Dartmouth College
Computer Science 10, Fall 2014
Solution to 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.

Solution: We replace node x by its immediate successor (or predecessor). This is the smallest item in x's right subtree, which cannot be empty because x has two children. We find this by going right once, then left until we find a node without a left child. Because it has no left child it is easy to delete by pointing its parent at its right child.

(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))`?

Solution: In the first case you are overriding the `removeVertex` method. This method and the one in the superclass both have parameter type `Vertex<V>`. Therefore without the `super` you would call the subclass `removeVertex` method recursively. In the second case you are overloading `getEdge`. The method in the superclass has two parameters of type `Vertex<V>`, while the one in the subclass has parameter types `V`. Therefore the call to `getEdge` must refer to the one in the superclass.

(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?

Solution: The maximum number of comparisons on a given level is 2. You continue searching on a given level while the key you are looking for is greater than the key in the skip list. You may be greater than the first key that you look at after you "drop down" to the current level. However, you cannot be greater than the second one. This is because the second one appeared in the list above, and the reason that you dropped down where you did is because you were less than or equal to that key on the next level up. Therefore you are always less than or equal to the second key, and will stop searching on the current level once you compare the key you are looking for to the second key on the current level.

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

Solution: Bounded buffers are circular queues with a limited amount of space. They are used to communicate between processes where one process produces output and the next process consumes it. This occurs when two processes are connected by a Unix pipe. The first process can add to the buffer until it is full, at which point it blocks until something is removed by the second process. The second process can remove things from the buffer until it is empty, at which point it blocks until something is added to the buffer.

(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?

Solution: When a recursive function is memoized, each solved subproblem is saved in a map. Before trying to solve a subproblem the problem is looked up in the map. If it is found, then the previously computed solution is returned. If not, it is solved recursively and the solution is added to the map.

It is useful when there are a relatively small number of subproblems, but they are generated over and over again (e.g. edit distance). It is not useful when each of the subproblems only arises once (e.g. merge sort).

(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?

Solution: Both use an array with no linked lists. The difference is how they handle collisions. Linear probing starts at the place specified by the hash function and walks through the array one step at a time looking for an empty spot to insert (wrapping if necessary). It similarly walks through one step at a time to find a value, quitting when the value or an empty place is found. Double hashing has two hash functions. The first says where to begin in the table to insert or search. The second says how large a step to to take (wrapping if necessary).

The drawback to linear probing is that clusters form, and the larger the cluster gets the more likely the cluster will grow at the next insertion. That is because once two keys collide they follow the same path and collide with the same things. Therefore an insertion that collides with anything in a cluster results in the cluster growing at the end. With double hashing if two keys collide they are unlikely to collide again, because each will step through by its own step size. Therefore clusters are less likely to form.

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.

