برنامه نویسی پویا

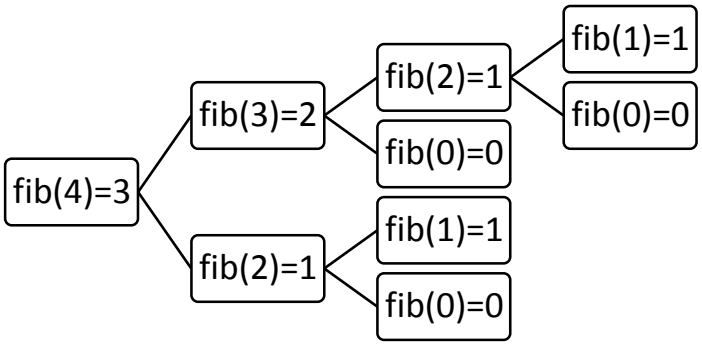
در این روش حل مساله را از کوچکترین مسائل شروع و همه آن ها را حل می کنیم و جواب آن ها را نگهداری می کنیم سپس به سطح بعدی می رویم و کلیه مسائل اندکی بزرگتر را حل می کنیم و سپس به حل مسائل سطح بعدی می پردازیم و کار را تا حل مساله اصلی ادامه می دهیم. برای هر یک از مسائل هر سطح می توان از حل کلیه سطوح پایین تر که لازم باشد استفاده کنیم. در واقع برنامه نویسی پویا یک روش پایین به بالاست. در این روش، بازگشتی فکر می کنیم ولی بازگشتی پیاده سازی نمی کنیم.

مثال۱

```
Int fib (int n);
  {
    If ( n<=1)
       Return n;
    Else
       Fib (n-1) +fib (n-2)
}
```

حل به روش تقسیم و حل

می خواهیم این مساله را به روش قدیم تقسیم و حل، پیش ببریم، پس مقدار n=4 را در نظر گرفته و ادامه می دهیم.



نکته:

در روش تقسیم و حل اگر پس از تقسیم یک نمونه زیرنمونه های بدست آمده مستقل از یکدیگر باشند مانند مرتب سازی ادغامی آنگاه معمولاً روش تقسیم و حل موثر است اما در مواردی که زیرنمونه ها مستقل از هم نباشند مثل سری فیبوناچی و زیرنمونه ها تکراری باشند آنگاه زمان اجرا یالگوریتم تقسیم و حل بدلیل حل نمونه های تکراری نمایی یا بدتر خواهد بود.

حل به روش پویا

برنامه زیر را در نظر بگیرید:

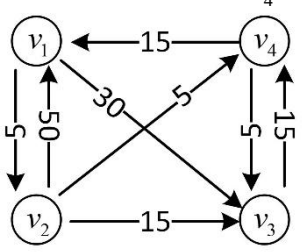
```
Fib [1]=1
Fib [2]=1
   For (i=3; f[i-1] + f [i-2]);
      Return f[n];
```

پیچیدگی این تابع $\theta(n)$ است.

مثال:

الگوریتم Floyd: یافتن کوتاهترین همه مسیرها

یکی از مسائل معمول یافتن کوتاهترین مسیر بین دو شهر است. جهت نمایش شهرها و طول مسافت بین آن ها می توان از یک گراف وزن‌دار که به ماتریس همجواری معروف است استفاده کرد. در مثال چهار شهر داریم به نام های v_1 تا v_4



این گراف را در یک ماتریس به نام w ذخیره می کنیم.

حالا مسیرهای بین شهرها (گره ها) را در ماتریس نشان می دهیم. اگر بین دو شهر (براساس گراف) مسیری وجود نداشت از نماد ∞ استفاده می کنیم.

	v_1	v_2	v_3	v_4
v_1	0	5	∞	∞
$w = v_2$	50	0	15	5
v_3	30	∞	0	15
v_4	15	∞	5	0

یک الگوریتم ساده و بسیار کند آن است که تمامی مسیرها بین گره i به j را پیدا کرده و سپس کمترین آنان را بدست آوریم. مثلاً در حالتی که گراف با n گره کامل باشد و از هر گره به تمامی گره های دیگر یالی وجود داشته باشد برای رفتن از گره i به j با فرض آن که در بدترین حالت بخواهیم از همه رئوس بگذریم باید از n-2 گره بگذریم پس از آن برای رفتن به گره سوم n-3 انتخاب داریم و الی آخر. در این حالت پیچیدگی برابر O(n-2)! و به صورت ساده O(n!) است.

