

## فصل ۴

### برنامه‌نویسی پویا

#### ۱- مقدمه

برنامه‌نویسی پویا، یک روش دیگر برای طراحی الگوریتم‌ها است و زمانی که راه حل مساله‌ای به صورت اتخاذ رشته‌ای از تصمیمات مطرح است، می‌توان از آن استفاده کرد. معمولاً "رشته تصمیمات اتخاذ شده باید منجر به بهینه‌سازی کمیتی شوند. چنین رشته‌ای را رشته بهینه خواهیم گفت. برای بعضی از مسایل، یک رشته بهینه را می‌توان با اتخاذ تصمیمات به صورت مرحله به مرحله پیدا کرد. این مطلب در همه مسایلی که به روش حریصانه قابل حل باشند، صادق است، ولی برای مسایل بسیاری، این تصمیمات مرحله به مرحله منجر به رشته بهینه نمی‌شود.

یک راه حل برای مسایلی که امکان تصمیم‌گیری مرحله به مرحله برای رسیدن به یک رشته بهینه از تصمیمات وجود ندارد، بررسی همه رشته تصمیمات ممکن است. می‌توان همه رشته تصمیمات را بررسی کرده و بهترین را انتخاب کرد. برنامه‌نویس پویا، اغلب به شکل قابل ملاحظه‌ای تعداد حالات مورد بررسی را با حذف بعضی از رشته‌های تصمیم که منجر به جواب بهینه نخواهد شد، کاهش می‌دهد.

در این روش، یک رشته بهینه از تصمیمات با توجه به اصل بهینه‌سازی حاصل می‌شود. طبق این اصل، یک رشته بهینه از تصمیمات این خاصیت را دارد که حالت و تصمیم‌گیری مرحله اول هر چه که باشد، تصمیمات باقیمانده، با توجه به این حالت و تصمیم‌گیری باید یک رشته بهینه تشکیل دهند.

مثال ۱: فرض کنید که در یک گراف وزن‌دار، هدف پیدا کردن کوتاه‌ترین مسیر از یک راس دیگر مثل زاست. فرض کنید  $i_1, i_2, \dots, i_k, j$  کوتاه‌ترین مسیر (رشته بهینه) از راس  $i$  به راس  $j$  باشد. در اینجا تصمیم اولیه، رفتن از راس  $i$  به راس  $i_1$  بوده و بعد از اتخاذ این تصمیم، حالت مسالة به وسیله راس  $i_1$  تعریف می‌شود. حال باید مسیری از  $i_1$  به  $j$  پیدا کرد. بدیهی است که رشته  $i_1, i_2, \dots, i_k$  و  $j$  باید کوتاه‌ترین مسیر از  $i$  به  $j$  باشد، اگر  $i_1, i_2, \dots, i_k, j$  کوتاه‌ترین مسیر از  $i$  به  $j$  باشد، در غیر این صورت  $i_1, i_2, \dots, i_k, j$  مسیری از  $i$  به  $j$  است که طول آن کوتاه‌تر از مسیر  $i_1, i_2, \dots, i_k, j$  خواهد بود. که خلاف فرض است.

## مثال ۲: مساله کوله‌پشتی 0/1

این مساله، مشابه مساله کوله‌پشتی مطرح شده در فصل روش‌های حریصانه است، با این تفاوت که در اینجا  $x_i$  ها فقط می‌توانند یکی از دو مقدار صحیح ۰ یا ۱ را به خود بگیرند.

با استفاده از  $(y, j)$  برای نمایش Knap (1, j) نمایش

$$\text{ماکزیمم کردن} \sum_{i=1}^j p_i x_i$$

$$\sum_{i=1}^j x_i w_i \leq y \quad x_i = 0 \text{ یا } 1$$

مساله کوله‌پشتی به صورت Knap (1, n, M) نمایش داده می‌شود. حال فرض کنید که  $y_1, y_2, \dots, y_n$  یک رشته بهینه از مقادیر ۰ یا ۱ به ترتیب برای  $x_1, x_2, \dots, x_n$  باشند، اگر  $y_1 = 0$ ، آن‌گاه بدیهی است که  $y_2, \dots, y_n$  باید یک رشته بهینه برای مساله Knap (2, n, M) تشکیل دهد، زیرا، در غیر این صورت  $y_1, y_2, \dots, y_n$  یک رشته بهینه برای knap(2, n, M) نخواهد بود. اگر  $y_1 = 1$  آن‌گاه  $y_2, \dots, y_n$  باید یک رشته بهینه برای knap (2, n, M-w<sub>1</sub>) تشکیل دهند، در غیر این صورت دنباله‌ای از ۰ و ۱ ها مثل

$$\sum_{i=2}^n z_i p_i > \sum_{i=2}^n y_i p_i \quad \sum_{i=1}^n z_i w_i \leq M - w_1 \quad z_1, z_2, \dots, z_n$$

برای مساله (1) با مقدار بیشتری از جواب بهینه خواهد بود که خلاف فرض است.

## مثال ۳: مساله کوله‌پشتی 0/1

حال با فرض آنکه  $g_i(y)$  مقدار بهینه حاصل برای مساله کوله‌پشتی (j+1, n, y) باشد، در این صورت بدیهی است که  $g(M)$  مقدار بهینه برای مساله (2, n, M) خواهد بود. با توجه به اصل بهینه‌سازی خواهیم داشت:

$$g(M) = \max \{g_1(M); g_1(M - w_1) + p_1\} \quad (2)$$

اصل بهینه‌سازی را می‌توان نسبت به حالات و تصمیمات فی‌مابین نیز تعمیم داد.

## مثال ۴: مساله کوله‌پشتی 0/1

فرض کنید  $y_1, y_2, \dots, y_n$  یک جواب بهینه برای مساله knap(1, n, M) باشد، برای هر  $1 \leq j \leq n$  و همچنین  $y_{j+1}, \dots, y_n$  باید به ترتیب راه حل‌های بهینه برای مسایل knap(j+1, n, M -  $\sum_{i=1}^j y_i w_i$ ) و knap(1, j,  $\sum_{i=1}^j y_i w_i$ ) باشند.

با توجه به مطالب گفته شده، فرمول 2 را می‌توان به صورت تعمیم یافته زیر نوشت:

$$g_i(y) = \max \{g_{i+1}(y), g_{i+1}(y - w_{i+1}) + p_{i+1}\} \quad (3)$$

رابطه بازگشتی 3 را می‌توان با توجه به این که  $g_n(y) = 0$  و برای  $y < 0$   $g_n(y) = -\infty$  است، حل کرد. از روی  $g_n(y)$  می‌توان  $g_{n-1}(y)$  را به دست آورد. از روی  $g_{n-2}(y)$  را حساب کرد. با تکرار این روند می‌توان  $g_1(y)$  و در نهایت  $g_0(M)$  را از روی فرمول 3 به دست آورد.

مثال ۵: با استفاده از فرمول 3 مساله کوله‌پشتی 0/1 زیر را حل کنید:

$$n = 3$$

$$M = 6$$

$$(w_1, w_2, w_3) = (2, 3, 4) \quad (p_1, p_2, p_3) = (1, 2, 5)$$

حال باید  $(6)_0$  را حساب کنیم:

$$g_0(6) = \text{Max}\{g_1(6), g_1(4)+1\}$$

$$g_1(6) = \text{Max}\{g_2(6), g_2(3)+2\}$$

$$g_2(6) = \text{Max}\{g_3(6), g_3(2)+5\} = \text{Max}\{0, 5\} = 5$$

$$g_2(3) = \text{Max}\{g_3(3), g_3(-1)+5\} = \text{Max}\{0, -\infty\} = 0$$

ولذا

$$g_1(6) = \text{Max}\{5, 2\} = 5$$

به طریق مشابه

$$g_1(4) = \text{Max}\{g_2(4), g_2(4-3)+2\}$$

$$g_2(4) = \text{Max}\{g_3(4), g_3(4-4)+5\} = \text{Max}\{0, 5\} = 5$$

$$g_2(1) = \text{Max}\{g_3(1), g_3(1-4)+5\} = \text{Max}\{0, -\infty\} = 0$$

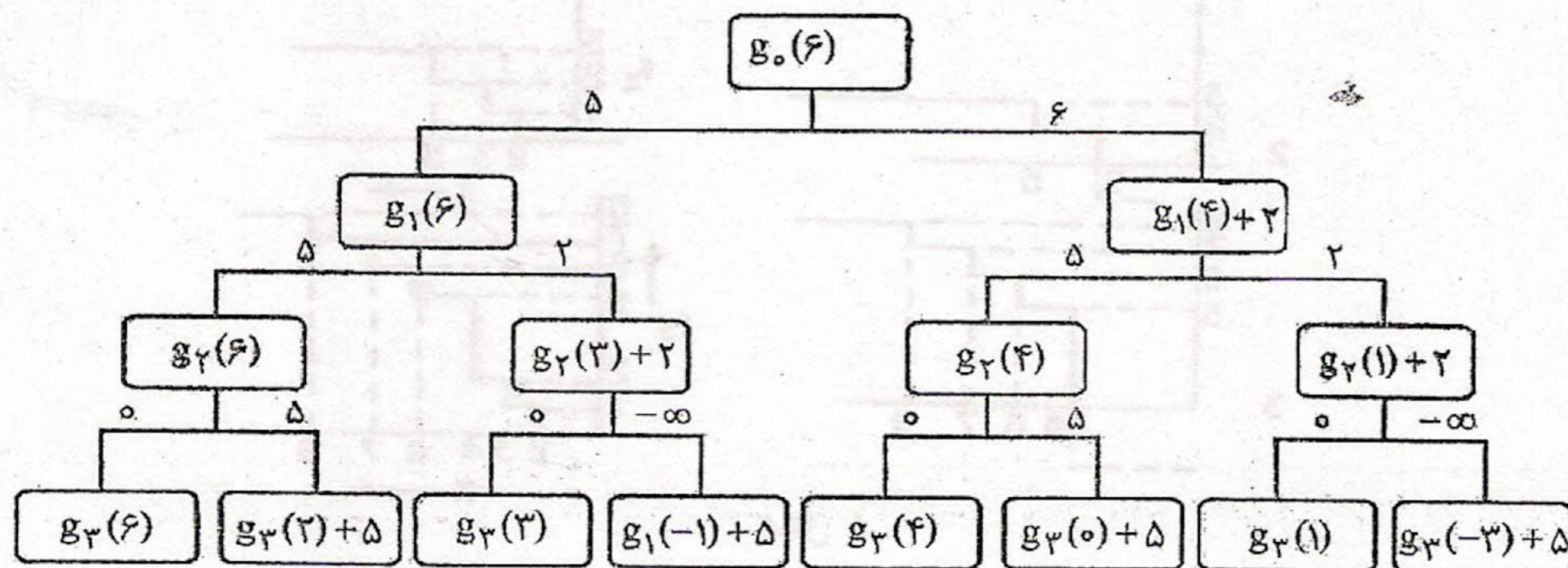
بنابراین

$$g_1(4) = \text{Max}\{5, 0\} = 5$$

در نتیجه

$$g_0(6) = \text{Max}\{5, 5+1\} = [6]$$

اگر  $(y)_0$  را به صورت پردازه‌ای تصور کنیم که فراخوانی آن به صورت  $(M)_0$  خواهد بود، در این صورت مراحل مختلف اجرای پردازه  $(6)_0$  برای مثال بالا به صورت زیر خواهد بود:



فرمول‌بندی یک رابطه بازگشتی در روش برنامه‌نویسی پویا، به یکی از دو روش پسرو یا پیشرو امکان‌پذیر است. فرض کنید که  $x_1, x_2, \dots, x_n$  متغیرهایی باشند که می‌باید برای آن‌ها رشته‌ای از تصمیمات اتخاذ کرد. در روش پیشرو، فرمول‌بندی برای تصمیم  $x_i$  به صورت عبارتی از رشته تصمیمات برای  $x_{i+1}, \dots, x_n$  است. در روش پسرو این فرمول‌بندی به صورت عبارتی از رشته تصمیمات بهینه برای  $x_1, x_2, \dots, x_{i-1}$  است. بنابراین فرمول 3 یک فرمول پیشرو برای حل مسأله کوله‌پشتی است. این مسأله را به صورت پسرو و به صورت زیر نیز می‌توان حل کرد:

### مساله کوله‌پشتی 0/1 (فرمول پیشو)

برای اتخاذ تصمیم بر روی  $x_n$  دو حالت داریم، یا  $x_n = 0$  است که در این حالت ظرفیت باقیمانده کوله‌پشتی  $M$  است و لذا سودی اضافه نمی‌شود، یا  $x_n = 1$ ، که در این حالت ظرفیت باقیمانده کوله‌پشتی  $M - w_n$  و مقدار  $p_n$  به سود حاصل اضافه خواهد شد. تصمیمات باقیمانده  $x_1, x_2, \dots, x_{n-1}$  صرفنظر از تصمیم گرفته شده بر روی  $x_n$ ، باید بهینه باشند. اگر  $(x)_i$  برابر با ارزش ماکریم حاصل از حل مساله کوله‌پشتی 0/1  $f_i(M) = \max\{f_{i-1}(M), f_{i-1}(M - w_i) + p_i\}$

(۴)

و یا بنا به تعمیم اصل بهینه‌سازی، برای یک  $i < n$  دلخواه

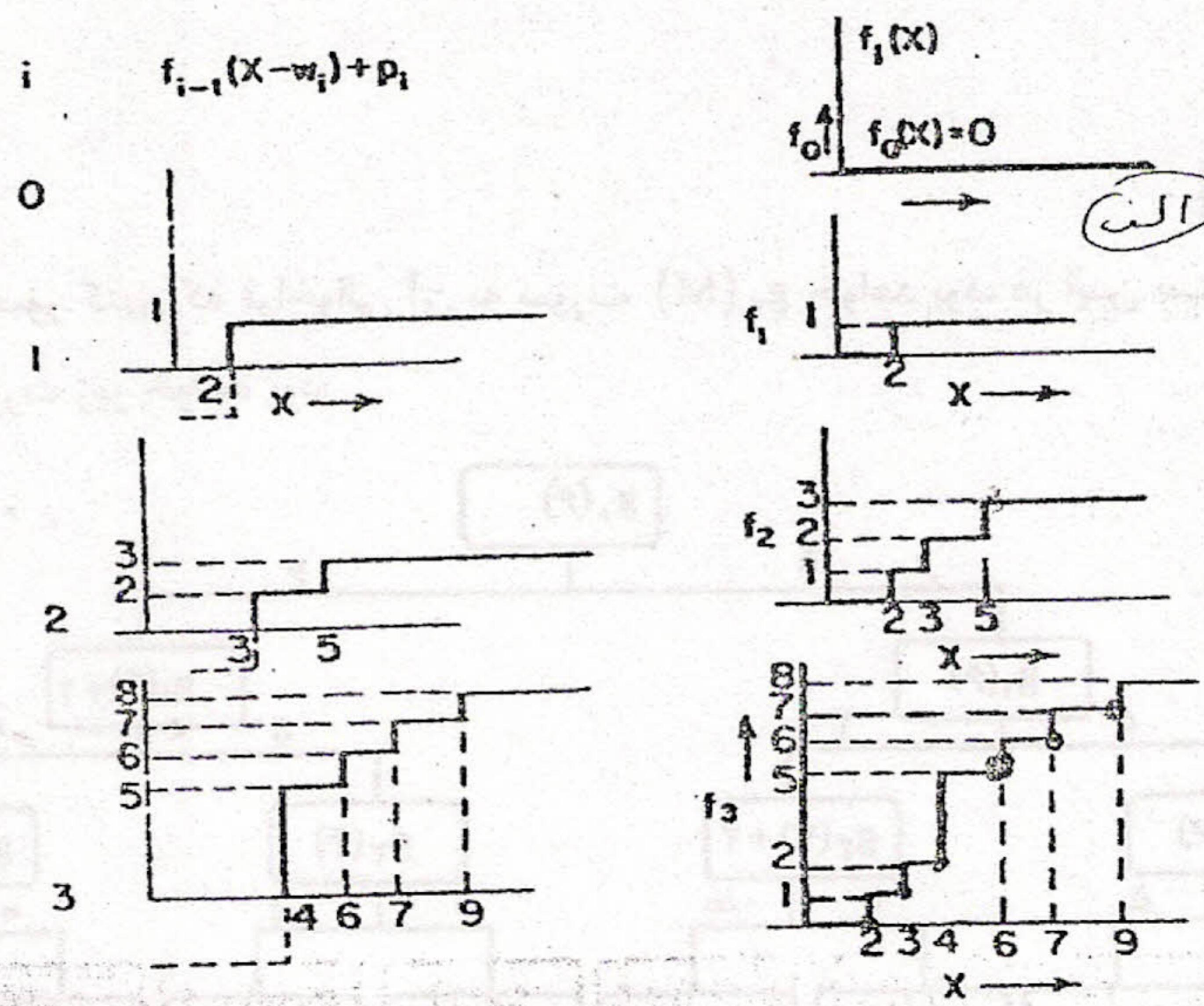
$$f_i(x) = \max\{f_{i-1}(x), f_{i-1}(x - w_i) + p_i\} \quad (5)$$

معادله (5) را می‌توان برای به دست آوردن  $f_n(M)$  که جواب مساله کوله‌پشتی 0/1  $f_n(1, n, M)$  است، به شرح زیر به کار برد.

با توجه به شرط مرزی

$$f_n(x) = 0, \quad x \geq 0$$

$$\forall i, 0 \leq i \leq n, f_i(x) = -\infty, \quad x < 0$$



شکل ۱

ابتدا منحنی  $(x)_0 f_0$  را رسم می‌کنیم (شکل الف)، سپس  $f_1, f_2, \dots, f_n$  را به صورت متوالی از روی فرمول ۵ رسم می‌کنیم. برای رسم  $f_i(x - w_i + p_i)$  کافی است که منحنی  $(x)_{i-1} f_{i-1}$  را به اندازه  $w_i$  به سمت راست و به اندازه  $p_i$  به سمت بالا شیفت بدھیم. منحنی  $(x)_i f_i$  نیز از ماکریم منحنی‌های  $(x)_{i-1} f_{i-1}, (x - w_i) + p_i$  حاصل می‌شود. جواب مثال (5) در نقطه  $x=6$  بر روی منحنی  $(x)_3 f_3$  خواهد بود (شکل ۱).

## نگرش مجموعه‌ای

در اشکال (1) مشاهده می‌شود که  $(x_i, f_i)$  ها، منحنی‌های صعودی و پله‌ای هستند. این منحنی‌ها را می‌شود با زوج‌های  $(x_j, y_j)$  تعریف کرد که در آن  $x$  مقدار  $x$  ای است که برای آن تابع  $(x_i, f_i)$  دارای جهشی برابر  $y_j = f_i(x_j)$  است. اگر  $r$  جهش وجود داشته باشد، آن‌گاه  $r$  زوج  $(x_j, y_j)$  مورد نیاز است. این  $r$  زوج برای تعریف  $(x_i, f_i)$  را می‌توان با مجموعه  $S^i = \{(x_j, y_j) \mid 1 \leq j \leq r\}$  نمایش داد. باید توجه داشت که  $S^0 = \{(0, 0)\}$  و در حالت عمومی، هر زوج  $S^i$  به صورت  $(W_k, P_k)$  است که در آن  $x_j = W_k$  و  $y_j = P_k$ . حال اگر تعریف کنیم:

$$S^i = \{(W + w_i, P + p_i) \mid (W, P) \in S^{i-1}\}$$

در این صورت اعضای مجموعه  $S^{i+1}$  را می‌توان از ادغام عناصر دو مجموعه  $S^i$  و  $S^i$  به دست آورد. اگر در حین ادغام این دو مجموعه، به دو زوج  $(W_k, P_k)$  و  $(W_j, P_j)$  با شرایط  $W_j \geq W_k$  و  $P_j \leq P_k$  برخورد کنیم، آن‌گاه زوج  $(W_j, P_j)$  را با توجه به فرمول (5) باید حذف کنیم. همچنین در حین تولید این مجموعه‌ها، هر زوج  $(W, P)$  با شرط  $W > M$  را باید حذف کرد، زیرا چنین زوج‌هایی مقدار تابع  $f_n$  را برای مقادیر  $M < x$  مشخص می‌کنند، که بزرگ‌تر از ظرفیت کوله‌پشتی بوده و مورد علاقه ما نیستند.

**مثال ۶: مساله کوله‌پشتی**  $(P_1, P_2, P_3) = (1, 2, 5)$  ،  $(W_1, W_2, W_3) = (2, 3, 4)$  را حل کنید.

: حل

$$S^0 = \{(0, 0)\} \quad S^1 = \{(2, 1)\}$$

$$S^1 = \{(0, 0), (2, 1)\} \quad S^2 = \{(3, 2), (5, 3)\}$$

$$S^2 = \{(0, 0), (2, 1), (3, 2), (5, 3)\} \quad S^3 = \{(4, 5), (6, 6)\}$$

$$S^3 = \{(0, 0), (2, 1), (3, 2), (4, 5), (6, 6)\}$$

زمانی که تمام زوج‌های  $(W_j, P_j)$  با شرط  $W_j > M$  از  $S^n$  حذف شدند، در آن صورت مقدار  $f_n(M)$  به وسیله مقدار  $P$  آخرین زوج در  $S^n$  مشخص می‌شود (توجه داشته باشید که  $S^i$  ها مجموعه‌های مرتب شده هستند). ذکر این نکته نیز ضروری است که با به دست آوردن  $S^n$  نه تنها جواب مساله کوله‌پشتی  $\frac{1}{0/1}$ ، knap(1, n, M)، بلکه همه جواب‌های مساله  $(1, n, M)$  را  $0 < x \leq M$ ، knap(1, n, M) هستیم، از بررسی بقیه حالات صرف نظر می‌کنیم.

اما بردار جواب برای مساله knap(1, n, M) را چگونه به دست آوریم؟ اگر  $(W_1, P_1)$  آخرین زوج در  $S^n$  است، آن‌گاه رشته‌ای از  $x_i$  ها (0 یا 1) به گونه‌ای که  $\sum p_i x_i = P_1$  و  $\sum w_i x_i = W_1$  را می‌توان با جستجوی عناصر  $S^i$  ها به طریق زیر به دست آورد:

$$\text{اگر } (W_2, P_2) = (W_1, P_1) \text{ و } x_n = 0 \text{ آن‌گاه } (W_1, P_1) \in S^{n-1}$$

اما اگر  $(W_2, P_2) = (W_1 - w_n, P_1 - p_n)$  ، در این صورت  $(W_1 - w_n, P_1 - p_n) \in S^{n-1}$  و بنابراین  $x_n = 1$  و

$$\text{اگر } x_{n-1} = 1 \text{ در این صورت } (W_2 - w_{n-1}, P_2 - p_{n-1}) \in S^{n-2} \text{ و لذا }$$

این روند را به صورت بازگشتی برای زوج‌های جدید اعمال می‌کنیم.

اگر این مطلب را بر روی مثال قبل اعمال کنیم، نتایج زیر را خواهیم داشت:

$$(6, 6) \in S^3 \text{ ولی } (6, 6) \notin S^2 \Rightarrow x_3 = 1$$

$$(6, 6) - (w_3, p_3) = (6, 6) - (4, 5) = (2, 1)$$

حال برای زوج جدید (2, 1)

$$(2,1) \in S^2, (2,1) \in S^1 \Rightarrow x_2 = 0$$

و سرانجام

$$(2,1) \in S^1 \text{ ولی } (2,1) \notin S^0 \Rightarrow x_1 = 1$$

مطلوب گفته شده، در الگوریتم DKP خلاصه شده‌اند:

Algorithm DPK(P, W, n, M)

$$S^0 = \{(0,0)\}$$

For i=1 to n-1 do

$$S^i = \{(Wx + w[i], Py + p[i]) \mid (Wx, Py) \in S^{i-1} \text{ and } Wx + w[i] \leq M\}$$

$$S^i = \text{Merge\_And\_Purge}(S^{i-1}, S^i)$$

}

$(Wx, Py) = S^n$  آخرین زوج

For i=n Downto 1 do

{

if  $(Wx, Py) \in S^{i-1}$  then  $x_i = 0$

else

$$\{x_i = 1, (Wx, Py) = (Wx, Py) - (w[i], p[i])\}$$

}

End DPK

با توجه به این که  $S^n$  در بدترین حالت می‌تواند  $2^n$  عضو داشته باشد، بنابراین پیچیدگی محاسباتی الگوریتم مذبور  $O(2^n)$  است.