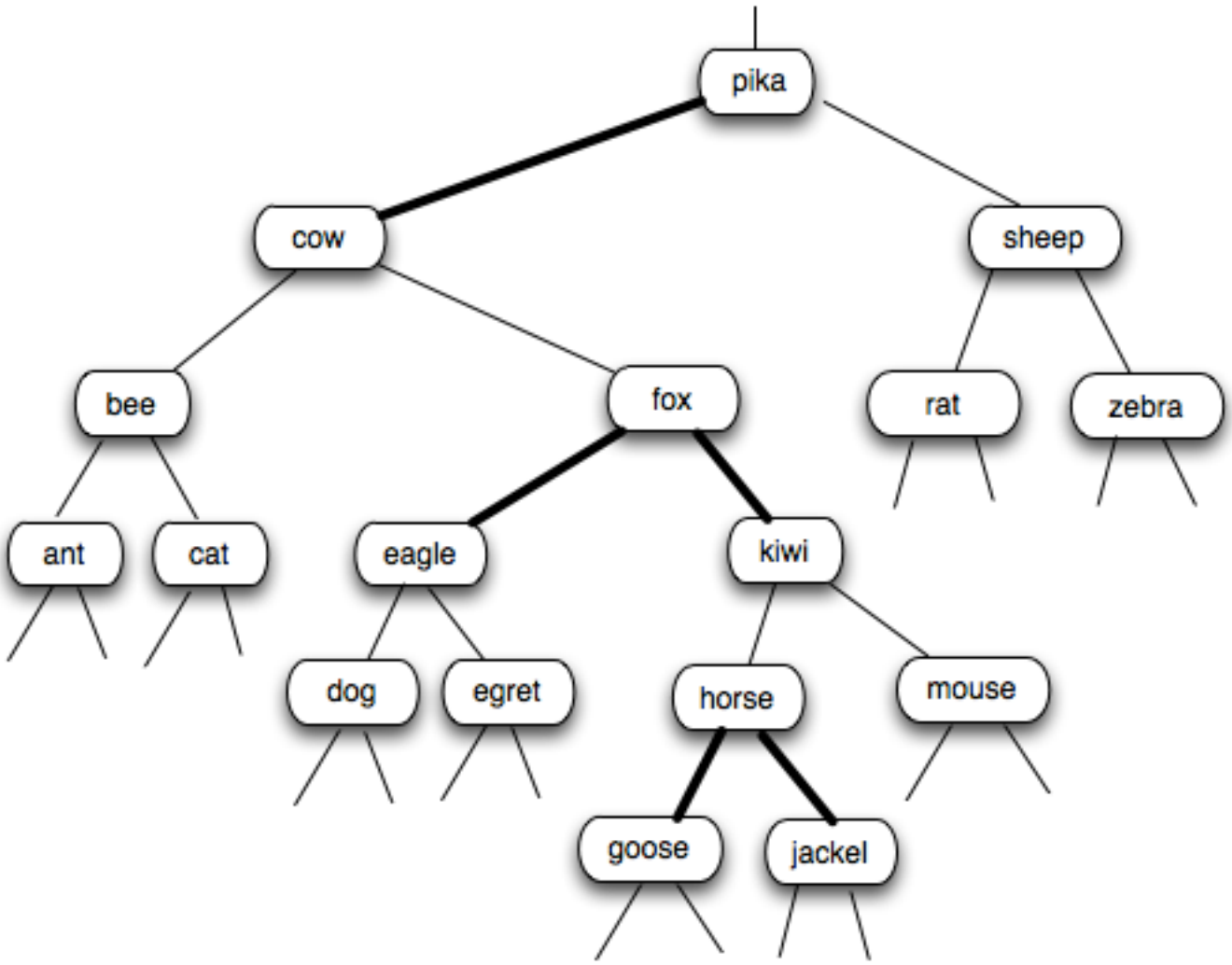
```
/**
 * Verifies that the subtree rooted at current has correct parent pointers.
 * @param current  The root of the subtree to check
 * @return  true if the parent pointers are all correct, false otherwise.
 */
public boolean verifyLinksHelper(Node current) {
```

Solution:

```
if(current == sentinel)
  return true;
else if(current.left != sentinel && current.left.parent != current)
  return false;
else if(current.right != sentinel && current.right.parent != current)
  return false;
else
  return verifyLinksHelper(current.left) && verifyLinksHelper(current.right);
}
```

