Points off	1	2	3	4	5		Total off	Net Score

CS 307 – Final – Spring 2010

Name_____

UTEID login name _____

Instructions:

1. Please turn off your cell phones.
2. There are 5 questions on this test.
3. You have 3 hours to complete the test.
4. You may not use a calculator on the test.
5. You may add helper methods if you wish when answering coding questions.
6. When answering coding questions assume the preconditions of the methods are met.

1. (1.5 point each, 30 points total) Short answer. Place you answers on the attached answer sheet.

For questions that ask what is the output:

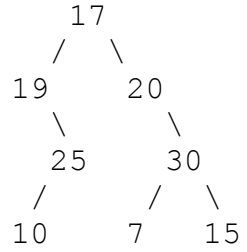
- If the code contains a syntax error or other compile error, answer "Compiler error".
- If the code would result in a runtime error or exception answer "Runtime error".
- If the code results in an infinite loop answer "Infinite loop".

On questions that ask for the Big O of a method or algorithm, recall that when asked for Big O your answer should be the most restrictive, correct Big O function. For example Selection Sort has an expected case Big O of $O(N^2)$, but per the formal definition of Big O it is correct to say Selection Sort also has a Big O of $O(N^3)$. Give the most restrictive, correct function. (Closest without going under.)

A. The following values are inserted one at a time in the order shown into an initially empty binary search tree using the traditional, naïve insertion algorithm. Draw the resulting tree.

```
37  42  13  101  -12
```

Consider the following binary tree. `17` is the root of the tree.



B. What is the result of a pre-order traversal of the binary tree shown above?

C. What is the result of a in-order traversal of the binary tree shown above?

D. What is the result of a post-order traversal of the binary tree above?

E. Is the tree shown above a binary search tree?

F. On the answer sheet, fill in the nodes of the given binary tree with integer values between 1 and 10 to make the tree shown a binary search tree.

G. What is the result of the following postfix expression? For subtraction the operand closest to the subtraction symbol is the infix equivalent of the right hand side operand.

```
10 5 11 3 - + *
```

H. What is output by the following code?

```
String data = "BINARY";
Stack<Character> st = new Stack<Character>();
for(int i = 0; i < data.length(); i++)
     st.push( data.charAt(i) );

st.pop();
st.pop();

for(int i = 0; i < 3; i++)
     System.out.print( st.pop() );
```

I. What is the Big O of method `total`? N = `list.size()`

```
public int total(LinkedList<Integer> list) {
     int total = 0;
     for(int i = 0; i < list.size(); i++)
          total += list.get(i);
     return total;
}
```

J. What is the worst case Big O of `printRev`?
Assume the `println` and `length` methods are O(1). N = `list.size()`

```
public void printRev(ArrayList<String> list) {
     for(int i = list.size() - 1; i >= 0; i--)
          if(list.get(i).length() < 5)
               System.out.println(list.get(i));
}
```

K. What is the Big O of method `empty` if `list` is a Java `LinkedList`?
N = `list.size()`

```
public int empty(List<String> list) {
     int total = 0;
     for(int i = 0; i < list.size(); i++) {
          total += list.get(0).length();
          list.remove(0);
     }
     return total;
}
```

L. What is the Big O of method `empty` if `list` is a Java `ArrayList`?
N = `list.size()`

M. What is the Big O of method `buildTree`? All values in `list` are distinct. (no duplicates) N = `list.length`. The `Arrays sort` method for `int`s uses the quicksort algorithm. The Java `Treeset` class uses a red-black tree as its internal storage container.

```
public TreeSet<Integer> buildTree(int[] list){
    TreeSet<Integer> result = new TreeSet<Integer>();
    Collections.sort(list);
    for(int i = 0; i < list.length; i++)
        result.add(list[i]);
    return result;
}
```

N. What is the Big O of method `buildTree2`? The `BST` class uses the traditional, naïve insertion algorithm.

```
public BST<Integer> buildTree2(int N){
    BST<Integer> result = new BST<Integer>();
    for(int i = 0; i < N; i++)
        result.add(i);
    return result;
}
```

O. What is the Big O of `methodO`? All values in `list` are distinct. (no duplicates) N = `list.length`. The `Collections shuffle` method is O(N).

```
public boolean methodO(int[] list) {
     ArrayList<Integer> hold = new ArrayList<Integer>();
     for(int i = 0; i < list.length; i++)
          hold.add(list[i]);

     Collections.shuffle(hold);

     int checkSum = 0;
     for(int i = 0; i < list.length; i++)
          if( hold.contains(list[i]) )
               checkSum++;

     return checkSum == list.length;
}
```

P. 1,000,000 distinct (no repeats) integers that are in random order are inserted one at a time into a binary search tree using the traditional, naïve insertion algorithm presented in class. What is the expected height of the resulting tree? Give the actual number as you did on the experiments in assignment 11, NOT the Big O of the height.

