



دانشگاه شیدا بهشتی کرمان

# هفت: درخت

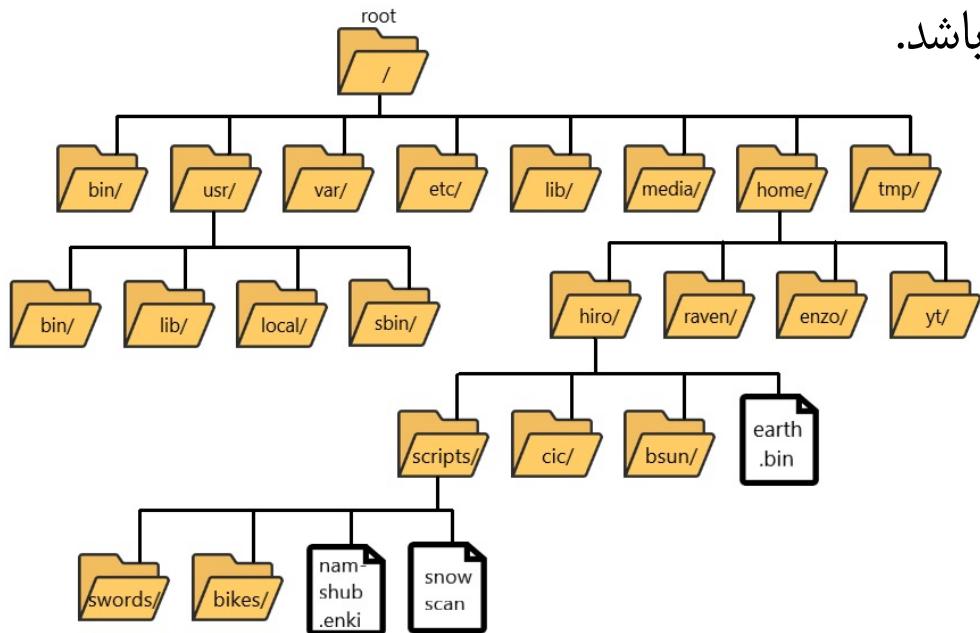
ساختمان داده ها و الگوریتم

مدرس: دکتر نجمه منصوری

نگارنده: سجاد هاشمیان

# درخت

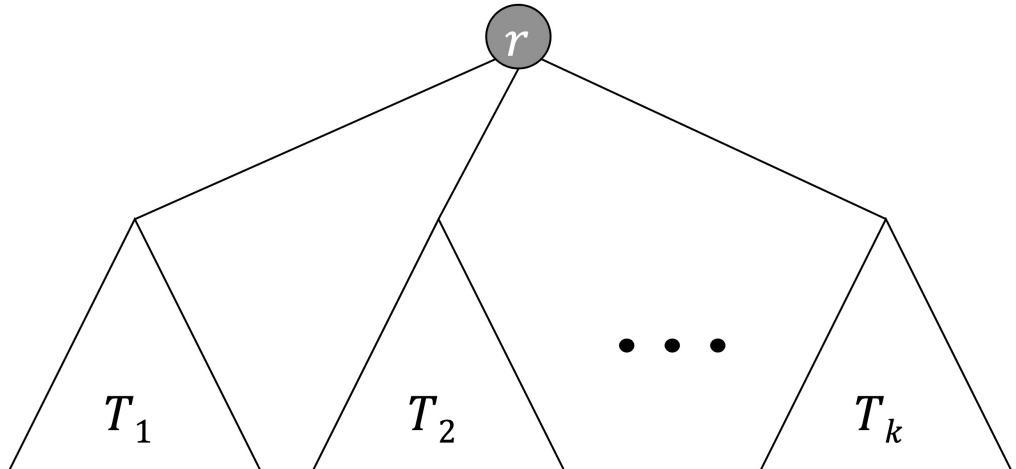
- درخت یک مدل انتزاعی برای نمایش ساختارهای سلسله‌مراتبی است.
- یک درخت شامل گره‌هایی است که بین آنها ساختار غیر خطی برقرار است.
- به جز بالاترین عنصر (ریشه)، هر عنصر دقیقاً یک پدر دارد.
- همچنین، هر عنصر در درخت میتواند صفر یا چند فرزند داشته باشد.



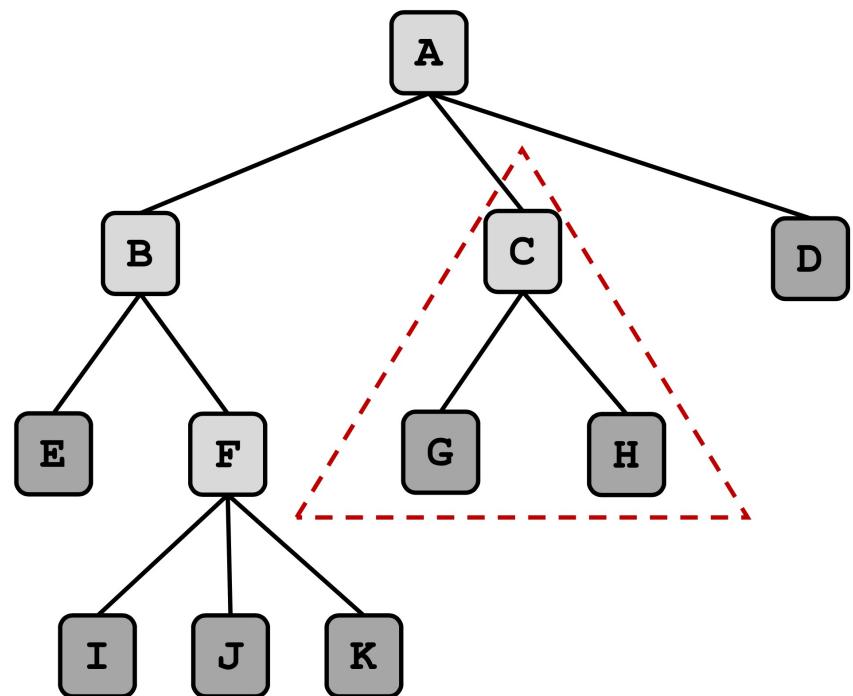
# درخت ریشه دار

درخت ریشه دار یک مجموعه از گره ها با ویژگی های زیر است:

- اگر تهی نباشد، دارای یک گره خاص به نام **ریشه** است.
- سایر گره ها به  $k \geq 0$  مجموعه ای مجزا مانند  $T_1, T_2, \dots, T_k$  تقسیم می شوند به گونه ای که هر یک از این مجموعه ها خود یک درخت ریشه دار هستند.

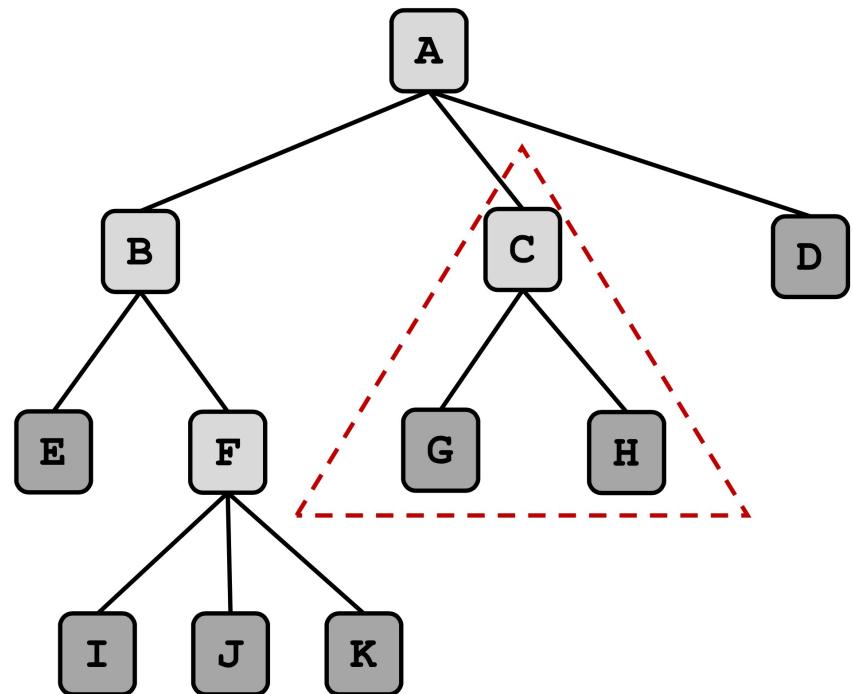


# اصطلاحات و تعاریف



- ریشه: گره بدون والد.
- گرہ داخلی: گرہای کے حداقل یک فرزند دارد.
- برگ (گرہ خارجی): گرہ بدون فرزند.
- گرہای همنیا: گرہ ہائی کے والد یکسان دارند.

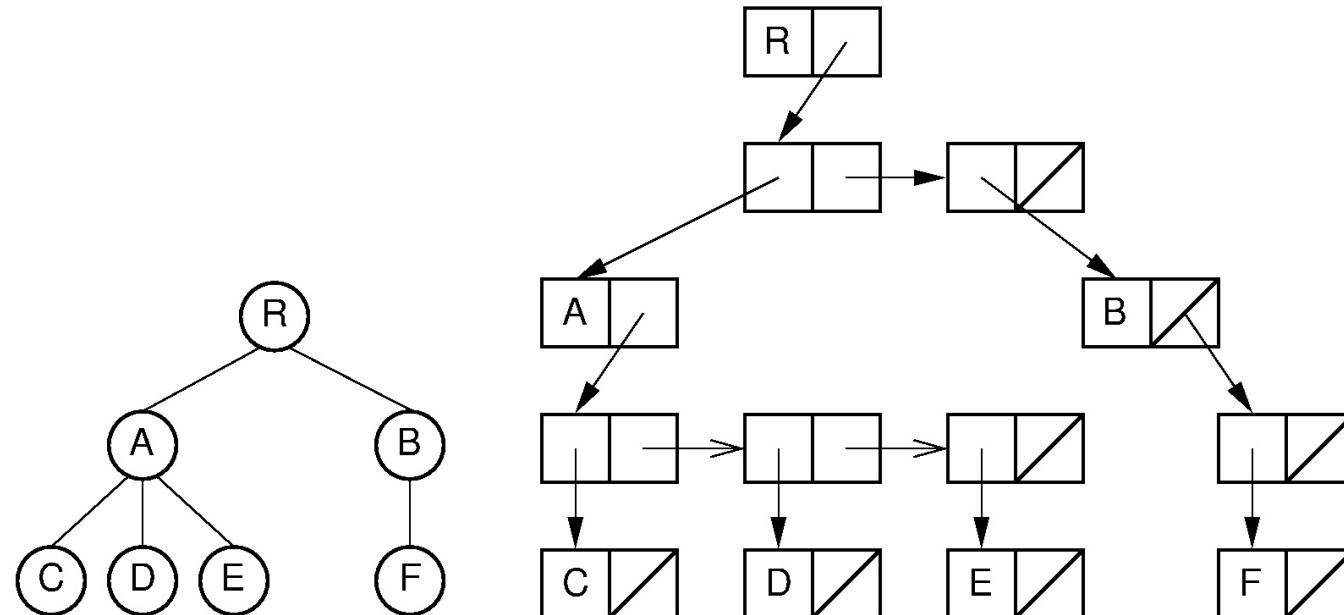
# اصطلاحات و تعاریف



- عمق گره: طول مسیر از ریشه تا آن گره.
- ارتفاع درخت: حداقل عمق گره‌ها در درخت.
- اجداد یک گره: پدر، پدر بزرگ، پدر پدر بزرگ و ...
- نسل‌های یک گره: فرزند، فرزند فرزند و ...

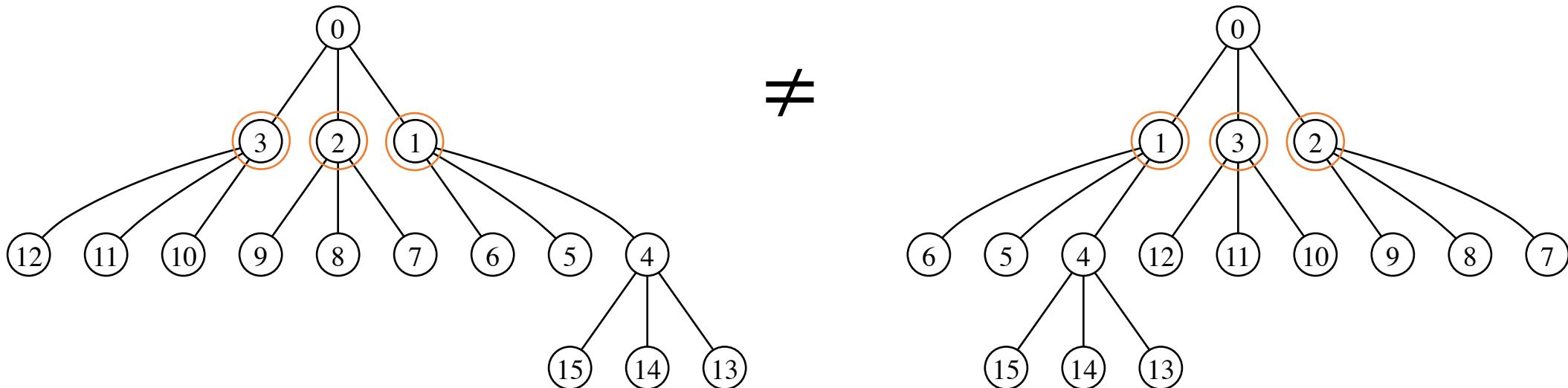
# نمایش درخت

با توجه به خلوت بودن گراف‌های درخت، عموماً از لیست مجاورت برای پیاده‌سازی درختان استفاده می‌شود و استفاده از ماتریس مجاورت پیشنهاد نمی‌شود.



# درخت k-تایی

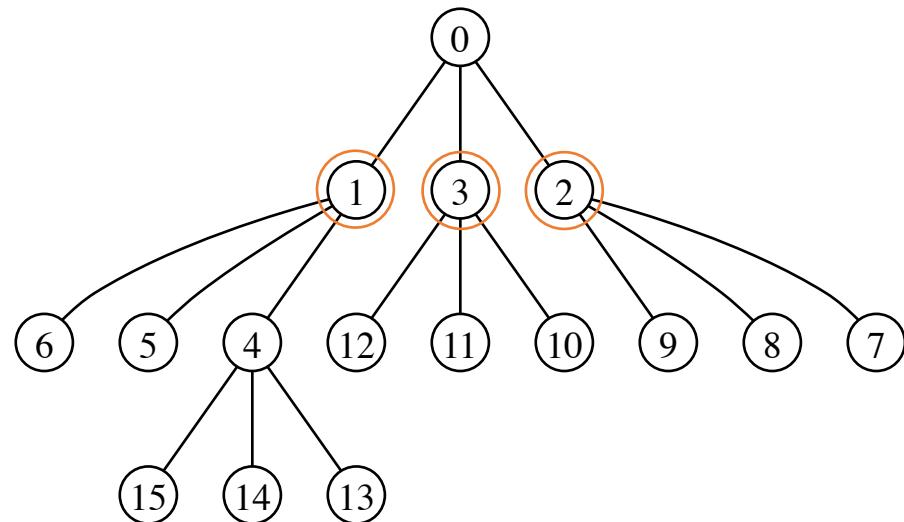
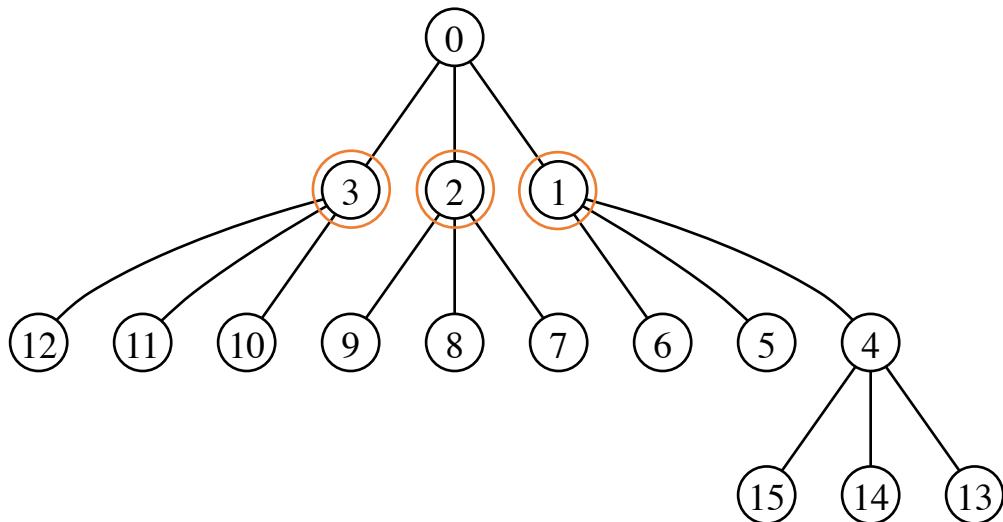
به درختی گفته می‌شود که هر گره داخلی آن دقیقا  $k$  فرزند داشته باشد؛  
برخلاف تعریف عمومی درختان، در درختان  $k$ -تایی ترتیب قرار گرفتن این فرزندان مهم است.



