Computer Science 136

Spring 2004
Professor Bruce

Final Examination

May 19, 2004

Question	Points	Score
1	10	____
2	8	____
3	15	____
4	12	____
5	12	____
6	8	____
7	10	____
TOTAL	65	____

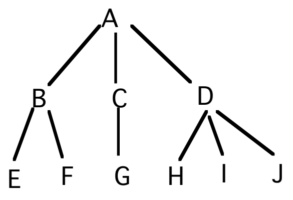
Your name (Please print) ____

I have neither given nor received aid on this examination.

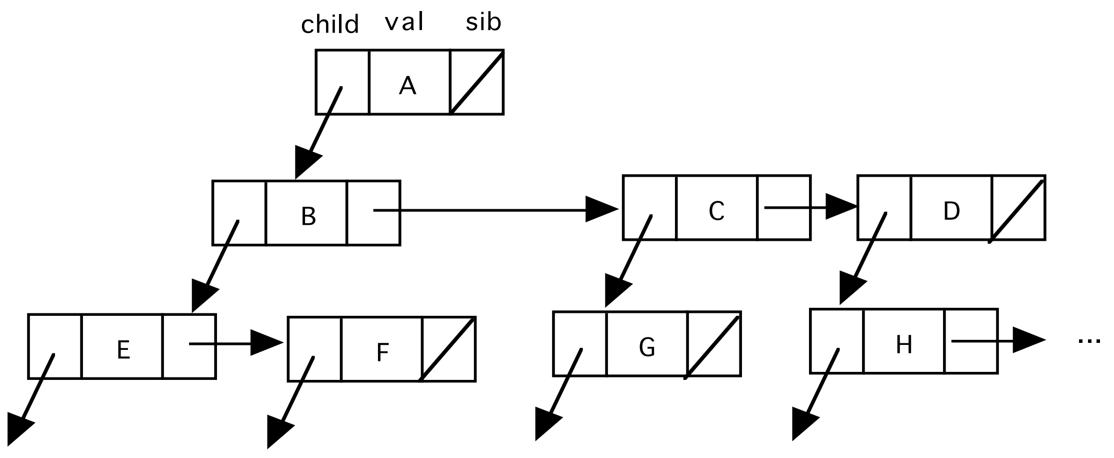
____

(signature)

1. In class we only discussed binary trees, that is, trees in which each node has at most two children. In general trees, nodes may have any number of children.



Since we do not know in advance how many children a node will have, we do not know how many child links to include in a node, we instead provide each node with a link to one of its children (you can think of it as an "eldest child" link) and then a link to its next eldest sibling (brother or sister). (We won't bother with a pointer to the parent.) The following structure represents the tree above.



Notice that this representation differs from that in the HexAPawn program in that the game tree there represented all of the children by a Vector, whereas here we use only GenTreeNode objects and have explicit sib field in each node.

a. A pre-order traversal is defined the same way as for binary trees: visit the root then do pre-order traversals of all the subtrees headed by the roots children, processing the child subtrees from left to right. Below we provide a definition of a general tree node. Please add the body of the preOrder method. You may presume there is a parameterless method visit() which can be sent to an object of type ValType (the type of the value field of the nodes). *I highly recommend using recursion.*

```
public class GenTreeNode{
   protected GenTreeNode child, sib;
   protected ValType val;
   public GenTreeNode(ValType theVal){
      val = theVal; child = null; sib = null;
   }
   public GenTreeNode getEldestChild(){ return child;}
```

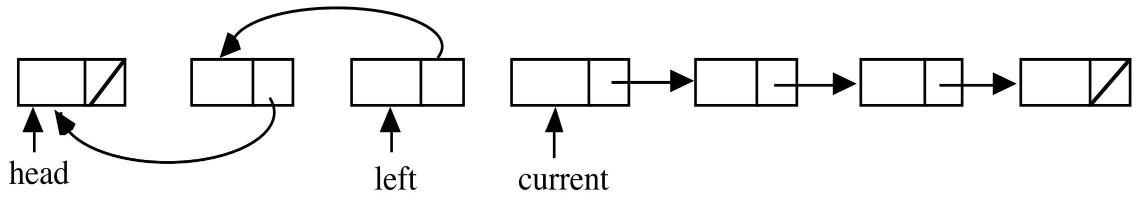
```
      public GenTreeNode getNextSibling(){ return sib;}
      public ValType getValue(){ return val;}
      ... // other methods omitted
```

```
   // Post: Performed pre-order traversal on tree headed by
   // this node, sending visit() to all nodes encountered.
   // Each node is visited exactly once in the traversal
   public void preOrder(){
```