Q. What is output by the following code? (The Queue class is not the Java Queue interface, rather it is a class that implements a traditional Queue.)

```
Queue<Integer> q = new Queue<Integer>();
for(int i = 5; i < 12; i++)
     if( i % 2 == 1 )
          q.enqueue(i);

q.dequeue();
q.dequeue();

while(!q.isEmpty())
     System.out.print( q.dequeue() + " " );
```

R. Suppose you want to encode 60 countries. What is the minimum number of bits per country required?

S. If you want to store elements in a set and are only concerned with minimizing the time of all operations should you use a `TreeSet` or a `HashSet`? Briefly explain your answer.

T. Consider the following Node class:

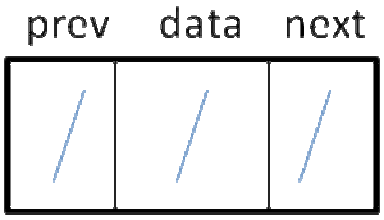
```
public class Node{
    public Object data;
    public Node prev;
    public Node next;

    public Node(Node p, Object o, Node n) {
        prev = p; data = o; next = n;
    }

    public Node(){
        data = null; prev = null; next = null;
    }
}
```

Draw the resulting variables and objects after the following code executes. Draw node objects as shown below. (The example has all three instance variables set to `null`.) Use arrows to show the references that exist and a forward slash to indicate variables that store `null`.

```
Node n1 = new Node();
Node n2 = new Node(n1, n1, null);
n1.next = n2;
n2.prev.next.next = n2;
n1 = n2;
```

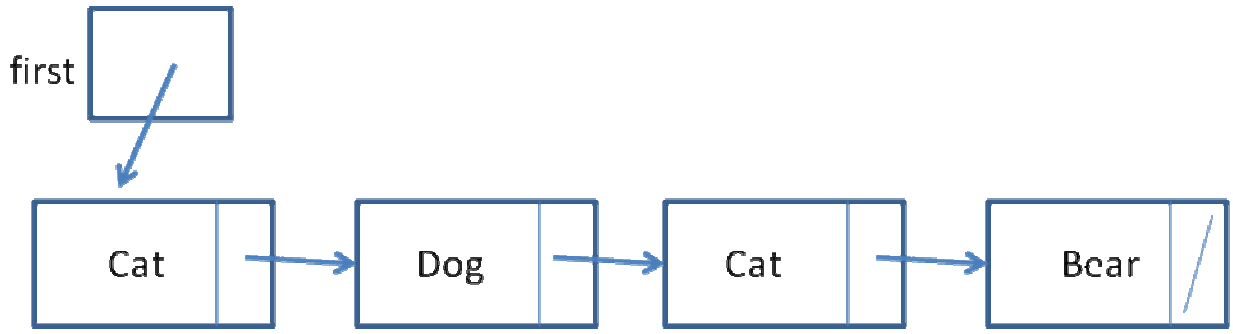


2. (Linked Lists, 15 points) Complete an instance method to remove the first occurrence of an element from a linked list.

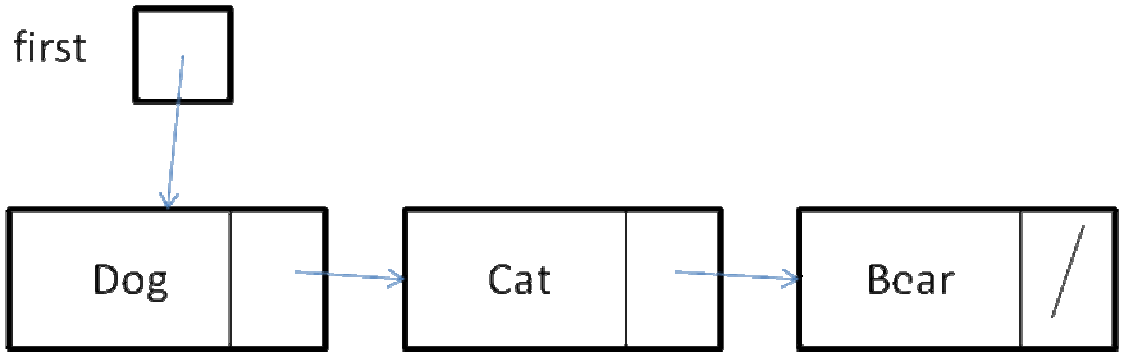
The linked list has the following properties:

