طراحی الگوریتم

موسسه آموزش عالی پاسارگاد شیراز

برنامه نویسی پویا
(Dynamic Programming)

- ضرایب دو جمله ای

$$(a+b)^n = \binom{n}{0} a^n + \binom{n}{1} a^{n-1}\,\mathrm{b} + \binom{n}{2} a^{n-2}\, b^2 + \ldots\ldots + \binom{n}{\mathrm{n}} b^n$$

$$\binom{n}{\mathrm{k}} = \frac{n!}{k!\,(n-k)!} \quad \text{OR} \quad \binom{n}{\mathrm{k}} = \begin{cases} 1 & k == 0 \;||\; k == n \\ \binom{n-1}{\mathrm{k}-1} + \binom{n-1}{\mathrm{k}} & else \end{cases}$$

- در روشهای بالا مرتبه اجرایی نمایی می باشد
- به کمک مثلث خیام پاسکال و استفاده از روش پویا می توان مرتبه اجرایی را کم کرد

برنامه نویسی پویا
(Dynamic Programming)

- ضرایب دو جمله ای
- به کمک مثلث خیام پاسکال و استفاده از روش پویا می توان مرتبه اجرایی را کم کرد

ماتریس B را به صورت زیر ایجاد می کنیم

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
……

$$B[i][j]=\begin{cases}1 & j==0 \;||\; j==i \\ B[i-1][j-1]+B[i-1][j] & else\end{cases}$$

برنامه نویسی پویا
(Dynamic Programming)

- ضرایب دو جمله ای
- به کمک مثلث خیام پاسکال و استفاده از روش پویا می توان مرتبه اجرایی را کم کرد

```
void bin(int B[][m], int n, int k)
{
        int i, j;
        for (i = 0; i <= n; i++)
                for (j = 0; j <= i; j++)
                {
                        if (i == j || j == 0)
                                B[i][j] = 1;
                        else
                                B[i][j] = B[i-1][j-1] + B[i-1][j];
                }
}
```

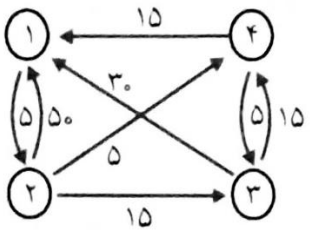
برنامه نویسی پویا
(Dynamic Programming)

- الگوریتم فلوید: پیدا کردن کوتاهترین مسیر بین نقاط

فرض کنید $G=(V,E)$ یک گراف جهت دار با n راس است که $V=\{1,2,...,n\}$ مجموعه رئوس گراف و E مجموعه یال‌ها است. همچنین فرض کنید C ماتریس هزینه‌های گراف است و به صورت زیر تعریف می‌شود:

$$C[i,j]=\begin{cases}0 & \text{اگر } i=j \text{ باشد}\\ \infty & \text{اگر از راس i به j یال نباشد.}\\ \text{وزن یال} & \text{اگر یالی از راس i به j باشد}\end{cases}$$

به عنوان مثال برای گراف مقابل ماتریس C عبارت است از:



$$\Rightarrow C=\begin{array}{c}\\ ۱\\ ۲\\ ۳\\ ۴\end{array}\begin{array}{c}\begin{array}{cccc}۱ & ۲ & ۳ & ۴\end{array}\\ \begin{bmatrix}۰ & ۵ & \infty & \infty\\ ۵۰ & ۰ & ۱۵ & ۵\\ ۳۰ & \infty & ۰ & ۱۵\\ ۱۵ & \infty & ۵ & ۰\end{bmatrix}\end{array}$$

برنامه نویسی پویا
(Dynamic Programming)

- الگوریتم فلوید: پیدا کردن کوتاهترین مسیر بین نقاط

می‌خواهیم تمامی کوتاهترین مسیرها بین رئوس گراف را بیابیم. به عبارتی می‌خواهیم یک ماتریس $A_{n\times n}$ بیابیم که درایه $A[i,j]$ حاوی طول کوتاهترین مسیر از راس i به راس j می‌باشد.

