

فصل ۳

روش‌های حریصانه

در مسایلی که به روش حریصانه حل می‌شوند، معمولاً "مجموعه‌ای از n ورودی مفروض است و از ما خواسته می‌شود تا زیر مجموعه‌ای از این n ورودی به دست بیاوریم که شرایط معینی داشته باشند. زیر مجموعه‌ای که این شرایط را داشته باشد، جواب امکان‌پذیر نامیده می‌شود. این جواب امکان‌پذیر باید یک تابع عینی را بهینه (بیشینه یا کمینه) کند. جواب امکان‌پذیری که این ویژگی را داشته باشد، جواب بهینه نامیده می‌شود. در بیشتر موارد، تعیین راه حل امکان‌پذیر، خیلی ساده‌تر از جواب بهینه است.

روش‌های حریصانه برخلاف روش‌های بازگشتی تقسیم و غلبه، روش‌های تکرار شونده هستند. در هر مرحله یکی از ورودی‌ها، بر حسب یک تابع گزینش انتخاب می‌شود. اگر اضافه کردن ورودی گزینش شده در یک مرحله به جواب بهینه جزئی حاصل، منجر به جواب امکان‌پذیر شود، آن‌گاه این ورودی به جواب بهینه جزئی اضافه می‌شود. مبنای تابع گزینش، خود بر پایه بعضی عمل بهینه‌سازی، استوار است.

Algorithm Greedy (A,n)

```
Solution =  $\emptyset$ 
for i=1 to n do
{
    x=Select (A)
    if feasible (Solotion , x)
        Solution = Union (Solution , x )
}
Return (Solution)
```

مساله کوله‌پشتی

فرض کنید یک کوله‌پشتی داریم که ظرفیت آن برابر M است و n جسم داریم که از 1 تا n شماره‌گذاری شده‌اند. وزن جسم i برابر w_i و ارزش جسم i هم برابر P_i فرض شده است. اگر کسر $x_i \leq 1$ از جسم i را در کوله‌پشتی قرار دهیم، آن‌گاه سود حاصل برابر $P_i x_i$ خواهد بود. هدف این است که کوله‌پشتی را از اجسامی پر کنیم تا سود کلی حاصل ماکزیمم شود. بدیهی است که مجموع اوزان اجسام انتخاب شده نباید از ظرفیت کوله‌پشتی تجاوز کند.

به عبارت دقیق‌تر، هدف بیشینه کردن کمیت:

$$\sum_{i=1}^n P_i x_i \quad (1)$$

$$\sum_{i=1}^n x_i w_i \leq M \quad (2)$$

با شرط $0 \leq x_i \leq 1$ است.

یک راه حل امکان‌پذیر، هر n تایی (x_1, x_2, \dots, x_n) است که در رابطه (2) صادق باشد و یک راه حل بهینه، راه حلی است که کمیت (1) را بیشینه کند.

مثال: نمونه مساله کوله‌پشتی زیر را در نظر بگیرید:

$$(P_1, P_2, P_3) = (25, 24, 15), (w_1, w_2, w_3) = (18, 15, 10), M = 20, n = 3$$

چند نمونه جواب امکان‌پذیر در زیر آمده است:

i	(x_1, x_2, x_3)	$\sum x_i w_i$	$\sum x_i p_i$
1	$\left(\frac{1}{2}, \frac{1}{3}, \frac{1}{4}\right)$	16.5	24.25
2	$\left(1, \frac{2}{5}, 0\right)$	20	28.2
3	$\left(0, \frac{2}{5}, 1\right)$	20	31
4	$\left(0, 1, \frac{1}{2}\right)$	20	31.5

در حالتی که $\sum_{i=1}^n w_i \leq M$ بدهی است که $1 \leq i \leq n, x_i = 1$ ، یک راه حل بهینه است. بنابراین برای جالب بودن مساله فرض خواهیم

کرد که $\sum_{i=1}^n w_i > M$. در این صورت مسلم است که تمامی x_i ها نمی‌توانند برابر 1 باشند. نکته مهم دیگری که باید در نظر داشت این

است که هر جواب بهینه‌ای باید کوله‌پشتی را دقیقاً پر کند.

جواب‌های امکان‌پذیر از 2 تا 4 در مثال قبل، که مجموع اوزان اجسام انتخاب شده برابر M است (کوله‌پشتی را پر کرده‌اند ولذا می‌توانند بهینه هم باشند!). به صورت‌های حریصانه زیر به دست آمده‌اند:

در سطر 2، اجسام به صورت نزولی ارزش آن‌ها در کوله‌پشتی قرار داده شده‌اند. اول جسم شماره 1 به طور کامل در کوله‌پشتی قرار داده شده است. اکنون ظرفیت باقیمانده کوله‌پشتی برابر 2 است. این دو واحد را باید از جسمی برداریم که ارزش آن در مقایسه با بقیه

اجسام باقیمانده بیشتر باشد. در مثال بالا، این جسم شماره 2 است که وزن آن برابر 15 است، بنابراین، تنها به اندازه $\frac{2}{15}$ از آن را می‌توان برای پر کردن کوله‌پشتی برداشت. با توجه به جواب شماره 3 که سود حاصل از آن، یعنی 31 بیشتر از سود حاصل از جواب 2 یعنی 28.2 است، در نظر گرفتن اجسام به صورت نزولی ارزش آن‌ها (حریص بودن صرف به ارزش بیشتر) منجر به جواب بهینه نخواهد شد. حال به صورت حریصانه دیگری اجسام را انتخاب می‌کنیم. این دفعه اجسام را به ترتیب صعودی وزنشان در نظر می‌گیریم، با این انتظار که اگر کوله‌پشتی با تعدد اجسام پر شود، جواب امکان‌پذیر حاصل، بهینه خواهد شد. جواب 3 مثال قبل، به این ترتیب حاصل شده است که با توجه به جواب شماره 4، مشاهده می‌شود که بهینه نیست.

جواب 4، تلفیقی از 2 روش حریصانه یاد شده است، یعنی برای به دست آوردن جواب 4 تابع گزینش، در نظر گرفتن اجسام به ترتیب نزولی ارزش بر واحد وزن آن‌ها بوده است، به عبارت دیگر اگر $\frac{P}{W}$ را تشکیل دهیم خواهیم داشت:

$$\frac{P}{W} = \left(\frac{P_1}{W_1}, \frac{P_2}{W_2}, \frac{P_3}{W_3} \right) = \left(\frac{25}{18}, \frac{24}{13}, \frac{15}{10} \right) = (1.38, 1.6, 1.5)$$

که در اینجا اولویت اول، انتخاب جسم شماره 2 است که وزن آن برابر 15 است و به طور کامل انتخاب می‌شود. اولویت بعدی با جسم شماره 3 است که چون ظرفیت باقیمانده کوله‌پشتی 5 واحد است، لذا آن را نمی‌توان به طور کامل انتخاب کرد، در نتیجه به اندازه $\frac{5}{10} = \frac{1}{2}$ از آن انتخاب شده است.

در حالت عمومی اثبات می‌شود که اگر اجسام را به ترتیب نزولی ارزش بر واحد وزن آن‌ها انتخاب بکنیم جواب حاصل بهینه است. بنابراین در مسأله کوله‌پشتی داده شده، روش حریصانه برای به دست آوردن جواب بهینه را می‌توان به صورت زیر ارایه داد. برای الگوریتم زیر فرض شده‌اند که اجسام به صورت نزولی ارزش بر واحد وزنشان مرتب شده‌اند، یعنی:

$$\frac{P_1}{W_1} \geq \frac{P_2}{W_2} \geq \dots \geq \frac{P_n}{W_n}$$

Algorithm Greedy_knapsack (p, w, M, X, n, Profit)

X=0 , Profit=0 / بردار جواب و سود حاصل /

Cu=M / ظرفیت باقیمانده کوله‌پشتی /

for i=1 to n do

{

if w[i] > Cu then exit

x [i] =1

Cu= Cu - w [i]

profit = profit + p [i]

}

X [i] =Cu / w [i]

Profit = Profit + P [i] *X [i]

مرتبه بزرگی الگوریتم مذبور، بدون در نظر گرفتن مرتب‌سازی اولیه، برابر $(n)^{\theta}$ است. بنابراین اگر درجه بزرگی عمل مرتب‌سازی قبل از الگوریتم را نیز در نظر بگیریم، مسأله کوله‌پشتی حریصانه را می‌توان با مرتبه بزرگی $(n \log n)^{\theta}$ حل کرد.

قضیه: اگر $\frac{P_1}{W_1} \geq \frac{P_2}{W_2} \geq \dots \geq \frac{P_n}{W_n}$ آن‌گاه، الگوریتم مذبور یک جواب بهینه برای نمونه مسأله کوله‌پشتی داده شده به دست می‌دهد.

مثال : بردار جواب و نیز ارزش حاصل از مساله کوله‌پشتی حریصانه زیر را به دست آورید.

$$n = 6$$

$$p = (p_1, p_2, p_3, p_4, p_5, p_6) = (10, 7, 12, 13, 6, 20)$$

$$w = (w_1, w_2, w_3, w_4, w_5, w_6) = (2, 1, 3, 2, 12, 8)$$

$$M = 14$$

حل:

$$\frac{p}{w} = (5, 7, 4, 6.5, 0.5, 2.5)$$

I	w _i	p _i	$\frac{p_i}{w_i}$	X _i	Cu	Profit
2	1	7	7	1	13	7
4	2	13	6.5	1	11	20
1	2	10	5	1	9	30
3	3	12	4	1	6	42
6	8	20	2.5	0.75	0	42+0.75+20=57
5	12	6	0.5	0		

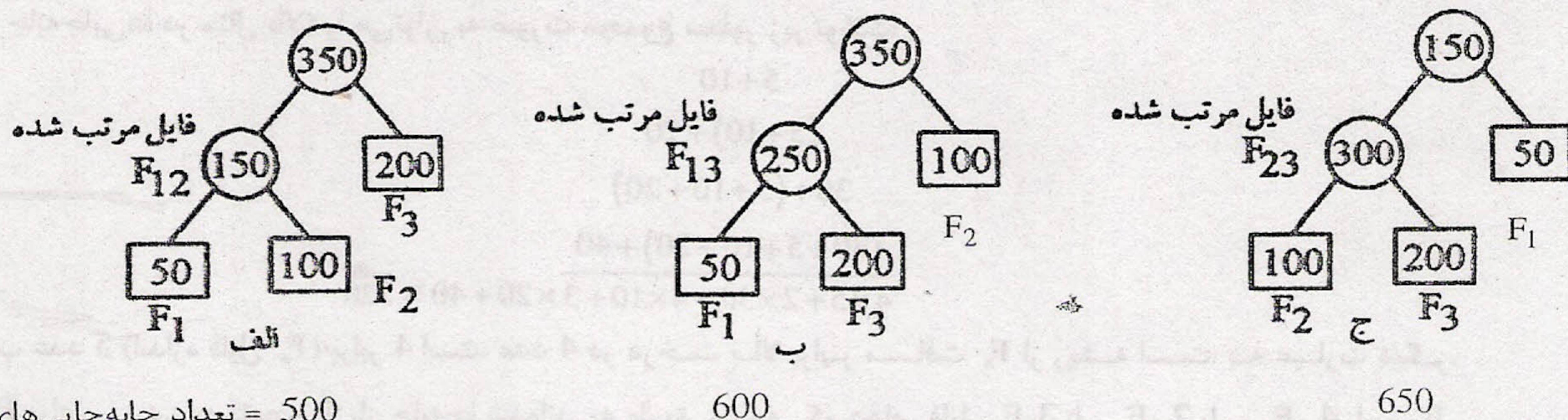
بنابراین بردار جواب $x = (1, 1, 1, 1, 0, 0.75)$ و ارزش مکزیمم حاصل برابر 57 است.

ادغام دودویی و بهینه فایل‌ها (آرایه‌ها)

قبل از دیدیم که اگر دو لیست مرتب شده با اندازه‌های n ، m داشته باشیم، می‌توان آن‌ها را در مدت زمان $(n+m)\theta$ ادغام کرده و لیست مرتب شده واحدی به دست آورد. اما زمانی که بیش از دو فایل مرتب شده داشته باشیم و بخواهیم آن‌ها را برای به دست آوردن یک فایل مرتب شده واحد ادغام کنیم، به صورت‌های مختلف می‌توان این کار را انجام داد. برای مثال سه فایل مرتب شده F_1 , F_2 , F_3 را در نظر بگیرید. می‌توان نخست F_1 را با F_2 برای به دست آوردن یک فایل مرتب شده واحد مثل F_{12} ادغام کرده و سپس F_{12} را با F_3 ادغام کرد. می‌توانیم نخست F_1 را با فایل F_3 ادغام بکنیم تا فایل مرتب شده‌ای مثل F_{13} به دست بیاید و سپس آن را با F_2 ادغام بکنیم. بالاخره می‌توانیم نخست دو فایل مرتب شده F_2 , F_3 را با هم ادغام و سپس نتیجه را برای به دست آوردن فایل مرتب شده واحد، با فایل F_1 ادغام بکنیم.

در حالت عمومی n فایل مرتب شده را می‌توان به صورت‌های مختلف دو یه دو با هم ادغام کرد و ادغام‌های مختلف منجر به زمان‌های اجرای متفاوتی خواهد شد. بنابراین در ادغام دودویی و بهینه فایل‌ها، هدف به دست آوردن یک راه حل بهینه برای ادغام دو بهدوی n فایل مرتب شده است.

مثال: اگر سه فایل مرتب شده F_1 , F_2 , F_3 به ترتیب با اندازه‌های 50, 100, 200 را در نظر بگیریم، به سه صورت زیر می‌توان آن‌ها را با هم ادغام کرد:



500 = تعداد جایه‌جایی‌های انجام شده

600

650

در حالت (الف)، اول فایل‌های F_1 , F_2 , F_3 ادغام شده‌اند. این ادغام 150 جایه‌جایی نیاز خواهد داشت. سپس نتیجه، یعنی F_{12} با فایل F_3 ادغام می‌شود. این ادغام هم به 350 جایه‌جایی دیگر نیاز دارد. بنابراین مجموع تعداد جایه‌جایی‌های انجام شده برابر 500 خواهد بود. تعداد کل این جایه‌جایی‌ها برای حالت (ب) و (ج) به ترتیب برابر 600, 650 است.

