Kick Start 2022 - Coding Practice with Kick Start Session #2

Analysis: Sherlock and Parentheses

View problem and solution walkthrough video

The problem translates to finding the maximum number of balanced non-empty substrings possible in a string, generated from the given number of left and right parentheses. A string S consisting only of characters `(` and/or `)` is *balanced* if:

- It is an empty string, or:
- It has the form `(`S`)`, where S is a balanced string, or:
- It has the form S_1S_2, where S_1 is a balanced string and S_2 is a balanced string.

Test Set 1

Naively, we can generate all the permutations of strings possible from $\mathbf{L}$ left parentheses `(` and $\mathbf{R}$ right parentheses `)`. The time complexity of generating all the permutations will be in order of possible permutations i.e. $O(\frac{(\mathbf{L}+\mathbf{R})!}{\mathbf{L}!\times\mathbf{R}!})$.

For counting the number of balanced non-empty substrings in a string, we can use a basic counter which increases by 1 for a left parenthesis `(` and decreases by 1 for a right parenthesis `)`. While iterating a given string, if the value of the counter becomes negative before reaching the end, then this string cannot be a balanced string because it denotes the occurrence of right `)` parenthesis before it's matching left `(` paranthesis. If the counter is 0 at the end of the string, avoiding being negative throughout the string then it denotes a balanced string. We can do this check for all the substrings in a efficient way using a nested for loop, which results in a Time complexity of $O(N^2)$ where N is the length of the string:

```
int CountBalancedSubstrings(String S) {
  int N = S.length();
  int balanced_substrings = 0;
  for (int i = 0; i < N; i++) {
    int counter = 0;
    for (int j = i; j < N; j++) {
      if (S[j] == '(')
        counter++;
      else
        counter--;
      if (counter < 0) break;
      if (counter == 0) balanced_substrings++;
    }
  }
  return balanced_substrings;
}
```

So overall, we can generate all the permutations of the string, count the balanced substrings and return the maximum possible number of balanced non-empty substrings in $O(\frac{(\mathbf{L}+\mathbf{R})!}{\mathbf{L}!\times\mathbf{R}!}\times(\mathbf{L}+\mathbf{R})^2)$ Time complexity.

Test Set 2

We cannot generate all the possible permutations of strings as the solution would timeout. We cannot even count the number of balanced substrings in a string using our earlier $O(N^2)$ algorithm as N can go upto 10^5.

We need to identify the optimal strategy to get the maximum number of balanced substrings. We can observe few things from our CountBalancedSubstrings algorithm:

- The counter should not become *negative* while iterating throughout the string.
- The number of balanced substrings we get is directly proportional to the number of times the counter becomes 0.

So this translates to the optimal strategy of arranging left parentheses `(` and right parentheses `)` like this `()()()()()` $\ldots\ldots$
Now every left parenthesis requires a right parenthesis and vice versa for forming a balanced string. So we can say this pattern will follow till $2 \times N$ length where $N = min(\mathbf{L}, \mathbf{R})$, and the rest of the string would not be a part of any balanced substring as it would either include all left `(` parantheses or all right `)` parantheses.
Now in this pattern we can see that the first left parenthesis `(` forms a balanced substring with all the N right parentheses `)` ahead of it. Similarly, every left parenthesis `(` forms a balanced substring with all the right parentheses `)` ahead of it till $2 \times N$ length. So the total number of balanced non-empty substrings will be $1 + 2 + 3 + 4 + \cdots + N = \frac{N \times (N+1)}{2}$

So for each test case, we can calculate N and then the maximum possible number of balanced non-empty substrings in $O(1)$ Time and Space complexity.