Code Jam 2020 - Qualification Round

Analysis: Nesting Depth

Test Set 1

To solve Test Set 1, we can put an opening parenthesis before each group of `1`s and a closing parenthesis after.

We can use the following trick to simplify the implementation: prepend and append one extra `0` to **S**. Then the implementation is just replacing `01` with `0(1` and `10` with `1)0`, which can be written in one line of code in some programming languages. Don't forget to remove the extra `0`s from the end of the resulting string!

Test Set 2

For convenience, let's once again use the trick described above: prepend and append extra `0`s to **S**, and then scan **S** from left to right.

Suppose we see some number A immediately followed by some larger number B and suppose all of the previously inserted parentheses would leave A at the right nesting depth — that is, there are exactly A unmatched opening parentheses preceding A, and no unmatched closing parentheses. For B to be at nesting depth B we need to add at least B - A opening parentheses. We can just do that and nothing else, to keep the final string length minimal. Any additional opening parentheses we would add would need to be closed before B, which would needlessly lengthen the string. Similarly, if we see some number A immediately followed by some smaller number B, we can just insert A - B closing parentheses. And in the case when A is equal to B, we don't need to add anything.

We don't need any parentheses before the temporary `0` in the beginning, or after the one in the end, so we can just drop them before printing the result.

Since we only add p parentheses when at least p are needed, the resulting string is of minimum length.

An inefficient but fun solution

The problem can be solved using only string replacements. First, replace each digit D with D `(`s, then the digit itself, then D `)`s. Then eliminate all instances of `)(`, collapsing the string each time, until there are no more to remove.

Here's a Python3 implementation:

```
for C in range(int(input())):
  rawstr = ''.join([int(x) * '(' + x + ')' * int(x) for x in str(input())])
  for _ in range(9):
    rawstr = rawstr.replace(')(', '')
  print("Case #{}: {}".format(C+1, rawstr))
```