The University of Mississippi
Department of Computer and Information Science
Data Structures and Algorithms Comprehensive Exam
Spring 2005

Instructions: Answer any **FIVE** of the following nine questions. The questions are equally weighted. You have three hours to complete the exam.

1. We can specify an abstract data type (ADT) by giving three types of information about the ADT:

   - the sets (i.e., types or domains) involved
   - the signatures (syntax) of the operations (i.e., functions)
   - the semantics of the operations

   Consider a `Table` (or dictionary) abstract data type (ADT) that provides a data structure in which the entries consist of key-value pairs. In such a structure, the key of a pair uniquely identifies the pair within the data structure and can be used to access the value.

   Give a specification. Use constructive, design-by-contract techniques (i.e., preconditions, postconditions, and perhaps invariants) to specify the semantics. The `Table` ADT will have at least the following operations:

   - create a new table
   - insert a new key-value pair into the table
   - delete a key-value pair from the table
   - return the value paired with a key in the table
   - determine whether a key has a value in the table
   - determine whether the table is empty

   Suppose you wish to implement the `Table` using an array (or arrays) in memory.

   Describe the design of the internal state of the ADT's implementation. ***Give an implementation (representation) invariant*** that specifies the relationship between the ADT's abstract value and the values of its implementing variables.

2. Suppose that you work in the information technology department of large supermarket chain with stores at many locations around the country. The company has a loyalty program in which frequent customers get discounts and other special offers. To obtain these benefits, customers must submit their loyalty program membership cards whenever they make purchases. The membership card includes an id number that is unique across the chain.

   The company maintains information about all customers in a large ***sequential file*** (not in a database) at its central data processing facility. (Let's call this the *master* file.) Each local store collects information about its local customers and uploads that information once per day to the centralized data processing facility so that the the master file can be updated. (Let's call these *transaction* files.) The transaction files include information about addition of customers, deletion of customers, and change in information about customers as well as information about the customer's purchases for that day.

   Suppose you are given the task of designing the programs to do the updates of the master file.

   (a) How would you order the master file? How would you organize the transaction files?

   (b) Outline an efficient algorithm for updating the master file using the daily transaction files.

3. Discrete mathematics. Answer any **THREE** of the following four parts of this problem.

   (a) A binary relation is a subset of the Cartesian product of two sets, that is, it is a set of ordered pairs. We are often interested in whether relations have the properties of being reflexive, symmetric, and transitive. If a relation has all three properties, then it is said to be an equivalence relation. Identify each of the following as being a reflexive, symmetric, transitive, or equivalence relation.

      i. $\{(a, b) : a/b \leq b/a\}$ where a and b are nonzero real numbers.

      ii. $\{(a, b) : |a| = |b|\}$ where a and b are real numbers.

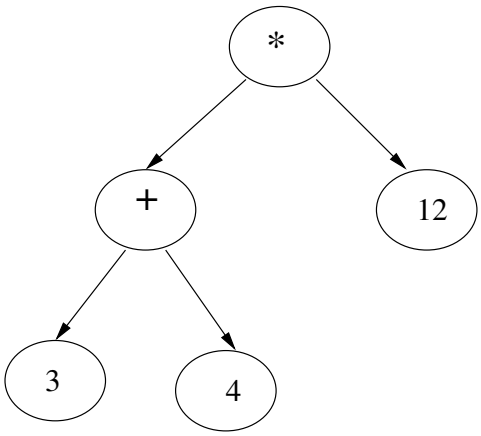
      iii. $\{(a, b) : a^2 + b^2 = 1\}$ where a and b are real numbers.

   (b) Prove or disprove that $2^n + 1$ is prime for all non-negative integers n.

   (c) A computer network consists of six computers. Each computer is directly connected to at least one of the other computers. Show that there are at least two computers in the network that are directly connected to the same number of other computers.

   (d) Use mathematical induction to prove that $1 + 2 + 2^2 + 2^3 + ...2^n = 2^{n+1} - 1$

4. Consider a simplified expression tree (an example is shown below). The tree is a binary tree where each internal node contains a character representing a mathematical operator (+, -, *, /). The leaves contain integers.



   (a) Using the syntax of your choice, show a possible declaration for the structure of the tree.

   (b) Explain the tradeoffs you considered, and why you chose the structure you did.

