Name:____

CSE373 Fall 2013, Second Midterm Examination
November 15, 2013

Please do not turn the page until the bell rings.

Rules:

- The exam is closed-book, closed-note, closed calculator, closed electronics.
- **Please stop promptly at 3:20.**
- There are **102 points** total, distributed **unevenly** among **8** questions (many with multiple parts):

Question	Max	Earned
1	14	
2	12	
3	11	
4	10	
5	8	
6	11	
7	27	
8	9	

Advice:

- Read questions carefully. Understand a question before you start writing.
- **Write down thoughts and intermediate steps so you can get partial credit. But clearly circle your final answer.**
- The questions are not necessarily in order of difficulty. **Skip around.** Make sure you get to all the problems.
- If you have questions, ask.
- Relax. You are here to learn.

Name:____

1. (**14** points) The uptrees used to represent sets in the union-find algorithm can be stored in two n-element arrays. The `up` array stores the parent of each node (or `-1` if it has no parent). The `weight` array stores the number of items in a set if the node is the representative node of a set (else the `weight` entry for the node does not matter and can be anything).

   The following shows a collection of sets containing the numbers 1 through 12, without the `weight` array filled in:

up	-1	10	-1	9	1	1	11	-1	7	11	-1	1
weight												
	1	2	3	4	5	6	7	8	9	10	11	12

   (a) Draw a picture of the uptrees represented by the data in the `up` array shown above.

   (b) Fill in the `weight` array above so that all entries that need to be correct are correct.

   (c) Suppose we did not keep the `weight` array updated as operations are performed. As you did in part (a), the algorithm could compute weights as needed. Would this asymptotically slow down `find` operations that use path compression?

   (d) Suppose we did not keep the `weight` array updated as operations are performed. As you did in part (a), the algorithm could compute weights as needed. Would this asymptotically slow down `union` operations that use union-by-weight?

   (e) Show the result of performing the operation `find(4)` with path compression by doing both of the following:

   - Redraw below any uptrees (from part (a)) that change as a result.
   - Update the array representation as appropriate by drawing a single slash ("/") through any numbers that change and writing the new number next to it.

Name:____

2. (**12** points) Short answer:

(a) Your maze-creation homework used your union-find data structure. Your union-find data structure implemented union-by-weight, which can do one of two different things when unioning sets with equal weight. Does the choice affect what maze gets created?

(b) Assume a `find` operation in union-find does path compression. Circle all of the following that are true:

- The operation that does the compression gets faster.
- The operation that does the compression gets slower, but only by a constant factor.
- The operation that does the compression gets asymptotically slower.
- Future operations may get faster.
- Future operations may get slower, but only by a constant factor.
- Future operations may get asymptotically slower.

(c) What is $\log_2(1,000,000,000)$ to the nearest whole number?

(d) What is $\log_2^*(1,000,000,000)$ to the nearest whole number, where $\log_2^*$ is the "log-star" operation we discussed when analyzing union-find?

(e) When studying amortization, we learned a way to implement a queue using two stacks. Suppose such a queue is used with a total of x `enqueue` operations and y `dequeue` operations in some order.

- In the approach we studied, what is the asymptotic worst-case running time of a <u>single</u> `dequeue` operation in terms of x and/or y?

- In the approach we studied, what is the asymptotic worst-case <u>total</u> running time of <u>all</u> `dequeue` operations in terms of x and/or y?

(f) Your friend says she does not like implementing algorithms that have amortized run-time guarantees because amortization makes it harder to debug her code. Why does this argument make no sense? (A one sentence answer is probably enough.)

Name:____

3. (**11** points) Consider a hashtable with separate chaining with N buckets and k items currently in the table.

   (a) k/N is the definition of a term used when discussing hashing. What is this term?

   (b) Is it necessary that $k < N$?

   (c) What is the average number of items in a bucket?

   (d) In the worst-case, how many items could be in a single bucket?

   (e) If $k > N$, is it possible that any buckets are empty?

   (f) If we resize the table to a table of size $2N$, what is the asymptotic running time in terms of k and N to put all the items in the new table?

Name:____

4. (**10** points)

   (a) Fill in the contents of the hash table below after inserting the items shown. To insert the item k, use the hash function k % TableSize and resolve collisions with **quadratic probing**.

   Insert: 74, 924, 83, 113, 5

0	1	2	3	4	5	6	7	8	9

   (b) We now consider looking up some items that are not in the table after doing the insertions above. For each, give the list of buckets that are looked at *in order* before determining that the item is not present. Include all the buckets examined, whether or not they contain an item.

      i. 65

      ii. 76

      iii. 100

Name:____

5. (**8** points) For each of the following errors when using hashing and hashtables, give the best answer as to what can go wrong. Notes:

   - "Not terminate" is often described as "go into an infinite loop."
   - The choices are the same in each problem.

   (a) You are using open addressing with quadratic probing and you allow the table (whose size is a prime number) to become more than half full:

      i. A lookup operation may not terminate.
      ii. A lookup operation may not find a value that is actually in the table.
      iii. Both (i) and (ii).
      iv. Neither (i) nor (ii).

   (b) You are using open addressing with quadratic probing and you delete an item by removing it from the table and leaving the bucket it held empty.

      i. A lookup operation may not terminate.
      ii. A lookup operation may not find a value that is actually in the table.
      iii. Both (i) and (ii).
      iv. Neither (i) nor (ii).

