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

        return new ArrayList<>(res);
    }

    public void dfs(StringBuilder sb, int rmL, int rmR, int open, Set<String> set, String s, int len) {
        if (rmL < 0 || rmR < 0 || open < 0) return;
        if (len == s.length()) {
            if (rmL == 0 && rmR == 0 && open == 0) {
                set.add(sb.toString());
            }
            return;
        }
        int size = sb.length();
        char c = s.charAt(len);

        if (c == '(') {
            dfs(sb, rmL-1, rmR, open, set, s, len+1);
            dfs(sb.append(c), rmL, rmR, open+1, set, s, len+1);
        } else if (c == ')') {
            dfs(sb, rmL, rmR-1, open, set, s, len+1);
            dfs(sb.append(c), rmL, rmR, open-1, set, s, len+1);
        } else {
            dfs(sb.append(c), rmL, rmR, open, set, s, len+1);
        }
        sb.setLength(size);
    }
}
```

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 `)`.

Example 1:

```
Input: "()())()"
```

```
Output: ["()()()", "(())()"]
```

Example 2:

```
Input: "(a)())()"
Output: ["(a)()()", "(a())()"]
```

Example 3:

```
Input: ")("
Output: [""]
```