GetAhead - Interview Practice 2

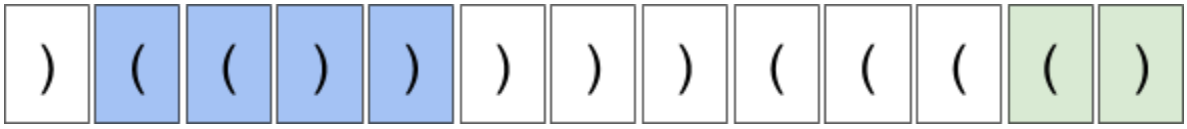
Balanced Parentheses - Solution

Given a string of parentheses, find the size of the longest contiguous substring of balanced parentheses. Parentheses are considered balanced when there is a valid closing parenthesis for an opening one.

Example:

In this string: `())(())`, the longest continuous set would be 4 characters long (the last 4 characters of the input):

For `)((((((((((((` , the max length would be 4:



Solution

The following three solutions use a linear approach to scan the string exactly once, and make use of a Stack to keep track of the open parentheses that still need to be closed. The ability of the candidate to choose the appropriate data structure to solve a problem is one of the qualities that interviewers look for in a candidate.

JAVA

```
// In an interview you can generally omit the import statements,
// however if you decide to use some library (like Stack in this example)
// it's a good idea to mention it to the Interviewer and ask if it's allowed.
import java.util.Stack;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class BalancedParentheses {
   public static int longestBalanced(String str) {
```

```
        Stack<Integer> stack = new Stack<>();
        int longest = 0;
        for (int i = 0; i < str.length(); ++i) {
            if (str.charAt(i) == '(') {
                // Remember index of the opening parenthesis.
                stack.push(i);
            } else if (!stack.empty()) {
                // If we find a closing parenthesis, and there is an
                // unclaimed open one, recall its index and remove it
                // from the stack, since we are claiming it.
                int open_i = stack.pop();
                // Compute the distance between it and the matching
                // closing parenthesis.
                int length = i - open_i + 1;
                // If this is the new maximum, remember it.
                longest = Math.max(length, longest);
            }
        }
        // Return the maximum.
        return longest;
    }

    // You don't necessarily have to write test code in an interview, but
    // you are still expected to provide meaningful test cases and try some
    // manually.
    @Test
    public void testLongestBalenced() {
        assertEquals(0, longestBalanced(""));
        assertEquals(0, longestBalanced("("));
        assertEquals(0, longestBalanced(")"));
        assertEquals(4, longestBalanced("(())"));
        assertEquals(2, longestBalanced("()("));
        assertEquals(2, longestBalanced("())"));
        assertEquals(2, longestBalanced("()()"));
        assertEquals(6, longestBalanced("((()))"));
        assertEquals(4, longestBalanced("()(())"));
        assertEquals(4, longestBalanced(")(((((()))))"));
    }
}
```

C++

```
// In an interview you can generally omit the include statements,
// however if you decide to use some library (like std::stack in this example)
```

```cpp
// it's a good idea to mention it to the Interviewer and ask if it's allowed.
#include <string>
#include <cassert>
#include <stack>

size_t longest_balanced(std::string str) {  // O(n) time
  std::stack<size_t> stack;  // O(n) space
  size_t longest = 0;
  for (size_t i = 0; i < str.size(); i++) {
    if (str[i] == '(') {
      // Remember index of the opening parenthesis.
      stack.push(i);
    } else if (!stack.empty()) {
      // If we find a closing parenthesis, and there is an unclaimed open one,
      // recall its index
      size_t open_i = stack.top();
      // compute the distance between it and the matching closing one,
      size_t length = i - open_i + 1;
      // and remove it from the stack, since we claimed it.
      stack.pop();
      // If this length is the new maximum, remember it.
      if (length > longest) longest = length;
    }
  }
  // And return it.
  return longest;
}

// You don't necessarily have to write test code in an interview, but
// you are still expected to provide meaningful test cases and try some
// manually.
int main() {
  assert(longest_balanced("(())") == 4);
  assert(longest_balanced("()(") == 2);
  assert(longest_balanced("())") == 2);
  assert(longest_balanced("()()") == 2);
  assert(longest_balanced("((()))") == 6);
  assert(longest_balanced("())(())") == 4);
  assert(longest_balanced(")((((((()))))") == 4);
}
```

Python

```python
def longest_balanced(string):
```

```
  stack = []  # O(n) space.
  longest = 0
  for i, char in enumerate(string):  # O(n) time.
    if char == '(':
      # Remember index of the opening parenthesis.
      stack.append(i)
    elif char == ')':
      # If we previously encountered an opening parenthesis,
      if stack:
        # We recall the index of the last one that hasn't been matched yet.
        open_i = stack.pop()
        # And compute the distance between them.
        # All the parentheses between these two must have been balanced!
        length = i - open_i + 1
        # Keep track of the longest encountered so far
        if length > longest:
          longest = length
  # And return it.
  return longest

# You don't necessarily have to write test code in an interview, but
# you are still expected to provide meaningful test cases and try some
# manually.
if __name__ == '__main__':
  assert longest_balanced('()(') == 2
  assert longest_balanced('())') == 2
  assert longest_balanced('(()())') == 6
  assert longest_balanced('())(())') == 4
  assert longest_balanced(')(((((((((()(') == 4
  assert longest_balanced('(())') == 4
```