به عنوان مثال برای گراف شکل فوق ماتریس A به صورت زیر بدست خواهد آمد:

$$A = \begin{array}{c} \\ ۱ \\ ۲ \\ ۳ \\ ۴ \end{array} \begin{array}{c} \begin{array}{cccc} ۱ & ۲ & ۳ & ۴ \end{array} \\ \begin{bmatrix} ۰ & ۵ & ۱۵ & ۱۰ \\ ۲۰ & ۰ & ۱۰ & ۵ \\ ۳۰ & ۳۵ & ۰ & ۱۵ \\ ۱۵ & ۲۰ & ۵ & ۰ \end{bmatrix} \end{array}$$

مثلاً درایه $A[۱,۳]=۱۵$ به این معنی است که کوتاهترین مسیر بین راس ۱ و راس ۳ که مسیر $<۱,۲,۴,۳>$ است دارای طول ۱۵ می‌باشد و به همین ترتیب.

اگر بخواهیم این مساله را با بررسی همه حالات حل کنیم مرتبه زمانی، بدتر از نمایی $(O(n!))$ خواهد بود ولی با برنامه‌ریزی پویا الگوریتمی ارائه می‌دهیم از مرتبه $O(n^۳)$.

لازم به ذکر است که در این مساله، اصل بهینگی برقرار است. زیرا اگر مسیر i به j که از k می‌گذرد کوتاهترین باشد آنگاه مسیر i به k و k به j نیز کوتاهترین است.

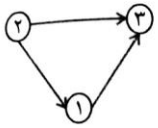
برای حل مساله، ماتریس‌های $A_۰$ تا A_n را بدست می‌آوریم. $A_۰$ همان ماتریس هزینه یعنی C می‌باشد و A_k از روی $A_{k-۱}$ بدست می‌آید. و A_n همان A یعنی جواب می‌باشد.

برنامه نویسی پویا
(Dynamic Programming)

- الگوریتم فلوید: پیدا کردن کوتاهترین مسیر بین نقاط

درایه‌های A_k طول کوتاهترین مسیرهایی را می‌دهند که فقط رئوس $\{1,2,...,k\}$ به عنوان رئوس میانی می‌توانند باشند.

مثلاً $A_1[2,3]$ یعنی طول کوتاهترین مسیر از راس ۲ به راس ۳ به شرطی که این مسیر فقط بتواند از راس ۱ عبور کند. می‌توان گفت



$$A_1[2,3] = \min\left(A_0[2,3]\ ,\ A_0[2,1] + A_0[1,3]\right)$$

زیرا کوتاهترین مسیر از راس ۲ به راس ۳، به شرطی که فقط راس ۱ بتواند در مسیر حضور داشته باشد یا مسیر مستقیم $<2,3>$ می‌باشد که هزینه آن $A_0[2,3]$ $(C[2,3])$ است یا مسیر $<2,1,3>$ می‌باشد که هزینه آن مجموع $A_0[2,1] + A_0[1,3]$ است.

پس از اینکه همه درایه‌های ماتریس A_1 محاسبه شد، ماتریس A_2 را محاسبه می‌کنیم یعنی تاثیر راس ۲ را روی تمام مسیرها محاسبه می‌کنیم. بنابراین $A_2[i,j]$ یعنی کوتاهترین مسیر از راس i به راس j به شرطی که رئوس $\{1,2\}$ بتوانند رئوس واسط باشند.

واضح است که

$$A_2[i,j] = \min\left(\underbrace{A_1[i,j]}_{\text{کوتاهترین مسیر از } i \text{ به } j \text{ با واسطه راس ۱}}\ ,\ \underbrace{A_1[i,2]}_{\text{کوتاهترین از } i \text{ به ۲ به واسطه راس ۱}} + \underbrace{A_1[2,j]}_{\text{کوتاهترین از ۲ به } j \text{ با واسطه راس ۱}}\right)$$

به طور کلی می‌توان فرمول بهینه‌سازی زیر را ارائه داد:

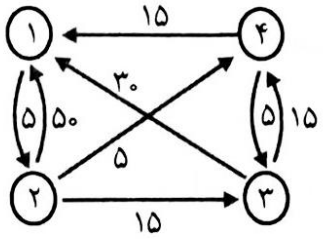
$$A_k[i,j] = \min\left(A_{k-1}[i,j]\ ,\ A_{k-1}[i,k] + A_{k-1}[k\ ,\ j]\right)$$

برنامه نویسی پویا
(Dynamic Programming)

- الگوریتم فلوید: پیدا کردن کوتاهترین مسیر بین نقاط
- ماتریس A_K :یعنی برای پیدا کردن کوتاهترین مسیر بین دو گره در صورت نیاز از گره V_K نیز به عنوان واسط استفاده کنیم.
- در نهایت A_n کوتاهترین مسیر بین گروه‌ها را مشخص می‌کند (طول مسیر)
- نکته : در ماتریس A_K مقدار قطر اصلی همیشه صفر هست.
- نکته : در ماتریس A_K مقدارسطر و ستون Kام از ماتریس A قبلی کپی می شود.