   (c) Write a function (method) to accept the root of an expression tree and return its value. Note that in the example above, the expression represented is (3+4)*12, which evaluates to 84.

5. This question deals with the problem of finding the largest item in a list of n items.

   (a) Write an *iterative* algorithm to solve the problem. Analyze your algorithm and show the results in order notation.

   (b) Write a *recursive* algorithm to solve the problem. Analyze your algorithm and show the results in order notation.

   (c) What is the lower bound for this problem? Explain your answer fully.

6. Choosing data structures.

   (a) A sparse array is one in which most entries are zero. Assume that we want to store a sparse n x n x n array of real numbers such that the operations `insert` and `delete` are efficient–these operations insert or delete an element in a given position (i, j, k) in the array. Deleting an element thus means making it zero. Explain which space-efficient data structure you would use if you knew that approximately all but n of the n^3 possible elements are zero at any time. We would like to get on the average about O(1) execution times for `insert` and `delete`, just as if we would with an ordinary array.

(b) For each of the questions, choose the best of the listed data structures and explain why your choice is best. Where several operations are listed, you should assume that the operations occur with about equal frequency.

i. Operations are: `Insert`, `DeleteMax`, and `DeleteMin`.

Choose data structure: balanced tree or sorted doubly-linked list

ii. Operations are: `Insert` and `FindMin`

Choose data structure: sorted linked list or sorted array

iii. You have a dictionary that contains at most 10 words.

Choose data structure: balanced tree or unordered array

iv. You have a large set of integers with operations `Insert`, `FindMax`, `DeleteMax`.

Choose data structure: unordered array or hasing with linear probing

7. Polynomial Reduction.

(a) Define polynomial reduction.

(b) What are two common uses for polynomial reductions. Give examples.

8. Minimum Spanning Tree.

(a) Define minimum spanning tree.

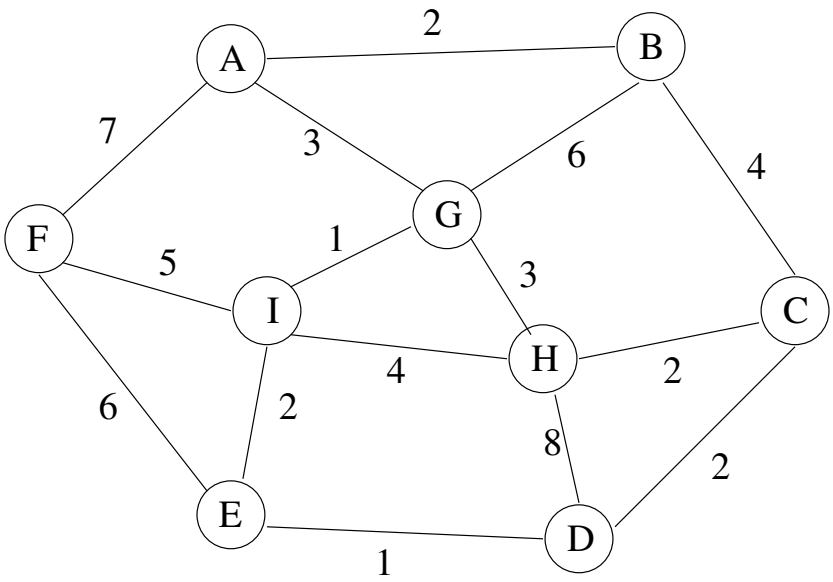
(b) Explain how Prim's MST algorithm works.

(c) Apply Prim's algorithm to the graph below. Show clearly the data structure at each stage.

(d) Explain how Krushal's MST algorithm works.

(e) Apply Kruskal's algorithm to the graph below. Show clearly the data structure at each stage.

(f) What happens if the algorithms receive an unconnected graph?



9. Java data structure programming. This problem refers to the attached Java program `ListTest`. The classes `List`, `Nil`, and `Cons` represent a simple recursive list structure (similar to that found in the language Lisp).

   Six functions are defined on this data structure. Note that some have recursive implementations. Most are straightforward. Function `map` returns a new list that is like the original list except that a single-argument function has been applied to every element.

   (a) What are the inheritance relationships among the classes `List`, `Nil`, and `Cons`.

   (b) Which of the classes `List`, `Nil`, and `Cons` can be instantiated?

   (c) The program uses polymorphism. Describe how the appropriate implementation of the `map` method is located at at runtime and executed.