- The nodes are singly linked.
- The `LinkedList` class keeps a reference to the first node in the linked structure, named `first`.
- The last node in the linked structure has its `next` reference set to `null`.
- If the linked list is empty the reference to the first node is set to `null`.
- The `LinkedList` class does not have an instance variable for its size.
- The Linked class achieves genericty by using Java's generics.

Consider the following linked list.



After a call to `removeFirst("Cat")` the linked list structure would now be:



Complete the `removeFirst(E obj)` method for the `LinkedList` class.

Imporant: Your method must be O(1) space meaning you can not use an auxillaury native array, `ArrayList`, or other data structure. Your method should be O(N) time where N is the number of elements initially in the `LinkedList`. You may not use any other methods from the `LinkedList` class unless you write them yourself on this question.

Here are some more examples of results of calls to `removeFirst(E obj)` as seen by a client of the `LinkedList` class. The return value is shown as well as the list after the call is completed.

```
[A, B, A, C, B].removeFirst(B) -> returns true, [A, A, C, B]
[A, B, A, C, B].removeFirst(A) -> returns true, [B, A, C, B]
[A, B, A, C, B].removeFirst(C) -> returns true, [A, B, A, B]
[A, B, A, C, B].removeFirst(D) -> returns false, [A, B, A, C, B]
[].removeFirst(A) -> returns false, []
[A].removeFirst(A) -> returns true, []
```

Here is the `Node` class the `LinkedList` class uses:

```
public class Node<E>{
    public Node(E value, Node<E> next);

    public Node<E> getNext();
    public E getValue();

    public void setNext(Node<E> newNext);
    public void setValue(E obj);
}
```

Here is the `LinkedList` class.

```
public class LinkedList<E>{

    // Reference to the front node of the linked list.
    // If this LinkedList is empty first == null
    private Node<E> first;

    // complete the following instance method:
    // pre: obj != null
    // post: Removes the first occurrence of obj in this list.
    //     Returns true if this list changed as a result of this
    //     method call.
    public boolean removeFirst(E obj) {
```

Complete this method on the next page

Complete this method on the next page

Complete this method on the next page

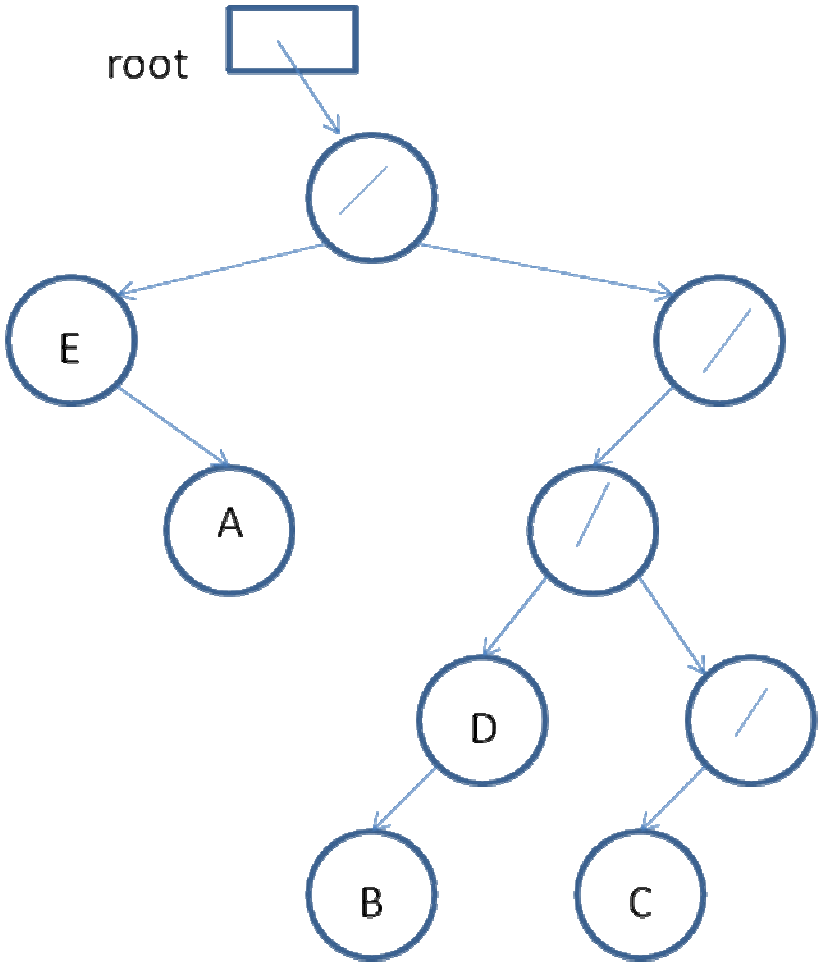
```
// pre: obj != null
// post: Removes the first occurrence of obj in this list.
//     Returns true if this list changed as a result of this
//     method call.
public boolean removeFirst(E obj) {
```