# درخت k تایی

برای هر درخت k تایی با n گره، همواره داریم:

اتصالات خالی	اتصالات پر	کل اتصالات
$nk - (n-1) = n(k-1) + 1$	$n-1$	$nk$

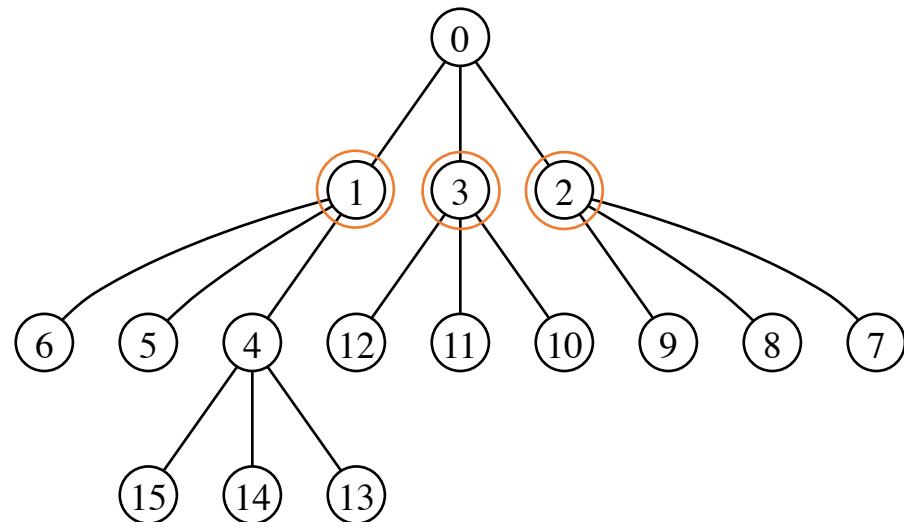
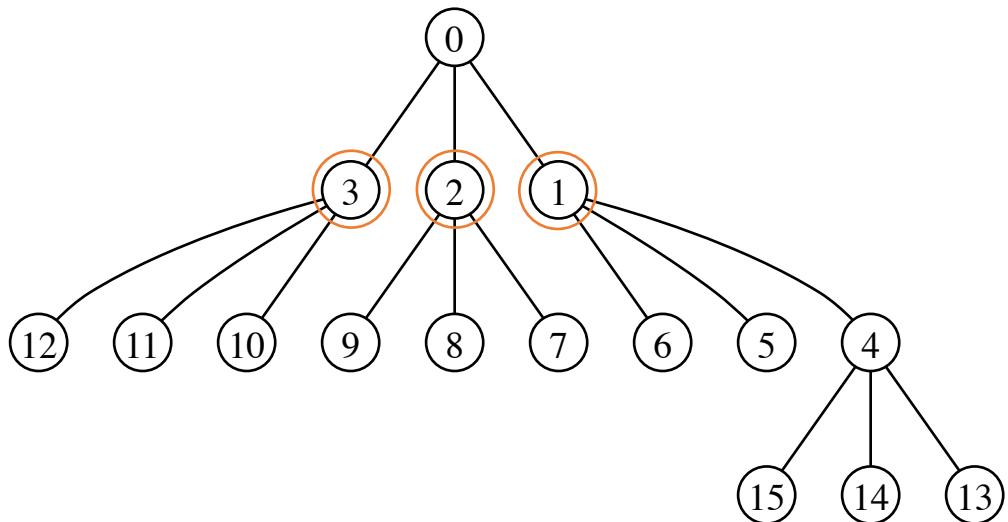


# درخت k-تایی

از آنجا که درخت ها دور ندارند، این برای هر درختی درست است.

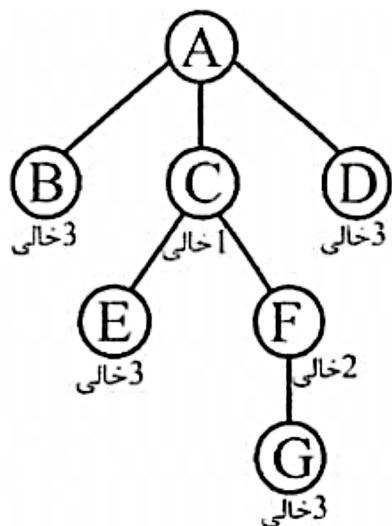
برای هر درخت k-تایی با n گره، همواره داریم:

اتصالات خالی	اتصالات پر	کل اتصالات
$nk - (n-1) = n(k-1) + 1$	$n-1$	$nk$



# مثال

برای یک درخت ۳تایی با ۷ گره، وضعیت اتصالات به این ترتیب است:



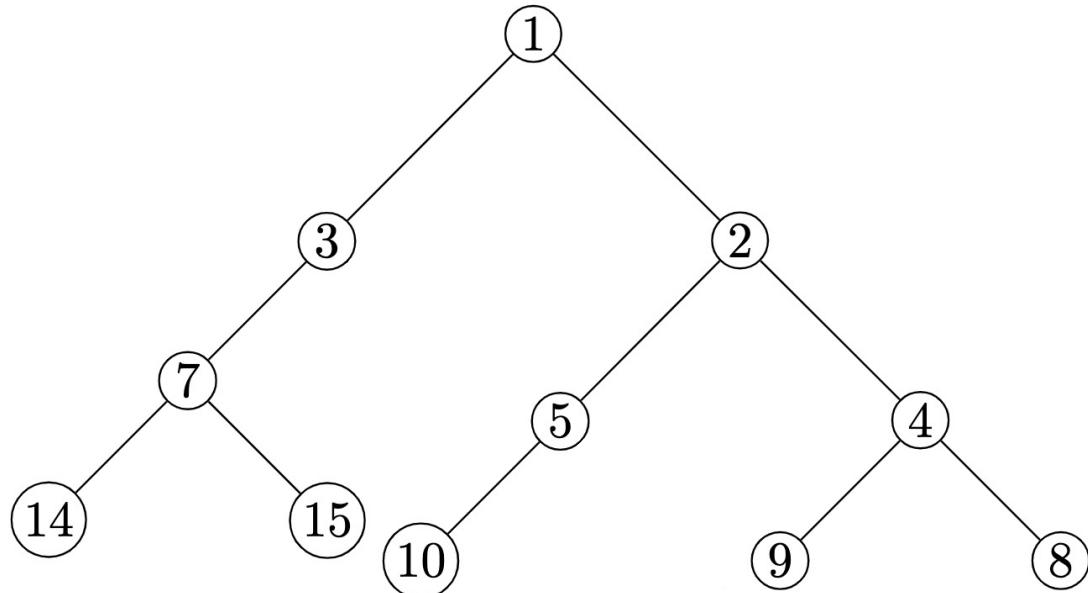
$$\text{کل اتصالات} = nk = 21$$

$$\text{اتصالات پر(یال‌ها)} = n - 1 = 6$$

$$(\text{null}) = \text{اتصالات خالی} = nk - (n - 1) = 15$$

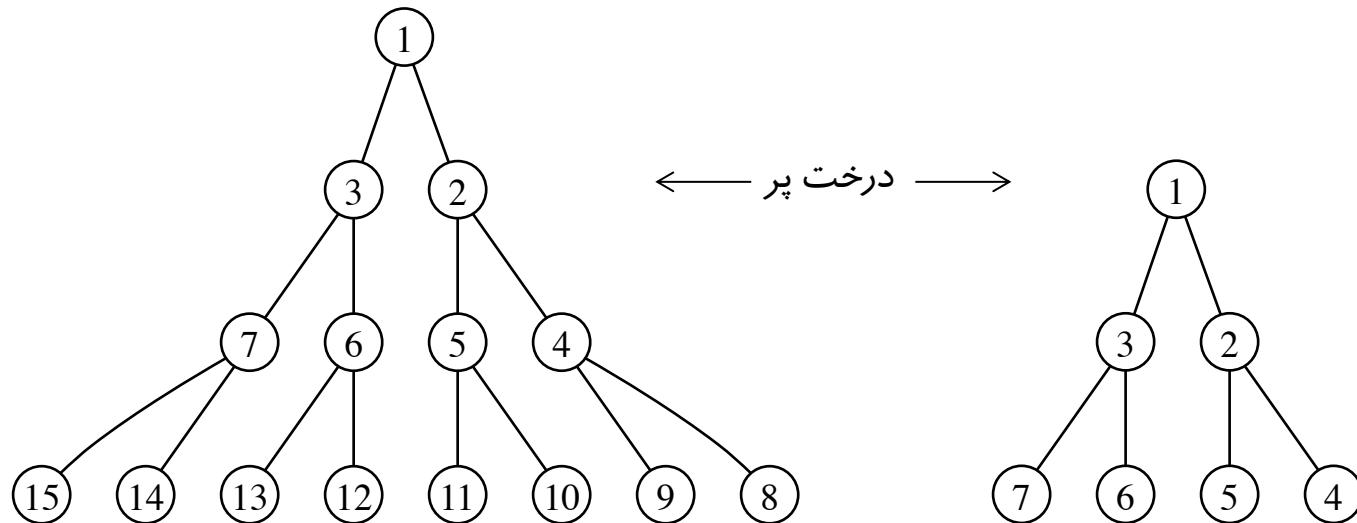
# درخت دودویی

یک حالت خاص از درختان کاتایی، که در آن  $k$  برابر با ۲ باشد.  
به دو زیر درخت هر راس، زیر درخت های چپ و راست آن گره اطلاق می‌شود.



# درخت دودویی

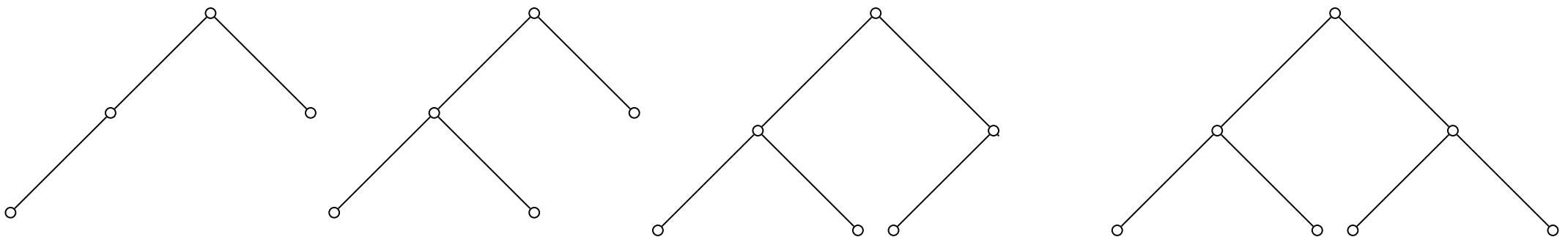
یک حالت خاص از درختان کاتایی، که در آن  $k$  برابر با ۲ باشد.  
به دو زیر درخت هر راس، زیر درخت های چپ و راست آن گره اطلاق می‌شود.



# درخت $k$ -تایی کامل

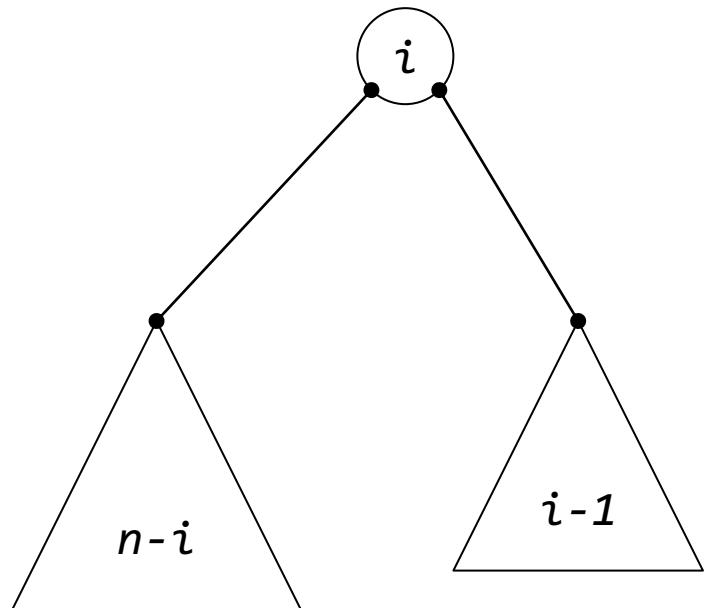
به درختی اطلاق می‌شود که به جز آخرین سطح، هر لایه بیشترین تعداد راس ممکن را داشته باشد؛ در سطح آخر نیز، از چپ شروع به پر کردن راس‌ها می‌کنیم.

مثالاً برای  $k=2$  داریم:



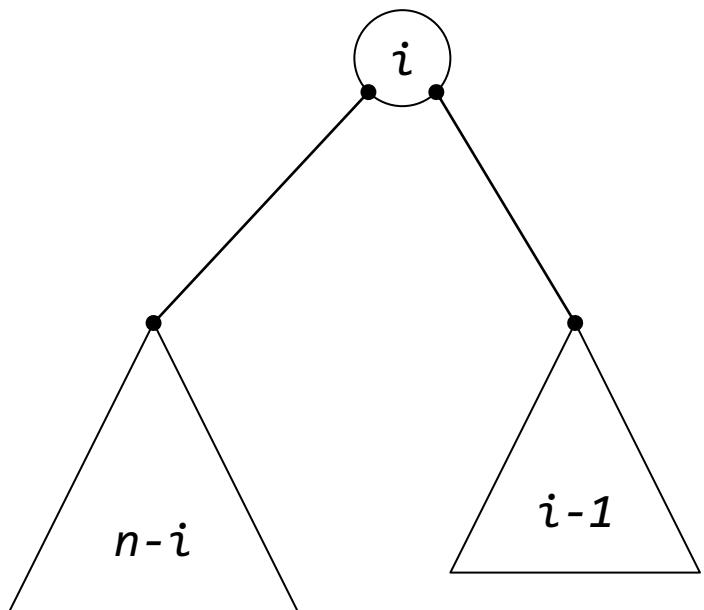
# تعداد درختان دودویی

تعداد درختان دودویی با  $n$  گره را بدست آورید.



