Dartmouth College
Computer Science 10, Winter 2012
Final Exam

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	20	
4	15	
5	20	
Total	100	

Question 1 30 points

Short answer

(a) (4 points)
Suppose that you are implementing a GUI for a pizza-order web page. What GUI component would you use to allow the user to pick the size of the pizza? What GUI component would you use to allow the user to select the set of toppings on the pizza? Why?

(b) (4 points)
What is the horizon effect in game tree search? How can it lead to seemingly irrational play?

(c) (4 points)
What is an admissible heuristic in A* search, and how can using an non-admissible heuristic lead to finding a suboptimal solution?

(d) (4 points)
A student failed to override `hashCode` in PS-6, and ended up evaluating so many nodes that the program ran out of memory on the 8-puzzle given as a test case. Explain why it is important to override `hashCode` when you override `equals` and how failure to do so could lead to the behavior described.

(e) (4 points)
How does Norvig's spell corrector, which we saw in class, avoid computing the edit distance between the word to be corrected and every word in the dictionary?

(f) (6 points)
What is the Decorator design pattern? How is it implemented? Explain how this pattern allowed us to implement a graph with directed edges.

(g) (4 points)
Consider the following proposed solution to the Dining Philosophers problem. Each philosopher has an `eat` method given in pseudocode below. Assume that the forks are implemented with methods tryToPickUpFork and putDownFork. The tryToPickUpFork method does not wait for the fork to be available but returns immediately. It returns true if it succeeds in picking up the fork and false if it fails because the fork is in use.

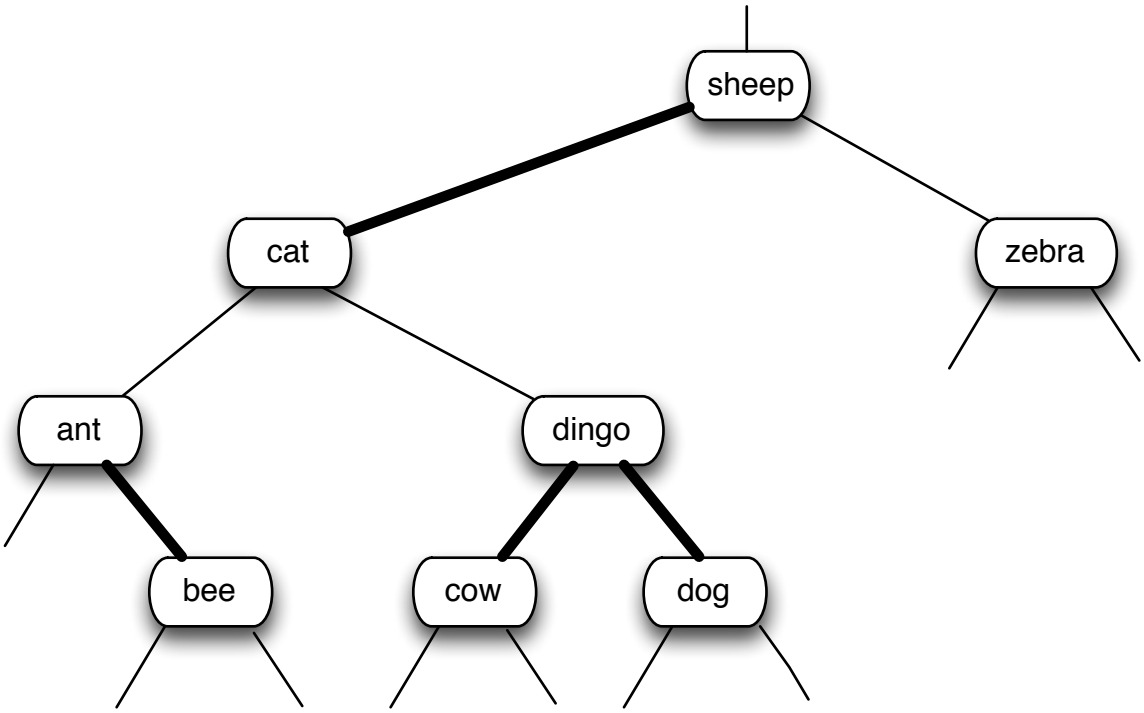
```
Method eat:
  while(true) {
    Think

    while(this philosopher does not hold two forks) {
      while(! tryToPickUpFork(left fork))
        ;
      if (! tryToPickUpFork(right fork)) {
        putDownFork(left fork)
        Think
      }
    }

    Eat
    putDownFork(left fork)
    putDownFork(right fork)
  }
```

That is, the philosopher waits for the left fork to become available. He or she then tries to pick up the right fork. If this succeeds the philosopher eats. If not the philosopher puts down the left fork, thinks a while, and tries again.

Can this deadlock? If so, explain how that can happen. If not, explain why not. Can it lead to starvation? If so, explain how that can happen. If not, explain why not.

Question 2 15 points

Consider the following red-black binary search tree. Dark edges indicate that the node at the bottom is red.



1. (5 points) Draw the 2-3-4 tree that this red-black tree encodes.

2. (10 points) Insert "eagle" into the original tree and re-draw the tree below. You should update the tree according to the rules of red-black tree insertion.

Question 3 20 points

You are to implement a method `getPath` that takes a list of vertices that form a path as a parameter and returns a list of the edges that comprise that path. Suppose that the graph contained vertices v1, v2, and v10. Then calling `getPath(vertices)`, where `vertices` is a list containing v1, v2, and v10 (in that order), should return a list containing an edge incident to v1 and v2 and an edge incident to v2 and v10. If some edge in the supposed path does not exist in the graph the method should throw an IllegalStateException.