مثال ۷: مسأله کوله‌پشتی  $(W_1, W_2, W_3, W_4, W_5, W_6) = (20, 30, 40, 10, 60, 50)$

$M=95$  و  $n=6$   $(P_1, P_2, P_3, P_4, P_5, P_6) = (10, 15, 20, 50, 25, 60)$  را حل کنید.

حل :

$$S^0 = \{(0,0)\}$$

$$S^1 = \{(20,10)\}$$

$$S^2 = \{(0,0), (20,10)\}$$

$$S^3 = \{(30,15), (50,25)\}$$

$$S^4 = \{(0,0), (20,10), (30,15), (40,20), (50,25), (60,30), (70,35), (90,45)\}$$

$$S^5 = \{(10,50), (30,60), (40,65), (50,70), (60,75), (70,80), (80,85)\}$$

$$S^6 = \{(0,0), (10,50), (30,60), (40,65), (50,70), (60,75), (70,80), (80,85)\}$$

$$S^7 = \{(60,25), (70,75), (90,85)\}$$

$$S^8 = \{(0,0), (10,50), (30,60), (40,65), (50,70), (60,75), (70,80), (80,85)\}$$

$$S^9 = \{(50,60), (60,110), (80,120), (90,125)\}$$

$$S^{10} = \{(0,0), (10,50), (30,60), (40,65), (50,70), (60,110), (80,120), (90,125)\}$$

بنابراین ارزش ماکزیمم حاصل از این مساله کوله‌پشتی برابر 125 است. برای مشخص کردن این‌که کدام یک از بسته‌های انتخاب شده منجر به این ارزش ماکزیمم شده‌اند به صورت زیر عمل می‌کنیم:

$$(90, 125) \notin S^5 \quad \text{بنابراین } x_6 = 1 \quad \text{و} \quad (90, 125) - (50, 60) = (40, 65)$$

$$(40, 65) \in S^4 \quad \text{بنابراین } x_5 = 0$$

$$(40, 65) - (10, 50) = (30, 15) \quad \text{بنابراین } x_4 = 1 \quad \text{و} \quad (40, 65) \notin S^3$$

$$(30, 15) \in S^2 \quad \text{بنابراین } x_3 = 0$$

$$(30, 15) - (30, 15) = (0, 0) \quad \text{بنابراین } x_2 = 1 \quad \text{و} \quad (30, 15) \notin S^1$$

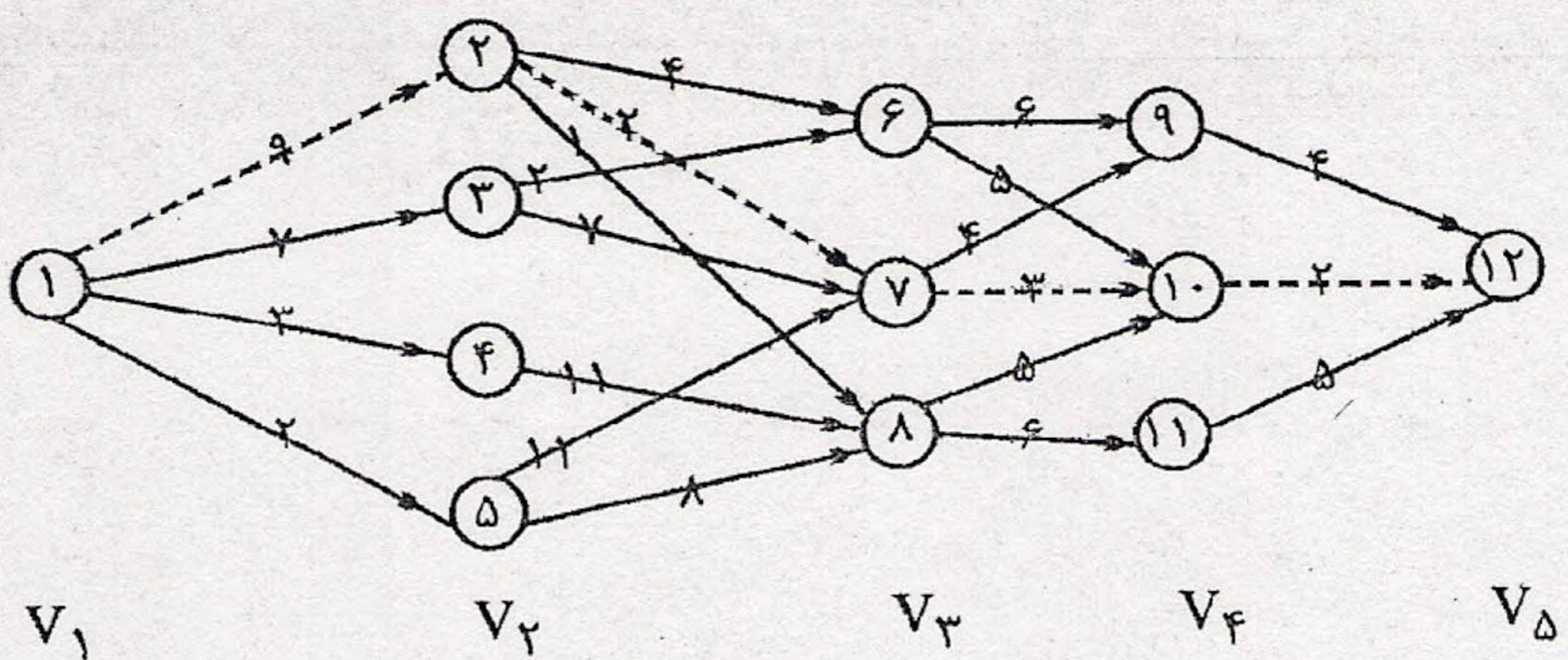
$$(0, 0) \in S^0 \quad \text{بنابراین } x_1 = 0$$

بنابراین بردار جواب برابر  $(x_1, x_2, x_3, x_4, x_5, x_6) = (0, 1, 0, 1, 0, 1)$  است.

## گراف‌های چند مرحله‌ای

یک گراف چند مرحله‌ای  $G = (V, E)$  گرافی است جهت‌دار که در آن رئوس  $V$  به  $k \geq 2$  زیر مجموعه مجزای  $V_i$ ،  $1 \leq i \leq k$ ، افزایش شده و برای هر یال  $(u, v)$  از  $E$ ، ای  $i$  موجود باشد، به گونه‌ای که  $u \in V_i$  و  $v \in V_{i+1}$ . مجموعه‌های  $V_1$  و  $V_k$  هر کدام دارای یک عضو هستند ( $|V_1| = |V_k| = 1$ ). فرض کنید که  $s$  و  $t$  به ترتیب رئوس  $V_1$  و  $V_k$  باشند.  $S$  را راس مبدأ و  $t$  را راس مقصدگوییم. فرض کنید که  $c(i, j)$  هزینه یال  $(i, j)$  باشد. هزینه هر مسیر از  $s$  به  $t$  به صورت مجموع هزینه یال‌های موجود در مسیر تعريف می‌شود. مسأله گراف‌های چند مرحله‌ای یافتن کوتاه‌ترین مسیر از  $s$  به  $t$  است. هر مجموعه  $V_i$  تعريف کننده یک مرحله در گراف است. با توجه به قیود موجود برای یال‌های  $E$ ، هر مسیر از  $s$  به  $t$  باید از مرحله 1 شروع و به مرحله 2 برود، بعد از آن به مرحله 3 و سپس به مرحله 4 و در نهایت به مرحله  $k$  برسد.

شکل زیر نشانگر یک گراف 5 مرحله‌ای است. مسیر با هزینه مینیمم به وسیله خط چین نمایش داده شده است:



با توجه به این که هر مسیر از  $s$  به  $t$  را می‌توان به صورت نتیجه یک رشته از  $k-2$  تصمیم در نظر گرفت، لذا یک گراف چند مرحله‌ای را می‌توان با استفاده از برنامه‌نویسی پویا فرمول‌بندی کرد. تصمیم آن مستلزم آن است که مشخص کنیم کدام راس از مجموعه  $1 \leq i \leq k-2$  باید در مسیر باشد. فرض کنید که  $P(i, j)$  مسیر با هزینه مینیمم از راس  $j$  در  $V_i$  به راس  $t$  باشد، همچنین فرض کنید که  $c(i, j) \cos t(i, j)$  هزینه این مسیر است. با به کارگیری روش پیشرو داریم:

$$\text{cost}(i, j) = \min_{\substack{l \in V_{i+1} \\ (j, l) \in E}} \{c(j, l) + \text{cost}(i+1, l)\} \quad (5)$$

جواب مسأله به صورت  $\text{cost}(1, s)$  است. با توجه به این که  $\text{cost}(k-1, j) = c(j, t)$  و  $\text{cost}(k-1, j) = \infty$  اگر  $c(j, t) \in E$  و لذا  $\text{cost}(1, s) = \text{cost}(1, j)$  را به صورت زیر به دست می‌آوریم. نخست  $\text{cost}(k-2, j) = c(k-2, j)$  را به ازای همه  $j \in V_{k-2}$  حل می‌کنیم. سپس  $\text{cost}(k-3, j)$  را به ازای همه  $j \in V_{k-3}$  به دست می‌آوریم و با ادامه این روند، سرانجام  $\text{cost}(1, s)$  را به دست می‌آوریم. مثال ۸: کوتاه‌ترین مسیر از راس 1 به راس 12 را برای گراف 5 مرحله‌ای شکل بالا به دست آورید.

با به کارگیری روش گفته شده بر روی این گراف نتایج زیر به دست خواهد آمد:

$$\text{cost}(3, 6) = \min \{6 + \text{cost}(4, 9), 5 + \text{cost}(4, 10)\} = 7$$

$$\text{cost}(3, 7) = \min \{4 + \text{cost}(4, 9), 3 + \text{cost}(4, 10)\} = 5$$

$$\text{cost}(3, 8) = \min \{5 + \text{cost}(4, 10), 6 + \text{cost}(4, 11)\} = 7$$

$$\text{cost}(2, 2) = \min \{4, \text{cost}(3, 6), 2 + \text{cost}(3, 7), 1 + \text{cost}(3, 8)\} = 7$$

$$\text{cost}(2, 3) = 9$$

$$\begin{aligned}\cos t(2,4) &= 18 \\ \cos t(2,5) &= 15 \\ \cos t(1,1) &= \min \{9 + \cos t(2,2), 7 + \cos t(2,3), 3 + \cos t(2,4), 1 + \cos t(2,5)\} = 16\end{aligned}$$

بنابراین مسیر با هزینهٔ مینیمم از  $s$  به  $t$  هزینهٔ مساوی 16 دارد. این مسیر را می‌توان با ذخیره‌سازی رئوس انتخاب شده در هر مرحله مشخص کرد. فرض کنید  $D(i,j)$  مساوی با راس  $i$  باشد که  $c(j,1) + \cos t(i+1,1)$  را مینیمم می‌سازد. برای گراف قبل داریم:

$$\begin{aligned}D(3,6) &= 10, D(3,7) = 10, D(3,8) = 10 \\ D(2,2) &= 4, D(2,3) = 6, D(2,4) = 8, D(2,5) = 8\end{aligned}$$

فرض کنید که مسیر با هزینهٔ مینیمم برابر  $s = 1, v_2, \dots, v_{k-1}, t$  است. مشاهده می‌شود که:

$$v_4 = D(3, D(2, D(1, 1))) = D(3, 7) = 10 \quad \text{و} \quad v_3 = D(2, D(1, 1)) = 7, v_2 = D(1, 1) = 2$$

تمرین: تحقیق کنید که مسیر 12, 10, 6, 3, 1 نیز هزینه‌ای برابر 16 دارد. آیا مسیر دیگری نیز با این هزینه وجود دارد؟

برای سهولت در ارایه یک برنامه برای فرمول (5) برای حل یک گراف  $k$  مرحله‌ای، فرض کنید که  $n$  راس  $V$  از 1 الی  $n$  شماره‌گذاری شده‌اند. این شماره‌گذاری به این ترتیب است که به راس  $s$  شماره 1 منسوب شده است. سپس رئوس  $v_2$ ، بعد از آن رئوس  $v_3$  شماره‌گذاری شده و سرانجام راس  $t$  با شماره  $n$  مشخص شده است. بدین ترتیب اندیس‌های رئوس  $v_{i+1}$  بزرگ‌تر از اندیس‌های منسوب به رئوس  $v_i$  هستند. این اندیس‌گذاری امکان به دست آوردن Cost و  $D$  را به ترتیب  $n-1, n-2, \dots, 1$  به دست می‌دهد. زیرا نویس‌های قبلی  $P$ , Cost,  $D$  فقط معرف شماره مراحل بودند که در برنامه بعدی حذف شده‌اند. الگوریتم حاصل FGraph است.

Algorithm FGraph ( $E, k, n, P$ )

```

Cost[n]=0
1 For j=n-1 Down To 1 do
2   r | (j,r) ∈ E & { c[j,r] + Cost[r] } مینیمم است
3   Cost[j] = c[j,r] + Cost[r]
4   D[j]=r
5
6 P[1]=1 ; P[k]=n
7 For j=2 to k-1 do
     P[j] = D[P[j-1]]
```

تحلیل پیچیدگی محاسباتی FGraph ساده است. اگر گراف  $G$  به وسیلهٔ لیست همسایگی تعریف شده باشد، آن‌گاه راس  $r$  در دستور 2 را می‌توان در زمان  $\Theta(\deg(j))$  پیدا کرد. اگر  $G$ ،  $|E|$  یا لداشته باشد، آن‌گاه زمان اجرای حلقة 1 از مرتبه  $(|V| + |E|)^2$  است. زمان اجرای حلقة دوم  $\Theta(k)$  است. بنابراین پیچیدگی محاسباتی الگوریتم  $(|V| + |E|)^2$  است.

مسئله گراف چند مرحله‌ای را می‌توان با روش پسرو نیز حل کرد. فرض کنید  $b_{cost}(i, j)$  مسیر با هزینهٔ مینیمم از  $s$  به راس  $j$  در  $v_i$  باشد. نیز فرض کنید که  $b_{cost}(i, j)$  هزینهٔ مسیر  $(j, i)$  است. با استفاده از روش پسرو داریم:

$$b_{cost}(i, j) = \min_{l \in V_{i-1}, (l, j) \in E} \{ b_{cost}(i-1, l) + c(l, j) \} \quad (2)$$

چون اگر  $(i, j) \in E$  آن‌گاه  $b_{cost}(j, i) = \infty$  و اگر  $b_{cost}(2, j) = \infty$  آن‌گاه  $b_{cost}(1, j) = \infty$  لذا  $b_{cost}(2, j) = c(1, j)$  است. رابطه (2) نخست با محاسبه  $b_{cost}(3, 6)$  برابر 9، سپس برای  $i=3, 4, \dots, 7$  حل کرد. برای گراف قبل داریم:

$$b_{cost}(3, 6) = \min \{ b_{cost}(2, 2) + 4, b_{cost}(2, 3) + 2 \} = 9$$

$$b_{cost}(3, 7) = \min \{ b_{cost}(2, 2) + 2, b_{cost}(2, 3) + 7, b_{cost}(2, 5) + 11 \} = 11$$

$$Bcost(3,8)=10$$

$$Bcost(4,9)=\min\{Bcost(3,6)+6, Bcost(3,7)+4\}=15$$

$$Bcost(4,10)=\min\{Bcost(3,6)+5, Bcost(3,7)+3, Bcost(3,8)+6\}=14$$

$$Bcost(4,11)=16$$

$$Bcost(5,12)=\min\{Bcost(4,9)+4, Bcost(4,10)+2, Bcost(4,11)+5\}=16$$

الگوریتم مربوط به روش پسرو در زیر ارایه شده است:

Algorithm BGraph (E,k,n,P)

$$Bcost[1]=0$$

For  $j=2$  to  $n$  do

{

$r | (r,j) \in E \text{ and } \{BCost[r] + c[r,j]\}$  مینیمم است

$$BCost[j] = Bcost[r] + c[r,j]$$

$$D[j] = r$$

}

$$P[1]=1; P[k]=n$$

For  $j=k-1$  downto 2 do

$$P[j]=D[P[j+1]]$$

### مساله تمامی کوتاه‌ترین مسیرها (الگوریتم فلوید - وارشاو)

فرض کنید  $G=(V, E, C)$  یک گراف جهت‌دار و وزن‌دار با  $n$  راس است که در آن  $V=\{1, 2, \dots, n\}$  مجموعه رئوس گراف و  $C$  ماتریس هزینه‌های آن است که به این صورت تعریف شده است:  $C[i,i]=0$  و برای  $j \neq i$  اگر  $(i,j) \in E(G)$  آن‌گاه  $C[i,j]$  حاوی هزینه یا  $(j,i)$  و در غیر این صورت حاوی مقدار  $\infty$  است. مساله تمامی کوتاه‌ترین مسیرها، تعیین یک ماتریس  $n \times n$  مثل  $A$  است، به گونه‌ای که  $[j,i] A$  حاوی طول کوتاه‌ترین مسیر از راس  $i$  به راس  $j$  است. ماتریس  $A$  را می‌توان با حل  $n$  مساله کوتاه‌ترین مسیرهای هم مبدا به دست آورد (الگوریتم کراسکال). چون به کارگیری هر بار این الگوریتم، پیچیدگی محاسباتی  $O(n^2)$  دارد، لذا ماتریس  $A$  را می‌توان در زمان  $O(n^3)$  به دست آورد با استفاده از اصل بهینه‌سازی نیز یک راه حل از مرتبه زمانی  $O(n^3)$  برای این مساله ارایه می‌دهیم. در این روش نیازی به این قید که هزینه یا  $i$  منفی نباشد، نیست. کافی است که  $G$  دوری به طول منفی نداشته باشد. مسلم است که اگر  $G$  دارای دوری به طول منفی باشد، آن‌گاه کوتاه‌ترین مسیر بین هر دو راس این دور برابر  $\infty$  خواهد بود. حال اجازه دهید که کوتاه‌ترین مسیر از راس  $i$  به راس  $j$  در  $G$  بررسی کنیم. این مسیر از راس  $i$  شروع و به راس  $j$  ختم می‌شود. می‌توان فرض کرد که این مسیر دور ندارد، زیرا در غیر این صورت می‌توان آن را بدون افزایش طول مسیر حذف کرد (هیچ دوری به طول منفی وجود ندارد). اگر  $k$  یک راس میانی در این کوتاه‌ترین مسیر باشد، آن‌گاه دو زیر مسیر از  $i$  به  $k$  و از  $k$  به  $j$  باید کوتاه‌ترین مسیرهای به ترتیب از  $i$  به  $k$  و از  $k$  به  $j$  باشند، در غیر این صورت مسیر از  $i$  به  $j$  کوتاه‌ترین مسیر نخواهد بود. بنابراین، اصل بهینه‌سازی در اینجا رعایت شده و می‌توان از برنامه‌نویسی پویا برای حل مساله استفاده کرد. اگر  $k$  یک راس میانی با بالاترین اندیس باشد، آن‌گاه مسیر از  $i$  به  $k$  کوتاه‌ترین مسیری در  $G$  است که از  $i$  شروع و به  $k$  ختم شده و از هیچ راس میانی با اندیس بزرگ‌تر از  $(k-1)$  عبور نمی‌کند. بنابراین، می‌توان ایجاد کوتاه‌ترین مسیر از  $i$  به  $j$  را به این صورت ملاحظه کرد که نخست تصمیم بگیریم که کدام راس، راس میانی با بالاترین اندیس است. وقتی چنین تصمیمی اتخاذ شد نیاز به مشخص کردن دو تا کوتاه‌ترین داریم، اول مسیر از  $i$  به  $k$  و دیگری مسیر از  $k$  به  $j$ . هیچ یک از این مسیر از راس میانی با اندیس بزرگ‌تر از  $(k-1)$  عبور نمی‌کند. با به کار

بردن  $A^k[i,j]$  برای نمایش طول کوتاهترین مسیر از  $i$  به  $j$ ، که از هیچ راس میانی با اندیس بزرگ‌تر از  $k$  عبور نکند، داریم:

$$A[i,j] = \min \left\{ A[i,j], \min \left\{ A^{k-1}[i,k] + A^{k-1}[k,j], 1 \leq k \leq n \right\} \right\} \quad (1)$$

بدیهی است که  $A^0[i,j] = C[i,j]$ ، برای  $1 \leq i \leq n$ ،  $1 \leq j \leq n$ . با استدلال مشابهی می‌توان یک رابطه بازگشتی برای  $A^k[i,j]$  به دست آورد. کوتاهترین مسیر از راس  $i$  به  $j$ ، که از راس با اندیس بزرگ‌تر از  $k$  عبور نکند، یا از خود راس  $k$  عبور می‌کند یا از آن عبور نمی‌کند. اگر این مسیر از راس  $k$  عبور کند در این صورت داریم:

$$A^k[i,j] = A^{k-1}[i,k] + A^{k-1}[k,j] \quad (2)$$

$$A^k[i,j] = A^{k-1}[i,j] \quad (3)$$

بنابراین

$$A^k[i,j] = \min \left\{ A^{k-1}[i,j], A^{k-1}[i,k] + A^{k-1}[k,j] \right\} \quad (4)$$

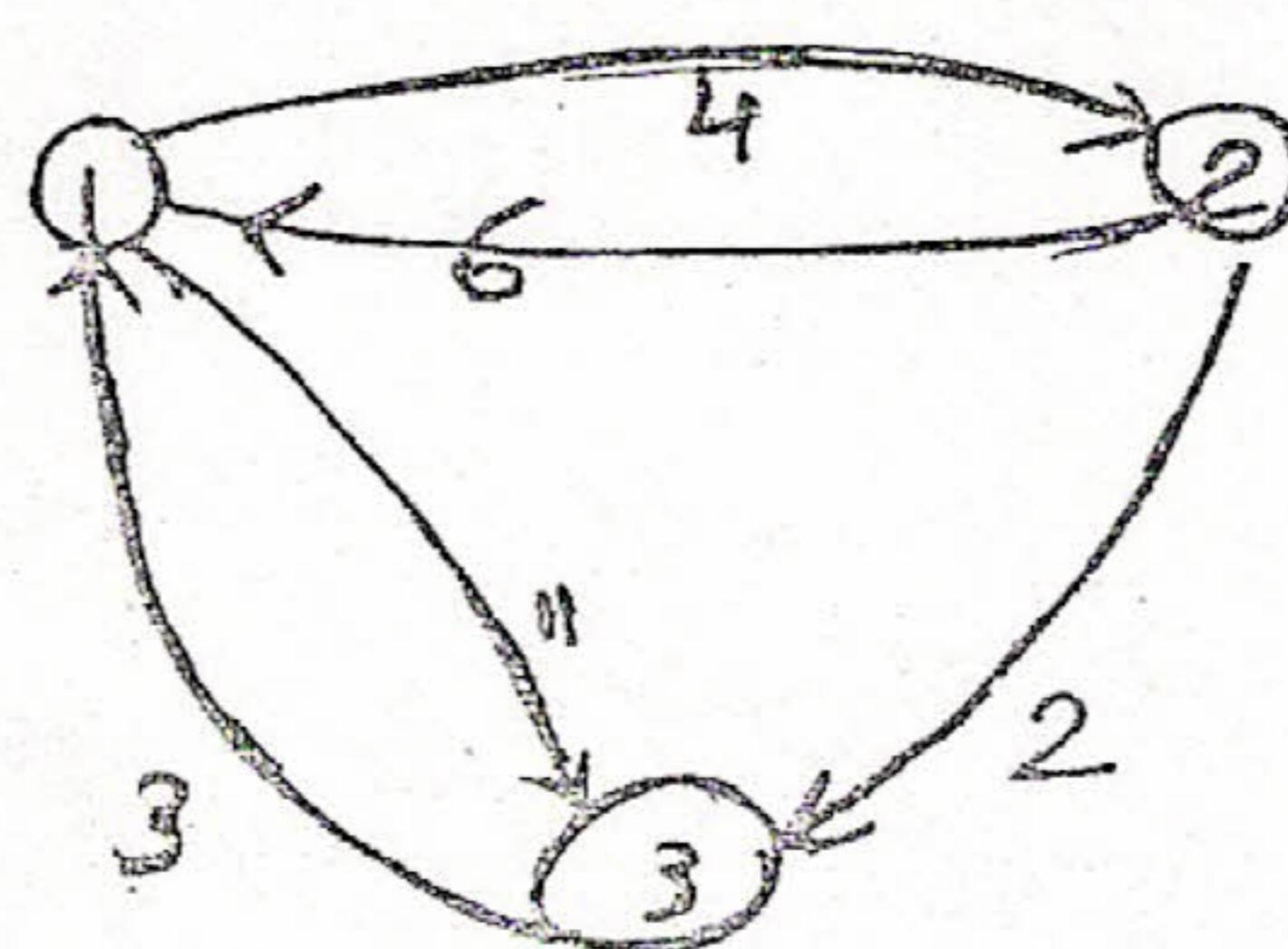
رابطه بازگشتی (4) را می‌توان نخست با محاسبه  $A^1$ ، سپس  $A^2$ ، ..... و در آخر برای  $A^n$  حل کرد. چون در  $G$  راسی با اندیس  $A[i,j] = A^n[i,j]$  بزرگ‌تر از  $n$  وجود ندارد، لذا

```
Algorithm All_Paths (Cost , A , n)
for i=1 to n do
{
    for j=1 to n do
        A [ i,j ] = Cost ( i,j )
}
for k=1 to n do
    for i=1 to n do
        for j=1 to n do
            A[i,j] = min { A[i,j] , A[i,k]+A[k,j] }
```

الگوریتم زیر  $A^n[i,j]$  را محاسبه می‌کند:

فرض کنید که  $M = \max \{ \text{Cost}[i,j] \}$ ، به آسانی می‌توان تحقیق کرد که  $A^{(n)}[i,j] \leq (n-1)M$ . با توجه به کارکرد All\_Paths به آسانی فهمیده می‌شود که اگر  $(i,j) \notin E(G)$  و  $i \neq j$ ، آن‌گاه  $\text{Cost}[i,j]$  را می‌توان به مقدار بزرگ‌تری از  $(n-1)M$  (به جای  $\infty$ ) مقدار اولیه داد. در خاتمه الگوریتم، اگر مقدار  $A^{(n)}[i,j]$  از  $(n-1)M$  بیشتر شده، مفهومش این است که مسیری از  $i$  به  $j$  وجود ندارد.

