CSE 373, Spring 2011
Final Practice Problems

Hashing

1. Simulate the behavior of a hash table that uses **linear probing** as described in lecture. Assume that the starting table size is 5, that we are storing objects of type `Integer` and that the hash function returns the `Integer` key's `int` value, mod (remainder) the size of the table, plus any probing needed. Assume that rehashing occurs at the start of an add where the load factor is 0.5.

   Given the following set of calls:

```
Set<Integer> set = new HashSet<Integer>();
set.add(15);
set.add(5);
set.add(13);
set.add(24);
set.add(32);
set.remove(13);
set.add(17);
set.add(44);
set.remove(15);
set.add(47);
```

   Draw the final state of the hash table array below, showing all elements of the array. Leave a box empty if an array entry is never used. Use the character "R" for any array entry that is used but later removed.

2. Simulate the behavior of a hash table set as described in lecture. Assume the following:

   - we are storing objects of type `Integer` into a hash table with array length of 11
   - the set uses **quadratic probing** for collision resolution
   - the hash function uses the key's `int` value, mod the size of the table, plus any probing needed
   - the table does not enlarge or rehash itself during this problem
   - an insertion is considered to have failed if more than half of the entries are tried (make note of any such failed insertions)
   - the table stores a special character "R" for any array entry that is used but later removed

   a. Given the following lines of code, draw the final state of the hash table below. Leave a box empty if an array element is unused.

```
Set set = new HashTableSet(11);
set.add(4);
set.add(52);
set.add(50);
set.add(39);
set.add(29);
set.remove(4);
set.remove(52);
set.add(70);
set.add(82);
set.add(15);
set.add(18);
```

   b. What is the size of the final `HashTableSet`?

   c. What is the capacity of the final `HashTableSet`?

   d. What is the load factor of the `HashTableSet`?

   e. Give an example of a search on this hash table that would cause approximately half the elements to be searched.

   f. Could a value fail to be inserted into the hash table?

3. Write the `hashCode` method for the class below. Your `hashCode` should return the same hash code value for equal objects as defined by the `equals` method found below. Your `hashCode` should minimize collisions. You may assume that any field in `TimeSpan` has an `equals` and a good implementation of `hashCode`.

```
public class TimeSpan {

   private int hours;
   private int minutes;
   private int seconds;

   public TimeSpan(int hours, int minutes, int seconds) {
         this.hours = hours;
         this.minutes = minutes;
         this.seconds = seconds;
   }

   public boolean equals(Object o) {
       if (!(o instanceof TimeSpan)) {
           return false;
       }

       TimeSpan other = (TimeSpan) o;

       int timeInSec = 3600 * hours + 60 * minutes + seconds;

       int oTimeInSec = 3600 * other.hours
                    + 60 * other.minutes
                    + other.seconds;

       return timeInSec == oTimeInSec;
   }

   public int hashCode() {









   }
}
```

Graphs

4. Given the following graph:

V = {V0, V1, V2, V3, V4, V5, V6}

E = {
(V0 -> V1 : 2),
(V0 -> V3 : 1),
(V1 -> V3 : 3),
(V1 -> V4 : 10),
(V2 -> V0 : 2),
(V2 -> V3 : 1),
(V2 -> V5 : 4),
(V3 -> V2 : 2),
(V3 -> V4 : 8),
(V3 -> V5 : 1),
(V3 -> V6 : 4),
(V4 -> V6 : 6),
(V5 -> V6 : 1),
(V6 -> V4 : 2),
(V6 -> V5 : 1)
}

   a. Draw the graph.

   b. Is the graph directed or undirected?

   c. Cyclic or acyclic?

   d. Weighted or unweighted?