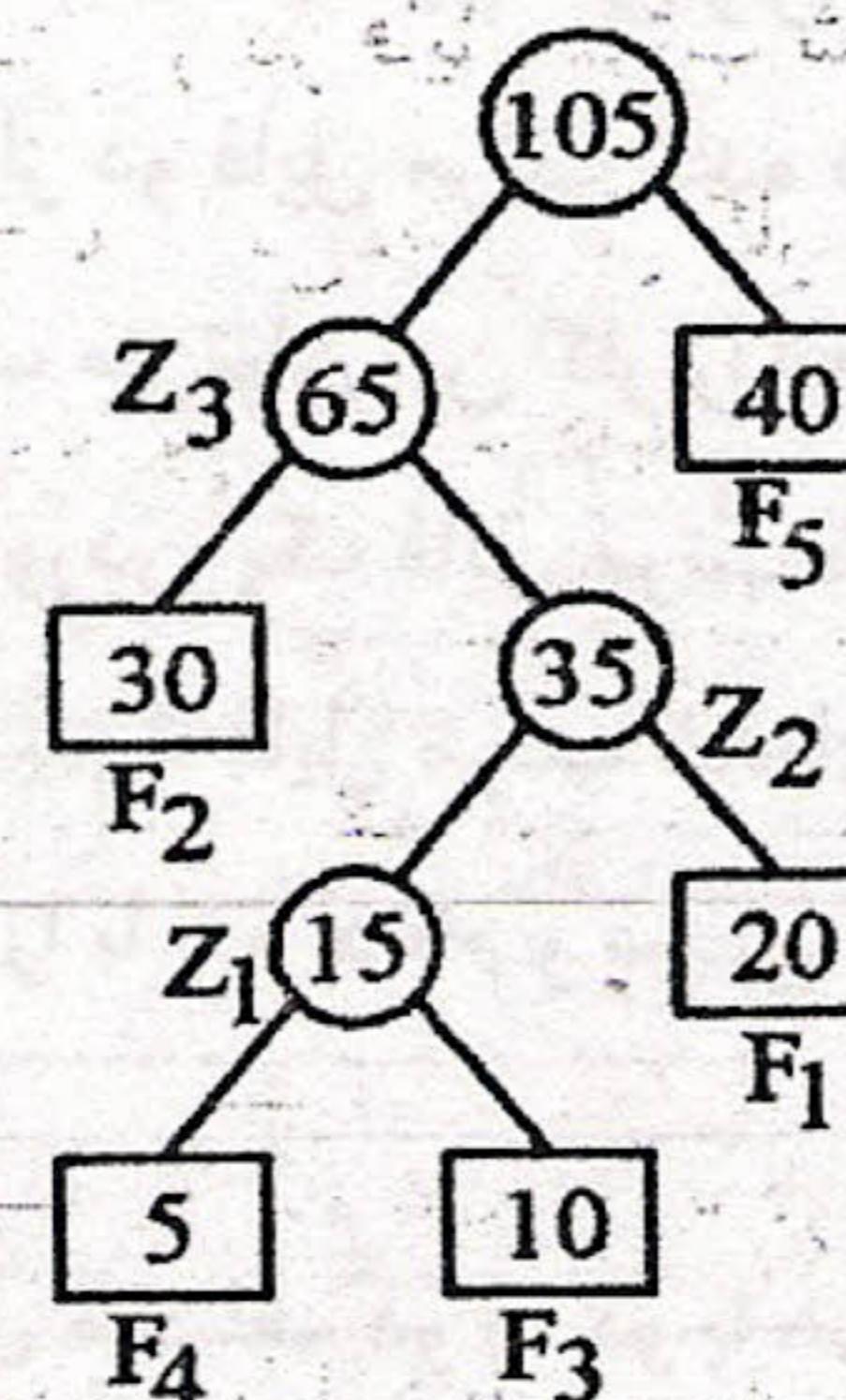
با توجه به این مثال، به فرمول در آوردن یک روش حریصانه برای ادغام بهینه n فایل مرتب شده ساده است.

چون ادغام یک فایل مرتب شده n رکوردی با یک فایل مرتب شده m رکوردی، نیاز به جایه‌جایی $n+m$ رکورد دارد، راه حل بدیهی برای انتخاب تابع گزینش به این صورت است که در هر مرحله دو فایل باکمترین اندازه‌ها را با هم ادغام می‌کنیم. برای مثال، اگر 5 فایل مرتب شده F_1 , F_2 , F_3 , F_4 , F_5 به ترتیب با اندازه‌های 20, 5, 10, 30, 40 داشته باشیم، الگوریتم حریصانه ما، الگوی ادغام زیر را به دست خواهد داد:

اول F_1 را با F_2 ادغام می‌کنیم. فرض نتیجه Z_1 است. اندازه آن 15 و تعداد جایه‌جایی‌های به کار رفته هم 15 است.
 دوم Z_1 را با F_1 ادغام می‌کنیم. فرض نتیجه Z_2 است. اندازه آن 35 و تعداد جایه‌جایی‌های به کار رفته هم 35 است.
 سوم Z_2 را با F_2 ادغام می‌کند. فرض نتیجه Z_3 است. اندازه آن 65 و تعداد جایه‌جایی‌های به کار رفته هم 65 است.
 چهارم Z_3 را با F_3 ادغام می‌کنیم. اندازه آن 105 و تعداد جایه‌جایی‌های به کار رفته هم برابر 105 است.

بنابراین تعداد کل جابه‌جایی‌های به کار رفته برابر 220 است که می‌توان تحقیق کرد، برای نمونه مسأله داده شده، این تعداد جابه‌جایی‌ها مینیمم است.

الگوی ادغام n فایل مرتب شده را می‌توان به وسیله درخت دودویی ادغام به تصویر کشید. شکل زیر درخت دودویی ادغام (بهینه) برای 5 فایل مثال قبل است.



درخت دودویی ادغام بهینه برای 5 فایل مثال قبل

گره‌های خارجی این درخت به صورت مربع رسم شده‌اند و معرف فایل‌ها هستند. گره‌های داخلی به صورت دایره رسم شده‌اند. هر گره داخلی دقیقاً 2 فرزند دارد و معرف فایلی است که از ادغام دو فایل نشان داده شده توسط فرزندانش حاصل شده است. عدد داخل هر گره، اندازه فایل نمایش داده شده توسط آن گره است.

عدد 220 (تعداد جابه‌جایی‌ها در مثال بالا) را می‌توان به صورت مجموع سطور زیر نوشت:

$$\begin{aligned} & 5 + 10 \\ & (5 + 10) + 20 \\ & 30 + (5 + 10 + 20) \\ & \frac{(30 + 5 + 10 + 20) + 40}{4 \times 5 + 2 \times 30 + 4 \times 10 + 3 \times 20 + 40} = 220 \end{aligned}$$

در این جمع ضریب عدد 5 (اندازه فایل F_4) برابر 4 است. عدد 4 در درخت بالا برابر مسافت F_4 از ریشه است. به عبارت دیگر، رکوردهای فایل F_4 برای رسیدن به نتیجه، 4 بار جابه‌جا شده‌اند. به طریق مشابه رکوردهای فایل F_1, F_2, F_3 ، 2 بار، 3 بار و F_5 ، 1 بار جابه‌جا شده‌اند.

در حالت عمومی، اگر d_i برابر مسافت فایل F_i در درخت دودویی ادغام باشد، در این صورت تعداد کل رکوردهای جابه‌جا شده از فرمول زیر حاصل می‌شود:

$$\sum_{i=1}^n d_i q_i$$

که در آن q_i اندازه (تعداد رکوردهای) فایل F_i است. این کمیت به WEPL (Weighted External Path Length) معروف است.

بنابراین در ادغام دودویی و بهینه فایل‌ها، هدف به دست آوردن یک درخت ادغام است که برای آن کمیت WEPL مینیمم باشد. الگوریتم زیر، روش حریصانه شرح داده شده را برای به دست آوردن یک درخت ادغام دودویی بهینه برای n فایل به دست می‌دهد. ورودی الگوریتم یک لیست L از n درخت است. هر گره درخت، سه میدان Weight, Rchild, Lchild دارد. در ابتدا هر درخت در L، دارای یک گره است. این گره، یک برگ بوده، Rchild, Lchild آن برابر صفر و Weight آن هم برابر با اندازه یکی از n فایل‌ها است.

در حین اجرای الگوریتم، برای هر درخت در لیست L که ریشه آن T است، $\text{Weight}(T)$ برابر اندازه فایل ادغام شده‌ای است که گره T معرف آن است ($\text{Weight}(T)$ برابر مجموع وزن گره‌های خارجی واقع در درخت به ریشه T است). الگوریتم Tree از سه زیر الگوریتم Insert(L, T)، Least(T)، Getnode(T) استفاده می‌کند. یک گره جدید برای تشکیل درخت موردنظر ایجاد می‌کند.

درخت با کمترین Weight را در L پیدا کرده و سپس T را از L حذف می‌کند. Insert(L, T)، درخت به ریشه T را در L درج می‌کند.

Algorithm Tree(L, n)

for $i=1$ to $n-1$ do

{

 Getnode(T)

 Lchild(T) = Least(L)

 Rchild(T) = Least(L)

 Weight(T) = Weight(Lchild(T)) + Weight(Rchild(T))

 Insert(L, T)

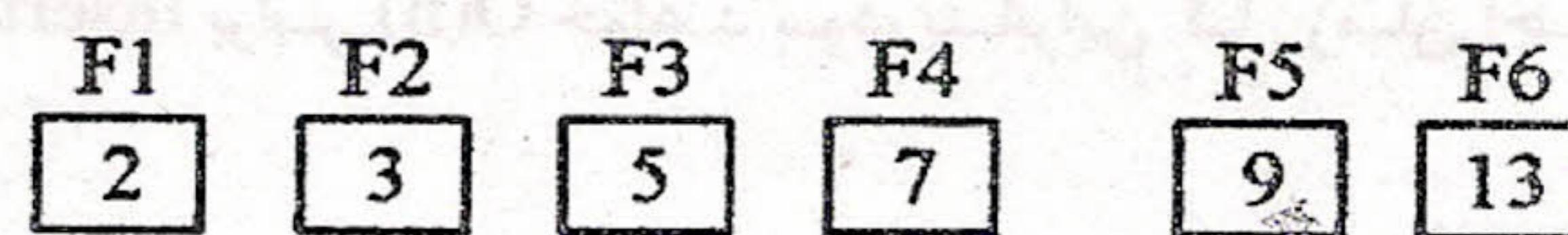
}

Return(L)

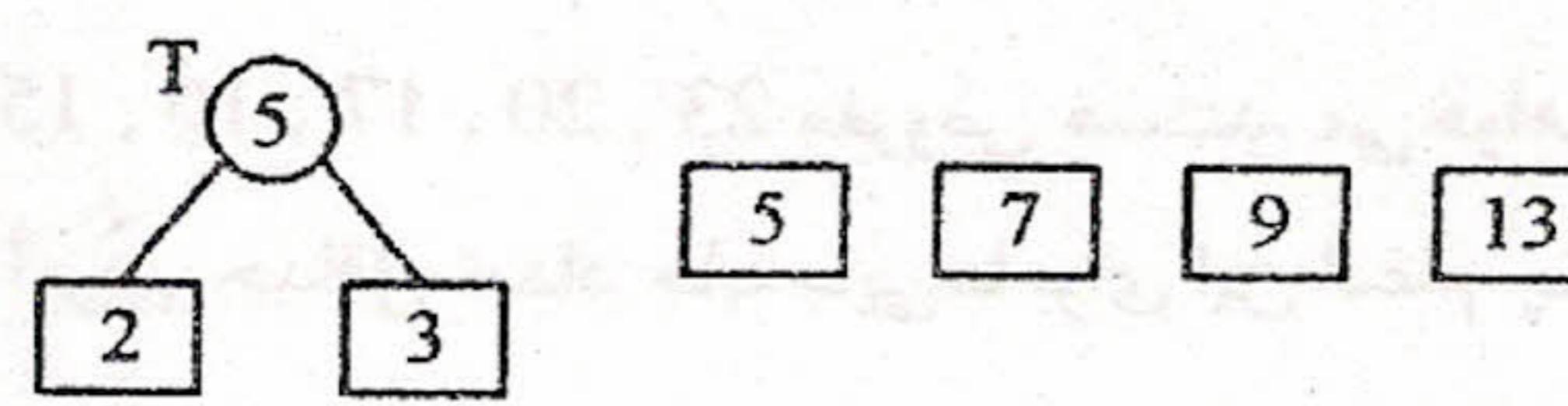
قضیه: اگر L در ابتداء دارای $n \geq 1$ گره با مقادیر Weight برابر $q_1, q_2, q_3, \dots, q_n$ باشد، آن‌گاه الگوریتم Tree، یک درخت ادغام دودویی بهینه برای n فایل با اندازه‌های فوق الذکر ایجاد می‌کند.

مثال: مراحل مختلف اجرای الگوریتم Tree را بر روی فایل‌های $F_1, F_2, F_3, F_4, F_5, F_6$ به ترتیب با اندازه‌های 2, 3, 5, 7, 9, 13 بنویسید.

