شهرزاد دیندار _ ۸۱۰۱۰۰۸۷۲

تمرین طراحی الگوریتم

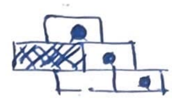
سوال ①

خب در این سوال قصد داریم راه های چیدن ۱۰۰ هزار آجر متصل شکل که نشان داده ایم را به دست آوریم

برای اینکه به روش dynamic programming این را حل کنیم، به طوری که به صورت بازگشتی عمل می‌کنیم:

تعداد راه های چیدن n آجر با این شرط را DP[n] در نظر می‌گیریم. ۳ حالت در نظر داریم برای چینش آجرها:

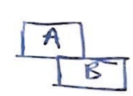
حالت ① : سمت راست ترین آجر، در ردیف اول روی زمین باشد و بر روی آن هیچ آجری قرار نگرفته باشد، در واقع تعداد حالاتی که چنین آجری داشته باشیم، به اندازه DP(n-1) است. چون که به جواب اول را داشته باشیم، این آجر را در سمت راست ترین سمت می‌گذاریم و همان تعداد جواب را داریم.

حالت ② : اگر سمت راست  را ببینیم، تعداد حالت هایی که سمت راستشان شکل را می‌بینیم را باید حساب کنیم آجرهایی را که با علامت " • " مشخص کرده ام را در نظر می‌گیریم فقط. تعداد حالات جواب ما برابر DP(n-3) هست

پس آجر  را نیز باید در نظر داشته باشم. اگر این آجر را نداشتیم. اصلاً چنین ترکیبی را برای آجرها نداشتیم

پس DP[n-3] را باید منهای تعداد حالات حالت هایی کنیم که  وجود ندارد یعنی DP[n-4]

را از آن مقدار کم می‌کنیم.

حالت ③ : شکل  را در نظر بگیرید. فرض کنید B سمت راست ترین آجر باشد، A، آجر بالای آن باشد. این آجر حالت دیگری برای حضور و چینش ندارد پس ۲ DP[n-2] حالت این ۲ آجر می‌توانند نسبت به هم قرار بگیرند

تعداد کل حالات برابر زیر است:

$$DP[n] = DP[n-1] + (DP[n-3] - DP[n-4]) + DP[n-2]$$

که به صورت بازگشتی تابع را نوشتیم.

برای کیس دیتی ها حتما: ~~گفته~~

DP[0]=1 ① حالت دارم	DP[۱]=1 ① حالت وجود دارد
DP[۲]=1 ① حالت دارم	DP[۳]=۲ ② حالت دارم.
DP[۳]=۲ ② حالت	DP[۴]=۳ ③ حالت دارم،

Time ما از مرتبه $O(n)$ است زیرا مطابق جدول با چیدن n خانه برای گرفتن $O(n)$ زمان لازم می‌شود است که به روش DP آن را حل کردیم.

سوال ۲

دوباره مسئله را حل [کنیم] اما با فرض نبودن همه ... که سعی میکنیم دستگاه طوری حل کنیم که یعنی با کمترین تعداد پول را خرد کند.

الف) v_i فروش تعداد دارد که پول s را میدهیم در $O(sn)$

و در ماشین بقیه خرد کند.

از DP استفاده میکنیم و جدول میکشیم. جدول مربوط را از سمت چپ به راست و در هر سطر افقی پر میکنیم.

تعداد لازم کمترین تعداد پول را از نظر بگیریم و در هر مرحله min میگیریم.

j →

i ↓

	0	1	...	$j-v_i$	...	j	...	s
0	0	0	0...	0	...	0	...	0
1	0							
⋮								
i-1								
n								

n+1

s+1

چند حالت داریم: حالت ۱: تا آن جایی که میتوانیم از این پول برمیداریم تا آن ستون را پر کنیم به مقدار

چون که باقی مانده کمتر از v_i است باید طبق تغییر cut & paste از حالت بهینه برداشته شود ثابت برداریم.

