<https://leetcode.com/problems/minimum-remove-to-make-valid-parentheses/description/>

Given a string s of '(' , ')' and lowercase English characters.

Your task is to remove the minimum number of parentheses ( '(' or ')', in any positions ) so that the resulting *parentheses string* is valid and return **any** valid string.

Formally, a *parentheses string* is valid if and only if:

It is the empty string, contains only lowercase characters, or

It can be written as AB (A concatenated with B), where A and B are valid strings, or

It can be written as (A), where A is a valid string.

**Example 1:**

Input: s = "lee(t(c)o)de)"

Output: "lee(t(c)o)de"

Explanation: "lee(t(co)de)" , "lee(t(c)ode)" would also be accepted.

**Example 2:**

Input: s = "a)b(c)d"

Output: "ab(c)d"

**Example 3:**

Input: s = "))(("

Output: ""

Explanation: An empty string is also valid.

**Constraints:**

1 <= s.length <= 10^5

s[i] is either'(' , ')', or lowercase English letter

**Attempt 1: 2023-08-12**

**Solution 1: Two Stacks (360min, TLE, 62/62)**

**Key points:**

**1. Paired opening bracket '(' will convert into '#', unpaired opening bracket '(' will ignore during final string builder**

**2. 'tmp' stack temporarily used to cache the substring before current closing bracket ')' till its corresponding opening bracket '(' or if no corresponding opening bracket '(' then cache till the beginning of original input string**

**3. Two stacks 'tmp' and 'stack' cooperate to go back and forth to construct "filtered" string by constructing session by session based on encountering closing bracket ')'**

**4. In final scanning "filtered" string still need to handle redundant opening bracket '('  (e.g "))((", "ab(c(d", "ab(c(d)"...)**

class Solution {

    public String minRemoveToMakeValid(String s) {

        Stack<Character> stack = new Stack<Character>();

        Stack<Character> tmp = new Stack<Character>();

        for(char c : s.toCharArray()) {

            if(c != ')') {

                stack.push(c);

            } else {

                while(!stack.isEmpty()) {

                    if(stack.peek() != '(') {

                        // Store all english letters in tmp stack

                        // including '#' which means updated previous

                        // used '(' and previous ')'

                        tmp.push(stack.pop());

                    } else {

                        // Find previous not used corresponding '(' from

                        // stack update it from '(' to '#' for later

                        // identify it is a already used one

                        stack.pop();

                        stack.push('#');

                        break;

                    }

                }

                // If after removing all elements from stack and not able

                // to find previous not used corresponding '(', then no need

                // to push ')' onto stack, since we have to remove all

                // non-pairable ')' also, only need to restore stack from

                // tmp stack, if not empty means we find one '(' to build

                // a pair, then need to restore stack and also push ')'

                if(stack.isEmpty()) {

                    while(!tmp.isEmpty()) {

                        stack.push(tmp.pop());

                    }

                } else {

                    while(!tmp.isEmpty()) {

                        stack.push(tmp.pop());

                    }

                    stack.push(c); // equal to stack.push(')')

                }

            }

        }

        StringBuilder sb = new StringBuilder();

        while(!stack.isEmpty()) {

            char c = stack.pop();

            // Skip all unpaired '(', e.g "))((", expected "" not "(("

            if(c != '(') {

                if(c == '#') {

                    sb.insert(0, '(');

                } else {

                    sb.insert(0, c);

                }

            }

        }

        return sb.toString();

    }

}

Time Complexity: O(N^2)

Space Complexity: O(N)

**Solution 2: Stack + boolean[] array (30min)**

class Solution {

    public String minRemoveToMakeValid(String s) {

        boolean[] b = new boolean[s.length()];

        Stack<Integer> stack = new Stack<Integer>();

        for(int i = 0; i < s.length(); i++) {

            char c = s.charAt(i);

            if(c == '(') {

                stack.push(i);

            } else if(c == ')') {

                if(!stack.isEmpty()) {

                    b[i] = true;

                    b[(int)stack.pop()] = true;

                }

            } else {

                b[i] = true;

            }

        }

        StringBuilder sb = new StringBuilder();

        for(int i = 0; i < s.length(); i++) {

            if(b[i]) {

                sb.append(s.charAt(i));

            }

        }

        return sb.toString();

    }

}

Time Complexity: O(N)

Space Complexity: O(N)

**Refer to**

<https://leetcode.com/problems/minimum-remove-to-make-valid-parentheses/solutions/419402/java-c-stack/>

**Intuition**

To make the string valid with minimum removals, we need to get rid of all parentheses that do not have a matching pair.

Push char index into the stack when we see '('.

Pop from the stack when we see ')'.

If the stack is empty, then we have ')' without the pair, and it needs to be removed.