# تعداد درختان دودویی

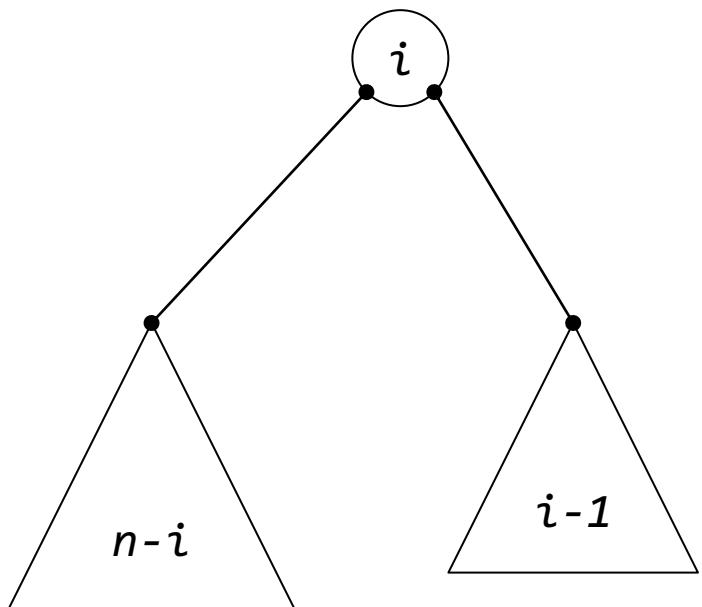
تعداد درختان دودویی با  $n$  گره را بدست آورید.



$$T(n) = \begin{cases} T(0) = 1 \\ T(n) = \sum_{i=1}^n T(i-1) \times T(n-i) & n \geq 1 \end{cases}$$

# تعداد درختان دودویی

تعداد درختان دودویی با  $n$  گره را بدست آورید.



$$T(n) = \begin{cases} T(0) = 1 \\ T(n) = \sum_{i=1}^n T(i-1) \times T(n-i) & n \geq 1 \end{cases}$$

$$T(n) = \frac{1}{n+1} \binom{2n}{n}$$

اعداد کاتالان

تعداد درختان کاتایی با  $n$  گره؟

# درختان منظم

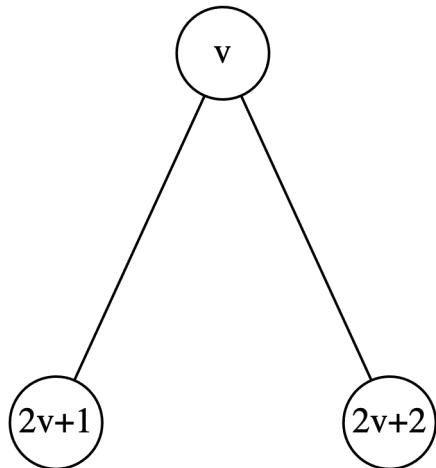
به علت منظم بودن درختان، بعضی محاسبه اطلاعات درخت با استفاده از مشخصاتش ساده‌تر است، مثلا:

- نگهداری و نمایش درختان به وسیله آرایه‌های یک بعدی
- محاسبه مشخصات ارتفاع، تعداد گره‌ها و تعداد فرزندان:
  - محاسبه پارامتر  $k$  با استفاده از  $h$  و  $n$
  - محاسبه پارامتر  $h$  با استفاده از  $k$  و  $n$
  - محاسبه پارامتر  $n$  با استفاده از  $h$  و  $k$

⋮

# نمایش آرایه‌ای درختان منظم

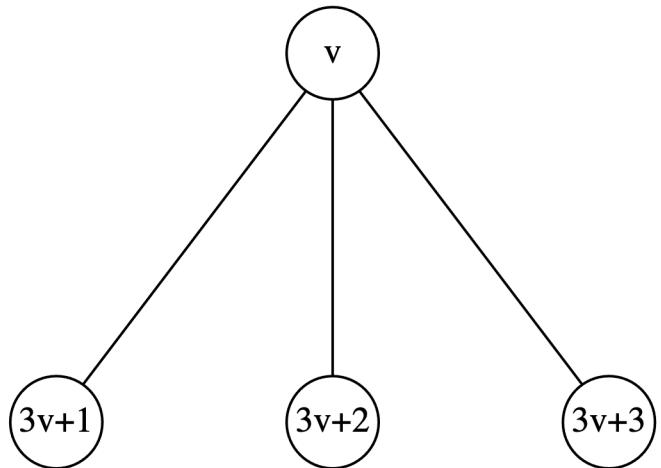
در این نمایش برای یک درخت  $k$ -تایی با  $n$  گره به ترتیب زیر عمل می‌کنیم:



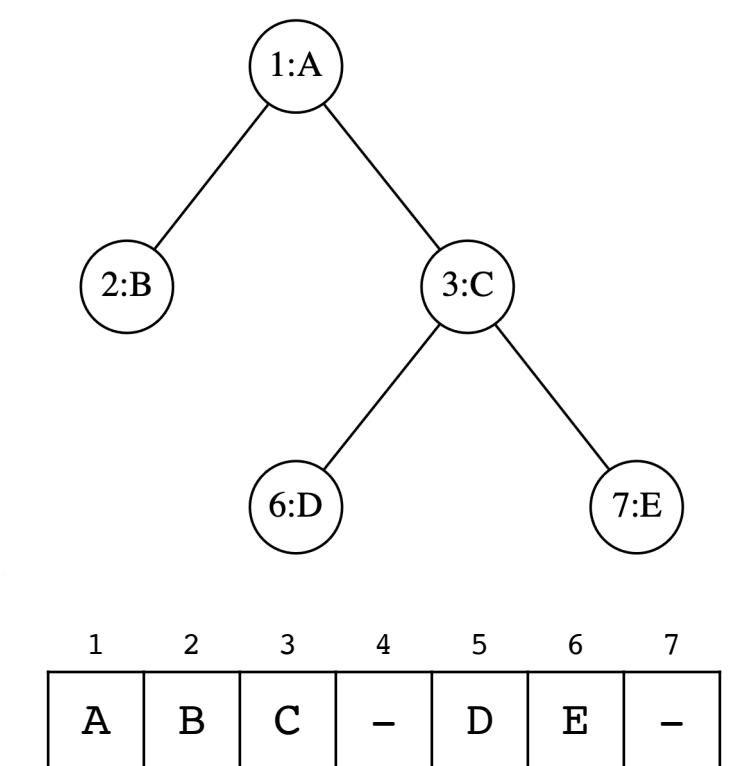
- به گره ریشه شماره (اندیس) 0 را می‌دهیم.

• در بقیه گره‌های درخت، برای هر گره با شماره  $v$ :

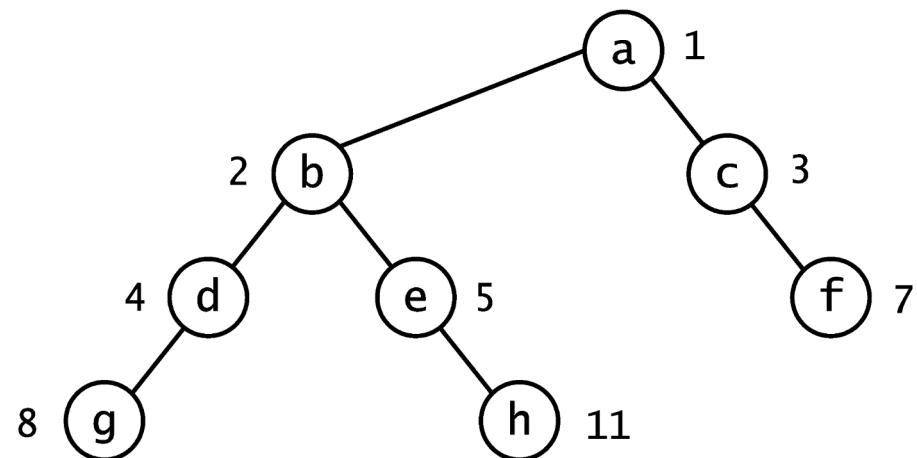
- به فرزند  $i$  ام این راس در صورت وجود شماره  $kv+i$  را اختصاص می‌دهیم.



# مثال



# مثال



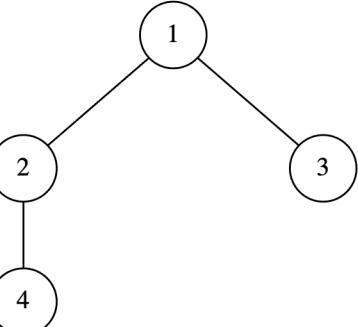
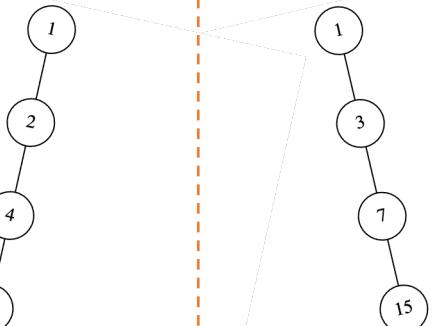
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a	b	c	d	e	-	f	g	-	-	h	-	-	-	-

# نمایش ترتیبی و تعداد خانه مورد نیاز

مثال(درخت دودویی)	حافظه مورد نیاز برای نمایش ترتیبی	درخت
<pre> graph TD     1((1)) --- 2((2))     1 --- 3((3))     2 --- 4((4))   </pre>	$n$	پر و کامل
<pre> graph TD     subgraph LeftTree [ ]         1L((1)) --- 2L((2))         1L --- 3L((3))         2L --- 4L((4))         2L --- 5L((5))         3L --- 6L((6))         3L --- 7L((7))         4L --- 8L((8))     end     subgraph RightTree [ ]         1R((1)) --- 2R((2))         1R --- 3R((3))         2R --- 4R((4))         2R --- 5R((5))         3R --- 6R((6))         3R --- 7R((7))         4R --- 8R((8))         4R --- 9R((9))         5R --- 10R((10))         5R --- 11R((11))         6R --- 12R((12))         6R --- 13R((13))         7R --- 14R((14))         7R --- 15R((15))     end     style LeftTree fill:none,stroke:none     style RightTree fill:none,stroke:none   </pre>	$\frac{1-k^h}{1-k}$	اریب

در واقع حداقل تعداد گره در یک درخت  $k$ -تایی مجموع یک سری هندسی متناهی است، که در درختان اریب  $O(n^h)$ .

# نمایش ترتیبی و تعداد خانه مورد نیاز

مثال(درخت دودویی)	حافظه مورد نیاز برای نمایش ترتیبی	درخت
	$n$	پر و کامل
	$\frac{1-k^h}{1-k}$	اریب

# نمایش ترتیبی و تعداد خانه مورد نیاز

مثال(درخت دودویی)	حافظه مورد نیاز برای نمایش ترتیبی	درخت
	$n$	پر و کامل
	$\frac{1-k^h}{1-k}$	اریب

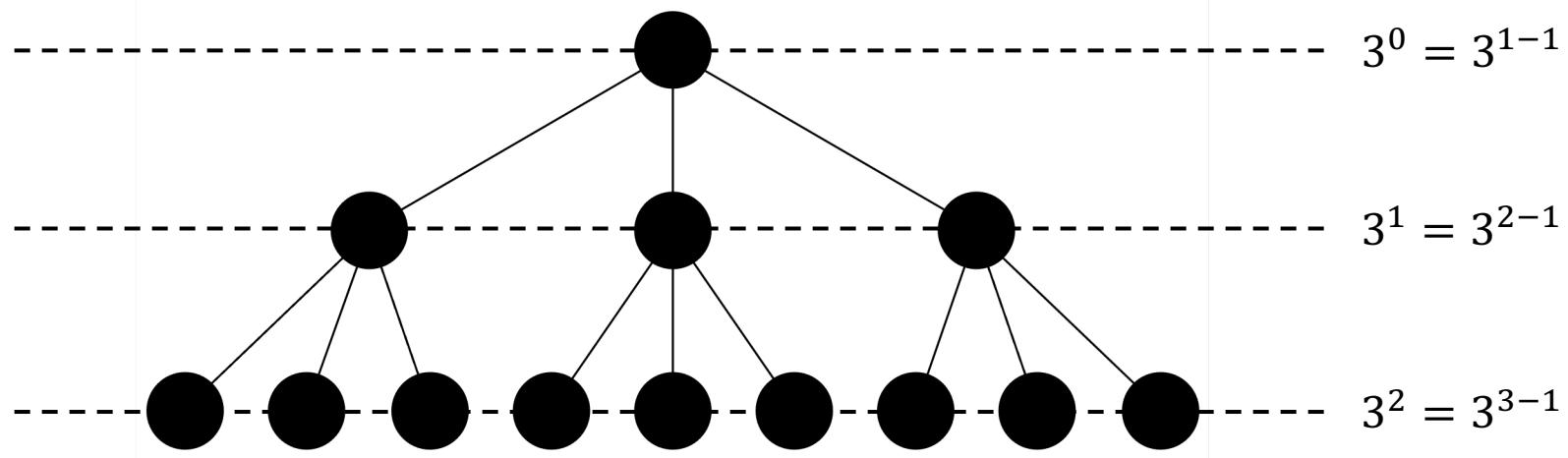
مثلما

در واقع حداقل تعداد گره در یک درخت kتایی مربوط به وقتی است که کمینه راس را در هر لایه از درخت داشته باشیم، آنگاه این مقدار مجموع یک سری هندسی است؛ که در درختان اریب  $O(n^h) = O(2^n)$  گره نیاز داریم!

# تعداد گره های سطح iام درخت kتایی

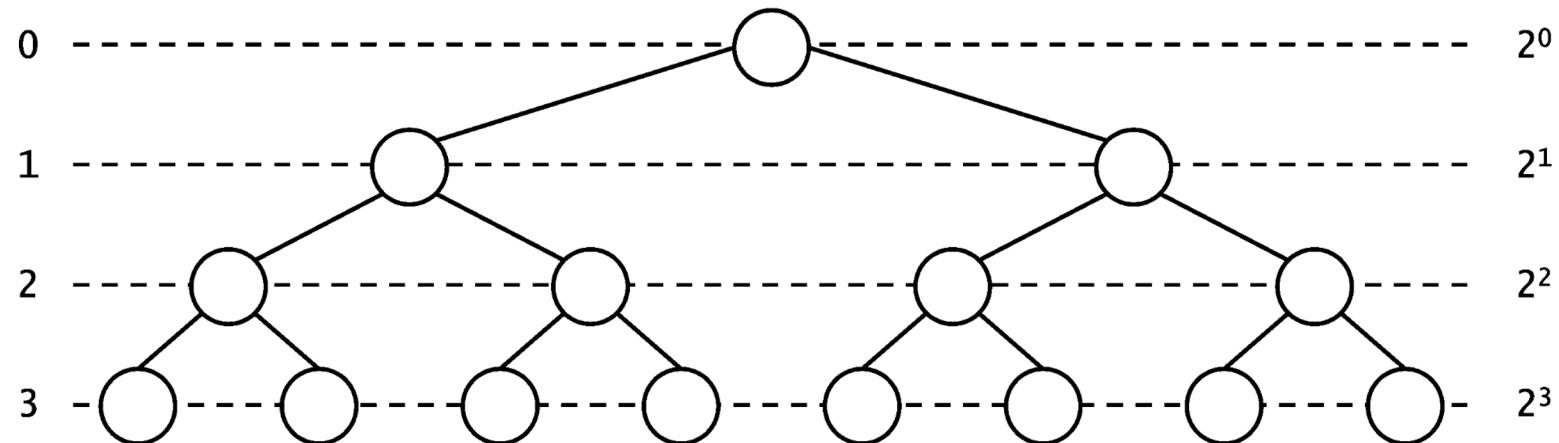
اگر راس ریشه در یک درخت kتایی در سطح 1 قرار داشته باشد، آنگاه تعداد گره های سطح iام درخت kتایی حداکثر برابر با  $k^{i-1}$  است؛ درنتیجه:

- کران پایین(حداقل) برای تعداد گره ها در سطح iام درخت kتایی دلخواه برابر با 1 است.
- کران بالا (حداکثر) برای تعداد گره ها در سطح iام درخت kتایی دلخواه برابر با  $k^{i-1}$  است.



