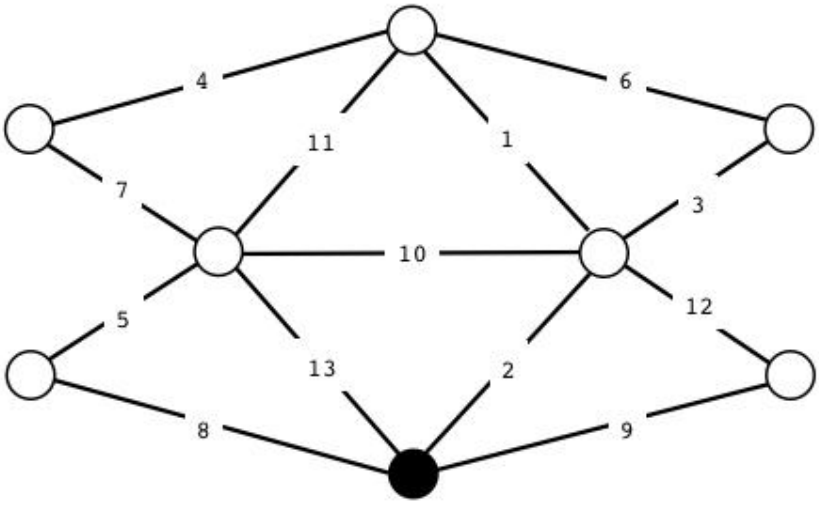
NAME:
login ID: **Precept (circle one):** `P01 P01A P01B P02 P02A P03`

COS 226 Final Exam, Spring 2011

This test is 15 questions, weighted as indicated. The exam is closed book, except that you are allowed to use a one page cheatsheet. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. *Put your name, login ID, and precept number on this page* (*now*), and write out and sign the Honor Code pledge before turning in the test. You have *three hours* to complete the test.
"I pledge my honor that I have not violated the Honor Code during this examination."

1.	MST	/10		
2.	KMP	/10		
3.	Algs match	/10		
4.	DFS trace	/15		
5.	LZW	/15		
			Subtotal	/60
6.	TST	/15		
7.	String sorts	/15		
8.	RE	/15		
9.	Bellman-Ford	/15		
10.	Codes	/15		
			Subtotal	/75
11.	3-way	/ 9		
12.	LP	/15		
13.	Subsequences	/20		
14.	Maxflow	/16		
15.	Intractability	/15		
			Subtotal	/75
			TOTAL	/210

1. **MST** (10 points). Consider the following graph (numbers are edge weights):



A. Give the list of edge weights in the MST in the order that *Kruskal's algorithm* inserts them.

B. Give the list of edge weights in the MST in the order that *Prim's algorithm* inserts them, assuming that it starts at the black vertex.

2. **KMP** (10 points). The following is a Knuth-Morris-Pratt state-transition table for a 9-character string.
A. Write the characters in the string in the blanks in the first line of the table.
B. Fill in the blanks in the table.

	0	1	2	3	4	5	6	7	8
	___	___	___	___	___	___	___	___	___
X	___	2	3	4	___	6	___	8	9
Y	___	___	___	___	___	___	___	___	___
Z	1	___	___	___	5	___	7	___	___

3. **Match the algorithms** (10 points). Consider the following algorithms and data structures.

   A. Bellman-Ford

   B. BST

   C. Dijkstra

   D. Ford-Fulkerson

   E. Hashing

   F. Insertion sort

   G. LZW

   H. Quicksort

   I. Simplex

   J. TST

   In the blank to the left of each problem below, fill in the letter of the most appropriate algorithm or data structure from the list above. "Most appropriate" means the algorithm or data structure is likely to be the basis of the most efficient correct solution to the problem among those on this list.

