کلید تکلیف چهارم ::

.1
تکه کد:

```
initial val[n + 1] = 0
for I from 0 to n:
        for j from 0 to I:
                maxVal = max(price[j] + val[i – j], maxVal)
        val[i] = maxVal
return val[n]
```

رابطه ی بازگشتی :

اگر بهترین برش ها و بیشترین ارزش‌هایی که ایجاد میکنند را برای تکه‌های به طول 0 تا $n - 1$ داشته باشیم، حال که یک تکه‌ای به طول n داریم میتوانیم از $n - 1$ جواب قبلی استفاده کنیم و بر روی تمامی حالات Max بگیریم که این حالات برابر زیر است :
$1 + (n – 1), 2 + (n – 2), 3 + (n – 3), \ldots , n + (0)$
که قسمت اول جمع مقدار price است که توسط سؤال داده شده و قسمت دوم جمع مقادیر value است که قبلاً محاسبه کرده ایم.

روش حل از مرجع :

A naive solution for this problem is to generate all configurations of different pieces and find the highest priced configuration. This solution is exponential in term of time complexity. Let us see how this problem possesses both important properties of a Dynamic Programming (DP) Problem and can efficiently solved using Dynamic Programming.

1) Optimal Substructure:
We can get the best price by making a cut at different positions and comparing the values obtained after a cut. We can recursively call the same function for a piece obtained after a cut.
Let cutRod(n) be the required (best possible price) value for a rod of length n. cutRod(n) can be written as following.

cutRod(n) = max(price[i] + cutRod(n-i-1)) for all i in {0, 1 .. n-1}

2) Overlapping Subproblems
Following is simple recursive implementation of the Rod Cutting problem. The implementation simply follows the recursive structure mentioned above.

توضیحات تصحیح: نوشتن تکه کد الزامی نیست ولی اگه تکه کد را صحیح نوشته بودند احتمالاً توضیحاتشون هم درسته
اگه تکه کد نوشته نشده بود باید توی رابطه ی بازگشتی و یا توضیحات مسأله به طریقه ی حل اشاره شده باشه

.2

تکه کد:

```
int max = -inf, end = 0
for I from 0 to n
        end = end + array[i]
        if end < 0
                end = 0
        if end > max
                max = end
return max
```

رابطه ی بازگشتی :
از ابتدای آرایه شروع میکنیم به جمع زدن عناصر متوالی. اگر عددی مثبت بود مسلماً میتوانیم به مجموعه ی قبلی اضافه کنیم و یک مجموع بزرگ‌تر بسازیم. ولی اگر عدد منفی بود باید ببینیم که آن را باید اضافه کنیم یا خیر. اگر با اضافه کردن آن عدد به مجموع، کل مجموع عددی منفی شود نباید آن را اضافه کنیم زیرا یک جمع متوالی بزرگ‌تر بدون حضور این عدد داریم. ولی اگر با اضافه کردن آن به مجموع، جمع مجموع منفی نشد میتوانیم آن را به مجموعه اضافه کنیم.

توضیحات مرجع :
Simple idea of the Kadane’s algorithm is to look for all positive contiguous segments of the array (max_ending_here is used for this). And keep track of maximum sum contiguous segment among all positive segments (max_so_far is used for this). Each time we get a positive sum compare it with max_so_far and update max_so_far if it is greater than max_so_far

توضیحات تصحیح: نوشتن تکه کد الزامی نیست ولی اگه تکه کد را صحیح نوشته بودند احتمالاً توضیحاتشون هم درسته
اگه تکه کد نوشته نشده بود باید توی رابطه ی بازگشتی و یا توضیحات مسأله به طریقه ی حل اشاره شده باشه

.3

راه حل اول : از طریق ترکیبیات میتوان دید که نیاز به شماردن تعداد جایگشت های m حرف D و n حرف L داریم که این برابر است با :

