CMSC 132 –Final Exam Sample Problems

Spring 2015

Notes

This is not intended to be an exhaustive review of all of the problems that might appear on your final exam. Instead, it is a selection of problems that we have either done in class or on previous exams that require additional attention. For example: I didn't include any specific question on Dijsktra's algorithm; use the slides that have been done so well by others and that are available to you on Elms for that topic.

Please use these in the spirit in which they are offered. Try them first, in earnest. Answers may be provided later.

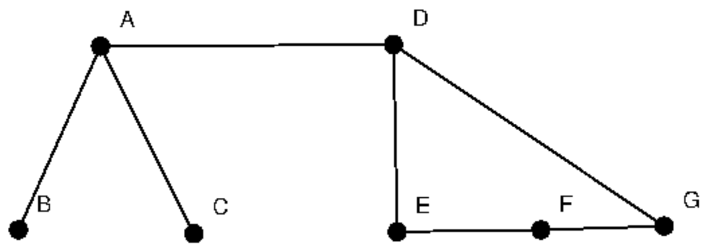
1. Using only the definitions provided by the References and standard Java statements, write the recursive function `removeLast` that takes any `LinkedList` and returns a copy with its last element removed. (This is a question from Midterm 1, re-tooled to use the kinds of definitions allowed on the exam.) Your implementation must be recursive and should be contained in one method.

```
public static <T> LinkedList<T> removeLast(  LinkedList< T>  list ) {
```

Solution:

```
public static <T> LinkedList<T> removeLast( LinkedList< T > list ) {
    if (isEmpty( list) ) return list;
    if (isEmpty( rest( list ) ) ) return rest(list);
    else
        return cons( first( list ), removeLast( rest( list ) ) );
```

2. Examine the undirected graph below:



and use it to answer the following questions:

(a) Give the list of vertices visited in an *ordered* Breath-First traversal of the graph, starting from vertex **B**:

(a) **B,A,C,D,E,G,F**

(b) Give the list of vertices visited in an *ordered* Depth-First traversal of the graph, starting from vertex **B**:

(b) **B,A,C,D,E,F,G**

(c) Express the graph used in the this question formally, i.e., as a set of Vertices and a set of Edges, . . . :

Solution: The $G = (V, E)$ be defined as the set of vertices,

$$V = \{A, B, C, D, E, F, G\},$$

and the set of edges,

$$\{e_1(A, B), e_2(A, C), e_3(A, D), e_4(D, E), e_5(D, G), e_6(E, F), e_7(F, G)\}.$$

Note: the by convention vertices and edges are listed in their natural ordering, but this is not a requirement.

3. Given any 2 BinaryTrees, tree_1 and tree_2, write the sameShape predicate that returns true if and only if tree_1 and tree_2 have the same shape, i.e., each may be superimposed on the other with no apparent differences othen that the labeling of their nodes.

```
public static <T> boolean sameShape( BinaryTree< T > tree_1,
                                     BinaryTree< T > tree_2 ) {
```

Solution:

```
public static <T> boolean sameShape( BinaryTree<T> tree_1,
                                     BinaryTree<T> tree_2 ) {
    if( isEmpty( tree_1 ) && isEmpty( tree_2 )  ) return true;
    else if ( ! (isEmpty( tree_1)) && ! (isEmpty( tree_2 ) ) )
       return sameShape( getLeft( tree_1 ), getLeft( tree_2 ) ) &&
              sameShape( getRight( tree_1 ), getRight( tree_2 ) );
    else return false;
}
```

4. Study the classroom example below:

```
class HelloWorldHello {
    String str1=''Hello'', str2=''World'';

    Thread T_1 = new Thread( new Thread() {
        public void run() {
            synchronized( str1 )
                synchonrized ( str2 )
                    System.out.println( str1 + str2 );
         }
     });

     Thread T_2 = new Thread( new Thread () {
         public void run() {
             synchronized( str2 )
                  synchronized( str1 )
                      System.out.println( str2 + str1 );
         }
     });
     public static void main( String[] args ) {
         HelloWorldHello hwh = new HellloWorldHello();
         hwh.T_1.start();
         hwh.T_2.start():
     }
}
```

Which statement best describes what happens when the `main` method executes?

