ECE264 Fall 2019 Exam 3

8-9 PM, November 20

Please keep the answer sheet clean. Do not use the answer sheet as your scratch space. The answer sheet should contain **only** the answers.

Do not write anything that is not the answers.
Otherwise, you may lose points.

Please use **DARK** ink. If your pen is too light, your answer may not be graded.
Please write at the center of each box.

1 Construct Binary Trees

In HW 15, your program constructs binary trees using in-order and post-order traversals. Please be aware that a binary tree is not necessarily a binary search tree.

1.1 Tree Construction

Choose **every** correct statement.
If a statement is correct and you do not choose it, you lose points for that statement.
If a statement is incorrect and you choose it, you lose points for that statement.
This question asks you the condition when a binary tree can be **uniquely** constructed. The emphasis is "uniquely".

1. Given the in-order and the post-order of a binary tree, the tree can be uniquely constructed.
2. Given the in-order and the pre-order of a binary tree, the tree can be uniquely constructed.
3. Given the in-order and the post-order of a binary tree, the tree can be uniquely constructed **only if** the tree is a binary search tree. If it is not a binary search tree, in-order and post-order cannot decide a unique binary tree.
4. The last value in the post-order is the root of the binary tree.
5. Given the pre-order and the post-order of a binary tree, the tree can be uniquely constructed.

1.2 Pre-Order and In-Order

If a binary tree has n nodes and it has the same pre-order and in-order. What is the largest possible value of n?
Hint: Your answer may be 0, 1, 2, 3, ... , or infinity. You can write ∞ if you prefer.

1.3 Pre-Order and Post-Order

If a binary tree has n nodes and it has the same pre-order and post-order. What is the largest possible value of n?
Hint: Your answer may be 0, 1, 2, 3, ... , or infinity. You can write ∞ if you prefer.

1.4 Ambiguity of Construction

Given the pre-order and the post-order of a binary tree with n nodes, it is possible that the tree cannot be uniquely constructed. In other words, it is possible that two different trees have the same pairs of pre-order and post-order. What is the smallest possible value of n that has at least two different trees with the same pre-order and post-order?

Hint: Your answer may be 0 (pre-order and post-order uniquely decides every binary tree), 1, 2, 3, ... , or infinity. You can write ∞ if you prefer. Please notice that this question does **not** ask you the situation when a tree's pre-order is the same as its post-order.

1.5 Tree Shape

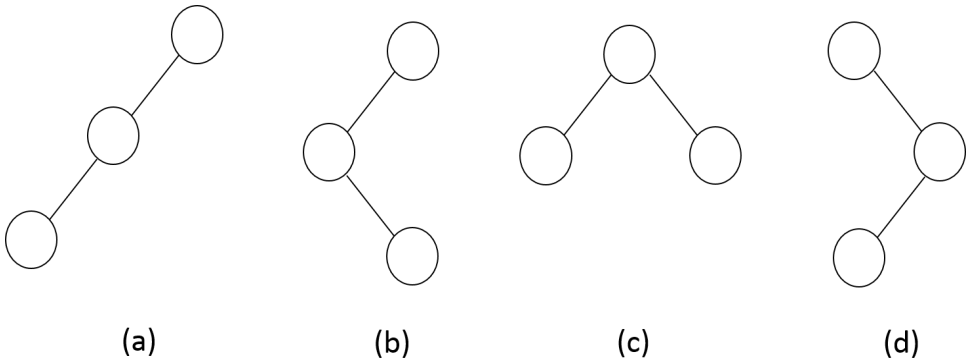

Figure 1: Different shapes of binary trees with three nodes.

Define $NL(r)$ as the number of nodes on the left side of node r; $NR(r)$ is the number of nodes on the right side of node r. Two trees T_1 and T_2 have same **shape** if

- $NL(r_1)$ is the same as $NL(r_2)$; $NR(r_1)$ is the same as $NR(r_2)$. Here, r_1 and r_2 are the roots of T_1 and T_2 respectively.
- The left side of r_1 has the same shape as the left side of r_2.
- The right side of r_1 has the same shape as the right side of r_2.

Please notice that this definition is recursive. Figure 1 shows four different shapes of binary trees, each with three nodes.
If a binary tree has n nodes and it has the same in-order and post-order, what is the largest possible number of shapes?
Answer:

1. 1 2 4: choosing 1, 2, 4 receiving one point each; not chosing 3, 5 receiving one point each
2. infinity or ∞: if every node has only right child. -2 if any finite value; -3 if zero; -5 if n (the question asks "What is the largest possible value of n?")
3. 1. -2 if the answer is 0 or 2
4. 2. -1 point if the answer is 3; -2 points if the answer is 4; -3 if the answer is 1
5. 1: if every node has only left child. -1 if the answer is 0 or 2. -3 any answer with n or infinity or 3 or 4.