مثال ۹: مسئله تمامی کوتاهترین مسیرها را برای گراف شکل ۲ حل کنید.



شکل ۱

ماتریس هزینه‌های این گراف در شکل (۲ الف) آمده است. مقدار اولیه  $A^{(0)}$  و نیز مقادیر سه مرحله بعد آن  $A^{(1)}$ ،  $A^{(2)}$  و  $A^{(3)}$ ، در شکل ۲ آمده‌اند.

$A^{(0)}$	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

$A^{(3)}$	1	2	3
1	0	4	6
2	5	0	2
3	3	7	0

(د)

$A^{(2)}$	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

(ج)

$A^{(1)}$	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

(ب)

شكل ٢

## درخت های جستجوی دودویی بهینه

قبل از پرداختن به بحث اصلی، یادآوری چند مطلب از درس ساختمان داده‌ها ضروری است.

تعریف درخت دودویی جستجو: یک درخت دودویی جستجو، درختی است دودویی که اگر تهی نیست شرایط زیر را دارد:

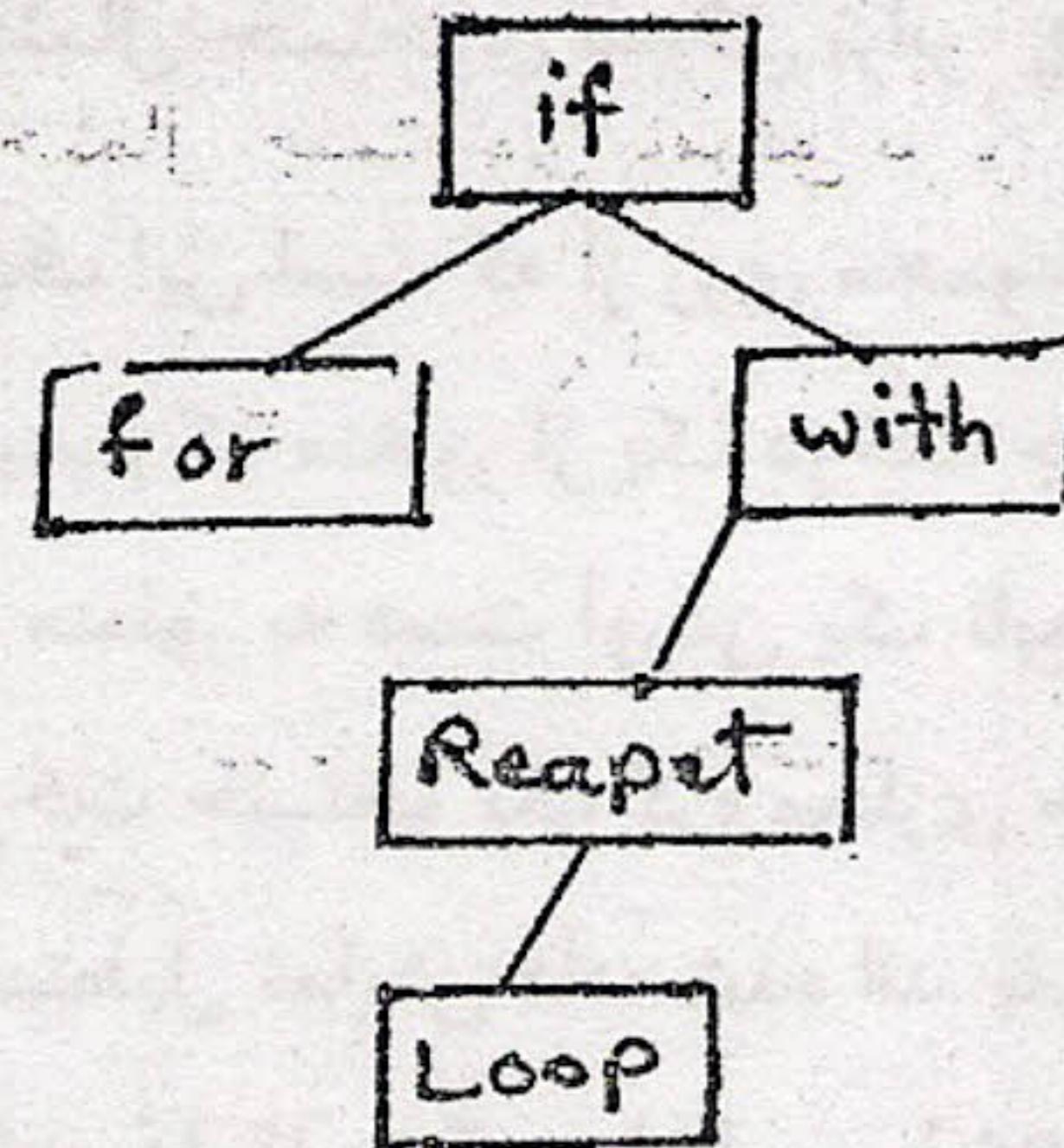
(الف) به ریشه، کلیدی منسوب است.

(ب) کلیدهای منسوب به زیر درخت چپ (در صورت وجود) کوچک‌تر از کلید ریشه هستند.

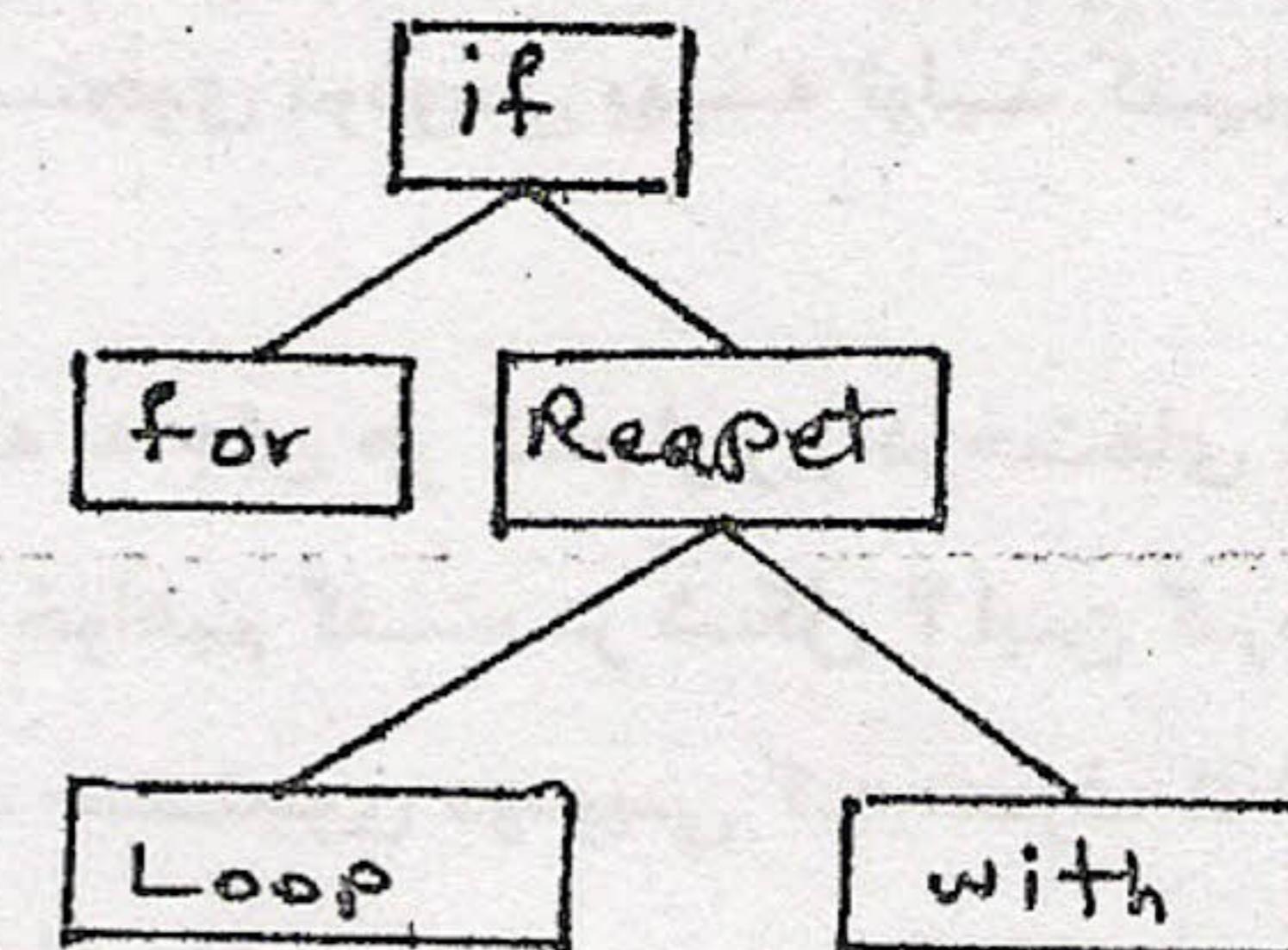
(ج) کلیدهای منسوب به زیر درخت راست (در صورت وجود) بزرگ‌تر از کلید ریشه هستند.

(د) زیر درخت‌های چپ و راست هر کدام یک درخت دودویی جستجو هستند.

شکل ۳ دو درخت دودویی جستجو را نشان می‌دهد



(ب)



(الف)

شکل ۳

معمولًا برای جستجوی این که آیا کلید  $x$  در یک درخت دودویی جستجو هست و یا خیر، از الگوریتم جستجوی دودویی، که در زیر آمده است، استفاده می‌شود. اگر  $x$  در درخت باشد الگوریتم فوق گرهی را بر می‌گرداند که کلید  $x$  را شامل است. در این حالت مشخص می‌شود که جستجو موفق بوده است، در غیر این صورت جستجو ناموفق بوده و توسط الگوریتم مذبور مقدار Null برگردانیده می‌شود.

## الگوریتم جستجوی دودویی

Algorithm Binary\_Search (tree , x , t)

T=tree

While t ≠ Null do

Case

- |                |                     |
|----------------|---------------------|
| : x < ident(t) | : t = Leftchild(t)  |
| : x = ident(t) | : Return            |
| : x > ident(t) | : t = Rightchild(t) |

End case

End Binary\_Search

برای یک مجموعه از شناسه‌های داده شده، می‌خواهیم یک درخت دودویی جستجو تشکیل بدهیم. برای این شناسه‌ها درخت‌های دودویی جستجو با ویژگی‌های متفاوت وجود دارد. برای مثال اگر شناسه‌ها متساوی الاحتمال باشند، در این صورت برای درخت (الف)

در بدترین حالت برای جستجوی موفق، ۴ مقایسه و در حالت میانگین،  $\frac{12}{5}$  مقایسه لازم است. برای درخت (ب) تعداد مقایسه‌ها در

بدترین حالت برای جستجوی موفق ۳ و تعداد این مقایسه‌ها برای حالت میانگین برابر  $\frac{11}{5}$  است. برای مثال، الگوریتم

برای یافتن شناسه‌های Binary\_Search برای درخت (الف) به ترتیب ۱, ۲, ۲, ۳, ۴ مقایسه به کار

$$\text{خواهد برد و بنابراین برای این درخت، میانگین تعداد مقایسه‌ها برای جستجوی موفق برابر } \frac{1+2+2+3+4}{5} = \frac{12}{5} \text{ است.}$$

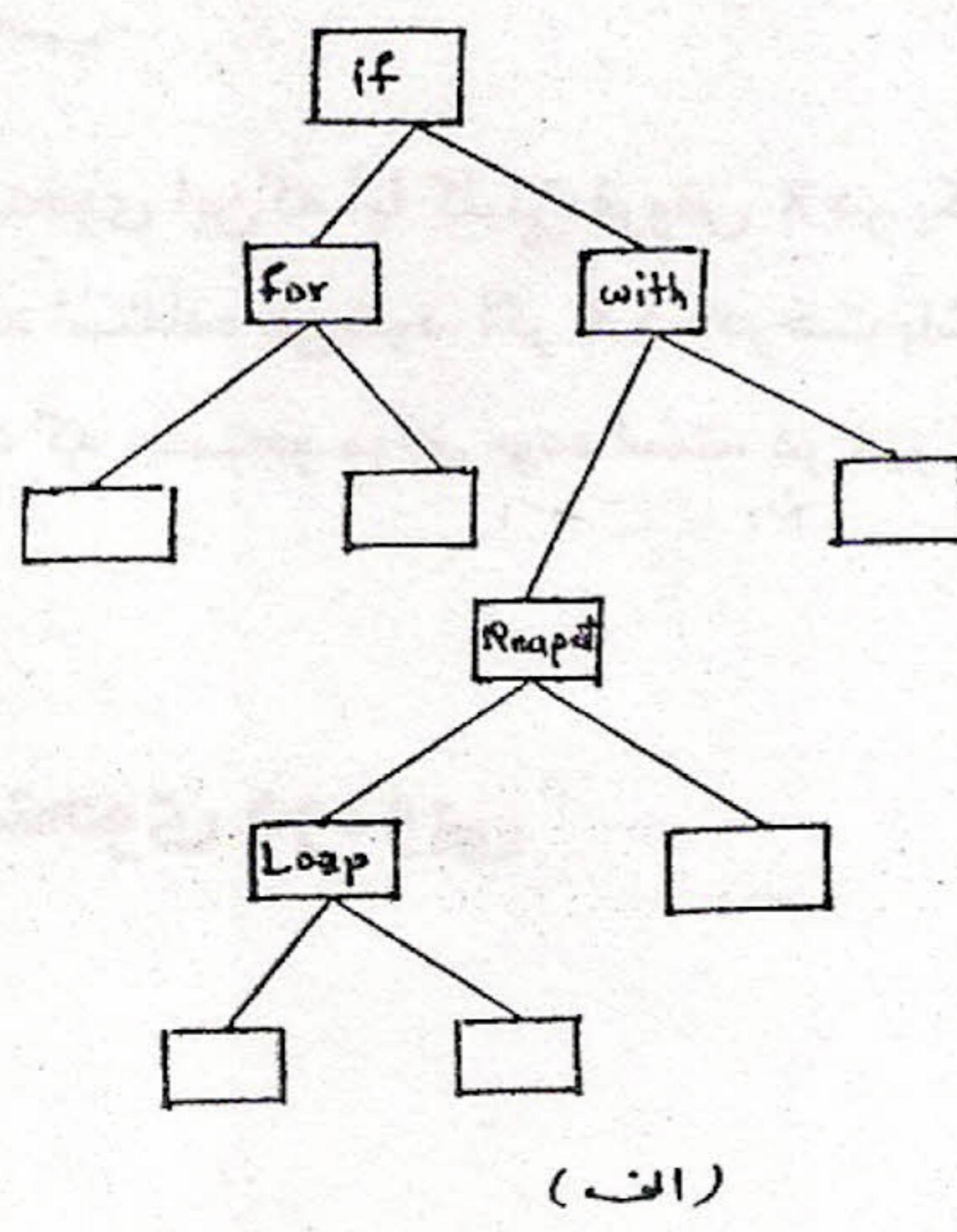
در حالت عمومی ما با شناسه‌های با احتمالات متفاوت سروکار داریم و گذشته از این باید حالت جستجوی ناموفق را هم در نظر بگیریم.

فرض  $\{a_1, a_2, \dots, a_n\}$  مجموعه‌ای از  $n$  شناسه باشد. نیز فرض کنید که  $i=1, 2, \dots, n$ ,  $p_i$  مساوی باشد با احتمال این که کلیدخواسته شده  $x$  برابر  $a_i$  باشد. همچنین فرض کنید که  $i=0, 1, \dots, n$ ,  $q_i$  مساوی باشد با احتمال این که کلید خواسته شده  $x$  در رابطه  $a_i < x < a_{i+1}$  صادق باشد (فرض شده است که  $a_0 = -\infty$ ,  $a_{n+1} = +\infty$ ). بنابراین، احتمال جستجوی موفق برابر  $p_i$  و

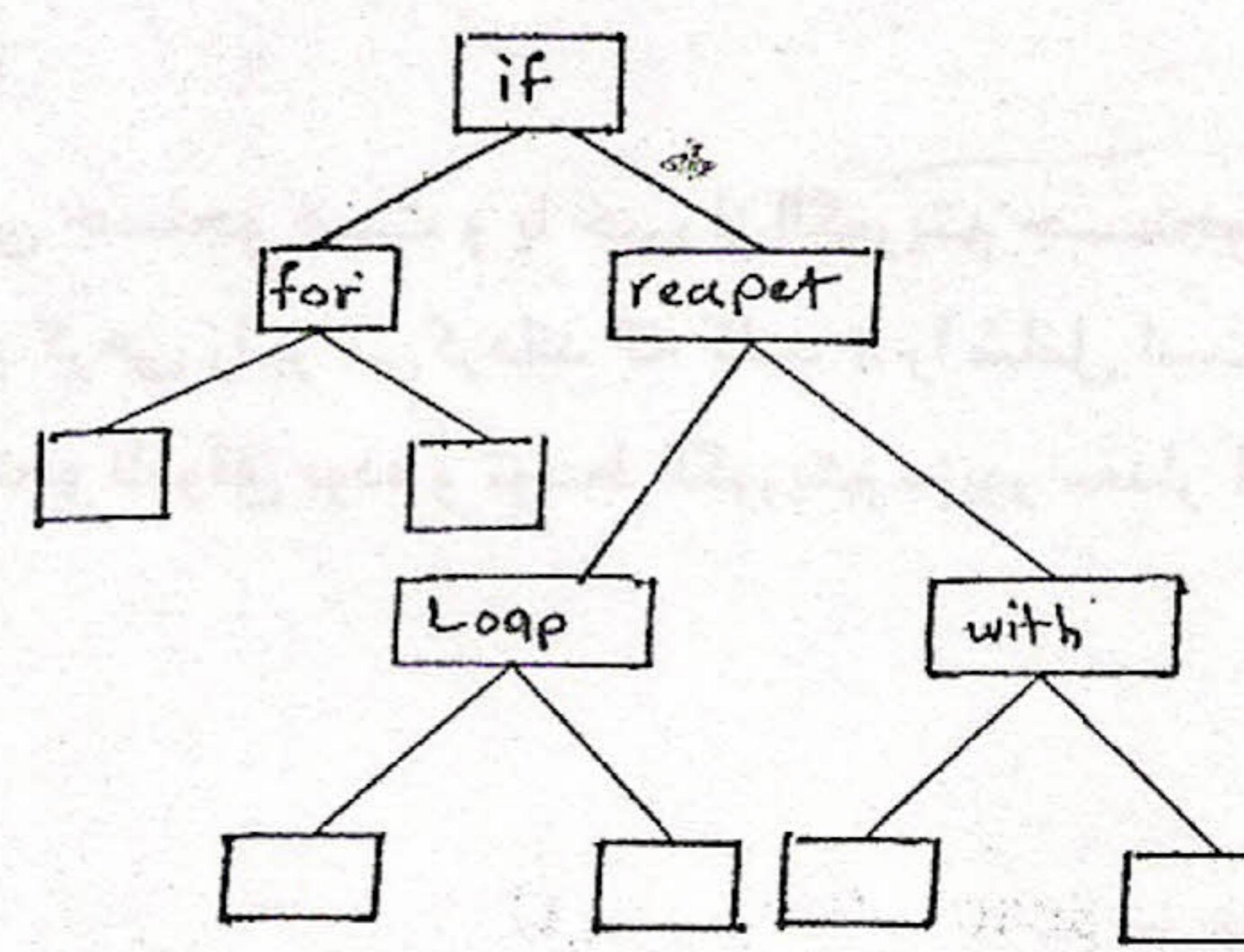
$$\sum_{i=1}^n p_i + \sum_{i=0}^n q_i = 1 \text{ است. بدیهی است که احتمال جستجوی ناموفق برابر } \sum_{i=0}^n q_i \text{ است.}$$

هدف این است که از روی مجموعه شناسه‌های داده شده  $\{a_1, a_2, \dots, a_n\}$  یک درخت جستجوی دودویی بهینه تولید کنیم. حال ببینیم که منظور از یک درخت جستجوی دودویی بهینه چیست؟

به منظور به دست آوردن یکتابع هزینه برای درخت‌های جستجوی دودویی، بهتر است که به جای هر کدام از زیردرخت‌های تهی در درخت جستجو یک گره مجازی جایگزین کنیم. چنین گره‌هایی خارجی (برگ) خواهیم گفت، در شکل ۴ این گره‌ها با مستطیل نمایش داده شده‌اند. بقیه گره‌ها را گره‌های داخلی خواهیم نامید. در یک درخت جستجوی دودویی که معرف  $n$  شناسه باشد، دقیقاً  $n+1$  گره خارجی وجود دارد. جستجوی موفق به یکی از گره‌های داخلی و جستجوی ناموفق به یکی از گره‌های خارجی ختم می‌شود.