این بیان معنی هست که در همان سطری که هستیم، به دنبال یک ستون بگردیم که مقدارش $j-v_i$ باشد برگردیم و این مقدار را با مقداری که جای که هیچ صدق کنیم، جمع کنیم.

```
for(i=0 to i=n){
    Dp[i][0]=0;}
```

حالت ۲: از سکه v_i استفاده نکنیم و یک سطر قبل و در ستون ثابت برویم.

در ابتدای الگوریتم سطر اول و ستون اول را ۰ پر میکنیم چون در ابتدای کار هیچ مقدار پولی نداریم.

$$Dp[i][j] = \min(Dp[i][j-v_i],\ Dp[i-1][j]+1)$$

```
for (j=0 to j=s){
    Dp[0][j]=0;}
```

ب) نه! خیر. order زمانی الگوریتم ما که $O(sn)$ هست چند جمله‌ای نیست و در مورد

باید بگوییم که شبه چندجمله‌ای هست در واقع.

s یک عدد است که از ورودی آن را میخوانیم و save میکنیم و اندازه s

برابر $\boxed{\log_2^s}$ هست. پس $s=2^x$ است و $s = 2^x = 2^{\log_2^s}$ لذا معلوم میشود که

$O(sn)$ چندجمله‌ای نیست. زیرا $O(sn) = O(s \times 2^x)$ و 2^x نیز از مرتبه نمایی

است. s یک عدد است و اندازه آن از مرتبه لگاریتمی است $(|s| = \log_2^s)$ نتیجه میشود که

$O(sn)$ چندجمله‌ای نیست.

سوال ③

الف) به order $O(n^2)$ می‌خواهیم الگوریتمی که گفته پیاده‌سازی کنیم. از ۲ loop تو در تو استفاده می‌کنیم. ترجیحاً از for استفاده می‌کنیم. در loop① تک تک عناصر را پیمایش می‌کنیم. و در loop② که درون for دوم هست یک حلقه تعریف می‌کنیم.

```
int temp sum = 0
int Max Sum = -inf
for (i=0 ; i<n-1 , i++ ){
    tempSum = 0;
    for (j=i ; j<n-1 ; j++ ){
        tempsum = A[i] + temp Sum ;
        max Sum = Max {(temp Sum , Max Sum)} ;
}}
```

[illegible]

یک متغیر به نام tempSum تعریف که در هر مرحله آن را با A[j] جمع می‌کنیم و طبق روش بالا با حداکثر تابع Max، ماکسیمم آن دو را حساب می‌کنیم و داخل Max Sum می‌ریزیم.

$$\sum_{i=0}^{n} \sum_{j=i}^{n} O(1) = O(n^2)$$

چون ۲ لوپ داریم پیچیدگی زمانی ما $O(n^2)$ هست.

سوال ۳ ـ ب) برای حل این قسمت از DP استفاده می‌کنیم که order الگوریتم بهتری ارائه می‌کند

متغیر جدیدی به نام P[i] تعریف می‌کنیم که، آن را ماکسیمم مقدار زیرآرایه تا عنصر i ام تعریف می‌کنیم. رابطه‌ای که معادل پاسخ داریم را با استفاده از تابع بازگشتی پیاده‌سازی کردیم:

n مقدار عناصر آرایه A می‌باشد.

~~func DP(A)~~

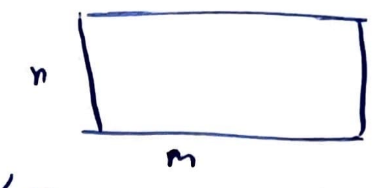
```
func   DP( A ) {
   temp_max = 0
   Max = -∞
   for ( i=0 ; i<n-1 ; i++ ){
         temp_max = ( Max { A[i], A[i]+ temp_max } ) }

   max = (temp_max > max)? temp_max : max   →

   return max ;  }
```

اگر temp-max از max بزرگتر بود مقدار آن را جایگزین max می‌کنیم و در max می‌ریزیم.

