CS 62 Midterm 2 Practice

2017-4-7

Please put your name on the back of the last page of the test.

Problem 1: Multiple Choice [8 points]

[2 points each] For each question, circle the correct answer. If you're unsure of the correct answer, you may mark a second guess with an asterisk.

1. What is the ordering constraint on values in a binary search tree?

   A. Values to the left are always less than or equal to values to the right.
   B. Values deeper in the tree are always less than values near the root.
   C. Each left child must be smaller than its parent, and right children must be larger.
   D. All of the above.

2. Why doesn't most parallel code run any faster than sequential code in big-O terms?

   A. That's not true: most parallel code *does* run faster in big-O terms.
   B. With a finite number of processors, speedup is a constant factor, which gets dropped in big-O notation.
   C. The overhead of starting threads means that the code will actually be slower most of the time.
   D. In big-O analysis, smaller terms get dropped, so when speedup subtracts some time, like O(n - n) the quantity reduces to O(n).

3. What does the `synchronized` keyword do when attached to a method?

   A. It causes that method to run in a new thread each time it's invoked.
   B. It prevents threads from entering that method unless they have a corresopnding lock (the object the mehod is attached to).
   C. It prevents the operating system from swapping to a different thread while inside of the method.
   D. It marks that method as one which uses shared ("synchronized") data.

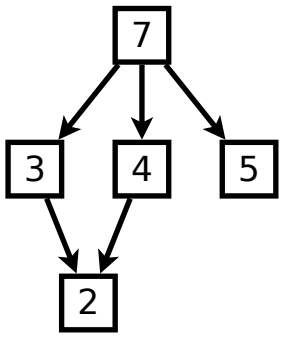
4. What is the `hashCode` method supposed to do?

   A. Look up an object in a hash table.
   B. Return a random number that's hard to guess.
   C. Return a semi-unique number based on an object's contents.
   D. Store source code in a hash table.

Problem 2: Short Answer [8 points]

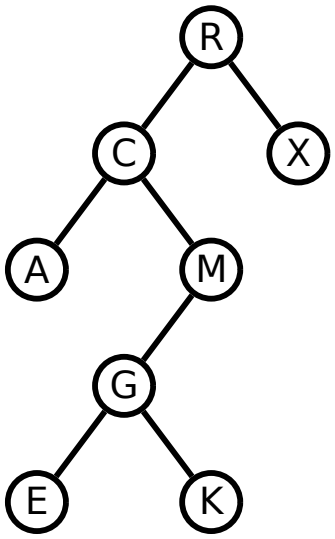
1. [4 points] **Splay Trees:** Explain why splay trees use three different operations to move items up to the root instead of just using a single "swap-with-parent" operation.

2. [4 points] **Parallelism:** Given the following program graph, calculate T_1 and T_∞. What is the theoretical maximum speedup for this graph when executed in parallel?



Problem 3: Binary Search Trees [10 points]

1. [2 points] Given the following binary search tree (with letters kept in alphabetical order), where would the letter 'P' be inserted (draw on the digram below)?



2. [4 points] When adding an item to a binary search tree which is a duplicate of an item already in the tree, where will that item wind up? Explain the role of the `predecessor` function in this process and what that function does. Indicate on the diagram above where a second 'M' would be inserted.

3. [4 points] What is the big-O run time of looking up a value in a binary search tree? Include best-case and worst-case times, and express each value twice: once in terms of h, the height of the tree, and again in terms of n, the total number of nodes in the tree.

Problem 4: Parallelism [12 points]

Consider the following code:

```
class ParallelMin extends RecursiveTask<Double> {
    protected double[] data;
    protected int lo, hi;
    private static final int CUTOFF = 1000;

    public ParallelMin(double[] data, int lo, int hi) {
        this.data = data;   this.lo = lo;   this.hi = hi;
    }
    public Double compute() {

        if (_____________________________________________) {
            double result = this.data[this.lo];
            for (int i = this.lo; i < this.hi; ++i) {
                if (this.data[i] < result) { result = this.data[i]; }
            }
            return result;
        } else {
            ParallelMin firstHalf = new ParallelMin(
                this.data,

                ____________________________,

                ____________________________
            );
            ParallelMin lastHalf = new ParallelMin(
                this.data,

                ____________________________,

                ____________________________
            );
            firstHalf.fork();
            result = firstHalf.join();
            if (lastHalf.compute() < result) { result = lastHalf.compute(); }
            return result;
        }
    }
}
```

(problem continues on next page)

1. [2 points] Fill in the five blanks above with lower and upper bounds so that the code divides the work in half with each recursive call until the `CUTOFF` value is reached.

2. [5 points] This code has two problems: first, it doesn't do any work in parallel, and second, it duplicates some work. Explain why these problems occur and what could be done to fix them.

3. [5 points] Suppose that each iteration of the `for` loop in the `ParallelMin` code above takes 1 microsecond. Ignoring the time taken by other code (including time to set up threads and combine results), about how long would it take to find the minimum of 1000 elements on a machine that can run 4 threads at once? What about finding the minimum of 8000 elements?

Problem 5: Concurrency [8 points]

The code below shows two classes: an outer class `Parent`, and an inner class `Child`, where the `Parent` object always contains a `Child` and the `Child` object knows what its parent is. Each can get its own name, or a description that includes its own name and the name of its parent/child. Note which methods are and are not `synchronized`.

```
class Parent {
    protected Child child;
    protected String name;

    public Parent(String name, String childName) {
        this.name = name;
        this.child = new Child(this, childName);
    }

    public synchronized String getName() { return this.name; }

    public synchronized String description() {
        return this.getName() + ", the parent of " + this.child.getName();
    }

    public String childDescription() {
        return this.child.description();
    }

    protected static class Child {
        protected Parent parent;
        protected String name;

        public Child(Parent parent, String name) {
            this.parent = parent;
            this.name = name;
        }

        public synchronized String getName() { return this.name; }

        public synchronized String description() {
            return this.getName() + ", the child of " + this.parent.getName();
        }
    }
}
```

(problem continues on next page)

1. [3 points] Assume that a single `Parent` object name 'A' with a single `Child` named 'B' has been constructed. When the `getName` method of the `Parent` object 'A' is called, what object serves as the lock for synchronization? Likewise, when the `getName` method of the `Child` object 'B' is called, what object serves as the lock?

2. [5 points] Your friend is using this `Parent` and `Child` code is a multi-threaded project where multiple threads use the `description` and `childDescription` methods of a shared `Parent` object 'A' at once. They notice that sometimes, their program seems to freeze, but they're confused because their program doesn't have any loops that could be infinite. Explain why their code freezes, and given an example of a sequence of events that causes it to freeze.

Name: _____