3. (Binary Trees, Using Data Structures 20 points) Morse code is a system for representing (encoding) characters, usually the english letters. The system typically consists of dots, dashes, and pauses between letters. When transmitting via radio the dots are shorter than the dashes, but in this question we will use period characters (.) to represent dots, hyphen characters (-) to represent dashes, and asterisks to represent the delimiters between letters.

The first five letters of the alphabet are encoded as follows:

```
A: .-
B: -...
C: -.-.
D: -..
E: .
```

In this question you will store the Morse codes for characters in a binary tree. The left child will represent a dot and the right child will represent a dash. Thus the resulting tree for the first five characters of the alphabet is as follows. Recall a dot is represented by a left reference and a dashe is represented by a right reference. Nodes that do not contain a character have their data value set to null.



Complete the constructor for the `MorseCodeTree` class. The constructor has a single parameter, a `Map<Character, String>`. The keys of the map are the characters in the Morse code system (A, B, C, D, E, etc.) and the values are Strings that contain the Morse code value for the character key. `".-"` for `A`, `"-..."` for `B` and so forth. Strings in the map will only contain period and hyphen characters.

The methods from the Map class you may use are:

`Set<K> keySet()`
Returns a `Set` view of the keys contained in this map.

`V get(Object key)`
Returns the value to which the specified key is mapped, or `null` if this map contains no mapping for the key.

The only method from the Set class you may use is:

`Iterator<E> iterator()`
Returns an iterator over the elements in this set.

The methods from the Iterator interface you may use are:

`boolean hasNext()`
Returns `true` if the iteration has more elements.
`E next()`
Returns the next element in the iteration.
`void remove()`
Removes from the underlying collection the last element returned by the iterator

The node class the `MorseCodeTree` class uses is:

```
public class BNode<E> {

     // create a new node with value, left, and right references
     // set to null.
     public BNode()

     // create a new node with the given values
     public BNode(E value, BNode<E> left, BNode<E> right)

     public E getValue()
     public BNode<E> getLeft()
     public BNode<E> getRight()

     public void setValue(E newValue)
     public void setLeft(BNode<E> newLeft)
     public void setRight(BNode<E> newRight)
}
```

Complete the following method. You may not use any other classes or methods except those from the `Map`, `Set`, `Iterator`, and `BNode` classes shown in the question and the `String` methods `charAt` and `length`. You are encouraged to break the problem up into different methods to improve readability.

```
public class MorseCodeTree {

     private BNode<Character> root;

/* pre: codes != null, all values in codes consist of periods
          and hyphens only.
   post: The MorseCodeTree has been created. root refers to
          the root node of the binary tree.
*/
public MorseCodeTree(Map<Character, String> codes) {
     // assume preconditions are met
```

```
// more room on next page if needed
```

```
// more room for MorseCodeTree constructor and helper methods
```

4. (Binary Trees 15 points) This question involves the same `MorseCodeTree` class as question 3. For this question assume the `MorseCodeTree` has been created correctly, regardless of what you wrote on question 3.

