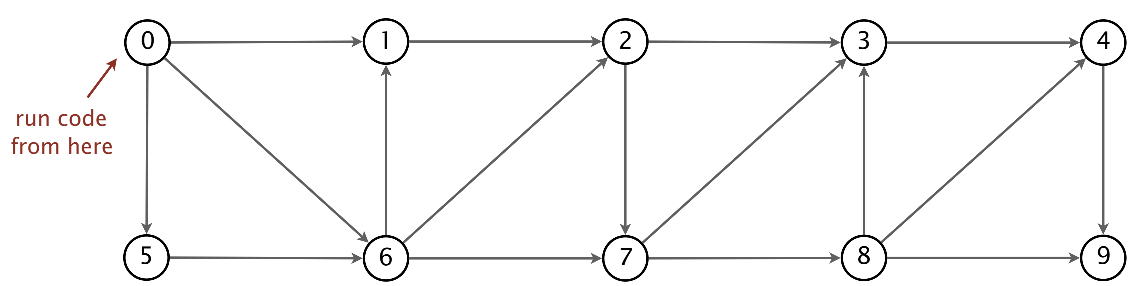
   (c) The above digraph does not have a *topological order*. If, however, you *delete* one edge, it will have a topological order. Which edge?

5. **Breadth-first search. (7 points)**

Consider the following *buggy* implementation of breadth-first search in a digraph.

```
private void bfs(Digraph G, int s) {
    marked = new boolean[G.V()];
    distTo = new int[G.V()];
    Queue<Integer> queue = new Queue<Integer>();

    queue.enqueue(s);
    while (!queue.isEmpty()) {
        int v = queue.dequeue();
        for (int w : G.adj(v)) {
            if (!marked[w]) {
                distTo[w] = distTo[v] + 1;
                queue.enqueue(w);
            }
        }
    }
}
```

(a) Suppose that you run the code fragment on the following DAG, starting from $s = 0$. Mark all statements below that are true.



■ It terminates.

□ Some vertices are added to the queue more than once.

□ At some point, the queue contains multiple copies of the same vertex.

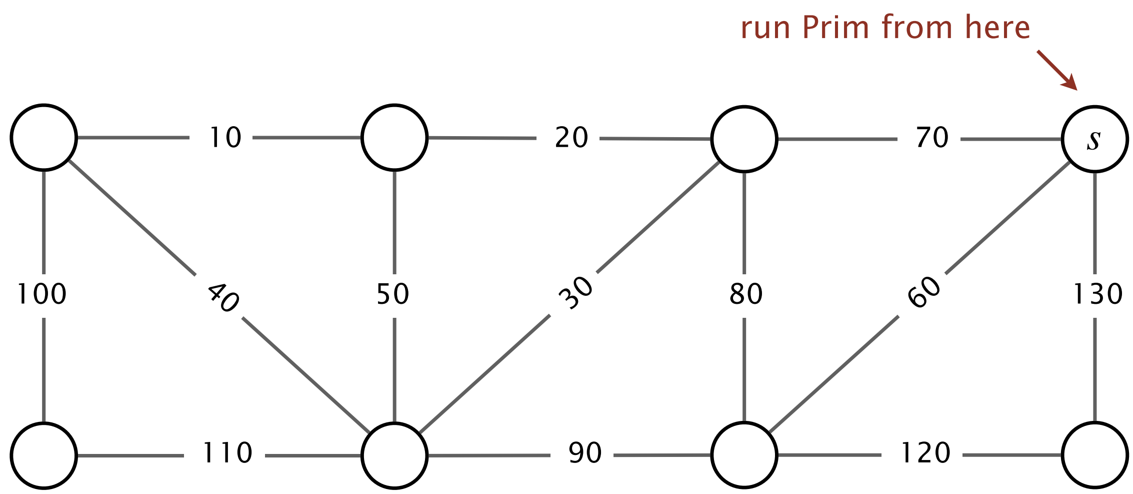
□ Upon termination, `distTo[1]` is `1` (the length of the *shortest* path from 0 to 1).

□ Upon termination, `distTo[9]` is `9` (the length of the *longest* path from 0 to 9).

(b) *Annotate* the code above to correct it.

6. **Minimum spanning tree. (6 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. **Knuth–Morris–Pratt substring search. (6 points)**

Consider the Knuth–Morris–Pratt DFA for the following string of length 8 over the alphabet { `A`, `B`, `C` }:

```
B B C B B B C A
```

(a) Complete the last three columns of this partially-completed DFA table.

