Dartmouth College
Computer Science 10, Fall 2014
Final Exam A

Professor Drysdale

Print your name: _____

- If you need more space to answer a question than we give you, you may use the backs of pages or you may use additional sheets of paper and attach them to the exam. Make sure that we know where to look for your answer!
- Read each question carefully and make sure that you answer everything asked for. Write legibly so that we can read your solutions. Please do not write anything in red.
- We suggest that you include comments for solutions that require you to write Java code. They will help your grader understand what you intend, which can help you get partial credit.
- Hand in only what you want graded. We do not want your scrap paper unless it contains solutions you want us to grade.
- You may use an 8.5 x 11 inch sheet of paper with notes. These notes should be handwritten or typed in with a 10 point font or larger. You may use both sides of the paper.
- We will supply Javadoc-style documentation for any class or interface that you need to answer a question.

Question	Value	Score
1	30	
2	15	
3	10	
4	15	
5	15	
6	15	
Total	100	

Question 1 30 points

Short answer

(a) (5 points)
In binary search tree deletion the tricky case is to delete a node with two children. We instead replace it by a different node which we can easily delete. Describe the node that we delete easily, and explain why it is easy to delete.

(b) (4 points)
You wrote a `NamedAdjacencyMapGraph` class for sa10. In the sample solution the new version of `removeVertex` was:

```
public void removeVertex(Vertex<V> v) throws IllegalArgumentException {
 vertexMap.remove(v.getElement());
 super.removeVertex(v);
}
```

while the new version of `getEdge` was:

```
public Edge<E> getEdge(V uName, V vName) throws IllegalArgumentException {
    return getEdge(getVertex(uName), getVertex(vName));
}
```

Why was "super" needed in the call to `super.removeVertex(v)` but not in the call to `getEdge(getVertex(uName), getVertex(vName))`?

(c) (5 points)
In the deterministic skip list (every other item is included in the next level up) what is the maximum number of comparisons that can be made on a single level of the skip list when searching for a key? Why can't you do more comparisons?

(d) (5 points)
What are bounded buffers and how do they work? What do they have to do with Unix pipes?

(e) (6 points)
What is memoization, and under what circumstances will it speed up a computation? Under what circumstances will it do little to speed up the computation?

(f) (5 points)
Describe the difference between linear probing and double hashing. What is the main drawback to linear probing, and how does double hashing solve this?

Question 2 15 points

You will add a helper method to the BST class that will verify that the parent/child links are correct. That is, the method will check every node to see if that node's children have parent pointers that point back to it. If the child is the sentinel it will not be tested. The appropriate data declarations and some methods of the BST class are included in the accompanying documentation. Note that because `Node` is an inner class your method can access its instance variables directly, so you do not have to use the get and set methods supplied. The method that calls your method is:

```
public boolean verifyLinks() {
 return verifyLinksHelper(root) && (root == sentinel || root.parent == sentinel);
}
```