   _____ Sort a list of keys which are already nearly in sorted order.
   _____ Sort a large list of keys in roughly random order.
   _____ Find a shortest path in an edge-weighted digraph with some negative edge weights but no negative cycles.
   _____ Find a shortest path in an edge-weighted digraph with no negative edge weights.
   _____ Find the maximum of a linear objective function subject to linear equality and inequality constraints.
   _____ Find a maximum flow in a flow network.
   _____ Compress an input stream, using a symbol table.
   _____ Maintain a symbol table that supports insert and search operations with primitive-type keys.
   _____ Maintain a symbol table that supports insert, search, sort and select operations with comparable keys.
   _____ Maintain a symbol table where keys are long strings.

4. **DFS trace** (15 points). Consider the following recursive depth-first search implementation for *undirected* graphs. Assume that `Graph G` is an instance variable of the class.

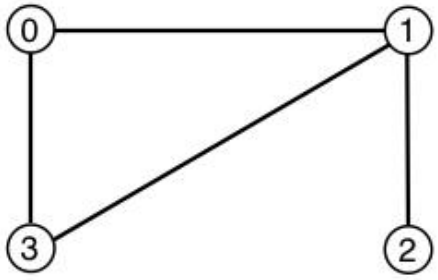
```
private void dfs(int v)
{
   marked[v] = true;
   for (int w : G.adj(v))
      if (!marked[w]) dfs(w);
}
```

At left is a trace of DFS for the call `dfs(0)` in a certain graph (made by instrumenting the `if` statement to print `check w` for marked vertices and `dfs(w)` for unmarked vertices, and adding a statement to print `done v` as the last statement of `dfs()`, all with appropriate indenting). To the right of the trace, draw the graph and give its adjacency lists. Then on the next page give a trace in the same style for the call `dfs(3)` in the graph on the next page.

A (4 points). Graph drawing **B** (4 points). Adjacency lists

```
dfs(0)
  dfs(3)
    check 0
    dfs(1)
      check 3
      dfs(2)
        check 1
        check 0
      2 done
      check 0
    1 done
  3 done
  check 2
  check 1
0 done
```

C (7 points). In the style of the example on the previous page, trace `dfs(3)` for the graph drawn below. Assume that all adjacency lists are in ascending order.



`dfs(3)`

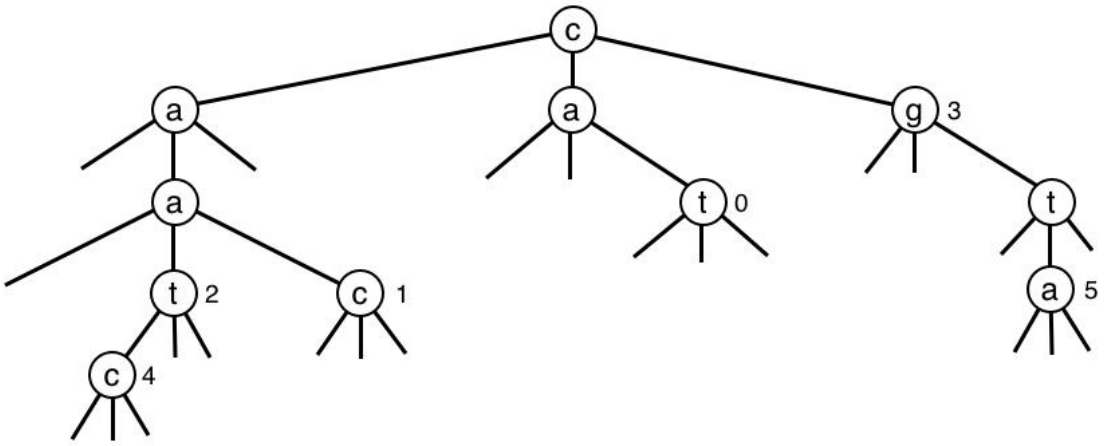
5. **LZW compression** (15 points). Complete the line labeled `out` and the following table for computing the LZW compression of the string

A A A A A B A A A A A B.

```
in:   A   A   A   A   A   A   B   A   A   A   A   A   B
out:  41  81                                            80
```

key	value
A A	81
	82
	83
	84
	85
	86
	87
	88
	89
	...