(الف)



(ب)

شکل ۴

اگر جستجوی موفق به یک گره داخلی واقع در سطح  $i$  برسد، آن‌گاه در الگوریتم قبل، ۱ مرحله تکرار خواهیم داشت، بنابراین سهم هزینه گره داخلی  $i$  برابر  $P(i)^* \text{Level}(a_i)$  خواهد بود.

جستجوی ناموفق به یکی از گره‌های خارجی می‌رسد. شناسه‌های ناموجود در درخت tree در الگوریتم Binary Search را می‌توان

به  $(n+1)$  کلاس هم ارزی  $E_i$ ,  $0 \leq i \leq n$  به صورت زیر افزای کرد:

$$E_0 = \{x \mid x < a_0\}, E_n = \{x \mid x > a_n\}, E_i = \{x \mid a_i < x < a_{i+1}\}, i = 1, 2, \dots, n-1$$

برای شناسه‌های متعلق به کلاس  $E_i$ , جستجوی ناموفق به یک گره خارجی می‌رسد، و برای شناسه‌های متعلق به دو کلاس متفاوت،

جستجوی ناموفق به دو گره خارجی متفاوت ختم می‌شود. اگر گره خارجی  $E_i$  در سطح level( $E_i$ ) قرار داشته باشد، آن‌گاه حلقة

در الگوریتم قبلی (l-1) بار تکرار خواهد شد، بنابراین سهم هزینه این گره برابر  $q(i)^*(level(E_i) - 1)$  خواهد بود.

با توجه به مطالب گفته شده میانگین هزینه برای یک درخت جستجوی دودویی مثل  $T$  برابر است با:

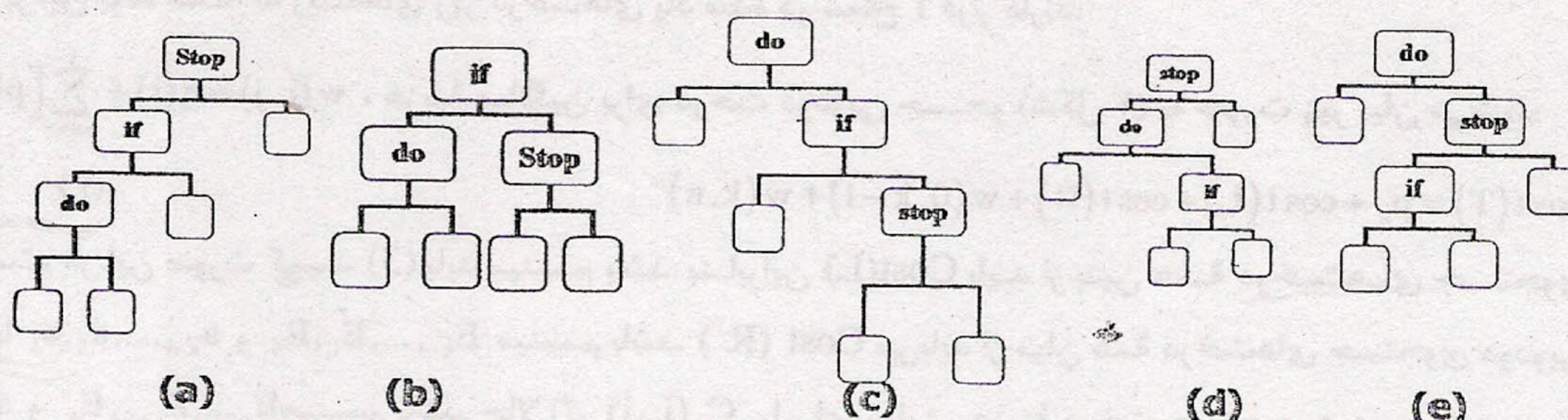
$$\text{cost}(T) = \sum_{i=1}^n P(i) * \text{level}(a_i) + \sum_{i=0}^n q(i) * (level(E_i) - 1) \quad (1)$$

## تعريف 2: (درخت جستجوی دودویی بهینه)

یک درخت جستجوی دودویی بهینه برای شناسه‌های  $a_1, a_2, \dots, a_n$  درختی است که برای آن کمیت (1) مینیمم باشد.

**مثال 1:** فرض کنید که  $(a_1, a_2, a_3) = (\text{do}, \text{if}, \text{stop})$

درخت‌های جستجوی دودویی زیر را می‌توان برای این سه شناسه تشکیل داد:



با داده‌های متساوی احتمال  $P(i) = q(i) = \frac{1}{7}$ , هزینه این درخت‌ها به صورت زیر خواهد بود:

$$\text{cost (a)} = \frac{15}{7} \quad (\text{درخت a})$$

$$\text{cost (b)} = \frac{13}{7} \quad (\text{درخت b})$$

$$\text{cost (c)} = \frac{15}{7} \quad (\text{درخت c})$$

$$\text{cost (d)} = \frac{15}{7} \quad (\text{درخت d})$$

$$\text{cost (e)} = \frac{15}{7} \quad (\text{درخت e})$$

در این حالت، همان‌گونه که می‌شد انتظار داشت درخت b بهینه است.

اگر  $P(1) = 0.5, P(2) = 0.1, P(3) = 0.05, q(0) = 0.15, q(1) = 0.1, q(2) = 0.05, q(3) = 0.05$  در این صورت خواهیم داشت:

$$\text{cost (a)} = 2.65$$

$$\text{cost (b)} = 1.9$$

$$\text{cost (c)} = 1.5$$

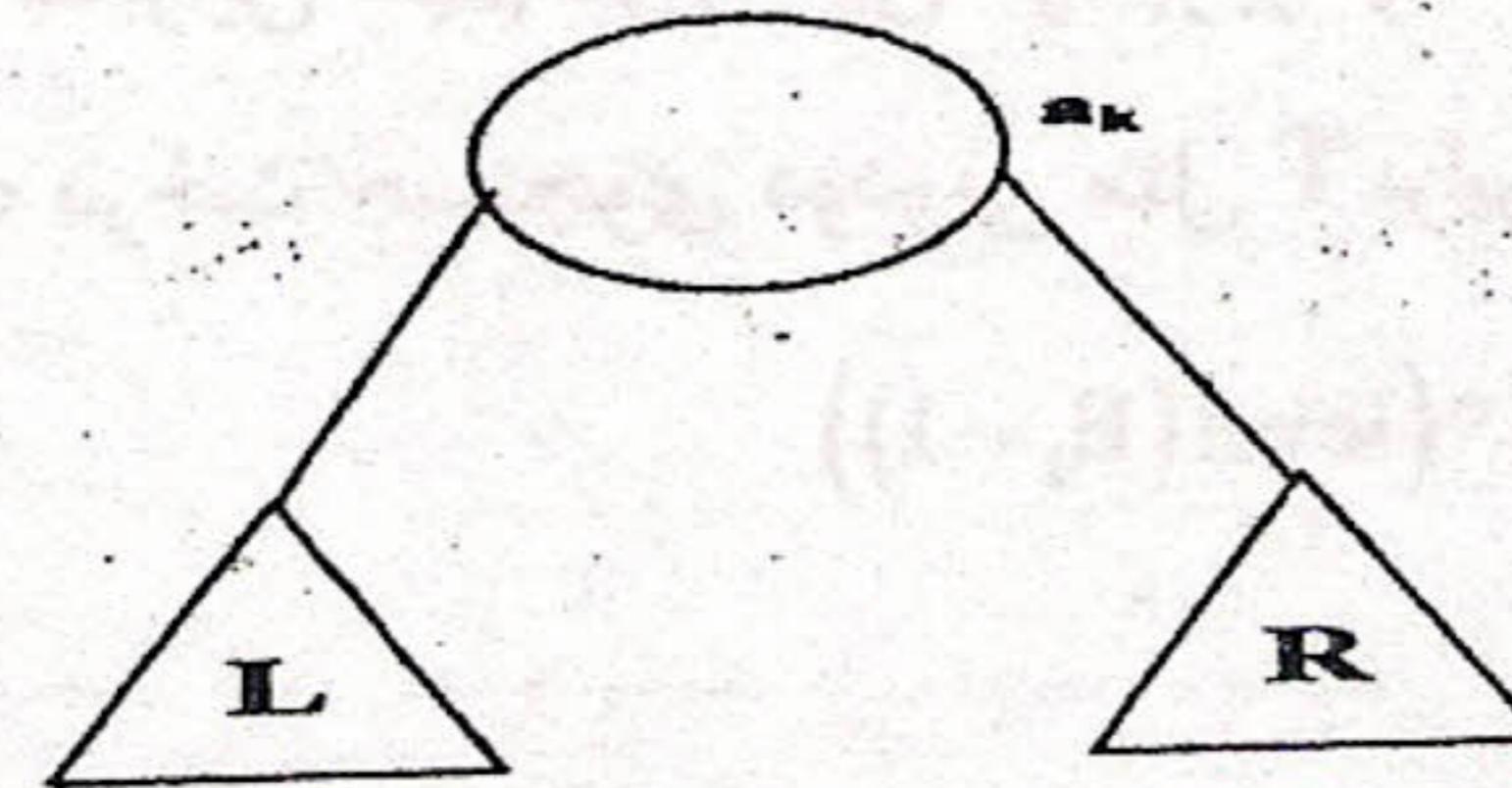
$$\text{cost (d)} = 2.05$$

$$\text{cost (e)} = 1.6$$

برای نمونه برای حالت اخیر، هزینه درخت a به صورت زیر حاصل شده است. سهم جستجوی موفق برابر  $3*0.5 + 2*0.1 + 0.05 = 1.75$  است. این سهم برای جستجوی ناموفق برابر  $3*0.15 + 3*0.1 + 2*0.05 + 0.05 = 1.05$  است. هزینه درخت‌های دیگر را نیز می‌توان به طریقه مشابه حساب کرد. برای این  $p$  ها و  $q$  ها مشاهده می‌شود که درخت c بهینه است.

## ارایه یک راه حل برنامه‌نویسی پویا برای مساله

برای به دست آوردن یک درخت جستجوی دودویی بهینه با استفاده از برنامه‌نویسی پویا، لازم است که تشکیل چنین درختی را به صورت نتیجه‌های از یک رشته تصمیمات در نظر گرفته و اصل بهینه‌سازی را هم فراموش نکنیم. یک روش برای انجام این کار این است که تصمیم بگیریم کدام یک از شناسه‌های  $a_i$  را به عنوان ریشه درخت  $T$  قرار دهیم، اگر  $a_k$  را به عنوان ریشه انتخاب کنیم، واضح است که گره‌های داخلی  $a_{k-1}, a_2, a_1, \dots, E_{k+1}, E_k, \dots, E_{k-1}$  و نیز گره‌های خارجی  $L$  و  $R$  باید در زیر درخت سمت چپ ( $L$ ) و گره‌های داخلی  $a_n, \dots, a_{k+2}, a_{k+1}$  هم‌چنین گره‌های خارجی  $E_n, \dots, E_{k+1}, E_k$  باید در زیر درخت سمت راست ( $R$ ) قرار بگیرند (شکل زیر).



حال تعریف می‌کنیم:

$$\text{cost}(L) = \sum_{i=1}^{k-1} P(i) * \text{level}(a_i) + \sum_{i=0}^{k-1} q(i) [\text{level}(E_i) - 1] \quad (2)$$

$$\text{cost}(R) = \sum_{i=k+1}^n P(i) * \text{level}(a_i) + \sum_{i=k}^n q(i) [\text{level}(E_i) - 1]$$

در هر دو حالت فرض بر این بوده است که ریشه‌های زیر درخت‌های یاد شده در سطح 1 قرار دارند.

با تعریف  $w(i, j) = q(i) + \sum_{l=i+1}^j [p(l) + q(l)]$  هزینه میانگین برای درخت دودویی جستجو (شکل ۱) به صورت زیر بیان می‌شود:

$$\text{cost}(T) = p_k + \text{cost}(L) + \text{cost}(R) + w(0, k-1) + w(k, n) \quad (3)$$

اگر  $T$  درخت بهینه است، در این صورت کمیت (3) باید مینیمم باشد. بنابراین  $\text{Cost}(L)$  باید از بین همه درخت‌های جستجوی دودویی تشکیل شده با  $a_1, a_2, a_3, \dots, E_1, E_0$  و  $a_{k-1}, \dots, E_{k-1}, E_k$  مینیمم باشد.  $\text{Cost}(R)$  می‌باید از میان همه درخت‌های جستجوی دودویی شامل  $a_n, \dots, a_{k+2}, a_{k+1}$  و  $E_n, \dots, E_{k+1}, E_k, \dots, E_{k-1}, E_i$  مینیمم باشد. حالا اگر  $(i, j)$  را برای نمایش هزینه درخت جستجوی دودویی بهینه  $T_{ij}$  بر روی شناسه‌های  $a_{i+1}, \dots, a_{j-1}, a_j, \dots, a_{i+1}$  و  $E_{i+1}, \dots, E_j, \dots, E_i$  به کار ببریم، باید  $\text{Cost}(L) = C(0, k-1)$ ,  $\text{Cost}(R) = C(k, n)$ . از طرف دیگر  $k$  نیز باید طوری انتخاب شود که

$$p(k) + c(0, k-1) + c(k, n) + w(0, k-1) + w(k, n)$$

مینیمم شود. بنابراین برای  $(0, n)$  نتیجه زیر به دست می‌آید:

$$c(0, n) = \min_{i < k \leq j} \{c(0, k-1) + c(k, n) + p_k + w(0, k-1) + w(k, n)\} \quad (3)$$

فرمول (3) را می‌توان به صورت زیر تعمیم داد:

$$c(i, j) = \min_{i < k \leq j} \{c(i, k-1) + c(k, j) + p_k + w(i, k-1) + w(k, j)\} \quad (4)$$

$$c(i, j) = \min_{i < k \leq j} \{C(i, k-1) + c(k, j)\} + w(i, j) \quad (5)$$

با توجه به شرایط مرزی  $c(i,i) = 0$ , برای رسیدن به جواب  $C(0,n)$  می‌توان به صورت زیر عمل کرد:  
 نخست  $c(i,j)$  را برای حالت  $j-i=1$  حساب می‌کنیم. سپس  $c(i,j)$  را برای حالت  $j-i=2$  به دست می‌آوریم. پس از آن  $c(i,j)$  را برای حالت  $j-i=3$  حساب می‌کنیم و الی آخر.  
 اگر در حین محاسبه  $c(i,j)$  ها، ریشه  $T_j$  هم ذخیره کنیم، آن‌گاه درخت جستجوی دودویی بهینه را می‌شود از روی

$r(i,j)$  ها تشکیل داد.  $r(i,j)$  مقدار  $k$  ای است که از فرمول (5) حاصل می‌شود.

مثال : درخت جستجوی دودویی بهینه را برای شناسه‌های  $(a_1, a_2, a_3, a_4) = (\text{do}, \text{if}, \text{read}, \text{while})$  با احتمالات داده شده در جدول زیر به دست آورید.

i	P <sub>i</sub>	q <sub>i</sub>
0	-	2
1	3	3
2	3	1
3	1	1
4	1	1

حل :

برای سهولت در انجام محاسبات  $p$  ها و  $q$  ها را به عدد 16 ضرب می‌کنیم. به عنوان مقادیر اولیه داریم:

$$0 \leq i \leq 4 \quad c(i,i) = 0, w(i,i) = q(i)$$

با استفاده از معادله (5) و این رابطه که  $w(i,j) = p(j) + q(j) + w(i,j-1)$  نتایج زیر حاصل خواهند شد:

برای  $j - I = 1$

$$w(0,1) = p_1 + q_1 + q_0 = 3 + 3 + 2 = 8$$

$$c(0,1) = \min\{c(0,0) + c(1,1)\} + w(0,1) = 8$$

$$r(0,1) = 1$$

$$w(1,2) = p_2 + q_2 + q_1 = 3 + 1 + 3 = 7$$

$$c(1,2) = \min\{c(1,1) + c(2,2)\} + w(1,2) = 7$$

$$r(1,2) = 2$$

$$w(2,3) = p_3 + q_3 + q_2 = 1 + 1 + 1 = 3$$

$$c(2,3) = \min\{c(2,2) + c(3,3)\} + w(2,3) = 3$$

$$r(2,3) = 3$$

$$w(3,4) = p_4 + q_4 + q_3 = 1 + 1 + 1 = 3$$

$$c(3,4) = \min\{c(3,3) + c(4,4)\} + w(3,4) = 3$$

$$r(3,4) = 4$$

برای  $j - i = 2$ 

$$w(0,2) = p_2 + q_2 + w(0,1) = 3 + 1 + 8 = 12$$

$$c(0,2) = \min \{ \underline{c(0,0) + c(1,2)}, c(0,1) + c(2,2) \} + w(0,2) = 19$$

$$r(0,2) = 1$$

$$w(1,3) = p_3 + q_3 + w(1,2) = 1 + 1 + 7 = 9$$

$$c(1,3) = \min \{ \underline{c(1,1) + c(2,3)}, c(1,2) + c(3,3) \} + w(1,3) = 12$$

$$r(1,3) = 2$$

$$w(2,4) = p_4 + q_4 + w(2,3) = 1 + 1 + 3 = 5$$

$$c(2,4) = \min \{ \underline{c(2,2) + c(3,4)}, c(2,3) + c(4,4) \} + w(2,4) = 8$$

$$r(2,4) = 3$$

برای  $j - i = 3$ 

$$w(0,3) = p_3 + q_3 + w(0,2) = 1 + 1 + 12 = 14$$

$$c(0,3) = \min \{ c(0,0) + c(1,3), \underline{c(0,1) + c(2,3)}, c(0,2) + c(3,3) \} + w(0,3) = 25$$

$$r(0,3) = 2$$

$$w(1,4) = p_4 + q_4 + w(1,3) = 1 + 1 + 9 = 11$$

$$c(1,4) = \min \{ \underline{c(1,1) + c(2,4)}, c(1,2) + c(3,4), c(1,3) + c(4,4) \} + w(1,4) = 19$$

$$R(1,4) = 2$$

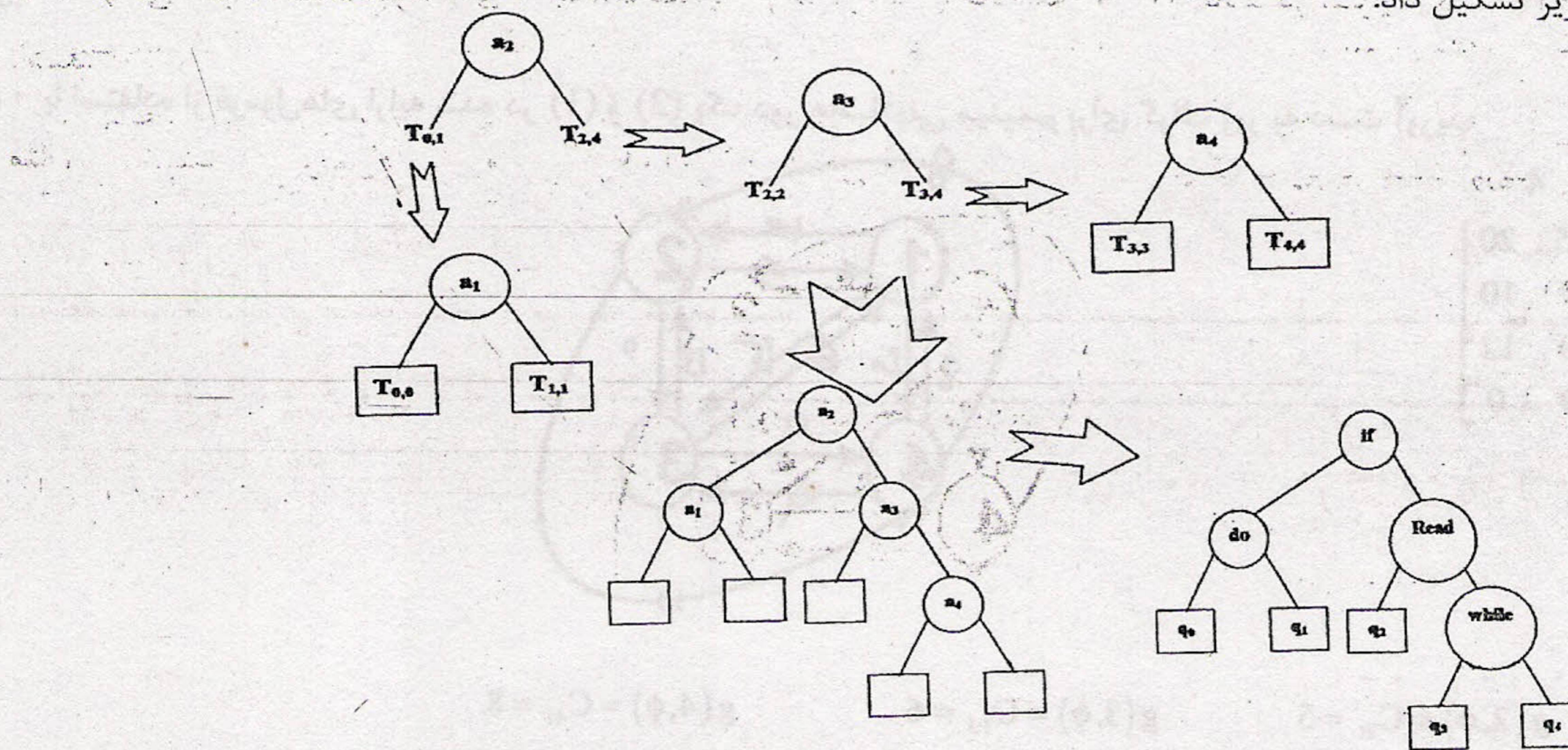
برای  $j - i = 4$ 

$$w(0,4) = p_4 + q_4 + w(0,3) = 1 + 1 + 14 = 16$$

$$c(0,4) = \min \{ c(0,0) + c(1,4), \underline{c(0,1) + c(2,4)}, c(0,2) + c(3,4), c(0,3) + c(4,4) \} + w(0,4) = 32$$

$$r(0,4) = 2$$

بنابراین  $a_2 = 32$  و  $T_{04}$  درخت دودویی جستجوی با هزینه مینیمم برای  $(a_1, a_2, a_3, a_4)$  است، ریشه درخت  $T_{04}$  شناسه  $a_2$  است. بنابراین زیر درخت چپ  $T_{01}$  و زیر درخت راست  $T_{24}$  است. درخت  $T_{01}$  دارای ریشه  $a_1$  و زیر درخت‌های  $T_{00}$  و  $T_{11}$  است. درخت  $T_{24}$  دارای ریشه  $a_3$  و زیر درخت‌های چپ و راست به ترتیب  $T_{34}, T_{22}$  است. بنابراین، با توجه به نتایج حاصل، درخت  $T_{0k}$  را می‌توان به صورت زیر تشکیل داد.



تمرین : آیا درخت جستجوی دودویی بهینه دیگری برای داده‌های بالا وجود دارد؟ آن را به دست آورید.

### مسئله فروشنده دوره گرد (TSP)

فرض کنید که  $G=(V,E,c)$  یک گراف وزن دار و جهت داری است که در آن  $n > 1, V = \{1, 2, 3, \dots, n\}$  مجموعه رئوس و  $c$  ماتریس وزن‌ها است.  $c_{ij}$  هزینه یال  $(j, i)$  است که به صورت زیر تعریف شده است. اگر  $E \in (i, j) \in \{0, c\}$  آن‌گاه  $c_{ij} < c$  و در غیر این صورت  $c_{ij} = +\infty$ .

یک دور هامیلتونی در  $G$ ، مدار ساده جهت داری است که همه رئوس  $V$  را شامل باشد. هزینه یک دور هامیلتونی برابر مجموع هزینه یال‌های تشکیل دهنده آن است. مسئله TSP، یافتن یک دور هامیلتونی با هزینه مینیمم است.