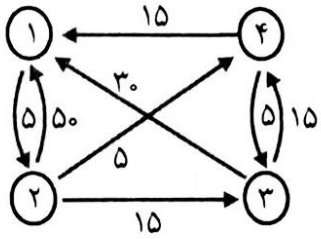
برنامه نویسی پویا
(Dynamic Programming)

- **مثال :** مراحل الگوریتم فلوید را روی گراف زیر نشان دهید.



برنامه نویسی پویا
(Dynamic Programming)

- **مثال :** مراحل الگوریتم فلوید را روی گراف زیر نشان دهید.



پاسخ:

$$A_0=C=\begin{array}{c} \\ ۱ \\ ۲ \\ ۳ \\ ۴ \end{array}\begin{array}{c} \begin{array}{cccc} ۱ & ۲ & ۳ & ۴ \end{array} \\ \begin{bmatrix} ۰ & ۵ & \infty & \infty \\ ۵۰ & ۰ & ۱۵ & ۵ \\ ۳۰ & \infty & ۰ & ۱۵ \\ ۱۵ & \infty & ۵ & ۰ \end{bmatrix} \end{array} \Rightarrow A_1=\begin{array}{c} \\ ۱ \\ ۲ \\ ۳ \\ ۴ \end{array}\begin{array}{c} \begin{array}{cccc} ۱ & ۲ & ۳ & ۴ \end{array} \\ \begin{bmatrix} ۰ & ۵ & \infty & \infty \\ ۵۰ & ۰ & ۱۵ & ۵ \\ ۳۰ & ۳۵ & ۰ & ۱۵ \\ ۱۵ & ۲۰ & ۵ & ۰ \end{bmatrix} \end{array} \Rightarrow A_2=\begin{array}{c} \\ ۱ \\ ۲ \\ ۳ \\ ۴ \end{array}\begin{array}{c} \begin{array}{cccc} ۱ & ۲ & ۳ & ۴ \end{array} \\ \begin{bmatrix} ۰ & ۵ & ۲۰ & ۱۰ \\ ۵۰ & ۰ & ۱۵ & ۵ \\ ۳۰ & ۳۵ & ۰ & ۱۵ \\ ۱۵ & ۲۰ & ۵ & ۰ \end{bmatrix} \end{array}$$

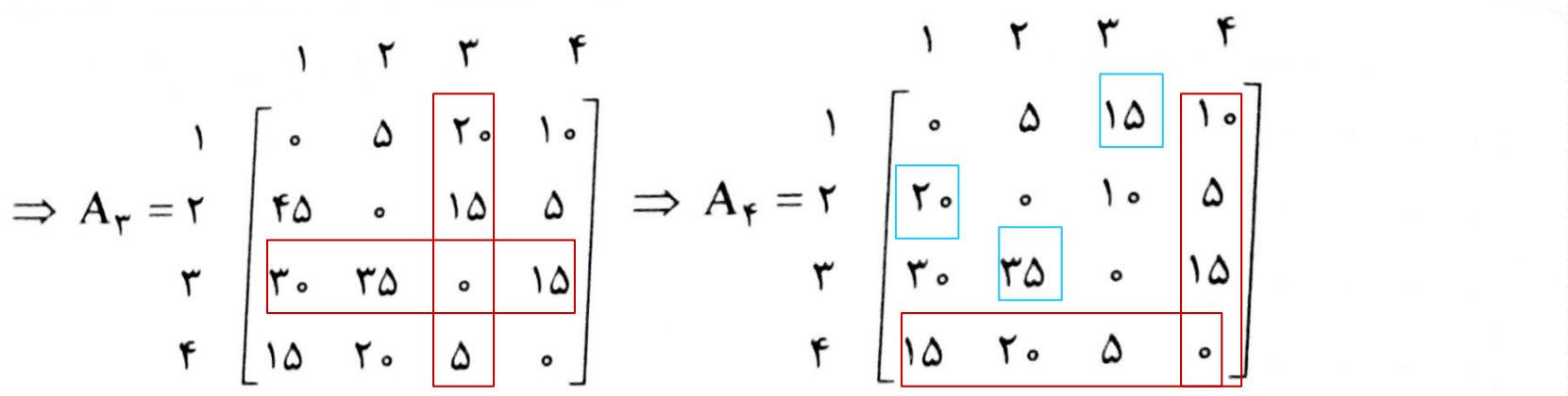
$A_1[3][2] = \min(A_0[3][2] , A_0[3][1] + A_0[1][2])=\min(\infty , 30+5)=35$