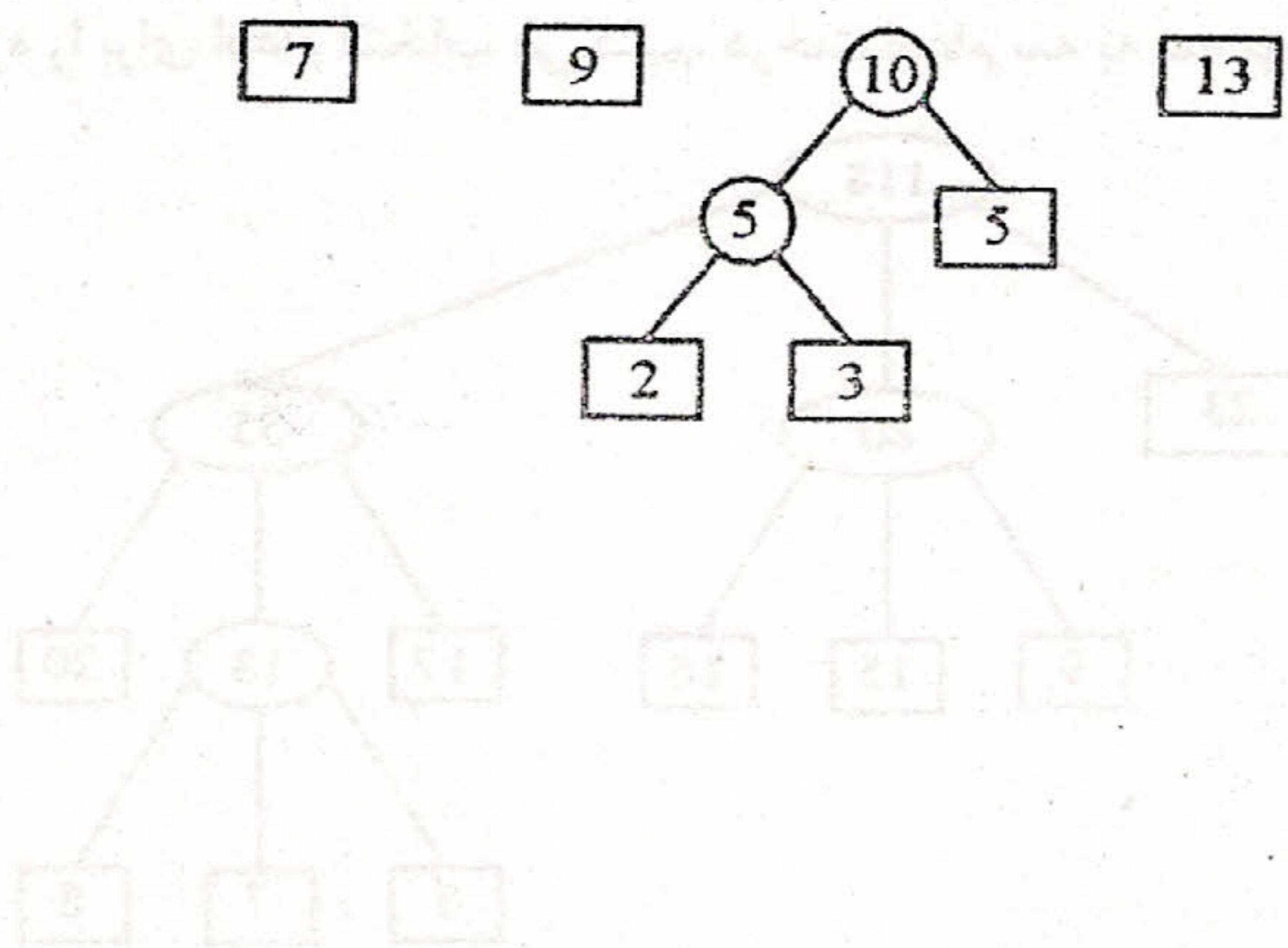
مرحله ۰ L

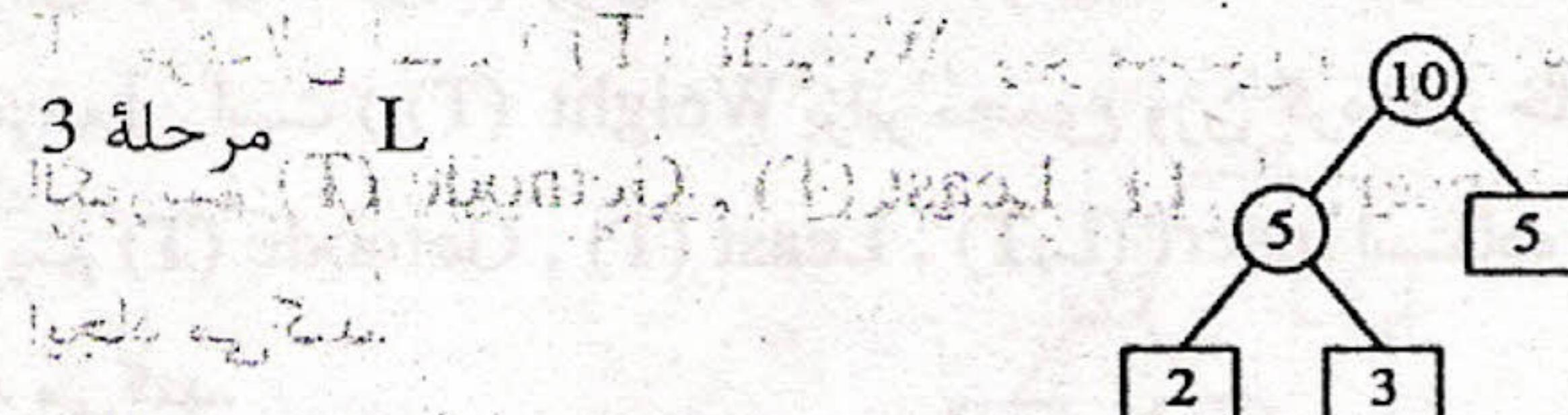


مرحله ۱ L

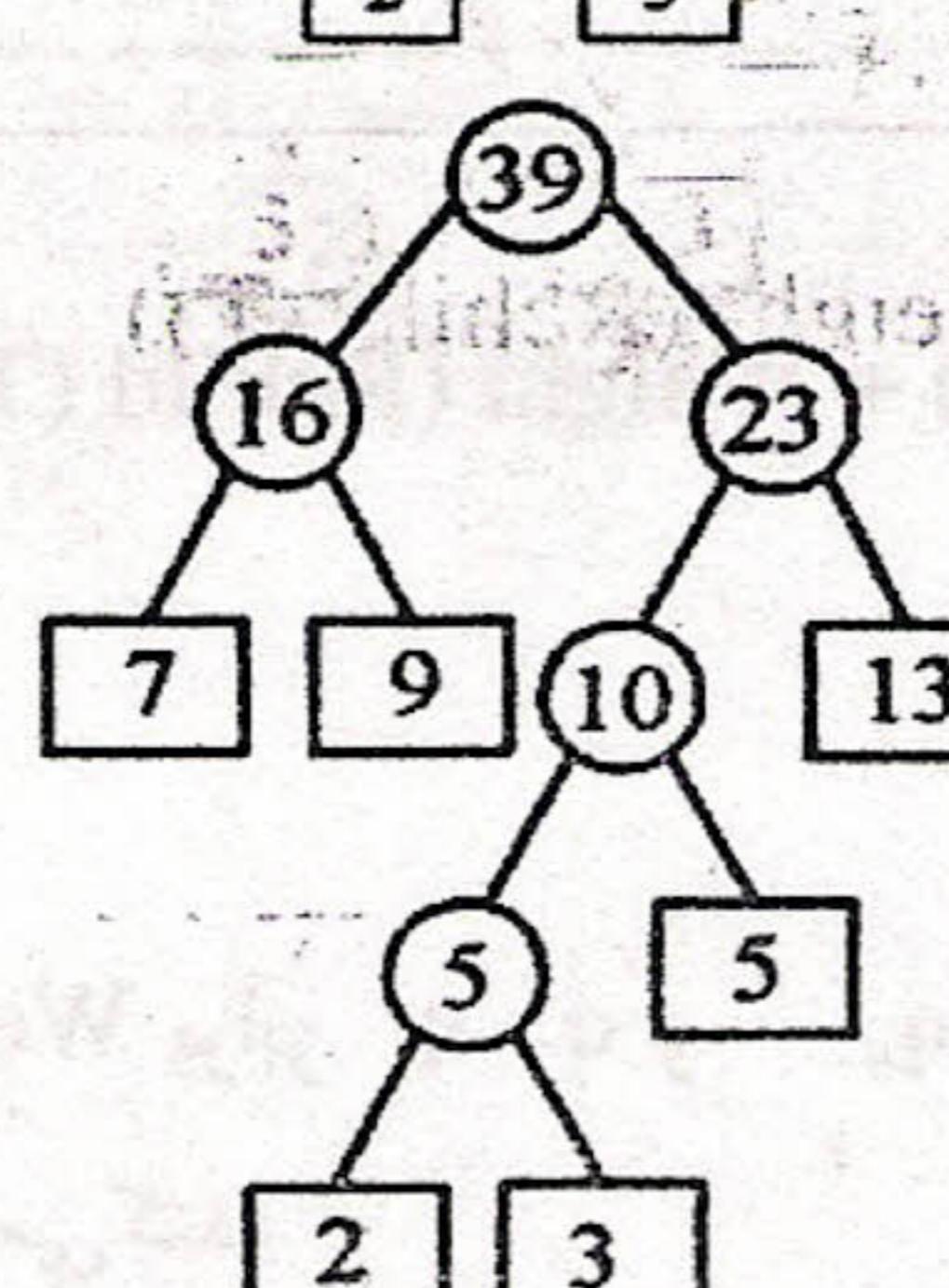


مرحله ۲ L





مرحله 5 L



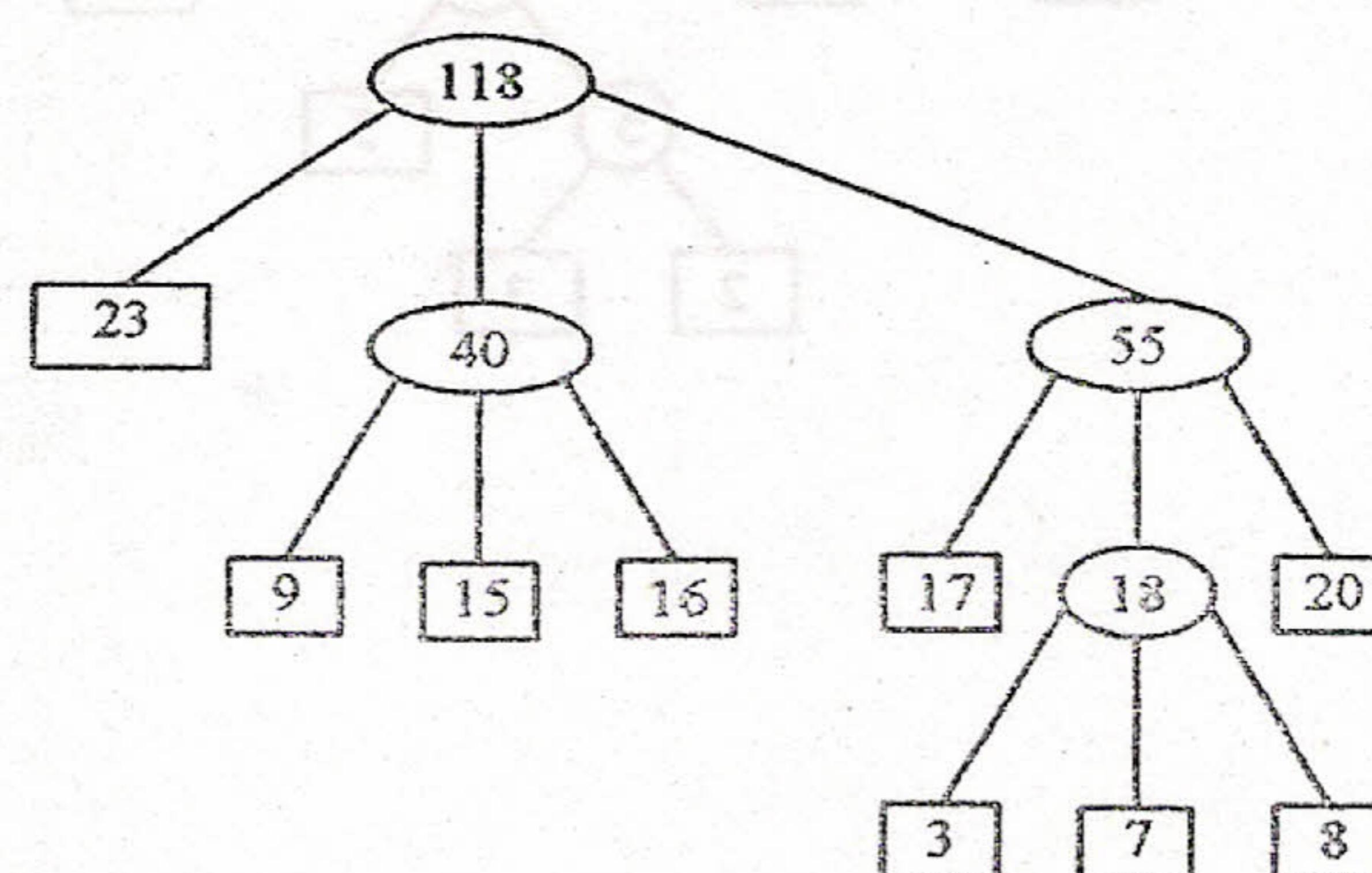
تحلیل الگوریتم Tree

حلقه اصلی $n-1$ بار اجرا می شود. اگر فرض کنیم که لیست L به ترتیب صعودی Weight ها مرتب شده است، در این صورت مرتبه بزرگی (L) برابر $O(1)$ و مرتبه بزرگی Insert (L, T) خواهد بود، بنابراین کل زمان اجرای Tree از مرتبه $O(n^2)$ خواهد بود.

مثال : 9 فایل مرتب شده به اندازه های 3 , 7 , 8 , 9 , 15 , 16 , 17 , 20 , 23 مفروض هستند. می خواهیم این 9 فایل را سه به سه با هم ادغام کرده و فایل مرتب شده واحدی به دست آوریم. حداقل تعداد جایه جایی ها برای این ادغام چقدر است؟

حل :

الگوریتم Tree برای ادغام دودویی را می توان برای ادغام سه به سه نیز تعمیم داد. در اینجا، در هر مرحله به جای انتخاب دو فایل با کمترین اندازه، 3 فایل با کمترین اندازه را برای ادغام انتخاب می کنیم. درخت ادغام سه به سه بهینه به صورت زیر خواهد بود:



و برای این درخت

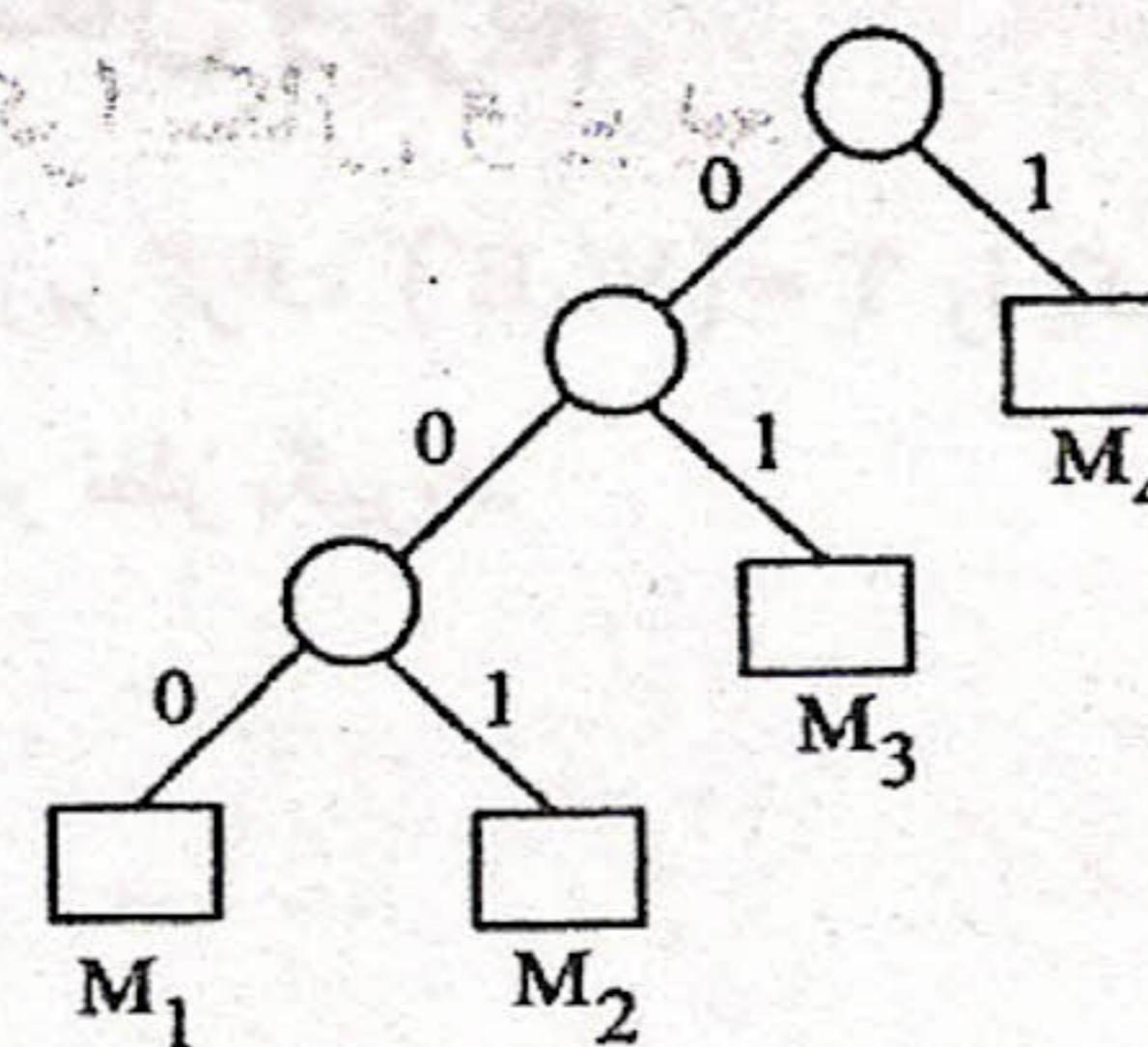
$$WEPL = 23 + (9+15+16) \times 2 + (17+20) \times 2 + (3+7+8) \times 3$$

$$23 + 80 + 74 + 54 = 231$$

کاربرد : کد هافمن (Huffman)

کاربرد دیگر درخت دودویی با WEPL مینیمم، به دست آوردن یک مجموعه بهینه از رموز برای n پیغام M_1, M_2, \dots, M_n است. هر رمز، یک رشته دودویی از ۰ و ۱ هاست که برای انتقال پیغام مورد نظر به کار می‌رود. در طرف گیرنده، رمز ارسال شده باید توسط یک درخت کشف رمز، رمزگشایی شود.