A. The program immediately deadlocks.

B. The program runs normally, but the order of Strings printed is unpredictable.

C. The program sometimes prints `"HelloWorld"` and other times `"WorldHello"`, but sometimes deadlocks.

D. We cannot say because we don't have enough information in the example.

5. Rewrite the following method by removing the `synchronized` from its methods and replacing each with the equivalent use of `synchronized` in the body of each method.

```
class Counter {
   private int counter=0;

   synchronized public incr() { counter++ };
   synchronized public decr(){ counter--; };
   synchronized public int value() { return counter; }
```

Rewrite this class with the required modifications in the space below:

Solution:

```
class Counter {
    private int counter = 0;
    public incr() {
       synchronized( this )  {
          { counter++ }
        }
    }
    // and each method is likewise re-written ..
```

Instructions & Definitions for CMSC 132(H) Final

- No use of `try-catch` statements is allowed in any code written in response to any question on this exam;
- For recursive methods: Use *only* the structures provided in the **References** provided in this document. Do not use any external data-structures, such as `ArrayLists`, `Stacks`, etc. with the intent of removing recursion from any recursive method that you write.
- No question that requires a recursive implementation requires or will allow the explicit use of any iterative statement.
- No auxiliary or helper methods should be required to respond to any of the programming questions on this exam.
- Graph searching questions assume that the *natural* ordering of vertices will be used in the expansion of adjacencies. In the case of numbers, assume $0, 1, 2, \ldots$, and in the case of literals or Strings, assume the normal rules of English, $a, b, c, \ldots$.

Allowed definitions for data-structures used on this exam.

Linked Lists

Use the following definition for questions regarding linked-lists. Note: All of these methods are `static`, accepting `LinkedList<T>`s and elements of any object type, `T`.

```
public static <T> boolean isEmpty( LinkedList< T > lst );
public static <T> T first( LinkedList< T > lst );
public static <T> LinkedList< T > rest( LinkedList< T > lst );
public static <T> LinkedList< T > cons( T item, LinkedList< T > lst );
```

Method Signature	**Description**
`boolean isEmpty( lst )`	Returns `true` if `lst` is empty.
`T first( lst )`	Returns the *value* of the first element in `lst`. Note, it is an error to call this function on an empty list.
`LinkedList rest( lst )`	Returns the "tail,"a `LinkedList` of the elements after the first element in a linked list. Note, calling this function on an empty lists causes an error.
`LinkedList cons( ele, lst)`	Returns a *new* `LinkedList` that is the result of adding the `ele` as its first element; elements are kept in their original order.

Table 1: Linked List Operations: All linked list operators are *transparent*, returning copies.

Binary Trees

Use the following definition for questions regarding Binary Trees.

```
public class BinaryTree< T > {
    public boolean isEmpty();
    public T getValue();
    public int height();
    public BinaryTree< T > getLeft();
    public BinaryTree< T > getRight();
}
```

Method Signature	**Description**
`boolean isEmpty()`	Returns `true` if this `BinaryTree` is empty.
`T getValue()`	Returns the value on this `BinaryTree` object; note, an error results if this tree is empty.
`int height()`	Returns the length of the longest path from the root of this `BinaryTree` to its deepest (most remote) leaf. Note: the `height()` of an empty tree is 0.
`BinaryTree<T> getLeft()`	Returns the left child. Note: calling this on an empty tree throws an exception.
`BinaryTree<T> getRight()`	Returns the right child. Note: calling this on an empty tree throws an exception.

Table 2: Binary Tree Operations

Ordering Relations

Unless told otherwise, assume that any references to *Binary Search Trees* are ordered such that their left contains elements less than or equal to the root, and the right contains elements greater than the root. All heaps are Max-Heaps, unless told otherwise.

Provided methods, classes, etc.

Assume that all methods work as described when answering any questions regarding Linked-Lists or Binary Trees on this exam. Do not assume that any other methods exist unless you defined them, using these methods. Do **not add** any properties to these class definitions.

- No use of `try-catch` statements will be accepted for any programming-related question on this examination.
- No use of Java "convenience" types, such as `ArrayList`, `Stack`, etc., will be accepted for any question on this exam.