The idea is to similar to validate a string only contains '(' and ')'. But extend it to tracking the lower and upper bound of valid '(' counts. My thinking process is as following.

scan from left to right, and record counts of unpaired ‘(’ for all possible cases. For ‘(’ and ‘)’, it is straightforward, just increment and decrement all counts, respectively.

When the character is '*', there are three cases, ‘(’, empty, or ‘)’, we can think those three cases as three branches in the ongoing route
Take “(**())” as an example. There are 6 chars:
----At step 0: only one count = 1.
----At step 1: the route will be diverted into three branches.
so there are three counts: 1 - 1, 1, 1 + 1 which is 0, 1, 2, for ‘)’, empty and ‘(’ respectively.
----At step 2 each route is diverged into three routes again. so there will be 9 possible routes now.
-- For count = 0, it will be diverted into 0 – 1, 0, 0 + 1, which is -1, 0, 1, but when the count is -1, that means there are more ‘)’s than ‘(’s, and we need to stop early at that route, since it is invalid. we end with 0, 1.
-- For count = 1, it will be diverted into 1 – 1, 1, 1 + 1, which is 0, 1, 2
-- For count = 2, it will be diverted into 2 – 1, 2, 2 + 1, which is 1, 2, 3
To summarize step 2, we end up with counts of 0,1,2,3
----At step 3, increment all counts --> 1,2,3,4
----At step 4, decrement all counts --> 0,1,2,3
----At step 5, decrement all counts --> -1, 0,1,2, the route with count -1 is invalid, so stop early at that route. Now we have 0,1,2.
In the very end, we find that there is a route that can reach count == 0. Which means all ‘(’ and ‘)’ are cancelled. So, the original String is valid.
Another finding is counts of unpaired ‘(’ for all valid routes are consecutive integers. So we only need to keep a lower and upper bound of that consecutive integers to save space.
One case doesn’t show up in the example is: if the upper bound is negative, that means all routes have more ‘)’ than ‘(’ --> all routes are invalid --> stop and return false.

```
class Solution {
    public boolean checkValidString(String s) {
        return checkHelper(s, 0, 0, 0);
    }
    private boolean checkHelper(String s, int l, int r, int idx) {
        if (idx == s.length()) {
            return l == r;
        }
        if (l - r > s.length() - idx)
            return false;
        char c = s.charAt(idx);
        if (c == '(') {
            return checkHelper(s, l+1, r, idx+1);
        } else if (c == ')') {
            return l > r && checkHelper(s, l, r+1, idx+1);
        } else {
            return checkHelper(s, l, r, idx+1) ||
                    checkHelper(s, l+1, r, idx+1) ||
                    l > r && checkHelper(s, l, r+1, idx+1);
        }
    }
```

```
public boolean checkValidString(String s) {
    int low = 0, high = 0;
    for (char c : s.toCharArray()) {
        if (c == '(') {
            low++;
            high++;
        } else if (c == ')') {
            if (low > 0) {
                low--;
            }
            high--;
        } else if (c == '*') {
            if (low > 0) {
                low--;
            }
            high++;
        }
        if (high < 0) {
            return false;
        }
    }
    return low == 0;
}}
```