در ادامه می خواهیم الگوریتمی بروش برنامه نویسی پویا ارائه دهیم که از مرتبه $\theta(n)^3$ باشد در واقع می خواهیم الگوریتمی ارائه دهیم که فقط طول کوتاهترین مسیرها را به ما بدهد و سپس در هر مرحله قدری آن را اصلاح می کنیم تا به ماتریس نهایی برسیم بنابراین در حال حاضر هدف فعلی ما محاسبه ماتریسی به نام D از روی ماتریس w می باشد.

	v_1	v_2	v_3	v_4
v_1	0	5	∞	∞
$w = v_2$	50	0	15	5
v_3	30	∞	0	15
v_4	15	∞	5	0

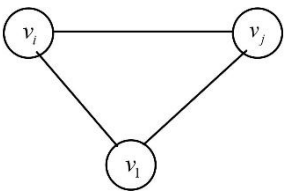
	1	2	3	4
1	0	5	15	10
$D = 2$	20	0	10	5
3	30	35	0	15
4	15	20	5	0

به عنوان مثال در این ماتریس $D[2][3]=10$ به این معناست که از گره v_2 به گره v_3 کوتاهترین مسیر طول ۱۰ را دارد. برای حل این مساله لازم است ماتریس های $D_0, D_1, \ldots D_n$ را حساب کنیم که n تعداد شهرها یا گره ها می باشد. برای حل مساله D_0 را برابر w فرض کرده و سپس D_1 را از روی D_0 و D_2 را از روی D_1 و ... محاسبه می کنیم.

منظور از $D_k[i][j]$ طول کوتاهترین مسیر از v_i به v_j فقط با استفاده از رئوس موجود در مجموعه $v_1, v_2, \ldots v_k$ به عنوان رأس واسطه می باشد.

منظور از $D_0[i][j]$ یعنی طول کوتاهترین مسیر از v_i به v_j بدون هیچ واسطی می باشد، پس بدیهی است سیستم همان w می باشد.

مفهوم $D_1[i][j]$ یعنی کوتاهترین مسیر بین v_i به v_j که واسط بین آن ها v_1 است یا مسیر بین v_i تا v_j مستقیم است یا مجموع v_i تا v_1 و v_1 به v_j



$$D_0[i][1] + D_0[1][j] = Min\left(D_0[i][j], D_0[i][1] + D_0[1][j]\right)$$

رابطه کلی:

$$D_2[i][j] = Min\left(D_1[i][j], D_1[i][2] + D_1[2][j]\right)$$
$$D_k[i][j] = Min\left(D_{k-1}[1][j], D_{k-1}[i][k] + D_{k-1}[k][j]\right)$$

```
For (k=1 , k<=n , k++);
    For (i=1 , i<=n , i++);
        For (j=1 , j<=n , j++);
            D[i][j]=min (D[i][j], D[1][k] + D[k][i]);
```

$$D_1\begin{bmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{bmatrix} \quad D_2\begin{bmatrix} 0 & 5 & 2 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{bmatrix} \quad D_3\begin{bmatrix} 0 & 5 & \infty & \infty \\ 45 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{bmatrix} \quad D_4\begin{bmatrix} 0 & 5 & 15 & 10 \\ 20 & 0 & 10 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{bmatrix}$$

پیچیدگی فضایی برابر n^2 اما پیچیدگی عادی n^3 است.