	0	1	2	3	4	5	6	7
A	0	0	0	0	0			
B	1	2	2	4	5			
C	0	0	3	0	0			

(b) In which state is the DFA after consuming the following sequence of characters? Mark the correct answer.

```
C B B B B C B B C C B B B C B B B C B B B C B B C B B C B B C B
```

0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○

8. **Java String library performance. (8 points)**

For each of the `String` expressions at left, write the letter of the best-matching *worst-case* running time (as a function of m and n) at right, where

- `s` is a string of length n
- `t` and `regexp` are strings of length m
- $m \leq n$

Assume the standard (Oracle or OpenJDK) Java 8 representation and implementation for the `String` *data type. You may use each letter once, more than once, or not at all.*

☐	`s.length() + t.length()`	**A.** 1
☐	`s.charAt(n/2)`	**B.** $\log m$
☐	`s.substring(n/2, n)`	**C.** $\log n$
☐	`s.equals(t)`	**D.** m
☐	`s.indexOf(t)`	**E.** n
☐	`s.matches(regexp)`	**F.** m^2
☐	`s += t`	**G.** mn
☐	`for (int i = 0; i < s.length(); i++) t += s.charAt(i);`	**H.** n^2
		I. 2^n

```
for (int i = 0; i < s.length(); i++)
     t += s.charAt(i);
```

9. **Burrows–Wheeler data compression. (5 points)**

Consider compressing strings of length $6n$ that contains n copies of `X X Y Y Z Z` concatenated together. For example, here is the string corresponding to $n = 5$.

`X X Y Y Z Z X X Y Y Z Z X X Y Y Z Z X X Y Y Z Z X X Y Y Z Z`

For each transformation at left, determine the compression ratio (as a function of n) and write the letter of the best-matching term at right. As usual, assume the alphabet size $R = 256$.

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

☐	Move-to-front encoding.	**A.** ~ 1
		B. $\sim 7/8$
☐	Burrows–Wheeler transform.	**C.** $\sim 1/2$
		D. $\sim 3/8$
☐	Huffman compression.	**E.** $\sim 5/24$
		F. $\sim 1/4$
☐	Move-to-front encoding, followed by Huffman compression.	**G.** $\sim 3/16$
		H. $\sim 1/8$
		I. $\sim 1/16$
☐	Burrows–Wheeler transform, followed by move-to-front encoding, followed by Huffman compression.	**J.** $\sim 5/768$
		K. $\sim 1/256$

10. **Why did we do that? (8 points)**

    For each pair of algorithms or data structures, identify a critical reason why we prefer the first to the second. Write the letter of the best-matching answer.

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

☐ Use *adjacency lists* instead of an *adjacency-matrix* to represent a sparse undirected graph.	**A.** Guarantees correctness.
☐ Use *union–find* instead of *depth-first search* for cycle detection in Kruskal's algorithm.	**B.** Improves performance in practice.
☐ Relax the vertices in *increasing order of distance from the source* in Dijkstra's algorithm instead of in *reverse DFS postorder* to compute shortest paths in digraphs with positive edge weights.	**C.** None of the above.
☐ Use *3-way radix quicksort* instead of *mergesort* to sort an array of strings.	
☐ Use *Boyer–Moore* instead of *Knuth–Morris–Pratt* for substring search.	
☐ Use *depth-first search* instead of *breadth-first search* to compute a topological order in a directed acyclic graph.	
☐ Use *depth-first search* instead of *breadth-first search* to determine all vertices reachable from a set of vertices in NFA simulation.	
☐ Use *breadth-first search* instead of *depth-first search* to find a shortest ancestral path in the *WordNet* assignment.	

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

Given a digraph G with positive edge weights, complete the constructor below to compute the length of the shortest path from s to each vertex. To do so, write the letter of one of the following code fragments in each provided space.