6. **TST** (15 points). Consider the TST below, which represents a symbol table.



A. (6 points) List the keys in the symbol table, in alphabetical order.

B. (9 points) Draw on the TST above the result of inserting the key-value pairs (`cab`, 6), (`abc`, 7) and (`t`, 8) into the symbol table.

7. **String sorts** (15 points). Suppose that a large array is to be sorted, where keys are random 100-character account identifiers. Fill in each entry in the table below with

   Y if the given algorithm (the standard version in the book) has the given property.
   N otherwise.

   For the purposes of this question, "sublinear" means "examines a small fraction of the characters in the keys" and "inplace" means "uses ~T space" where T is the total amount of space needed to hold the data.

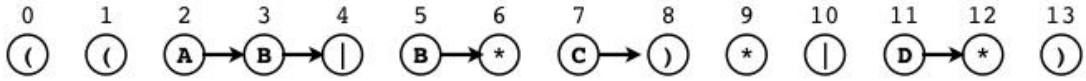
	stable	inplace	sublinear
quicksort			
mergesort			
LSD string sort			
MSD string sort			
3-way string quicksort			

8. **Regular Expression pattern matching** (15 points).

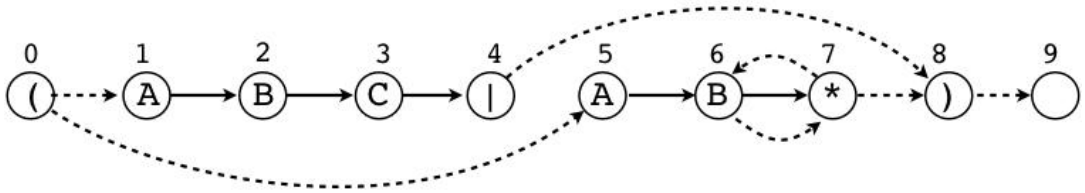
A. (8 points) Drawn below is a partial diagram of an NFA (nondeterministic finite state automata) that recognizes the same language that the regular expression

```
((AB | B*C )* | D* )
```

describes. Complete the diagram by drawing all the null (epsilon) transitions.



B. (7 points). Simulate the operation of the NFA drawn below as it recognizes the string ABB, by writing a set of states in the blank to the right of each description. Null transitions are indicated with dotted lines.



set of states reachable from null transitions from start _____

set of states reachable after matching A _____

set of states reachable via null transitions after matching A _____

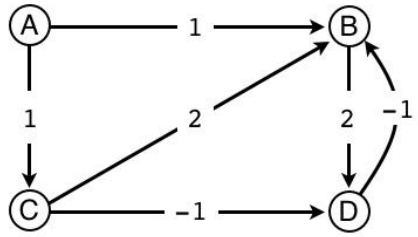
set of states reachable after matching AB _____

set of states reachable via null transitions after matching AB _____

set of states reachable after matching ABB _____

set of states reachable via null transitions after matching ABB _____

9. **Bellman-Ford** (15 points). Consider the edge-weighted digraph drawn at left, which has negative weights but no negative cycles. Your task is to finish the trace below of the Bellman-Ford algorithm, for an implementation that starts with vertex A, proceeds in V phases, relaxing all of the edges in the graph in each phase. For the purpose of this question, to *relax* an edge is to check for evidence of a shorter path to its target than the best known so far, and a *successful relax* is one where a shorter path is found. The first phase is done for you.



		successful relax?	*shortest known distance to*			
			A	B	C	D
			0			
phase 1	D→B					
	C→D					
	C→B					
	B→D					
	A→C	**X**			**1**	
	A→B	**X**		**1**		
phase 2	D→B					
	C→D					
	C→B					
	B→D					
	A→C					
	A→B					
phase 3	D→B					
	C→D					
	C→B					
	B→D					
	A→C					
	A→B					
phase 4	D→B					
	C→D					
	C→B					
	B→D					
	A→C					
	A→B					

