Follow up1: how to optimize space

```
// LastMax is an optimization to S[i], using O(1) space.
00 public int getMax(int[] s) {
01    if (s == null || s.length < 1) {
02          return 0;
03    }
04    int max = Integer.MIN_VALUE;
05    int lastMax = s[0];
06    for (int i = 1; i < s.length; i++) {
07          lastMax = Math.max(s[i],  s[i] + lastMax);
08          max = Math.max(max, lastMax);
09    }
10    return max;
11 }
```

FollowUp 2

What if you are **requested** to return the left and right index of the subarray that has the largest sum.

index 0 1 2 3 4 5 6

input {1 , 2, 4, -1, -12, 10 -1},

L

R

solu_L

solu_R

Question1: when to update solu_L = L (when max is updated)

when to update solu_R = R (when max is updated)

when to update L (L = i when M[i-1] < 0)

when to update R (update all the time each round)

Solution2: DP

String1 = n letters xxxxxxxxx n

String2 = m letters yyyyyyyyyyy m

M[n][m] = ????????? M[n-1][m] or M[n][m-1] or M[n-1][m-1]

xxxxxxxxx

xxxxxxxxx

xxxxYxxxx Y = [i][j] represents the minimum number of actions needed to convert

xxxxxxxxx the first i-letters of s1 to the first j letters of s2

xxxxxxxxx

xxxxxxxxx

xxxxxxxxX [n][m]

Solution:

1. Base case: M[0][0] = 0, and M[0][m] = m AND M[n][0] = n;

0 1 2 3 4 5 6 7
1 x x x xx
2
3

2. Induction rule:
   a. **M[i][j] represents the minimum number of actions to transform substring (the first i letters of s1 to the first j letters of s2.**
   b. **M[i][j] = case 1(do nothing) M[i-1][j-1]** if the i-th letter of s1 == the j-th letter of s2

      case 2 (replace) 1 + M(i-1, j-1)
      case 3 (delete) 1 + M(i-1, j)
      case 4 (insert) 1 + M(i, j-1)

      = min (case 1, case 2 , case3, case 4)

subproblem: M[i][j]: represents the max size of square with the coordinate[i][j] as its bottom right corner.

```
M[i][j] = 0                                          if  A[i][j] == 0;
        = 1 + min(M[i-1][j-1], M[i][j-1], M[i-1][j])  else
```

```
0 0 0 0 0
1 1 1 1 0
1 1 1 1 0
1 1 1 0 0
1 1 1 0 0
```

```
M[i][j]=
0 0 0 0 0
1 1 x x x
1 x x x x
1 x x x x
1 x x x x
```