# مثال

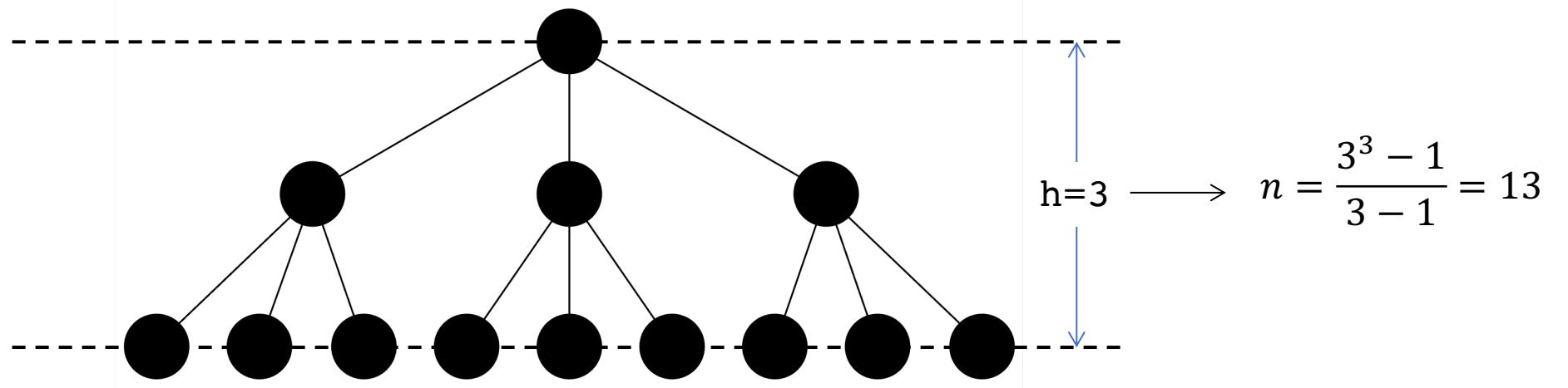
برای محاسبه تعداد گره های سطح  $i$ ام درخت دودویی، کافیست تا در رابطه  $k^{i-1}$  مقدار متغیر  $k$  را برابر با ۲ در نظر بگیریم.



# تعداد گره‌های درختان k-تایی با ارتفاع h

تعداد گره‌های درخت k-تایی پر با ارتفاع h برابر است با:

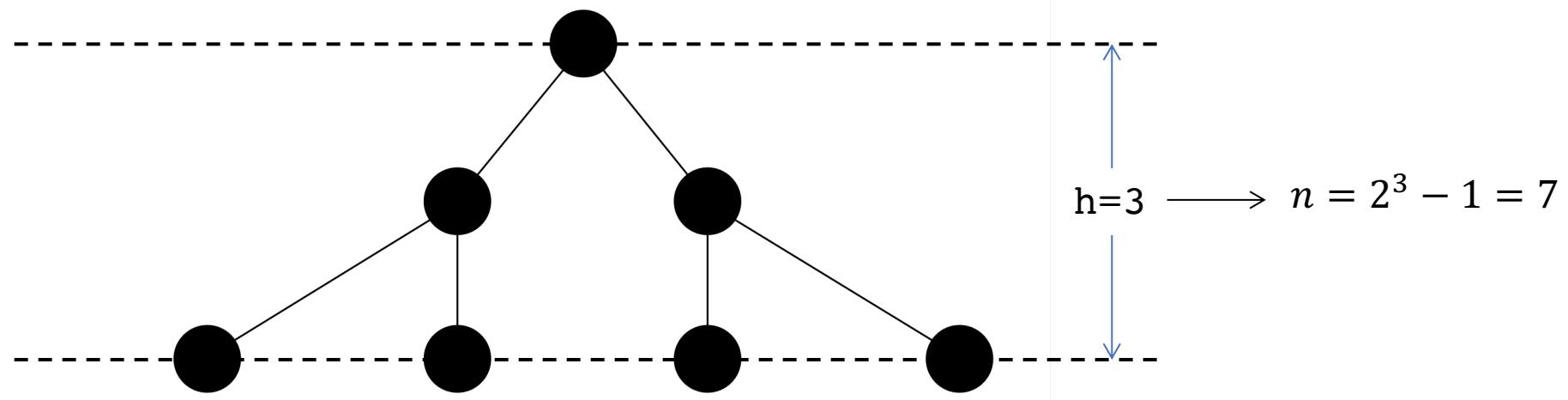
$$n \leq \sum_{i=1}^h k^{i-1} = \frac{k^h - 1}{k - 1}$$



# مثال

تعداد گره‌های درخت دودویی، پر با ارتفاع  $h$  برابر است با:

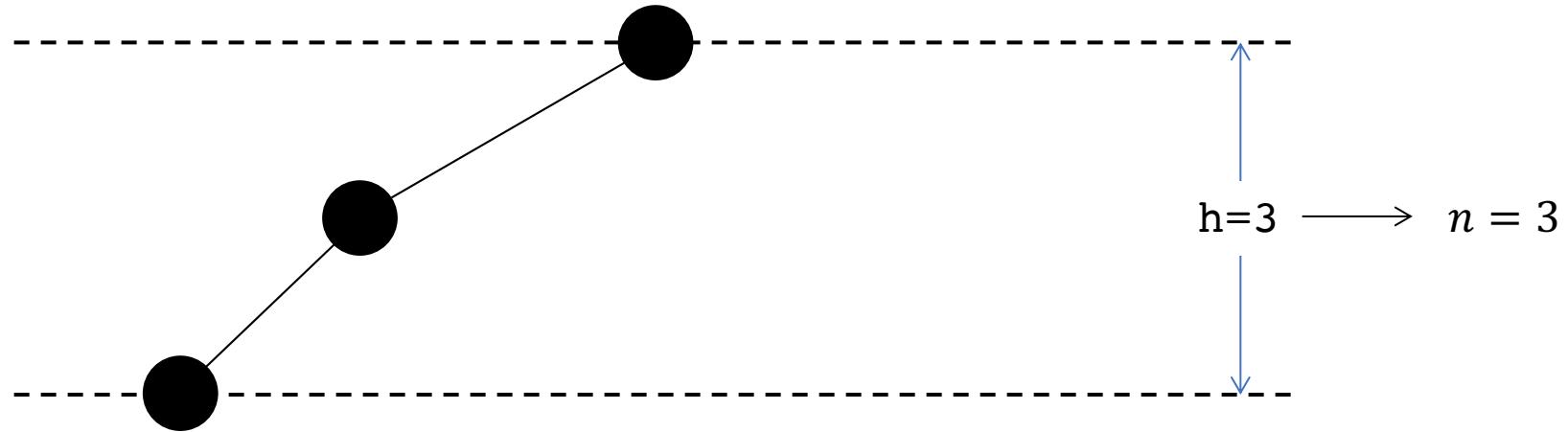
$$n \leq \sum_{i=1}^h 2^{i-1} = \frac{2^h - 1}{2 - 1} = 2^h - 1$$



# تعداد گره‌های درختان k-تایی با ارتفاع h

حداکثر تعداد گره نیز در هر درخت k-تایی با ارتفاع h، در حالتی است که درخت اریب ساده باشد:

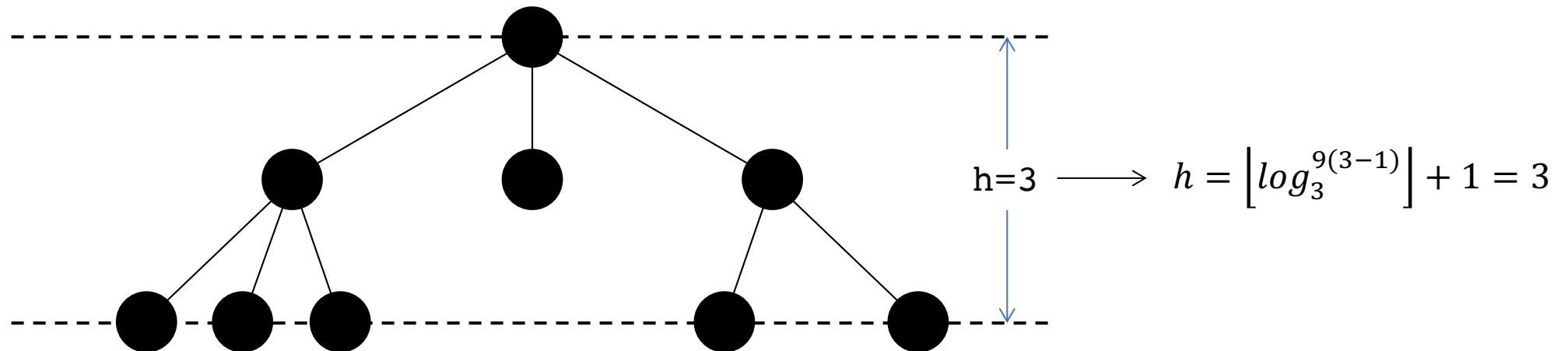
$$h \leq n \leq \frac{k^h - 1}{k - 1}$$



# ارتفاع درخت k-تایی با n گره

ارتفاع( $h$ ) در هر درخت k-تایی با n گره، همواره در محدوده زیر قرار دارد:

$$\left\lfloor \log_k^{n(k-1)} \right\rfloor + 1 \leq h \leq n$$



# ارتفاع درخت k تایی با n گره

ارتفاع( $h$ ) در هر درخت k تایی با n گره، همواره در محدوده زیر قرار دارد:

$$\left\lfloor \log_k^{n(k-1)} \right\rfloor + 1 \leq h \leq n$$

$$n = \frac{k^h - 1}{k - 1} \implies n(k - 1) = k^h - 1$$

$$\implies k^h = n(k - 1) + 1$$

$$\implies \log_k k^h = \log_k n(k - 1) + 1$$

$$\implies h = \left\lfloor \log_k n(k - 1) \right\rfloor + 1$$

اثبات: برای کران بالا رابطه گفته شده،

به سادگی درختان اریب را در نظر بگیرید؛

اما برای کران پایین داریم:

# تعداد برگ‌ها ( $n_0$ )

در صورتی که  $T$  یک درخت  $k$ -تایی با  $n$  گره باشد و  $n_i$  نشان دهنده تعداد رئوس درجه  $i$  باشد ( برای  $\{0,1,2,\dots\}$  همواره داریم):

$$n = n_0 + n_1 + n_2 + \cdots + n_k$$

بنابراین برای محاسبه تعداد برگ‌ها می‌توانیم از رابطه زیر استفاده کنیم:

$$n_0 = (k - 1)n_k + (k - 2)n_{k-1} + (k - 3)n_{k-2} + \cdots + 2n_3 + n_2 + 1$$

# تعداد برگ‌ها ( $n_0$ )

در صورتی که  $T$  یک درخت  $k$ -تایی با  $n$  گره باشد و  $n_i$  نشان دهنده تعداد رئوس درجه  $i$  باشد (برای  $\{0, 1, 2, \dots\}$ .)

تعداد برگ‌ها	درخت $k$ -تایی	همواره داریم
$n_2 + 1$	دودویی	
$2n_3 + n_2 + 1$	۳-تایی	
$3n_4 + 2n_3 + n_2 + 1$	۴-تایی	بنابراین برای

$$n_0 = (k - 1)n_k + (k - 2)n_{k-1} + (k - 3)n_{k-2} + \cdots + 2n_3 + n_2 + 1$$

# تعداد برگ‌ها ( $n_0$ )

در صورتی که  $T$  یک درخت  $k$ -تایی با  $n$  گره باشد و  $n_i$  نشان دهنده تعداد رئوس درجه  $i$  باشد ( برای  $\{0,1,2,\dots\}$  )، همواره داریم:

همان‌طور که دیده می‌شود تعداد گره‌های برگ‌های یک درخت  $k$ -تایی همواره مستقل از تعداد گره‌های تک فرزندی  $n_1$  می‌باشد و در نتیجه افزایش یا کاهش تعداد گره‌های تک فرزندی  $n_1$  تاثیری در تعداد برگ‌ها نداشته و فقط عمق آن‌ها را تغییر می‌دهد.

بنابراین برای محاسبه تعداد برگ،

$$n_0 = (k - 1)n_k + (k - 2)n_{k-1} + (k - 3)n_{k-2} + \cdots + 2n_3 + n_2 + 1$$

# مثال

در یک درخت ۴-تایی، ۲ گره ۴ فرزندی، ۱ گره ۳ فرزندی و ۵ گره ۲ فرزندی وجود دارد. تعداد برگ‌های (گره‌های صفر فرزندی) کدام است؟

۱۰ (1)

۱۳ (2)

۱۴ (3)

هیچ‌کدام (4)

# مثال

در یک درخت ۴-تایی، ۲ گره ۴ فرزندی، ۱ گره ۳ فرزندی و ۵ گره ۲ فرزندی وجود دارد. تعداد برگ‌های (گره‌های صفر فرزندی) کدام است؟

۱۰ (۱)

۱۳ (۲)

۱۴ (۳)

$$\begin{aligned}n_0 &= 3n_4 + 2n_3 + n_2 + 1 \\&= 3 \times 2 + 2 \times 1 + 5 + 1 \\&= 14\end{aligned}$$

هیچ کدام (۴)

# پیمایش درخت

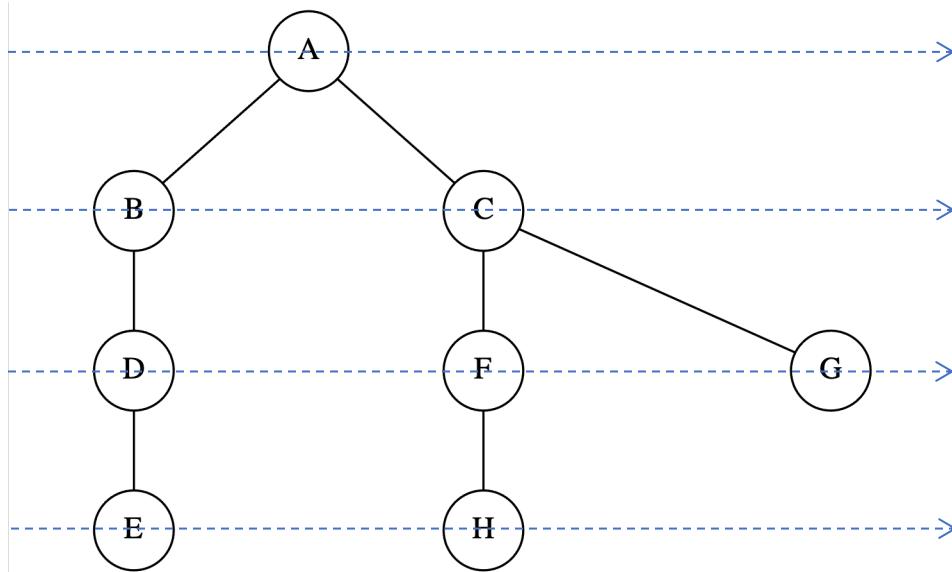
برای پیمایش گراف ها به طور عمومی دو روش سطح اول(BFS) و عمق اول(DFS) تعریف می شود که هر دو با مرتبه  $O(n + e)$  اجرا می شوند، از آنجا که  $|E(G)|$  برای درختان برابر است با  $n - 1$ ، پس این دو الگوریتم از مرتبه  $O(n)$  اجرا می شوند.

