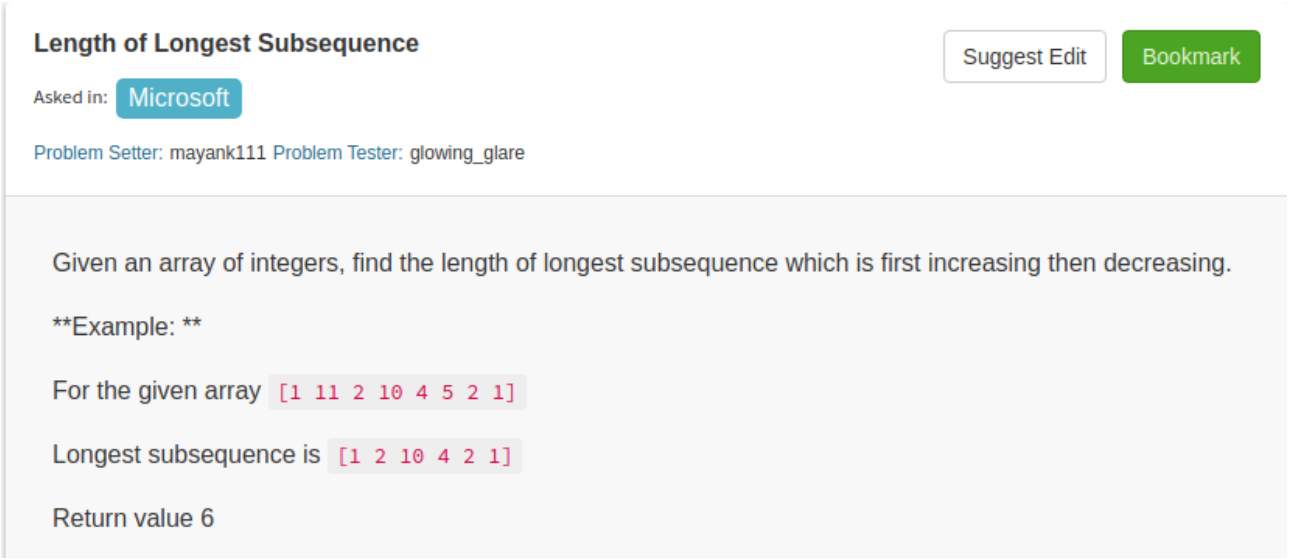
1.



Length of Longest Subsequence

Asked in: Microsoft

Problem Setter: mayank111 Problem Tester: glowing_glare

Suggest Edit | Bookmark

Given an array of integers, find the length of longest subsequence which is first increasing then decreasing.

**Example: **

For the given array `[1 11 2 10 4 5 2 1]`

Longest subsequence is `[1 2 10 4 2 1]`

Return value 6

```
public class Solution {
    // DO NOT MODIFY THE LIST. IT IS READ ONLY
    public int longestSubsequenceLength(final List<Integer> A) {
        int n=A.size();
        if(n==0)
         return 0;
        int a[]=new int[n];
        int b[]=new int[n];
        for(int i=0;i<n;i++){
            a[i]=1;
             b[i]=1;
        }
        for(int i=1;i<n;i++)
        {
            for(int j=0;j<i;j++){
                if(A.get(i)>A.get(j) && a[i]<a[j]+1 ){
                    a[i]=a[j]+1;
                 }
            }
        }
        for(int i=n-2;i>=0;i--){
            for(int j=n-1;j>i;j--){
                if(A.get(i)>A.get(j) && b[i]<b[j]+1){
                    b[i]=b[j]+1;
                 }
            }
        }
        int max=a[0]+b[0]-1;
        for(int i=1;i<n;i++){
```

```
            if(a[i]+b[i]-1>max){
                max=a[i]+b[i]-1;
            }
        }
        return max;

    }

}
```

2.