Write an instance method for the `MorseCodeTree` class that has a single parameter, a String. The String sent as a parameter contains only periods, hyphens, and asterisks. Periods represent dots, hyphens represent dashes, and asterisks represent delimiters between letters.

Complete the following instance method for the `MorseCodeTree` class:

```
/*   pre: encodedMessage != null, encodedMessage.length() > 0,
     encodedMessage consists of periods, hyphens, and asterisks
     only. The last character in code will be an asterisk.
     post: return what encodedMessage decodes to if it is
     correctly formed or null if encodedMessage is badly formed.
     (Some part of the code leads to a node that does not exist or
     to a node that does not contain a letter.)
*/
public String decode(String encodedMessage) {
```

For example, if the `MorseCodeTree` consisted only of the first 5 letters of the alphabet as shown in question 3 here are the results of various calls to `decode`.

`mct` is a `MorseCodeTree` variable that refers to a tree that has already been constructed.

```
mct.decode("-.-.*.-*-...*") -> returns "CAB"
mct.decode(".*.*.*.*") -> returns "EEEE"
mct.decode("-...*.*.-*-..*") -> returns "BEAD"
mct.decode(".....*.*.*") -> returns null, invalid encoded message
mct.decode("*") -> returns null, invalid encoded message
mct.decode(".*.-*-.-.-.-*.*.*") -> returns null, invalid encoded
                                          message
```

On this question you may not use any classes other than the `BNode<E>` class from question 3 and the String class. (Any methods from the String class you want including String concatenation.)

Recall the method you are completing is in the MorseCodeTree class so you have access to the private instance variable named root, which refers to the root node of the binary tree.

```
public class MorseCodeTree {

     private BNode<Character> root;
```

Complete the method on the next page.

```
/*   pre: encodedMessage != null, encodedMessage.length() > 0,
     encodedMessage consists of periods, hyphens, and asterisks
     only. The last character in code will be an asterisk.
     post: return what encodedMessage decodes to if it is
     correctly formed or null if encodedMessage is badly formed.
     (Some part of the code leads to a node that does not exist or
     to a node that does not contain a letter.)
*/
public String decode(String encodedMessage) {
```

5. (New data structures, 20 points) Completely implement an iterator class for a hash table.

Recall a hash table is a data structure that stores elements based on their hash code. Items are placed into a native array. The `Object` class defines a method named `hashCode` that returns an `int`. Thus, all Java classes have a `hashCode` method.

When an object is added to a hash table its hash code is obtained. The hash code is divided by the length of the hash table's native array to get the remainder. The remainder gives the index in the array where the object being added belongs.

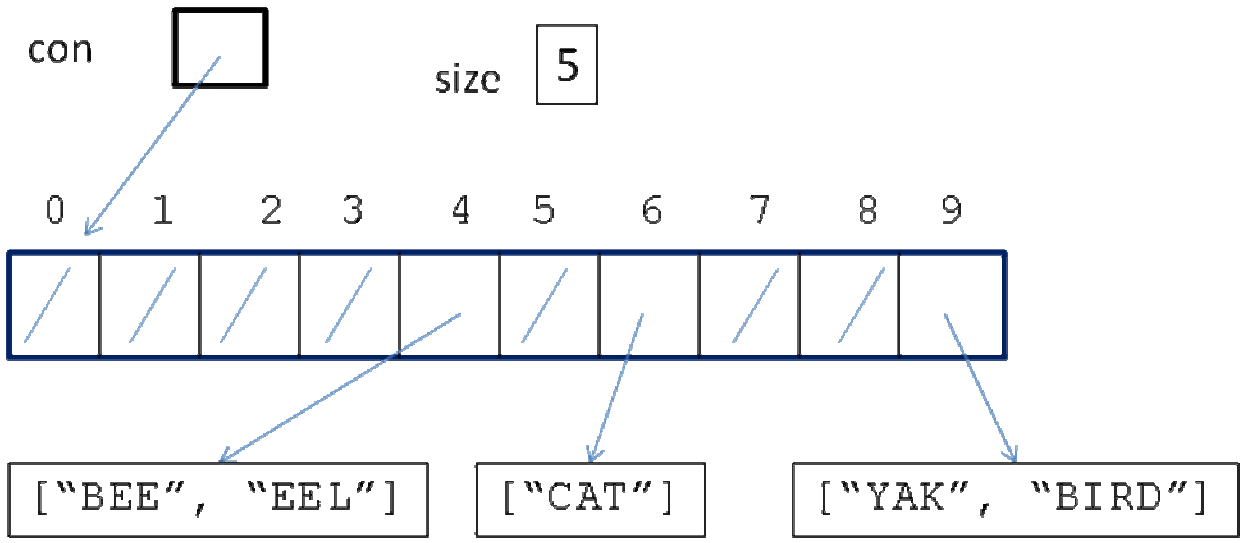
A collision occurs if the target element in the array is already occupied. Collisions can be resolved by open address hashing or chaining. The hash table in this question uses chaining to resolve collisions. Thus, each element (also referred to as a chain or bucket) in the hash tables array is actually a reference to an `ArrayList`. If no objects are in a particular element in the hash table's index the reference is set to `null` instead of to an empty `ArrayList`.