$A_1[4][3] = \min(A_0[4][3] , A_0[4][1] + A_0[1][3])=\min(5 , 15+ \infty)=5$

...

برنامه نویسی پویا
(Dynamic Programming)

- الگوریتم فلوید: پیدا کردن کوتاهترین مسیر بین نقاط



$A_4[1][3] = \min(A_3[1][3], A_3[1][4] + A_3[4][3])$=min(20 , 10+5)=15

$A_4[2][1] = \min(A_3[2][1], A_3[2][4] + A_3[4][1])$=min(45 , 5+15)=20

$A_4[3][2] = \min(A_3[3][2], A_3[3][4] + A_3[4][2])$=min(35 , 15+20)=35

……

برنامه نویسی پویا
(Dynamic Programming)

- الگوریتم فلوید: پیدا کردن کوتاهترین مسیر بین نقاط

```
void floyd(int A[][n])
{
        int i, j,k;
        for (k =1; k <= n; k++)
                for (i = 1; i <= n; i++)
                        for (j = 1; j <= n; j++)
                                A[i][j] =min( A[i][j] , A[i][k]+ A[k][j]);

}
```

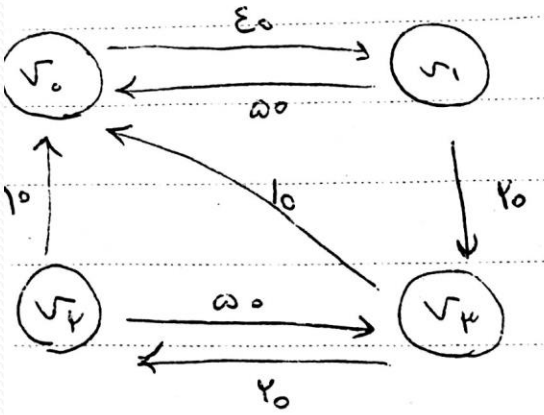
برنامه نویسی پویا
(Dynamic Programming)

- ماتریس A_n تنها فاصله کوتاهترین مسیر بین گره‌های i و j را نشان می دهد ولی نحوه به دست آوردن این کوتاهترین مسیر را تعیین نمی کند. برای حل این مشکل ماتریس کمکی به نام ماتریس مسیر(**path**) استفاده می شود که این ماتریس نیز n×n بوده و مقدار اولیه عناصر آن ۱- است و با تغییر دادن الگوریتم فلوید به صورت زیر این ماتریس به دست می‌آید

```
void floyd(int A[][n])
{
        int i, j,k;
        for (k =1; k <= n; k++)
                for (i = 1; i <= n; i++)
                        for (j = 1; j <= n; j++)
                                if (A[i][j] > A[i][k] + A[k][j])
                                {
                                        A[i][j] = A[i][k] + A[k][j];
                                        path[i][j] = k;
                                }
}
```

برنامه نویسی پویا
(Dynamic Programming)

- خواندن ماتریس مسیر

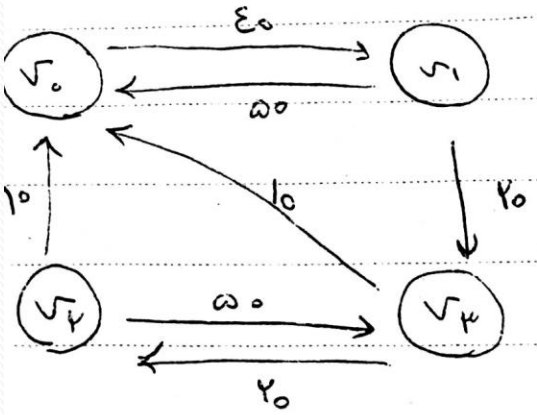


path

	۰	۱	۲	۳
۰	-۱	-۱	۳	۱
۱	۳	-۱	۳	-۱
۲	-۱	۰	-۱	-۱
۳	-۱	۰	-۱	-۱

برنامه نویسی پویا
(Dynamic Programming)