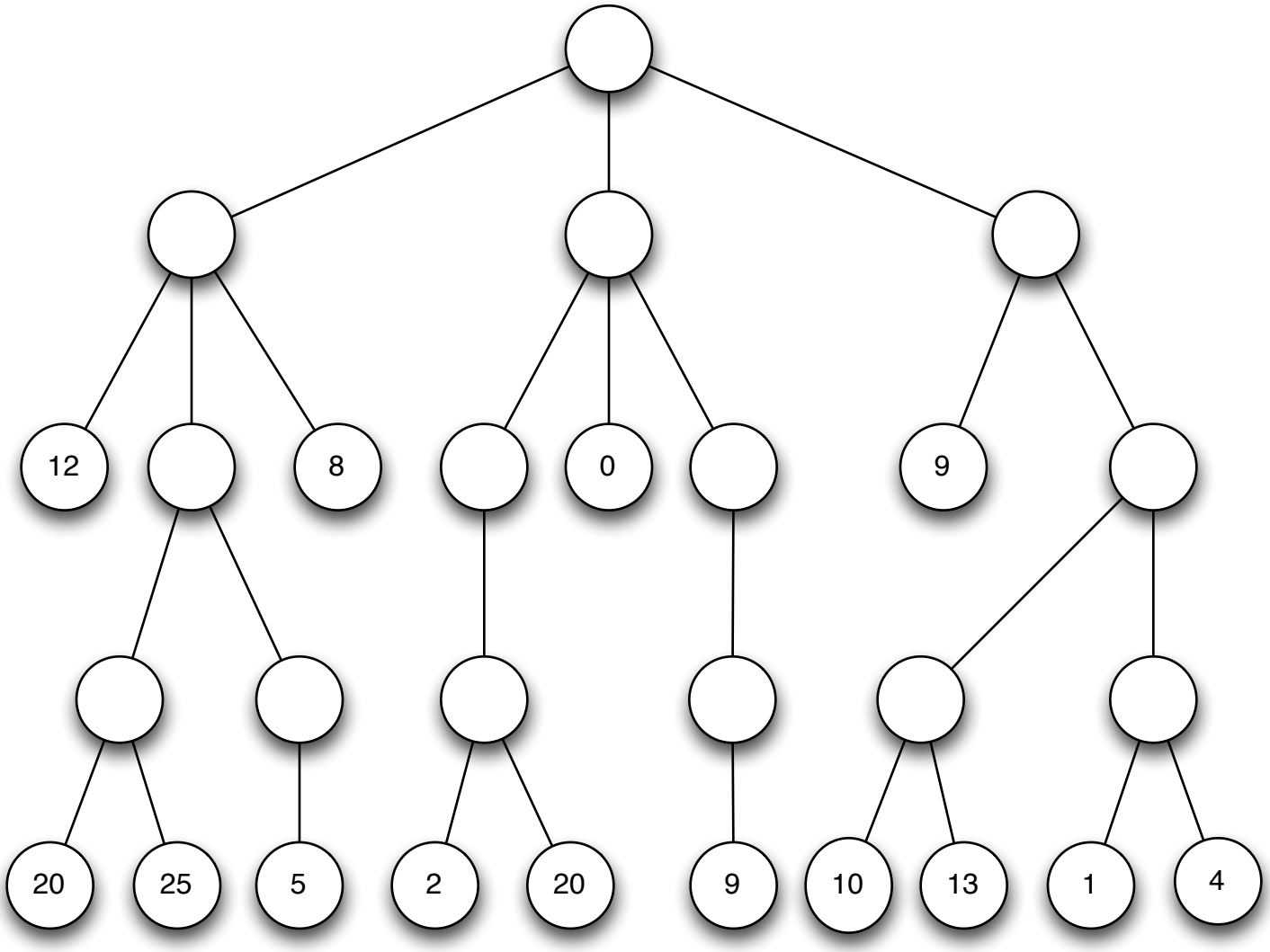
Complete the method started below, assuming that it is in a class that implements the `Graph` interface. The graph where the vertices and edges are stored is `this`.

```
/**
 * Finds the edges in a path given the vertices.
 * Precondition - verts is not an empty list.
 * @param verts a list containing the vertices in the path
 * @return the list of edges in the path
 * @throws IllegalStateException if not a valid path
 */
public List<Edge<E>> getPath(List<Vertex<V>> verts) {
```

Question 4 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 5 20 points

Write a backtracking method that generates all sequences of numbers taken from a set of positive integers that add up to a given target. Thus if the set of numbers is {1, 6, 3, 2, 9} and your target is 9, your method should print the following (where all of these sequences should appear, but the order in which they are printed does not matter):

```
[1, 2, 6]
[1, 6, 2]
[2, 1, 6]
[2, 6, 1]
[3, 6]
[6, 1, 2]
[6, 2, 1]
[6, 3]
[9]
```

The method that generated the output above is:

```
/**
  * Uses backtracking to find all sequences of numbers from a given set of
  * positive integers that add up to target.  No repetitions allowed.
  * @param numbers the set of numbers that sequences are chosen from
  * @param target the desired sum of the sequence
  */
public static void sequenceSum(Set<Integer> numbers, int target) {
  sequenceSum(numbers, target, new ArrayList<Integer>(), 0);
}
```

You are to complete the helper method begun on the next page. You should do appropriate pruning as you generate partial solutions.

```
/**
 * Helper method.
 * Uses backtracking to find all sequences of numbers from a given set of
 * positive integers that add up to target.  No repetitions allowed.
 * @param numbers the set of numbers that sequences are chosen from
 * @param target the desired sum of the sequence
 * @param sequence the current sequence of integers
 * @param sum the sum of the integers in sequence
 */
public static void sequenceSum(Set<Integer> numbers, int target,
            ArrayList<Integer> sequence, int sum) {
```