بدون این‌که به عمومیت مسئله لطمه‌ای وارد شود، فرض می‌کنیم که دور مورد نظر از راس 1 شروع و به همین راس نیز ختم می‌شود: هر دور هامیلتونی، تشکیل شده است از یک یال مثل  $\{l, k\} \in V - \{1\}$ ، و مسیری که باید از راس  $k$  شروع و به راس 1 ختم شده و از هر راس  $\{1, k\} - V$  نیز فقط یک بار عبور کند. بدیهی است که برای یک دور هامیلتونی مینیمم، مسیر از  $k$  به 1 کوتاه‌ترین مسیر از بین همه مسیرهایی است که باید از راس  $k$  شروع و به راس 1 ختم شده و از کلیه رئوس  $\{1, k\} - V$  نیز عبور بکند. بنابراین اصل بهینه‌سازی برای این مسئله قابل اعمال است.

فرض کنید که  $(i, s, g)$  طول کوتاه‌ترین مسیری باشد که از راس  $i$  شروع شده، از کلیه رئوس  $S$  عبور کرده و به راس  $i$  یک ختم می‌شود. بدیهی است که با توجه به این تعریف، طول یک دور هامیلتونی با هزینه مینیمم، برابر  $(1, V - \{1\}, g)$  است.

با توجه به اصل بهینه‌سازی داریم:

$$g(1, V - \{1\}) = \min_{2 \leq k \leq n} \{C_{ik} + g(k, V - \{1, k\})\} \quad (1)$$

و با توجه به تعمیم اصل بهینه‌سازی داریم:

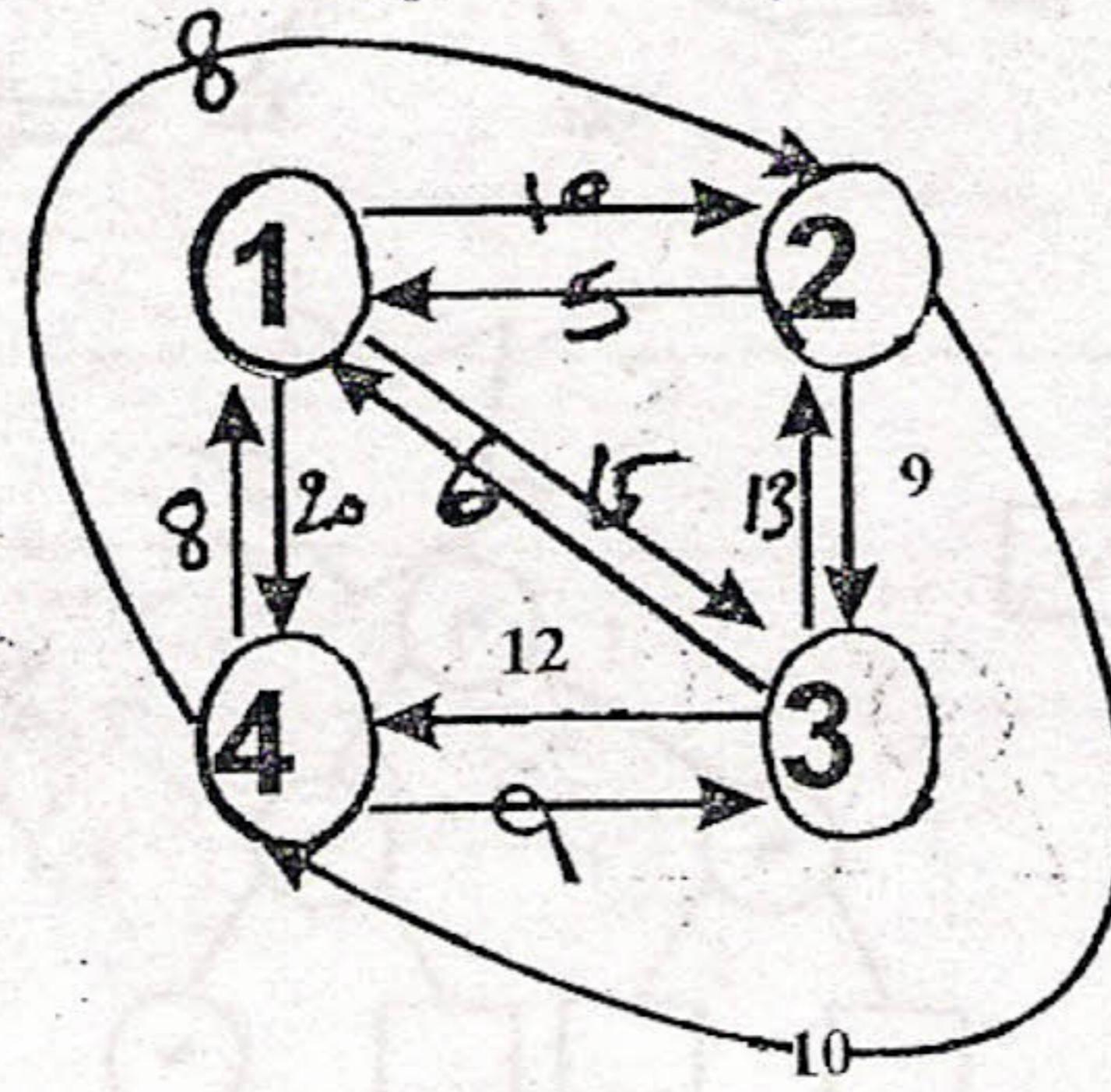
$$g(i, S) = \min_{j \in S, i \in S} \{C_{ij} + g(j, S - \{j\})\} \quad (2)$$

اگر برای هر انتخاب  $k$ ، مقدار  $g(k, V - \{1, k\})$  را بدانیم، می‌توانیم (1) را برای به دست آوردن  $\{1, V - \{1, k\}\}$  حل کنیم. این کار را به صورت زیر می‌توان انجام داد:

برای حالت  $|S|=0$ ،  $g(i, \phi) = c_{i,i}$ ، بنابراین رابطه بازگشتی (2) را می‌توان نخست برای حالت  $|S|=1$ ، بعد برای حالت  $|S|=2$  و الی آخر به دست آورد. برای  $|S| < n-1$ ، مقادیر  $i$  و  $S$  لازم برای محاسبه  $g(i, S)$  باید در شرایط  $i \in S, i \neq 1, 1 \notin S$  و  $i \in S$  صادق باشند.

مثال: با استفاده از فرمول‌های ارایه شده در (1) و (2) یک دور هامیلتونی مینیمم برای گراف زیر به دست آورید:

1	2	3	4
1	0 10 15 20		
2	5 0 9 10		
3	6 13 0 12		
4	8 8 9 0		



حل:

$$|S|=0 \Rightarrow g(2, \phi) = C_{21} = 5 \quad g(3, \phi) = C_{31} = 6 \quad g(4, \phi) = C_{41} = 8$$

$$|S|=1 \Rightarrow g(2, \{3\}) = C_{23} + g(3, \phi) = 9 + 6 = 15 \quad g(2, \{4\}) = C_{24} + g(4, \phi) = 10 + 8 = 18$$

$$g(3, \{2\}) = 18 \quad g(3, \{4\}) = 20$$

$$g(4, \{2\}) = 13 \quad g(4, \{3\}) = 15$$

$$|S|=2 \Rightarrow g(2, \{3, 4\}) = \min \{C_{23} + g(3, \{4\}), C_{24} + g(4, \{3\})\} = 25$$

$$g(3, \{2, 4\}) = \min \{C_{32} + g(2, \{4\}), C_{34} + g(4, \{2\})\} = 25$$

$$g(4, \{2, 3\}) = \min \{C_{42} + g(2, \{3\}), C_{43} + g(3, \{2\})\} = 23$$

$$|S|=3 \Rightarrow g(1, \{2, 3, 4\}) = \min \{C_{12} + g(2, \{3, 4\}), C_{13} + g(3, \{2, 4\}), C_{14} + g(4, \{2, 3\})\} = 35$$

دور هامیلتونی با هزینه مینیمم را می‌توان به صورت زیر به دست آورد. در هر مرحله مقدار زای را که از رابطه (2) برای  $(i, S)$  حاصل می‌شود، ذخیره می‌کنیم. فرض کنید که  $(i, S)$  زاین مقدار باشد.

$j(1, \{2, 3, 4\}) = 2 \Rightarrow (1, 2)$  یال دور هامیلتونی است

بقیه دور هامیلتونی را می‌شود از  $(2, \{3, 4\})$  به دست آورد:

$J(2, \{3, 4\}) = 4 \Rightarrow (4, 3)$  یال دور هامیلتونی است

$J(4, \{3\}) = 3 \Rightarrow (4, 3)$  یال دور هامیلتونی است

بنابراین دور هامیلتونی با هزینه مینیمم عبارت است از:  $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$

فرض کنید  $N$  تعداد  $(i, S)$  های لازم برای محاسبه باشد. برای هر  $k=1, |S|=n-1$  انتخاب برای  $i$  وجود دارد. تعداد زیر مجموعه‌های

مجازایی که شامل  $i$  نباشد برابر با  $C_{n-2}^k$  است. بنابراین  $N = \sum_{k=0}^{n-2} (n-1) C_{n-2}^k$  در نتیجه  $(n-1) 2^{n-2} = N$ . اما هر یک از

گها مستلزم حداقل  $n$  مقایسه (دقیقاً  $k$  مقایسه، اگر  $|S|=k$ ) است. بنابراین پیچیدگی محاسباتی الگوریتم مزبور برابر  $O(n^2 2^n)$  است.

## مسئله ضرب زنجیری از ماتریس‌ها

فرض کنید رشته‌ای (زنجیری) از  $n$  ماتریس  $\langle A_1, A_2, \dots, A_n \rangle$  داده شده و هدف محاسبه ضرب ماتریسی  $A_1 A_2 \dots A_n$

است. برای این‌که انجام محاسبه عبارت بالا بدون ابهام صورت پذیرد باید آن را به طور کامل پرانتزگذاری کرد. یک ضرب ماتریسی را به‌طور کامل پرانتزگذاری شده گوییم هر گاه از ماتریسی واحد و یا از دو ضرب ماتریسی به‌طور کامل پرانتزگذاری شده و محصور بین دو پرانتز تشکیل شده باشد. با توجه به این‌که ضرب ماتریس‌ها شرکت‌پذیر است، همهٔ پرانتزگذاری‌های کامل نتیجهٔ یکسانی خواهد داشت. برای مثال برای زنجیر ماتریسی  $\langle A_1, A_2, A_3, A_4 \rangle$ ، حاصل ضرب  $A_1 A_2 A_3 A_4$  را می‌توان به پنج حالت متمایز به صورت زیر به‌طور کامل پرانتزگذاری کرد:

$$(A_1 (A_2 (A_3, A_4))) \quad (\text{الف})$$

$$((A_1 (A_2 A_3)) A_4) \quad (\text{ب})$$

$$((A_1 A_2) (A_3 A_4)) \quad (\text{ج})$$

$$(A_1 ((A_2 A_3) A_4)) \quad (\text{د})$$

$$(((A_1 A_2) A_3) A_4) \quad (\text{ر})$$

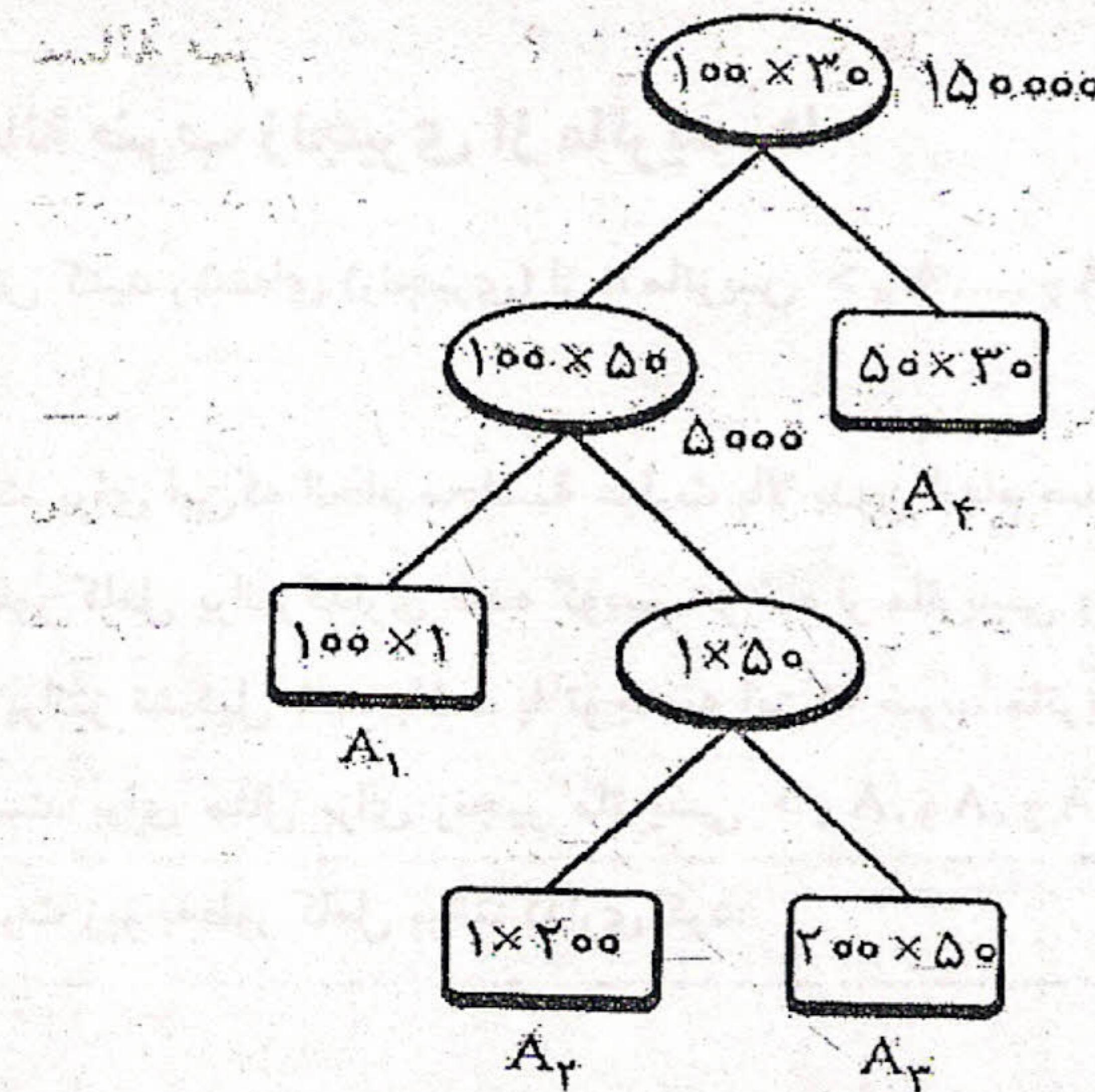
ماتریس نتیجهٔ همهٔ این ضرب‌ها یکسان است، ولی تعداد ضرب‌های اسکالر به کار رفته می‌تواند برای بعضی از آن‌ها چشمگیر باشد. برای درک این مطلب، فرض کنید که برای ضرب دو ماتریس از روش معمولی استفاده می‌کنیم. در فصل تقسیم و غلبه دیدیم که اگر ماتریسی  $A$  ماتریسی  $m \times p$  و  $B$  ماتریسی  $p \times n$  باشد، تعداد ضرب‌های اسکالر به کار رفته برای  $AB$  برابر  $mnp$  است. برای به تصویر کشیدن موضوع فرض کنید اندیازهای چهار ماتریس  $A_1$  و  $A_2$  و  $A_3$  و  $A_4$  به ترتیب عبارت‌اند از  $1 \times 200$ ،  $100 \times 1$ ،  $50 \times 50$  و  $50 \times 30$ . اگر این چهار ماتریس را به صورت  $(\text{ف})$  در هم ضرب کنیم، تعداد ضرب‌های اسکالر به کار رفته برابر  $309000$  است. این عدد مجموع سه کمیت زیر است. برای محاسبه  $A_3 A_4$  به  $300000 = 300 \times 50 \times 30$  ضرب اسکالر نیاز است. نتیجه، ماتریسی  $200 \times 30$  مثل 34 است. حال باید حاصل ضرب  $A_2 A_3$  را به‌دست آوریم. برای به‌دست آوردن این نتیجه به  $6000 = 200 \times 30$  ضرب اسکالر دیگر نیاز داریم. تا این‌جا، حاصل ضرب، ماتریسی  $1 \times 30$  مثل 24 است که باید به  $A_1$  ضرب شود. برای به‌دست آوردن نتیجهٔ نهایی به  $3000 = 100 \times 1 \times 30$  ضرب اسکالر نیاز داریم. مجموع سه عدد یاد شده برابر  $309000$  است.

بنابراین اگر هزینهٔ زنجیر ماتریسی را برابر با تعداد ضرب‌های اسکالر به کار رفته برای رسیدن به نتیجه در نظر بگیریم، هزینهٔ زنجیر ماتریسی  $\text{الف}$  برابر  $309000$  است. نتایج برای چهار حالت باقیمانده در شکل زیر آمده است. این شکل نشان می‌دهد که ضرب این چهار ماتریس به صورت  $(\text{د})$  حدود 80 برابر سریع‌تر از ضرب آن‌ها به صورت  $(\text{ر})$  است.

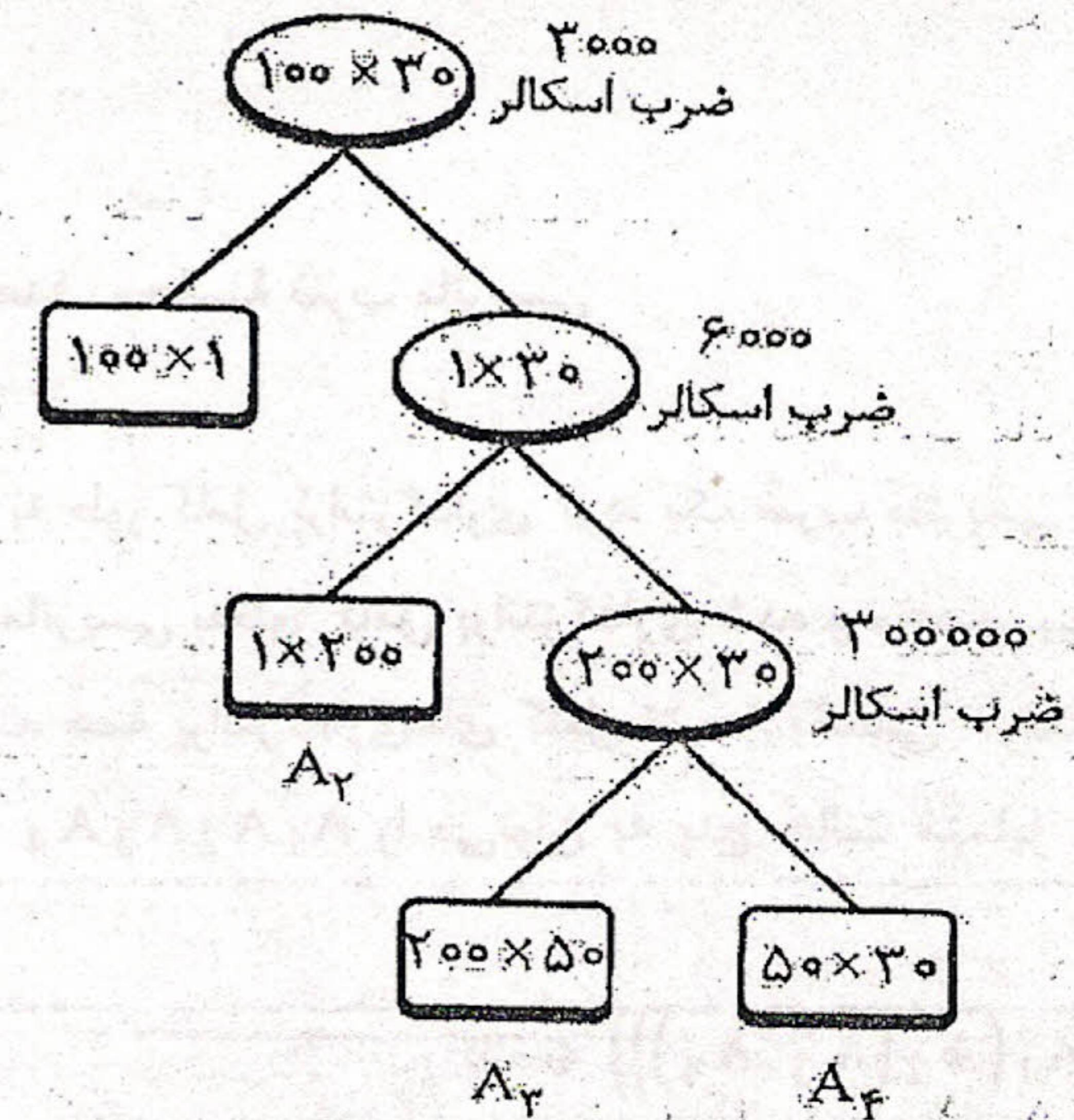
تعداد حالت‌هایی که  $(n+1)$  ماتریس را می‌توان در هم ضرب کرد برابر با تعداد درخت‌های دودویی است که با  $1 + n$  گره خارجی (یا  $n$  گره داخلی) می‌توان تشکیل داد. قبلًا مشاهده کردیم که تعداد این درخت‌ها برابر عدد کاتالان  $C_{2n}^n = \frac{1}{n+1} \Omega\left(\frac{4^n}{n^3/2}\right)$  است که تابع رشد آن به

$$\text{صورت} \quad \Omega\left(\frac{4^n}{n^3/2}\right) \text{ است.}$$

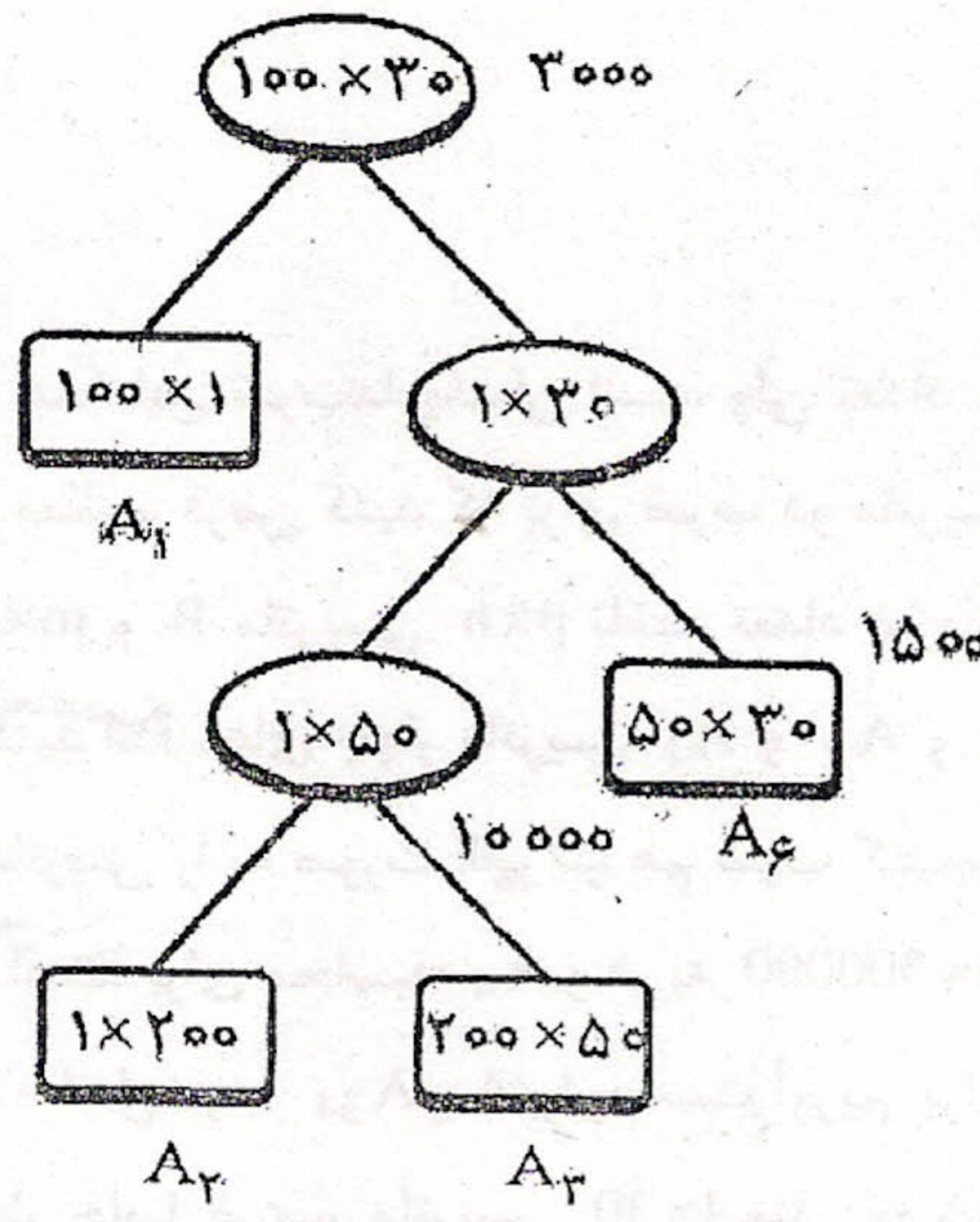
حل مسئله زنجیر ماتریسی به روش سعی و خطایعنی بررسی همهٔ پرانتزگذاری‌های کامل و انتخاب پرانتزگذاری کامل بهینه برای  $n$  های بزرگ امکان‌پذیر نیست. با استفاده از روش برنامه‌نویسی پویا می‌توان الگوریتمی کارآ با مرتبه بزرگی  $\Theta(n^3)$  برای حل این مسئله ارایه داد.