Question 3 10 points

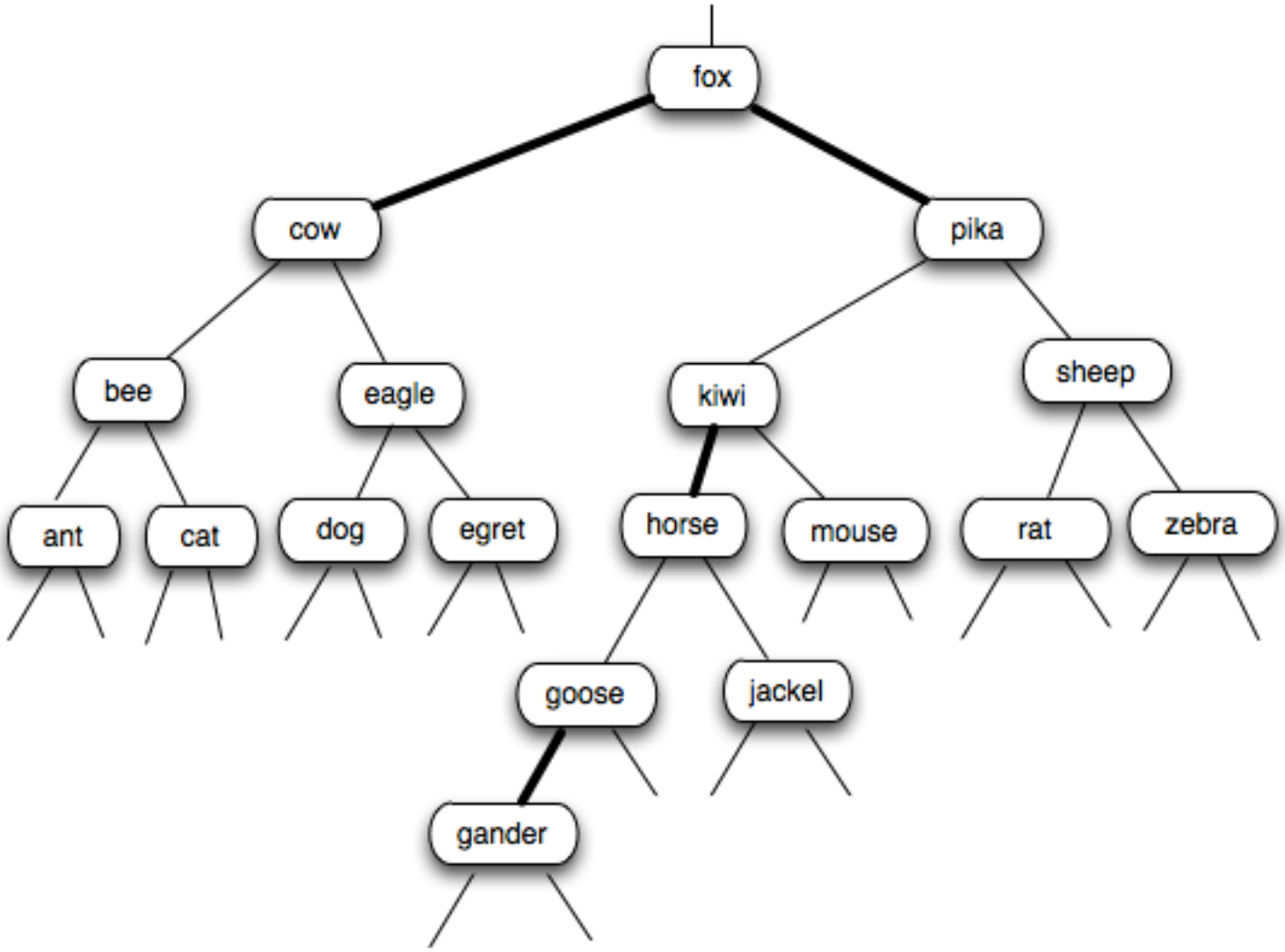
Consider the following red-black binary search tree, where words are ordered alphabetically. Dark edges indicate that the node at the bottom is red.