- خواندن ماتریس مسیر



path

	٠	١	٢	٣
٠	-١	-١	٣	١
١	٣	-١	٣	-١
٢	-١	٠	-١	-١
٣	-١	٠	-١	-١

Path[0][2]=3

$V_0 \rightarrow V_2$

$V_0 \rightarrow V_3 \rightarrow V_2$

Path[0][3]=1

Path[3][2]= -1

$V_0 \rightarrow V_1 \rightarrow V_3 \rightarrow V_2$ √

Path[0][1]= -1

Path[1][3]= -1

برنامه نویسی پویا
(Dynamic Programming)

- نکته: الگوریتم هایی مانند فلوید که از بین جواب های مختلف برای یک مسئله بهترین گزینه را انتخاب میکند جزء مسائل بهینه سازی است
- در مسائل بهینه سازی به دنبال مجموعه ای هستیم که بهترین جواب را حاصل می کند.

برنامه نویسی پویا
(Dynamic Programming)

- ضرب زنجیره ای ماتریس ها

همانطور که می‌دانیم ضرب ماتریس‌ها دارای خاصیت شرکت‌پذیری می‌باشد ولی جابجایی ندارد. علاوه بر این دو ماتریس A و B فقط به شرطی به صورت $A \times B$ قابل ضرب هستند که بُعد وسط آنها یکسان باشد یعنی تعداد ستون‌های A با تعداد سطرهای B مساوی باشد. علاوه بر این می‌دانیم که ضرب $A_{m\times n} \times B_{n\times p}$ نیاز به $m \times n \times p$ عمل ضرب دارد.

مثال : ضرب سه ماتریس $A_{۷\times ۵}$ و $B_{۵\times ۳}$ و $C_{۳\times ۲}$ به صورت ABC حداقل چند عمل ضرب نیاز دارد؟

پاسخ: سه ماتریس را می‌توان به ۲ صورت (AB)C و A(BC) در هم ضرب کرد. بنابراین هر دو حالت را بررسی می‌کنیم:

$$(A_{۷\times ۵} \times B_{۵\times ۳}) \times C_{۳\times ۲} \Rightarrow \text{تعداد ضرب} = ۷\times ۵\times ۳ + ۷\times ۳\times ۲ = ۱۴۷$$

$$A_{۷\times ۵} \times (B_{۵\times ۳} \times C_{۳\times ۲}) \Rightarrow \text{تعداد ضرب} = ۵\times ۳\times ۲ + ۷\times ۵\times ۲ = ۱۰۰$$

برنامه نویسی پویا
(Dynamic Programming)

- ضرب زنجیره ای ماتریس ها

مثال : چه رابطه‌ای بین w و x و y و z برقرار باشد تا ضرب ۳ ماتریس $A_{w\times x}$ و $B_{x\times y}$ و $C_{y\times z}$ به صورت (AB)C از A(BC) بهتر باشد یعنی تعداد عملیات ضرب کمتری داشته باشد؟

پاسخ:

$A_{w\times x} \times B_{x\times y} \Rightarrow$ تعداد ضرب $= wxy$

$(AB)_{w\times y} \times C_{y\times z} \Rightarrow$ تعداد ضرب $= wyz$

$(AB)C \Rightarrow$ تعداد ضرب $= wxy + wyz$

به همین ترتیب تعداد ضرب $A(BC)$ برابر $xyz + wxz$ می‌باشد بنابراین

$$wxy + wyz < xyz + wxz$$

طرفین رابطه را به wxyz تقسیم می‌کنیم:

$$\frac{1}{z} + \frac{1}{x} < \frac{1}{w} + \frac{1}{y}$$

برنامه نویسی پویا
(Dynamic Programming)

- ضرب زنجیره ای ماتریس ها

حال می‌خواهیم ترتیب ضرب بهینه n ماتریس $M = M_1 \times M_2 \times \ldots \times M_n$ را پیدا کنیم، یعنی بررسی کنیم چگونه پرانتزگذاری کنیم که تعداد عمل ضرب عددی، مینیمم شود.