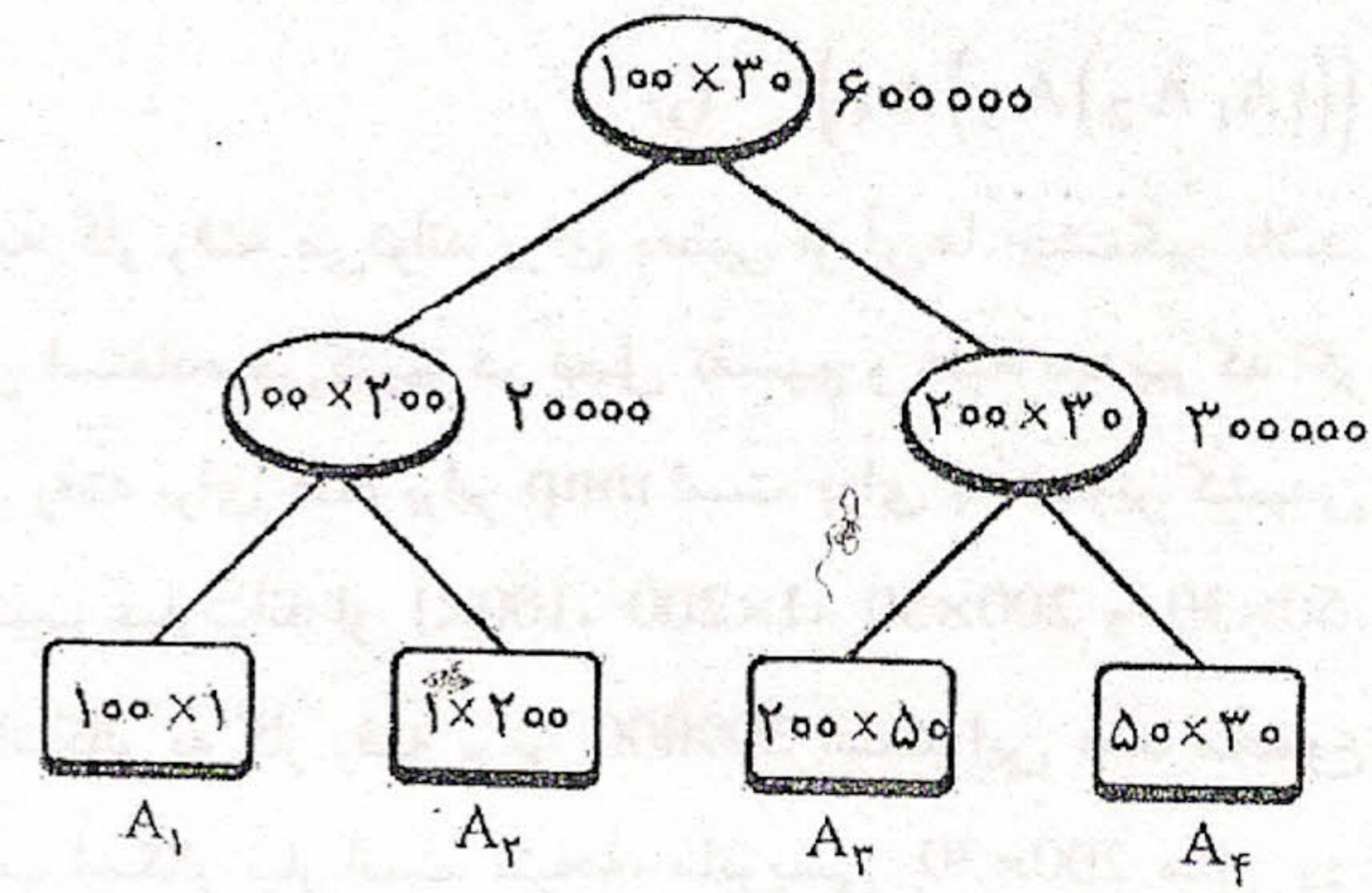
(ب) تعداد ضرب‌های اسکالر به کار رفته برابر 165000 است.



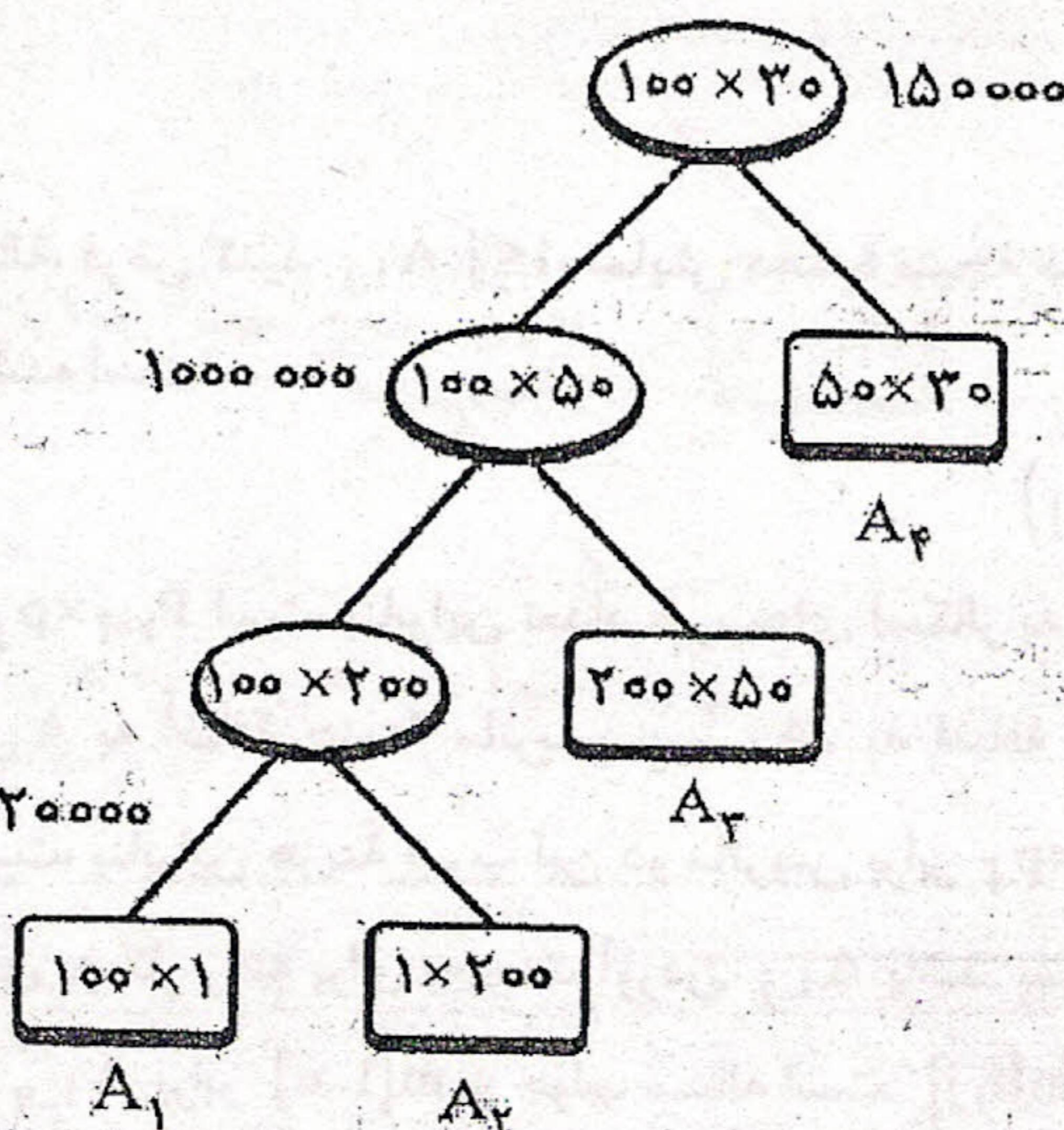
(الف) تعداد ضرب‌های اسکالر برابر 309000=300000+6000+3000 است.



(د) تعداد ضرب‌های اسکالر به کار رفته برابر 14500 است.



(ج) تعداد ضرب‌های اسکالر به کار رفته برابر 920000 است.



(ر) تعداد ضربهای اسکالار به کار رفته برابر ۱۱۷۰ ۰۰۰ است.

مسالة ضرب زنجیر ماتریسی را می‌توان به صورت زیر بیان کرد. فرض کنید زنجیری از  $n$  ماتریس  $\langle A_1, A_2, \dots, A_n \rangle$  داده شده که در آن اندازه ماتریس  $A_i$ ,  $i=1, 2, \dots, n$ , برابر  $p_{i-1}^* p_i$  است. هدف، پرانتزگذاری کامل عبارت  $A_1 A_2 \dots A_n$  به گونه‌ای است که تعداد ضربهای اسکالار به کار رفته حداقل باشد. در واقع در مسالة ضرب زنجیر ماتریسی، هدف ضرب ماتریس‌ها نیست، بلکه می‌خواهیم صرفاً ترتیبی برای ضرب این ماتریس‌ها به دست آوریم که کمترین هزینه را داشته باشد. مثال قبل، ارزش این سرمایه‌گذاری زمانی را برای تعیین ترتیبی پنهانه برای ضرب ماتریس‌ها قبل از ضرب خود ماتریس‌ها نشان می‌دهد (۱۴۵۰۰ ضرب اسکالار به جای ۱۱۷۰ ۰۰۰).

## ارایه راه حل پویا برای مساله

برای بهدست آوردن راه حلی پویا برای این مساله، فرض کنید  $A_{i,j} \dots A_{i,k}$  نمایش دهنده نتیجه ضرب  $A_i A_{i+1} \dots A_j$  است. برای  $j < i$ ، فرض کنید  $A_{i,j}$  به صورت زیر پرانتزگذاری شده است:

$$(A_i A_{i+1} \dots A_k)(A_{k+1} A_{k+2} \dots A_j) \quad (1)$$

با توجه به این که اندازه ماتریس  $A_{j..i}$  برابر  $p_{j..i} = p_{i-1} \times p_{i-2} \times \dots \times p_j$  است، بنابراین تعداد ضربهای اسکالر به کار رفته (هزینه) برای بهدست آوردن  $A_{i..j}$  برابر خواهد بود با هزینه ماتریس  $A_{i..k} A_{k+1..j}$ ، به اضافه هزینه ماتریس  $A_{k..j}$ . چون  $A_{i..k} A_{k+1..j}$  یک ماتریس  $p_{i..k} \times p_k \times p_{k..j}$  است، بنابراین هزینه ضرب این دو ماتریس برابر  $p_{i..k} \times p_k \times p_{k..j}$  خواهد بود.

فرض کنید که  $m[i,j]$  حداقل تعداد ضربهای به کار رفته برای بهدست آوردن  $A_{i..j}$  باشد. بدیهی است که حداقل تعداد ضربهای به کار رفته برای محاسبه  $A_{1..n} = A_1 A_2 A_3 \dots A_n$  برابر  $m[1..n]$  و جواب مساله است.  $m[i,j]$  را می‌توان به صورت بازگشتی تعریف کرد. اگر  $j = i$ ، حل مساله بدیهی است. زنجیر، فقط شامل ماتریس  $A_{i..i} = A_i$  است و نیازی به هیچ ضرب اسکالر نیست. بنابراین برای  $m[i,i] = 0, i = 1, 2, \dots, n$  برای حالت  $j < i$ ، فرض کنید جواب بهینه برای محاسبه  $A_{i..j}$  از حاصل ضرب  $A_{i..k} A_{k+1..j}$  به دست آمده است. بنا به اصل بهینه‌سازی و مطالب گفته شده باید داشته باشیم:

$$m[i,j] = m[i,k] + m[k+1,j] + p_{i-1} \times p_k \times p_j \quad (2)$$

برای نوشتن رابطه 2 فرض بر این است که مقدار  $k$  از قبل معلوم است. از آن جا که فقط  $i-j$ -مقدار ممکن برای  $k$  به کار رفته در رابطه 2 وجود دارد، مشخصاً  $j-1, i+1, \dots, i$ ، بنابراین، تعریف بازگشتی برای حداقل تعداد ضربهای اسکالر به کار رفته برای محاسبه  $A_1 A_2 \dots A_n$  به صورت زیر خواهد بود.

$$m[i,j] = \min_{i \leq k < j} \{m[i,k] + m[k+1,j] + p_{i-1} \times p_k \times p_j\} \quad (3)$$

اگر فرمول 3 را به صورت یک پردازه بازگشتی پیاده‌سازی کنیم، زمان اجرای آن نمایی خواهد بود ولی اگر  $m$  را به صورت ماتریس  $n \times n$  نگاه کنیم، هزینه مینیمم  $m[1..n]$  را می‌توان با الگوریتمی از مرتبه  $\Theta(n^3)$  به دست آورد.

از آنجا که  $m[i,j]$  ارایه شده در رابطه 3 فقط مقادیر هزینه‌های مینیمم را برای مسایل به دست می‌دهد لذا برای این که به نحوه پرانتزگذاری برای رسیدن به این هزینه‌های مینیمم پی ببریم از ماتریسی کمکی  $n \times n$  مثل  $s$  استفاده می‌کنیم. در  $[j,i]$  مقدار  $s$  ای را قرار خواهیم داد که عبارت 3 را مینیمم می‌کند.

با توجه به مطالب گفته شده، روش پویا برای حل مساله زنجیر ماتریس‌ها به صورت زیر خلاصه می‌شود. ورودی برای این مساله را می‌توان آرایه‌ای به صورت  $P[0..n]$  در نظر گرفت. اندازه ماتریس  $A_i$  به صورت  $P[i-1] \times P[i]$  بیان می‌شود. با توجه به شرط  $m[i,j]$  را می‌توان به صورت ماتریس بالا مثلثی نگاه کرد. عناصر این ماتریس بالا مثلثی را برای رسیدن به جواب بهینه  $m[1..n]$  به صورت زیر پر می‌کنیم. از شرط مرزی  $m[i,i] = 0, i = 1, 2, \dots, n$ ، همه عناصر روی قطر اصلی ماتریس را مساوی صفر قرار می‌دهیم. برای حالت  $i = j$  (عناصر بالای قطر اصلی)، از فرمول 3 داریم.

$$\begin{aligned} m[i,i+1] &= m[i,i] + m[i+1,i+1] + p[i-1]p[i] \times p[i+1] \\ &= p[i-1] \times p[i] \times p[i+1], \quad i = 1, 2, \dots, n-1 \end{aligned} \quad (4)$$

سپس برای حالت  $i = j-1$  (عناصر دومین قطر روی قطر اصلی) نتایج زیر حاصل می‌شوند.

$$m[i,i+2] = \min \{m[i,i] + m[i+1,i+2] + p[i-1] \times p[i] \times p[i+2], m[i,i+1] + m[i+2,i+2] + p[i-1] \times p[i+1] \times p[i+2]\} \quad (5)$$

$$i = 1, 2, \dots, n-2$$

بدین ترتیب با توجه به این که همه کمیت‌های ظاهر شده در ۵ قبلاً محاسبه و در ماتریس  $m$  ذخیره شده‌اند، مساله برای حالت  $j-i=2$  نیز حل می‌شود.

با ادامه این روند مساله به ترتیب برای  $j-i=3, \dots, j-i=n-1$  حل می‌شود. برای حالت  $j-i=n-1$ ، بدیهی است که  $i=1$  و  $n=j$  است. بنابراین در این حالت فقط عنصر  $[1, n]_m$ ، یعنی جواب نهایی مساله را باید حساب کنیم.

مطلوب گفته شده در الگوریتم زیر خلاصه شده‌اند. تنها ورودی برای این الگوریتم آرایه  $P[0..n]$  است که در آن اندازه ماتریس  $p[i-1] \times p[i]$  است. در الگوریتم مزبور از آرایه کمکی  $s$  برای رسیدن به نحوه پرانتزگذاری جواب بهینه استفاده شده است. در  $[i, j]_s$  مقدار  $k$  ای ذخیره می‌شود که کمیت ۳ را مینیمم می‌کند.

Algorithm Matrix\_Chain\_Order( $p$ )

```

for i=1 to n do m[i,i]=0
for l=1 to n-1 do
    for i=1 to n-1 do
        {
            j=i+1
            m[i,j]=∞
            for k=i to j-1 do
                q=m[i,k]+m[k+1,j]+p[i-1]×p[k]×p[j] (*)  

                If q < m[i,j] then
                {
                    m[i,j]=q
                    s[i,j]=k
                }
        }
    return m,s

```

این الگوریتم ماتریس  $m$  را قطر به قطر با شروع از قطر اصلی پر می‌کند. ابتدا عناصر قطر اصلی مقداردهی می‌شوند. تعداد عناصر قطر اصلی برای  $n$  است. سپس عناصر بالای قطر اصلی پر می‌شوند. تعداد این عناصر  $(n-1)$  است و با توجه به رابطه ۵، نیازی به مینیمم سازی نیست. در حالت عمومی تعداد عناصر قطر  $k$ ام روی قطر اصلی، برابر  $n-k$  است و نیاز به مینیمم‌گیری روی  $k$  کمیت است. بنابراین مرتبه بزرگی الگوریتم بالا را می‌توان از رابطه زیر بدست آورد.

$$\sum_{k=1}^{n-1} k(n-k) = \Theta(n^3)$$

مثال: مساله ضرب بهینه را برای زنجیر ماتریس‌های  $A_1 A_2 A_3 A_4$  و  $p[0..4]=[100, 1, 200, 50, 30]$  حل کنید. در اینجا ماتریسی بالا مثلثی  $4 \times 4$  است. در ابتدا داریم:

$$i=1, 2, 3, 4, \quad m[i,i]=0$$

برای  $j-i=1$  (عناصر روی قطر اصلی) داریم:

$m[1,2]=p[0] \times p[1] \times p[2]=20\ 000$	$s[1,2]=1$
$m[2,3]=p[1] \times p[2] \times p[3]=10\ 000$	$s[2,3]=2$
$m[3,4]=p[2] \times p[3] \times p[4]=30\ 000$	$s[3,4]=3$

برای حالت  $i=2, j=3$  (دومین عنصر زوی قطر اصلی) داریم:

$$m[1,3] = \min \left\{ \begin{array}{l} m[1,1]+m[2,3]+p[0]\times p[1]\times p[3], \\ m[1,2]+m[3,3]+p[0]\times p[2]\times p[3] \end{array} \right\}$$

$$= \min \{ 0+10000+5000, 20000+0+1000000 \} = 150000$$

$$s[1,3]=1$$

$$m[2,4] = \min \left\{ \begin{array}{l} m[2,2]+m[3,4]+p[1]\times p[2]\times p[4], \\ m[2,3]+m[4,4]+p[1]\times p[3]\times p[4] \end{array} \right\}$$

$$= \min \{ 0+300000+6000, 10000+0+1500 \} = 11500$$

$$s[2,4]=2$$

و سرانجام برای  $i=3, j=3$  جواب نهایی زیر به دست می‌آید.

$$m[1,4] = \min \left\{ \begin{array}{l} m[1,1]+m[2,4]+p[0]\times p[1]\times p[4], \\ m[1,2]+m[3,4]+p[0]\times p[2]\times p[4], \\ m[1,3]+m[4,4]+p[1]\times p[3]\times p[4] \end{array} \right\}$$

$$= \min \left\{ \begin{array}{l} 0+11500+3000, \\ 20000+300000+600000, \\ 15000+0+150000 \end{array} \right\}$$

$$= 14500$$

$$s[1,4]=1$$

حال برای این که به نحوه پرانتزگذاری ماتریس‌ها برای رسیدن به این جواب بهینه پی بریم از ماتریس کمکی  $s$  کمک می‌گیریم. با توجه به این که  $s[1,4]=1$ ، نتیجه می‌شود که ماتریس‌ها به صورت  $A_1(A_2 A_3 A_4)$  در هم ضرب شده‌اند:

$$A_{1..4} = A_{1..1} A_{2..4}, \quad k=1$$

حال برای این که ببینیم  $A_{2..4}$  چگونه به دست آمده است به  $s[2,4]=2$  مراجعه می‌کنیم. چون  $s[2,4]=2$  است بنابراین  $A_{1..4} = A_1((A_2 A_3) A_4)$  و در نتیجه، نحوه پرانتزگذاری این چهار ماتریس به صورت  $(A_1((A_2 A_3) A_4))$  است.

1	2	3	4
	1	1	1
		2	2
			3

ماتریس  $s$

1	2	3	4
0	20000	15000	14500
	0	10000	11500
		0	300000
			0

ماتریس  $m$

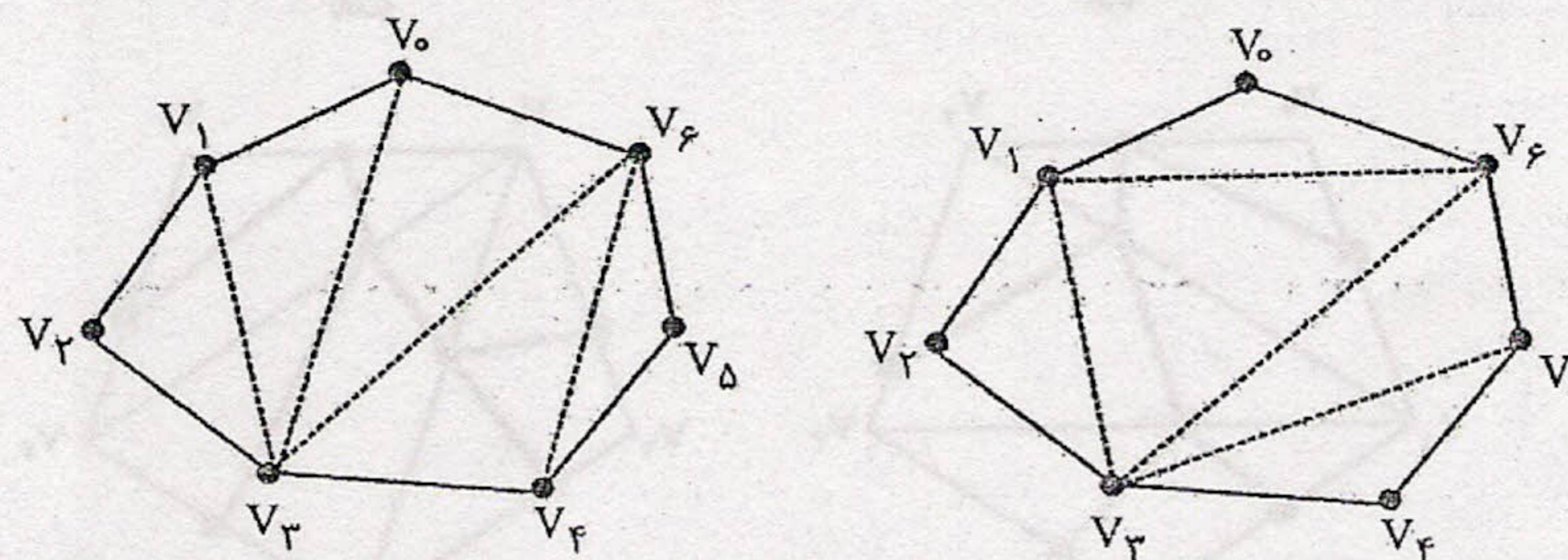
## مثلثبندی بهینه یک چندضلعی محدب

در این بخش به مسأله مثلثبندی یک چندضلعی محدب می‌پردازیم. این مسأله هندسی علی‌رغم اختلاف ظاهری مشابهت زیادی با مسأله ضرب زنجیر ماتریس‌ها دارد.

