Question: https://leetcode.com/problems/longest-valid-parentheses/

I used dp to solve the solution

so the dp matrix was first implemented to keep track of how many brackets are opening and closing later I found out that the marix could be eliminated there by saving some space cause I was evaluating the longest string while putting matrix values.

So as we know that if we are constructing matrix for substring then it is always the left top to right bottom diagonal along with its right portion because the other side substrings woul be like(3,0)or(7,3) which is not possible.

therefore always j starts from the value of i.

now for every row if we find out that there is a open parenthesis then we will increase pairtrack value by 1, or else we would reduce it. Now if we get negative that means there are more closing parenthesis then opening in case that will never result to valid substring so we break out of it and move to next row.

and if pairtrack becomes zero that means there is a valid substring present.

In this second approach I have used stack and it is best explained in <https://www.youtube.com/watch?v=VdQuwtEd10M>

Code:  
class Solution {

public int longestValidParentheses(String s) {

// int longestvalidlength=0;

// // int dp[][] = new int[s.length()][s.length()];

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

// int j=i,pairtrack=0;

// while(j<s.length()){

// if(s.charAt(j)=='('){

// pairtrack++;

// }else if(s.charAt(j)==')'){

// pairtrack--;

// }

// // dp[i][j]=pairtrack;

// if(pairtrack<0){

// break;

// }else if(pairtrack==0){

// if(longestvalidlength<(j-i+1)){

// longestvalidlength=j-i+1;

// }

// }

// j++;

// }

// }

// return longestvalidlength;

int len=0;

Stack<Integer> st= new Stack();

st.push(-1);

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

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

st.push(i);

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

st.pop();

if(st.isEmpty()){

st.push(i);

}else{

if(len<(i-st.peek())){

len=i-st.peek();

}

}

}

}

return len;

}

}

Github Link :<https://lnkd.in/ecwtJeaz>