می‌خواهیم با مرتبه چند جمله‌ای و با استفاده از برنامه‌ریزی پویا، ضرب بهینه را بیابیم. روشن است که اصل بهینگی برای ضرب ماتریس‌ها برقرار است زیرا اگر مثلاً ضرب $(M_1 M_2)(M_3 M_4)$ بهینه باشد باید $M_1 M_2$ و $M_3 M_4$ بهینه باشند.

در ادامه بحث فرض می‌کنیم ماتریس M_i دارای ابعاد $r_{i-1} \times r_i$ $(i = 1, 2, \ldots, n)$ می‌باشد مثلاً ماتریس M_1 یک ماتریس $r_0 \times r_1$ می‌باشد. همچنین فرض می‌کنیم m_{ij} حداقل تعداد عملیات ضرب عددی برای محاسبه حاصل $M_i \times M_{i+1} \ldots \times M_j$ می‌باشد.

مثلاً m_{13} یعنی حداقل تعداد ضرب ماتریس‌های $M_1 \times M_2 \times M_3$. واضح است که $m_{ii} = 0$ می‌باشد.

برنامه نویسی پویا
(Dynamic Programming)

- ضرب زنجیره ای ماتریس ها

برای محاسبه m_{ij} فرض کنید ضرب $M_i \times ... \times M_j$ را به صورت زیر نوشته‌ایم: (هر طور پرانتزگذاری کنیم یک نقطه شکست مثل k وجود دارد)

$$(M_i \times ... \times M_k).(M_{k+1} \times ... \times M_j) \quad (1)$$

بنابراین m_{ij} از رابطه زیر بدست می‌آید:

$$m_{ij} = \min_{i \le k < j} \left(m_{ik} + m_{k+1,j} + r_{i-1} \times r_k \times r_j \right) \quad (2)$$

که در این رابطه m_{ik} تعداد حداقل اعمال ضرب لازم برای محاسبه

$$M_i \times M_{i+1} \times ... \times M_k \quad (3)$$

می‌باشد و $m_{k+1,j}$ حداقل ضرب لازم برای محاسبه

$$M_{k+1} \times M_{k+2} \times ... \times M_j \quad (4)$$

می‌باشد و جمله $r_{i-1} \times r_k \times r_j$ تعداد ضرب لازم برای ضرب دو ماتریس حاصل از هر قسمت (۳) و (۴) می‌باشد.

برنامه نویسی پویا
(Dynamic Programming)

- ضرب زنجیره ای ماتریس ها

با توجه به فرمول (۲) مراحل زیر را انجام می‌دهیم تا به m_{1n} یعنی جواب برسیم:

مرحلـه ۰: هـیچ عمـل ضـرب ماتریـسی. ایـن مرحلـه معـادل یادداشـت $m_{ii} = ۰$ ، $i = ۱,۲,...,n$ می‌باشد.

مرحله ۱: یک عمل ضـرب ماتریـسی. در ایـن مرحلـه کلیـه مقـادیر $m_{i,i+۱}$ و $i = ۱,۲,...,n-۱$ را محاسبه و در جدول می‌گذاریم.

مرحله ۲: دو عمل ضرب ماتریسی. در این مرحله کلیه مقادیر $m_{i,i+۲}$ و $i = ۱,۲,...,n-۲$ را محاسـبه می‌کنیم.

⋮

مرحله n-۱ : n-۱ عمل ضرب ماتریسی. در این مرحله m_{1n} را محاسبه می‌کنیم.