درخت کشف رمز، یک درخت دودویی است که برگ‌های آن معرف پیغام‌ها بوده و یال‌های آن با عناصر ۰ و ۱ بر چسب خورده‌اند. برای مثال، برای چهار پیغام M_1, M_2, M_3, M_4 یک درخت کشف رمز می‌تواند به شکل زیر باشد:



درخت کشف رمز

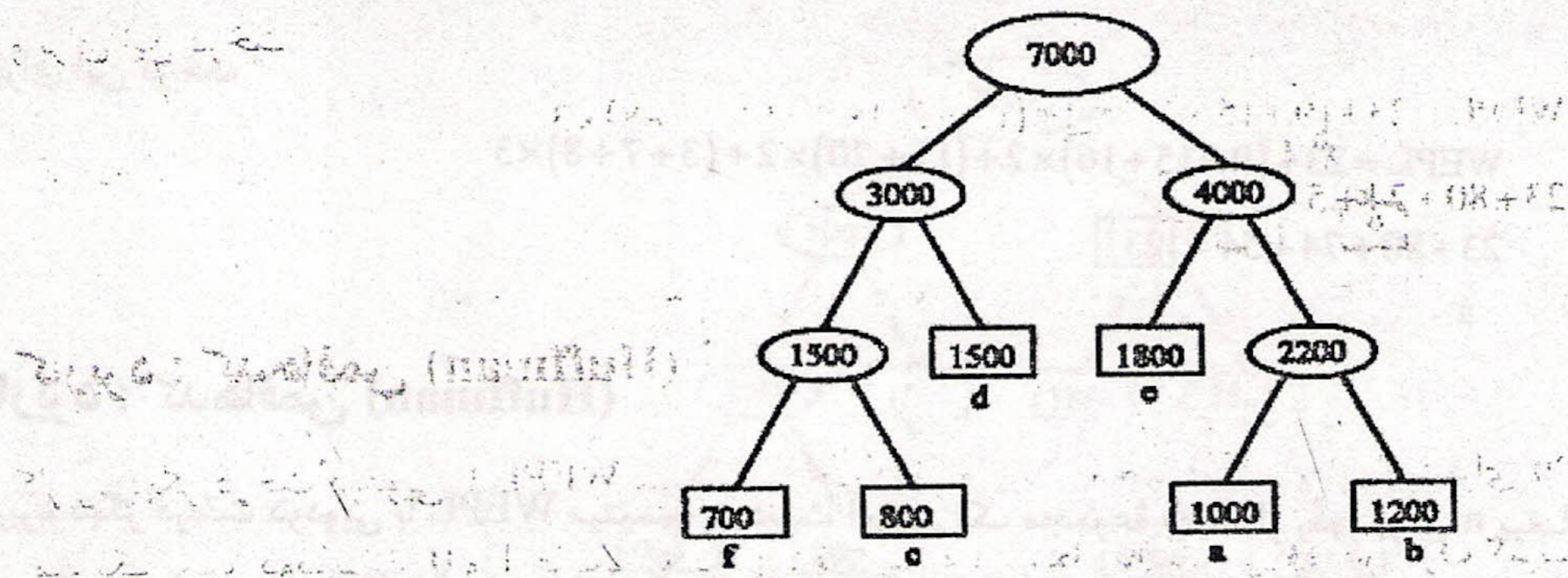
بیت موجود در کلمه رمز برای یک پیغام، مشخص کننده شاخه‌ای خواهد بود که طرف گیرنده باید در درخت کشف رمز برای رسیدن به گره خارجی صحیح (پیغام) انتخاب کند. برای مثال، اگر در درخت کشف رمز، ۰ به معنی شاخه چپ و ۱ به معنی شاخه راست باشد، آن‌گاه در درخت کشف رمز بالا $000, 001, 01, 1$ ، به ترتیب رموز پیغام‌های M_1 تا M_4 خواهد بود. این رموزها کد هافمن نامیده می‌شوند.

مسلمان" هزینه ارسال و کشف یک رمز متناسب با تعداد بیت‌های به کار رفته در آن است. در درخت کشف رمز این تعداد برابر با مسافت گره خارجی مربوطه از ریشه است. اگر q_i فرکانس نسبی ارسال پیغام M_i باشد، در این صورت میانگین هزینه ارسال یا کشف رمز متناسب با $\sum_{i=1}^n q_i d_i$ خواهد بود که در آن d_i مسافت پیغام M_i از ریشه است. این میانگین وقتی مینیمم است که رموزها به گونه‌ای انتخاب شوند که منجر به درخت کشف رمز با WEPL مینیمم شوند.

مثال: متنی شامل 7000 حرف از حروف a, b, c, d, e, f با دفعات تکرار $f=700, e=1800, d=1500, c=800, b=1200, a=1000$ موجود است. اگر کدی بهینه برای هر حرف بالا انتخاب بکنیم، تعداد کل بیت‌های لازم برای تبدیل متن مذکور به مجموعه‌ای از بیت‌ها چقدر است؟

حل :

اگر این 7 حرف را مشابه هفت پیغام و دفعات تکرار آن‌ها را مشابه فرکانس نسبی مورد اشاره بالا تعبیر کنیم، درخت دودویی با WEPL مینیمم به صورت زیر خواهد بود:

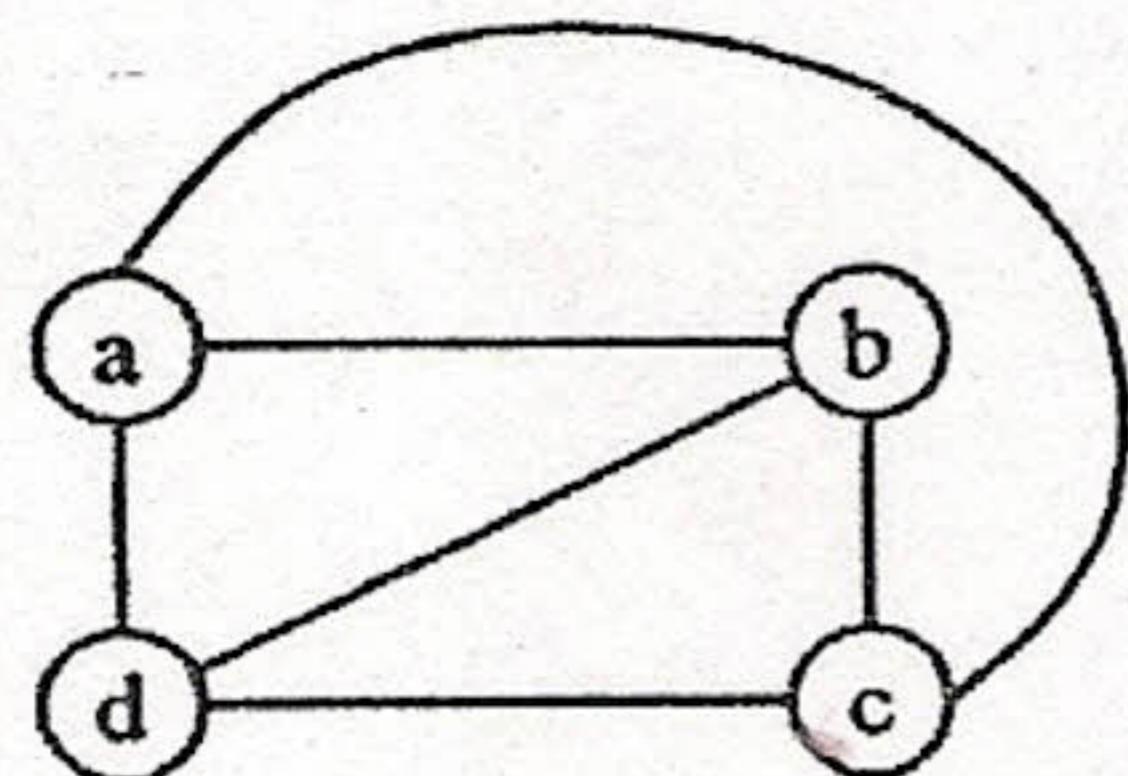


$$WEPL = 700 \times 3 + 800 \times 3 + 1500 \times 2 + 1800 \times 2 + 1000 \times 3 + 1200 \times 3 \Rightarrow$$

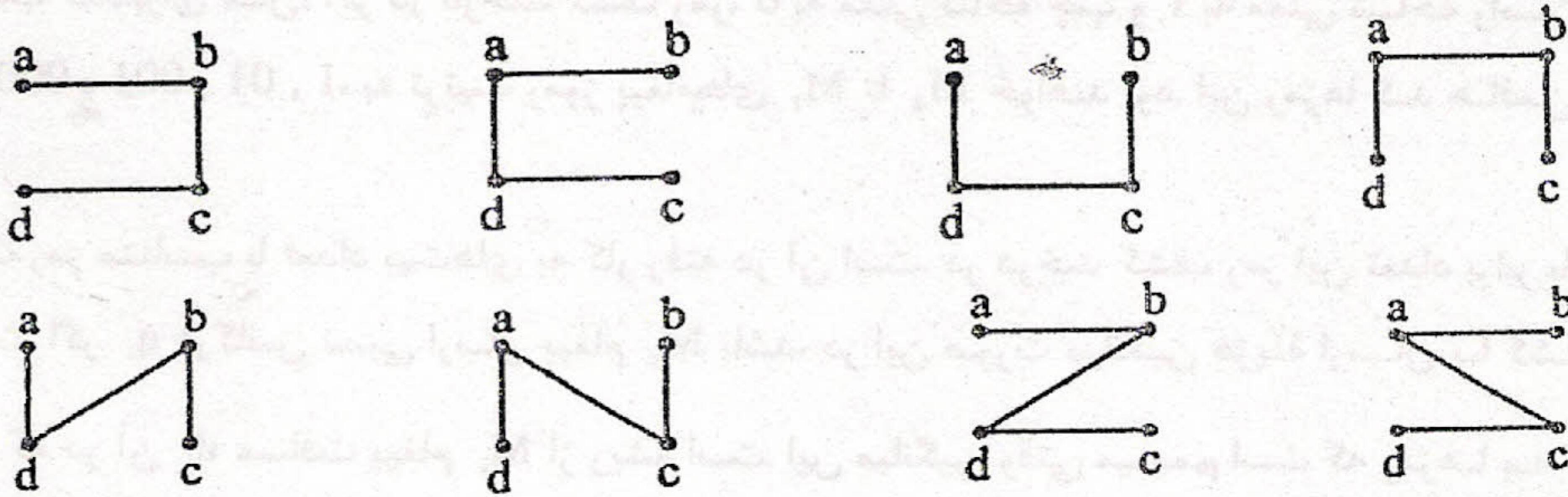
$$WEPL = 17700$$

درخت‌های پوشای مینیمم و الگوریتم‌های کراسکال و پریم

فرض کنید $G = (V, E)$ یک گراف همبند و بدون جهت باشد. زیر گراف $T = (V', E')$ را یک درخت پوشای برای G می‌گوییم، هرگاه $V' = V$ و T یک درخت باشد. برای مثال، گراف کامل K_4 را در نظر بگیرید.



درخت‌های پوشای زیر چند نمونه از 16 درخت پوشای K_4 هستند:



تمرین: بقیه 8 درخت پوشای K_4 را رسم کنید.

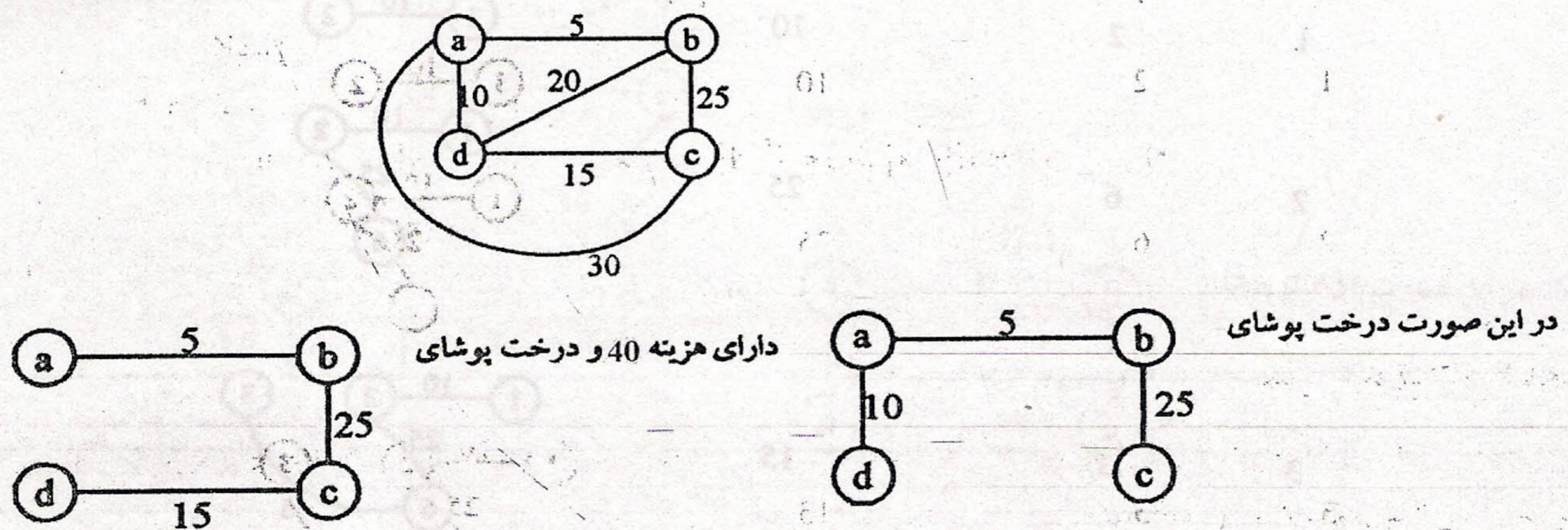
قضیه 1: گراف کامل K_n دارای n^{n-2} درخت پوشای است.

قضیه 2: اگر گراف همبند $G = (V, E)$, n راس داشته باشد، هر درخت پوشای برای آن دارای $n-1$ یال خواهد بود.

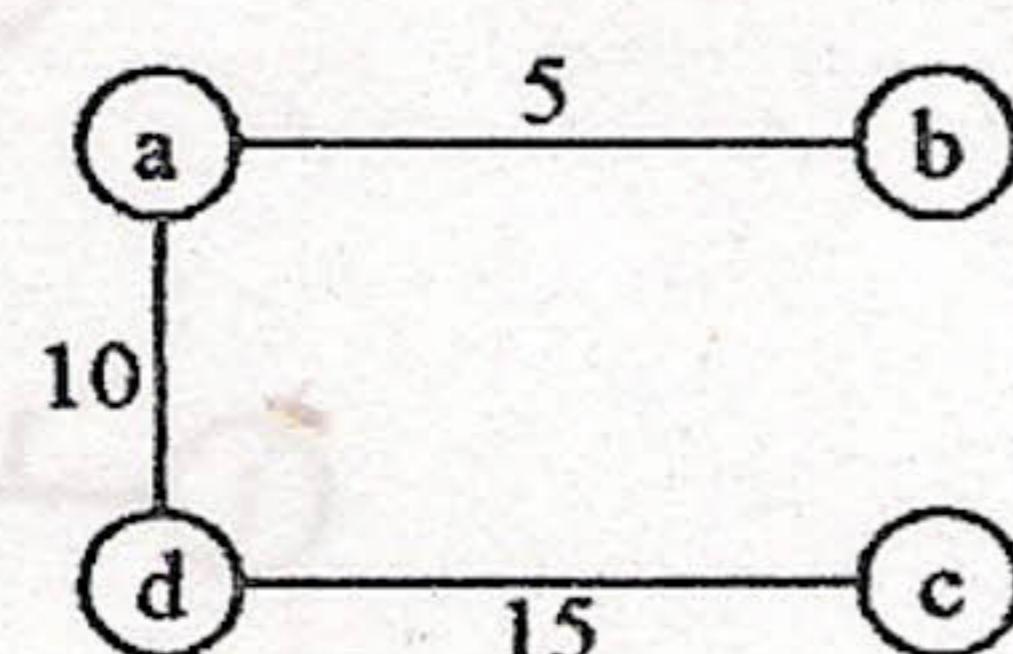
اگر گراف $G = (V, E)$ را به منزله یک شبکه ارتباطی در نظر بگیریم، در این صورت یک درخت پوشای برای این گراف، معرف حداقل شبکه ارتباطی خواهد بود. مثلاً اگر n راس گراف، معرف n شهر باشد، در این صورت حداقل تعداد راههایی که می‌تواند این n شهر را به هم مرتبط کند، $n-1$ است. هر درخت پوشای برای یک گراف، یک جواب امکان‌پذیر برای G است.