   (c) You are putting objects of a class you defined into the Java standard library's hashtable. Your class overrides `equals` but not `hashCode`.

      i. A lookup operation may not terminate.
      ii. A lookup operation may not find a value that is actually in the table.
      iii. Both (i) and (ii).
      iv. Neither (i) nor (ii).

   (d) You write a really bad hash function that causes all objects to initially hash to the same bucket.

      i. A lookup operation may not terminate.
      ii. A lookup operation may not find a value that is actually in the table.
      iii. Both (i) and (ii).
      iv. Neither (i) nor (ii).

Name:____

6. (**11** points) ***Don't miss part (b).***

(a) The Java code below provides an adjacency-list representation for a directed graph where the n nodes are labeled with the numbers 0, 1, ... $n-1$. Complete the started-for-you method `printDoubleEdges` so that it prints one line for each pair of nodes i and j where there is an edge from i to j and an edge from j to i. Do not print self-edges (so i is not equal to j). The line printed should have the lesser number first, so we might see an output line like `13 17` but not `17 13`. No line should be printed twice. You should need somewhere around 10 lines of code – not necessarily exactly 10, but to give you a sense if you are writing far too much or far too little.

```
public class ListNode {
    public int x;
    public ListNode next;
}
public class Graph {
    // Adjacency list representation with n nodes where n is also the array length.
    // Out edges for node i are in array index i.
    private ListNode [] adjacencyLists;
    public Graph() {
        // ... constructor not shown; assume it is correct
    }
    private void printPair(int i, int j) {
        System.out.println(i + " " + j);
    }
    public void printDoubleEdges() {
        for(int i=0; i < adjacencyLists.length; i++) {
            ListNode dests = adjacencyLists[i];
            while(dests != null) {
                  // YOUR CODE GOES HERE
            }
        }
    }
}
```

(b) Give a tight asymptotic worst-case running-time bound for your code in terms of $|V|$ the number of nodes, $|E|$ the number of edges, and d the maximum out-degree of any node.

Name:____

7. (**27** points) These three questions about graphs all have the same subparts. Note that for parts (iii), (iv), and (v), your answer should be in terms of an arbitrary k, not assuming $k = 4$.

   (a) Suppose a directed graph has k nodes, where each node corresponds to a number (1, 2, ..., k) and there is an edge from node i to node j if and only if $i < j$.

      i. Draw the graph (using circles and arrows) assuming $k = 4$.
      ii. Draw an adjacency matrix representation of the graph assuming $k = 4$.
      iii. In terms of k, exactly how many edges are in the graph?
      iv. Is this graph dense or sparse?
      v. In terms of k (if k is relevant), exactly how many correct results for topological sort that does this graph have?

   (b) Suppose a directed graph has k nodes and every possible edge except there are no edges from nodes to themselves

      i. Draw the graph (using circles and arrows) assuming $k = 4$.
      ii. Draw an adjacency matrix representation of the graph assuming $k = 4$.
      iii. In terms of k, exactly how many edges are in the graph?
      iv. Is this graph dense or sparse?
      v. In terms of k (if k is relevant), exactly how many correct results for topological sort that does this graph have?

   (c) Suppose a directed graph has k nodes, where one "special" node has an edge from itself to every other node except itself and there are no other edges at all in the graph.

      i. Draw the graph (using circles and arrows) assuming $k = 4$.
      ii. Draw an adjacency matrix representation of the graph assuming $k = 4$.
      iii. In terms of k, exactly how many edges are in the graph?
      iv. Is this graph dense or sparse?
      v. In terms of k (if k is relevant), exactly how many correct results for topological sort that does this graph have?

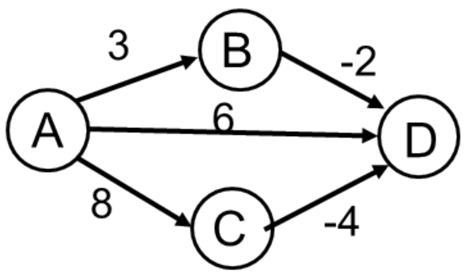
Name:____

8. (**9** points) Dijkstra's algorithm for computing lowest-cost paths from a single source node is always correct for graphs without negative-cost edges. If a graph has negative-cost edges, the algorithm might or might not give the right answer. For each directed graph below:

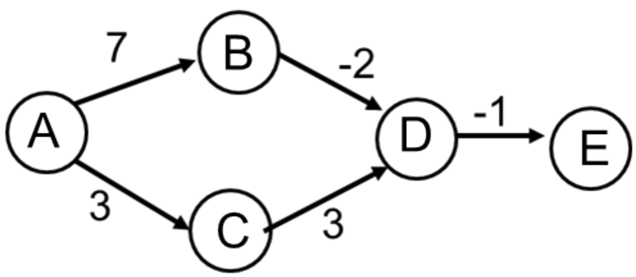
- If running Dijkstra's algorithm with start node A produces the correct lowest-cost path to every other node, then say "correct."
- Else list all nodes for which Dijkstra's algorithm produces the wrong path. For all such nodes, write the path Dijkstra's algorithm produces and write the correct lowest-cost path.

You do not need to show your work.

(a)



(b)



Name:____

An extra page in case you find it useful