برنامه نویسی پویا
(Dynamic Programming)

مثال : فرض کنید ماتریس‌های M_1 و M_2 و M_3 و M_4 به ترتیب ۱۰×۲۰ و ۲۰×۵۰ و ۵۰×۱ و ۱×۱۰۰ باشند. کمیت‌های گفته شده را محاسبه می‌کنیم:

$$m_{11} = m_{22} = m_{33} = m_{44} = 0$$

$$\begin{cases} m_{12} = \min(m_{11} + m_{22} + 10\times20\times50) = 10000 \\ m_{23} = \min(m_{22} + m_{33} + 20\times50\times1) = 1000 \\ m_{34} = \min(m_{33} + m_{44} + 50\times1\times100) = 5000 \end{cases}$$

$$\begin{cases} m_{13} = \min(m_{11} + m_{23} + 10\times20\times1, m_{12} + m_{33} + 10\times50\times1) = 1200 \\ m_{24} = \min(m_{22} + m_{34} + 20\times50\times100, m_{23} + m_{44} + 20\times1\times100) = 3000 \end{cases}$$

$$m_{14} = \min(m_{11} + m_{24} + 10\times20\times100, m_{12} + m_{34} + 10\times50\times100, m_{13} + m_{44} + 10\times1\times100) = 2200$$

خلاصه عملیات فوق در جدول زیر آمده است:

مرحله ۰	$m_{11} = 0$	$m_{22} = 0$	$m_{33} = 0$	$m_{44} = 0$
مرحله ۱	$m_{12} = 10000$	$m_{23} = 1000$	$m_{34} = 5000$	
مرحله ۲	$m_{13} = 1200$	$m_{24} = 3000$		
مرحله ۳	$m_{14} = 2200$			

برنامه نویسی پویا
(Dynamic Programming)

مثال : فرض کنید ماتریس‌های M_1 و M_2 و M_3 و M_4 به ترتیب ۱۰×۲۰ و ۲۰×۵۰ و ۵۰×۱ و ۱×۱۰۰ باشند. کمیت‌های گفته شده را محاسبه می‌کنیم:

$$m_{11} = m_{22} = m_{33} = m_{44} = 0$$

r[10 , 20 , 50 , 1 , 100]

m[i][j]=min(m[i][k]+m[k+1][j]+$r_{i-1} * r_k * r_j$)

K=1 $m_{12} = \min(m_{11} + m_{22} + 10\times20\times50) = 10000$

K=2 $m_{23} = \min(m_{22} + m_{33} + 20\times50\times1) = 1000$

K=3 $m_{34} = \min(m_{33} + m_{44} + 50\times1\times100) = 5000$

K=1 K=2

$$\begin{cases} m_{13} = \min(m_{11} + m_{23} + 10\times20\times1, m_{12} + m_{33} + 10\times50\times1) = 1200 \\ m_{24} = \min(m_{22} + m_{34} + 20\times50\times100, m_{23} + m_{44} + 20\times1\times100) = 3000 \end{cases}$$

$$m_{14} = \min(m_{11} + m_{24} + 10\times20\times100, m_{12} + m_{34} + 10\times50\times100, m_{13} + m_{44} + 10\times1\times100) = 2200$$

K=1 K=2 K=3

خلاصه عملیات فوق در جدول زیر آمده است:

مرحله ۰	$m_{11} = 0$	$m_{22} = 0$	$m_{33} = 0$	$m_{44} = 0$
مرحله ۱	$m_{12} = 10000$	$m_{23} = 1000$	$m_{34} = 5000$	
مرحله ۲	$m_{13} = 1200$	$m_{24} = 3000$		
مرحله ۳	$m_{14} = 2200$			

برنامه نویسی پویا
(Dynamic Programming)

الگوریتم optmm تعداد حداقل ضرب را برای n ماتریس بدست می‌آورد:

```
int optmm (int m[۱..n][۱..n] , int n)
 {
   int i, j, L;
   for (i = ۱;i <= n ; i + +)m[i][i] = ۰;
    for (L = ۱;L <= n;L + +)//
    for (i = ۱; i <= n – L; i + +)
    {
       j = i + L;
       m[i][j] = min {m[i][k] + m[k + ۱][j] + r_{i–۱} * r_k * r_j}
                 i≤k<j
    }
   return m [1][n];
}
```

بهتر است $L <= n - ۱$ باشد

الگوریتم فوق از مرتبه $O(n^۳)$ می‌باشد. زیرا تعداد اجرای جمله اصلی که همان مینیمم‌گیری است برابر است با:

$$\sum_{L=۱}^{n} \sum_{i=۱}^{n-L} \sum_{k=i}^{i+L-۱} ۱ = \frac{n^۲(n+۱)}{۲} - \frac{n(n+۱)(۲n+۱)}{۶} = \frac{n^۳ - n}{۶}$$

