GetAhead - Interview Practice 3

Longest Path in Tree - Solution

Write a function that computes the length of the longest path of consecutive integers in a tree.

A node in the tree has a value and a set of children nodes. A tree has no cycles and each node has exactly one parent. A path where each node has a value 1 greater than its parent is a path of consecutive integers (e.g. 1,2,3 not 1,3,5).

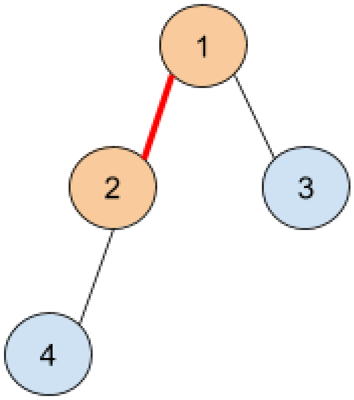
A few things to clarify:

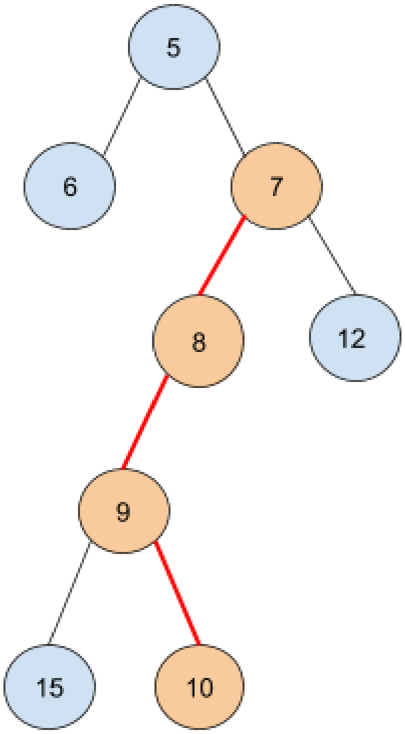
- Integers are all positive
- Integers appear only once in the tree

Test Cases

Note that there may be other valid answers.

For the tree on the left, the length of the longest path is 2, for that on the right, it's 4





Solution

JAVA

```java
// Tree.java
// =========

import java.util.Collections;
import java.util.List;

public class Tree {
   private int mValue;
   private List<Tree> mChildren;

   public Tree(int value, List<Tree> children) {
       mValue = value;
       mChildren = Collections.unmodifiableList(children);
   }

   public int longestPath() {
       return longestPath_Rec(Integer.MIN_VALUE, 0);
   }

   public int longestPath_Rec(int valueInParent, int pathLengthInParent) {
       // Check if this node extends parent's path.
       // If not, this node forms a new path of length 1.
       int currentPathLength = (this.mValue == valueInParent + 1)
               ? pathLengthInParent + 1 : 1;
       int maxLength = currentPathLength;
       // Recursively invoke on all children and find if any form
       // an even longer continuous path.
       for (Tree child : mChildren) {
           int maxChildLength = child.longestPath_Rec(
                   this.mValue, currentPathLength);
           maxLength = Math.max(maxLength, maxChildLength);
       }
       // Return length of longest path from this node and its children.
       return maxLength;
   }
}
```

```
// TreeTest.java
// =============

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import java.util.List;

class TreeTest {
   @Test
   public void testLongestPath_SingleNode() {
       Tree tree = new Tree(1, List.of());
       assertEquals(1, tree.longestPath());
   }

   @Test
   public void testLongestPath_PathStartsInRoot() {
       Tree tree = new Tree(1, List.of(
               new Tree(2, List.of(new Tree(4, List.of()))),
               new Tree(3, List.of())));
       assertEquals(2, tree.longestPath());
   }

   @Test
   public void testLongestPath_PathStartsInChild() {
       Tree tree = new Tree(5, List.of(
               new Tree(6, List.of()),
               new Tree(7, List.of(
                       new Tree(8, List.of(
                               new Tree(9, List.of(
                                       new Tree(15, List.of()),
                                       new Tree(10, List.of()))))),
                       new Tree(12, List.of())))));
       assertEquals(4, tree.longestPath());
   }
}
```

C++

```cpp
#include <cassert>
#include <iostream>
#include <vector>

using namespace std;

struct tree {
  int value;
  vector<tree> children;
};


int longest_path(
    const tree &current,
    int parent_value, int current_chain, int longest) {
  if (current.value == parent_value + 1) {
    // This node is part of a continued chain.
    current_chain++;
  } else {
    // This node is the start of a new chain.
    current_chain = 1;
  }
  if (current_chain > longest) {
    longest = current_chain;
  }
  for (const tree &child : current.children) {
    // Recurse into the children and record their longest length.
    int child_length = longest_path(child, current.value, current_chain, longest);
    if (child_length > longest) {
      longest = child_length;
    }
  }
  return longest;
}

// Base case
int longest_path(const tree &root) {
  return longest_path(root, INT32_MIN, 1, 0);
}


// Test cases
int main() {
  assert (longest_path(tree{0}) == 1);
  assert (longest_path(
    tree{1, {
      tree{2, {
        tree{4}
      }},
      tree{3}
    }}
  ) == 2);

  assert (longest_path(
    tree{5, {
      tree{6},
      tree{7, {
        tree{8, {
          tree{9, {
            tree{15},
            tree{10}
          }}
        }},
        tree{12}
      }}
    }}
  ) == 4);
}
```

Python

```python
class Tree:
  def __init__(self, value, *children):
    self.value = value
    self.children = children


# It's Python, so we walk the tree by iterating through it, yielding the length of the path ending at each node we encounter, then take the max of that.
def longest_path(tree):

  def rec(current, parent_value=0, parent_path_length=0):
    # Length of the longest chain this node is a part of.
    current_path_length = (parent_path_length + 1
        if current.value == parent_value + 1 else 1)
    # Emit the length for this node.
    yield current_path_length
    # Recurse into the children
    for child in current.children:
      # For each of the descendant nodes, emit their lengths as well
      for value in rec(child, current.value, current_path_length):
        yield value

  # Take the overall maximum length.
  return max(rec(tree))


# "Tests"
if __name__ == '__main__':
  assert longest_path(Tree(1)) == 1

  assert longest_path(
    Tree(1,
      Tree(2,
        Tree(4)),
      Tree(3))
    ) == 2

  assert longest_path(
    Tree(5,
      Tree(6),
      Tree(7,
        Tree(8,
          Tree(9,
            Tree(15),
            Tree(10))),
        Tree(12)))
      ) == 4
```