2 Binary Search Tree

Consider the following binary search tree.

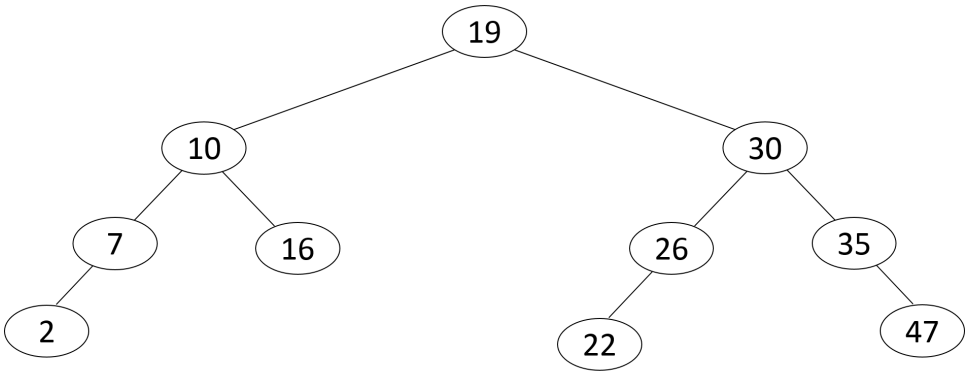

Figure 2: A binary search tree.

Every question starts from this tree.

2.1 Add Nodes

Add 14 to the binary search tree shown in Figure 2. Consider the pre-order traversal. What is the **sum** of the fifth and the sixth values?

2.2 Delete Nodes

Delete 26 from the binary search tree shown in Figure 2. Consider the post-order traversal. What is the **sum** of the fifth and the sixth values?

2.3 Delete Nodes using Incorrect Function

Consider the following **incorrect** function. Do not correct it.

```
1  // treedelete.c
2  #include "tree.h"
3  #include <stdlib.h>
4  TreeNode * Tree_delete(TreeNode * tn, int val)
5  {
6    if (tn == NULL) { return NULL; }
7    if (val < (tn -> value))
8      {
9        tn -> left = Tree_delete(tn -> left, val);
10       return tn;
11     }
```

```
12    if (val > (tn -> value))
13      {
14        tn -> right = Tree_delete(tn -> right, val);
15        return tn;
16      }
17    // v is the same as tn ->  value
18    if (((tn ->  left) == NULL) && ((tn ->  right) == NULL))
19      {
20        // tn has no child
21        free (tn);
22        return NULL;
23      }
24    if ((tn ->  left) == NULL)
25      {
26        // tn ->  right must not be NULL
27        TreeNode * rc = tn ->  right;
28        free (tn);
29        return rc;
30      }
31    if ((tn ->  right) == NULL)
32      {
33        // tn ->  left must not be NULL
34        TreeNode * lc = tn ->  left;
35        free (tn);
36        return lc;
37      }
38    // tn have two children
39
40    // find the immediate successor
41    // The following is correct. It is in a comment.
42    // TreeNode * su = tn ->  right; // su must not be NULL
43    // The following is wrong. This is what is used.
44    // DO NOT CORRECT THE MISTAKE.
45
46    TreeNode * su = tn; // <-------- ERROR
47
48
49    while ((su -> left) != NULL)
50      {
51        su = su -> left;
52      }
53    // su is tn's immediate successor
```

```
54    // swap their values
55    tn ->  value = su -> value;
56    su -> value = val;
57    // delete su
58    tn ->  right = Tree_delete(tn ->  right, val);
59    return tn;
60  }
```

If 7 is deleted from the binary search tree shown in Figure 2. Consider the pre-order traversal. What is the **sum** of the third and the fourth values?

2.4 Delete Nodes using Incorrect Function

Continue using the **incorrect** function shown earlier.
If 10 is deleted from the binary search tree shown in Figure 2. Consider the in-order traversal. What is the **sum** of the first and the second values?