Stairs

Asked in: Morgan Stanley | Amazon | Intel

Suggest Edit | Bookmark

You are climbing a stair case. It takes n steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

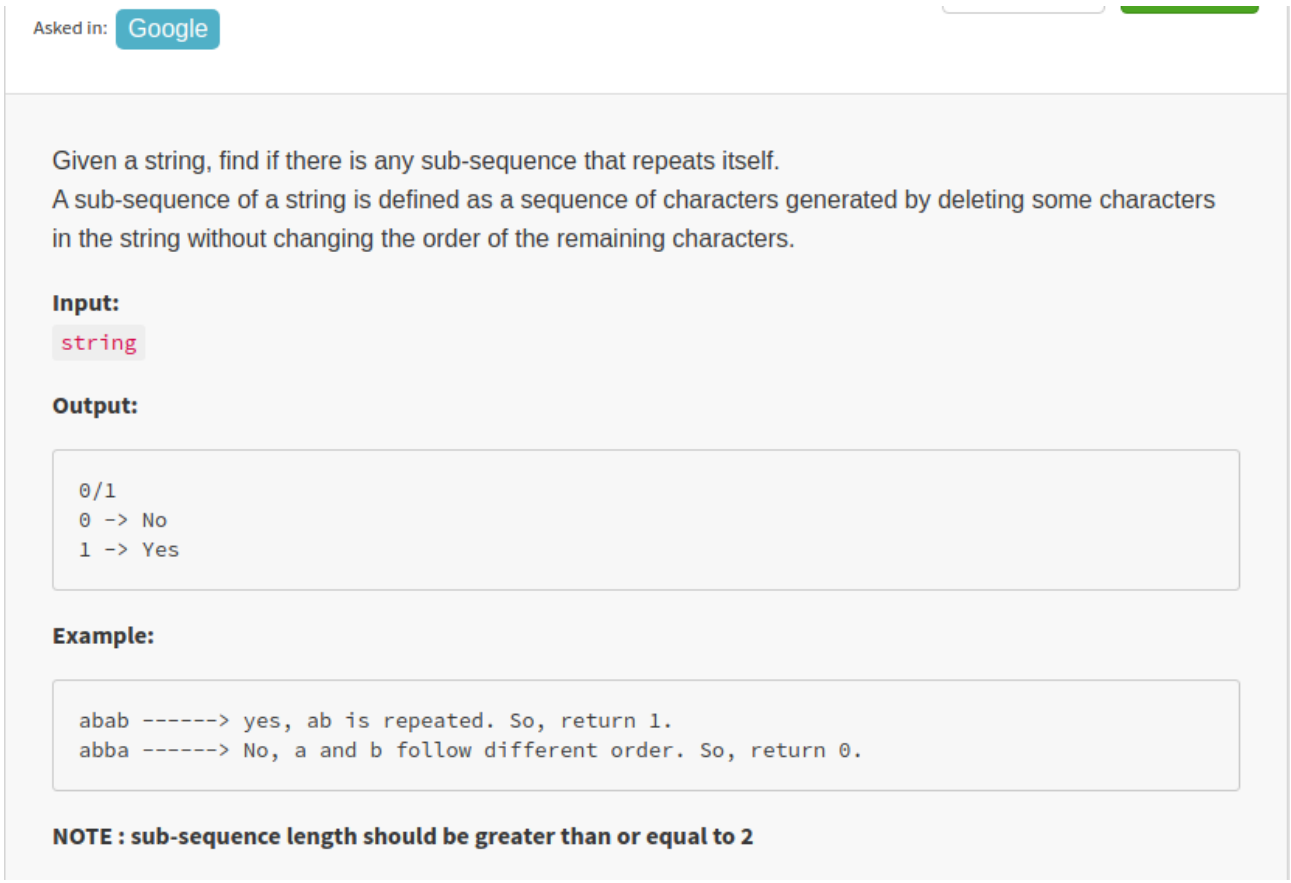
Example :

```
Input : 3
Return : 3

Steps : [1 1 1], [1 2], [2 1]
```

```
//I think it can be easily solved by fibonacci Series
//When we can take either 1 or 2 steps then we can add the previous two value to get the
//answer at that step

public class Solution {
    public int climbStairs(int n) {
        int[] a=new int[n];
        Arrays.fill(a,-1);
       return find_two_step(n,a);

    }
    public int  find_two_step(int n,int[] a){
    if(n<0){
            return 0;
        }
    else if(n==0){
            return 1;
        }
    else if(a[n]>-1){
        return a[n];
```

```
    }
    else
       a[n]=climbStairs(n-1)+climbStairs(n-2);
       return a[n];
    }

}
```