تا اینجا ماتریس D را پیدا کردیم که حداقل طول مسیر از هر گره i به j را نشان می دهد در ادامه می خواهیم رئوسی که در این مسیر کوتاه i به j وجود دارند را چاپ کنیم. برای این کار کافی است تغییر اندکی به برنامه محاسبه D بدهیم و ماتریس P را نیز بدست آوریم، ماتریس P ماتریس n×n با اندیس های 1 تا n که مقدار اولیه آن ها صفر است. سپس خط آخر برنامه را حذف کرده و به شکل زیر تغییر می دهیم:

```
For (k=1 , k<=n , k++);
```

```
    For (i=1 , i<=n , i++);
        For (j=1 , j<=n , j++);
            If (D[i][k] + D[k][j]  < D[i][j] {
                P[i][j]=k;
                D[i][j]= D[i][k]+ D[k][j];
```

$$P=\begin{bmatrix}0 & 0 & 4 & 2\\ 4 & 0 & 4 & 0\\ 0 & 1 & 0 & 0\\ 0 & 1 & 0 & 0\end{bmatrix}$$

برای مثال کوتاهترین مسیر از گره ۱ به گره ۳ براساس ماتریس P به صورت زیر محاسبه می شود: P[1][3]=4

الگوریتم Floyd: تابع محاسبه واسط بین کوتاهترین رئوس

```
Void path (int  I , int j);
 {
   If (P[i][j]!=0);
     {
       Path (i,P[i][j]);
       Print f  (P[i][j]);
       Path (P[i][j]);
     }
 }
```

مثال دوم از برنامه نویسی پویا: ضرب زنجیره ای ماتریس ها

نکته: ضرب دو ماتریس A در B به شرطی تایید می شود که بعد وسط آن ها یسکان باشد. یعنی تعداد ستون های ماتریس اول با تعداد سطرهای ماتریس دوم برابر باشد.

$$\begin{bmatrix}1 & 4 & 1\\ 5 & 5 & 1\end{bmatrix}\begin{bmatrix}4 & 8\\ 1 & 2\\ 6 & 5\end{bmatrix}=\begin{bmatrix}14 & 13\\ 31 & 55\end{bmatrix}$$

نکته: ماتریس حاصل به مقدار تعداد سطر ماتریس اول در تعداد ستون ماتریس دوم درایه دارد.

$$A_{n\times m}\times B_{m\times k}=C_{n\times k}$$

نکته: تعداد کل ضرب ها برابر با $n\times m\times k$ (تعداد سطر ماتریس اول در تعداد ستون ماتریس اول در تعداد ستون ماتریس دوم) می باشد.

مثال

سه ماتریس زیر مفروض است، به چه ترتیب ضرب انجام شود تا تعداد ضرب های کمتری داشته باشیم:

$$A_{5\times 10}\times B_{10\times 30}\times C_{20\times 4}$$

روش ۱

$$A_{5\times 10}\times B_{10\times 20}=(AB)_{5\times 20}\rightarrow 5\times 10\times 20=1000$$
$$(AB)_{5\times 20}\times C_{20\times 4}=(ABC)_{5\times 4}\rightarrow 5\times 20\times 4=400$$
$$1000+400=1400$$

روش ۲

$$B_{10\times 20}\times C_{20\times 4}=(BC)_{10\times 4}\rightarrow 10\times 20\times 4=800$$
$$A_{5\times 10}\times (BC)_{10\times 4}=(ABC)_{5\times 4}\rightarrow 5\times 10\times 4=200$$
$$800+200=1000$$

همانطور که می بینید در روش دوم تعداد ضرب کمتر است.

به صورت سرانگشتی می توان گفت ابتدا ماتریس هایی را درهم ضرب می کنیم که بعد وسط آن ها بزرگتر و بعد کناری کوچک تر باشد در نتیجه ماتریس حاصل کوچکتر می شود البته همیشه این روش جوابگو نیست.

یک روش ساده ولی بسیار کند آن است که همه ترتیب های ممکن جهت ضرب n ماتریس را در نظر گرفته سپس ترتیبی را انتخاب کنیم که کمترین تعداد ضرب را داشته باشد این الگوریتم ساده حداقل از مرتبه نمایی است.