2.5 Delete Nodes using Incorrect Function

Continue using the **incorrect** function shown earlier.
If 16 is deleted from the binary search tree shown in Figure 2. Consider the post-order traversal. What is the **sum** of the third and the fourth values?
Answer:

1. 30; -1 if 18 or 44; -2 if 9 or 56.
2. 69; -1 if 32 or 82; -2 if 26 or 65.
3. 18; -1 if 12 or 46; -2 if 29 or 57.
4. 17; -1 if 12; -2 if 9 or 26 or 23 or 18.
5. 32; -1 if 17 or 69; -2 if 9 or 82.

Explanaion of partial credits:

1. 30 = 16 + 14; -1 if 18 = 2 + 16 or 44 = 14 + 30; -2 if 9 = 7 + 2 or 56 = 30 + 26
2. 69 = 22 + 47; -1 if 32 = 10 + 22 or 82 = 47 + 35; -2 if 26 = 16 + 10 or 65 = 35 + 30
3. 18 = 2 + 16; -1 if 12 = 10 + 2 or 46 = 16 + 30; -2 if 29 = 19 + 10 or 57 = 22 + 35. The bug is not triggered when deleting 7.
4. 17 = 10 + 7; -1 if 12 = 10 + 2; -2 if 9 = 7 + 2 or 26 = 10 + 16 or 23 = 7 + 16 or 18 = 2 + 16. The bug is triggered when deleting 10.
5. 32 = 10 + 22; -1 if 17 = 7 + 10 or 69 = 22 + 47; -2 if 9 = 2 + 7 or 82 = 47 + 35. The bug is not triggered when deleting 16.

3 Infix and Postfix

3.1 Calculate Postfix Expression

What is the result of this postfix expression?

```
3  5  +  2  1  +  *  6  4  -  *
```

3.2 Convert Postfix to Infix

Convert the previous postifx expression to infix. How many pairs of parentheses does this expression have?

3.3 Convert Infix to Postfix

The following infix expression

```
( 1  +  2  +  3 )  *  ( 4  +  5  +  6 )
```

is converted to postfix as

```
1  2  + 3  + 4  5  + 6  +  *
```

If the postfix expession is stored in a linked list, `1` is the first node and `*` is the eleventh node.
Convert

```
( 1  +  2  +  3 )  *  ( 4  +  5 ) * ( 6 + 7 + 8 )
```

to postfix expression.
The first `*` (multiplication) is the x^{th} node and the second `*` is the y^{th} node. What is $x + y$?

3.4 Valid Postfix Expressions

This question considers only binary operator, i.e., needing two operands. A valid postfix expession is stored in a linked list and this list has n nodes. Consider where an operator (`+`, `-`, `*`) may appear. Let x be the first possible location where an operator may appear. In order words, if the first node in the linked list may be an operator, then x is 1. If the first node in the linked list must not be an operator but the second node may be an operator, then x is 2. Let y be the last possible location where an operator may appear. What is the $x + y$?

3.5 True / False

A binary operator needs two operands. For example, in this infix expression `2 * 6`, `*` is a binary operator. The postfix expression is `2 6 *`. This question considers only binary operators and valid postfix expressions.
Write T (true) or F (false) whether the following statement is correct.

"If a linked list stores a valid postfix expression and the linked list's length is n, then n must be a multiple of 3"

"Multiple of 3" means that you can find an integer k such that $n = 3k$.
Please write only T or F. **DO not write anything else.** You will receive **no** point if you write True, true, False, or false.
Answer:

1. 48
2. 3; -1 if 2 or 4; -2 if 1 or 5
3. 24 or 29; -1 if 23, 28, or 30; -2 if 22, 25, 27, or 31
4. $n + 3$; -1 if $n + 2$ or $n + 4$; -2 if $n + 1$ or $n + 5$
5. F

Explanation of partial credits:

1. (3 + 5) * (2 + 1) * (6 - 4)
2. 3 (3 + 5) * (2 + 1) * (6 - 4)
3. 24 or 29: 24 = 9 + 15 because 1 2 + 3 + 4 5 + * 6 7 + 8 + *; 29 = 14 + 15 because 1 2 + 3 + 4 5 + 6 7 + 8 + * *
4. $n + 3$
5. F, 1 + 2 + 3 becomes 1 2 3 + + and n is 5

4 Linked List

This question asks you to write code for detecting whether two linked lists share one or multiple nodes (i.e., pointing to the same memory addresses). For simplicity, this question considers only singly linked list; i.e., each node has only the `next` pointer and no `prev` pointer. Consider Figure 3 as an example. Assume both linked lists end with `NULL`, i.e., not circular linked list.