$(m + n)! / m!n!$

راه حل دوم :
به ازای هر خانه، کافی است تعداد حالت‌هایی که از خانه ی سمت راست خود و تعداد حالت‌هایی که از خانه ی سمت بالای خود قبلاً به دست آمده را با یکدیگر جمع کنیم
یعنی

$table[x][y] = table[x + 1][y] + table[x][y + 1]$

توضیحات مرجع :

```
countPaths(n, m)
{
    // If we reach bottom or top left, we are
    // have only one way to reach (0, 0)
    if (n==0 || m==0)
        return 1;

    // Else count sum of both ways
    return (countPaths(n-1, m) + countPaths(n, m-1));
}
```

توضیحات تصحیح: نوشتن تکه کد الزامی نیست ولی اگه تکه کد را صحیح نوشته بودند احتمالاً توضیحاتشون هم درسته
اگه تکه کد نوشته نشده بود باید توی رابطه ی بازگشتی و یا توضیحات مسأله به طریقه ی حل اشاره شده باشه
اشاره به هر کدام از راه حل‌های اول یا دوم کافی است
اگر از طریق راه حل دوم نوشته شده بود بین $m - 1$ یا $m + 1$ تفاوتی وجود ندارد و بستگی به این دارد که جهت های مختصاتی جدول را به چه شکل در نظر گرفته باشند.

.4
رابطه ی بازگشتی :

مقدار هر خانه ی ماتریس که معادل بزرگترین زیرماتریس مربعی است که این خانه نقطه ی انتهایی آن باشد از رابطه ی بازگشتی زیر به دست می‌آید

```
if(table[x][y] == 1)
        s[x][y] = min(s[x – 1][y], s[x][y – 1], s[x – 1][y – 1]) + 1
else
        s[x][y] = 0
```

و نهایتاً بزرگترین مقدار s[x][y] را به ازای تمام مقادیر x و y پیدا میکنیم

توضیحات مرجع :

```
1) Construct a sum matrix S[R][C] for the given M[R][C].
     a)    Copy first row and first columns as it is from M[][] to S[][]
     b)    For other entries, use following expressions to construct S[][]
         If M[i][j] is 1 then
            S[i][j] = min(S[i][j-1], S[i-1][j], S[i-1][j-1]) + 1
         Else /*If M[i][j] is 0*/
            S[i][j] = 0
2) Find the maximum entry in S[R][C]
3) Using the value and coordinates of maximum entry in S[i], print
   sub-matrix of M[][]
```

توضیحات تصحیح: نوشتن تکه کد الزامی نیست ولی اگه تکه کد را صحیح نوشته بودند احتمالاً توضیحاتشون هم درسته
اگه تکه کد نوشته نشده بود باید توی رابطه ی بازگشتی و یا توضیحات مسأله به طریقه ی حل اشاره شده باشه
در نظر گرفتن دو حالتی که مقدار table[x][y] برابر 0 باشه یا برابر 1 باشه مهم ه
نوشتن رابطه ی s[x][y] = min(s[x – 1][y], s[x][y – 1], s[x – 1][y – 1]) + 1 کافی است

.5

این مسأله ی بزرگترین زیر رشته ی palindrome است.
راه حل بازگشتی:
هر بار از رشته ی اصلی زیر رشته ای که از I تا j است را انتخاب میکنیم و بررسی میکنیم. اگر کاراکتر I ام با کاراکتر j ام برابر بود بزرگترین زیر رشته متقارن از I تا j برابر است با

$$s[i][j] = s[i + 1][j - 1] + 2$$

در غیر این صورت (اگر دو کاراکتر برابر نبود)

$$s[i][j] = \max(s[i][j - 1], s[i + 1][j])$$

مقدار دهی اولیه:
هر کاراکتر خود یک زیررشته ی متقارن به طول 1 است
(هر دو کاراکتر متوالی در صورت برابری یک زیررشته ی متقارن به طول 2 است)

راه حل مرجع :
The naive solution for this problem is to generate all subsequences of the given sequence and find the longest palindromic subsequence. This solution is exponential in term of time complexity. Let us see how this problem possesses both important properties of a Dynamic Programming (DP) Problem and can efficiently solved using Dynamic Programming.