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":

Solution:



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) {
```

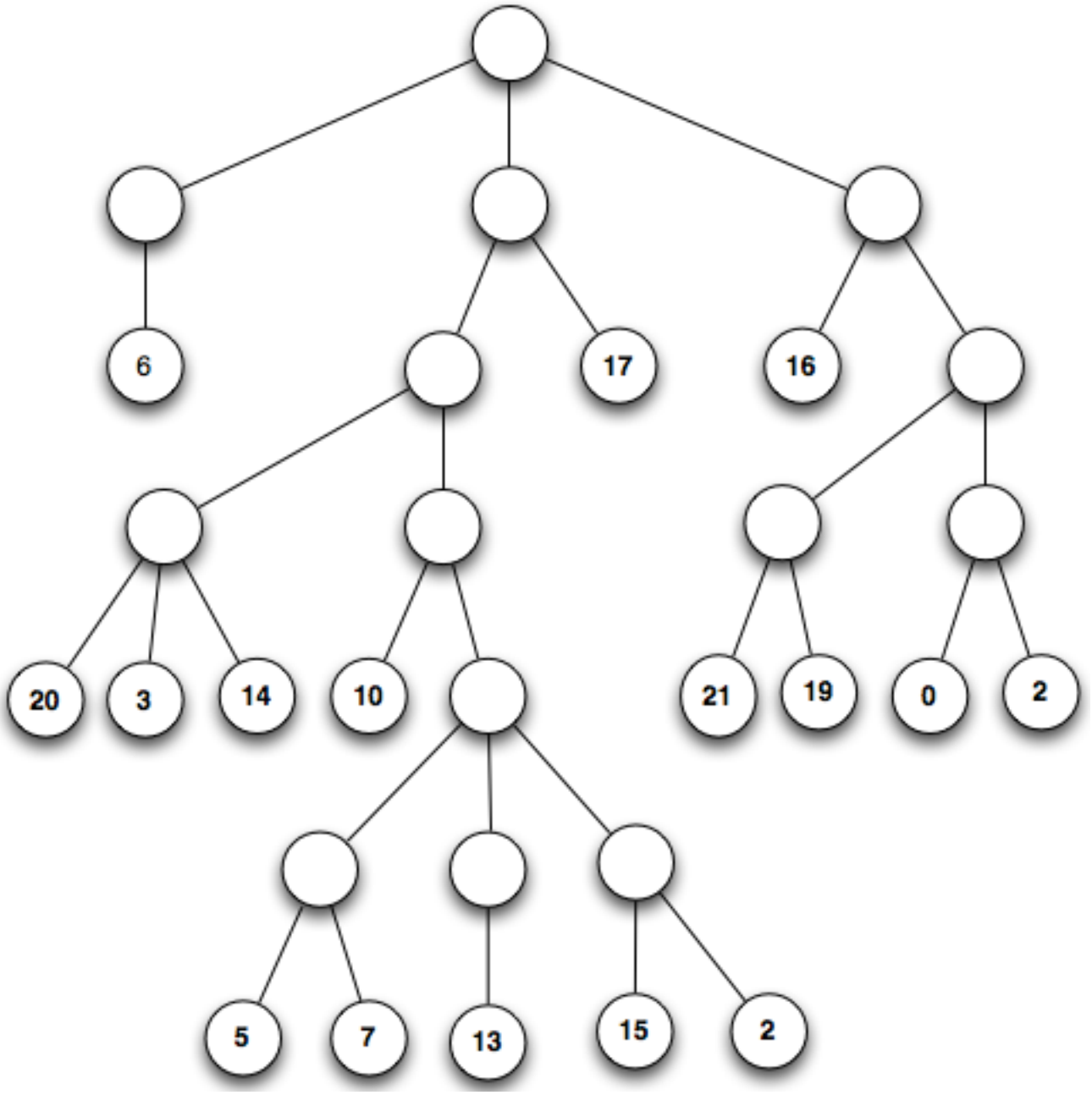
Solution:

