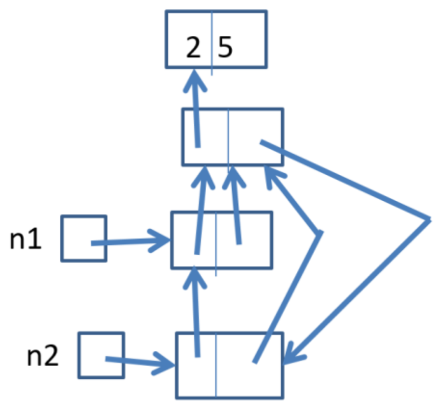
CS314 Spring 2017 Final Exam Solution and Grading Criteria.

Grading acronyms:
AIOBE - Array Index out of Bounds Exception may occur
BOD - Benefit of the Doubt. Not certain code works, but, can't prove otherwise
Gacky or Gack - Code very hard to understand even though it works. (Solution is not elegant.)
LE - Logic error in code.
NN - Not necessary. Code is unneeded. Generally no points off
NPE - Null Pointer Exception may occur
OBOE - Off by one error. Calculation is off by one.
RTQ - Read the question. Violated restrictions or made incorrect assumption.

1. Answer as shown or -1 unless question allows partial credit.



A. Lists with the same elements in a different order will have the same hash code. (OR WORDS TO THAT EFFECT.)

B. $O(N^2)$ (get on linked list is O(N))

C. See picture to the right -------------------->

D. 0.88 seconds

E. 0.06 seconds (With only 50 distinct values, O(N))

F. 4

G. -1 -2 3 0 -5 9 7 5

H. Runtime error. (IllegalStateException due to calling remove twice in a row.)

I. --->

```
       10
     /    \
    4      9
   / \    / \
  2   4  9   4
```

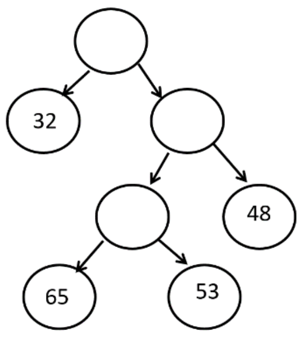
J. Vertices were countries. Edge between 2 vertices if the countries shared a non-zero length border.

K. The adjacency list. The outdegree is simply the size of the list. With an adjacency matrix we must traverse the row (or column) to count the number of edges out.

L. $O(N^2)$ Inner dependent loop is O(N).

M. An int greater than 0.

N. --->

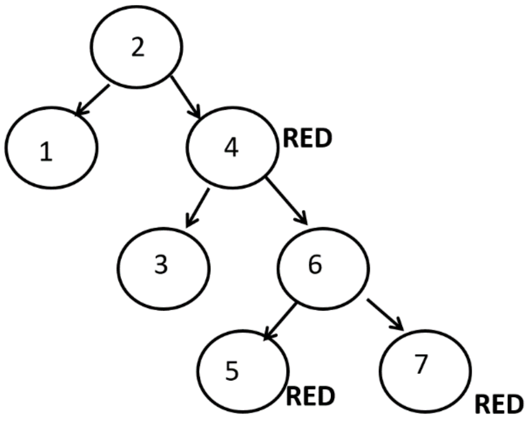


O. 4 seconds (Size of hash table doesn't matter, remove is average case O(1) from hash table.)

P. If the number of distinct values in the original is small compared to the possible number of values, the new header format with explicit codes will be shorter OR if average code length is less than 16 (OR WORDS TO THAT EFFECT.)

Q. ------------------------------------->

Must be BST, height must be 3, path rule must be 2, no red nodes in a row



R. O(NlogN), base 2 okay

S. {2=3, 3=2, 6=0} (order must match, differences in spacing and grouping symbols okay

T. 731t41021

2. Comments.

```
public LinkedList314 combine(LinkedList314 other) {
     LinkedList314 result = new LinkedList314();

     // handle special case when one or both lists are empty
     if (first == null || other.first == null) {
          return result;
     }

     // create first node in result
     result.first = new Node(first.data + other.first.data);
     Node n1 = first.next;
     Node n2 = other.first.next;
     Node n3 = result.first;

     // handle rest of nodes
     while (n1 != null && n2 != null) {
          Node newNode = new Node(n1.data + n2.data);
          n3.next = newNode;
          n1 = n1.next;
          n2 = n2.next;
          n3 = n3.next;
     }
     return result;
```

16 points , Criteria:

- create result and temporary node references for all three lists, 1 point
- loop while both lists still have elements, 4 points
- correctly handle case for first node (can be before loop), 3 points
- correctly handle general case nodes after first, 3 points
- move all three temp references, 4 points
- return result 1 point

Other penalties:

- destroy either list: -6
- other large data structure: -8 (array, other list)
- Worse than O(N), -6
- use other methods unless write: size - 6, getNode - 8
- -2 for ||instead of && logic error in while loop
- -2 for first NPE