حال اگر به یال‌های این گراف هزینه‌ای منسوب شود، مثل هزینه ساخت جاده بین دو شهر، هزینه ایجاد ارتباط بین دو ایستگاه مخابراتی و غیره، در این صورت هزینه یک درخت پوشای (هزینه ایجاد حداقل راههای ارتباطی بین n شهر) برابر مجموع هزینه یال‌های تشکیل دهنده آن خواهد بود.

برای مثال، اگر گراف زیر معرف هزینه ساخت جاده بین چهار شهر d,c,b,a باشد:



دارای هزینه 45 است، مشاهده می شود که درخت های پوشای متعدد، هزینه های متفاوتی دارند. هدف به دست آوردن درخت پوشایی است که این هزینه برای آن مینیمم باشد. همان گونه که بعداً مشاهده خواهیم کرد، برای مثال بالا درخت پوشای مینیمم به صورت زیر خواهد بود:



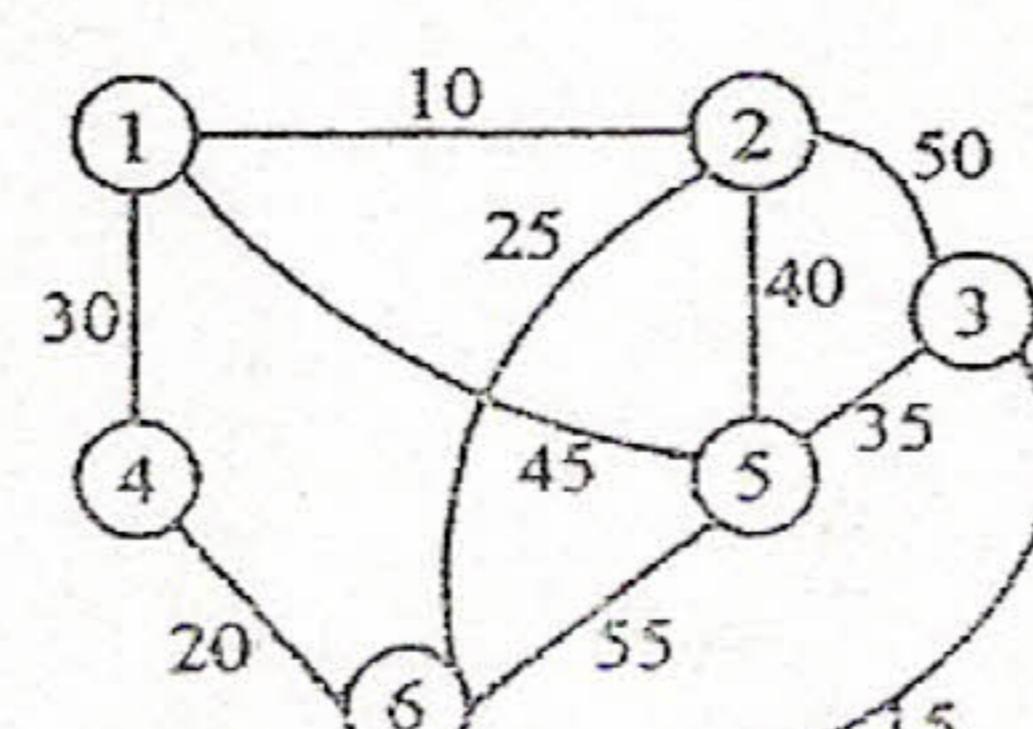
که هزینه آن برابر 30 است.

برای به دست آوردن درخت پوشای یک گراف وزن دار، الگوریتم های کراسکال و پریم ارایه می شوند.

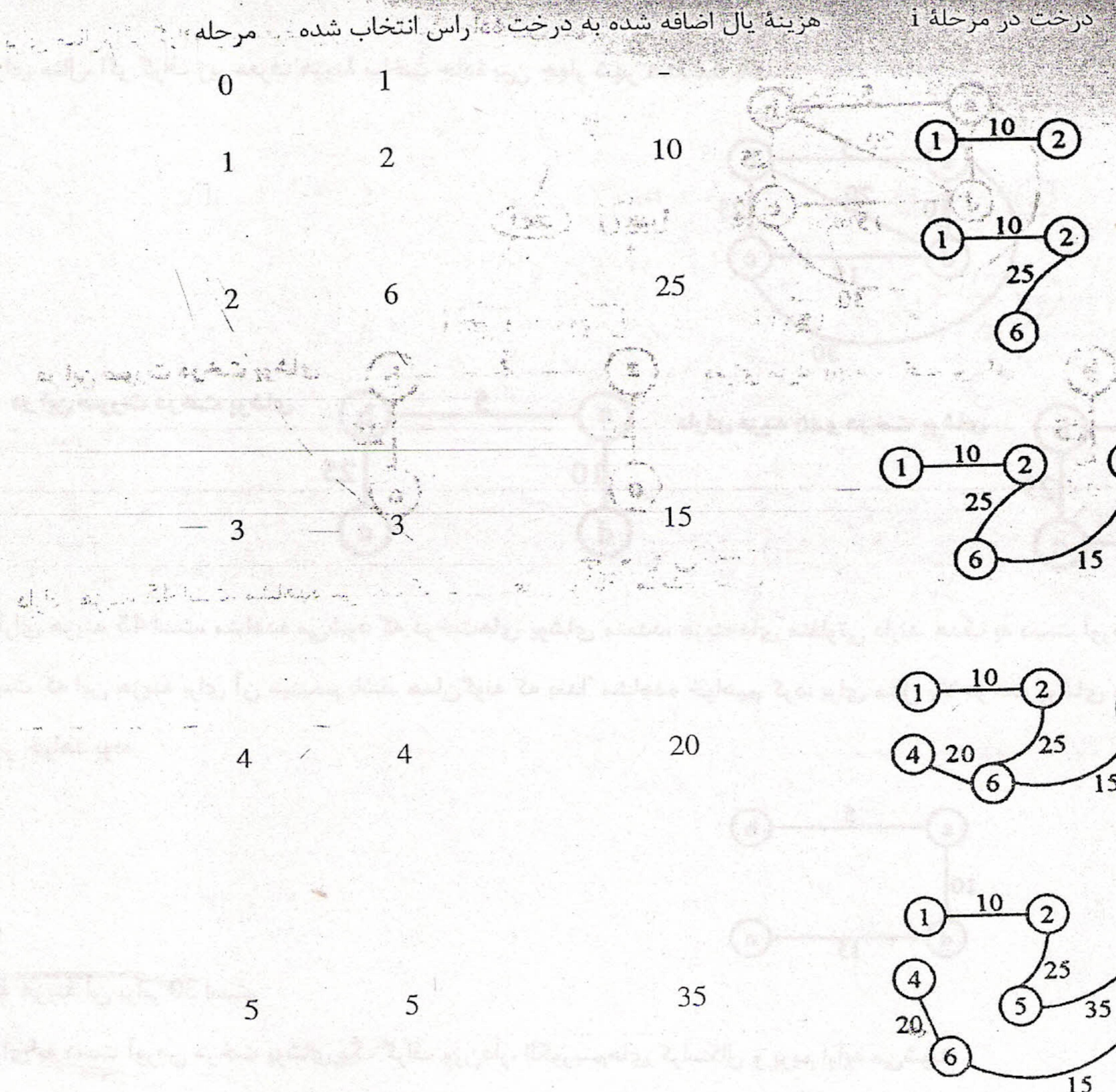
در هر کدام از این الگوریتم ها فرض بر این است که ورودی $G=(V, E, \text{cost})$ یک گراف همبند و وزن دار است که در آن $V=\{1, 2, \dots, n\}$ ، مجموعه رئوس و $\text{cost}[i, j] = 1, 2, \dots, n$ ، یک ماتریس $n \times n$ و معرف هزینه یال هاست (ماتریس هزینه ها). خروجی هر کدام از این الگوریتم ها، آرایه ای از $(n-1)$ یال برای درخت پوشای مینیمم خواهد بود.

الگوریتم پریم: ایده این الگوریتم به این صورت است که از یک راس دلخواه شروع می کنیم (برای سهولت از راس 1 شروع خواهیم کرد)، این راس را به عنوان راس پردازش شده تلقی خواهیم کرد و در یک مرحله عمومی، راسی از بین رئوس پردازش نشده به گونه ای انتخاب خواهیم کرد که با حداقل هزینه به یکی از رئوس قبلاً پردازش شده وصل شده باشد.

شکل های زیر مراحل مختلف انتخاب رئوس را برای گراف شکل مقابل نشان می دهند.



شکل ۱



و جمع هزینه یال‌های انتخاب شده 105 است.

همان‌گونه که مشاهده شد، الگوریتم با درختی که فقط یک راس است (برای سهولت راس 1) شروع می‌شود. بعد از $n-1$ یال درخت پوشای مینیمم، مرحله به این درخت اضافه می‌شوند. یال بعدی $\{j, i\}$ که به این درخت باید اضافه شود به گونه‌ای است که راس i قبل‌اً در درخت بوده و j راسی است که در درخت نیست و $[j, i]$ از بین تمامی $[i, k]$ ها، که در آن k در درخت بوده و 1 در درخت نباشد، مینیمم است. اگر آرایه $Near[1..n]$ را به صورت زیر تعریف کنیم:

$Near[j] = 0$ اگر و تنها اگر راس j قبل‌اً در درخت قرار داده شده و $Near[j] \neq i$ اگر و تنها اگر $[j, i]$ مینیمم باشد، در این صورت مطالب ارایه شده را می‌توان به صورت دقیق‌تر در الگوریتم زیر بیان کرد:

```

Algorithm Prim (cost , n , T , mincost)
mincost=0
for i=2 to n do Near [i]=1
Near [1]=0
for i=1 to n-1 do
{
    jرا به گونه‌ای انتخاب کنید که
    (Near[j]≠0)and cost[j , Near[j]] (*)
```

$$(T[i,1], T[i,2]) = (j, \text{Near}[j])$$

$$m \leftarrow \min \cos t = \min \cos t + \cos [j, \text{Near}[j]]$$

$$\text{Near}[j] = 0$$

for $k = 2$ to n do

if ($\text{Near}[k] \neq 0$) and ($\cos [k, \text{Near}[k]] > \cos [k, j]$) then

$$\text{Near}[k] = j$$

}

(**)

تحلیل الگوریتم پریم

حلقه اصلی $n-1$ بار تکرار می‌شود. زمان اجرای دستور (*) هم $\Theta(n)$ است. حلقة داخلی (دستور **) هم $n-1$ بار اجرا می‌شود. بنابراین مرتبه بزرگی الگوریتم Prim $\Theta(n^2)$ است.

مثال : مراحل مختلف اجرای الگوریتم پریم را برای گراف شکل (1) بنویسید.

حل :

این مراحل در جدول زیر به نمایش در آمدند:

0 مرحله	j	Near[j]	$\cos [j, \text{Near}[j]]$
	1	0	-
	2	1	10
	3	1	∞
	4	1	30
	5	1	45
	6	1	∞

T	1	2	Mincost
1			0
2			
3			
4			
5			

\downarrow
 $j=2$

1 مرحله	j	Near[j]	$\cos [j, \text{Near}[j]]$
	1	0	-
	2	0	-
	3	2	50
	4	1	30
	5	2	40
	6	2	25