- پیمایش سطحی
- پیمایش عمقی
- پیش ترتیب
- میان ترتیب
- پس ترتیب

# پیمایش سطح اول

در پیمایش سطح ترتیب، پیمایش از ریشه آغاز شده و از سطح دوم به بعد در هر سطح، گره‌ها را از سمت چپ به راست رویت کرده و به سطح بعدی می‌رویم تا این‌که سطح آخر نیز دیده شود؛ این پیمایش به کمک صف انجام می‌شود.

```
def BFS(v) :  
    Q = empty Queue  
    Q.push(v)  
    visited[v]=True  
    while(len(Q)!=0) :  
        v=Q.pop()  
        for u in adj[v] :  
            if(visited[u]==False) :  
                Q.push(u)  
                visited[u]=True  
    return
```



دنباله پیمایش

# پیمايش عمق اول

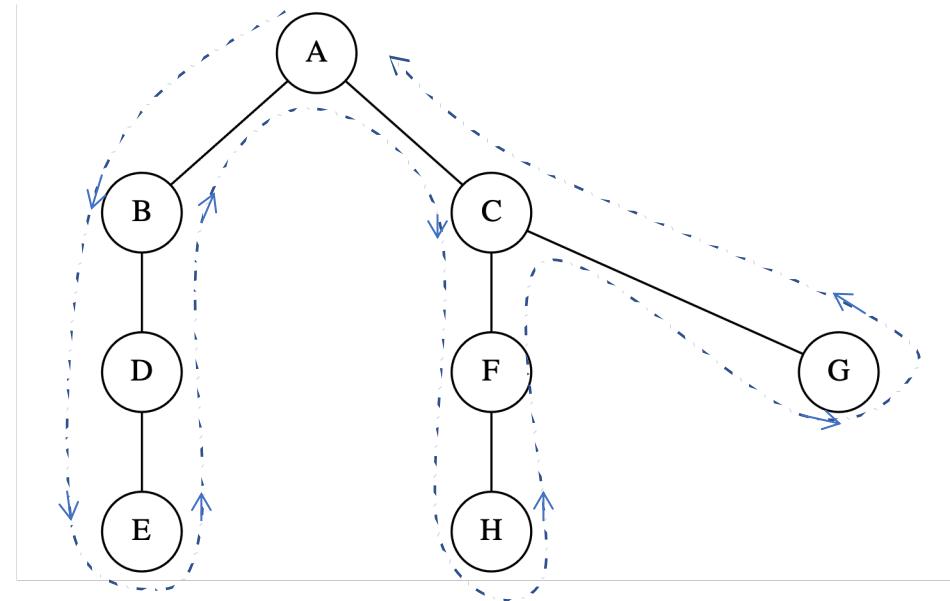
اين پیمايش از ریشه آغاز می شود و به ترتیب از بالا به پایین حرکت می کند، با رسیدن به هر راس ملاقات گره و یا رفتن به سمت گره های فرزند انجام می شود؛ در واقع با توجه به ترتیب انجام این دو عمل انواع مختلفی از پیمايش های عمق اول به وجود می آیند.

توضیحات	پیمايش
ابتدا گره را ملاقات می کنیم و سپس به سمت فرزندان می رویم.	پیش ترتیب (Pre Order)
ابتدا یکی (چپ ترین، در درختانی که ترتیب گره ها اهمیت دارد) از فرزندان ملاقات می شود، سپس خود گره و بعد از آن به سمت بقیه فرزندان می رویم.	میان ترتیب (In Order)
ابتدا تمامی فرزندان را ملاقات می کنیم، سپس خود گره را ملاقات می کنیم.	پس ترتیب (Post Order)

# پیمایش پیش ترتیب

برای هر گره، ابتدا برای آن گره را عملیات ملاقات را انجام می‌دهیم و سپس به سمت فرزندان می‌رویم.

```
def DFS(v) :  
    f(v) ← عمليات ملاقات  
    visited[v]=True  
    for u in adj[v] :  
        if(visited[u]==False) :  
            counter[v]+=1 ← تعداد فرزندان  
            ملاقات شده  
            رفتن به سمت فرزندان  
            DFS(u)
```



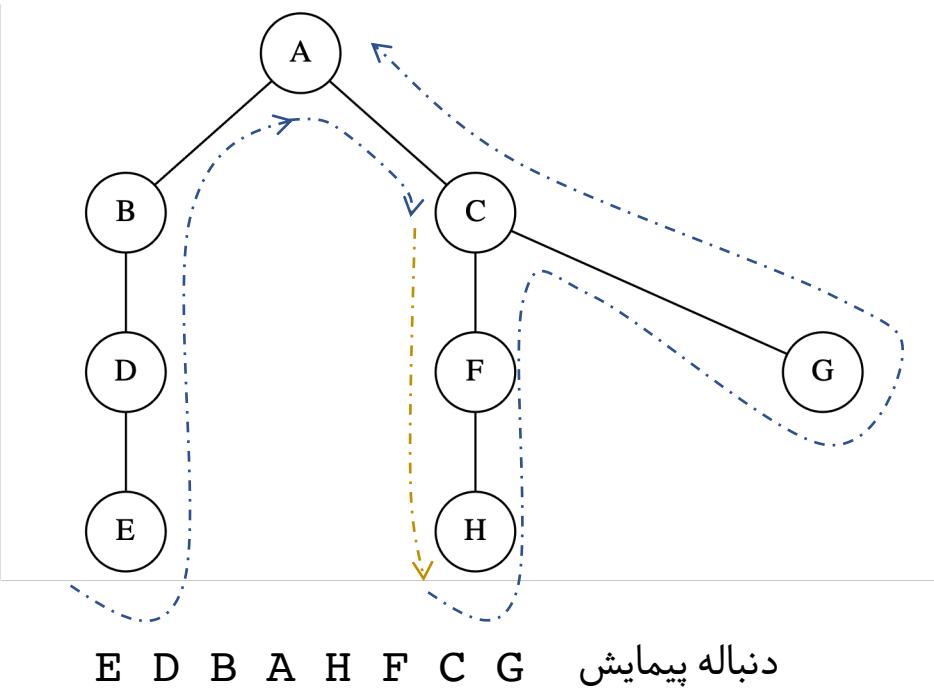
# پیمایش میان ترتیب

ابتدا اولین فرزندان ملاقات می‌شود، سپس خود گره و بعد از آن به سمت بقیه فرزندان می‌رویم.

```
def DFS(v) :  
    visited[v]=True  
    for u in adj[v] :  
        if(visited[u]==False) :  
            counter[v]+=1  
            DFS(u)  
        if(counter[v]==1) :  
            f(v)  
    if(len(adj[v])==0) :  
        f(v)
```

رفتن به سمت فرزندان

عملیات ملاقات

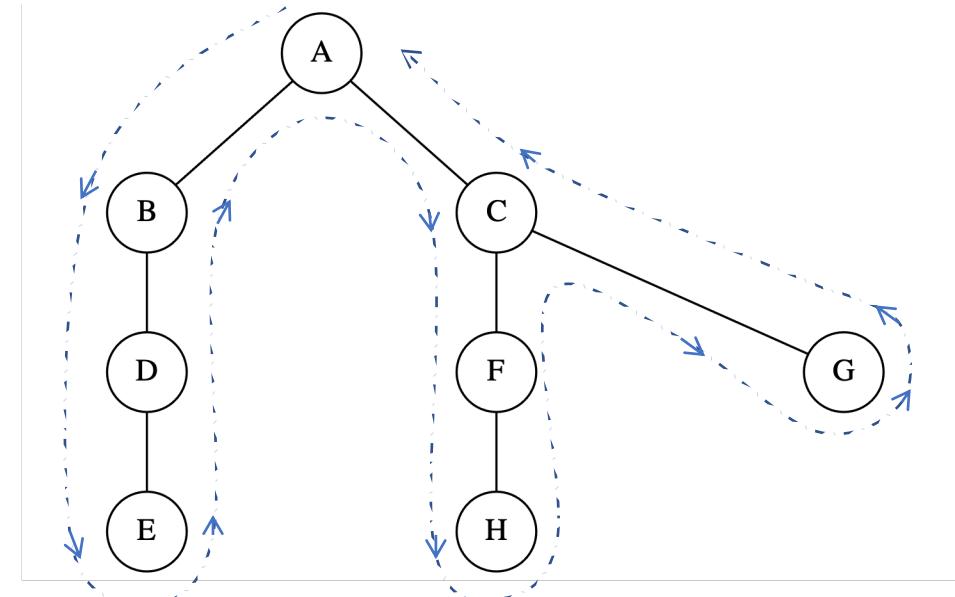


# پیمایش پس ترتیب

ابتدا تمامی فرزندان را ملاقات می‌کنیم، سپس خود گره را ملاقات می‌کنیم.

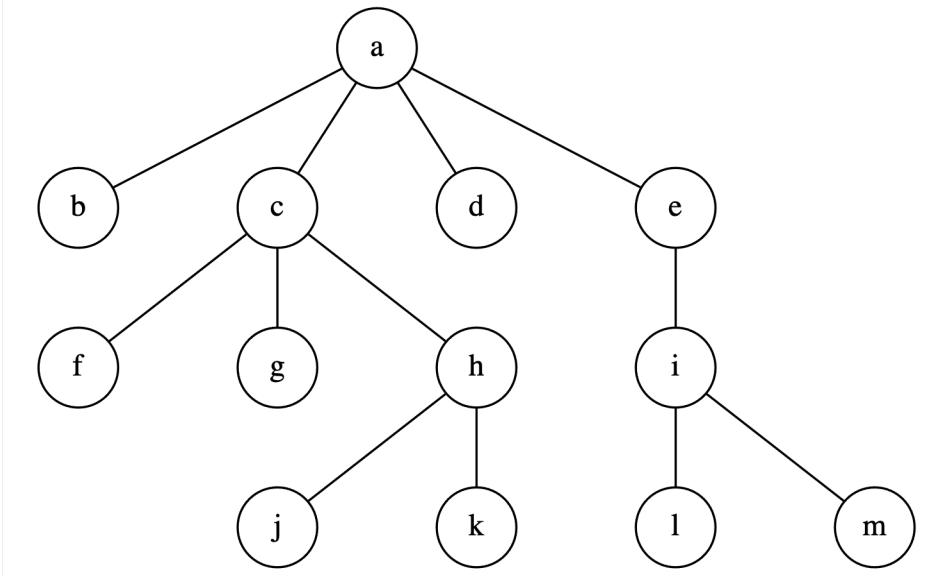
```
def DFS(v) :  
    visited[v]=True  
    for u in adj[v] :  
        if(visited[u]==False) :  
            counter[v]+=1  
            DFS(u)  
  
f(v) ←———— عملیات ملاقات
```

رفتن به سمت فرزندان



دنباله پیمایش E D B H F G C A

# مثال

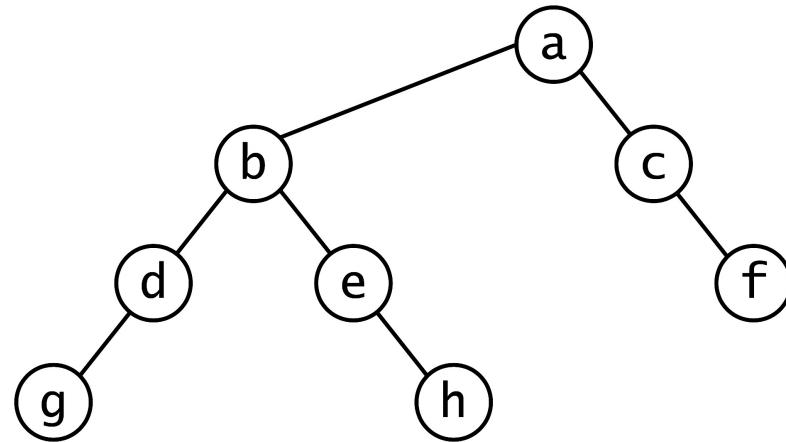


`preorder(T) : a, b, c, f, g, h, j, k, d, e, i, l, m`

`inorder(T) : b, a, f, c, g, j, h, k, d, l, i, m, e`

`postorder(T) : b, f, g, j, k, h, c, d, l, m, i, e, a`

# مثال



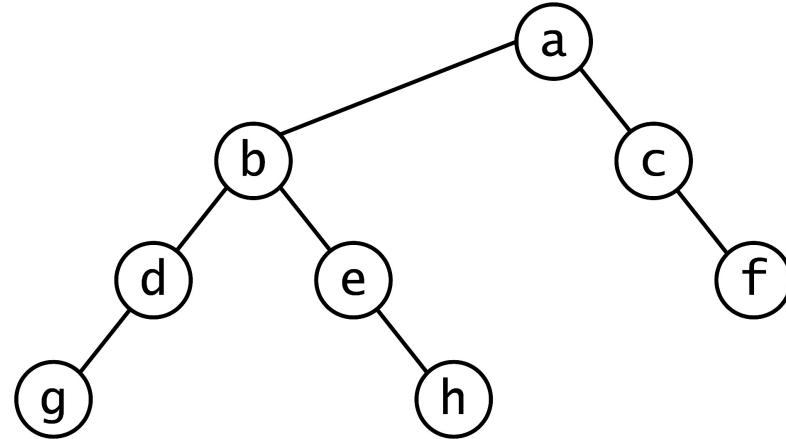
`preorder(T) : a, b, d, g, e, h, c, f`

`inorder(T) : g, d, b, e, h, a, c, f`

`postorder(T) : g, d, h, e, b, f, c, a`

# مثال

دقت کنید که ترتیب دیدن برگ‌ها در هر سه پیمایش یکسان است!



`preorder(T) : a, b, d, g, e, h, c, f`

`inorder(T) : g, d, b, e, h, a, c, f`

`postorder(T) : g, d, h, e, b, f, c, a`

# مسئله: رسم درخت دودویی واحد به کمک لیست پیمایش

برای ایجاد (ساختن) یک درخت دودویی منحصر به فرد به کمک پیمایش می‌توان از قواعد زیر استفاده کرد:  
الف) در حالت کلی برای رسیدن به درخت دودویی منحصر به فرد، داشتن یکی از جفت پیمایش‌های زیر الزامی است:

جفت پیمایش	توضیحات
Pre Order	(V :node, Left Child, Right Child)
In Order	(Left Child, V :node, Right Child)
Post Order	(Left Child, Right Child, V :node)
In Order	(Left Child, V :node, Right Child)
BFS	(Level Order)
In Order	(Left Child, V :node, Right Child)

# مسئله: رسم درخت دودویی واحد به کمک لیست پیمایش

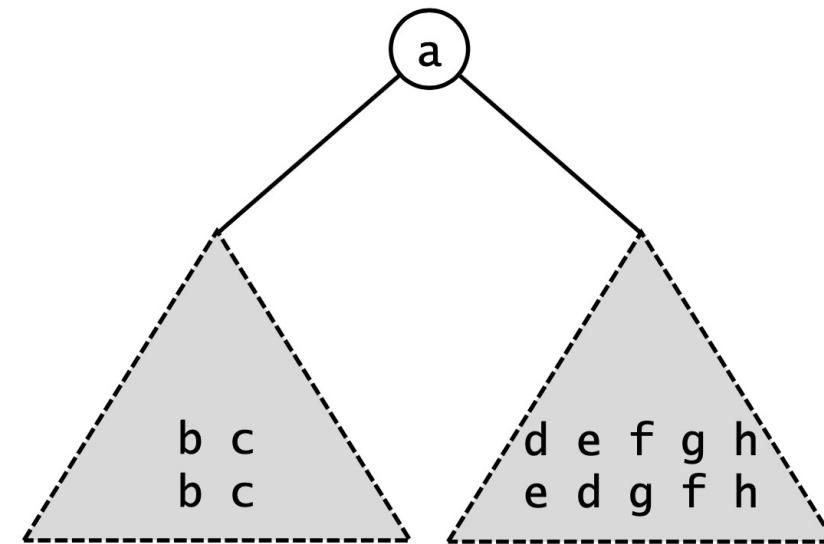
برای ایجاد (ساختن) یک درخت دودویی منحصر به فرد به کمک پیمایش می‌توان از قواعد زیر استفاده کرد:

ب) در هر کدام از جفت پیمایش‌ها؛ پیمایش اول ریشه‌ها را مشخص کرده و پیمایش دوم (In Order) وضعیت چپ و راست بودن فرزندان را تعیین می‌کند.

ج) برای ساختن درخت در هر یک از جفت پیمایش‌ها به وسیله پیمایش اول از سمتی که ریشه وجود دارد حرکت کرده و وضعیت چپ یا راست بودن برای هر گره را از پیمایش دوم مشخص می‌کنیم.