The main differences between a general tree and a graph is that a tree has a special node designated to be a root and may not include any cycles among its edges. That is, there are no paths from a node back to itself that do not retrace any edges.

b. We remarked in class that a depth-first search of a graph is similar to a pre-order traversal of a tree. What extra complications must you be concerned about when doing a depth-first search of a graph that you didn't have to worry about with the general tree?

2. With a more thoughtful implementation, it is possible to traverse a singly-linked list both forward and backward from the current node. This is done by reversing links each time the current node is moved to the right and 'fixing' the link each time the current node is moved to the left. For example, if the current node were the fourth node in the list, the list would look like this:



"Current" points to the current node and "left" to the node on its left. Note that all of the nodes to the left of "current" have their links reversed, that is, they point to their predecessor instead of their successor. Write a method moveRight() which moves the current node one node to the right from an arbitrary position of this form, adjusting all necessary pointers. Note that if "current" points to the first node of the list, "left" will have value null. When the move is completed, the list should again appear as above (i.e. pointers to the right of current pointing to the right, those to the left, pointing to the left).

You may use `SinglyLinkedListElement` from the structures library:

```
class SinglyLinkedListElement {
    protected Object data; // value stored in this element
    protected SinglyLinkedListElement nextElement;
      ...
    public SinglyLinkedListElement next(){...}
    // post: returns reference to next value in list

    public void setNext(SinglyLinkedListElement next){...}
    // post: sets reference to new next value

    public Object value(){...}
    // post: returns value associated with this element

    public void setValue(Object value){...}
    // post: sets value associated with this element

}


public class TraversibleList{
   SinglyLinkedListElement head, left, current;
   public TraversibleList(){ head = left = current = null;}
```

```
/* pre:  List non-empty, current refers to node of list.

   post:  The "current" node is moved one to the right.  "left"
points to node to left of current. If already at last node, set
current to NIL and "left" to point to last node.  */

public void moveRight(){       // fill in details
```

3. (Hash tables)

a. If you are using a hash function based on the division function then the best choice for table size is: (a) a power of two b) a prime number (c) a perfect square.

b. With a hash function based on the "mid-square" method, the best choice for table size is: (a) a power of two (b) a prime number (c) a perfect square.

c. One of the best methods for converting a string to an integer in order to apply a hash function is to add up the integer equivalents of each of the letters in the key:
True False

d. Secondary clustering refers to a clash occurring in the second or subsequent locations in which we try to place an element in a hash table: True False

e. With a good hashing function and low load factor, the complexity of inserting an element in a hash table with n elements should be
(a) O(1) (b) O(log n) (c) O(n)

f. With a bad hashing function and linear probing for resolving hash clashes, the complexity of inserting an element in a hash table with n elements could be as bad as:
(a) O(1) (b) O(log n) (c) O(n) (d) O(n^2) (e) O(2^n)

g. For each of the following techniques for handling hash collisions, determine whether that technique has problems with each of primary and secondary clustering. Wrote "OK" in the appropriate box if clustering is not a problem with that technique, "Bad" if clustering is a major problem and "Mod" if it is a moderate problem.

Technique	**Primary**	**Secondary**
linear probing		
double hashing		
external chaining		

h. The HashTable classes in java.util and in the structures library have a provision that if the load factor in the table becomes too high, a new table with twice the capacity is created (similar to the way Vectors behave). Would it be a good idea to simply copy over elements from the old table into corresponding positions of the new table? Why or why not?