1) Optimal Substructure:
Let X[0..n-1] be the input sequence of length n and L(0, n-1) be the length of the longest palindromic subsequence of X[0..n-1].
If last and first characters of X are same, then L(0, n-1) = L(1, n-2) + 2.
Else L(0, n-1) = MAX (L(1, n-1), L(0, n-2)).

توضیحات تصحیح: نوشتن تکه کد الزامی نیست ولی اگه تکه کد را صحیح نوشته بودند احتمالاً توضیحاتشون هم درسته
اگه تکه کد نوشته نشده بود باید توی رابطه ی بازگشتی و یا توضیحات مسأله به طریقه ی حل اشاره شده باشه
مقدار دهی های اولیه باید به درستی انجام شده باشد (ذکر تنها اولی کافی است)

6. (اختیاری 1)

راه حل :
اگر به ازای هر عضو بتوانیم زیردنباله ی متوالی که در آن این عضو کوچکترین عضو آن زیر دنباله است را به دست بیاوریم مسأله به راحتی قابل حل است (در این حالت حاصل این زیردنباله برابر مقدار آن عضو در طول زیر دنباله است. حال کافی است به ازای تمام زیر دنباله ها که تعداد آن‌ها n تا است این مقدار را محاسبه کرده و max بگیریم)
برای این کار به شکلی که در زیر گفته شده است از stack استفاده میکنیم.

حل مرجع :
We traverse all bars from left to right, maintain a stack of bars. Every bar is pushed to stack once. A bar is popped from stack when a bar of smaller height is seen. When a bar is popped, we calculate the area with the popped bar as smallest bar. How do we get left and right indexes of the popped bar – the current index tells us the 'right index' and index of previous item in stack is the 'left index'. Following is the complete algorithm.

1)Create an empty stack.
2)Start from first bar, and do following for every bar 'hist[i]' where 'i' varies from 0 to n-1.
……a)If stack is empty or hist[i] is higher than the bar at top of stack, then push 'i' to stack.
……b)If this bar is smaller than the top of stack, then keep removing the top of stack while top of the stack is greater. Let the removed bar be hist[tp]. Calculate area of rectangle with hist[tp] as smallest bar. For hist[tp], the 'left index' is previous (previous to tp) item in stack and 'right index' is 'i' (current index).
3)If the stack is not empty, then one by one remove all bars from stack and do step 2.b for every removed bar.
(در این مسأله به جای bar مقادیر عددی آن‌ها در یک آرایه داده شده است)

توضیحات تصحیح: نوشتن تکه کد الزامی نیست ولی اگه تکه کد را صحیح نوشته بودند احتمالاً توضیحاتشون هم درسته
اگه تکه کد نوشته نشده بود باید توی رابطه ی بازگشتی و یا توضیحات مسأله به طریقه ی حل اشاره شده باشه

7. (اختیاری 2)

به ازای هر عضو در سطر k اگر آن عضو 1 بود، تعداد 1 هایی که بالای سر آن قرار دارد را ابتدا به دست می آوریم. رابطه ی بازگشتی آن به شکل زیر است

```
if(table[k][x])
        s[k][x] = s[k – 1][x] + 1
else
        s[k][x] = 0
```

حال در سطر k ام یک دنباله ی عددی به شکل bar داریم که میخواهیم max histogram area آن را حساب کنیم که برای این کار از الگوریتم سؤال قبل استفاده میکنیم
نهایتاً بین مقادیر به دست آمده در تمامی سطر ها max میگیریم

توضیحات تصحیح: نوشتن تکه کد الزامی نیست ولی اگه تکه کد را صحیح نوشته بودند احتمالاً توضیحاتشون هم درسته
اگه تکه کد نوشته نشده بود باید توی رابطه ی بازگشتی و یا توضیحات مسأله به طریقه ی حل اشاره شده باشه