3. Repeating Subsequence



Asked in: Google

Given a string, find if there is any sub-sequence that repeats itself.
A sub-sequence of a string is defined as a sequence of characters generated by deleting some characters in the string without changing the order of the remaining characters.

Input:

`string`

Output:

```
0/1
0 -> No
1 -> Yes
```

Example:

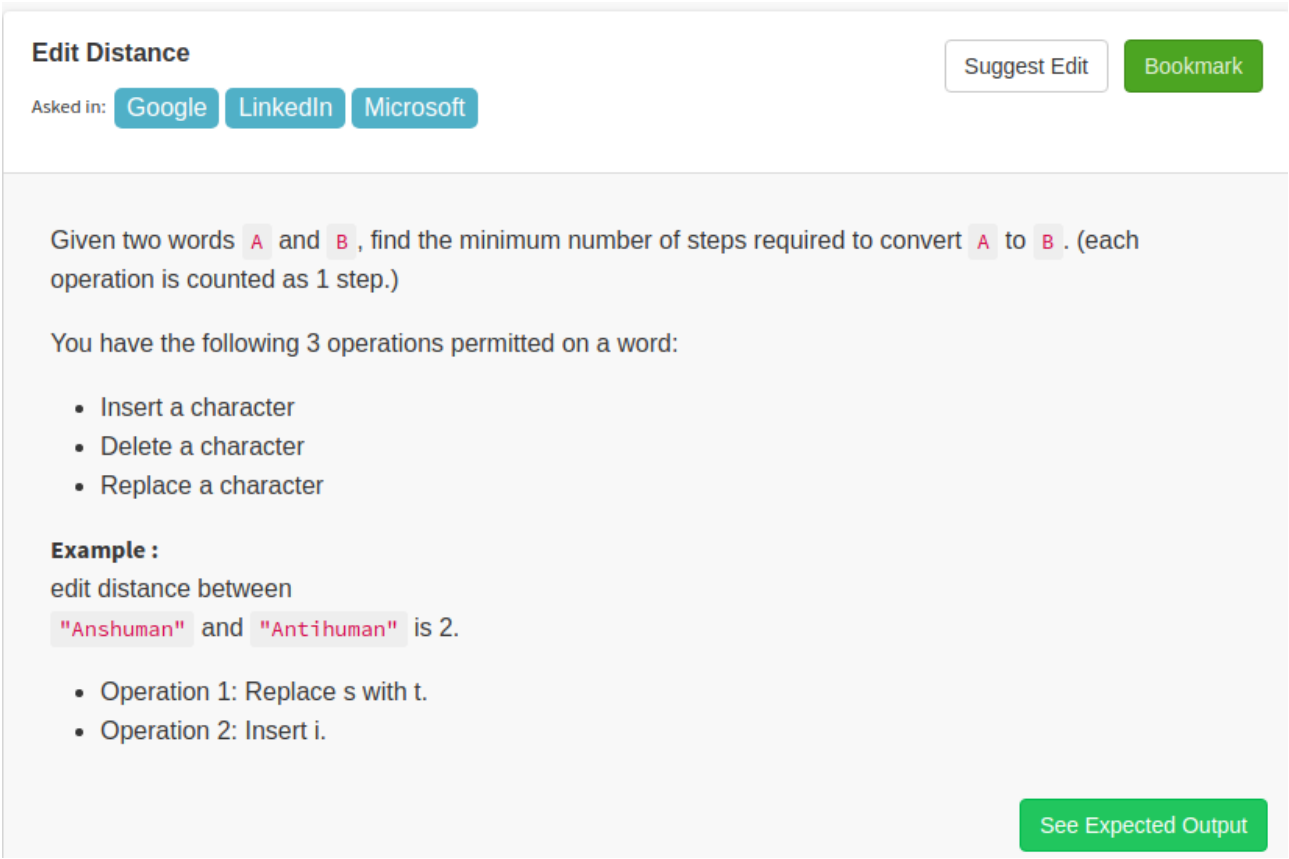
```
abab ------> yes, ab is repeated. So, return 1.
abba ------> No, a and b follow different order. So, return 0.
```