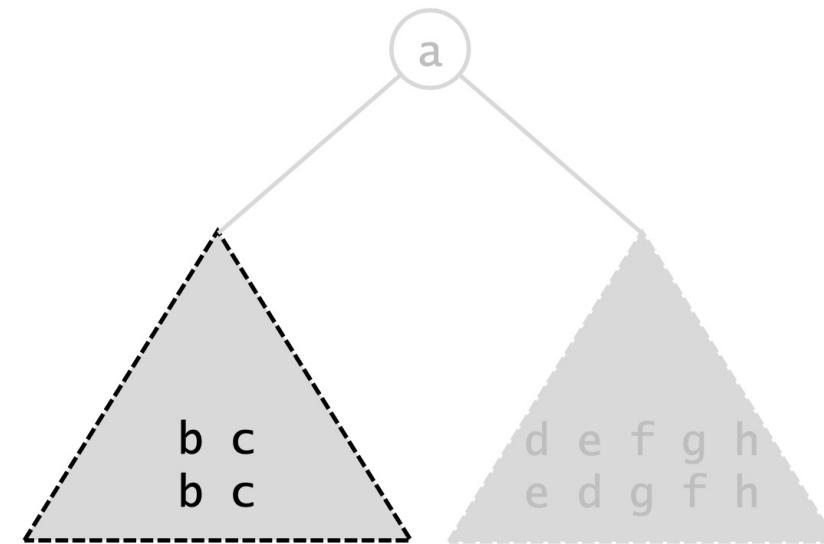
# رسم درخت دودویی واحد به کمک لیست پیمايش

pre	a	b	c	d	e	f	g	h
in	b	c	a	d	e	f	g	h



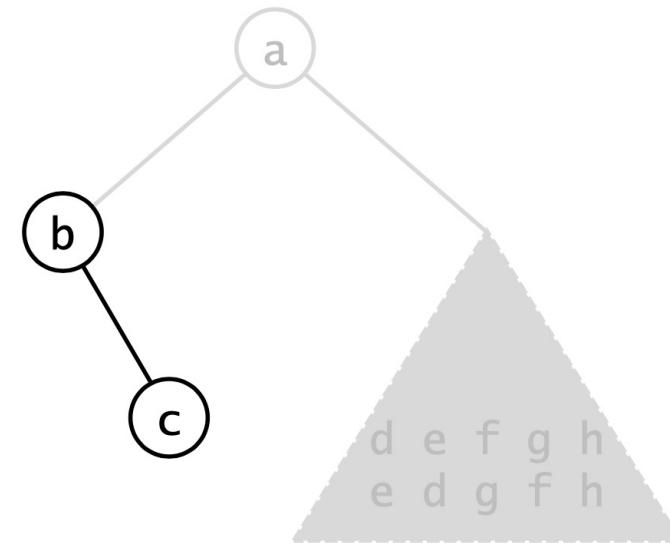
# رسم درخت دودویی واحد به کمک لیست پیمايش

pre	a	b	c	d	e	f	g	h
in	b	c	a	e	d	g	f	h



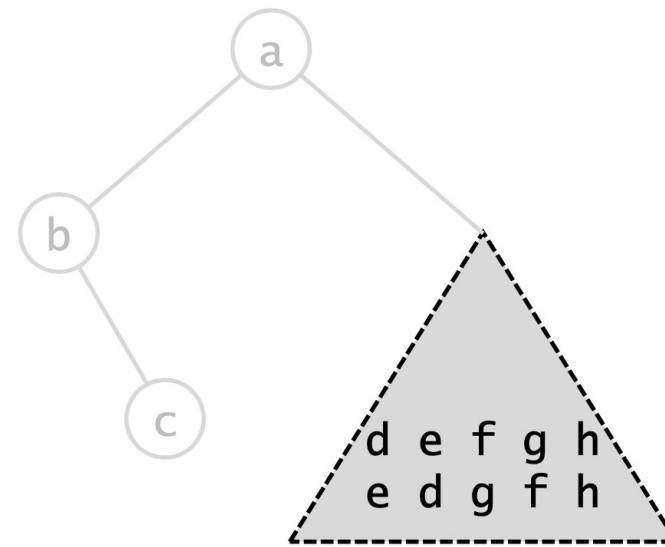
# رسم درخت دودویی واحد به کمک لیست پیمايش

pre	a	b	c	d	e	f	g	h
in	b	c	a	e	d	g	f	h



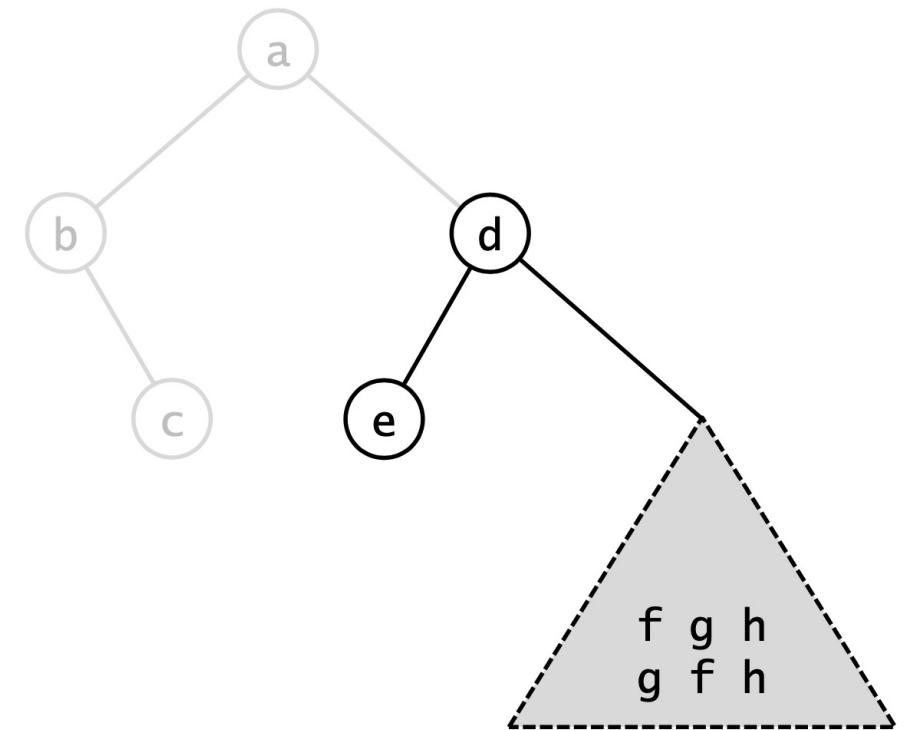
# رسم درخت دودویی واحد به کمک لیست پیمايش

pre	a	b	c	d	e	f	g	h
in	b	c	a	e	d	g	f	h



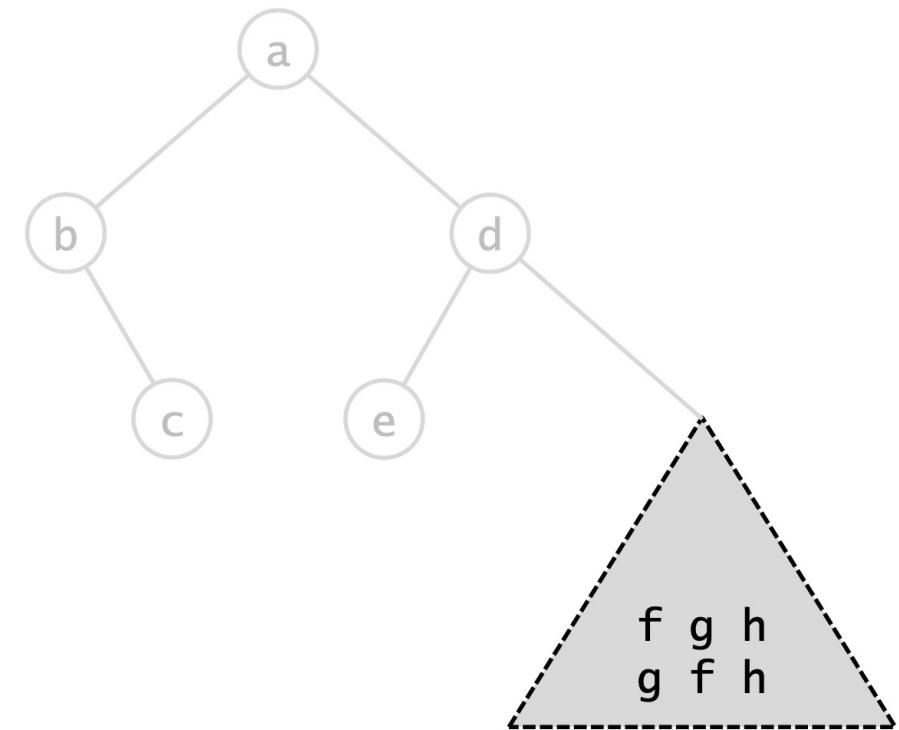
# رسم درخت دودویی واحد به کمک لیست پیمايش

pre	a	b	c	d	e	f	g	h
in	b	c	a	e	d	g	f	h



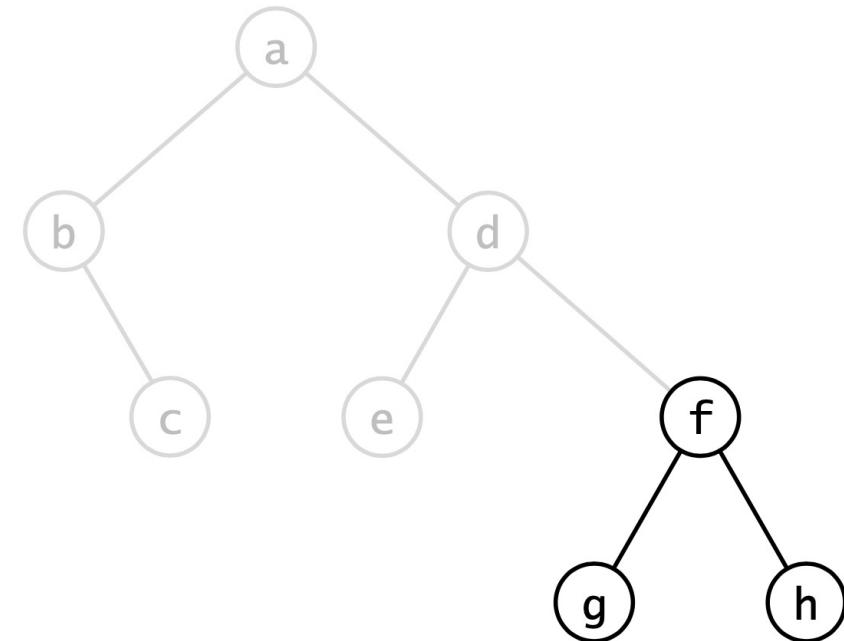
# رسم درخت دودویی واحد به کمک لیست پیمايش

pre	a	b	c	d	e	g	f	h
in	b	c	a	e	d	g	f	h



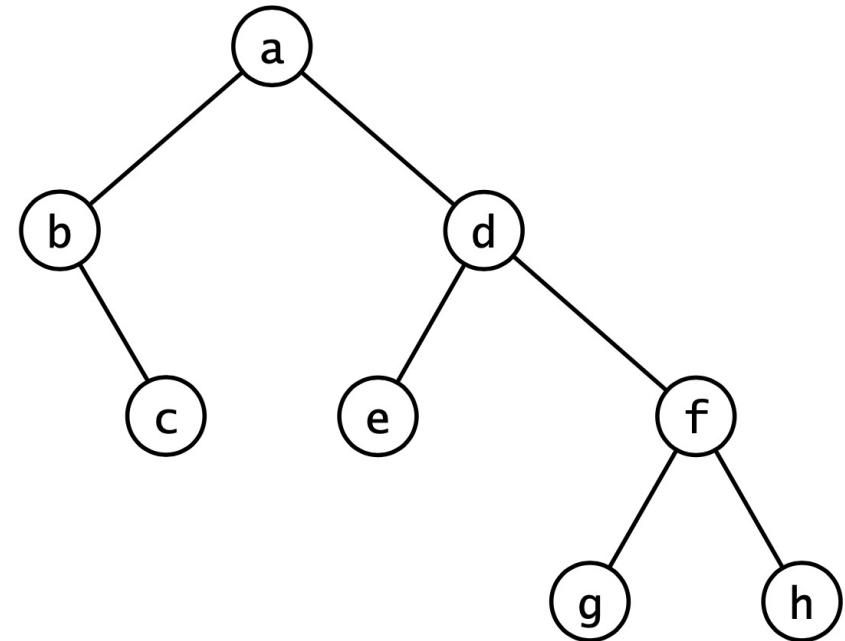
# رسم درخت دودویی واحد به کمک لیست پیمايش

pre	a	b	c	d	e	f	g	h
in	b	c	a	e	d	g	f	h



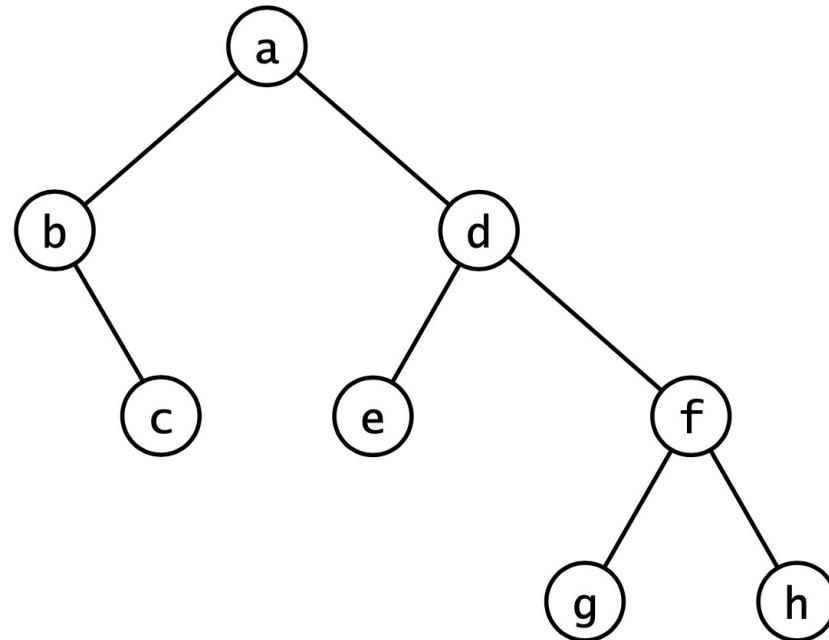
# رسم درخت دودویی واحد به کمک لیست پیمايش

pre	a	b	c	d	e	f	g	h
in	b	c	a	e	d	g	f	h
pos	c	b	e	g	h	f	d	a



# رسم درخت دودویی واحد به کمک لیست پیمايش

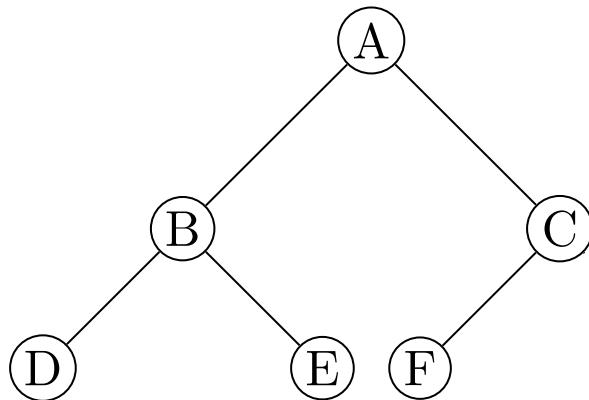
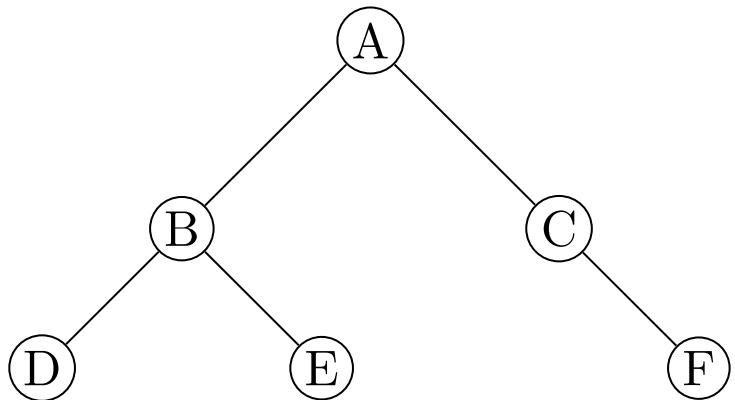
pre	a	b	c	d	e	f	g	h
in	b	c	a	e	d	g	f	h
pos	c	b	e	g	h	f	d	a



بنابراین با داشتن پیمايش Inorder و حداقل یکی از پیمايش های دیگر درخت دودویی قابل ترسیم است.  
اما با داشتن پیمايش Preorder و Postorder درخت دودویی به صورت یکتا قابل ترسیم نیست؛ چرا؟

# پیمایش PostOrder و PreOrder

همانطور که میبینید، پیمایش‌های گفته شده برای دو درخت زیر با هم یکسان است؛ در واقع گره چپ یا راست بودن، در این جا تاثیری در ترتیب پیمایش ایجاد نمی‌کند.



