پاسخ تمرین ۵

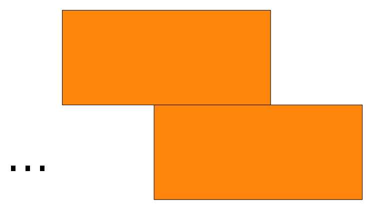
۱ - فرض می‌کنیم که تعداد راه‌های چیدن n آجر با شرایط مطرح شده در سؤال f_n باشد. حال سمت راست ترین آجری که روی آن آجر دیگری نباشد و با برداشتن سمت راست‌ترین آجر لایه اول فرو می‌ریزد را در نظر می‌گیریم و روی آن حالت بندی می‌کنیم:

حالت اول: این آجر روی زمین باشد:

در این حالت با برداشتن این آجر، باقی آجر ها به f_{n-1} طریق قابل چیده شدن هستند.

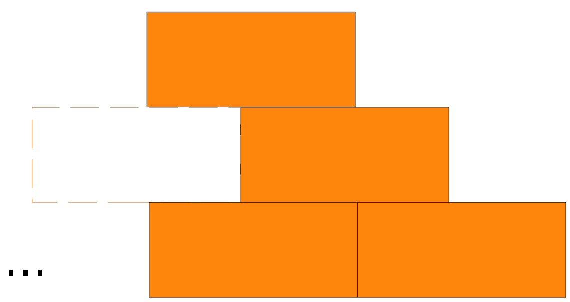
حالت دوم: زیر این آجر فقط یک آجر دیگر قرار داشته باشد:

در این حالت با برداشتن این ۲ آجر، باقی آجر ها به f_{n-2} طریق قابل چیده شدن هستند.



حالت سوم: زیر این آجر ۲ آجر دیگر هم قرار داشته باشند:

در این حالت، با برداشتن این ۳ آجر باقی آجر ها به f_{n-3} طریق قابل چیده شدن خواهند بود ولی نکته‌ای که باید به آن دقت کنیم این است که اگر در قسمتی که با نقطه‌چین نمایش داده شده است، آجری نباشد، این آجرچینی معتبر نیست. پس بایستی حالاتی که در قسمت نقطه‌چین آجری نیست را از f_{n-3} کم کنیم که این حالات f_{n-4} تایند.



پس رابطه بازگشتی برای این سؤال به صورت $f_n = f_{n-1} + f_{n-2} + f_{n-3} - f_{n-4}$ خواهد بود. همچنین پایه ها به شکل زیرند:
$f_0 = f_1 = f_2 = 1, f_3 = 2$

پس با داشتن پایه‌ها و رابطه بازگشتی کافیست یک for برای پر کردن جدول dp بنویسیم. ($dp[n] = f_n$)

از آن جا که با یک حلقه for می‌توان پاسخ را به دست آورد، پس پیچیدگی زمانی چنین الگوریتمی $O(n)$ خواهد بود. کد مربوط به این الگوریتم در صفحه بعد آمده است:

```python
n = int(input())

dp = [1 for i in range(n)]

dp[2] = 2

for i in range(3, n) :
    dp[i] = dp[i - 1] + dp[i - 2] + dp[i - 3] - dp[i - 4]

print(dp[n - 1])
```

۲-

آ) dp را $n*(s+1)$ در نظر می‌گیریم و در ادامه $dp[i][j]$ را به این شکل تعریف می‌کنیم که با اختیار انتخاب از بین i سکه اول، برای خرد کردن مقدار j پول، حداقل به چند سکه نیاز داریم. حال ۲ حالت خواهیم داشت:

حالت اول: سکه i جزو انتخاب های‌مان نباشد. در این صورت $dp[i][j]$ همان $dp[i-1][j]$ خواهد بود.