NOTE : sub-sequence length should be greater than or equal to 2

```
public class Solution {
    public int anytwo(String A) {
        int n=A.length();
        int temp[][]=new int[n+1][n+1];
        for(int i=1;i<=n;i++){
            for(int j=1;j<=n;j++){
                if(A.charAt(i-1)==A.charAt(j-1) && i!=j){
                    temp[i][j]=temp[i-1][j-1]+1;

                }
                else{

                    temp[i][j]=Math.max(temp[i-1][j],temp[i][j-1]);
```

```
            }
        }
    }
    if(temp[n][n]>1){
        return 1;
    }
    return 0;
  }
}
```

4.




```
public class Solution {

    public int minDistance(String a, String b) {
        int an=a.length();
        int bn=b.length();

        int[][] temp = new int[an+1][bn+1];
        for(int i=0;i<temp.length;i++){
            temp[i][0]=i;
        }
        for(int i=0;i<temp[0].length;i++){
            temp[0][i]=i;
        }
```

```
        for(int i=1;i<=an;i++){
            for(int j=1;j<=bn;j++){
                if(a.charAt(i-1)==b.charAt(j-1)){
                    temp[i][j]=temp[i-1][j-1];
                }
                else{

                    temp[i][j]=1+Math.min(temp[i-1][j-1],Math.min(temp[i-1][j],temp[i][j-1]));
                }


            }
        }

        return temp[an][bn];

    }
}
```

5.

Distinct Subsequences

Asked in: Google

Suggest Edit | Bookmark

Given two sequences S, T, count number of unique ways in sequence S, to form a subsequence that is identical to the sequence T.

> *Subsequence : A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, `"ACE"` is a subsequence of `"ABCDE"` while `"AEC"` is not).*

Example :

```
S = "rabbbit"
T = "rabbit"
```

Return `3` . And the formations as follows:

Return `3` . And the formations as follows:

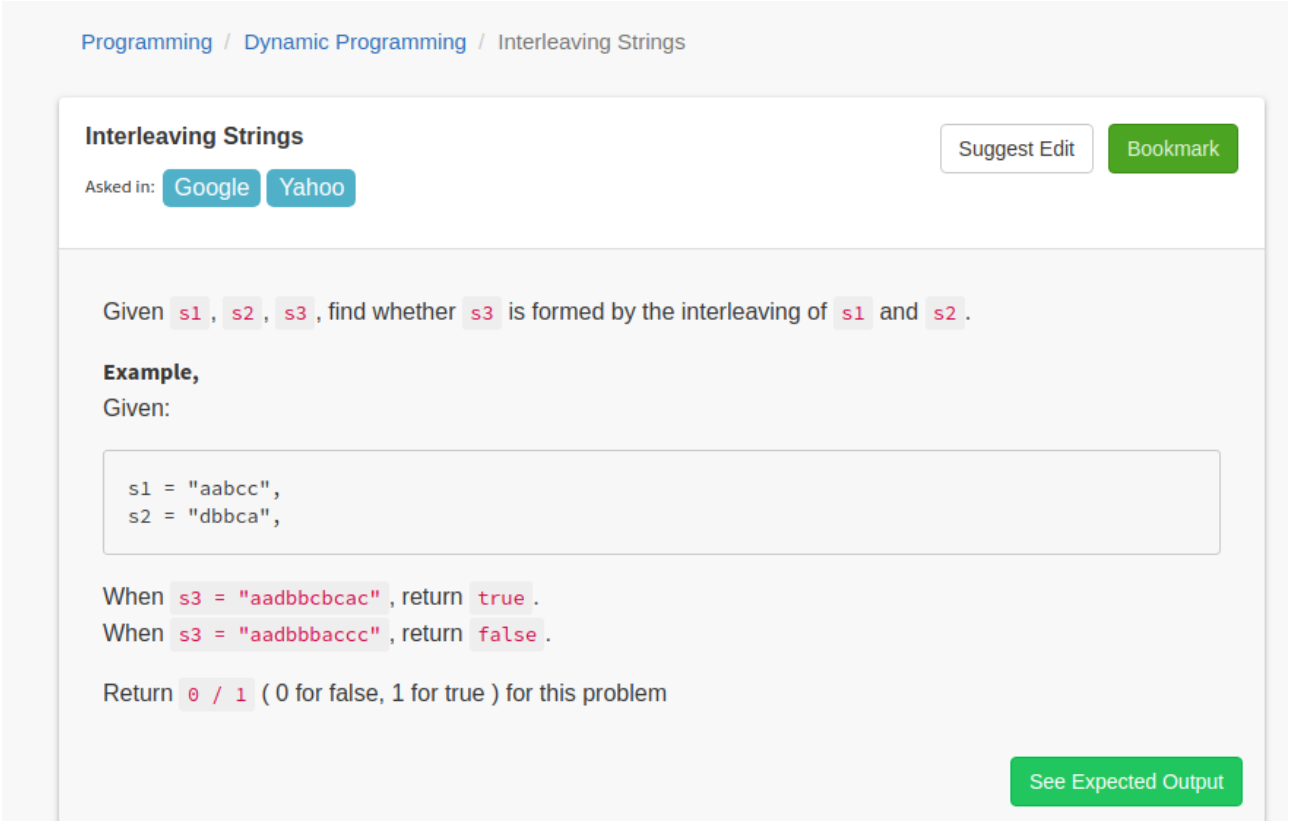
```
S1= "ra_bbit"
S2= "rab_bit"
S3="rabb_it"
```

`"_"` marks the removed character.

See Expected Output

```
public class Solution {
    public int numDistinct(String A, String B) {
        int m=A.length();
        int n=A.length();
        int[][] dp=new int[n+1][m+1];
        for(int i=0;i<=m;i++){
            dp[i][0]=1;
        }
        for(int i=1;i<=m;i++){
            for(int j=1;j<=n;j++){
                if(A.charAt(i-1)==B.charAt(j-1)){
                    dp[i][j]=dp[i-1][j]+dp[i-1][j-1];
                }
                else{
                    dp[i][j]=dp[i-1][j];
                }


            }
        }
        return dp[m][n];
        }
}
```

6.


Programming / Dynamic Programming / Interleaving Strings
Interleaving Strings
Asked in: Google Yahoo
Suggest Edit
Bookmark
Given s1 , s2 , s3 , find whether s3 is formed by the interleaving of s1 and s2 .
Example,
Given:
s1 = "aabcc",
s2 = "dbbca",
When s3 = "aadbbcbcac" , return true .
When s3 = "aadbbbaccc" , return false .
Return 0 / 1 (0 for false, 1 for true) for this problem
See Expected Output