3. Suggested Solution:
```
private boolean redRuleMet() {
      return redHelp(root);
}

private boolean redHelp(RBNode<E> n) {
      // base case if n stores null, empty tree, no problem
      if (n == null) {
            return true;
      } else {
            if (!n.isBlack) {
                  // This is a red node. If children exist, they must be black.
                  if (n.left != null && !n.left.isBlack) {
                        return false; // left child is red
                  }
                  if (n.right != null && !n.right.isBlack) {
                        return false; // right child is red
                  }
            }
            // If we get here either this is a black node or a red node with
            // black children. Check subtrees.
            return redHelp(n.left) && redHelp(n.right);
      }}
```

16 points, Criteria:

- create helper with correct params, 2 points
- base case for null. (Can include leaves), 3 points
- Check if current node is red (or if parent was red via parameter) 3 points
- Correctly check if two red nodes in a row and return false immediately if true. 4 points
  (Could be done with parameter showing parent's red status.
- correct recursive calls if this node is okay, 4 points

Other deductions:

- destroy tree: -6
- not handling nulls correctly -4

Alternate solution that sends color of parent node
```
private boolean redRuleMet() {
      return redHelp(root, false);
}
private boolean redHelp(RBNode<E> n, boolean parentRed) {
      // base case if n stores null
      if (n == null) {
            return true;
      } else {
            // first check myself against my parent
            if (parentRed && !n.isBlack) {
                  return false;
            } else {
                  // check descendants
                  return redHelp(n.left, !n.isBlack)
                              && redHelp(n.right, !n.isBlack);
      } }
```

4.
Suggested Solution:

```
    private BNode help(BNode n) {
        // base case, I'm a leaf or null
        if (n == null ||
                (n.left == null && n.right == null)) {
            return n;
        } else {
            // I'm an internal node. Fix my children.
            n.left = help(n.left);
            n.right = help(n.right);
            // if I have 2 children, done return myself
            if (n.left != null && n.right != null) {
                return n;
            } else if (n.left != null) {
                // return single child to left
                return n.left;
            } else {
                 // return single child to right
                return n.right;
            }
        }
    }
```

16 points, Criteria:

- header for helper correct, 2 points
- base case of null or leaf, 4 points
- recursive calls for internal nodes to set children, 4 points
- after setting children check I still have 2 children, return self if true, 3 points
- otherwise if single child, return that child, 3 points

Other:

- early return on has 2, -6
- no cutting of nodes, -5
-

5. Suggested Solution:

```
public boolean remove(E val) {
     int oldSize = size;
     // get the index for this bucket
     int index = val.hashCode() % con.length;
     index = Math.abs(index);
     // only need to check ff chain actually exists
     if (con[index] != null) {
          // Due to single linked nodes, must use trailer or look ahead.
          // Check special case if first node in chain stores val.
          if (con[index].data.equals(val)) {
               size--;
               con[index] = con[index].next;
          } else {
               // General case to search chain. I'll use look ahead.
               Node temp = con[index];
               while (temp.next != null && size == oldSize) {
                    E nextData = temp.next.data;
                    if (val.equals(nextData)) {
                         // found it, cut next node out of chain
                         size--;
                         temp.next = temp.next.next;
                    }
                    temp = temp.next;
               }
          }
     }
     return oldSize != size;
```

16 points, Criteria:

- get hash code, 1 point
- remainder by length of table, 1 points
- absolute value for negative numbers, 1 point
- check if index null, do nothing, 2 points
- special case when first value, 2 points
- general case look-ahead or trailer, move reference correctly 2 points
- cut node out correctly, 2 points
- use equals correctly, 2 points
- decrement size correctly, 2 points
- return correct value, 1 point

Other:

- effeciency, O(N^2) solution, -6

6. Comments:

Suggested Solution 1:
```
public boolean formAClique(Set<String> names) {
      for (String name : names) {
            Vertex v = vertices.get(name);
            int count = 0;

            for (Edge e : v.adjacent) {
                  if (names.contains(e.dest.name)) {
                        count++;
                  }
            }

            if (count != names.size() - 1) {
                  return false;
            }
      }
      return true;
}
```

Suggested Solution 2:
```
// For every vertex, ensure it is connected to each other vertex. Could improve
// efficiency some by removing vertex after checking connected to all others, but
// not clear if can alter original set
public boolean formAClique(Set<String> names) {
      for (String s : names) {
            Vertex current = vertices.get(s);
            for (String tgt : names) {
                  // don't check self
                  if (!tgt.equals(s)) {
                        // find an edge to tgt
                        boolean found = false;
                        Iterator<Edge> it = current.adjacent;
                        while (it.hasNext() && !found) {
                              String destinationName = it.next().dest.name;
                              found = tgt.equals(destinationName);
                        }
                        // if didn't find an edge to tgt this isn't a clique
                        if (!found) {
                              return false;
                        }
                  }
            }
      }
      return true;
}
```

16 points, Criteria:

- outer loop for every vertex, 2 points
- inner loop for all other vertices, 2 points
- don't check current against self, 3 points
- loop through edges of current, 2 points

- check destination and target same correctly, 3 points
- stop when find edge, 2 points
- return false as soon as we know not a clique, 2 points

Other:
create any data structures besides iterators, -4
efficiency, -3
creating new data structures, -5