4. Short answers

a. What is the main advantage of using a splay tree over a general binary search tree?

b. What is the advantage of using the array representation of trees over the usual linked representation when performing a heapsort?

c. Explain briefly why we have usually preferred to use interfaces over classes for the types of variables.

d. In writing programs with threads, we normally execute sleep(PAUSE_TIME) method to get the object to pause for PAUSE_TIME milliseconds before continuing. Alternatively, we could write a while loop as follows to pause:

```
int startTime = System.currentTimeMillis();
while (System.currentTimeMillis() < startTime + PAUSE_TIME) {}
```

The program would continue executing the while loop until PAUSE_TIME milliseconds have passed. What is the most important reason to prefer using the sleep command over this while loop? (Hint: the reason has nothing to do with accuracy of the delay time.)

e. Does it ever make sense to have a class with exactly one method labeled `synchronized`? Why or why not?

f. We designed classes for expression trees with and without the Visitor pattern. In the original version (without visitors), each type of expression class contained method to evaluate the tree and print it out. In the version using the Visitor pattern, each class simply contained a process method, while there were separate visitor classes for evaluating and printing out the tree. Please discuss briefly the advantages and disadvantages of each of these two designs, especially with regard to extensibility (e.g., adding new kinds of expressions or operations).

5. This question has to do with algorithms used in the adjacency list and adjacency matrix implementations of undirected graphs. Recall that in the adjacency matrix representation of graphs, the implementation keeps track of a *free list* of rows and columns of the matrix not currently in use.

For each of the following give a very short description of the algorithm as well as its time complexity. In computing the complexity you may assume that the graph has n vertices and each vertex has at most d edges connected to it.

To give you an example of the kind of answer desired, here is a sample.
Sample Question: Add a vertex to a graph in the adjacency list implementation.
Answer: *Insert an association consisting of the vertex label and an empty list into the dictionary (hash table) instance variable holding vertices. O(1)*

a. Given labels lab1 and lab2 of vertices in the graph, add a new edge from lab1 to lab2 in the adjacency matrix implementation. *(If the edge is already there, replace it by the new one, but do not allow two edges with the same vertices.)*

b. Given the labels lab1 and lab2 of nodes in the graph, add a new edge from lab1 to lab2 in the adjacency list implementation. *(If the edge is already there, replace it by the new one, but do not allow two edges with the same vertices.)*

c. Delete a vertex in the adjacency matrix representation *(don't forget to get rid of all edges).*

d. Delete a vertex in the adjacency list representation *(don't forget to get rid of all edges).*

6. Suppose you are given n lists, each of which is of size n and each of which is sorted in increasing order. Thus there are a total of n^2 elements in the lists. Consider the following two methods for merging the lists into one sorted list L:

I) At each step, examine the smallest element from each list; take the smallest of all of those elements, remove it from its list and add it at the end of list L. Repeat until all lists are empty.

II) Merge the lists in pairs, obtaining n/2 lists of size 2n. Repeat, obtaining n/4 lists of size 4n. Repeat as necessary until 1 list remains.

Each of these methods takes more than $O(n^2)$ time (which isn't surprising given that there are a total of n^2 elements to be merged). Determine the time complexity of each algorithm and explain your results. (Recall that a list of size k and a list of size m may be merged in O(k+m) time.)

7. An organization that possesses a small computer wants to maintain records on its members. Further, it wants to have very fast access when searching on each of two different keys: by name and by social security number. Both of these attributes have natural orderings. We wish to be able to search for elements via each kind of key. Assume that a search by name is most common and must be done as quickly as possible (O(1) is ideal) whereas the search by social security number is less common and so may be somewhat slower than the other (O(log n) is desirable). Insertions and deletions are common. Requests for lists of members ordered by social security number are common (the IRS is auditing your membership), while lists ordered by name are relatively infrequent. Unfortunately, the computer does not have in memory sufficient space to hold two copies of all of the records if, for example, the organization tried to maintain two separate arrays of records, each one sorted on a different key. (The problem here is that each record is itself quite large, requiring over 100 times as much memory as the key alone would.)

   Suggest how the records might be organized with no redundant copies of information (with the possible exception of the keys). Suggest a substantially different method for each of the two keys and discuss the time (in "big O" notation) to search for an element in each key. Also indicate the amount of time necessary to make insertions and deletions, as well as the time necessary to list the elements by social security number. Your grade on this question will depend on how fast the operations will be performed, and how little space is needed. You may draw pictures if necessary to illustrate your answer.