Max Sum Without Adjacent Elements

Asked in: Epic systems

Suggest Edit | Bookmark

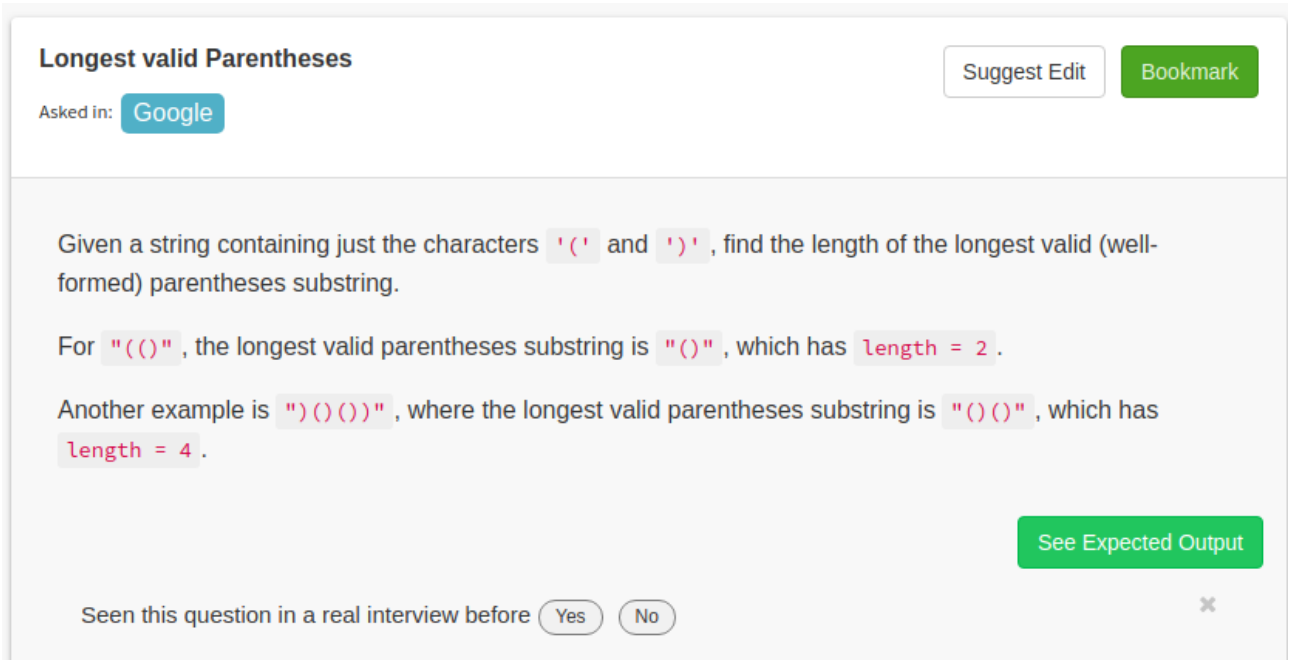
Given a `2 * N` Grid of numbers, choose numbers such that the sum of the numbers is maximum and no two chosen numbers are adjacent horizontally, vertically or diagonally, and return it.

Example:

```
Grid:
        1 2 3 4
        2 3 4 5
so we will choose
3 and 5 so sum will be 3 + 5 = 8
```

Note that you can choose more than 2 numbers

```
public class Solution {
    public int adjacent(ArrayList<ArrayList<Integer>> A) {
        int inclusive=Math.max(A.get(0).get(0),A.get(1).get(0));
        int exclusive=0;
        for(int i=1;i<A.get(0).size();i++){
            int new_excl=Math.max(inclusive,exclusive);
            inclusive=exclusive+Math.max(A.get(0).get(i),A.get(1).get(i));
            exclusive=new_excl;
        }
        return Math.max(inclusive,exclusive);
    }
}
```




```
public class Solution {
    public int longestValidParentheses(String A) {
        int result=0;
        Stack<Integer> st=new Stack<Integer>();
        st.push(-1);
        for(int i=0;i<A.length();i++){
            if(A.charAt(i)=='('){
                st.push(i);
            }
            else{
                st.pop();
                if(!st.isEmpty()){
                    result=Math.max(i-st.peek(),result);
                }
                else{
                    st.push(i);
                }

            }
        }
        return  result;
    }
}
```

Max Product Subarray

Asked in: Amazon LinkedIn Microsoft

Suggest Edit | Bookmark

Find the contiguous subarray within an array (containing at least one number) which has the largest product.
Return an integer corresponding to the maximum product possible.