Pre Order	:	A B D E C F
Post Order	:	D E B F C A

# بدست آوردن گرهای تک فرزندی

از سمت چپ پیمایش PreOrder شروع می‌کنیم، هر ترتیب دوتایی کنار هم از گره‌ها را در نظر می‌گیریم و در پیمایش PostOrder بررسی می‌کنیم، اگر آن را به طور معکوس مشاهده کردیم، آنگاه گره سمت چپ در پیمایش PreOrder تک فرزندی است.

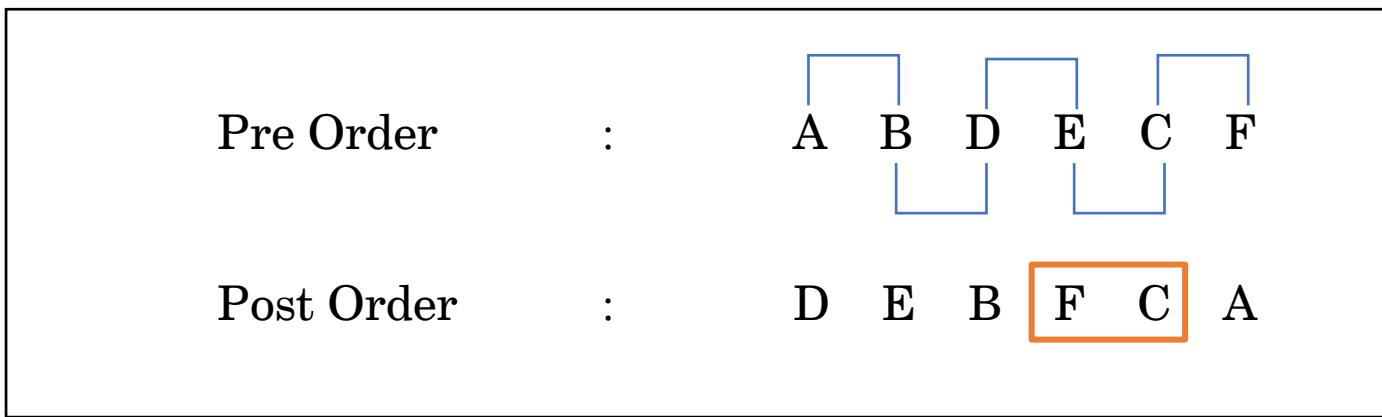
در ادامه می‌توانیم تعداد برگ‌ها و گره‌های دو فرزندی را بیابیم:

$$\begin{cases} n = n_2 + n_1 + n_2 \\ n_0 = n_2 + 1 \end{cases}$$

از آنجا که ترتیب ملاقات برگ‌ها در دو پیمایش یکسان است، لذا به راحتی مشخص می‌شوند؛ این یعنی گره‌های دو فرزندی، تک فرزندی و برگ‌ها در یک درخت دودوبی با داشتن پیمایش PreOrder و PostOrder قابل تشخیص هستند.

# بدست آوردن گره‌های تک فرزندی

از سمت چپ پیمایش PreOrder شروع می‌کنیم، هر ترتیب دوتایی کنار هم از گره‌ها را در نظر می‌گیریم و در پیمایش بررسی می‌کنیم، اگر آن را به طور معکوس مشاهده کردیم، آنگاه گره سمت چپ در پیمایش PreOrder تک فرزندی است.



از آنجا که ترتیب ملاقل بر پیمایش پیش از بررسی سریع‌تر است، بزرگی سریع‌تری در پیمایش داشتیم. این یعنی گره‌های دو فرزندی، تک فرزندی و برگ‌ها در یک درخت دودوبی با داشتن پیمایش PreOrder و PostOrder قابل تشخیص هستند.

# رسم درخت دودویی واحد به کمک لیست پیمایش

اگر درخت دودویی ما کامل یا پر باشد، از آنجا که شکل مربوط به درخت مشخص است، تنها با یک پیمایش دلخواه می‌توان به درخت دست‌یافت.

مثال. در صورتی که پیمایش میان‌وندی یک درخت کامل به صورت زیر باشد، پیمایش پیشوندی آن کدام است؟

Inorder: abcdfe

bacfed .1

dbafec .2

dbacef .3

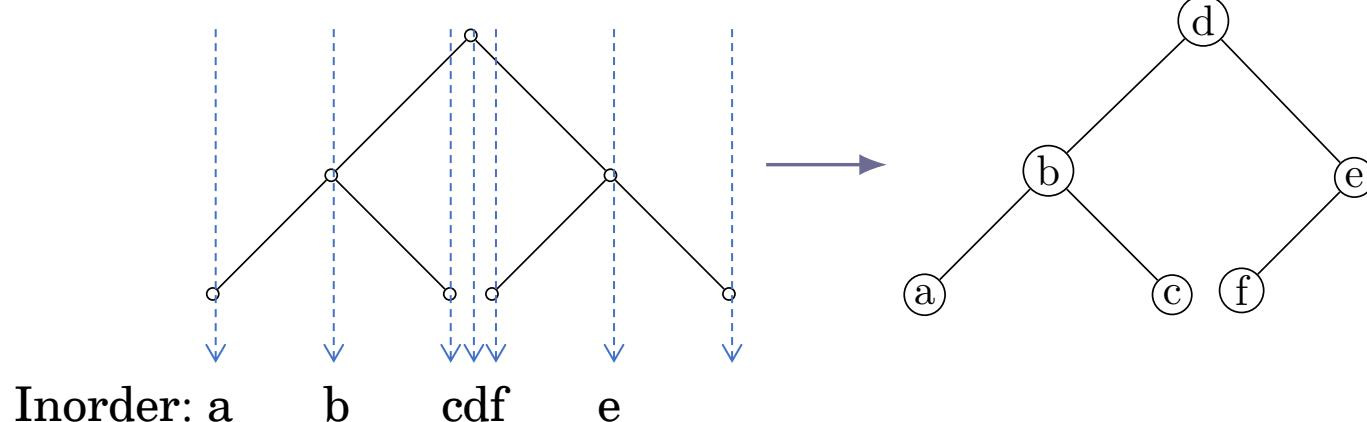
bacefd .4

# رسم درخت دودویی واحد به کمک لیست پیمایش

اگر درخت دودویی ما کامل یا پر باشد، از آنجا که شکل مربوط به درخت مشخص است، تنها با یک پیمایش دلخواه می‌توان به درخت دست‌یافت.

مثال. در صورتی که پیمایش میان‌وندی یک درخت کامل به صورت زیر باشد، پیمایش پیشوندی آن کدام است؟

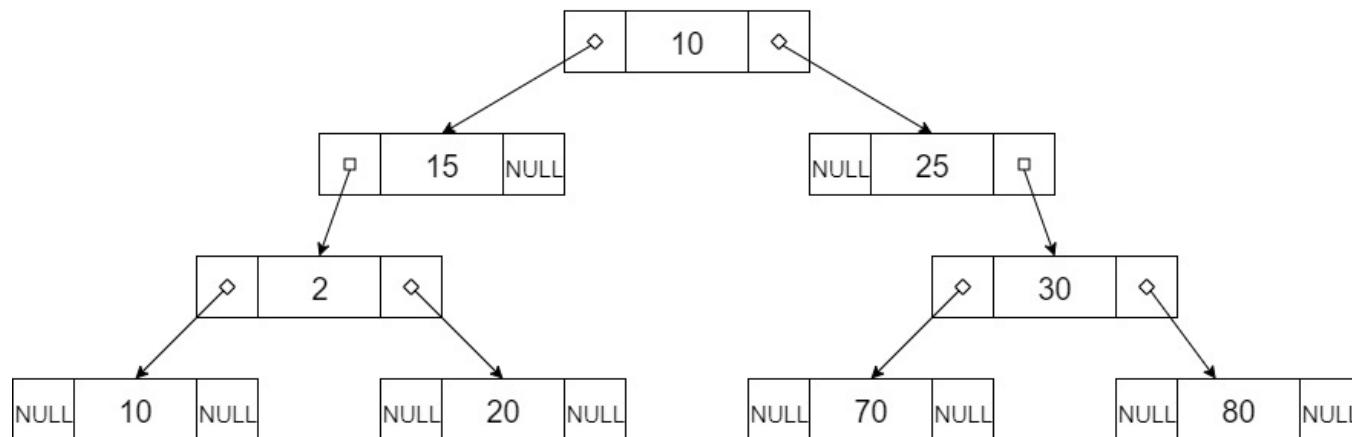
Inorder: abcdfe



bacfed	.1
dbafec	.2
dbacef	.3
bacefd	.4

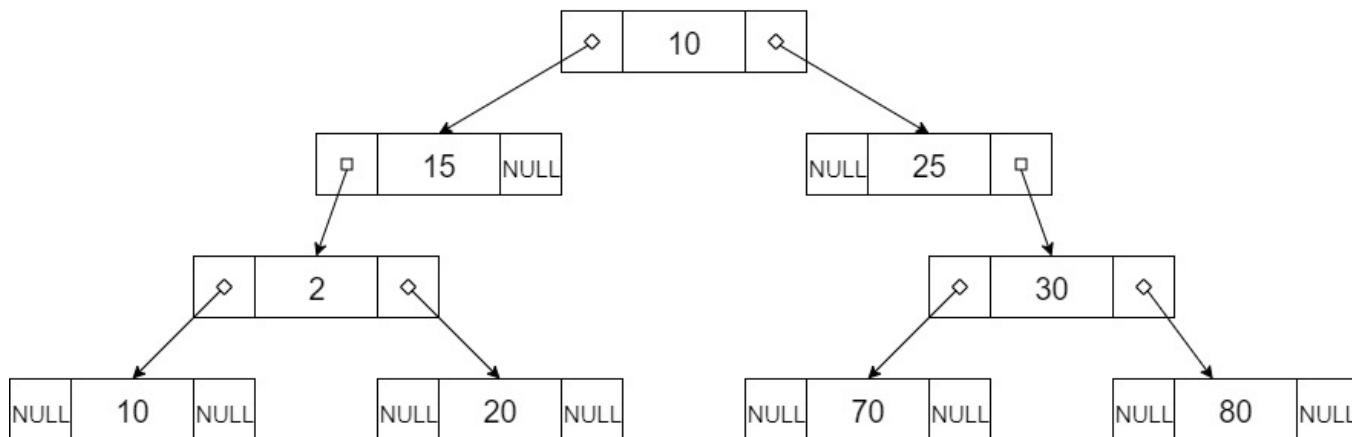
# اشاره‌گرها در درخت دودویی

Left Child	Data	Right Child
------------	------	-------------



# اشاره‌گرها در درخت دودویی

Left Child	Data	Right Child
------------	------	-------------



تعداد اشاره‌گرهای پر	تعداد اشاره‌گرهای خالی	تعداد کل اشاره‌گرهای
$n-1$	$n+1$	$2n$

در هر درخت دودویی، همواره:

# درخت نخی دودویی

## Binary Thread Tree

با استفاده از اشاره‌گرهای null می‌توان به عناصر قبلی یا بعدی در یک پیمایش خاص اشاره نمود و در نتیجه در هنگام پیمایش به نحو سریع‌تری درخت را پیمایش کرد. درختی که اشاره‌گرهای بدون استفاده آن بدین صورت مورد استفاده قرار گرفته است، درخت نخی یا نخ‌کشی شده نام دارد.

معمولًاً اشاره‌گر سمت چپ در صورتی که بلا استفاده باشد، جهت اشاره به عنصری استفاده می‌شود که در یک پیمایش مورد نظر قبل از این عنصر قرار دارد و اشاره‌گر سمت راست در صورت بدون استفاده بودن برای اشاره به عنصر بعدی در پیمایش مورد نظر مورد استفاده قرار می‌گیرد.

# درخت نخی دودویی

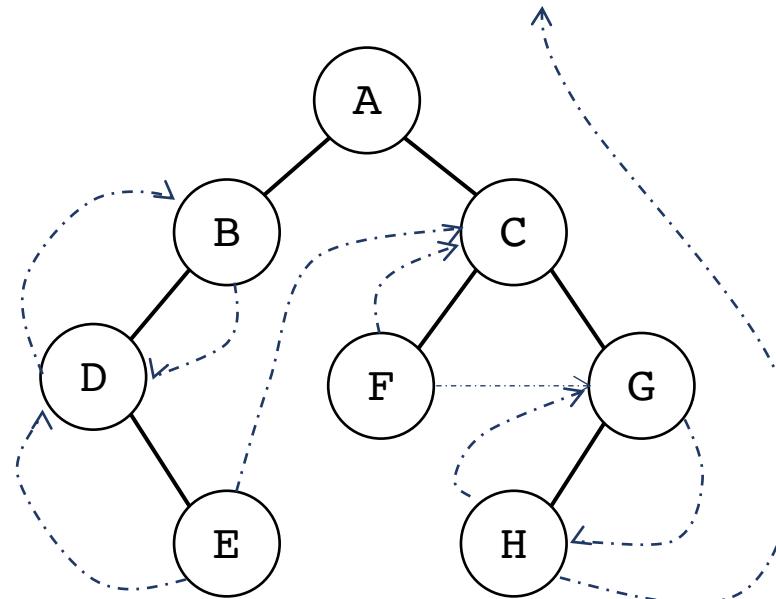
البته با وجود این مزیت، اکنون باید اشاره‌گرهای نخی از اشاره‌گرهای واقعی از اشاره‌گرهای نخی به نحوی متمایز شوند و بدین منظور باید در هر گره بخشی برای مشخص کردن نوع اشاره‌گر موجود باشد.

اگر  $Lflag=1$  باشد آنگاه Lchild یک اشاره‌گر نخی است و در غیر این صورت اشاره‌گر عادی به فرزند چپ است.

اگر  $Rflag=1$  باشد آنگاه Rchild یک اشاره‌گر نخی است و در غیر این صورت اشاره‌گر عادی به فرزند راست است.