$P[i] = \max\{A[i]+P[i] , A[i]\}$ می‌باشد.

محسن

سوال ۴ .

مستطیل  (m×n) را داریم و می‌خواهیم زیر مستطیلی را پیدا کنیم که جمع اعداد

داخل آن max باشد. برای آنکه به روش DP حل کنیم یک جدول تشکیل می‌دهیم که به صورت زیر هست.

	1	2	3		m
1					
2					
⋮					
n					

به ازای همه حالت های ممکن مستطیل از این جدول m×n حالت $O(n^2)$ داریم. مثل سوال (۳ ـ الف) با الگوریتم در $O(n)$ محاسبات را انجام می‌دهیم.

این متغیر باید جمع اعداد داخل را و همچنین ابعاد را باید خروجی دهیم. متغیر ① Sum_Max ~~[illegible]~~ ② S_max_left ③ S_max_Right ④ S_Max_up ⑤ S_max_Down را تعریف کنیم.

در این مرحله باید L و R (U ها) ذکر شده را هر سری به روز رسانی کنیم.

ضلع چهارم کدام ها هستند. یعنی ضلع بالا مستطیل روی کدام خانه است.

ادامه ۴

Time complexity از $O(n^3)$ و Space complexity از $O(n^2)$ هست

می‌کنیم

```
for ( L=0 ; L<m ; L++ ){
    for (R =0 ; R<m ; R++ ){
        Max_Row = Max ( ماکسیمم این row را حساب می‌کنیم ) ;
        a = cont + Sum ;
    }
}
```

$O(n)$ ← جمع‌اول

که در اردر $O(n)$ عمل می‌کند.

درحقیقت زمانی که سوال می‌گه این الگوریتم اجرا شود به اندازه $O(m^2 \times n)$ هست حال به اندازه همهٔ حالات $L \leq R$ که $L \leq R$ الت جواب $sum + count$ را تکرار می‌کند و با sum_MAX مقایسه می‌کند درصورتی که بزرگتر بود، مقادیر تعداد MAX آپدیت می‌شوند. و ماکسیمم آن row را حساب می‌کنیم. طبق این سوال از طریق DP، تشکیل جدول در $O(n^3)$ مسئله حل کنیم

سوال ⑤

حالا در این مسئله باز قدوس کار جدیدی کرده است. با حرکت های $\rightarrow$ و $\downarrow$ به خانه راست پایین جدول بیاییم

فرض مطابق جدول باید از خانه A به خانه B برویم.

ارزش هر خانه جدول را با V نمایش میدیم. و به صورت بازگشتی مسئله را حل می‌کنیم. $B[i,j]$ را طول تعریف می‌کنیم که max طول دریافت شده است.

~~DP~~ ، ~~...~~

$c[i][j]$ را تعریف می‌کنیم.

$$c[i][j] = Max(c[i-1][j] , c[i][j-1]) + V[i][j]$$

ارزش هر خانه.

```
c[0][0] = V[0][0]
for( i=0 to i<m){
    c[i][0] = c[i-1][0] + V[i][0] ; }
for( j=0 to j<n){
    c[0][j] = c[0][j-1] + V[0][j] ; }
for (j=1 to j<n ){
  for i=1 to i<m){
    c[i][j] = max( c[i-1][j] , c[i][j-1] ) + V[i][j] ;}}

return c[m-1][n-1]
```

ادامه ⑤

این الگوریتم با DP عمل می‌کند، جواب نهایی را return می‌کند

یعنی در هر مرحله چک می‌کند که کدام حالت ارزش بهینه جوک است و طبق cut & paste جواب بهینه را برمی‌گرداند

مسئله حل شد!