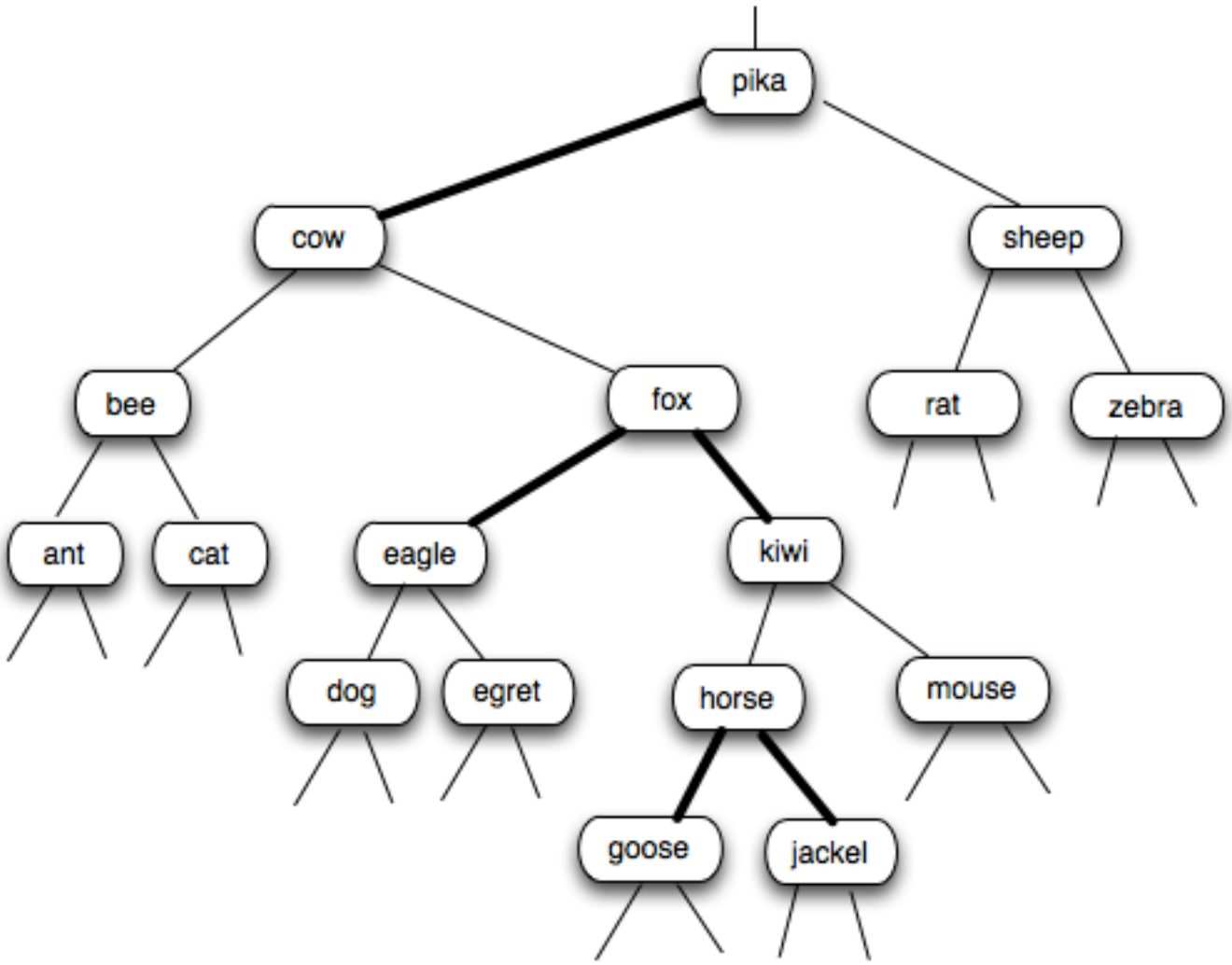
Note that it verifies that the parent pointer of the root of a non-empty tree points to the sentinel, so the helper method does not need to check this.

Complete the helper method `vertifyLinksHelper` below.

```
/**
 * Verifies that the subtree rooted at current has correct parent pointers.
 * @param current  The root of the subtree to check
 * @return  true if the parent pointers are all correct, false otherwise.
 */
public boolean verifyLinksHelper(Node current) {
```

Question 3 10 points

Consider the following red-black binary search tree, where words are ordered alphabetically. Dark edges indicate that the node at the bottom is red.



Insert "gander" into this tree and draw the updated tree on the next page. You should update the tree according to the rules of red-black tree insertion.

The red-back tree after inserting “gander”:

Question 4 15 points

A useful operation on a graph is: given a vertex v and an edge label l, find an outgoing edge e from v that has label l and return the vertex at the destination of e. So if v is vertex and there is an edge from v to u labelled "a", then the result of the operation given v and the label "a" should be u.