Example :

```
Input : [2, 3, -2, 4]
Return : 6

Possible with [2, 3]
```

See Expected Output

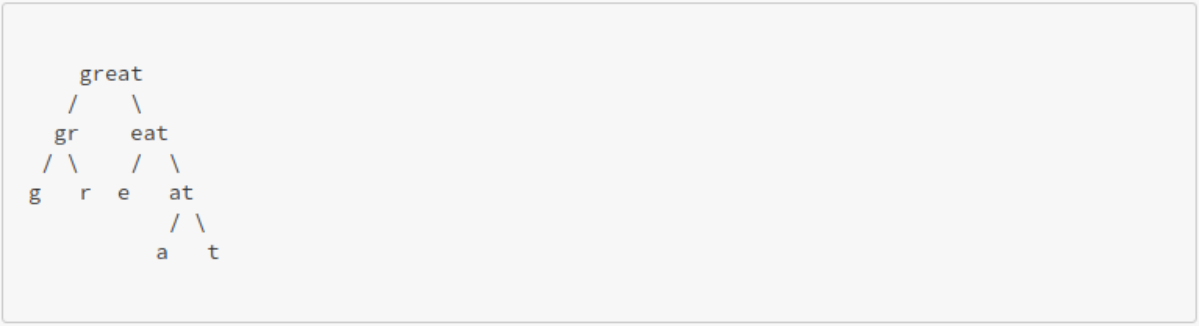
```
public class Solution {
    // DO NOT MODIFY THE LIST. IT IS READ ONLY
    public int maxProduct(final List<Integer> A) {
        int local_max=A.get(0);
        int local_min=A.get(0);
        int max_prod=A.get(0);

        for(int i=1;i<A.size();i++){
            if(A.get(i)>0){
                local_max=Math.max(A.get(i)*local_max,A.get(i));
                local_min=Math.min(A.get(i)*local_min,A.get(i));
            }
            else{
                int local_max_neg=Math.max(A.get(i)*local_min,A.get(i));
                local_min=Math.min(A.get(i)*local_max,A.get(i));
                local_max= local_max_neg;
            }
            max_prod=Math.max(local_max,max_prod);
        }
        return max_prod;
    }
}
```

Scramble String

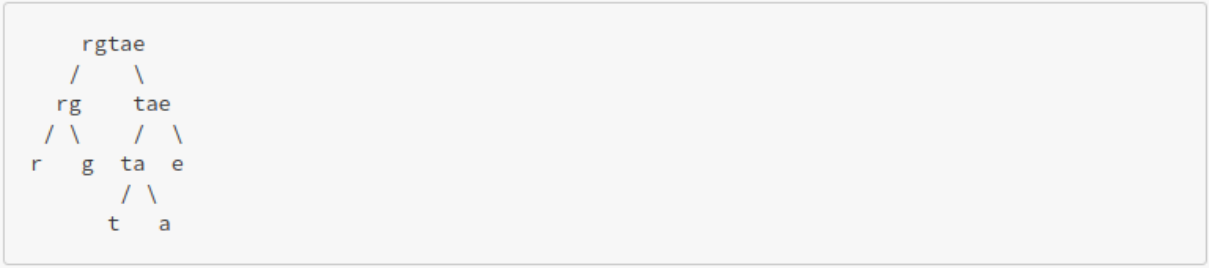
Suggest Edit | Bookmark

Given a string s1, we may represent it as a binary tree by partitioning it to two non-empty substrings recursively.

Below is one possible representation of s1 = “great”:



Similarly, if we continue to swap the children of nodes “eat” and “at”, it produces a scrambled string “rgtae”.



We say that “rgtae” is a scrambled string of “great”.

Given two strings s1 and s2 of the same length, determine if s2 is a scrambled string of s1. Return 0/1 for this problem.

```
public class Solution {
    // DO NOT MODIFY THE ARGUMENTS WITH "final" PREFIX. IT IS READ ONLY
    public int isScramble(final String s1, final String s2) {

        boolean flag=isScrambleString(s1,s2);
        if(flag){
            return 1;

        }
        return 0;

    }
    public static boolean isScrambleString(final String s1, final String s2){
      if(s1==null||s2==null||s1.length()!=s2.length())
            return false;
        if(s1.equals(s2))
```

```
            return true;
        char a1[],a2[];
        a1=s1.toCharArray();
        a2=s2.toCharArray();
        Arrays.sort(a1);Arrays.sort(a2);
        if(!(new String(a1).equals(new String(a2))) )
            return false;
        for(int i=1;i<s1.length();i++){
            if(isScrambleString(s1.substring(0,i),s2.substring(0,i)) &&
isScrambleString(s1.substring(i),s2.substring(i)) )
                return true;
            if(isScrambleString(s1.substring(0,i),s2.substring(s2.length()-i))
&&isScrambleString(s1.substring(i),s2.substring(0,s2.length()-i)) )
                return true;
        }
        return false;
    }
}
```