سوال ⑥

خب در این سوال تعدادی داریم بیشترین درس‌ها را با بودجه‌ای که داریم انتخاب کنیم. کد آرایه P[m,n]

تعریف می‌کنیم و با DP حل می‌کنیم که m تعداد سال‌های تحصیل است و n پول اندازه است.

یعنی P[m,n] بیشترین درس ممکن است که تحصیل در m سال و با پول اندازه n داشت. P[N,S]

را حساب می‌کنیم

$$P[m,n] = \begin{cases} \max \left\{ \underbrace{P[m-1, n-\sum_{i=1}^{m'} c_i] + i}, \underbrace{P[m-1,n]}, \underbrace{P[m-1, n-\min(\sum c_{im})]+1} \right\} \\ 0 \qquad \text{if } m=0 \text{ or } n=0 \end{cases}$$

$m' =$ درس‌های سال m

ماکسیمم ۳ حالت را باید حساب کنیم: حالت اول موجود این است که اونهمه درس سال تحصیلی m ام همه درس‌ها را برمی‌دارد، حالت دوم این است که هیچ درسی برنمی‌دارد و حالت سوم این است که در سال m ام فقط درس با min هزینه را برمی‌دارد.

Max (حالت اول ، حالت دوم ، حالت سوم)

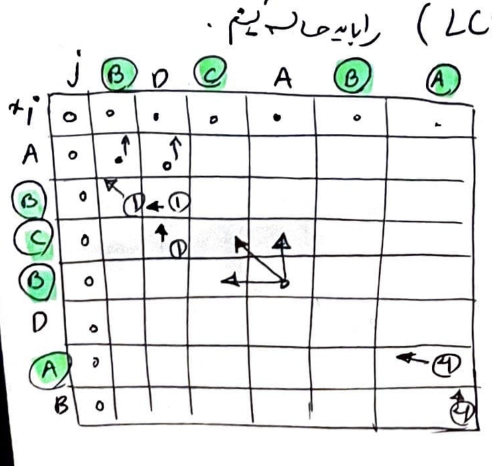
این را در بالا حساب کردیم

سوال ⑦

الف) طولانی‌ترین longest commat substring (LCS) را باید حساب کنیم.

مطابق شکل LCS که داشتیم عمل می‌کنیم. یک جدول شکل داریم. برای ۲ رشته X = <A,B,C,B,D,A,B> ،

Y = <B,D,C,A,B,A> مثال داریم.



ادامه ۷

آرایه c[i,j] را برای این جدول تعریف می‌کنیم که LCS، X_1 تا X_i و Y_1 تا Y_j را return می‌کند.

- ساختار بهینه
- معادله

```
LCS(X,Y){
  m = length[X]
  n = length[Y]
  for (j=1 to n){ c[0,j]=0 }
  for (i=1 to m){ c[i,0]=0 }

  for (i=1 to m){
    for (j=1 to n){
      if (x_i = y_j){
        c[i,j] = c[i-1, j-1] + 1 ;
        b[i,j] = "↖" ; }
      else if (c[i-1,j] >= c[i,j-1]){
        c[i,j] = c[i-1,j] ;
        b[i,j] = "↑" ; }
      else
        c[i,j] = c[i,j-1]
        b[i,j] = "←"

  return b, c ;
}
```

سطر ۰ ام و ستون ۰ ام را برابر ۰ قرار می‌دهد

سمت بالا چپ

سمت بالا اشاره دارد

این الگوریتم از $O(m \times n)$ است.

ب) برای ثبت کردن این آرایه B[i,j] را تعریف می‌کنیم به طوری که با توجه به مقدار c[i,j] جهت را save کند. که می‌تواند "←"، "↑"، "↖" باشد. فقط در جایی که حرکت ما به سمت ↖ باشد ما باید کاراکتر مورد نظر را چاپ کنیم.

```
print_LCS(b, X, i, j){
  if (i==0 or j==0)
    return
  if (b[i,j] = "↖"){
    print_LCS(b, c, i-1, j-1) ;
    print x ; }
  else if (b[i,j] = "↑"){
    print_LCS(b, c, i-1, j) }
  else
    print_LCS(b, c, i, j-1)
```

سوال ۱

قصد داریم با استفاده از DP الگوریتمی ارائه دهیم که مسئله را حل کند.