حل به روش برنامه نویسی پویا

فرض کنیم می خواهیم چهار ماتریس زیر را در هم ضرب کنیم:

A_1	A_2	A_3	A_4
5×2	2×3	3×4	4×6
$d_0\ d_1$	$d_1\ d_2$	$d_2\ d_3$	$d_3\ d_4$

به این صورت یک آرایه داریم که ابعاد ماتریس یعنی d_0 تا d_4 را در آن ذخیره می کنیم:

5	2	3	4	6
d_0	d_1	d_2	d_3	d_4

برای حل این مساله به روش برنامه نویسی پویا از یک ماتریس M با n سطر و n ستون (M[n][n]) استفاده می کنیم. (n تعداد ماتریس هایی است که می خواهیم درهم ضرب کنیم) خانه های این ماتریس را با مقادیر زیر پر می کنیم:

	1	2	3	4
1	0	30	64	132
2		0	24	72
3			0	72
4				0

اگر i=j باشد M[i][j]=0

اگر i<j باشد حداقل تعداد ضرب های لازم برای ضرب A_i تا A_j =M[i][j]

برای حل این ماتریس باید قسمت بالای قطر اصلی که i بزرگتر از j است را محاسبه کنیم، در مرحله اول

ضرب چهار ماتریس A_1 ، A_2 ، A_3 و A_4 به صورت بازگشتی یکی از ۳ حالت زیر را دارد، یعنی اولین نقطه جدا کننده (اولین پرانتز باز) می تواند بعد از A_1 ، یا بعد از A_2 یا بعد از A_3 باشد. به شکل زیر:

$$(A_1)(A_2A_3A_4)$$
$$(A_1A_2)(A_3A_4)$$
$$(A_1A_2A_3)(A_4)$$

$$(A_1)_{d_0d_1}(A_2A_3A_4)_{d_1d_4} = M[1][1]+M[2][4]+d_0d_1d_4$$
$$(A_1A_2)_{d_0d_2}(A_3A_4)_{d_2d_4} = M[1][2]+M[3][4]+d_0d_2d_4$$
$$(A_1A_2A_3)_{d_0d_3}(A_4)_{d_3d_4} = M[1][3]+M[4][4]+d_0d_3d_4$$

فرمول نهایی به صورت زیر خواهد بود:

اگر $i=j$ باشد پس $M[i][j]=0$

اگر $i<j$ باشد پس $M[i][j]=\min_{i\le k\le j-1}\left(M[i][k]+M[k+1][j]+d_{i-1}d_kd_j\right)$

مثال

حالا می خواهیم براساس نمونه قبلی M[1][2] را حساب کنیم:

چون i=1 و j=2 است و j<k می باشد پس:

$$M[1][2]=\min_{1\le k\le 1}\left(M[1][1]+M[2][2]+d_0d_1d_2\right)=30$$

سپس i=1 و j=3 پس:

$$M[1][3]=\min_{1\le k\le 2}\left(\min\left(M[1][1]+M[2][3]+d_0d_1d_3\right)\right)=64$$

K نقطه جداکننده را تعیین می کند و در جدول زیر قرار می گیرد:

	1	2	3	4
1		1	1	1
2			2	3
3				3
4				

و نقاط جدا کننده به شکل زیر نمایش داده خواهند شد:

$$\left(A_1\left(A_2A_3\right)A_4\right)$$

الگوریتم حداقل تعداد ضرب ها

```
int    minmult  (int n, cons tint d[] , index [][]);
   {
     Index i,j,k,d;
     Int m [1…n][1…n]
        For (i=1 , i<=n , i++);
           M[i][j]=0
             For (d=1 , d<=n-1  , d++);
                  For (i=1 , i<=n-d , i++);
                     j=i+d;
                     M[i][j]=minimum (M[i][k]+M[k+1][j]+d[i-1]d[k]d[j]);
                     P[i][j]= a value of k that gave the minimum;
           }
      Return M[1][n];
}
```

الگوریتم محاسبه ماتریس P

```
Void   order(index  i, index j);
   {
      If (i==j);
         Cout <<"A"<<i;
           {
      Else
          K=p[i][j];
          Cout << "(" ;
              Order (i,k);
              Order (k+1,j);
               Cout << ")" ;
      }
}
```