T	1	2	Mincost
1	1	2	10
2			
3			
4			
5			

\downarrow
 $j=6$

2 مرحله

j	Near [j]	$\cos[j, \text{Near}[j]]$
1	0	-
2	0	-
3	6	(15)
4	6	20
5	2	40
6	0	-

↓
j=3

3 مرحله

j	Near [j]	$\cos[j, \text{Near}[j]]$
1	0	-
2	0	-
3	0	-
4	6	(20)
5	3	35
6	0	-

↓
j=4

4 مرحله

j	Near [j]	$\cos[j, \text{Near}[j]]$
1	0	-
2	0	-
3	0	-
4	0	-
5	3	(35)
6	0	-

↓
j=5

$(\{1, 2, 3, 4, 5\}) - (\{2, 3, 4, 5\})$

T	1	2	Mincost
1	1	2	35
2	6	2	70
3			
4			
5			

T	1	2	Mincost
1	1	2	50
2	6	2	
3	3	6	
4			
5			

T	1	2	Mincost
1	1	2	70
2	6	2	
3	3	6	
4	4	6	
5			

Mincost

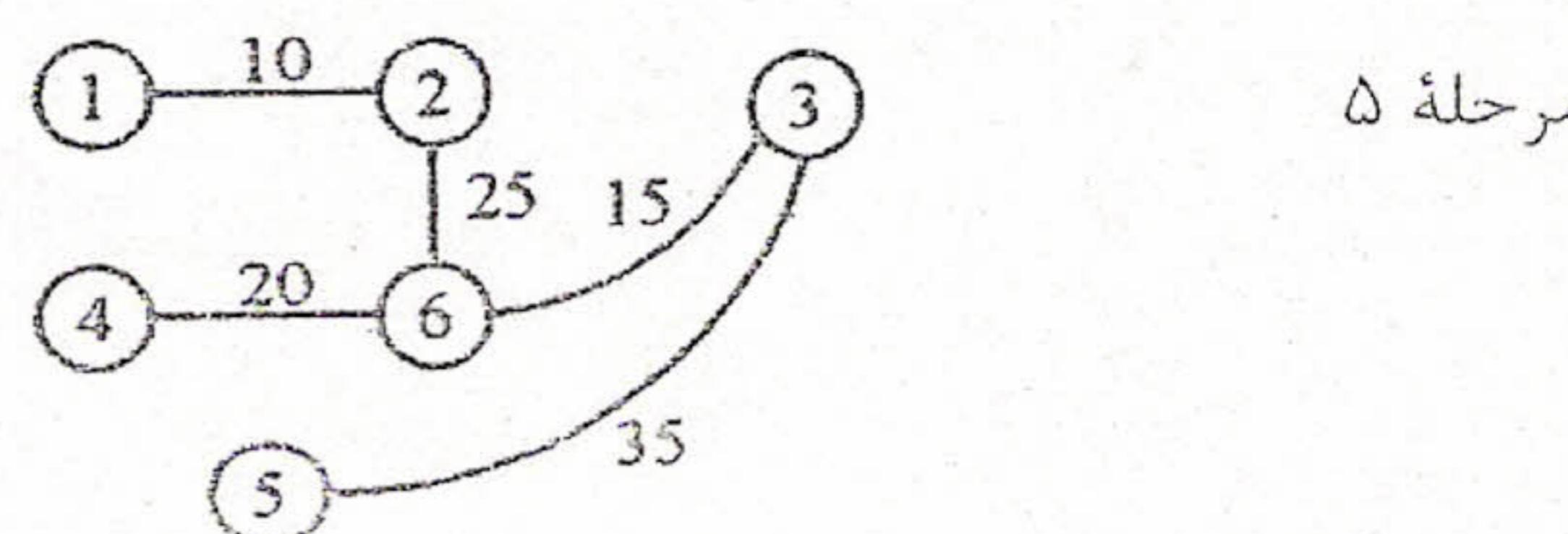
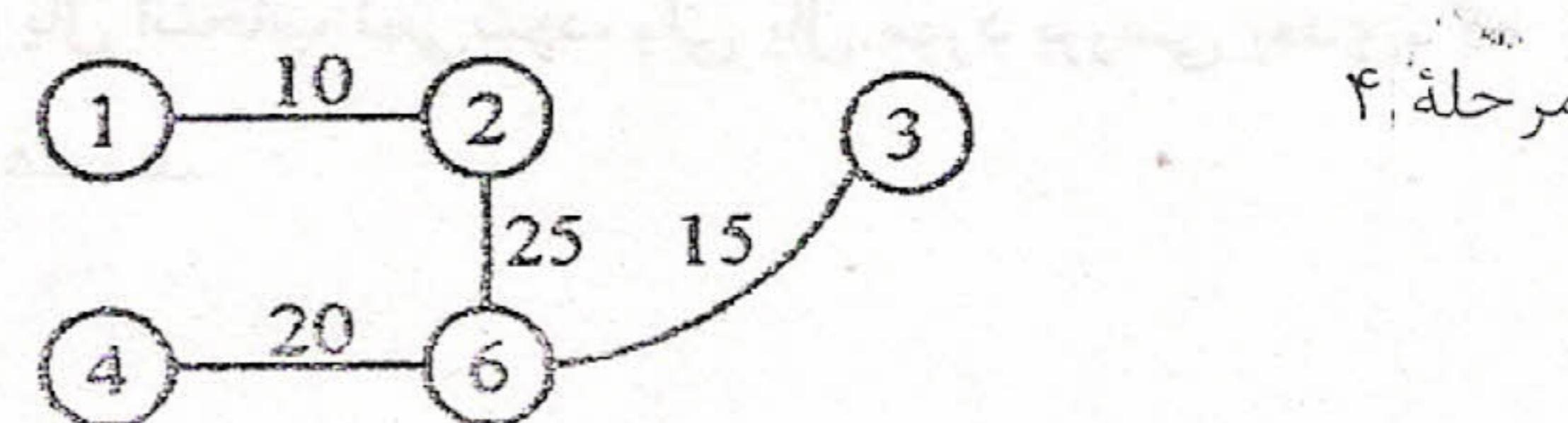
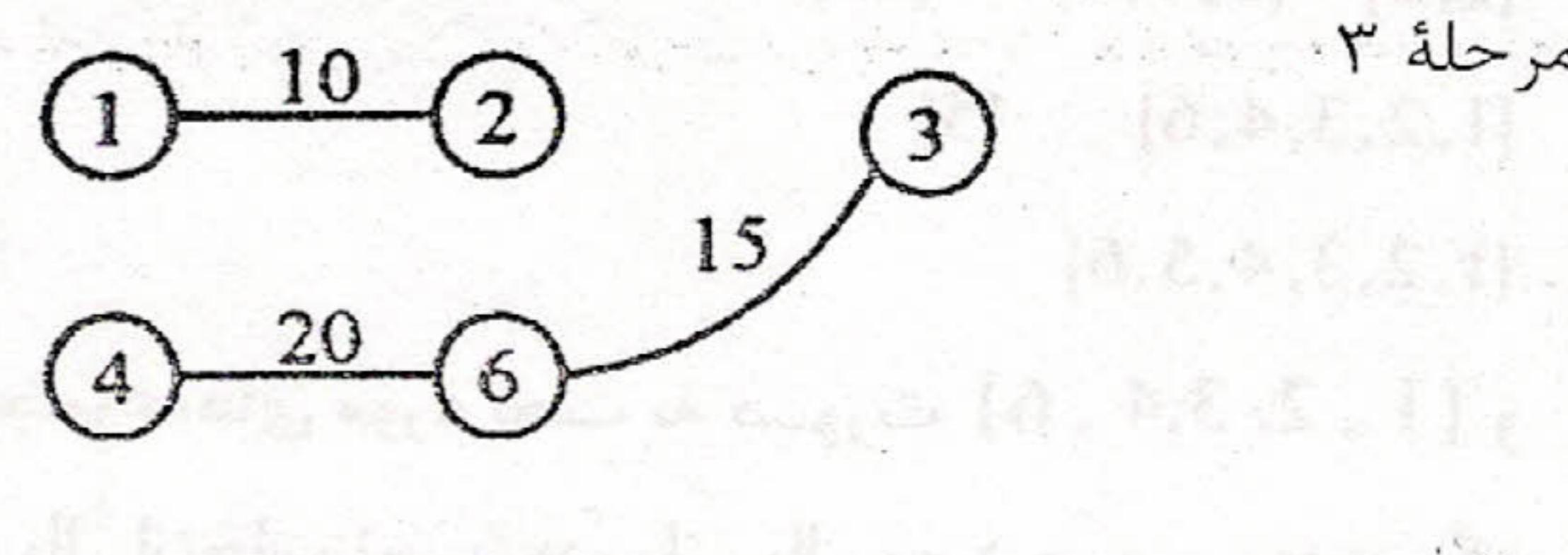
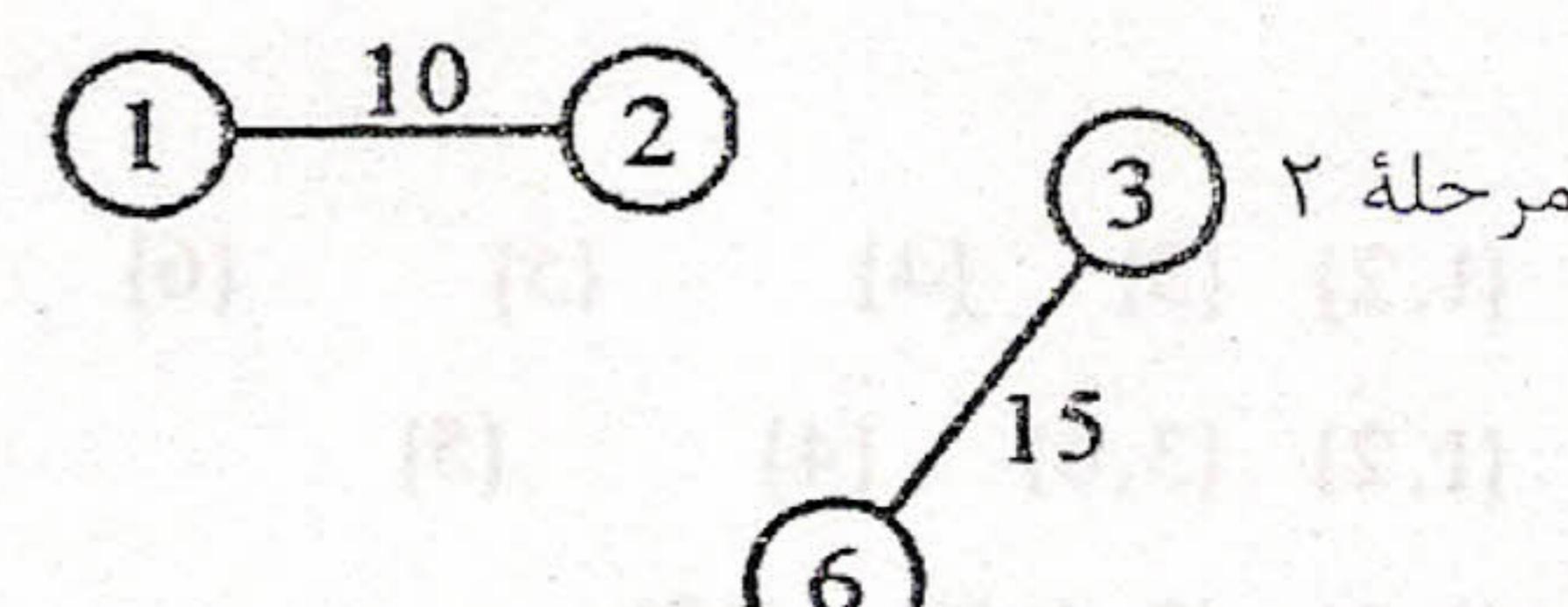
105

الگوریتم کراسکال

خلاصه الگوریتم کراسکال به این صورت است. که ابتداً یال‌ها را به صورت صعودی هزینه آن‌ها مرتب می‌کنیم. سپس با شروع از ایت‌ای این لیست مرتب شده، یالی را انتخاب خواهیم کرد که اضافه کردن آن به مجموعه یال‌های قبلاً انتخاب شده تشکیل دور ندهد. این عمل بعد از انتخاب $(n-1)$ امین یال خاتمه خواهد پذیرفت. مراحل اجرای الگوریتم کراسکال برای گراف شکل ۱، در اشکال زیر به تصویر کشیده شده‌اند:

i	یال	هزینه	وضعیت انتخاب سیال
1	1,2	10	✓
2	3,6	15	✓
3	4,6	20	✓
4	2,6	25	✓
5	1,4	30	-
6	3,5	35	✓
7	2,5	40	
8	1,5	45	
9	2,3	50	
10	5,6	55	

لیست مرتب شده E



با توجه به مطالب بالا الگوریتم کراسکال را می‌توان به صورت دقیق‌تر زیر نوشت:

برای این الگوریتم فرض شده است که نخست یال‌های گراف به صورت صعودی بر حسب هزینه آن‌ها مرتب شده و در آرایه E ذخیره شده‌اند.

Algorithm Kruskal ($E, n, T, \text{Mincost}$)

$\text{Mincost} = 0, N_{\text{edge}} = 0, i = 1, T = \emptyset$

While $N_{\text{edge}} < (n-1)$ do

{

$e = (u, v) = (E[i, 1], E[i, 2])$

if (اضافه کردن یال e به T تشکیل دور ندهد) then

{

$T = TU\{e\}$

$\text{Mincost} = \text{mincost} + \text{cost}[u, v]$

$N_{\text{edge}} = N_{\text{edge}} + 1$

}

$i = i + 1$

}

تحلیل الگوریتم کراسکال: اگر $|E|$ تعداد یال‌های گراف باشد، در این صورت مرتب سازی آن‌ها مستلزم زمان اجرای $O(|E| \log |E|)$ است. حلقه While الگوریتم کراسکال $(|E|)$ بار اجرا می‌شود. مشخص کردن این‌که اضافه کردن E به T تشکیل دور می‌دهد یا نه، مستلزم زمان اجرای $O(\log n)$ است. این عمل با استفاده از ساختمان داده‌های مربوط به مجموعه‌های مجزای امکان‌پذیر است. در شروع کار n مجموعه مجزای $\{1\}, \{2\}, \dots, \{n\}$ را داریم. در مرحله عمومی، یک مجموعه شامل رئوسی خواهد بود که در T به هم مرتبط هستند. بنابراین اگر دو راس u, v ، هر دو متعلق به یک مجموعه بودند، چون بین v, u قبلاً مسیری در T بوده، لذا اضافه کردن یال (u, v) به T تشکیل دور خواهد داد و در نتیجه این یال انتخاب نمی‌شود. اما اگر v, u در دو مجموعه مجزا باشند، در این صورت می‌توان یال (v, u) را به T اضافه کرد، منتهی در این حالت بعد از انتخاب یال (u, v) باید دو زیر مجموعه شامل رئوس u, v را، با اجتماع این دو مجموعه جایگزین کرد. مطالب مزبور برای مراحل 1 تا 5 مثال قبل، در زیر به تصویر کشیده شده است.

در مرحله ۱ داریم:

$\{1, 2\}, \{3\}, \{4\}, \{5\}, \{6\}$

در مرحله ۲ داریم:

$\{1, 2\}, \{3, 6\}, \{4\}, \{5\}$

در مرحله ۳ داریم:

$\{1, 2\}, \{3, 4, 6\}, \{15\}$

در مرحله ۴ داریم:

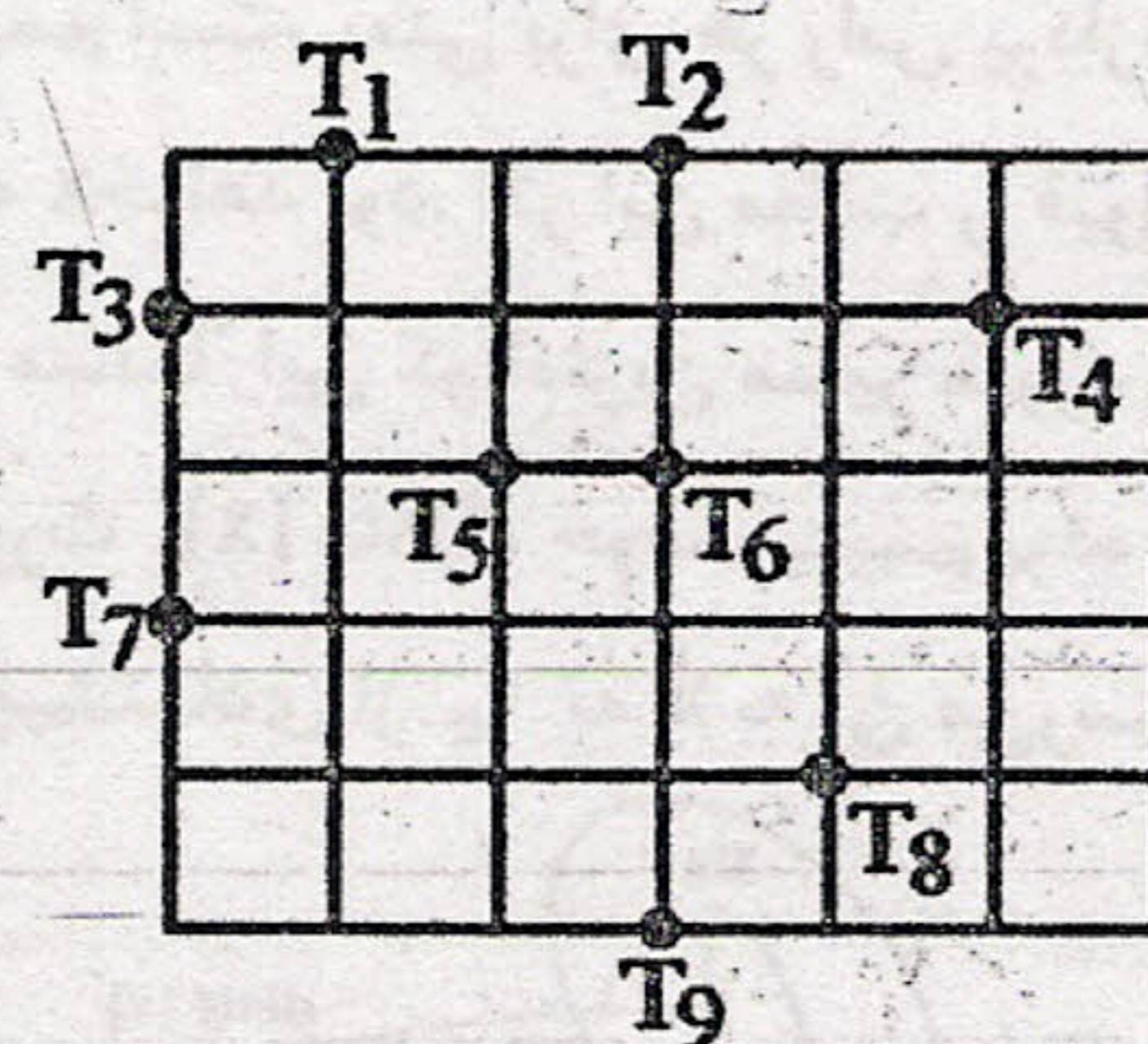
$\{1, 2, 3, 4, 6\}, \{5\}$

در مرحله ۵ داریم:

$\{1, 2, 3, 4, 5, 6\}$

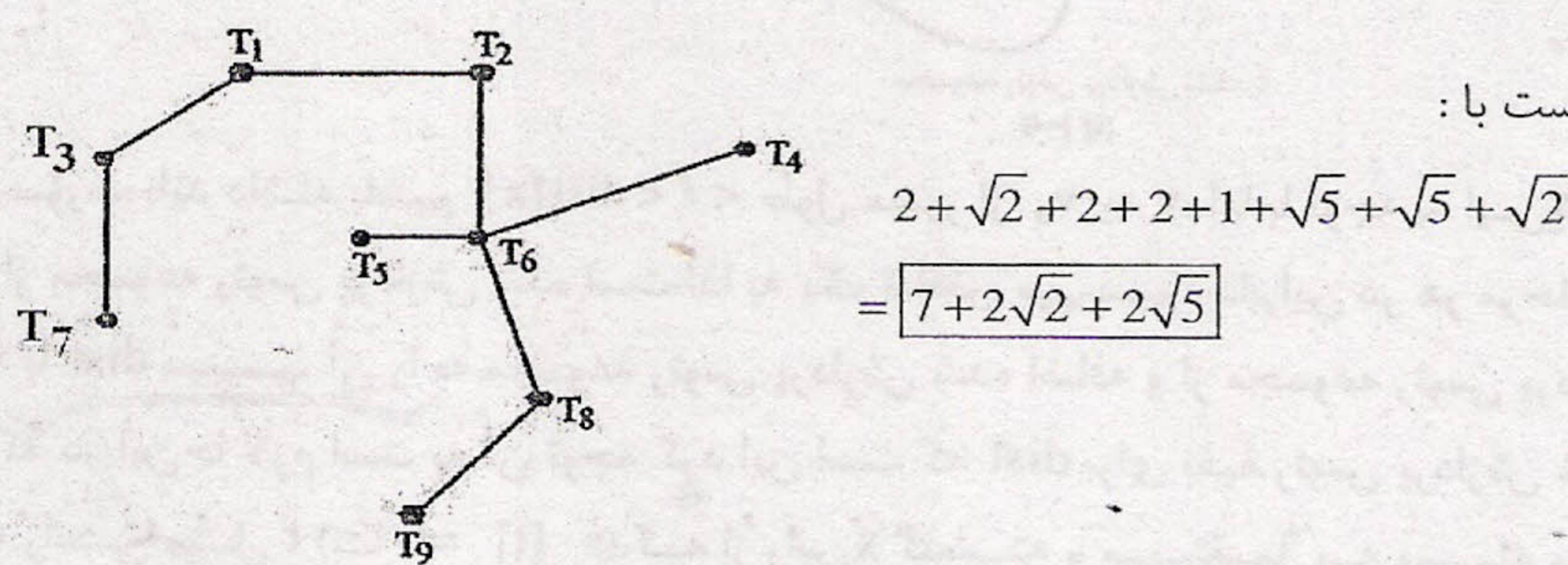
برای مثال، در مرحله 5 نخست یال $\{1, 4\}$ بررسی می‌شود. چون در مرحله 4 مجموعه‌های مورد بحث به صورت $\{1, 2, 3, 4, 6\}$ و $\{5\}$ هستند و هر دو راس 4, 1 متعلق به یکی از این دو مجموعه هستند، لذا این یال انتخاب نمی‌شود، ولی یال مورد بررسی بعدی، که یال $\{3, 5\}$ است، انتخاب می‌شود، زیرا رئوس 3, 5 متعلق به دو مجموعه مجزا هستند.

مثال : یک مهندس برق مداری با ۹ ترمینال طراحی کرده است که باید ولتاژی معادل ۵ ولت به کمترین سیم‌بندی در مدار به کار برود، حداقل سیم‌های به کار رفته را به دست آورید. فاصله هر سطر و ستون را برابر یک سانتی‌متر در نظر بگیرید.



حل :

اگر این ترمینال‌ها (نقاط) را به منزله هفت راس یک گراف کامل در نظر بگیریم و وزن هر یال از این گراف کامل را برابر فاصله اقلیدسی دو راس (نقطه) به حساب آوریم، در این صورت با استفاده از الگوریتم کراسکال می‌توان درخت پوشای مینیمم زیر را برای گراف مذبور به دست آورد:



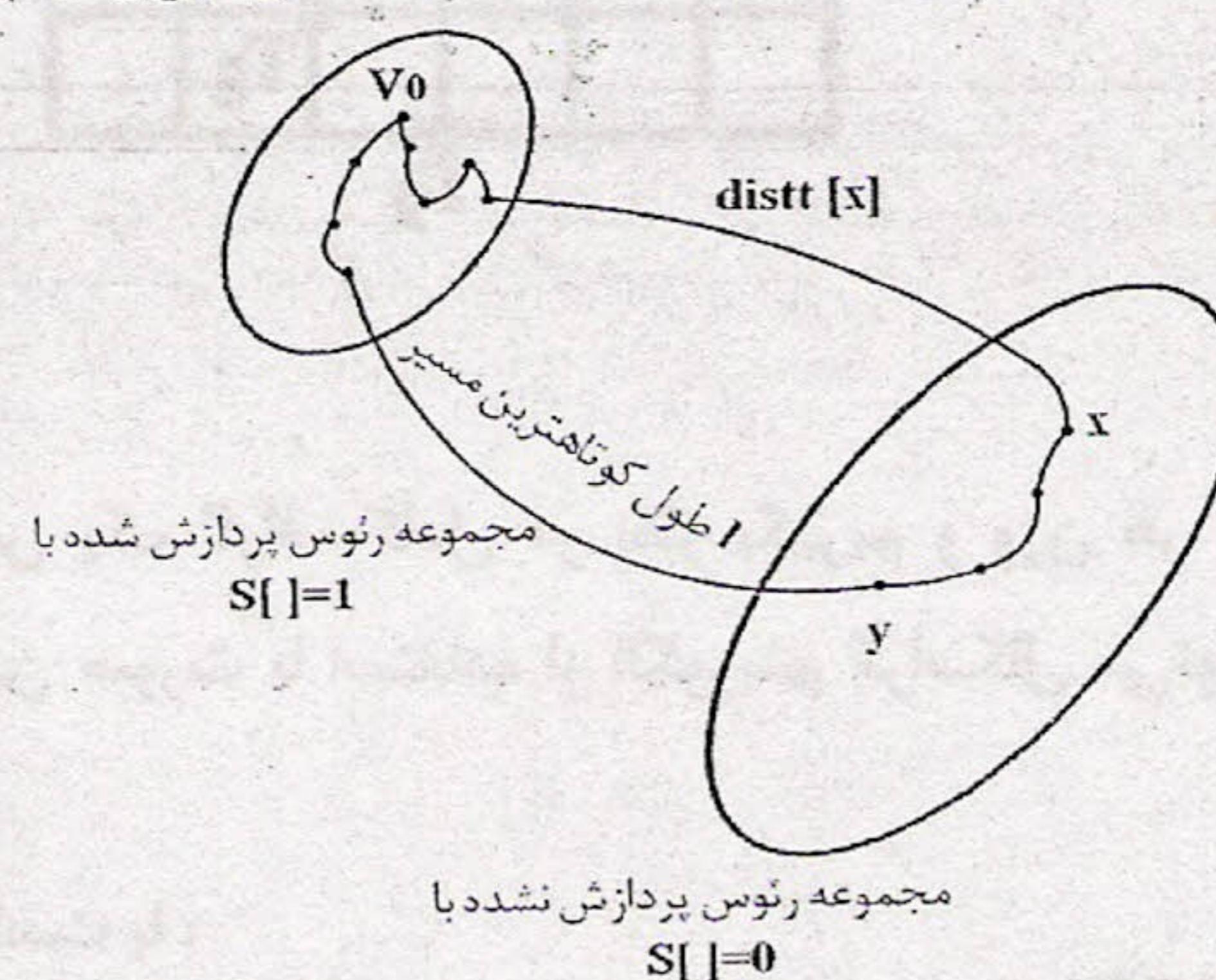
کوتاه‌ترین مسیرهای هم مبدأ (الگوریتم دایکسترا)

فرض کنید $G=(V, E, Cost)$ یک گراف وزن‌دار است. اگر V مجموعه رئوس را به منزله n شهر در نظر بگیریم و E مجموعه یال‌ها را مشابه جاده‌های بین شهرها تصور کنیم، در این صورت برای $[j, i]$ می‌توان مفاهیمی چون مسافت بین دو شهر i, j یا میانگین مدت زمان رانندگی بین این دو شهر و غیره را در نظر گرفت. در هر صورت $[j, i]$ Cost را وزن یا هزینه یال $[j, i]$ می‌نامیم. طول یک مسیر برابر مجموع وزن یال‌های تشکیل دهنده آن است. اولین راس مسیر را راس مبدأ، آخرین راس را راس مقصد و بقیه رئوس را رئوس داخلی مسیر می‌نامیم.

در مسأله کوتاه‌ترین مسیرهای هم مبدأ، هدف تعیین کوتاه‌ترین مسیرها از یک راس مشخص مثل v_0 به عنوان راس مبدأ به کلیه رئوس گراف است. برای این منظور الگوریتم دایکسترا ارایه می‌شود. در این الگوریتم حریصانه، کوتاه‌ترین مسیرها از v_0 به بقیه رئوس به ترتیب صعودی طول آن‌ها به دست می‌آیند. در ابتدا کوتاه‌ترین مسیر از v_0 به دست می‌آید و در مرحله بعد، کوتاه‌ترین مسیر بعدی و الی آخر.

ایده الگوریتم دایکسترا تقریباً مشابه الگوریتم پریم است. در اینجا نیز در هر مرحله، رئوس V به دو مجموعه پردازش شده و پردازش نشده افزایش می‌شوند. در شروع کار v_0 (راس مبدأ) جزو رئوس پردازش شده قرار می‌گیرد، بقیه رئوس پردازش نشده‌اند. برای این منظور می‌توان از آرایه بولی $S[i] = 1..n$ استفاده کرد. اگر برای مثال i راس v_i پردازش شده است. در هر مرحله کوتاه‌ترین مسیر از

v_0 به راسی از رئوس پردازش نشده مثل j ($S[j]=0$) پیدا می‌شود که کوتاهترین فاصله را از بین بقیه رئوس از v_0 دارد. برای این منظور، $[j] = 0$ را برای S به این صورت تعریف می‌کنیم که $[j]$ $dist$ برابر طول کوتاهترین مسیر از راس v_0 به راس j است، به گونه‌ای که کلیه رئوس داخلی آن (در صورت وجود) از مجموعه رئوس پردازش شده باشند. مسلماً $[j]$ برابر طول کوتاهترین مسیر از v_0 به j نیست، زیرا کوتاهترین مسیر از v_0 به j می‌تواند شامل رئوی باشد که هنوز پردازش نشده‌اند، با وجود این اگر فرض کنیم که برای راس پردازش نشده‌ای مثل x , $[x]$ مینیمم است، یعنی برای هر راس پردازش نشده دیگری مثل j , $dist[x] \leq dist[j]$ ، در این صورت $[x]$ طول کوتاهترین مسیر از v_0 به x خواهد بود. اگر این مطلب را قبول ندارید، فرض کنید که طول کوتاهترین مسیر از v_0 به x برابر ℓ است، بنابراین $\ell < dist[x]$. مسلماً این کوتاهترین مسیر مورد ادعا، حداقل یک راس داخلی از مجموعه رئوس پردازش نشده خواهد داشت، زیرا، در غیر این صورت $[x]$ خود، مینیمم بوده است. فرض کنید y اولین راس از مجموعه رئوس پردازش نشده است که در این کوتاهترین مسیر پیشنهادی از v_0 به x به آن می‌رسیم (به شکل زیر مراجعه کنید).



در این صورت باید داشته باشیم $\ell < dist[x]$ طول مسیر از v_0 به y ، اما با توجه به این که مسیر از v_0 به y تنها شامل رئوس داخلی از مجموعه رئوس پردازش شده است، لذا به یک تناقض می‌رسیم. بنابراین در هر مرحله، بعد از مشخص کردن راس پردازش نشده x با $dist$ مینیمم، آن را به مجموعه رئوس پردازش شده اضافه و از مجموعه رئوس پردازش نشده کم می‌کنیم ($S[x]=1$). نکته مهمی که در اینجا لازم است به آن توجه کرد این است که $dist$ برای بقیه رئوس پردازش شده، باید به هنگام شود. زیرا اگر مسیری از v_0 به راسی مثل t با $t=0$ (با s), که از راس x گذشته و مستقیماً به وسیله یالی به آن وصل شده باشد و رابطه $dist[x] + cost[x, t] < dist[t]$ برقرار باشد، بدیهی است که کمیت $dist[x] + cost[x, t] = dist[t]$ باشد. جایگزین $dist[t]$ شود ($dist[t] = dist[x] + cost[x, t]$).

مطلوب گفته شده به صورت دقیق‌تر در الگوریتم منسوب به دایکسترا آورده شده‌اند:

Algorithm Shortest_Paths (v_0 , $cost$, $dist$, n)

```

for i=1 to n do
{
    S[i]=0, dist[i]=cost[v0, i]
}
S[v0]=1, dist[v0]=0
for num=2 to (n-1) do
{

```

فرض کنید x راسی باشد که $S[x]=0$ و $dist[x]$ مینیمم است.

$S[x]=1$

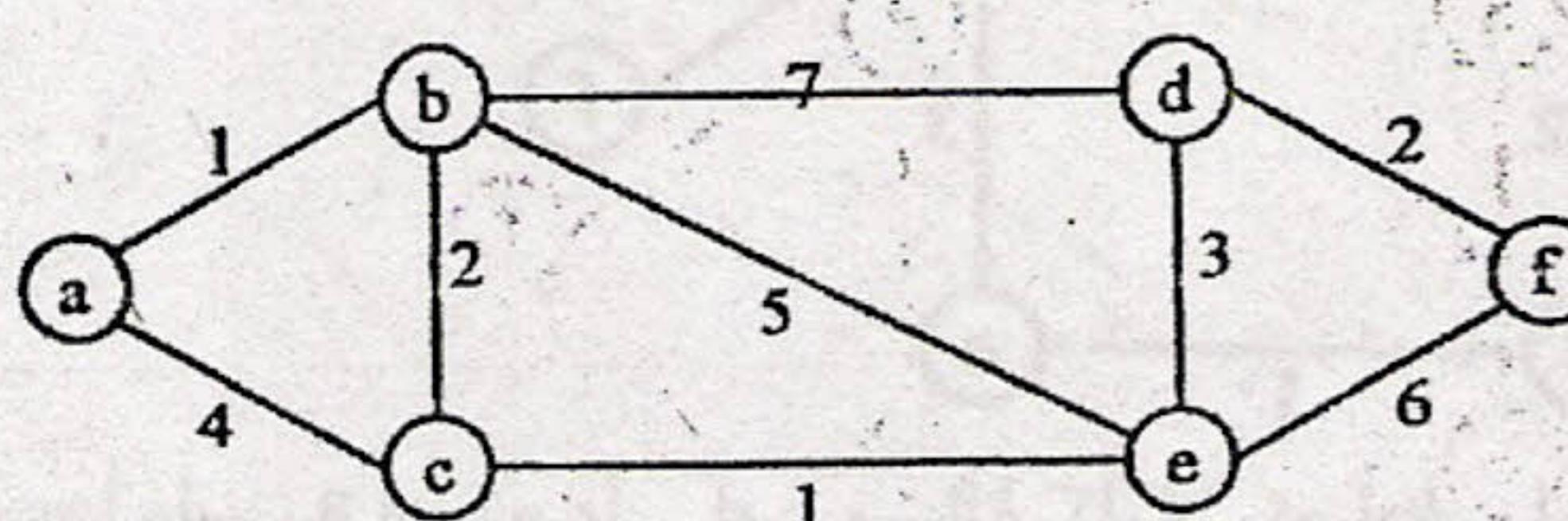
for (برای هر راس t حادث از) do

```

if (S[t]=0) and (dist[x]+cost[x, t] < dist[t]) then
    dist[t]=dist[x]+cost[x, t]
}
```

تحلیل الگوریتم مذبور: حلقة اصلی $2 - n$ بار تکرار می‌شود. پیدا کردن راس x با $[x]$ dist مینیمم مستلزم زمان اجرای $O(n)$ است. حلقة for داخلی نیز از مرتبه $O(n)$ است. بنابراین مرتبه بزرگی الگوریتم ارایه شده $O(n^2)$ است.

مثال: مراحل مختلف اجرای الگوریتم بالا را برای گراف ارایه شده در زیر، برای تعیین کوتاهترین مسیر از راس $(V_0 = a)$ به کلیه رئوس دیگر به دست آورید.