In the end, the stack will contain indexes of '(' without the pair, if any. We need to remove all of them too

**If you are using a stack, you can maintain a simple boolean array to decide whether to take a character into result or not.**

<https://leetcode.com/problems/minimum-remove-to-make-valid-parentheses/solutions/419402/java-c-stack/comments/413619>

class Solution {

    public String minRemoveToMakeValid(String s) {

        int len = s.length();

        boolean[] b = new boolean[s.length()];

        StringBuilder res = new StringBuilder("");

        Stack<Integer> st = new Stack<Integer>();

        for(int i=0;i<len;++i){

            if(s.charAt(i) == '(') st.push(i);

            else if(s.charAt(i) == ')'){

                if(!st.isEmpty()){

                  // match these pairs, all unmatched are false anyway

                    b[i] = true;

                    b[(int)st.pop()] = true;

                }

            }else{

                b[i] = true; // any character other than ( and ) are true anyway

            }

        }

        for(int i=0;i<len;++i){

            if(b[i]) res.append(s.charAt(i));

        }

        return res.toString();

    }

}

**Solution 3: Stack to record all opening bracket '(' index + Set to record redundant closing bracket ')' index  (30min)**

class Solution {

    public String minRemoveToMakeValid(String s) {

        Set<Integer> set = new HashSet<Integer>();

        Stack<Integer> stack = new Stack<Integer>();

        for(int i = 0; i < s.length(); i++) {

            char c = s.charAt(i);

            if(c == '(') {

                stack.push(i);

            } else if(c == ')') {

                if(!stack.isEmpty()) {

                    // Find valid '(' on stack to build "()"

                    // with current ')', pop out '(', then

                    // index for '(' removed from stack

                    stack.pop();

                } else {

                    // Since no valid '(' on stack, ignore ')'

                    // and store ')' index in set

                    set.add(i);

                }

            }

        }

        StringBuilder sb = new StringBuilder();

        // Why we need to reversely scan the array ?

        // Because stack store redundant '(' indexes reversely,

        // if scan from 0 to n - 1, stack.peek() will always not

        // able to match i, because order reversed

        // Test out by: "))(("

        for(int i = s.length() - 1; i >= 0; i--) {

            char c = s.charAt(i);

            if(c == '(') {

                if(!stack.isEmpty() && stack.peek() == i) {

                    // It's a redundant '(', remove it

                    stack.pop();

                } else {

                    // It's not a redundant '(' since its index not

                    // stored on stack, then add to final string

                    sb.insert(0, '(');

                }

            } else if (c == ')') {

                // If not a redundant ')' then add to final string

                if(!set.contains(i)) {

                    sb.insert(0, ')');

                }

            } else {

                sb.insert(0, c);

            }

        }

        return sb.toString();

    }

}

Time Complexity: O(N)

Space Complexity: O(N)

**Solution 4: Stack without Set (30min)**

**Based on Solution 3: we can remove Set**

class Solution {

    public String minRemoveToMakeValid(String s) {

        StringBuilder sb = new StringBuilder(s);

        Stack<Integer> stack = new Stack<Integer>();

        for(int i = 0; i < s.length(); i++) {

            char c = s.charAt(i);

            if(c == '(') {

                stack.push(i);

            } else if(c == ')') {

                if(!stack.isEmpty()) {

                    // Find valid '(' on stack to build "()"

                    // with current ')', pop out '(', then

                    // index for '(' removed from stack

                    stack.pop();

                } else {

                    // Since no valid '(' on stack, ignore ')'

                    // use StringBuilder directly, mask as '\*'

                    sb.setCharAt(i, '\*');

                }

            }

        }

        // Update all recorded redundant '(' and mask same as '\*'

        while(!stack.isEmpty()) {

            sb.setCharAt(stack.pop(), '\*');

        }

        // Remove all '\*' placeholder and left result string

        return sb.toString().replaceAll("\\\*", "");

    }

}

Time Complexity: O(N)

Space Complexity: O(N)

**Refer to**

<https://leetcode.com/problems/minimum-remove-to-make-valid-parentheses/solutions/419402/java-c-stack/>

**Approach 1: Stack and Placeholder**

We mark removed parentheses with '\*', and erase all of them in the end.

**Java**

public String minRemoveToMakeValid(String s) {

  StringBuilder sb = new StringBuilder(s);

  Stack<Integer> st = new Stack<>();

  for (int i = 0; i < sb.length(); ++i) {

    if (sb.charAt(i) == '(') st.add(i);

    if (sb.charAt(i) == ')') {

      if (!st.empty()) st.pop();

      else sb.setCharAt(i, '\*');

    }

  }

  while (!st.empty())

    sb.setCharAt(st.pop(), '\*');

  return sb.toString().replaceAll("\\\*", "");

}