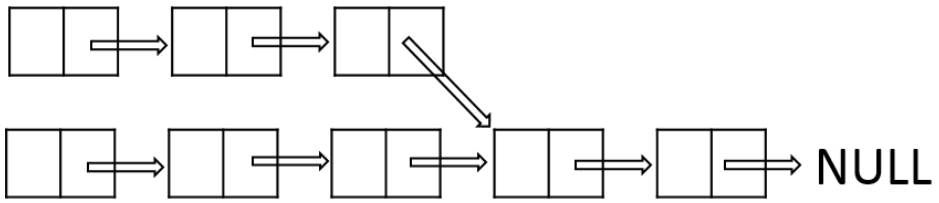

Figure 3: Two linked lists share nodes.

Hint: If two nodes reside in different memory addresses, they are not considered sharing even if they store the same value.

4.1 True / False

Write T (true) or F (false) whether the following statement is correct.

"Two linked lists share nodes if and only if they have the same last node."

Please write only T or F. **DO not write anything else.** You will receive **no** point if you write True, true, False, or false.
`NULL` is not considered a valid node. Thus, two linked lists are not considered sharing because both end with `NULL`.

4.2 Find Last Node

Consider this definition of `ListNode`:

```
1 // list.h
2 typedef struct lnode
3 {
4   int value;
5   struct lnode * next;
6 } ListNode;
```

The following function finds the last node of a linked list.

```
1  // This function returns the last node of a linked list
2  // If the linked list is empty (i.e., NULL), return NULL
3  #include <list.h>
4
5  ListNode * findLast(ListNode * head)
6  {
7    if (head == NULL) // empty list
8      {
9        return NULL;
10     }
11   ListNode * p = head;
12   while ( (/* <--- ANSWER HERE ---> */) != NULL)
13     {
14       p = p -> next;
15     }
16   return p;
17 }
```

4.3 Determine Sharing

The following function determines whether two linked lists shares one or more nodes.

```
1  // Determine whether two linked lists share one or more nodes
2  // If they share a node, return true.
3  // Otherwise, return false.
4  //
5  // If both lists are empty, they are not considered sharing.
6
7  #include <list.h>
8
9  bool shareNode(ListNode * list1, ListNode * list2)
10 {
11   ListNode * tail1 = findLast(list1);
12   ListNode * tail2 = findLast(list2);
13   if ((tail1 != NULL) && ( /* <--- ANSWER HERE ---> */ ))
14     {
15       return true;
16     }
17   return false;
18 }
```

4.4 Find First Shared Node

The following functions find the first shared node of two linked lists. If they do not share a node, this function returns `NULL`.

```
 1 #include <list.h>
 2 // calculate the length of a linked list
 3 static int findLength(ListNode * head)
 4 {
 5   int len = 0;
 6   while (/* <--- ANSWER D ---> */)
 7     {
 8       len ++;
 9       head = head -> next;
10     }
11   return len;
12 }
13
14 // This function finds the first shared nodes in two linked lists
15 // return NULL if the two linked lists do not share node (or nodes)
16 ListNode * findFirstShare(ListNode * list1, ListNode * list2)
17 {
18   // If either linked list is empty (i.e., NULL), return NULL
19   if ((list1 == NULL) || (list2 == NULL))
20     {
21       return NULL;
22     }
23   // method:
24   // 1. find the lengths of the lists, called len1 and len2
25   // 2. suppose len1 is larger, from the head of list1, move
26   //    forward as many as (len1 - len2) nodes
27   // 3. move forward list1 and list2 simutaneously until finding
28   //    the same node
29   int len1 = findLength(list1);
30   int len2 = findLength(list2);
31   ListNode * p1 = list1;
32   ListNode * p2 = list2;
33   if (len1 < len2)
34     {
35       // swap the pointers and the lengths
36       p1 = list2;
37       p2 = list1;
38       int temp = len1;
39       len1 = len2;
```

```
40        len2 = temp;
41      }
42    // len1 must be >= len2 now
43    while ((len1 > len2) && (p1 != NULL))
44      {
45        // move forward
46        len1 --;
47        p1 = p1 -> next;
48      }
49    // from p1 to NULL has the same number of nodes as
50    // from p2 to NULL
51    while ((p1 != NULL) && (p2 != NULL))
52      {
53        if (p1 == p2)
54          {
55            // <--- ANSWER E --->
56          }
57        p1 = p1 -> next;
58        p2 = p2 -> next;
59      }
60    return NULL; // do not find same node
61  }
```

Answer:
T

```
p -> next

tail1 == tail2

head != NULL

return p1; (or return p2;)
```