```
  for(Edge<String> e: g.outgoingEdges(v)) {
    if(label.equals(e.getElement())) {
      return g.opposite(v, e);
    }
  }
  throw new IllegalStateException();
}
```

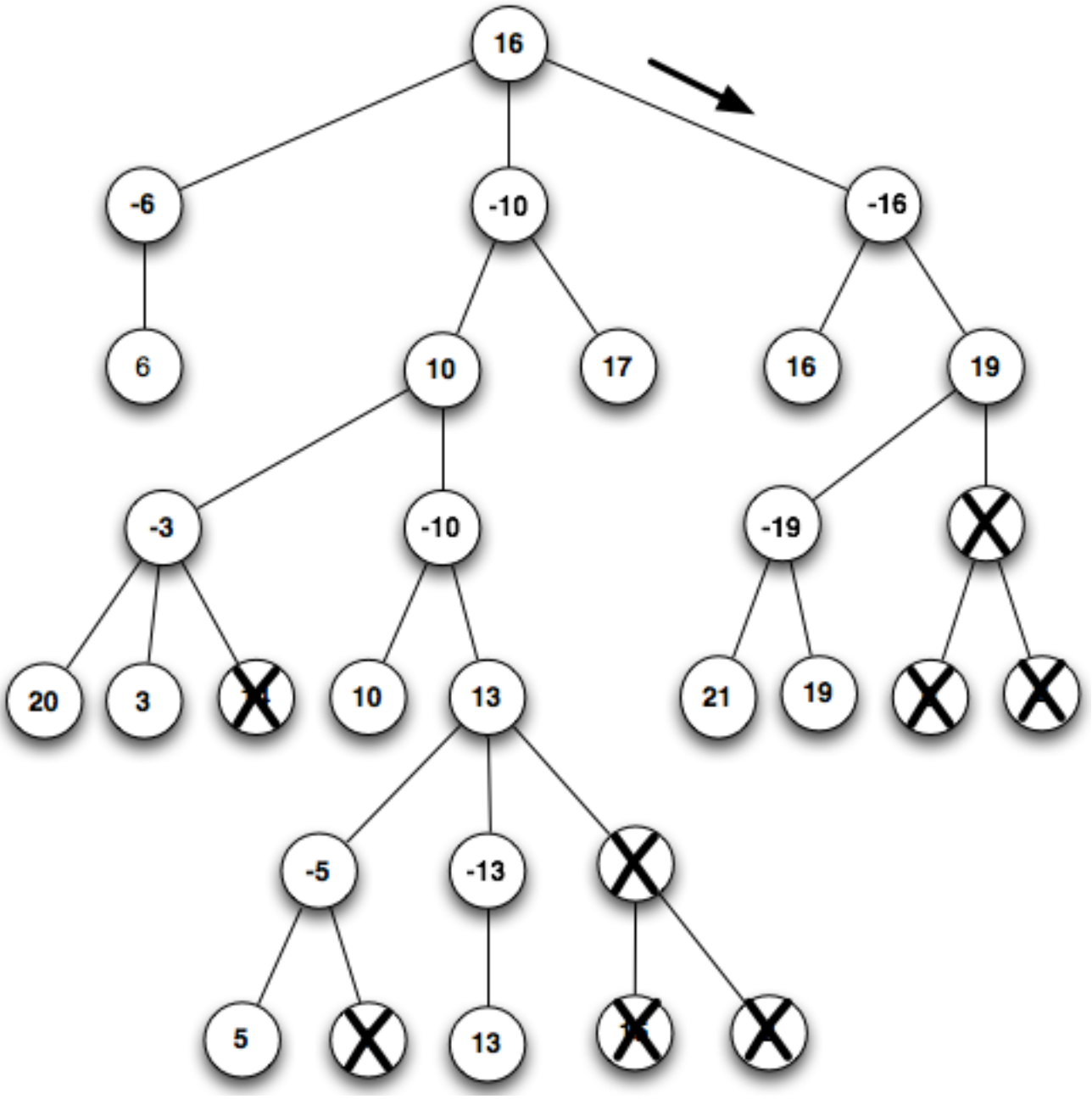
Question 5 15 points

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.



Solution:



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) {
```

Solution:

```
  if (current.length() >= n)
    System.out.println(current);
  else {
    for(char letter:alphabet) {
      if(current.length() == 0 ||letter != current.charAt(current.length()-1))
        generateStrings(current + letter, alphabet, n);
    }
  }
 }
```