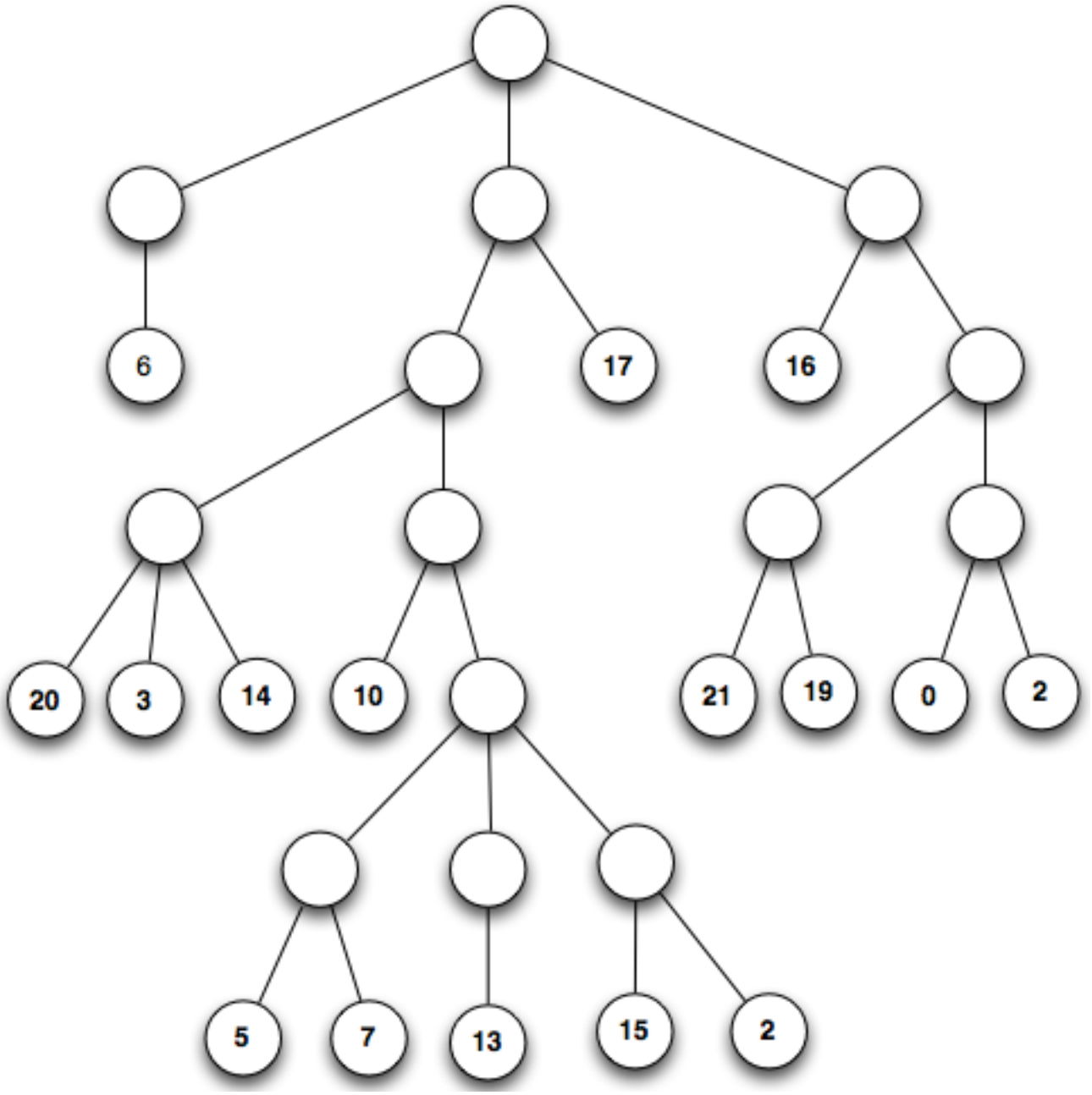
You are to implement a method `followEdge` that implements this operation. You may assume that all of the edges out of a vertex have different labels. If there is no outgoing edge with the given label the method should throw an IllegalStateException.

```
/**
 * Finds and follows an outgoing edge e from v with a given label,
 * returning the vertex at e's destination.
 *
 * @param g  the graph containing the vertex v
 * @param v  the vertex to search from
 * @param label  the label on the desired outgoing edge from v
 * @return the destination of the outgoing edge with label "label"
 */
public static Vertex<String> followEdge(Graph<String,String> g,
        Vertex<String> v, String label) {
```

Question 5 15 points

A game tree is shown below. The player on move alternates on each level. Evaluate the tree using game tree search with alpha-beta pruning. Fill in the values of the nodes that get evaluated and draw x's through the nodes that get pruned. Give the value of the game for the first player and show the best first move for that player.

You may evaluate the tree by either having maximizing and minimizing levels or by maximizing at every level but negating the values from the level below, as Kalah does.



Question 6 15 points

Write a backtracking method `generateStrings` that generates all possible strings of length n from a given alphabet that have no immediately repeated letter. (So "acc" is not allowed, but "cac" is.). The code:

```
char[] alphabet = {'a', 'b', 'c'};
generateStrings("", alphabet, 3);
```

should generate the output:

```
aba
abc
aca
acb
bab
bac
bca
bcb
cab
cac
cba
cbc
```

Complete the method whose header is on the next page.

```
/**
 * Generates all possible strings of length n from the given alphabet
 * that have no immediately repeated letter.  (So "acc" is not allowed,
 * but `"cac" is.)
 *
 * @param current  the current partial string
 * @param alphabet  the letters allowed.  No letter will be repeated.
 * @param n  the length of the desired strings.
 */
public static void generateStrings(String current, char[] alphabet, int n) {
```