Consider the following example. The following Strings are inserted in the order shown to a hash table that initially has an array of length 10.

```
"BEE".hashCode() returns 65634 -> 65634 % 10 = 4
"YAK".hashCode() returns 87619 -> 87619 % 10 = 9
"EEL".hashCode() returns 68524 -> 68524 % 10 = 4
"CAT".hashCode() returns 66486 -> 66486 % 10 = 6
"BIRD".hashCode() returns 2038969 -> 2038969 % 10 = 9

HashTable<String> ht = new HashTable<String>();
ht.add("BEE");
ht.add("YAK");
ht.add("EEL");
ht.add("CAT");
ht.add("BIRD");
```

After the code above executes `ht` would refer to a hash table object with instance variables as shown below:



You will implement an iterator for the `HashTable` class. The following code:

```
HashTable<String> ht = new HashTable<String>();
ht.add("BEE");
ht.add("YAK");
ht.add("EEL");
ht.add("CAT");
ht.add("BIRD");

Iterator<String> it = ht.iterator();
while(it.hasNext())
    System.out.print( it.next() + " " );
```

would produce the following output:

```
BEE EEL CAT YAK BIRD
```

IMPORTANT: Your method must be O(1) space meaning you cannot use an extra native array or List.

Here is the `HashTable` class:

```
public class HashTable<E> implements Iterable{

     private ArrayList<E>[] container;
     private int size; // number of elements in hash table

     public Iterator<E> iterator(){
         return new HashIterator();
     }

     // other methods not shown

     // Complete the following inner class:

     private HashIterator implements Iterator<E> {

          // instance variables
          private boolean removeOK;
          // add your instance variables here;



          private HashIterator() {
               removeOK = false;
               // add your code here



          } // end of constructor
```

```
public boolean hasNext() {
     // complete this method









} // end of hasNext



public E next() {
     assert hasNext();
     removeOK = true;
     // complete this method









} // end of next
```

```
    public void remove(){
         assert removeOK;
         removeOK = false;
         // complete this method









    } // end of remove

    // more room if needed

  } // end of HashIterator class
} // end of HashTable class
```

Name:_____

Answer sheet for question 1, short answer questions. **Put answers next to the letter.**

_____

A:

_____

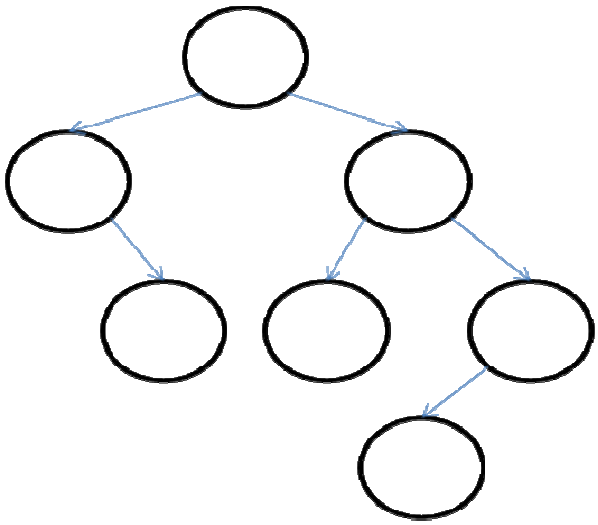
B. _____

C._____

D._____

E._____

F.



G._____

H._____

I._____

J._____

K._____

L._____

M._____

N._____

O._____

P._____

Q._____

R._____

S._____

T._____