Assignment 1 – PS10 - My Paths

**Design Document**

**Group 020**

# **Problem Statement**

You want to walk in a forest but you can only walk the paths where the sum is your lucky number (given). You start at a fixed point forming a tree of paths. Don’t worry at least one valid path will always be there.

Eg: Input: 1,4,3,null,null,-10,null,10,2::5, Paths: 1,4;1,3,-10,11

Requirements:

1. Implement the above problem statement as a DFS.

2. Analyze the time complexity of your algorithm.

3. Implement the above problem statement using Python 3.7.

4. Make sure proper exception handling is written for the code.

# **Design Considerations**

## Data Structure Consideration:

The input (format of tree) provided for us in the file is not a perfect binary tree representation of the array. We have to parse the input and construct our own Tree before finding the lucky number. Have used a linked list to create a Binary tree by parsing the given input.

## Operations:

| Operation | Description | Time Complexity | Efficiency |
| --- | --- | --- | --- |
| constructTreeList | The input is converted to a binary tree having a linked list as the data structure. | O(n) | The time complexity does not exceed the number of nodes to be added in the binary tree. |
| findLuckyNumberPath | A recursive function to traverse the binary tree in a DFS fashion to find the lucky number path using a string path variable. | O(n+v) | The time complexity does not exceed the number of nodes and vertices of the binary tree. |
|  |  |  |  |

## Algorithm:

Algorithm createNode(value):

node <- new TreeNode()

node->value <- value

node->leftNode <- null

node->rightNode <- null

return node

# Time complexity O(n)

Algorithm constructTreeList(inputTreeArray):

rootNode <- createNode(inputTreeArray[1])

leafQueue <- createQueue()

leafQueue.enqueue(rootNode)

sizeOfInputArray <- size(inputTreeArray)

for(j <- 2, j <= sizeOfInputArray, j <- j + 2)

leaf <- leafQueue.dequeue()

if leaf == null

return error

leftValue = inputTreeArray[j]

if leftValue != null

leaf->leftNode <- createNode(leftValue)

leafQueue.enqueue(leaf->leftNode)

rightValue = inputTreeArray[j+1]

if rightValue != null

leaf->rightNode <- createNode(rightValue)

leafQueue.enqueue(leaf->rightNode)

empty(leafQueue)

return rootNode

# Time complexity O(n+v)

# Using string path variable to print path

Algorithm findLuckyNumberPath(node, path, sum, luckyNum):

if node == null

return

valueOfNode <- node->value

path <- path + valueOfNode + ','

sum <- sum + valueOfNode

if sum == luckyNum

#Print luckyNum path

print(path)

# Traverse left sub tree

findLuckyNumberPath(node->leftNode, path, sum, luckyNum)

# Traverse right sub tree

findLuckyNumberPath(node->rightNode, path, sum, luckyNum)

return

## Alternative Solution with Cost Implication

Alternate Solution: The problem can be modeled through an iterative method using stacks having a time complexity of O(n²).

Highlights:

* Avoids potential stack overflow issues
* Better control over memory management

Low Lights:

* The time complexity being O(n²) is greater in comparison to the chosen approach

# **Implementation Details**

Attached Program ‘my\_paths.py’ reads input lines, parses them, constructs a binary tree, finds paths, and writes the output to the specified file.The code is organized into functions, promoting modularity and code readability.

Key Implementation Details:

1. Input and Output Files: The program uses constants INPUT\_FILE and OUTPUT\_FILE to specify input and output file names.
2. Binary Tree Representation : Class ‘TreeNode’ represents the node value.
3. Tree Construction: ‘construct\_tree\_list’ function builds a binary tree from the list of numbers
4. Finding the Path: ‘find\_my\_paths’ function traverses all paths from the root to leaf nodes in the binary tree. It calculates the sum of values along the path and checks if the sum equals a specified lucky number. For the Paths that satisfy the condition are stored in a list ‘paths’.
5. Input Data Parsing: ‘parse\_input\_line’ function extracts the value for the binary tree and the lucky number from a line of input , and also handles the ‘ValueError’ exceptions during the conversion process.
6. Output File Updation: ‘write\_to\_file’ function writes the output paths to the specified output file
7. Exception Handling: try-except blocks are used for exception handling, ensuring that potential errors, such as ‘ValueError’, are caught and handled.

# **Test Results**

Test Result Summary:

| S. No | Input Data | Output Data | Result | Comments |
| --- | --- | --- | --- | --- |
| 1 | 5,4,8,11,null,9,4,-7,2,null,null,5,1::22 | 5,4,11,2;5,8,9;5,8,4,5 | Pass |  |
| 2 | 1,4,3,null,null,-10,null,11,2::5 | 1,4;1,3,-10,11 | Pass |  |
| 3 | 1,2,3,4,5,null,-4,1::0 | 1,3,-4 | Pass |  |
| 4 | 7,-2,15,9,20,-2,null,null,null,-4,-5::20 | 7,-2,20,-5;7,15,-2 | Pass |  |
| 5 | 10::10 | 10 | Pass | Only Root Node |
| 6 | 9,7,5,null,null,null,null::14 | 9,5 | Pass |  |
| 7 | 10,5,-3,3,9,11,10,3,-2,null,-5,6,8,null,0,1,2,null,null,null,-4,null,null,7,-2,null,4,null,null,null,null,null,9,null,null,null,null,null,3::24 | 10,5,9,-5,-4,9;10,-3,11,6;10,-3,11,8,-2;10,-3,10,0,4,3 | Pass |  |
| 8 | 0,0,0,0,0::0 | 0,0,0;0,0,0;0,0 | Pass |  |
| 9 | 1,2,3,4,5,null,null,null,null,null,6::22 |  | Pass | Negative Scenario: No Valid Path |
| 10 | ::5 |  | Pass | Negative Scenario: Invalid Input |
| 11 | 1,2,3,null,null,4,5::10 |  | Pass | Negative Scenario: No Valid Path |
| 12 | null,2,3,4,5,6::9 |  | Pass | Negative Scenario: Root Node is null |
| 13 | 1,2,3,4,5,6::null |  | Pass | Negative Scenario: Lucky Number is null |