برنامه نویسی پویا
(Dynamic Programming)

آرایه m برای مثال قبل عبارت است از:

$$m = \begin{array}{c} \\ ۱ \\ ۲ \\ ۳ \\ ۴ \end{array} \begin{array}{c} \begin{array}{cccc} ۱ & ۲ & ۳ & ۴ \end{array} \\ \begin{bmatrix} ۰ & ۱۰۰۰۰ & ۱۲۰۰ & ۲۲۰۰ \\ & ۰ & ۱۰۰۰ & ۳۰۰۰ \\ & & ۰ & ۵۰۰۰ \\ & & & ۰ \end{bmatrix} \end{array}$$

که $m[۱][۴] = ۲۲۰۰$ تعداد حداقل ضرب ۴ ماتریس گفته شده را نشان می‌دهد.

برای تعیین ترتیب بهینه ضرب ماتریس‌ها آرایه دیگری به نام p که ابعاد آن برابر ابعاد m است تعریف می‌کنیم و هر بار مقدار k را که به ازای آن m_{ij} مینیمم شد در محل P_{ij} ذخیره می‌کنیم.

در مثال ضرب $M_۱ \times M_۲ \times M_۳ \times M_۴$ دیدیم که $m_{۱۴}$ از رابطه زیر بدست آمد:

$$m_{۱۴} = \min\left(\underbrace{۲۳۰۰۰}_{k=۱}, \underbrace{۶۵۰۰۰}_{k=۲}, \underbrace{۲۲۰۰}_{k=۳} \right) = ۲۲۰۰$$

یعنی مینیمم ضرب $m_{۱۴}$ به ازای $k = ۳$ بدست آمد یعنی: $(M_۱ \times M_۲ \times M_۳)$. $(M_۴)$

برنامه نویسی پویا
(Dynamic Programming)

حال برای آنکه ببینیم نقطه شکست $M_1 \times M_2 \times M_3$ کجاست، به m_{13} نگاه می‌کنیم:

$$m_{13} = \min\left(\underbrace{۱۲۰۰}_{k=1} , \underbrace{۱۰۵۰۰}_{k=2} \right) = ۱۲۰۰$$

پس نقطه شکست $M_1 M_2 M_3$، $k = ۱$ (M_1) $.(M_2 M_3)$ می‌باشد یعنی ضرب بهینه به صورت $(M_1.(M_2.M_3)).M_4$ می‌باشد.

ماتریس p برای این مثال به شکل زیر است:

$$p = \begin{array}{c} \\ ۱ \\ ۲ \\ ۳ \\ ۴ \end{array} \begin{array}{c} \begin{array}{cccc} ۱ & ۲ & ۳ & ۴ \end{array} \\ \begin{bmatrix} ۱ & ۱ & ۳ \\ & ۲ & ۳ \\ & & ۳ \\ & & \end{bmatrix} \end{array}$$

برنامه نویسی پویا
(Dynamic Programming)

- **مثال :** می‌خواهیم ۶ ماتریس $M_1M_2M_3M_4M_5M_6$ را در هم ضرب کنیم. ماتریس p به صورت زیر بدست آمده است. ترتیب بهینه ضرب را مشخص کنید.

$$p=\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}\begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ & & 2 & 3 & 4 & 5 \\ & & & 3 & 4 & 5 \\ & & & & 4 & 5 \\ & & & & & 5 \\ & & & & & \end{bmatrix} \end{array}$$

پاسخ: عنصر $p[1,6]=1$ یعنی نقطه شکست برای ضرب $M_1..M_6$، نقطه ۱ است پس:

$$(M_1)(M_2\,M_3\,M_4\,M_5\,M_6)$$

حال باید $p[2,6]$ را بررسی کنیم که مقدار آن ۵ می‌باشد پس نقطه شکست $M_2..M_6$ نقطه ۵ است یعنی:

$$(M_1)((M_2\,M_3\,M_4\,M_5)(M_6))$$

حال $p[2,5]=4$ پس:

$$(M_1)((M_2\,M_3\,M_4)(M_5)(M_6))$$

و با توجه به اینکه $p[2,4]=3$ در نتیجه:

$$M_1(((M_2\,M_3)M_4)M_5)M_6$$

ترتیب بهینه ضرب است.