حالت دوم: سکه i جزو انتخاب‌های‌مان باشد. در این صورت چون باز هم می‌توانیم سکه i را انتخاب نماییم (تعداد سکه‌ها نامحدود است) پس خواهیم داشت:

$$dp[i][j]=dp[i][j-value\,of\,coins[i]]+1$$

پس چون کم‌ترین تعداد سکه را می‌خواهیم کافیست بین این دو حالت $minimum$ بگیریم:

$$dp[i][j]=min(dp[i-1][j],dp[i][j-value\,of\,coins[i]]+1)$$

پایه‌های dp هم به این صورت تعریف می‌شوند که اگر $s=0$ باشد، پس صفر سکه نیاز خواهد بود پس:

$$dp[i][0]=0\,\forall\,0\leq i\leq n-1$$

همچنین می‌توان نوشت:

$$dp[i][value\,of\,coin[i]]=1\,\forall\,0\leq i\leq n-1$$

و برای سطر اول هم اگر j به مقدار اولین سکه بخش پذیر باشد، در خانه نظیرش، مقسوم علیه مقدار سکه اول به j را قرار می‌دهیم:

$$dp[0][j]=j\div value\,of\,coin[0]\,if\,j\,\%\,coin[0]=0\,\forall\,1\leq j\leq s$$

کد متناظر با این سؤال به صورت زیر خواهد بود:

```python
from math import inf

n, s = map(int, input().split())
coins = list(map(int, input().split()))

dp = [[inf for j in range(s + 1)] for i in range(n)]

for i in range(n):
    dp[i][0] = 0

for i in range(n):
    dp[i][coins[i]] = 1

for j in range(1, s + 1):
    if j % coins[0] == 0:
        dp[0][j] = j // coins[0]

for i in range(1, n):
    for j in range(1, s + 1):
        if j >= coins[i]:
            dp[i][j] = min(dp[i][j - coins[i]] + 1, dp[i - 1][j])
        else:
            dp[i][j] = dp[i - 1][j]

if dp[n - 1][s] == inf:
    print('Changing money using these coins is impossible')
else:
    print(f'Minimum coins needed to change money is: {dp[n - 1][s]}')
```

ب) خیر چند جمله‌ای نیست. چرا که در هر صورت بایستی بین تمام مقادیر و سکه‌ها پیمایش نماییم که ذاتا نمایی است. در واقع s می‌تواند از مرتبه n^m باشد یا برعکس (n هم می‌تواند از مرتبه s^t باشد) که در هر صورت، مرتبه زمانی به شکل نمایی خواهد شد و نه چند جمله‌ای.

۳-

آ) dp را $n*n$ در نظر می‌گیریم. در ادامه $dp[i][j]$ را برابر با مجموع اعداد یک زیر بازه از اندیس i تا اندیس j در نظر می‌گیریم. حال اگر $dp[i][j-1]$ را داشته باشیم، خواهیم داشت:

$$dp[i][j]=dp[i][j-1]+number[j]$$

پایه‌های dp را به شکل زیر پر می‌کنیم:

$$dp[i][i]=number[i]\ \forall\ 0\leq i<n$$

هم‌چنین بیشترین مجموع اعداد زیربازه‌ها را در ابتدا با اولین عدد مقداردهی می‌کنیم. در ادامه کافیست dp را پر نماییم و در حین پر کردن هر خانه آن، آن خانه را با بیشترین مجموع زیربازه‌ها مقایسه کنیم و اگر آن خانه از بیشترین مجموع زیربازه‌ها بیشتر بود، بیشترین مجموع زیربازه‌ها را به روز نماییم:

```
n = int(input())
nums = list(map(int, input().split()))

dp = [[None for j in range(n)] for i in range(n)]

for i in range(n):
    dp[i][i] = nums[i]

max_sub = nums[0]

for i in range(n):
    for j in range(i + 1, n):
        dp[i][j] = dp[i][j - 1] + nums[j]
        if dp[i][j] > max_sub:
            max_sub = dp[i][j]

print(max_sub)
```

ب) فرض کنید $maximumSum_i$ برابر با بزرگ‌ترین زیربازه‌ای باشد که به عنصر i ختم می‌شود. اگر چنین چیزی برای هر i از ۱ تا n مقداردهی شده باشد جواب مسئله برابر بیشینه آن‌ها خواهد بود چراکه بازه جواب یا به عنصر ۱ ختم می‌شود یا به عنصر ۲ یا … یا به عنصر n . حال برای محاسبه $maximumSum_i$ دو حالت خواهیم داشت:

حالت اول: $maximumSum_i$ برابر با خود عنصر i ام ($number[i]$) است.