Palindrome Partitioning II

Asked in: Amazon Google

Suggest Edit | Bookmark

Given a string s, partition s such that every substring of the partition is a palindrome.

Return the minimum cuts needed for a palindrome partitioning of s.

Example :

Given

`s = "aab"`,

Return `1` since the palindrome partitioning `["aa","b"]` could be produced using `1` cut.

See Expected Output

Seen this question in a real interview before Yes No

```
public class Solution {
    public int minCut(String s) {
        int n=s.length();
        boolean[][] p=new boolean[n][n];
        int[][] c=new int[n][n];
        int i,j,k,l;
        for(i=0;i<n;i++){
            p[i][i]=true;
            c[i][i]=0;
        }
        for(l=2;l<=n;l++){
```

```
        for(i=0;i<n-l+1;i++){
            j=i+l-1;
            if(l==2){
                p[i][j]=(s.charAt(i)==s.charAt(j));
            }
            else{

                p[i][j]=(s.charAt(i)==s.charAt(j) && p[i+1][j-1]);
            }
            if(p[i][j]){
                c[i][j]=0;
            }
            else{
                c[i][j]=Integer.MAX_VALUE;
                for(k=i;k<=j-1;k++)
                {
                    c[i][j]=Math.min(c[i][j],c[i][k]+c[k+1][j]+1);
                }
            }
        }
    }
    return c[0][n-1];
  }
}
```

Arrange II

Asked in: Amazon

Suggest Edit | Bookmark

You are given a sequence of black and white horses, and a set of K stables numbered 1 to K. You have to accommodate the horses into the stables in such a way that the following **conditions are satisfied:**

> “
>
> - You fill the horses into the stables preserving the relative order of horses. For instance, you cannot put horse 1 into stable 2 and horse 2 into stable 1. You have to preserve the ordering of the horses.
> - No stable should be empty and no horse should be left unaccommodated.
> - Take the product (`number of white horses * number of black horses`) for each stable and take the sum of all these products. This value should be the minimum among all possible accommodation arrangements
>
> ”

Example:

```
Input: {WWWB} , K = 2
Output: 0

Explanation:
We have 3 choices {W, WWB}, {WW, WB}, {WWW, B}
for first choice we will get 1*0 + 2*1 = 2.
for second choice we will get 2*0 + 1*1 = 1.
for third choice we will get 3*0 + 0*1 = 0.

Of the 3 choices, the third choice is the best option.
```

If a solution is not possible, then return -1

```
public class Solution {
    public int arrange(String A, int B) {
        int l=A.length();
        if(l<B){
            return -1;
        }
        int[][] dp=new int[l][B];
        int wn=0;
        int bn=0;
        for(int i=0;i<l;i++){
            if(A.charAt(i)=='B'){
                bn++;
            }
            else{
```

```
            wn++;
          }
          dp[i][0]=bn*wn;
        }
        for(int i=1;i<B;i++){
          for(int j=0;j<l;j++){
            if(i>j){
              dp[j][i]=Integer.MAX_VALUE;
            }
            else{
              int mv=Integer.MAX_VALUE;
              wn=0;
              bn=0;
              for(int k=j-1;k>=0;k--){
                if(A.charAt(k+1)=='B'){
                  bn++;
                }
                else{
                  wn++;
                }
                if(dp[k][i-1]+bn*wn>=0){
                  mv=Math.min(mv,dp[k][i-1]+wn*bn);
                }
              }
              dp[j][i]=mv;
            }
          }
        }
        return (dp[l-1][B-1]>0?dp[l-1][B-1]:0);
    }
}
```