   (d) Class `ListTransducer` implements the builtin Java `Iterator` interface. (That is, its design reflects the Iterator design pattern.) What are the purposes and characteristics of an iterator on a data structure?

   (e) Give the implementations of a new method `length()` in `List`, `Nil`, and `Cons`. This method should count the elements in the list and return the count as an `int`.

   (f) Show the output produced by the lines marked `/* 1 */` through `/* 8 */` in the attached program.

ATTACHMENT – Return with Examination

```
import java.util.*;

public class ListTest
{   public static void main(String[] args)
    {   List nilList = new Nil();
/* 1 */ System.out.println("nilList: " + nilList);
        List twoList = new Cons("one", new Cons("two", nilList));
/* 2 */ System.out.println("twoList: " + twoList);
/* 3 */ System.out.println("contains \"one\"? " + twoList.contains("one"));
/* 4 */ System.out.println("contains \"two\"? " + twoList.contains("two"));
/* 5 */ System.out.println("contains \"six\"? " + twoList.contains("six"));
        ListTransducer tr = new ListTransducer
            (twoList,new NotEqualSelector("two"),new MapToUpper());
        for(int i = 1; tr.hasNext(); i++)
/* 6 */     System.out.println("transducer: " + i + " " + tr.next());
        List oneList = twoList.remove("two");
/* 7 */ System.out.println("remove \"two\": " + oneList);
        Converter m = new MapToUpper();
/* 8 */ System.out.println("map to uppercase: " + twoList.map(m));
    }
}

abstract class List
{   public static final List NIL = new Nil();
    abstract public List    remove(Object e);   // remove all occurrences
    abstract public boolean isEmpty();          // has no elements?
    abstract public boolean contains(Object e); // is argument in list?
    public List  map(Converter m) { return this; }  // return converted list
    abstract public Object  getHead();  // head element
    abstract public List    getTail();  // tail list
}

class Nil extends List  //  an empty list
{   public List    remove(Object e) { return this; }  //  return self
    public boolean isEmpty() { return true; }
    public boolean contains(Object e) { return false; }
    public Object  getHead() { return null; }   // head element
    public List    getTail() { return this; }   // tail list
    public String  toString() { return ".NIL."; }
}
```

```
class Cons extends List // nonempty list
{   public Cons(Object h, List t) { head = h; tail = t; }

    public List remove(Object e)
    {   if (head.equals(e))  // by default, use built-in equality test
        {   return tail.remove(e); }
        else
        {   return new Cons(head,tail.remove(e)); }
    }

    public List map (Converter m)
    {   return new Cons(m.convert(head),tail.map(m)); }

    public boolean isEmpty() { return false; }
    public boolean contains(Object e)
    {   return ( head.equals(e) || tail.contains(e) ); }

    public Object getHead() { return head; }
    public List   getTail() { return tail; }
    public String toString() { return (head + " " + tail); }

    private Object head;    // first element
    private List   tail;    // rest of list
}

interface Converter
{   abstract public Object convert(Object e);
}

class NullConverter implements Converter
{   public Object convert(Object e) { return e; }
}

class MapToUpper implements Converter
{   public Object convert(Object o)
    {   if (o instanceof String)
        {   String s = (String) o;
            return s.toUpperCase();
        }
        else
        {  return o; }
    }
}
```

```
interface Selector
{   abstract public boolean selects(Object e);
}

class NullSelector implements Selector
{   public boolean selects(Object e) { return true; }
}

class NotEqualSelector implements Selector
{   public NotEqualSelector(Object c) { check = c; }
    public boolean selects(Object e) { return !check.equals(e); }
    private Object check;
}

class ListTransducer implements Iterator
{   public ListTransducer(List l, Selector s, Converter c)
    {   base = l; sel = s; conv = c; }

    public void remove() { }
    public boolean hasNext()
    {   advance();
        return ( !base.isEmpty() );
    }

    public Object next()
    {  advance();
       if (!base.isEmpty())
       {   Object nxt = conv.convert(base.getHead());
           base = base.getTail();
           return nxt;
        }
        else
        {   return List.NIL; }
    }

    private void advance() // advance base to first element selected
    {   while ( !base.isEmpty() && !sel.selects(base.getHead()) )
        {   base = base.getTail(); }
    }
    private List base;
    private Converter conv;
    private Selector sel;
}
```