10. **Prefix-free codes** (15 points). Consider the following 36-character text string:

F C F C E C A C B D E D F E A B F B A F F C D C B E D F F F C C D E E F

The following table defines four variable-length codes for encoding the above string.

symbol	*frequency*	code 1	code 2	code 3	code 4
A	3	011	011	1110	100
B	4	010	010	1111	101
C	8	00	00	00	01
D	5	110	101	110	110
E	6	001	100	10	111
F	10	10	11	01	00

In the space to the right of each code name below, match the codes with the following properties. Write as many letters as apply. If none apply, write *none*. Answers left blank will be marked incorrect.

A. Prefix-free code.

B. Huffman code (could be created by the Huffman algorithm, assuming that subtrees might be put in either order when merged).

C. Optimal prefix-free code.

code 1 _____

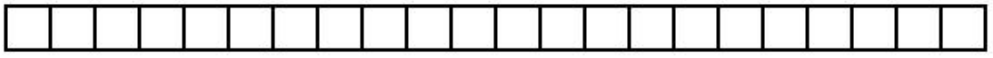
code 2 _____

code 3 _____

code 4 _____

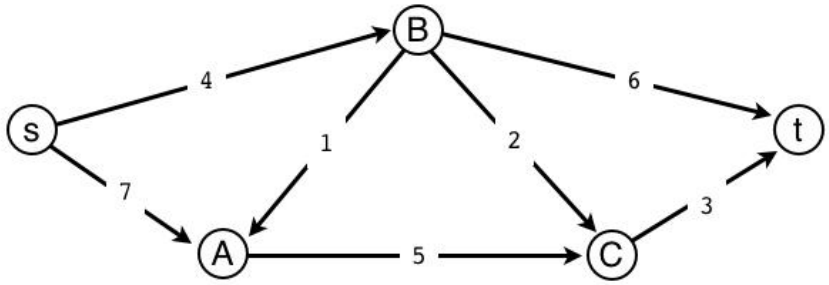
11. **3-way partitioning** (9 points). Fill in the diagram below with the result of partitioning the array with 3-way quicksort partitioning (taking the H at the left as the partitioning item). Also give the number of exchanges.

H U M U M H U M U N U K U N U K U A P U A A



Number of exchanges _____

12. **Linear programming** (15 points). Consider the following flow network with source `s` and sink `t` (numbers on edges represent capacities).



Formulate, but do not solve, the maxflow problem for this network as a linear program. Your linear program does *not* need to be in standard form.

13. **Subsequences** (20 points). A *subsequence* of a sequence is a subset of the sequence that preserves the order of the elements. For instance, suppose that the file, `tiny.txt`, contains the sequence

```
3  1  6  9  5  12  8  7  0  11  2  14
```

Examples of subsequence of this sequence are `3 0 2` and `6 12 8 7 14`. Now, define a *d-subsequence* to be an increasing subsequence where each element differs from the previous by exactly d (if the length of the subsequence is greater than 1). For instance, `5 8 11 14` is a 3-subsequence of the sequence above.

The code below implements a Java program `Subsequence` that takes an integer `d` as command-line argument and prints on standard output the maximal `d`-subsequences in the sequence of integer values on standard input. Several code snippets are missing.

Assume that integers are distinct.

```
public class Subsequence {
   public static void main(String[] args) {
      int d = Integer.parseInt(args[0]);
      RedBlackBST<Integer, Queue<Integer>> st;
      st = new RedBlackBST<Integer, Queue<Integer>>();
      while (!StdIn.isEmpty()) {
         int next = StdIn.readInt();
         if (!st.contains(next))
            // Missing line A
         // Missing line B
         // Missing line C
         q.enqueue(next);
         // Missing line D
      }

      for (int val : st.keys()) {
         for (int x : st.get(val))
            StdOut.print(x + " ");
         StdOut.println();
      }
   }
}
```

```
% java Subsequence 3 < tiny.txt
0
1
2
7
3 6 9 12
5 8 11 14
```

The implementation of `Subsequences` on the previous page uses a symbol table of queues to accomplish the task. Answer the following questions, which address the missing code and performance.