یک چندضلعی را محدب گوییم هر گاه اگر دو نقطه دلخواه در داخل آن انتخاب کنیم، پاره خط وصل بین این دو نقطه، به طور کامل در داخل این چندضلعی قرار بگیرد. یک  $n$ -ضلعی محدب را می‌توان با فهرست کردن رئوس آن به ترتیب خلاف چرخش عقربه‌های

ساعت به صورت  $P = \langle V_0, V_1, \dots, V_{n-1} \rangle$  نمایش داد.  $n$  ضلع این چندضلعی محدب عبارت‌اند از  $\overline{V_{n-1}V_0}, \overline{V_0V_1}, \dots, \overline{V_{n-2}V_{n-1}}, \overline{V_{n-1}V_0}$

مثلثبندی یک چندضلعی محدب، مجموعه‌ای مثل  $T$  از قطرهای است به گونه‌ای که چندضلعی محدب را به مثلثهای متمایز تقسیم کند. شکل زیر دو نمونه از نحوه مثلثبندی یک 7-ضلعی محدب را نشان می‌دهد. در مثلثبندی، هیچ دو قطر هم‌دیگر را (به جز در رئوس) قطع نمی‌کنند و مجموعه  $T$  نیز ماکزیمال است (هر قطری که در  $T$  نباشد، بعضی از قطرهای  $T$  را قطع می‌کند). هر مثلثبندی بر روی یک  $n$ -ضلعی محدب 3-قطر دارد و چندضلعی را به  $n-2$  مثلث تقسیم می‌کند.



دو نمونه از نحوه مثلثبندی یک 7-ضلعی محدب. هر مثلثبندی یک 7-ضلعی محدب 3-قطر دارد که آن را به 5-7 مثلث تقسیم می‌کند.

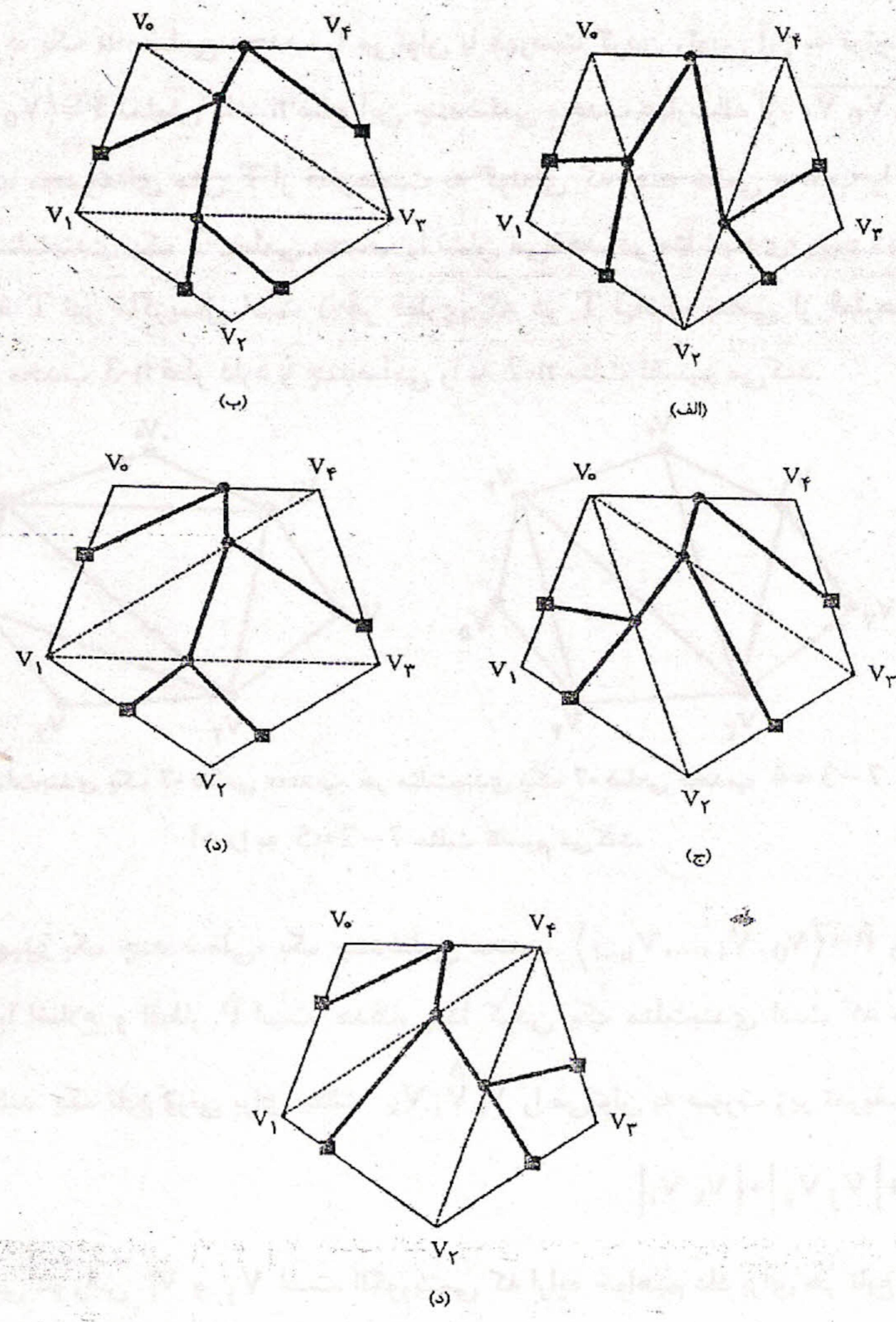
مفروضات مسأله در مثلثبندی بهینه یک چندضلعی، یک چندضلعی محدب  $P = \langle V_0, V_1, \dots, V_{n-1} \rangle$  و یک تابع وزنی تعريف شده بر روی مثلثهای تشکیل شده با اضلاع و اقطار  $P$  است. هدف، پیدا کردن یک مثلثبندی است که مجموع وزن‌های منسوب به مثلثهای مثلثبندی را مینیمم کند. یک تابع وزنی برای مثلث  $\Delta_{V_i V_j V_k}$  را می‌توان به صورت زیر تعريف کرد.

$$W\left(\Delta_{V_i V_j V_k}\right) = |V_i V_j| + |V_j V_k| + |V_k V_i|$$

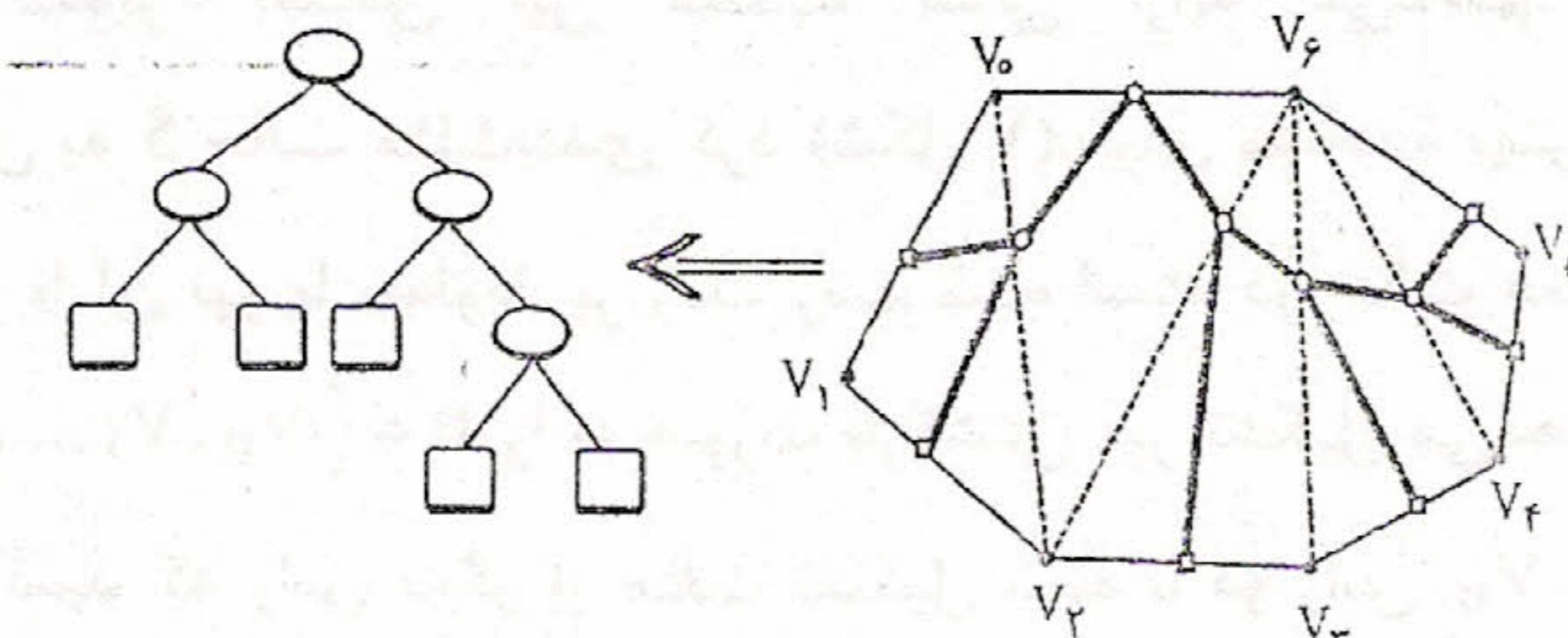
که در آن  $|V_i V_j|$  فاصله اقلیدسی دو راس  $V_i$  و  $V_j$  است. الگوریتمی که ارایه خواهیم داد برای هر تابع وزنی کار می‌کند.

بین مسأله مثلثبندی یک چندضلعی محدب و مسأله ضرب زنجیر ماتریس‌ها و در نتیجه تعداد درخت‌های دودویی، یک تناظر شگفت‌آور وجود دارد. برای به تصویر کشیدن این مطلب، مثالی ارایه می‌دهیم. یک 5-ضلعی محدب مثل  $P = \langle V_0, V_1, V_2, V_3, V_4 \rangle$  را می‌توان به 5 حالت مثلثبندی کرد (شکل ۱). برای مشاهده بهتر ارتباط اشاره شده در بالا به همراه هر مثلثبندی، درخت دودویی متناظر با آن نیز با خطوط پر رنگ رسم شده است. در حالت عمومی درخت دودویی متناظر با یک مثلثبندی بر روی  $n$ -ضلعی  $P = \langle V_0, V_1, \dots, V_{n-1} \rangle$  را به صورت بازگشتی زیر تشکیل می‌دهیم. ریشه درخت دودویی را بر روی ضلع  $\overline{V_0 V_{n-1}}$  قرار می‌دهیم. فرض کنید که راس دیگر از مثلث تشکیل شده با دو راس  $V_0$  و  $V_{n-1}$  در مثلثبندی مذبور راس  $V_k$  است ( $0 < k < n-1$ ). یکی از مثلثهای مثلثبندی است. ریشه فرزند چپ درخت دودویی را بر روی ضلع  $\Delta_{V_0 V_k V_{n-1}}$  رسم می‌کنیم. اگر  $\overline{V_0 V_k}$  و ریشه فرزند راست را بر روی ضلع  $\overline{V_0 V_k}$  ضلعی از چندضلعی است فقط یک برگ بر

روی آن رسم خواهیم کرد و اگر قطری از چندضلعی است یک گره داخلی بر روی آن رسم می‌کنیم و رسم زیر درخت چپ را به صورت بازگشتی بر روی مثلثبندی تشکیل شده بر روی چند ضلعی محدب  $\langle V_0, V_1, \dots, V_k \rangle$  ادامه می‌دهیم. به طریق مشابه اگر ضلع  $V_k V_{n-1}$  از چند ضلعی محدب است، فقط یک برگ را بر روی آن رسم خواهیم کرد در غیر این صورت بر روی ضلع  $V_k V_{n-1}$  گرهی داخلی به عنوان ریشه زیر درخت سمت راست رسم کرده و این عمل را به صورت بازگشتی بر روی مثلثبندی انجام شده بر روی چندضلعی محدب  $\langle V_k, V_{k+1}, \dots, V_{n-1} \rangle$  ادامه می‌دهیم. شکل ۲ مطالب گفته شده را به تصویر کشیده است.



شکل ۱



شکل ۲ نحوه رسم یک درخت دودویی متناظر با یک مثلثبندی.

### ساختار یک مثلثبندی بهینه

یک مثلثبندی بهینه  $T$  بر روی یک  $P = \langle V_0, V_1, \dots, V_n \rangle$  را در نظر بگیرید که شامل مثلث

است. هزینه  $T$  برابر است با وزن  $V_0 V_k V_n$  به اضافه وزن مثلثهای مثلثبندی تشکیل شده بر روی  $\Delta$   $V_0 V_k V_n$  چند ضلعی محدب  $\langle V_0, V_1, \dots, V_k \rangle$  به اضافه وزن مثلثهای مثلثبندی تشکیل شده بر روی چند ضلعی محدب  $\langle V_k, V_{k+1}, \dots, V_n \rangle$ . بنا به اصل بهینه‌سازی، هزینه هر کدام از این دو مثلثبندی باید مینیمم باشد.

همان‌گونه که در مسأله ضرب زنجیره‌ای از ماتریس‌ها،  $[i, j] m[i, j]$  هزینه مینیمم برای محاسبه حاصل ضرب  $A_i A_{i+1} \dots A_j$  بود، در اینجا نیز  $[j, i] t[j, i]$  را وزن یک مثلثبندی بهینه تشکیل شده بر روی چند ضلعی محدب  $\langle V_{i-1}, V_i, \dots, V_j \rangle$  تعریف می‌کنیم. برای سهولت، وزن چند ضلعی زوال یافته  $\langle V_{i-1}, V_i \rangle$  را برابر ۰ قرار می‌دهیم. بنابراین، وزن یک مثلثبندی بهینه برای چند ضلعی  $P$  توسط  $t[1, n]$  داده می‌شود. حال با استفاده از اصل بهینه‌سازی، یک رابطه بازگشتی برای  $t[i, j]$  ارایه می‌دهیم. با توجه به تعریف، برای یک ۱-ضلعی زوال یافته  $\langle V_{i-1}, V_i \rangle$ ، شرط مرزی  $t[i, i] = 0$ ، را داریم. برای حالت  $j < i$ ، چند ضلعی  $\langle V_{i-1}, V_i, \dots, V_j \rangle$  را داریم که حداقل دارای ۳ راس است.

فرض کنید  $k = i, i+1, \dots, j-1, V_k$  راسی باشد. که با دو راس  $V_{i-1}$  و  $V_j$  یک مثلث در مثلثبندی تشکیل می‌دهد (شکل زیر). در این صورت وزن یک مثلثبندی تشکیل شده بر روی چند ضلعی محدب  $\langle V_{i-1}, V_i, \dots, V_j \rangle$  برابر مجموعه سه کمیت زیر است:

الف) وزن مثلث  $V_{i-1} V_k V_j$

ب) وزن مثلثبندی تشکیل شده بر روی چند ضلعی محدب  $\langle V_{i-1}, V_i, \dots, V_k \rangle$

ج) وزن مثلثبندی تشکیل شده بر روی چند ضلعی محدب  $\langle V_k, V_{k+1}, \dots, V_j \rangle$

طبق اصل بهینه‌سازی برای این که وزن این مثلثبندی مینیمم باشد ( $t[i, j]$ ، اولاً باید وزن مثلثبندی قسمت (ب) مینیمم باشد ( $t[i, k]$ ، وزن مثلثبندی قسمت (ج) مینیمم باشد ( $t[k+1, j]$ ) و  $k$  نیز باید به گونه‌ای انتخاب شود که کمیت زیر مینیمم شود.

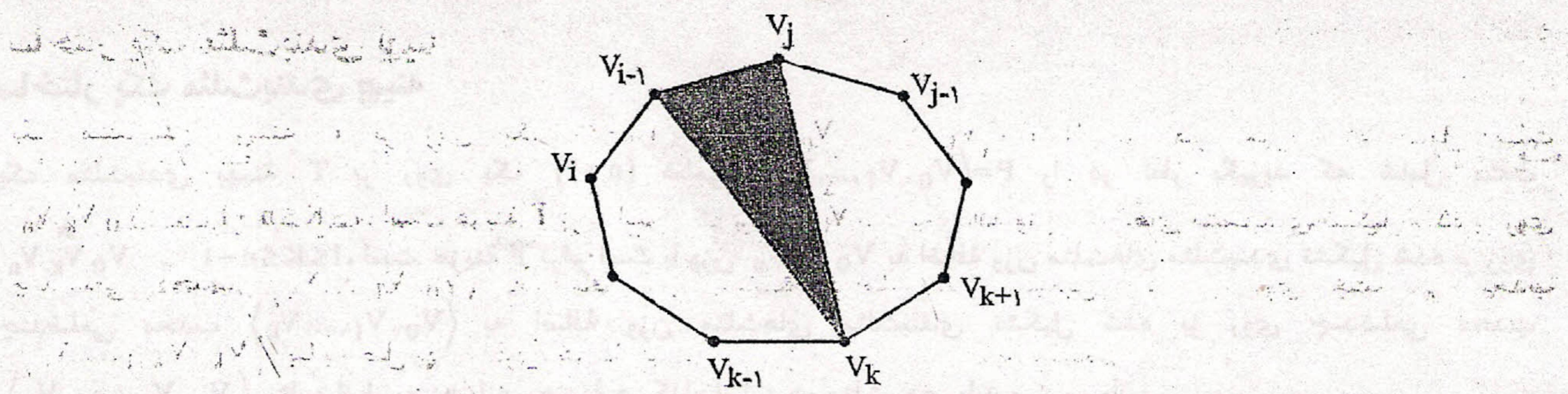
$$t[i, k] + t[k+1, j] + w(V_{i-1} V_k V_j), \quad i \leq k < j$$

به عبارت دیگر، رابطه بازگشتی زیر را برای یک مثلثبندی بهینه داریم:

$$\begin{cases} t[i, j] = \min_{i \leq k < j} \left\{ t[i, k] + t[k+1, j] + w(V_{i-1} V_k V_j) \right\}, & i < j \\ t[i, i] = 0 \end{cases}$$

۹

اگر این رابطه بازگشتی را با رابطه بازگشتی ارایه شده برای حداقل تعداد ضربهای  $m[i, j]$  مربوط به ضرب زنجیر ماتریسی  $A_i A_{i+1} \dots A_j$  مقایسه کنید، متوجه می‌شوید که به استثنایتابع وزنی به کار رفته، مشابه‌اند. بنابراین با تغییرات جزئی در الگوریتم Matrix\_Chain\_Order می‌توانیم وزن یک مثلثبندی بهینه را به دست آوریم. مشابه Matrix\_Chain\_Order مثلثبندی بهینه نیز از مرتبه  $\Theta(n^3)$  است.



وزن مثلثبندی چندضلعی محدب  $\langle V_{i-1}, V_i, \dots, V_j \rangle$  برابر مجموع سه کمیت زیر است:

(الف) وزن مثلثبندی چندضلعی  $\langle V_{i-1}, V_i, \dots, V_k \rangle$

(ب) وزن مثلث  $\Delta_{V_{i-1} V_k V_j}$

(ج) وزن مثلثبندی  $\langle V_k, V_{k+1}, \dots, V_j \rangle$

### طولانی‌ترین زیر رشته مشترک (LCS)

رشته  $Z = \langle z_1, z_2, \dots, z_k \rangle$  را یک زیر رشته برای رشته  $X = \langle x_1, x_2, \dots, x_n \rangle$  و  $Y = \langle y_1, y_2, \dots, y_m \rangle$  گوییم هر گاه رشته‌ای اکیداً صعودی مثل  $i_1, i_2, \dots, i_k$  از اندیس‌های  $X$  وجود داشته باشد به گونه‌ای که برای همه  $j = 1, 2, \dots, k$ ، رابطه  $x_{i_j} = z_j = y_{i_{j+1}}$  برقرار باشد. برای مثال  $\langle 2, 3, 5, 7 \rangle$  یک زیر رشته برای رشته  $\langle A, B, C, B, D, A, B \rangle$  است که در آن رشته اندیس‌های مورد اشاره  $\langle B, C, D, B \rangle$  است.

رشته  $Z$  را یک زیر رشته مشترک برای دو رشته مفروض  $X$  و  $Y$  گوییم هر گاه  $Z$  هم زیر رشته  $X$  و هم زیر رشته  $Y$  باشد. برای مثال  $Z = \langle B, C, A \rangle$  یک زیر رشته مشترک برای دو رشته  $X = \langle A, B, C, B, D, A, B \rangle$  و  $Y = \langle B, D, C, A, B, A \rangle$  است. اما این زیر رشته به طول 3، طولانی‌ترین زیر رشته مشترک  $X$  و  $Y$  نیست زیرا  $\langle B, C, B, A \rangle$  نیز یک رشته مشترک برای  $X$  و  $Y$  است که طول آن برابر 4 است. چون هیچ زیر رشته مشترک دیگری به طول بیشتر از 4 برای  $X$  و  $Y$  وجود ندارد، لذا  $\langle B, C, B, A \rangle$  طولانی‌ترین زیر رشته مشترک (LCS) دو رشته  $X$  و  $Y$  است.

مفهومات در مسأله طولانی‌ترین زیر رشته مشترک دو رشته، رشته‌های  $\langle x_1, x_2, \dots, x_n \rangle$  و  $\langle y_1, y_2, \dots, y_m \rangle$  است و هدف این است که طولانی‌ترین زیر رشته مشترک  $X$  و  $Y$  را پیدا کنیم.

روش سعی و خطا برای حل مسأله LCS این است که همه زیر رشته‌های  $X$  را تشکیل داده هر کدام را برای تشخیص این‌ها آیا زیر رشته‌ای از  $Y$  است بررسی کنیم. هر زیر رشته از  $X$  متناظر با زیر مجموعه‌ای از اندیس‌های  $\{1, 2, \dots, n\}$  از  $X$  است. تعداد زیر رشته‌های  $X$  برابر  $n^2$  است بنابراین چنین نگرشی برای حل مسأله، مستلزم زمانی نمایی بوده برای رشته‌های طولانی عملی نیست. با استفاده از اصل بهینه‌سازی می‌توان یک راه حل کارآمد برای این مسأله ارایه داد.

با فرض  $X = \langle x_1, x_2, \dots, x_n \rangle$  پیشوند ام  $X$  برای  $i = 0, 1, 2, \dots, n$  به صورت  $X_i = \langle x_1, x_2, \dots, x_i \rangle$  تعریف می‌شود. برای مثال اگر  $X_3 = \langle A, L, I \rangle$  در این صورت  $X_0 = \langle A, L, I, Z, A, D \rangle$  است. بدیهی است که  $X_0$  دنباله‌تهی و  $X_n = X$  است.

