678. Valid Parenthesis String



- Description (/problems/valid-parenthesis-string/description/)
- ♀ Hints (/problems/valid-parenthesis-string/hints/)

Quick Navigation -

View in Article (/articles/valid-parenthesis-string)



Approach #1: Brute Force [Time Limit Exceeded]

Intuition and Algorithm

For each asterisk, let's try both possibilities.

```
■ Copy
Java
       Python
 1
    class Solution {
2
        boolean ans = false;
 4
        public boolean checkValidString(String s) {
5
            solve(new StringBuilder(s), 0);
 6
            return ans;
7
9
        public void solve(StringBuilder sb, int i) {
            if (i == sb.length()) {
10
11
                ans |= valid(sb);
12
            } else if (sb.charAt(i) == '*') {
13
                for (char c: "() ".toCharArray()) {
                    sb.setCharAt(i, c);
14
15
                    solve(sb, i+1);
16
                    if (ans) return;
17
18
                sb.setCharAt(i, '*');
19
            } else
20
                solve(sb, i + 1);
21
22
        public boolean valid(StringBuilder sb) {
23
24
            int bal = 0;
            for (int i = 0; i < sb.length(); i++) {
25
26
                char c = sb.charAt(i);
                if (c == '(') bal++;
27
                if (c == ')') bal--;
28
```

Complexity Analysis

- Time Complexity: $O(N * 3^N)$, where N is the length of the string. For each asterisk we try 3 different values. Thus, we could be checking the validity of up to 3^N strings. Then, each check of validity is O(N).
- Space Complexity: O(N), the space used by our character array.

Approach #2: Dynamic Programming [Accepted]

Intuition and Algorithm

Let dp[i][j] be true if and only if the interval s[i], s[i+1], ..., s[j] can be made valid. Then dp[i] [j] is true only if:

- s[i] is '*', and the interval s[i+1], s[i+2], ..., s[j] can be made valid;
- or, s[i] can be made to be '(', and there is some k in [i+1, j] such that s[k] can be made to be
 ')', plus the two intervals cut by s[k] (s[i+1: k] and s[k+1: j+1]) can be made valid;



Complexity Analysis

- Time Complexity: $O(N^3)$, where N is the length of the string. There are $O(N^2)$ states corresponding to entries of dp , and we do an average of O(N) work on each state.
- Space Complexity: $O(N^2)$, the space used to store intermediate results in dp.

Approach #3: Greedy [Accepted]

Intuition

When checking whether the string is valid, we only cared about the "balance": the number of extra, open left brackets as we parsed through the string. For example, when checking whether '(()())' is valid, we had a balance of 1, 2, 1, 0 as we parse through the string: '(' has 1 left bracket, '((' has 2, '(()' has 1, and so on. This means that after parsing the first i symbols, (which may include asterisks,) we only need to keep track of what the balance could be.

For example, if we have string '(***)', then as we parse each symbol, the set of possible values for the balance is [1] for '('; [0, 1, 2] for '(*'; [0, 1, 2, 3] for '(***'; [0, 1, 2, 3] for '(***', and [0, 1, 2, 3] for '(***)'.

Furthermore, we can prove these states always form a contiguous interval. Thus, we only need to know the left and right bounds of this interval. That is, we would keep those intermediate states described above as [lo, hi] = [1, 1], [0, 2], [0, 3], [0, 4], [0, 3].

Algorithm

Let lo, hi respectively be the smallest and largest possible number of open left brackets after processing the current character in the string.

If we encounter a left bracket (c = '('), then l_{0++} , otherwise we could write a right bracket, so l_{0--} . If we encounter what can be a left bracket (c != ')'), then hi++, otherwise we must write a right bracket, so hi--. If hi < 0, then the current prefix can't be made valid no matter what our choices are. Also, we can never have less than 0 open left brackets. At the end, we should check that we can have exactly 0 open left brackets.

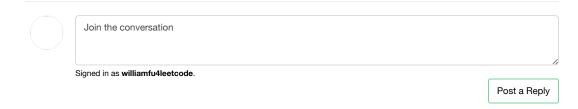


Complexity Analysis

T

- Time Complexity: O(N), where N is the length of the string. We iterate through the string once.
- ullet Space Complexity: O(1), the space used by our lo and hi pointers. However, creating a new character array will take O(N) space.

Analysis written by: @awice (https://leetcode.com/awice)



theoptips commented 19 hours ago

@david301 (https://discuss.leetcode.com/uid/1620) I was wondering about that too, (https://discuss.leetcode.com/user/theoptips) but I think in the case of * empty string the check valid function won't pick it up if char == '(' char ==')' change the balance, but * doesn't, so it will automatically pass.

D david301 commented 3 months ago

How do you handle the case where * is an empty string? (https://discuss.leetcode.com/user/david301)

N nergi.r commented 7 months ago

@shafiul (https://discuss.leetcode.com/uid/319785) Here it is (https://discuss.leetcode.com/user/nergi-

```
class Solution {
public:
    int solve(string &s, int pos, int cnt, vector<vector<int>> &memo) {
        if(cnt<0) return 0;
        if(pos==s.size()) return (cnt==0);
        int &ret = memo[pos][cnt+100];
        if(ret!=-1) return ret;
        if(s[pos]=='(') return ret = solve(s, pos+1, cnt+1, memo);
        else if(s[pos]==')') return ret = solve(s, pos+1, cnt-1, memo);
        ret = solve(s, pos+1, cnt, memo);
        return ret = max(ret, max(solve(s, pos+1, cnt+1, memo), solve(s, pos+1, cnt-1, me
    }
    bool checkValidString(string s) {
        vector<vector<int> > memo(s.size(), vector<int>(500, -1));
        return solve(s, 0, 0, memo);
    }
};
```