حالت دوم: $maximumSum_i$ شامل $number_{i-1}$ هم می‌شود. که در این صورت بایستی تا جایی عقب برویم که مجموع آن بیشینه باشد. به عبارتی می‌خواهیم حاصل $number_i + \sum_{k=j}^{k=i-1} number_k$ بیشینه شود. مقدار $number_i$ که ثابت است. پس بایستی $\sum_{k=j}^{k=i-1} number_k$ بیشینه شود که این همان تعریف $maximumSum_{i-1}$ است. پس می‌توان نوشت:

$$maximumSum_i = max(number_i, number_i + maximumSum_{i-1})$$

در نهایت کافیست بین عناصر dp ماکسیمم بگیریم. (چون نمی‌دانیم بزرگ‌ترین زیربازه به کدام عنصر ختم خواهد شد.)

به عنوان پایه dp هم کافیست عنصر اول آن را با اولین عددی که داریم، مقدار دهی نماییم. ($dp[0] = number[0]$)

```python
n = int(input())
nums = list(map(int, input().split()))

dp = [None for i in range(n)]

dp[0] = nums[0]

for i in range(1, n):
    dp[i] = max(nums[i], nums[i] + dp[i - 1])

print(max(dp))
```

٤– ابتدا یک جدول $n \times n$ به نام S تعریف می‌کنیم. در این جدول $S[i][j]$ برابر است با مجموع تمام عناصر زیر مستطیلی که مختصات گوشه سمت چپ و بالای آن $(0,0)$ و مختصات گوشه سمت راست و پایین آن (i,j) است.

حال می‌خواهیم برای تمام زیر مستطیل‌های موجود بین سطر i و خود سطر j ، مجموع تمام عناصرشان را محاسبه نماییم. برای این کار بین تمام سطرهای مستطیل از i تا خود j پیمایش می‌کنیم و به ازای هر جفت i و j ای، هر

خانه از آرایه یک بعدی R را به ازای هر ستون مانند k با کم کردن $S[i-1][k]$ از $S[j][k]$ به دست می‌آوریم.

حال کافیست از الگوریتمی که در سؤال قبلی (سؤال سوم) برای محاسبه بیشترین مجموع زیربازه‌ها ارائه کردیم، استفاده نماییم و این الگوریتم را روی R اعمال نماییم و با استفاده از آن بیشترین مجموعی که تا کنون را به دست آوردیم را به روز نماییم.

این الگوریتم را می‌توان با نگه‌داری تنها یک سطر از S در هر مرحله از لحاظ حافظه بهینه تر هم کرد که کد بهینه شده آن در ادامه آمده است:

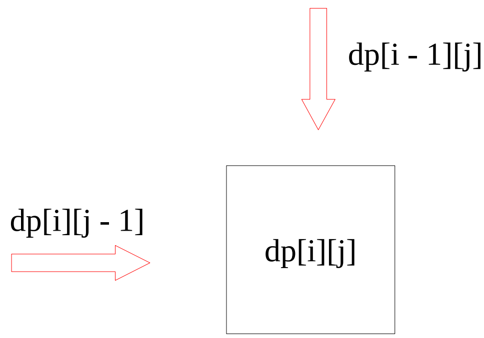
```python
def max_sub_rectangle(matrix):
    n = len(matrix)
    max_sum = float('-inf')
    top, left, bottom, right = None, None, None, None

    for i in range(n):
        temp = [0] * n
        for j in range(i, n):
            for k in range(n):
                temp[k] += matrix[j][k]

            # Find the maximum subarray sum of temp
            curr_sum = 0
            curr_left = 0
            for l in range(n):
                curr_sum += temp[l]
                # Update info
                if curr_sum > max_sum:
                    max_sum = curr_sum
                    top = i
                    left = curr_left
                    bottom = j
                    right = l
                if curr_sum < 0:
                    curr_sum = 0
                    curr_left = l + 1

    return (max_sum, top, left, bottom, right)
```

