Remove Invalid Parentheses

Remove the minimum number of invalid parentheses in order to make the input string valid. Return all possible results.

Note: The input string may contain letters other than the parentheses `(` and `)`.

Examples:

```
"()())()" -> ["()()()", "(())()"]
"(a)())()" -> ["(a)()()", "(a())()"]
")(" -> [""]
```

Credits:
Special thanks to @hpplayer for adding this problem and creating all test cases.

Solution 1

The idea is straightforward, with the input string `s`, we generate all possible states by removing one `(` or `)`, check if they are valid, if found valid ones on the current level, put them to the final result list and we are done, otherwise, add them to a queue and carry on to the next level.

The good thing of using BFS is that we can guarantee the number of parentheses that need to be removed is minimal, also no recursion call is needed in BFS.

Thanks to @peisi, we don't need stack to check valid parentheses.

Time complexity:

In BFS we handle the states level by level, in the worst case, we need to handle all the levels, we can analyze the time complexity level by level and add them up to get the final complexity.

On the first level, there's only one string which is the input string `s`, let's say the length of it is `n`, to check whether it's valid, we need `O(n)` time. On the second level, we remove one `(` or `)` from the first level, so there are `C(n, n-1)` new strings, each of them has `n-1` characters, and for each string, we need to check whether it's valid or not, thus the total time complexity on this level is `(n-1)` x `C(n, n-1)`. Come to the third level, total time complexity is `(n-2)` x `C(n, n-2)`, so on and so forth...

Finally we have this formula:

`T(n)` = `n` x `C(n, n)` + `(n-1)` x `C(n, n-1)` + ... + `1` x `C(n, 1)` = `n` x `2^(n-1)`.

Following is the Java solution:

```java
public class Solution {
    public List<String> removeInvalidParentheses(String s) {
      List<String> res = new ArrayList<>();

      // sanity check
      if (s == null) return res;

      Set<String> visited = new HashSet<>();
      Queue<String> queue = new LinkedList<>();

      // initialize
      queue.add(s);
      visited.add(s);

      boolean found = false;

      while (!queue.isEmpty()) {
        s = queue.poll();

        if (isValid(s)) {
          // found an answer, add to the result
          res.add(s);
          found = true;
        }

        if (found) continue;

        // generate all possible states
        for (int i = 0; i < s.length(); i++) {
          // we only try to remove left or right paren
          if (s.charAt(i) != '(' && s.charAt(i) != ')') continue;

          String t = s.substring(0, i) + s.substring(i + 1);

          if (!visited.contains(t)) {
            // for each state, if it's not visited, add it to the queue
            queue.add(t);
            visited.add(t);
          }
        }
      }

      return res;
    }

    // helper function checks if string s contains valid parantheses
    boolean isValid(String s) {
      int count = 0;

      for (int i = 0; i < s.length(); i++) {
        char c = s.charAt(i);
        if (c == '(') count++;
        if (c == ')' && count-- == 0) return false;
      }

      return count == 0;
    }
}
```

written by jeantimex original link here

Solution 2

Here I share my `DFS` or `backtracking` solution. It's `10X` faster than optimized `BFS`.

1. Limit max removal `rmL` and `rmR` for backtracking boundary. Otherwise it will exhaust all possible valid substrings, not shortest ones.
2. Scan from left to right, avoiding invalid strs (on the fly) by checking num of `open` parens.
3. If it's `'('`, either use it, or remove it.
4. If it's `'('`, either use it, or remove it.
5. Otherwise just append it.
6. Lastly set `StringBuilder` to the last decision point.

In each step, make sure:

1. `i` does not exceed `s.length()`.
2. Max removal `rmL` `rmR` and num of `open` parens are non negative.
3. De-duplicate by adding to a `HashSet`.

Compared to `106 ms` `BFS (Queue & Set)`, it's faster and easier. Hope it helps! Thanks.

```
public List<String> removeInvalidParentheses(String s) {
    Set<String> res = new HashSet<>();
    int rmL = 0, rmR = 0;
    for(int i = 0; i < s.length(); i++) {
        if(s.charAt(i) == '(') rmL++;
        if(s.charAt(i) == ')') {
            if(rmL != 0) rmL--;
            else rmR++;
        }
    }
    DFS(res, s, 0, rmL, rmR, 0, new StringBuilder());
    return new ArrayList<String>(res);
}

public void DFS(Set<String> res, String s, int i, int rmL, int rmR, int open, Stri
ngBuilder sb) {
    if(i == s.length() && rmL == 0 && rmR == 0 && open == 0) {
        res.add(sb.toString());
        return;
    }
    if(i == s.length() || rmL < 0 || rmR < 0 || open < 0) return;

    char c = s.charAt(i);
    int len = sb.length();

    if(c == '(') {
        DFS(res, s, i + 1, rmL - 1, rmR, open, sb);
        DFS(res, s, i + 1, rmL, rmR, open + 1, sb.append(c));

    } else if(c == ')') {
        DFS(res, s, i + 1, rmL, rmR - 1, open, sb);
        DFS(res, s, i + 1, rmL, rmR, open - 1, sb.append(c));

    } else {
        DFS(res, s, i + 1, rmL, rmR, open, sb.append(c));
    }

    sb.setLength(len);
}
```

written by yavinci original link here

Solution 3

```
class Solution {
public:
    vector<string> removeInvalidParentheses(string s) {
        unordered_set<string> result;
        int left_removed = 0;
        int right_removed = 0;
        for(auto c : s) {
            if(c == '(') {
                ++left_removed;
            }
            if(c == ')') {
                if(left_removed != 0) {
                    --left_removed;
                }
                else {
                    ++right_removed;
                }
            }
        }
        helper(s, 0, left_removed, right_removed, 0, "", result);
        return vector<string>(result.begin(), result.end());
    }
private:
    void helper(string s, int index, int left_removed, int right_removed, int pair
, string path, unordered_set<string>& result) {
        if(index == s.size()) {
            if(left_removed == 0 && right_removed == 0 && pair == 0) {
                result.insert(path);
            }
            return;
        }
        if(s[index] != '(' && s[index] != ')') {
            helper(s, index + 1, left_removed, right_removed, pair, path + s[inde
x], result);
        }
        else {
            if(s[index] == '(') {
                if(left_removed > 0) {
                    helper(s, index + 1, left_removed - 1, right_removed, pair, p
ath, result);
                }
                helper(s, index + 1, left_removed, right_removed, pair + 1, path
+ s[index], result);
            }
            if(s[index] == ')') {
                if(right_removed > 0) {
                    helper(s, index + 1, left_removed, right_removed - 1, pair, p
ath, result);
                }
                if(pair > 0) {
                    helper(s, index + 1, left_removed, right_removed, pair - 1, p
ath + s[index], result);
                }
            }
        }
    }
};
```

written by ethan_noc original link here