$D[i]$ ← بهترین پارتی که (i) ریشه باشد.

$P[i]$ ← بهترین پارتی که (i) ریشه باشد ولی در مهمانی دعوت نشده باشد.

$$D[i] = \max\Big(\underbrace{v_i}_{\text{میزان محبوبیت}} + \sum P[j]\ ,\ P[i]\Big)$$

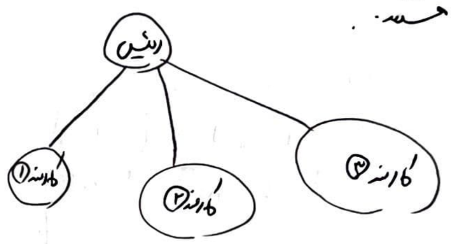
$j \in child(i)$

$$P[i] = \sum_{j \in child(i)} D[j]$$

برای محاسبه $D[i]$ و $P[i]$ باید در ابتدا تمام فرزندان آنها را محاسبه کنیم. یک DFS روی درخت مدیریت اول بزنیم. تا جواب مورد نظر بخش بالایی انتخابی را ببینیم هر ریشه رئیس است و کارمندان متعلق فرزندان آن هستند.

complexity الگوریتم ← $O(n)$

DFS پیمایش space complexity ← $O(n)$



سوال ۲

اعداد فیبوناچی را می‌خواهیم در order $O(\log(n))$ حساب کنیم با استفاده از ضرب ماتریس‌ها

عملیات را انجام میدهیم.

$$\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix} = X^{n-1} \begin{pmatrix} F_1 \\ F_0 \end{pmatrix} = X^{n-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

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

همانطور که اگر سطر اول $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ را در ستون اول $\begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix}$ ضرب کنیم حاصل F_n می‌شود و واضح است اگر سطر دوم آن را در ستون اول " ضرب کنیم F_{n-1} را می‌دهد.

توان ماتریس به روش زیر محاسبه می‌شود.

$$M^n \rightarrow \begin{cases} (M^{n/2})^2 & \text{if } n \text{ is even} \\ (M^{\frac{n-1}{2}})^2 \times M & \text{if } n \text{ is odd} \end{cases}$$

در هر مرحله نصف می‌شود چون $n \times (1/2)^k = 1$ است ← $n = 2^k$ و $\boxed{\log n = k}$ و $\boxed{\text{از } O(\log n)}$ است.

۹-ب) باید یک ماتریس پیدا کنیم که در ضرب آن در ماتریس a_1 تا a_k، F_n را بدهد. که در پایین آن را توضیح داده ام:

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & \cdots & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & \cdots & & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & & \cdots & & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & & \cdots & & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & & \cdots & & \vdots & \\ \vdots & & \cdots & & & & & 1 & 0 & 0 \\ & & & & & & & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & & \cdots & & & \end{bmatrix}}_{A} \times \begin{bmatrix} F_{n-1} \\ F_{n-2} \\ \vdots \\ F_{n-k} \end{bmatrix} = \begin{bmatrix} F_n \\ F_{n-1} \\ \vdots \\ F_{n-k+1} \end{bmatrix}$$

طبق ضرب ماتریس که از order $O(\log^{n}_{2})$ بود.

این از order $O(\log^{n}_{n-k})$ میباشد.

$$A \times \begin{bmatrix} F_{k-1} \\ F_{k-2} \\ \vdots \\ F_0 \end{bmatrix} = \begin{bmatrix} F_n \\ F_{n-1} \\ \vdots \\ F_{n-k} \end{bmatrix}$$