۵- جدول dp را $m \times n$ در نظر می‌گیریم. حال از آنجا که تعداد دقیق صفر و یک‌های موجود در یک خانه را می‌دانیم، ارزش یک خانه را با کم کردن تعداد صفر های آن خانه از تعداد یک های آن به دست می‌آوریم و این کار برای هر خانه‌ای که به آن برسیم، در $O(1)$ ممکن است. حال هدف مسأله این است که در انتها از ارزشمند ترین خانه‌ها عبور کرده باشیم تا بتوانیم بیشترین گالیون طلایی ممکن را دریافت نماییم. حال $dp[i][j]$ را برابر با بیشترین ارزشی که می‌توانیم به دست آوریم، در نظر می گیریم. از آنجا که به هر خانه یا از سمت بالا یا از سمت راست آمده‌ایم، رابطه dp به شکل زیر خواهد شد:



$$dp[i][j]=max(dp[i][j-1], dp[i-1][j])+valueOfTable[i][j]$$

پایه dp، $dp[0][0]$ است که کافیست آن را برابر با تفاضل تعداد صفرهای خانه اول جدول از تعداد یک‌های خانه اول جدول قرار دهیم.

شبه کد این سؤال به شکل زیر خواهد بود:

```
dp = array(m, n)

dp[0][0] = table[0][0].onesCount - table[0][0].zerosCount

for (j = 1; j < n; j++) do
    dp[0][j] = dp[0][j - 1] + table[0][j].onesCount - table[0][j].zerosCount
end

for (i = 1; i < m; i++) do
    dp[i][0] = dp[i - 1][0] + table[i][0].onesCount - table[i][0].zerosCount
end

for (i = 1; i < m; i++) do
    for (j = 1; j < n; j++) do
        dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]) + table[i][j].onesCount - table[i][j].zerosCount
    end
end

print(dp[m - 1][n - 1])
```

پیچیدگی زمانی الگوریتم هم $O(m\times n)$ است. چرا که بایستی جدول dp را پر کنیم.

۶- فرض می‌کنیم در سال تحصیلی j ام، n_j تا درس ارائه شده باشد. در ابتدا برای هر سال تحصیلی کم‌هزینه‌ترین درس و مجموع هزینه تمام دروس را به دست می‌آوریم. (انجام این کار برای هر سال تحصیلی در $O(n_j)$ امکان‌پذیر است و چون N سال تحصیلی داریم، هزینه کل آن $n_{max}\times N$ خواهد شد.)

در ادامه یک dp با ابعاد $(N+1)\times(S+1)$ تعریف می‌کنیم و $dp[i][j]$ برابر است بیشترین تعداد درسی که می‌توان با داشتن j پوند پول برداشت اگر قرار باشد i سال در هاگوارتز تحصیل کرد. پس پاسخ مسأله $dp[N][S]$ خواهد بود.

برای پر کردن $dp[i][j]$ ۲ حالت داریم:

حالت اول: از یک سال تحصیلی هیچ درسی برنداریم که در این صورت داریم:

$$dp[i][j]=dp[i-1][j]$$

حالت دوم: خودش شامل ۲ حالت خواهد شد:

- از یک سال تحصیلی کم‌هزینه‌ترین درس را برداریم:

$$dp[i][j]=dp[i-1][j-cheapestCourseCost]+1$$

- تمام دروس ارائه شده در آن سال تحصیلی را برداریم:

$$dp[i][j]=dp[i-1][j-totalCoursesCost]+n_j$$

پس بایستی بین این ۳ حالت ماکسیمم بگیریم.

پایه‌های dp به این شکل تعریف می‌شوند:

$$dp[i][0]=0\ \forall\ 0\le i\le N$$

$$dp[0][j]=0\ \forall\ 0\le j\le S$$

شبه کد این سؤال در صفحه بعد آمده است. پیچیدگی زمانی این الگوریتم هم از مرتبه $max(n_{max}\times N,(N+1)\times(S+1))$ خواهد بود.

```
dp = array(N + 1, S + 1)

totalCostOfYear = array(N + 1)
minimumCostOfYear = array(N + 1)

for (i = 0; i < N + 1; i++) do
    totalCost, minimumCost = 0, inf
    for (j = 0; j < year[i].coursesCount; j++) do
        totalCost += year[i].courses[j].cost
        if year[i].courses[j].cost < minimumCost do
            minimumCost = year[i].courses[j].cost
        end
    end
    totalCostOfYear.append(totalCost)
    minimumCostOfYear.append(minimumCost)
end

for (i = 0; i < N + 1; i++) do
    dp[i][0] = 0
end

for (j = 0; i < S + 1; j++) do
    dp[0][j] = 0
end

for (i = 1; i < N + 1; i++) do
    for (j = 1; j < S + 1; j++) do
        if minimumCostOfYear[i] > j do
            dp[i][j] = dp[i - 1][j]
        end
        else if minimumCostOfYear[i] =< j < totalCostOfYear[i] do
            dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - minimumCostOfYear[i]] + 1)
        end
        else do
            dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - minimumCostOfYear[i]] + 1,
                                        dp[i - 1][j - totalCostOfYear[i]] + year[i].coursesCount)
        end
    end
end

print(dp[N + 1][S + 1])
```

۷- آ) تعریف: می‌گوییم کلمه s زیردنباله کلمه p است، اگر بتوان با حذف تعدادی از کاراکترهای p به s رسید. اگر طول رشته اول برابر با m و طول رشته دوم برابر با n باشد، dp را به شکل یک جدول $(m+1)\times(n+1)$ در نظر می‌گیریم. حال فرض کنید i کاراکتر اول یک رشته مانند s به شکل $s_{[1...i]}$ نشان داده شود. در این صورت

$dp[i][j]$ را برابر با بزرگ‌ترین زیررشته مشترک $s_{[1...i]}$ و $p_{[1...i]}$ تعریف می‌کنیم (s و p به ترتیب همان رشته‌های اول و دومند). حال برای محاسبه $dp[i][j]$ دو حالت خواهیم داشت:

حالت اول: $s_i \neq p_j$ در این حالت چون یا s_i یا p_j در بزرگ‌ترین زیررشته مشترک حضور نخواهند داشت، خواهیم داشت:

$$dp[i][j]=max(dp[i-1][j], dp[i][j-1])$$

حالت دوم: $s_i = p_j$ در این حالت ممکن است هر دوی s_i و p_j در بزرگ‌ترین زیررشته مشترک حضور داشته باشند یا این که مانند حالت اول فقط یکی از آن‌ها در بزرگ‌ترین زیررشته مشترک حضور داشته باشد که در این صورت خواهیم داشت:

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

پایه‌های dp به شکل زیر تعریف می‌شوند:

$$dp[0][j]=0 \ \forall\, 0 \leq j \leq n$$

$$dp[i][0]=0 \ \forall\, 0 \leq i \leq m$$

حال کافیست جدول dp را پر نماییم. جواب مسأله همان $dp[m][n]$ خواهد بود. شبه کد این سؤال در صفحه بعد آورده شده است و ایم الگوریتم از مرتبه $O(m \times n)$ خواهد بود.

```
// s, p are first and seccond input strings
m = len(s)
n = len(p)

dp = array(m + 1, n + 1)

for (j = 0; j < n + 1; j++) do
    dp[0][j] = 0
end

for (i = 0; i < m + 1; i++) do
    dp[i][0] = 0
end

for (i = 1; i < m + 1; i++) do
    for (j = 1; j < n + 1; j++) do
        if s[i] != p[j] do
            dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])
        end
        else do
            dp[i][j] = max(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1] + 1)
        end
    end
end

print(dp[m][n])
```

ب) برای خروجی دادن جواب بزرگ‌ترین زیررشته مشترک کافیست روند بروز رسانی dp را ذخیره نماییم. یعنی برای هر خانه dp نگه داریم که از کدام خانه به روز شده است و از روی آن می‌توان فهمید که کاراکتر i ام در بزرگ‌ترین زیررشته مشترک آمده یا خیر. کد این سؤال در صفحه بعد آمده است.مرتبه زمانی الگوریتم همانند قسمت قبل $O(m\times n)$ است.

```python
p = input()
s = input()

m = len(p)
n = len(s)

dp = [[ ['', 0] for j in range(m) ] for i in range(n)]

if p[0] == s[0] :
    dp[0][0][0] = s[0]
    dp[0][0][1] += 1

for j in range(1, m) :
    if p[j] == s[0] :
        dp[0][j][0] += s[0]
        dp[0][j][1] += 1
    else :
        dp[0][j] = dp[0][j - 1]

for i in range(1, n) :
    if p[0] == s[i] :
        dp[i][0][0] += s[i]
        dp[i][0][1] += 1
    else :
        dp[i][0] = dp[i - 1][0]

for i in range(1, n) :
    for j in range(1, m) :
        if s[i] != p[j] :
            if dp[i - 1][j][1] > dp[i][j - 1][1] :
                dp[i][j] = dp[i - 1][j]
            else :
                dp[i][j] = dp[i][j - 1]
        else :
            tmp = max(dp[i - 1][j][1], dp[i][j - 1][1], dp[i - 1][j - 1][1] + 1)
            if tmp == dp[i - 1][j - 1][1] + 1 :
                dp[i][j][0] = dp[i - 1][j - 1][0] + s[i]
                dp[i][j][1] = tmp
            elif tmp == dp[i][j - 1][1] :
                dp[i][j] = dp[i][j - 1]
            else :
                dp[i][j] = dp[i - 1][j]

print(dp[n - 1][m - 1][1])
print(dp[n - 1][m - 1][0])
```

۸- از آنجا که هر شخصی به جز رئیس کل وزارت جادو یک رئیس دارد، می‌توان رابطه رئیس — زیردست را به شکل یک درخت در نظر گرفت که رئیس کل وزارت جادو در ریشه آن قرار دارد. فرض کنید میزان صمیمی بودن یک شخص مانند p را به شکل $cordiality(p)$ نشان دهیم. حال برای هر شخص مانند p دو مؤلفه زیر را تعریف می‌کنیم:

$T[p]$: بیشترین میزان صمیمی بودن زیر درختی که شخص p ریشه آن است و خود شخص p هم در مهمانی شرکت می‌کند.

$F[p]$: بیشترین میزان صمیمی بودن زیر درختی که شخص p ریشه آن است ولی خود شخص p در مهمانی شرکت نمی‌کند.

در این صورت پاسخ نهایی مسأله، به شکل زیر است:

$$max(F(\text{The head of the Ministry of Magic}), T(\text{The head of the Ministry of Magic}))$$

حال برای پر کردن هر یک از مؤلفه‌های T و F برای هر شخص p از روابط زیر استفاده می‌کنیم:

$$T[p] = cordiality(p) + \sum_{i=1}^{i=k} F[A_k]$$

$$F[p] = \sum_{i=1}^{i=k} max(T[A_k], F[A_k])$$

که هر یک از A_i ها فرزندان مستقیم شخص p در درخت رابطه رئیس — زیردست هستند. از آنجا که این مقادیر برای هر گره از درخت یک بار محاسبه می‌شوند، این الگوریتم از مرتبه $O(n)$ خواهد بود.هم‌چنین برای محاسبه مقادیر T و F یک رأس، در ابتدا باید مقادیر T و F فرزندان آن رأس محاسبه شده باشند. که برای انجام چنین کاری می‌توان از پیمایش DFS روی درخت استفاده کرد که شبه کد آن به شکل زیر است:

```
function DFS(node p) do
    if p is leaf do
        F[p] = 0
        T[p] = cordiality(p);
    end
    else
        for each (v child of p) do
            DFS(v)
        end
        T[p] = cordiality(p) + sum(F[v] for each (v child of p))
        F[p] = sum(max(T[p], F[p]) for each (v child of p))
    end
```

۹- آ) اگر یک ماتریس به شکل $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ در نظر بگیریم، آن گاه می‌توان گفت توان n ام چنین ماتریسی برابر است با:

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$$

یعنی می‌توان نوشت:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$$

پس کافیست محاسبه توان n ام ماتریس $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ را در $O(logn)$ انجام دهیم که برای چنین چیزی کافیست از روش $memoization$ استفاده نماییم. به این صورت که یک لغت‌نامه از توان به ماتریس در نظر می‌گیریم و آن را از پایین به بالا پر می‌کنیم. به این صورت که برای محاسبه توان p اگر p در کلیدهای لغت‌نامه وجود داشت که مقدار آن کلید را به عنوان پاسخ برمی‌گردانیم. در غیر این صورت اگر p زوج باشد، ماتریس توان $p/2$ را محاسبه نموده و در خودش ضرب می‌کنیم و اگر p فرد باشد ماتریس توان $(p-1)/2$ محاسبه کرده، را در خودش ضرب می‌کنیم و در خود ماتریس هم ضرب می‌کنیم. به عنوان پایه هم اگر p ، ۱ باشد خود ماتریس را برمی‌گردانیم. در انتها هم جواب را در $memoization[p]$ ذخیره می‌کنیم. چون در هر مرحله توان ماتریس نصف می‌شود، پس مرتبه زمانی این الگوریتم $O(logn)$ خواهد بود (دقت شود با توجه به تابعی که برای محاسبه ضرب دو ماتریس ۲×۲ نوشته‌ایم، هزینه ضرب ماتریس‌ها $O(1)$ خواهد بود). در ادامه شبه کد این الگوریتم آورده شده است.

```
memoization = dict

function multiply_matrices(M1, M2) do
    a11 = M1[0][0]*M2[0][0] + M1[0][1]*M2[1][0]
    a12 = M1[0][0]*M2[0][1] + M1[0][1]*M2[1][1]
    a21 = M1[1][0]*M2[0][0] + M1[1][1]*M2[1][0]
    a22 = M1[1][0]*M2[0][1] + M1[1][1]*M2[1][1]

    return [[a11, a12], [a21, a22]]
end
```

```

function matrice_power(M, p) do
    if p in memoization do
        return memoization[p]
    end

    if p == 1 do
        memoization[p] = result
        return M
    end

    else do
        result = 0
        if p % 2 == 0 do
            tmp = matrice_power(M, p / 2)
            result = multiply_matrices(tmp, tmp)
        end
        else do
            tmp = matrice_power(M, (p - 1) / 2)
            result =  multiply_matrices(multiply_matrices(tmp, tmp), M)
        end
        memoization[p] = result
        return result
    end

end

M = [[1, 1], [1, 0]]
M_p = matrice_power(M, p)
print(M_p[0][1])
```

ب) این بار یک ماتریس واحد $k \times k$ بالا مثلثی در نظر می‌گیریم. (ماتریسی که تمام درایه‌های زیر قطر اصلی آن صفرند و باقی درایه‌هایش یکند)به عنوان مثال برای $k=4$ این ماتریس به شکل زیر خواهد بود:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

پاسخ تمرین ۵

حال برای محاسبه جمله n ام در ابتدا توان $n-k$ ماتریس داده شده را با استفاده از الگوریتم توان ماتریسی که در قسمت قبل بیان شد، به دست می‌آوریم. در ادامه هم یک ماتریس ستونی $k\times1$ تشکیل می‌دهیم که $k-1$ خانه اول آن را با $k-1$ جمله اول دنباله پر می‌کنیم. (به عنوان پایه وجود این جملات ضروری است) و خانه آخر آن را هم با جمله k ام دنباله که برابر با مجموع $k-1$ جمله اول است، پر می‌کنیم. در انتها برای محاسبه جمله n ام کافیست توان $n-k$ ماتریس داده شده را در ماتریس ستونی‌ای که تعریف شد، ضرب نماییم و درایه سطر اول و ستون اول را به عنوان جواب نمایش دهیم. هم‌چنین به ازای n های کم‌تر از k کافیست حاصل جمع n جمله اول را برگردانیم.

تفاوت این قسمت با قسمت قبل در این است که برای محاسبه ضرب ماتریس‌ها دیگر نمی‌توان از تابع ضرب ماتریس قسمت قبل (که هزینه‌اش $O(1)$ بود) استفاده کرد. بلکه این بار هزینه ضرب k^3 می‌شود. چرا که ماتریس‌های $k\times k$ را در هم ضرب می‌کنیم پس هزینه الگوریتم $O(k^3\times logn)$ خواهد بود. الگوریتم ضرب ماتریس‌ها به شکل زیر است و باقی قسمت‌ها مشابه قسمت قبلند.

```python
def matrix_multiply(a, b):
    rows_a = len(a)
    cols_a = len(a[0])
    rows_b = len(b)
    cols_b = len(b[0])
    if cols_a != rows_b:
        raise ValueError("Matrices cannot be multiplied")
    result = [[0]*cols_b for _ in range(rows_a)]
    for i in range(rows_a):
        for j in range(cols_b):
            for k in range(cols_a):
                result[i][j] += a[i][k] * b[k][j]
    return result
```