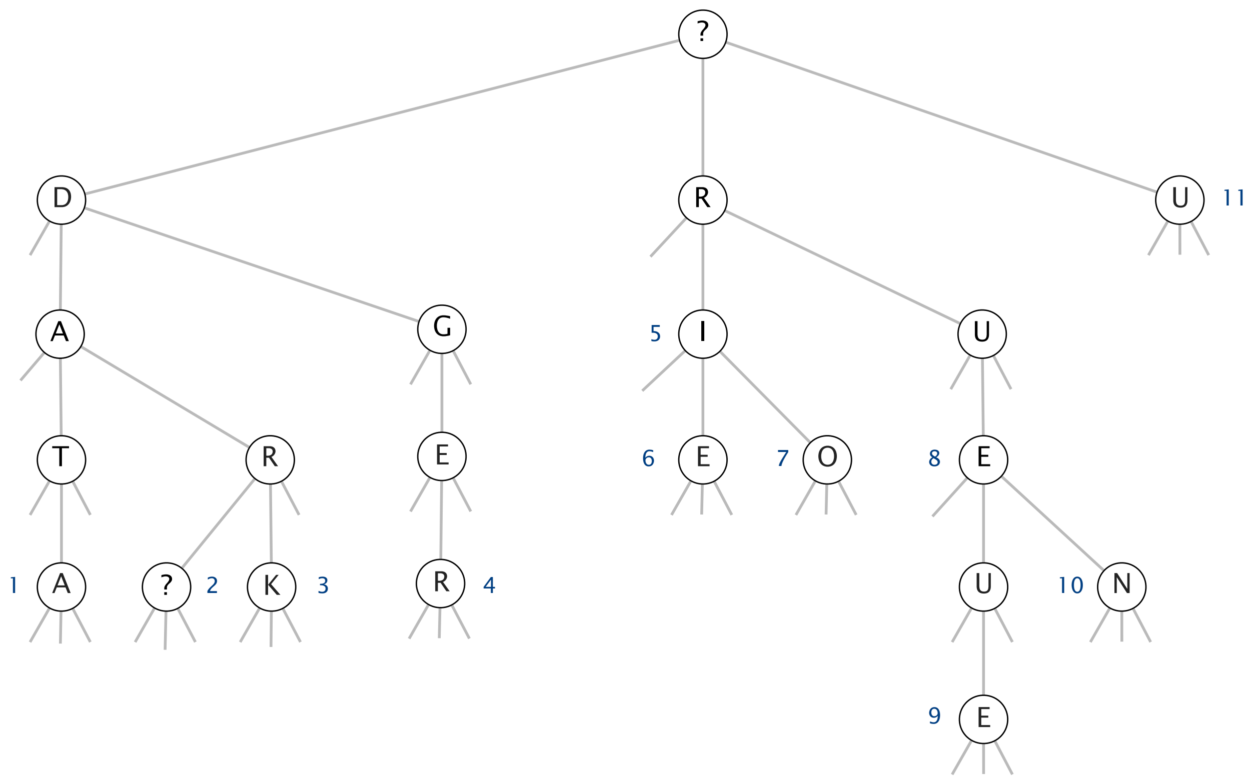
A.	`(int i = 1; i < G.V(); i++)`	**F.**	`Double.NEGATIVE_INFINITY`	**J.**	`0`
B.	`(int i = 1; i < G.E(); i++)`	**G.**	`Double.POSITIVE_INFINITY`	**K.**	`distTo[v]`
C.	`(int v = 0; v < G.V(); v++)`	**H.**	`distTo[v] + e.weight()`	**L.**	`distTo[w]`
D.	`(int v = 0; v < G.E(); v++)`	**I.**	`distTo[w] + e.weight()`		
E.	`(DirectedEdge e :  G.adj(v))`				

You may use each letter once, more than once, or not at all. No other code is allowed.

```
public BellmanFordSP(EdgeWeightedDigraph G, int s) {

     distTo = new double[G.V()];

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

          distTo[v] = _________ ;

     distTo[s] = _________ ;


     for  _________  {

          for  _________  {

               for  _________  {

                    int w = e.to();

                    if  ( _________   >   _________ )

                          _________   =   _________ ;
               }
          }
     }
}
```

12. **Ternary search tries. (5 points)**

Consider the following TST, where the integer values are shown next to the nodes of the corresponding string keys. Each node labeled with a `?` contains some uppercase letter (possibly different for each node).



Which of the following string keys are (or could possibly be) in the TST? Mark all that apply.

DATA	BRIE	DAD	DARK	DO	FUN
■	☐	☐	☐	☐	☐

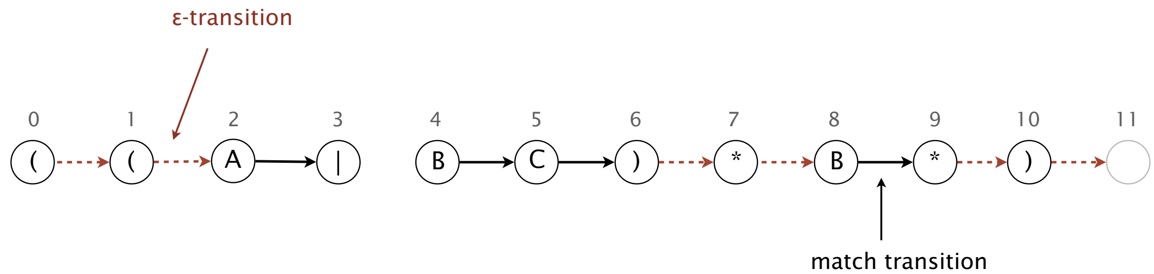
HUE	PRO	QUEUE	TRIE	TRUE
☐	☐	☐	☐	☐

13. **Regular expressions. (6 points)**

Consider the NFA that results from applying the RE-to-NFA construction algorithm from lecture and the textbook to the regular expression

```
( ( A | B C ) * B * )
```

The states and match transitions (solid lines) are shown below, but some of the ϵ-transitions (dotted lines) are suppressed.