A. (3 points) Missing line A uses the `put()` method in `RedBlackBST` to associate an empty queue with an integer key. Fill in line A with *one line of code*.

B. (3 points) Missing line B declares a local variable `q` and uses the `get()` method in `RedBlackBST` to retrieve a queue from the symbol table and store a reference to it in `q`. Fill in line B with *one line of code*

C. (3 points) Missing line C uses the `delete()` method in `RedBlackBST` to remove a queue from the symbol table. Fill in line C with *one line of code*.

D. (3 points) Missing line D uses the `put()` method in `RedBlackBST`. Fill in line D with *one line of code*.

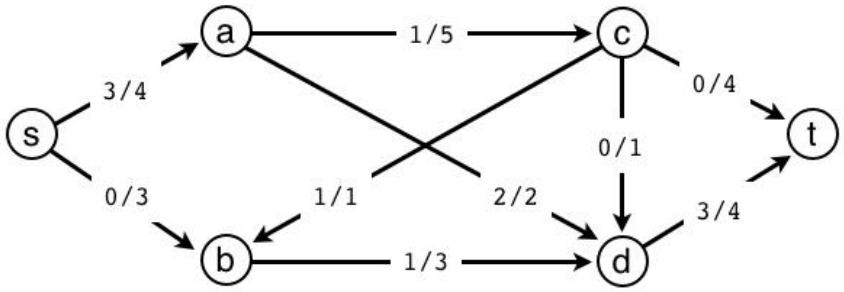
For Parts E and F, assume that `Queue` uses the standard linked-list implementation and `RedBlackBST` uses a left-leaning red-black BST. Give your answers in terms of N, the number of values on standard input and Q, the number of maximal subsequences.

E. (4 points) Give the order of growth of the total running time, in the worst case.

F. (4 points) Give the order of growth of the total memory usage, in the worst case.

14. **Maxflow** (16 points). This question contains modified multiple choice questions. Each of these questions may have multiple correct answers. You should circle ***all*** that apply. If none apply write the word **none** to the left of the choices.

A. Consider the following network with source s and sink t. Each edge is given an annotation of the form "*f*/*c*" where *c* is capacity and *f* is flow along this edge.



Circle the augmenting paths in the list below.

i. `s->a->c->t`
ii. `s->a->d->t`
iii. `s->b->c->t`
iv. `s->b->d->a->t`
v. `s->b->d->a->c->t`
vi. `s->b->d->c->t`
vii. `s->a->d->b->c->t`
viii. `s->a->c->d->b->c->t`

B. Suppose that a flow in a flow network has value *x*, but that no flow in this network has value larger than *x*. Circle the true statements in the list below.

i. The net flow across some cut is exactly equal to *x*.
ii. The net flow across every cut is exactly equal to *x*.
iii. The capacity of some cut is equal to *x*.
iv. The capacity of every cut is equal to *x*.
v. The capacity of every cut is at most *x*.
vi. The capacity of every cut is at least *x*.
vii. There exists an augmenting path for this flow.
viii. There does not exist an augmenting path for this flow.

15. **Intractability** (15 points). Fill in each entry of the following table with

Y if the given problem is known to be in the given class,
N if the given problem is known not to be in the given class, or
? if it is not known whether or not the given problem is in the given class.

If your answer depends on the assumption that **P** is not equal to **NP**, mark it with an asterisk. Entries left blank will be marked incorrect.

	P	**NP**	**NP**-complete
linear equation satisfiability			
linear inequality satisfiability			
0-1 integer linear inequality satisfiability			
boolean satisfiability			
integer factoring			