**[Remove Invalid Parentheses](https://leetcode.com/problems/remove-invalid-parentheses/)**

**import** java.util.ArrayList;

**import** java.util.HashSet;

**import** java.util.LinkedList;

**import** java.util.List;

**import** java.util.Queue;

**import** java.util.Set;

**public** **class** RemoveParanthesis {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

System.***out***.println(*removeInvalidParentheses*(""));

}

**public** **static** List<String> removeInvalidParentheses(String s) {

**if**(s == **null**) {

**return** **new** ArrayList<String>();

}

Queue<String> queue = **new** LinkedList<>(); //to maintain each possible substring from given string

Set<String> visited = **new** HashSet<>(); //to track what all possible substring generated are already visited

List<String> result = **new** ArrayList<>(); //to add the valid parentheses

queue.offer(s);

visited.add(s);

**boolean** found = **false**;

**while**(!queue.isEmpty()) {

String temp = queue.poll();

**if**(*isValidSubstring*(temp)) { //to check if the generated string is valid parentheses sequence or not

result.add(temp);

found = **true**;

}

**if**(found)

**continue**;

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

**if**(temp.charAt(i) != '(' && temp.charAt(i) != ')') //to ignore characters other than ( and )

**continue**;

String str = temp.substring(0 , i) + temp.substring(i+1);//to create substring from ignoring each parentheses

**if**(!visited.contains(str)) { //if not visited substring then add to set and queue

visited.add(str);

queue.offer(str);

}

}

}

**return** result;

}

// helper function checks if string s contains valid parantheses

**public** **static** **boolean** isValidSubstring(String word) {

**if**(word == **null** || word.length() == 0) **return** **true**;

**int** count = 0;

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

**if**(word.charAt(i) == '(') {

count++;

}

**else** **if**(word.charAt(i) == ')') {

count--;

**if**(count < 0) {

**return** **false**;

}

}

}

**return** count == 0;

}

}

Time Complexity : O(C(n, k) + n) where k is the number of chars needs remove

Space Complexity : O(C(n, k) + n) where k is the number of chars needs remove