(a) Mark all edges in the ϵ-transition digraph.

0→1	1→2	1→3	1→4	1→7	3→4	3→5	3→6	6→7
■	■	□	□	□	□	□	□	■

7→1	7→3	7→6	7→8	8→9	9→1	9→8	9→10	10→11
□	□	□	■	□	□	□	■	■

(b) Suppose that you want to construct an NFA for the regular expression

```
( ( A | B C ) ? B * )
```

where the operator `?` means *zero or one copy* of the expression that precedes it. What minimal change(s) would you make (e.g., adding or removing ϵ-transitions) to the NFA you defined in part (a)?

14. **Prefix-free codes. (10 points)**

(a) For a final exam question, an absentminded professor created a Huffman code for a set of 7 symbols. Unfortunately, she forgot to write down the codeword for one of the symbols.

symbol	*codeword*
A	00
B	01100
E	10
P	0111
R	01101
S	?
T	11

Deduce the codeword associated with the symbol `S`.

(b) Given a Huffman code (or optimal prefix-free code) for a set of $n \geq 3$ symbols, with one codeword missing, design an algorithm to deduce the missing codeword. The input to the problem is an array of the $n - 1$ known codewords.

Write your answer in the spaces provided on the next page.

Your answers to (b) and (c) will be graded for correctness, efficiency, and clarity. For full credit, your algorithm must take time linear in the input size (the total number of bits to represent the codewords) in the worst case.

Briefly describe your algorithm in the space below.

Draw a diagram of your data structure(s) for deducing the missing codeword when the known codewords are 00, 01100, 10, 0111, 01101, and 11, as in part (a).

(c) Now, suppose that there are *two* codewords missing. Design an algorithm to deduce the *two* missing codewords. Do *not* repeat details from part (b) if they are identical.

15. **Writing seminar assignment problem. (10 points)**

A prominent northeastern university assigns n students to m writing seminars. Each student ranks the writing seminars in order of preference (from favorite to least favorite). Each writing seminar has space for as many as p students. Design an algorithm to determine whether it is possible to assign the students to the writing seminars so that each student gets one of their *top two choices*. To do so, model the problem as a *maximum flow problem.*

An example. Here is an example input with $n = 6$ students (Abigail, Bjarne, Čazir, DeAndre, Eun-jung, and Flor) and $m = 3$ writing seminars (*X-Ray Crystallography*, *Your Life in Numbers*, and *Zoom!*), where each seminar has space for $p = 2$ students.

	1^{st}	2^{nd}	3^{rd}
A	X	Y	Z
B	Z	X	Y
C	X	Y	Z
D	Y	Z	X
E	X	Y	Z
F	Y	X	Z

In this example, there is no assignment in which each student gets their first choice (because three students rank X as their first choice). However, there is an assignment in which each student gets one of their top two choices:

assignment
A—X
B—Z
C—X
D—Z
E—Y
F—Y

(a) Draw the *flow network* that you would construct in order to solve the writing seminar assignment problem on the facing page (with 6 students and 3 writing seminars). Be sure to label the source and destination vertices and specify the edge capacities.

(b) After solving such a maximum flow problem, how would you determine whether there exists an assignment in which each students gets one of their *top two choices*?

(c) In the worst case, how many augmenting paths will the Ford–Fulkerson algorithm find (as a function of m and n)? Assume $n \geq m$. Mark the best answer.

m	n	$m+n$	$n \log n$	$m \log m$	mn	mn^2	m^2n	m^2n^2
○	○	○	○	○	○	○	○	○

(d) Let k be an integer between 1 and m. Suppose that you want to know whether it is possible to assign the students to writing seminars so that each students gets one of their *top k choices* (instead of top 2 choices). Briefly describe how you would modify your solution to (a).

(e) Design an efficient algorithm to find the *smallest* integer k for which it is possible to assign the students to writing seminars so that each student gets one of their *top k choices*. Your algorithm should be substantially faster in the worst case than repeatedly applying (d) to solve m maximum flow problems (one for each possible value of k).