حل:

این مراحل در جداول زیر خلاصه شده‌اند:

راس	s	dist	مسیرها
a	1	-	-
b	0	1	a,b
c	0	4	a,c
d	0	∞	-
e	0	∞	-
f	0	∞	-

راس	s	dist	مسیرها
a	1	-	-
b	1	1	a,b
c	0	3	a,b,c
d	0	8	a,b,d
e	0	6	a,b,e
f	0	∞	-

$$\Rightarrow x = b$$

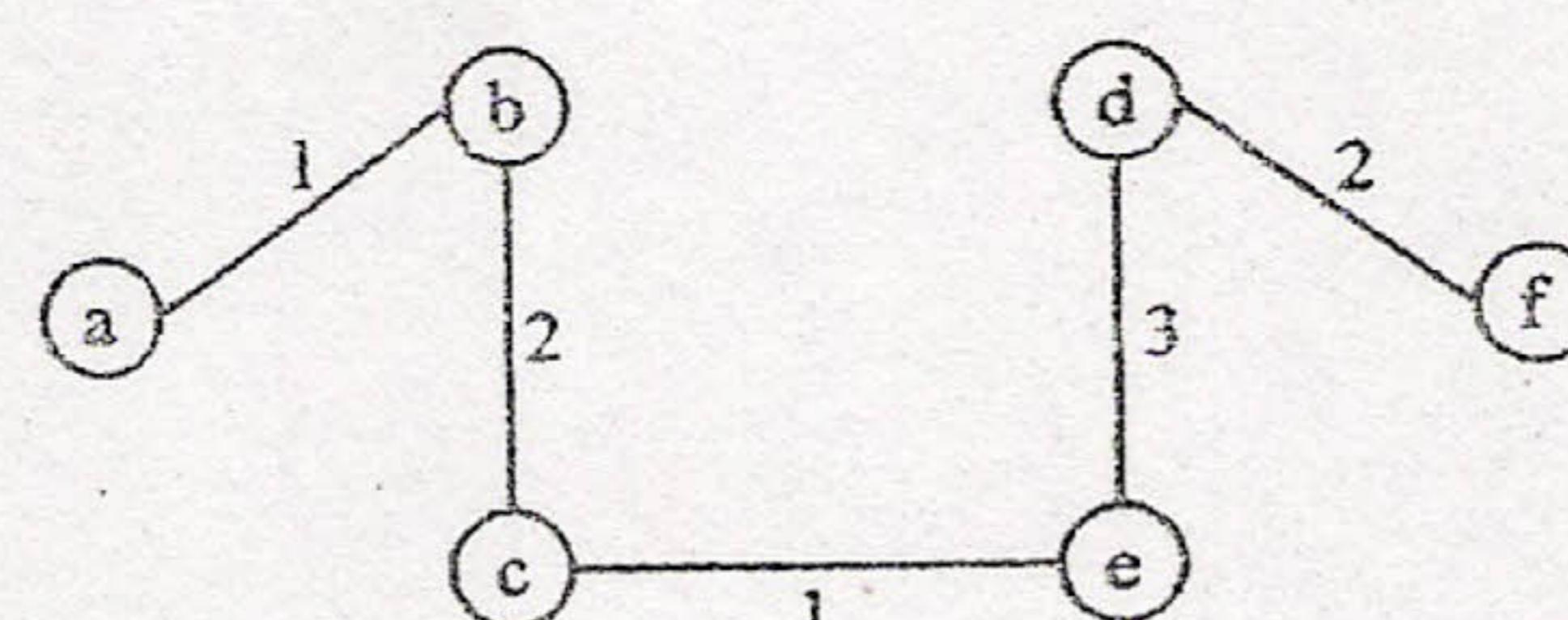
راس	s	dist	مسیرها
a	1	-	-
b	1	1	a,b
c	1	3	a,c
d	0	8	-
e	0	4	-
f	0	∞	-

راس	s	Dist	مسیرها
a	1	-	-
b	1	1	a,b
c	1	3	a,b,c
d	0	7	a,b,c,e,d
e	1	4	a,b,c,e
f	0	10	a,b,c,e

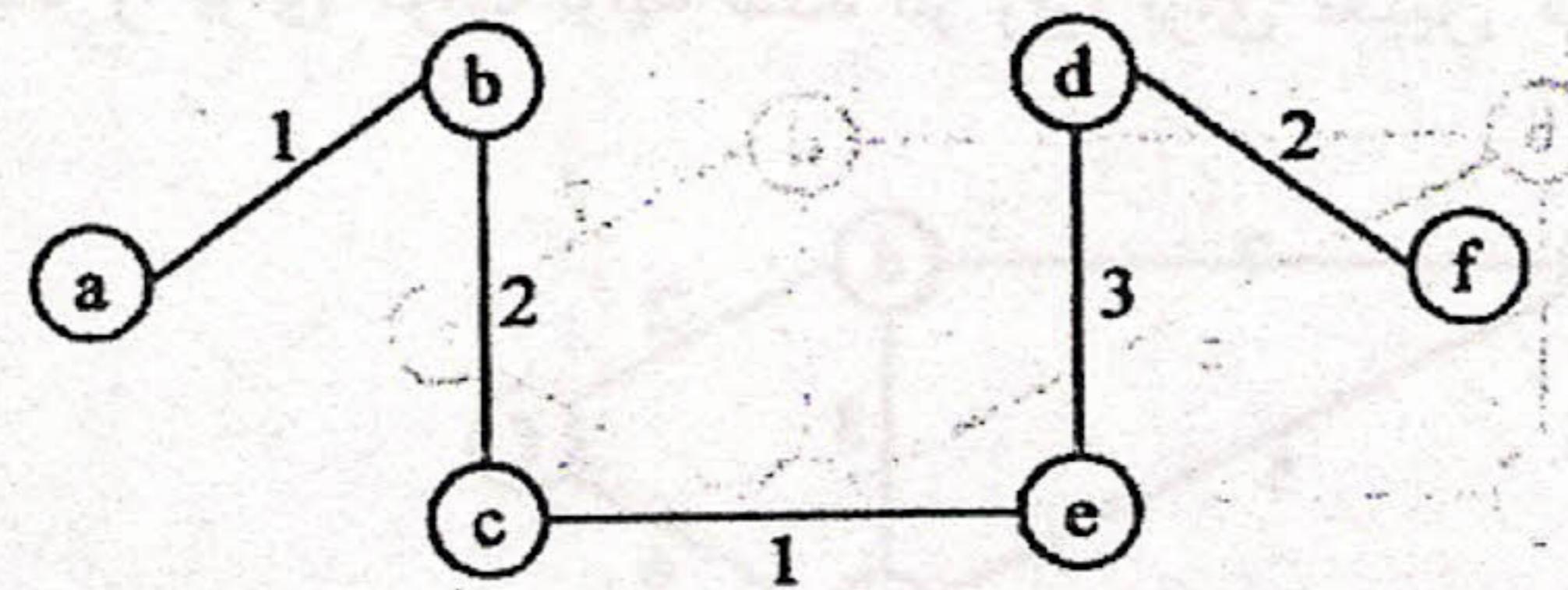
$$\Rightarrow x = e$$

راس	s	dist	مسیرها
a	1	-	-
b	1	1	a,b
c	1	3	a,b,c
d	1	7	a,b,c,e,d
e	1	4	a,b,c,e
f	0	9	a,b,c,e,d,f

اگر در مثال مذبور، کوتاهترین مسیرهای از راس a به بقیه رئوس را رسم کنیم، نتیجه زیر را خواهیم داشت:

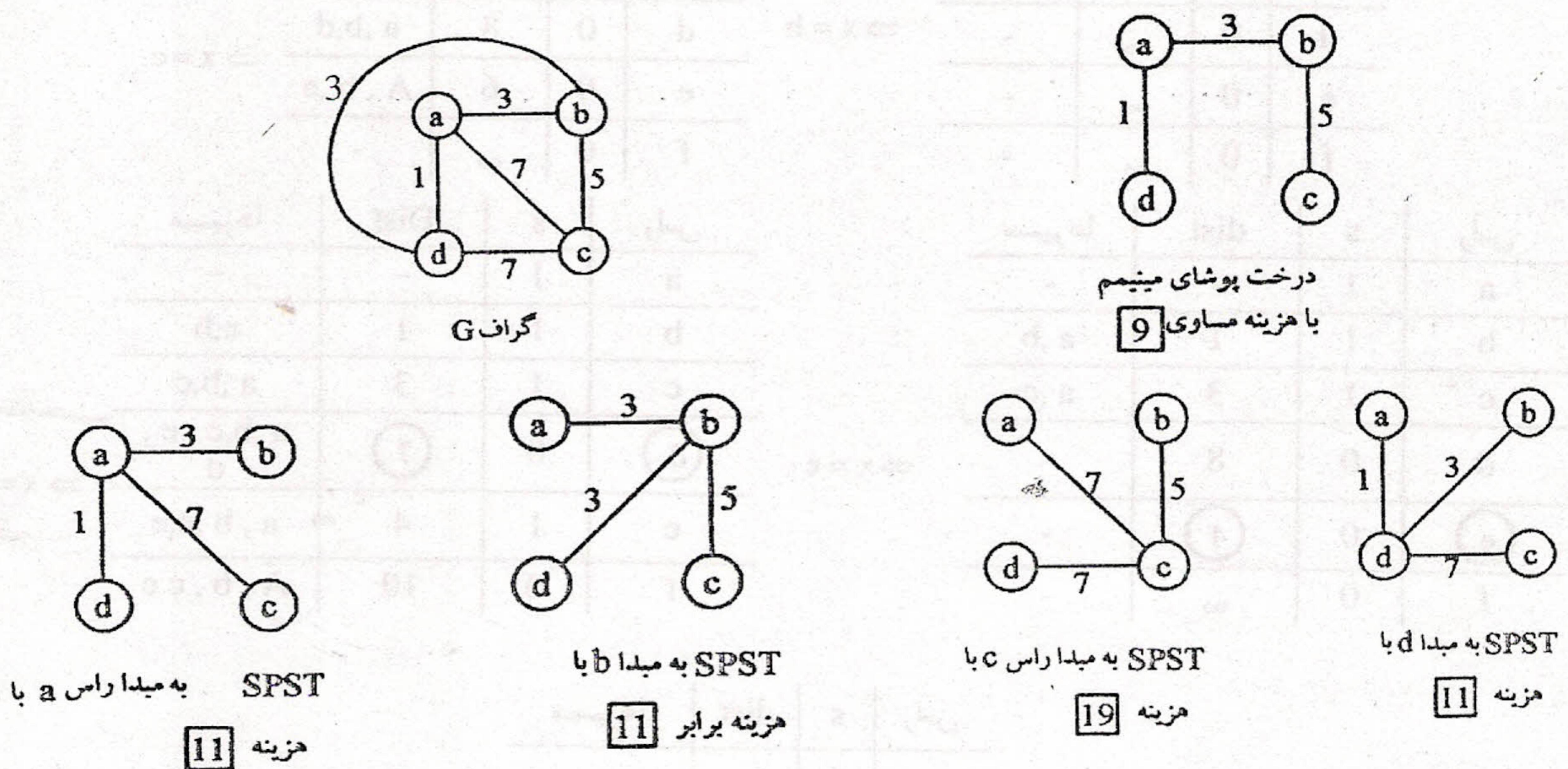


که یک درخت پوشای کوتاهترین مسیر (SPST)^۱ به مبدأ a گوییم. در حالت عمومی اگر گراف G همبند و بدون جهت باشد، الگوریتم ShortestPaths با راس مبدأ v_0 منجر به یک درخت پوشای می شود که به نام درخت پوشای کوتاهترین مسیر به مبدأ v_0 نامیده می شود. حال سؤال این است که آیا یک SPST الزاماً یک درخت پوشای مینیمم است؟ اگر درخت پوشای مینیمم را برای گراف مزبور به دست آوریم، نتیجه به صورت زیر خواهد بود:



که عیناً مشابه SPST به مبدأ راس a است، ولی این مسأله کاملاً تصادفی است. در حالت عمومی حتی اگر SPST را برای گراف مینیمم باز انتخاب کنیم، باز امکان دارد که درخت پوشای مینیمم حاصل نشود. در این مورد به مثال زیر توجه کنید:

مثال : درخت پوشای مینیمم و نیز همه SPST ها را برای گراف زیر به دست آورده و مطالب بالا را تحقیق کنید.



مشاهده می شود که هزینه همه SPST های گراف مزبور بیشتر از هزینه درخت پوشای مینیمم است. البته علت امر بدیهی است. دیدیم که برای مثال گراف کامل k^n دارای n^{n-2} درخت پوشای مینیمم است، حال این که تعداد SPST هایی که برای یک گراف می توان به دست آورد برابر n است.

¹- Shortest path spanning Tree

مسئله انتخاب فعالیت‌ها (زمان‌بندی فعالیت‌ها)

مجموعه‌ای از n فعالیت $\{1, 2, \dots, n\}$ داده شده است که می‌خواهند از یک منبع واحد مثل یک سالن کنفرانس یا سالن مطالعه استفاده کنند. در یک زمان داده شده فقط یک فعالیت می‌تواند از این منبع استفاده کند. هر فعالیت i دارای زمان شروع s_i و زمان خاتمه f_i است. فعالیت i در صورت انتخاب در فاصله زمانی $[s_i, f_i]$ انجام خواهد شد. فعالیت‌های i و j را سازگار گوییم هر گاه همپوشانی در فاصله زمانی نداشته باشند به عبارت دیگر، i و j سازگار هستند هر گاه $s_i \geq f_j$ یا $s_j \geq f_i$. مسئله انتخاب فعالیت‌ها، انتخاب بیشترین تعداد از فعالیت‌های دو به دو سازگار است.

الگوریتم حریصانه برای مسئله انتخاب فعالیت‌ها در زیر ارایه شده است، فرض بر این است که فعالیت‌های ورودی به صورت صعودی زمان خاتمه‌شان مرتب شده‌اند:

$$f_1 \leq f_2 \leq f_3 \leq \dots \leq f_n$$

در غیر این صورت باید آن‌ها به ترتیب یاد شده مرتب کرد. همچنین فرض بر این است که زمان‌های شروع و خاتمه فعالیت‌ها به ترتیب توسط آرایه‌های s و f به عنوان ورودی به این الگوریتم داده شده‌اند.

الگوریتم انتخاب فعالیت‌ها

Algorithm Activity_Selection(s, f)

```

A = {1}, k = 1
for i = 2 to n do
    if  $s[i] \geq f[k]$  then
    {
        A = A ∪ {i}
        k = i
    }
return A

```

فعالیت‌های انتخاب شده در مجموعه A جمع می‌شوند. متغیر k معرف آخرین فعالیت انتخاب شده (اضافه شده به A) است. چون فعالیت‌ها به ترتیب صعودی زمان خاتمه آن‌ها در نظر گرفته می‌شوند، f_k همواره بیشترین زمان خاتمه از میان زمان خاتمه هر فعالیت‌ها موجود در A خواهد بود:

$$f_k = \text{Max} \{ f_i | i \in A \}$$

با این فرض که فعالیت‌ها از قبل به صورت صعودی زمان خاتمه‌شان مرتب‌اند، مرتبه زمانی الگوریتم مذبور $O(n^{\theta})$ است. بنابراین مسئله انتخاب فعالیت‌ها را می‌توان در مرتبه زمانی $O(n \log n)$ حل کرد.

مثال: مراحل مختلف اجرای الگوریتم فوق را برای فعالیت‌های زیر بنویسید.

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

حل:

با توجه به این که فعالیت‌ها از اول به صورت صعودی زمان خاتمه‌شان مرتب شده‌اند، نیازی به مرتب‌سازی آن‌ها نیست.

مراحل مختلف اجرای الگوریتم مذکور در شکل زیر آمده است:

