Subject:
Year: Month: Date:

Sa Su Mo Tu We Th

① برای حل این سوال که به House Robber معروف است

استفاده می‌کنیم. در این الگوریتم ابتدا آرایه‌ای به اسم dp با سایز آرایه

اصلی می‌سازیم و ابتدا با 0 پر می‌کنیم. در این سوال می‌دانیم به نهایت تعداد

خانه‌هایی می‌توانیم انتخاب کنیم دو تا خانه است پس برای چک کردن

از ابتدای آرایه تا index ای از این آرایه با هر دوی آن

هستیم و دو تا خانه قبلی و بالعکس را انتخاب کنیم. لازم به ذکر است به

خانه اول dp با آرایه اولیه یکسان است و خانه دوم یعنی dp[1] را

باید بیشترین بین arr[0] و arr[1] جایگذاری کنیم، از خانه با

index = 2 شرط در بالا گفته شده را تا انتهای آرایه اصلی چک می‌کنیم

آرایه dp را پر می‌کنیم. شبه کد به شکل زیر است:

```
House_Robber(arr):

    dp = [0] * arr.length
    dp = arr[0]
```

MEHR

```
dp[۱] = MAX {arr[۰], arr[۱]}

for i in range(۲, arr.length)

    dp[i] = MAX {dp[i-۲], dp[i-۱], arr[i]}

return dp[arr.length - ۱]
```

۲) برای حل این سوال از یک آرایه به اسم dp استفاده می‌کنیم

که به اندازه تعداد خانه‌ها است و ابتدا آن را با ۰ پر می‌کنیم. مقدار هر خانه

dp نشان دهنده تعداد راه‌های رسیدن به آن خانه از خانه اول است.

در ابتدا dp[0] را برابر ۱ قرار می‌دهیم چون خانه اول است. چون

همه مسیرها از خانه اول به خانه آخر باید ختم شود، مقدار dp خانه آخر را

باید برابر با بیشترین dp در سه خانه قبلی قرار دهیم چون به آن خانه برسد یعنی

بعد از آن هم به خانه آخر می‌رسد و نباید تعداد اضافه کرد. برای اجرای

این الگوریتم به دو حلقه نیاز داریم، اولی از ۱ تا n-۲ را پیمایش می‌کند

و برای هر خانه، وضعیت صندلی‌ها در سه خانه قبلی چک می‌کند و اگر در هر کدام

صندلی وجود داشته باشد، یعنی یک راه به این خانه وجود دارد پس به مقدار dp

آن خانه افزوده می‌شود، در حلقه دوم مقدار dp خانه آخر را برابر با ماکسیمم

سه خانه قبلی آن قرار می‌دهیم و جواب نهایی مقدار dp خانه آخر است

چون در برنامه فقط از دو حلقه n-2 تایی و ۳ تایی استفاده کردیم،

پیچیدگی زمانی برنامه از مرتبه O(n) است. فرض کنیم در

صندلی در خانه ها هم به شکل ۰ و ۱ است به این صورت که اگر در آن

خانه صندلی وجود داشته باشد ۱ و در غیر این صورت ۰. در آرایه seats

قرار بگیرد، شبه کد:

```
Count_Paths(seats):

    dp = [0] * seats.length
    dp[0] = 1

    for i in range(1, seats.length-2):
        if seats[i-1] == 1:
            dp[i] += dp[i-1]
        if seats[i-2] == 1:
            dp[i] += dp[i-2]
        if seats[i-3] == 1:
            dp[i] += dp[i-3]
```

```
int max = 0
for i in range (n-2, n-1):
    if dp[i] > 0 :
        max = dp[i]

dp[seats.length - 1] = max

return dp[seats.length - 1]
```

۳) برای جواب دادن به این سوال می‌توان یک لیست به اندازه سایز آرایه اولیه ایجاد کنیم که آن را با ۱ پر کنیم و سپس با استفاده از دو تا حلقه تو در تو برای هر index از آرایه، آن را با تمام index های قبلی چک کنیم و اگر صعودی باشد، به ازای هر عدد کوچکتر از index i، یک مقدار به آرایه list[i] اضافه کنیم.

پس از پیمایش بر روی کل آرایه بزرگترین عضو آرایه list برابر با طول بزرگترین زیررشته صعودی است.

شبه کد:

```
LCIS(arr):
    List = [1] * arr.length
    for i in range(1, arr.length)
        for j in range(0, i)
            if arr[i] > arr[j] && List[i] < list[j]+1
                List[i] = List[j]+1
    return MAX(List)
```

محمد مهدی عطاری

۳

A_1: ۱۰ × ۳۰

A_2: ۳۰ × ۵

A_3: ۵ × ۹۰

A_4: ۹۰ × ۱۰

p = [۱۰, ۳۰, ۵, ۹۰, ۱۰]

$$m[i,j] = \min\{ m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \}$$

$m[1][1] = m[2][2] = m[3][3] = m[4][4] = 0$

m[۱][۲] = ۱۰ × ۳۰ × ۵ = ۱۵~

m[۲][۳] = ۳۰ × ۵ × ۹۰ = ۹...

m[۳][۴] = ۵ × ۹۰ × ۱۰ = ۴...

$$m[1][3] = \min \begin{cases} m[1][1] + m[2][3] + P_0 P_1 P_3 \\ m[1][2] + m[3][3] + P_0 P_2 P_3 \end{cases}$$

$$= \min \begin{cases} 0 + 9000 + 10 \times 30 \times 60 \\ 1500 + 0 + 10 \times 5 \times 60 \end{cases} = \min \begin{cases} 27000 \\ 4500 \end{cases}$$

$$= 4500$$

$$m[2][4] = \min \begin{cases} m[2][2] + m[3][4] + P_1 P_2 P_4 \\ m[2][3] + m[4][4] + P_1 P_3 P_4 \end{cases}$$

$$= \min \begin{cases} 0 + 3000 + 30 \times 5 \times 10 \\ 9000 + 0 + 30 \times 60 \times 10 \end{cases} = \min \begin{cases} 4500 \\ 27000 \end{cases}$$

$$= 4500$$

$$
m[1][4] = \min \begin{cases} m[1][1] + m[2][4] + P_0 P_1 P_4 \\ m[1][2] + m[3][4] + P_0 P_2 P_4 \\ m[1][3] + m[4][4] + P_0 P_3 P_4 \end{cases}
$$

$$
= \min \begin{cases} 0 + 4500 + 10 \times 30 \times 10 = 7500 \\ 1500 + 3000 + 10 \times 5 \times 10 = 5000 \\ 4500 + 0 + 10 \times 60 \times 10 = 10500 \end{cases}
$$

$$
= 5000
$$

مینیمم تعداد عملیات ضرب برابر ۵۰۰۰ است.

نحوه پرانتز گذاری نهایی به شکل زیر است:

$$((A_1 A_2)(A_3 A_4))$$