### ارایه یک راه حل پویا برای مسأله

فرض کنید  $Z = \langle z_1, z_2, \dots, z_k \rangle$ ,  $Y = \langle y_1, y_2, \dots, y_m \rangle$  و  $X = \langle x_1, x_2, \dots, x_n \rangle$  برای  $X$  و  $Y$ ,  $k \leq \min\{n, m\}$ . است. در مقایسه دو عنصر آخر  $X$  و  $Y$  دو حالت زیر امکان‌پذیر است.

یا  $x_n = y_m$  که در این حالت بدیهی است  $z_k = x_n = y_m$  و بنا به اصل بهینه‌سازی  $Z_{k-1}$  باید یک  $LCS$  برای  $X_{n-1}$  و  $Y_{m-1}$  باشد.

} (ii)  $z_k \neq x_n$  (i)  $z_k \neq y_m$

ب) یا  $x_n \neq y_m$  که در این حالت

در حالت (ب) اگر  $z_k \neq x_n$ , بنا به اصل بهینه‌سازی  $Z$  برای  $X_{n-1}$  و  $Y$  باشد, در غیر این صورت

$$Z = LCS(X, Y_{m-1})$$

همانند مسأله ضرب زنجیری از ماتریس‌ها، راه حل بازگشتی برای مسأله  $LCS$  مستلزم پیدا کردن رابطه بازگشتی برای مقدار جواب بهینه است. برای این منظور ماتریس  $c$  را به صورت زیر تعریف می‌کنیم. فرض کنید  $c[i, j]$  برابر با طول  $LCS$  دو رشته  $X_i$  و  $Y_j$  باشد. شرط مرزی برای رابطه بازگشتی مورد نظر به ازای مقادیر  $i=0$  یا  $j=0$  حاصل می‌شود. در این حالت یکی از رشته‌ها تهی بوده و طول هر زیررشته مشترک برابر صفر است. با توجه به مطالعه گفته شده نتایج زیر را برای مسأله  $LCS$  داریم:

$$c[i, j] = \begin{cases} 0 & j=0, i=0 \\ c[i-1, j-1]+1 & j>0, i>0, x_i=y_j \\ \max\{c[i, j-1], c[i-1, j]\} & j>0, i>0, x_i \neq y_j \end{cases}$$

ورودی برای الگوریتم  $LCS\_Length$  که در زیر ارایه می‌شود دو رشته  $X_n = \langle x_1, x_2, \dots, x_n \rangle$  و  $Y_m = \langle y_1, y_2, \dots, y_m \rangle$  است. مقادیر برای  $c[i, j]$  در ماتریس  $c[0..n, 0..m]$  ذخیره می‌شود. این مقادیر به ترتیب غالب سطحی محاسبه می‌شوند یعنی نخست سطر 0 از  $c$  از چپ به راست پر می‌شود بعد سطر دوم و سطح بعدی به همین ترتیب پر می‌شوند. برای به دست آوردن خود رشته  $LCS$  از ماتریس  $c$  استفاده شده است. در  $c[i, j]$  اشاره‌گری قرار داده می‌شود که مشخص کننده انتخاب جواب بهینه برای زیرمسائل متناظر با مقدار محاسبه شده برای  $c[i, j]$  است. خروجی الگوریتم مذبور، ماتریس‌های  $c$  و  $b$  هستند. مقدار جواب بهینه برای طول  $LCS(X, Y)$  در  $c[n, m]$  ذخیره می‌شود. با شروع از  $b$  و تعقیب اشاره‌گرهای ذخیره شده در  $b$  می‌توان به جواب راه حل بهینه نیز رسید.

Algortim LCS\_Length( X, Y )

for  $i=1$  to  $n$ , do  $c[i,0]=0$

for  $j=0$  to  $m$  do  $c[0,j]=0$

for  $i=1$  to  $n$  do

    for  $j=1$  to  $m$  do

        if  $x[i]=y[j]$  then

        {

$c[i,j]=c[i-1,j-1]+1$

$b[i,j]=\nwarrow$

        }

        else

            if  $c[i-1,j] \geq c[i,j-1]$  then

            {

$c[i,j]=c[i-1,j]$

$b[i,j]=\uparrow$

            }

            else

            {

$c[i,j]=c[i,j-1]$

$b[i,j]=\leftarrow$

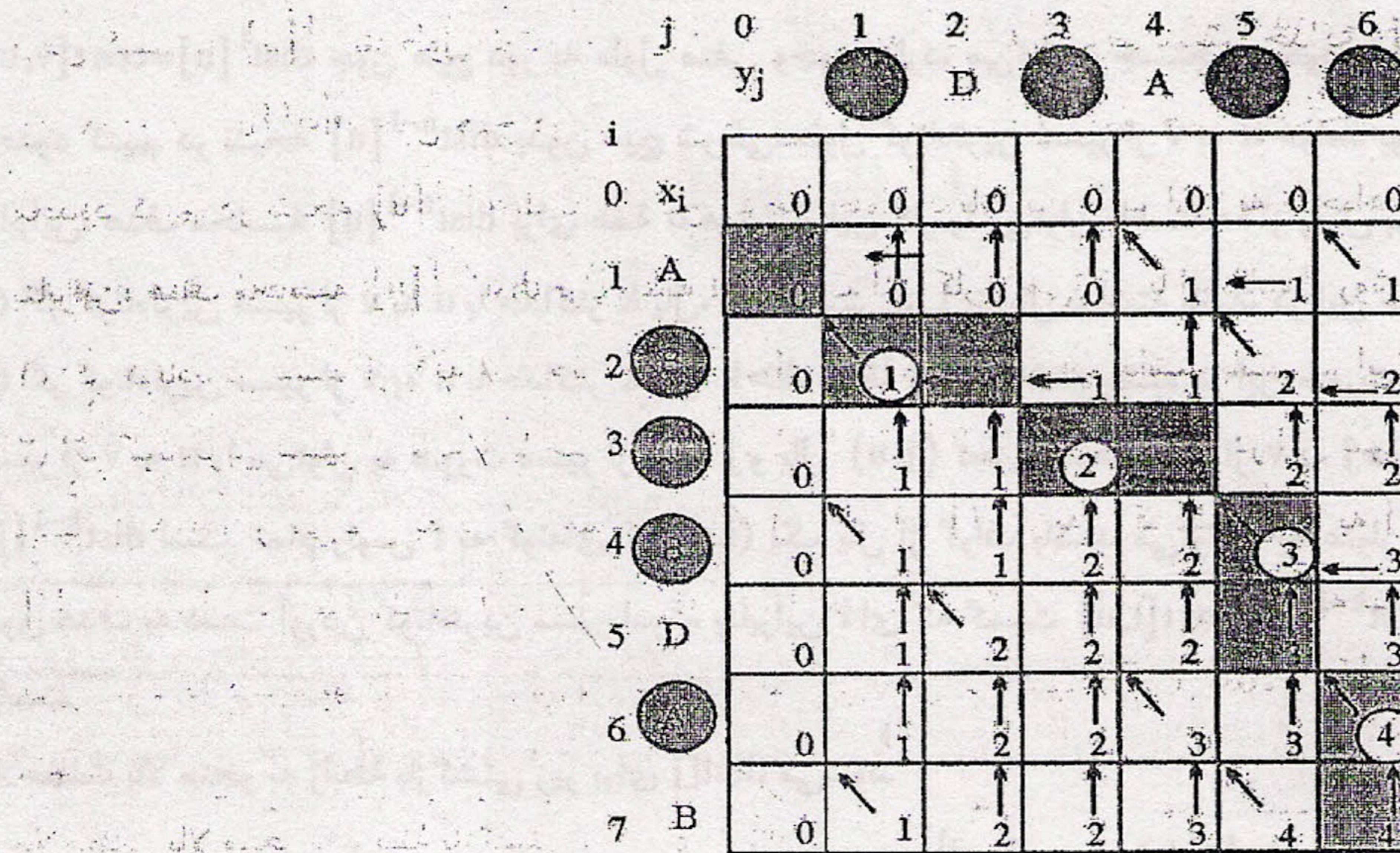
            }

Return c and b

پیچیدگی زمانی این الگوریتم  $\theta(nm)$  است.

مثال : مسأله LCS را برای دو رشته  $X=\langle A, B, C, B, D, A, B \rangle$  و  $Y=\langle B, D, C, A, B, A \rangle$  حل کنید.

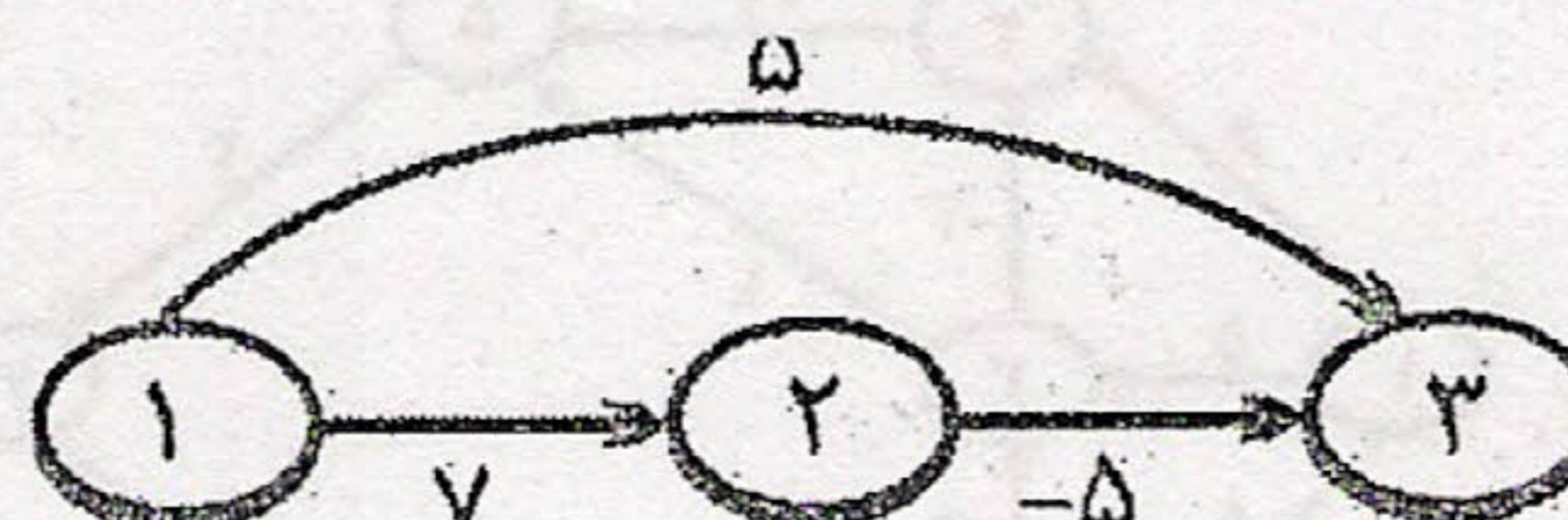
حل : خلاصه محاسبات در شکل زیر آمده است. در سطر ۱ و ستون ۰ این جدول، مقدار  $c[i,0]$  و نیز اشاره‌گر مربوط به  $b[i,0]$  آمده است. مقدار ۴ در  $c[7,6]$  برابر با طول LCS برای دو رشته X و Y این مثال است.  $LCS(X, Y) = \langle B, C, B, A \rangle$ . برای به دست آوردن  $LCS(X, Y)$  کافی است از  $b[7,6]$  شروع کرده و مسیر فلش‌ها را تعقیب کنیم. این مسیر در شکل به صورت هاشور زده نمایش داده شده است. هر فلش " $\nwarrow$ " در مسیر که در شکل با دایره کوچک مشخص شده است بیانگر حالت  $x[i]=y[j]$  بوده و معرف عنصری از LCS است.



### مسئله کوتاه‌ترین مسیرهای هم مبدا: وزن‌های تعمیم یافته (الگوریتم بلمن - فورد)

مسئله کوتاه‌ترین مسیرهای هم مبدا ارایه شده در فصل حریصانه را مجدداً در نظر بگیرید. این بار بعضی یال‌های گراف یا همه آن‌ها می‌توانند وزن منفی داشته باشند. الگوریتم دایکسترا که در فصل حریصانه برای این مسئله ارایه شد برای چنین گرافی الزاماً جواب درستی نخواهد داد. برای مشاهده این امر، گراف شکل زیر را در نظر بگیرید. فرض کنید که راس مبدا راس 1 است. اگر الگوریتم دیکسترا را بر روی این گراف اعمال کنیم  $\text{dist}[3] = 3$  برابر 5 خواهد شد، در صورتی که طول کوتاه‌ترین مسیر از راس 1 به 3 برابر 2 است (مسیر 1 و 2 و 3)

وقتی که به یال‌ها اجازه داشتن وزن منفی را می‌دهیم لازم است که گراف هیچ دوری با طول منفی نداشته باشد. این شرط برای اطمینان از این‌که کوتاه‌ترین مسیر از تعداد یال‌های متناهی تشکیل خواهد شد لازم است. برای مثال در گراف شکل زیر طول کوتاه‌ترین مسیر از راس 1 به راس 3 برابر  $-\infty$  است. طول مسیر 3, 1, 2, 1, 2, ..., 1, 2, 3 را می‌توان کمیت خیلی کوچکی قرار داد. دلیل این امر وجود دور 1 و 2 و 1 با طول مساوی 1 است.



گراف جهت‌دار با یک یال با وزن منفی

زمانی که هیچ دوری با طول منفی وجود نداشته باشد، حداقل تعداد یال‌های کوتاه‌ترین مسیر بین دو راس دلخواه برابر  $n-1$  خواهد بود که در آن،  $n$  تعداد رئوس گراف است. برای مشاهده این امر، مسیری که بیش از  $n-1$  یال داشته باشد، حداقل یک راس تکراری دارد و بنابراین شامل دور است. حذف دورهای این مسیر منجر به مسیر دیگری با همان رئوس مبدا و مقصد خواهد شد. این مسیر جدید دور ندارد و طول آن کوچک‌تر یا مساوی مسیر قبلی است زیرا طول دورهای حذف شده بزرگ‌تر یا مساوی صفر است. از این ملاحظه در مورد حداقل تعداد یال‌های موجود در کوتاه‌ترین مسیر بدون دور می‌توان برای بدست آوردن الگوریتمی برای محاسبه کوتاه‌ترین مسیر از راسی به کلیه رئوس دیگر استفاده کرد.

فرض کنید که طول کوتاهترین مسیر از راس  $v$  به راس  $u$  با شرط داشتن حداقل ۱ یال باشد. بدیهی است که با این تعریف  $\text{dist}^1[u] = \text{cost}[v, u]$  چون هیچ دور به طول منفی وجود ندارد، می‌توانیم جستجوی خود را فقط به مسیرهای با حداقل  $n-1$  یال محدود کنیم. در نتیجه  $\text{dist}^{n-1}[u]$  بدون هیچ شرطی، طول کوتاهترین مسیر از  $v$  به  $u$  خواهد بود.

بنابراین، هدف محاسبه  $\text{dist}^{n-1}[u]$  برای همه  $u$  ها است. این کار را می‌توان با استفاده از روش برنامه‌نویسی پویا انجام داد:

- ۱) اگر کوتاهترین مسیر از  $v$  به  $u$  با حداقل  $k$  یال،  $k > 1$  بیش از  $k-1$  یال نداشته باشد، در این صورت  $\text{dist}^k[u] = \text{dist}^{k-1}[u]$
- ۲) اگر کوتاهترین مسیر از  $v$  به  $u$  با حداقل  $k$  یال، دقیقاً  $k$  یال داشته باشد، در این صورت راسی مثل  $j$  وجود دارد به گونه‌ای که مسیر از  $v$  به  $u$  را می‌توان به صورت مسیر از  $v$  به  $j$  و  $j$  به  $u$  (ج) تجزیه کرد. مسیر از  $v$  به  $j$  دقیقاً دارای  $k-1$  یال بوده طول آن برابر  $\text{dist}^{k-1}[j]$  است. تمام رئوس  $i$  به گونه‌ای که  $(i, u)$  یک یال از گراف باشند می‌توانند به عنوان نامزدی برای  $j$  در نظر گرفته شوند. چون هدف به دست آوردن کوتاهترین مسیر است، بنابراین  $i$  ای که کمیت  $\text{dist}^{k-1}[i] + \text{cost}[i, u]$  را مینیمم کند، جواب صحیح برای  $j$  است.

ملاحظات بالا منجر به رابطه بازگشتی زیر برای  $\text{dist}^k$  می‌شود.

$$\text{dist}^k[u] = \min \left\{ \text{dist}^{k-1}[u], \min_i \left\{ \text{dist}^{k-1}[i] + \text{cost}[i, u] \right\} \right\}$$

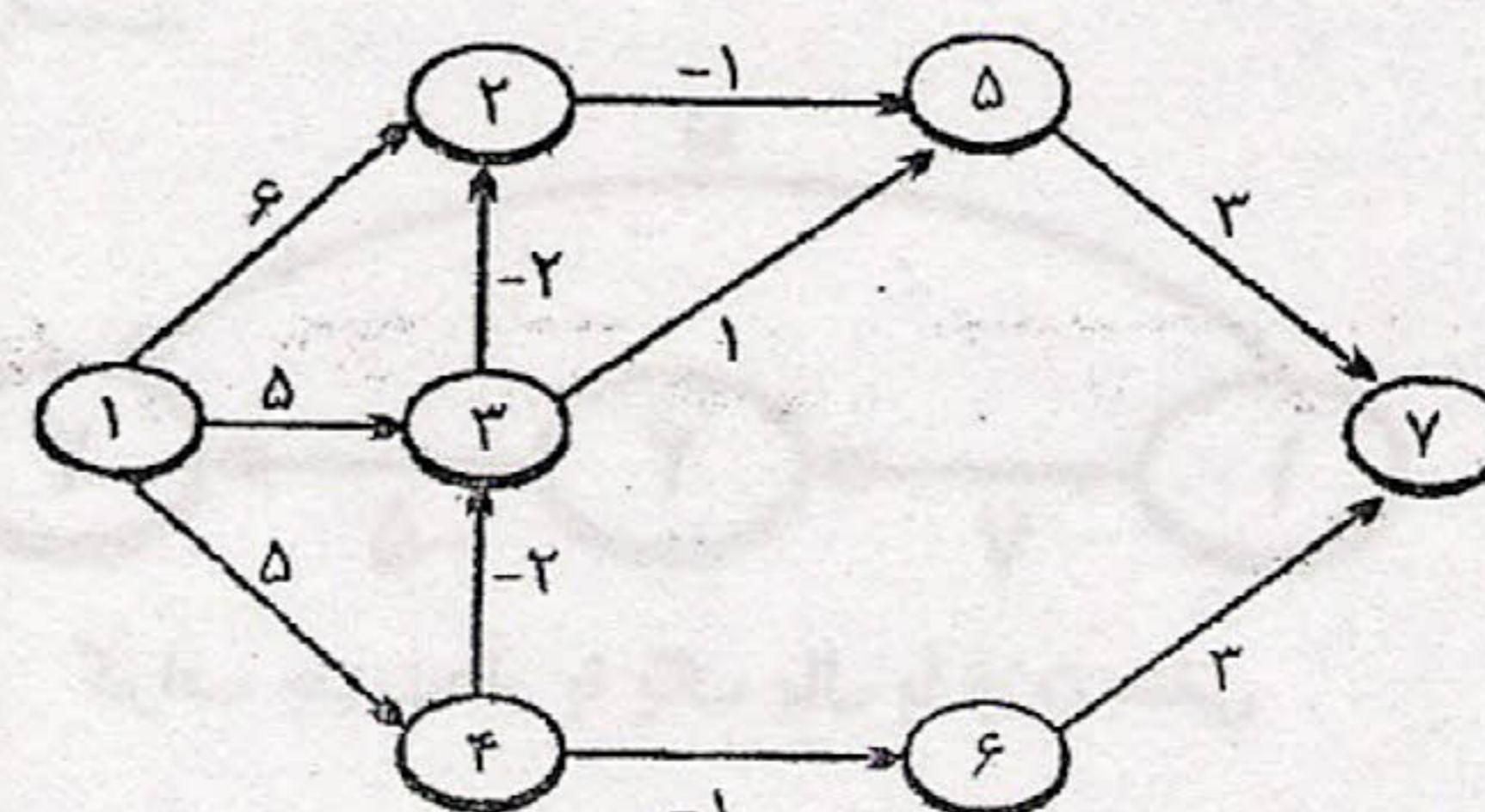
این رابطه بازگشتی را می‌توان برای محاسبه  $\text{dist}^k$  از روی  $\text{dist}^{k-1}$  برای  $k=1, 2, \dots, n-1$  به کار برد.

مثال: شکل زیر گرافی را با ۷ راس به همراه آرایه  $\text{dist}^k$  برای  $k=2, 3, \dots, n-1$  نشان می‌دهد.

این آرایه با استفاده از رابطه بازگشتی بالا محاسبه شده است. چون راس ۱ راس مبدأ است لذا  $\text{dist}^1[1] = 0$ . همچنین  $\text{dist}^1[2] = 5$ ،  $\text{dist}^1[3] = 5$  و  $\text{dist}^1[4] = 5$ ، زیرا از راس ۱ به این رئوس یالی در گراف وجود دارد.  $\text{dist}^1[5] = 6$  و  $\text{dist}^1[6] = 6$  و  $\text{dist}^1[7] = 3$  برابر  $\infty$  است، چون هیچ یالی از راس ۱ به این رئوس وجود ندارد.

$$\begin{aligned} \text{dist}^2[2] &= \min \left\{ \text{dist}^1[2], \min_i \left\{ \text{dist}^1[i] + \text{cost}[i, 2] \right\} \right\} \\ &= \min \{ 6, 0+6, 5-2, 5+\infty, \infty+\infty, \infty+\infty, \infty+\infty \} = 3 \end{aligned}$$

در عبارت بالا، جملات  $6-2, 0+6, 5-2, 0+6, 5-2, 0+6, 5-2, 0+6$  و  $\infty+\infty$  به ترتیب مربوط به انتخاب  $i$  مساوی با ۱، ۳، ۴، ۵، ۶ و ۷ هستند. بقیه داده‌ها به طریقه مشابه حساب می‌شوند.



k	$\text{dist}^k[1..7]$						
	1	2	3	4	5	6	7
1	0	6	5	5	$\infty$	$\infty$	$\infty$
2	0	3	3	5	5	4	$\infty$
3	0	1	3	5	2	4	7
4	0	1	3	5	0	4	5
5	0	1	3	5	0	4	3
6	0	1	3	5	0	4	3

اندیس  $k$  در  $\text{dist}^k[u]$  صرفاً برای درک بهتر مطلب بوده و در الگوریتم زیر که منسوب به بلمن فورد است از نوشتن آن اجتناب شده است. این الگوریتم، طول کوتاه‌ترین مسیرهای از راس  $v$  به بقیه رئوس را محاسبه می‌کند. مقدار نهایی  $\text{dist}[u]$  همان  $\text{dist}^{n-1}$  خواهد بود.

Algorithm Bellman\_Ford( $v, \text{cost}, \text{dist}, n$ )

```

1 for i=1 to n do dist[i]=cost[v,i]
2 for k=2 to n-1
3 for (هر راس u به گونه‌ای که v ≠ u و u دارای حداقل یک یال ورودی است) do
4 for {هر یال (i,u) در گراف} do
5   if dist[u]>dist[i]+cost[i,u] then
6     dist[u]=dist[i]+cost[i,u]

```

اگر برای نمایش گراف از ماتریس همسایگی استفاده شود در این صورت هر بار اجرای حلقه بیرونی 2 مستلزم زمان اجرای  $O(n^2)$  است. بنابراین پیچیدگی زمانی الگوریتم  $O(n^3)$  خواهد بود. در صورت استفاده از لیست همسایگی این مرتبه زمانی برابر  $O(en)$  خواهد بود.