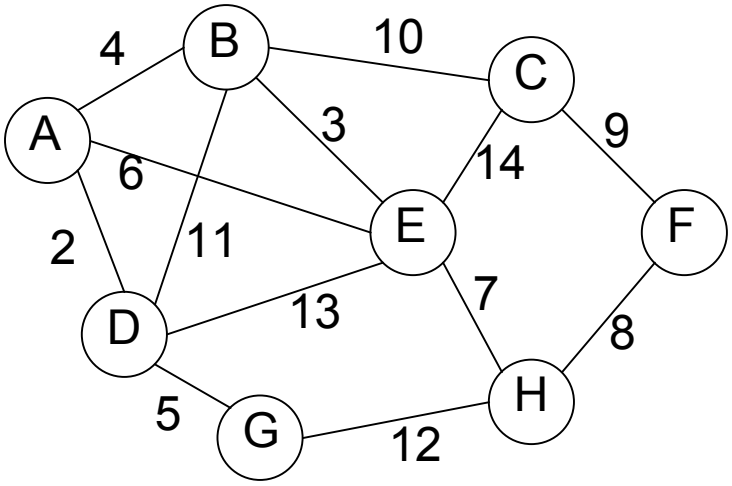
   e. What is the in-degree of vertex V3? What is its out-degree?

   f. Do a depth-first search for a path between V0 and V4. (Assume that a vertex's neighbor list is in numerical order.)

   g. Do a breadth-first search for a path between V0 and V4. Show the tables/lists computed by the algorithm. (Assume that a vertex's neighbor list is in numerical order.)

   h. Do a search for a path between V0 and V4 using Dijkstra's algorithm. Show the tables/lists computed by the algorithm. (Assume that a vertex's neighbor list is in numerical order.)

5. Minimum Spanning Trees



a. Using Prim's Algorithm starting with A, list the vertices in the order they are added to the MST.

b. Using Kruskal's Algorithm, list the edges of the MST in the order that they are added.

6. Assume we modified `Graph.java` from Programming Project #4 to represent a directed graph (i.e. when we added and edge, we only added it to the given source vertex's neighbors list and not to the destination vertex neighbor's list).

   Write a method named `numStalkers` that could be added to your Graph class from Homework 8. The `numStalkers` method accepts a vertex as a parameter and reports the number of vertices that are "stalking" a given vertex of a `Graph`.

   For example, assume that the graph's vertices represent chat buddy names, and an edge between A and B means that A has B on his/her buddy list. A "stalker" is someone who has you on their buddy list, but who you don't have on your buddy list. In other words, if A has B on his buddy list but B doesn't have A, then A is a "stalker" of B.

   What is the Big-Oh runtime of your method?

Disjoint Sets

7. Consider the set of initially unrelated elements:

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

   a. Draw the final forest of up-trees that result from the following sequence of operations using on union-by-rank. Break ties in size by keeping the root of the set the first argument belongs to as the new root. Perform a find (without path compression) if you need to find which set an element belongs to and then union the two sets as you normally would.

      `Union(0,6), Union(6,7), Union(7,9), Union(9,3), Union(0, 14), Union (5,8), Union(12,6), Union(1,12), Union(11, 2), Union(7,11), Union(4, 7)`

   b. Draw the array representation of your answer above. Use the value of (negative) size to represent a root. For example if 3 is the root of the set {4, 3, 1, 2} the element found at the 3rd index should be -4.

   c. Draw the new forest of up-trees that results from doing a Find(2) with path compression on your forest of up-trees from (a).

8. Write a method named `changeRep` that takes as parameters an integer array that represents a disjoint set and an element that is in the disjoint set. Alter the disjoint set array so that at the end of the method the element passed will be the new "representative" (i.e. the root) of its subset. If the element that was passed in was already the representative of its subset, no changes should be made to the disjoint set.

   You may assume that the initial array passed in is in a valid state.

   You may want to use the following implementation of `find` to help you write `changeRep`:

```
public int find(int[] disjointSet, int x) {
   if (disjointSet[x] < 0) {
      return x;
   } else {
      return find(disjointSet, disjointSet[x]);
   }
}
```

B-Trees

9. Given the following parameters:

    - Disk access time = 1milli-sec per byte
    - 1 Page on disk = 1024 bytes
    - Key = 16 bytes
    - Pointer = 4 bytes
    - Data = 128 bytes per record (includes key)

    What are the best values for:

    M =

    L =

10. Using the method described in class and in the book, insert the values:

    2, 3, 9, 4, 6, 1, 7, 8

    (in that order) into a B tree with L=2 and M=3.