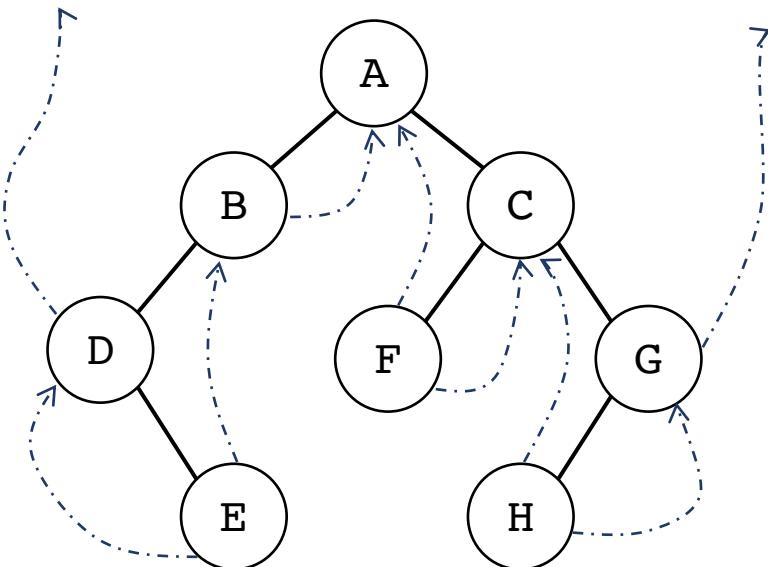
Lflag	Left Child	Data	Right Child	Rflag
-------	------------	------	-------------	-------

# درخت نخی پیمایش پیشوندی



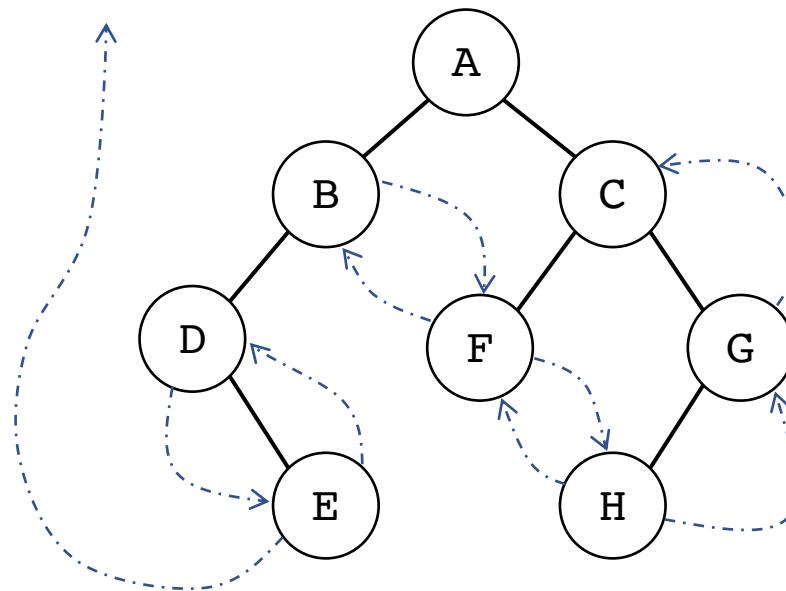
A B D E C F G H

# درخت نخی پیمایش میان و ندی



D E B A F C H G

# درخت نخی پیمایش پسوندی



E D B F H G C A



کاربرد ها

# درخت عبارت

- عبارت پرانتزگذاری شده.

$E := a \mid (\alpha E) \mid (E \beta E)$

$a := \text{variable}$

$\alpha := \text{unary operator} \quad [\sim, \log, \sin, \cos, \dots]$

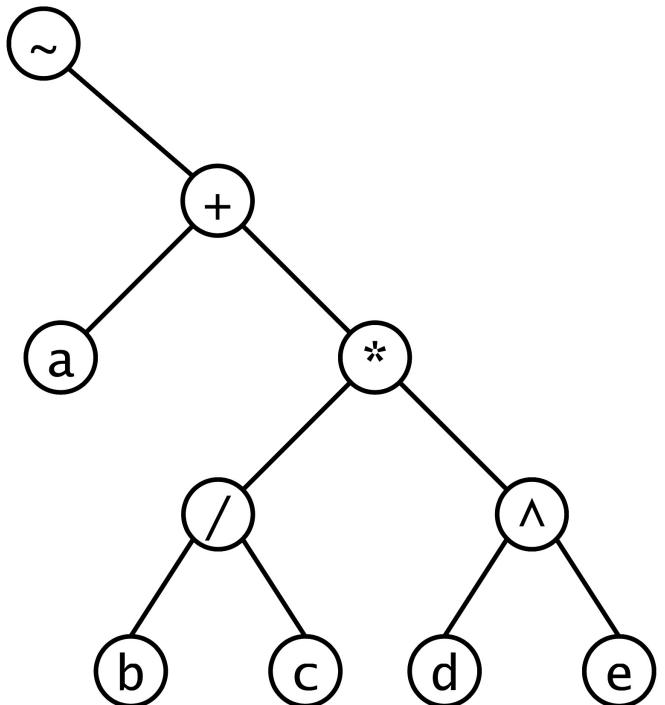
$\beta := \text{binary operator} \quad [\wedge, \times, /, +, -, \dots]$

- مثال

( ~ ( a + ( ( b / c ) \* ( d ^ e ) ) ) )

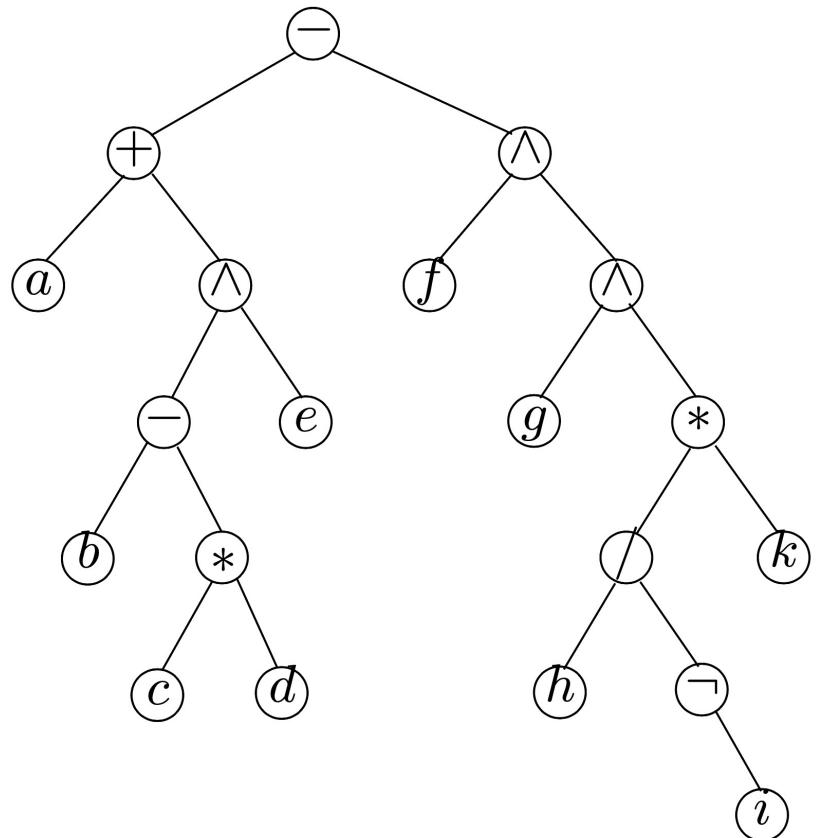
# درخت عبارت

یک مدل برای یک عبارت ریاضی استفاده از درخت است.


$$(~ a + ( b / c ) * ( d ^ e ))$$

پیمایش‌های پیش‌ترتیب، میان‌ترتیب و پس‌ترتیب درخت عبارت به ترتیب معادل فرم‌های پیشوندی، میانوندی و پسوندی عبارت هستند.

# درخت عبارت

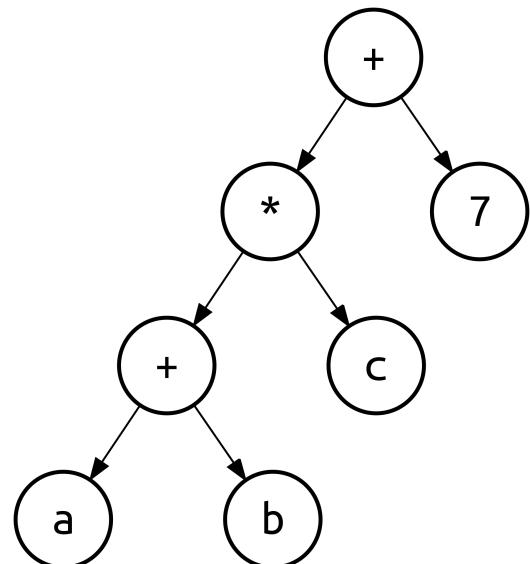


a + (b - c \* d) ^ e - f ^ g ^ (h / -i \* k)

# پیمایش و نمایش درخت عبارت

هر کدام از نمایش‌ها استفاده‌های خود را دارند.

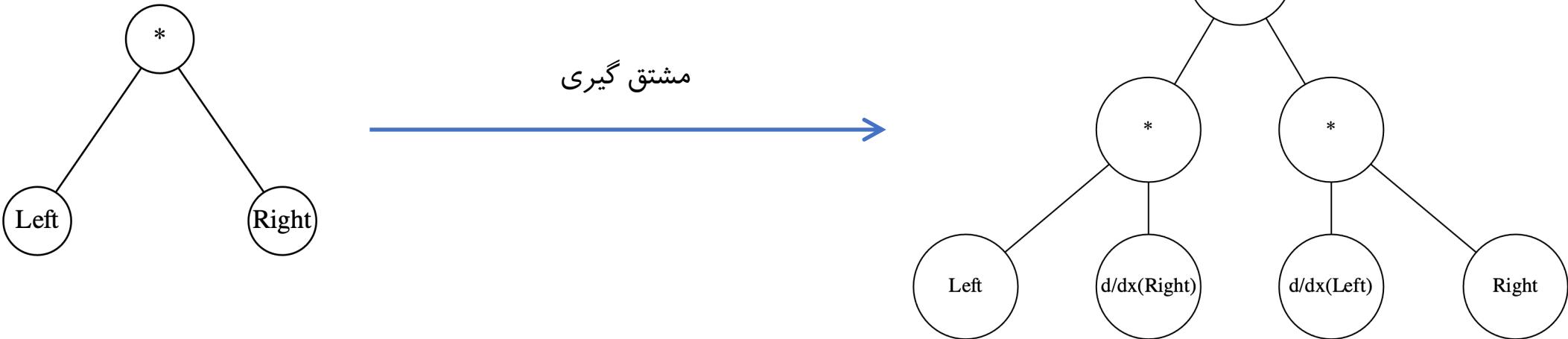
نکته‌ی جالب درباره‌ی نمایش پیش‌ترتیب و پس‌ترتیب این است که نیازی به پرانتزگذاری ندارند و بدون ابهام تفسیر می‌شوند.



```
void prefix_print(node *cur)
{ // + * + a b c 7
    cur-> print();
    if (cur-> l)
        prefix_print(cur-> l);
    if (cur-> r)
        prefix_print(cur-> r);
}
```

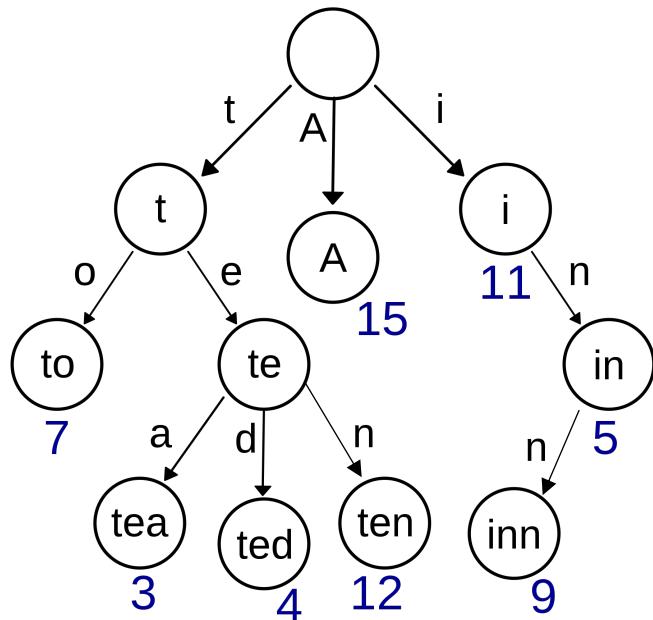
# کاربرد درخت عبارت

عموما برای محاسبه توابع پیچیده روی عبارت ها از درخت عبارت بهره می گیریم.  
برای مثال برای مشتق گیری از یک عبارت، می توان با استفاده از ساختار بازگشته درخت ها، به راحتی این عمل را انجام داد:



# درخت پیشوندی

درخت پیشوندی (Trie Tree) یک داده‌ساختار درختی است که برای ذخیره‌سازی نگاشت‌ها استفاده می‌شود. یادآوری: نگاشت مجموعه‌ای از زوج مرتب‌های (کلید، مقدار) است که هر کلید حداقل یک بار می‌آید.

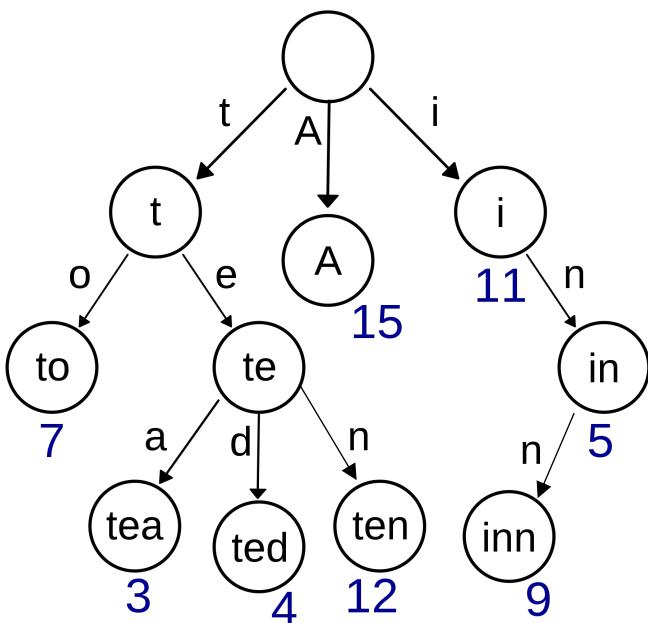


# درخت پیشوندی

به بیان عمومی اما، درخت ترای یک ساختمان داده برای ذخیره‌سازی رشته‌ها و جستجو در آن‌ها است.

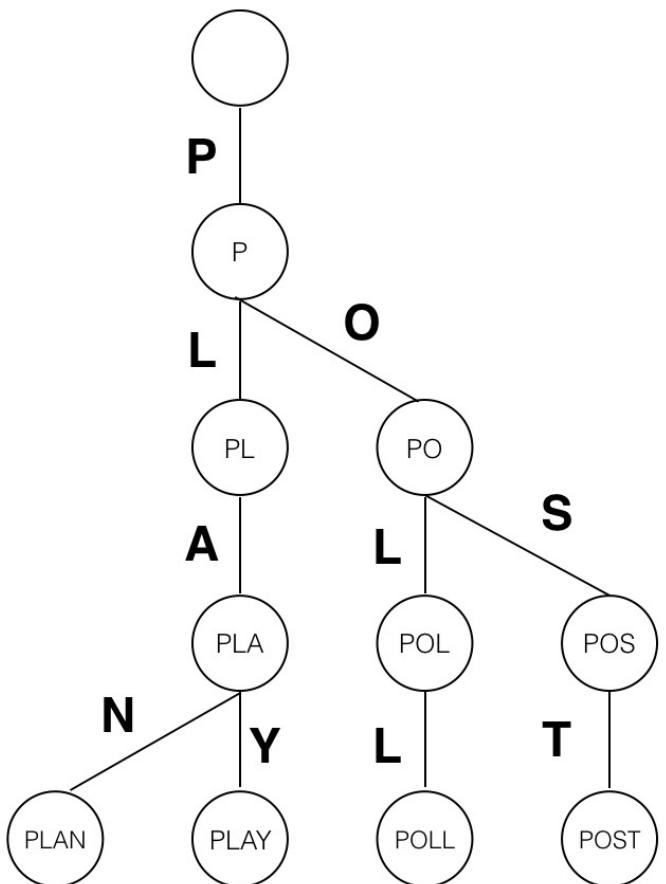
در این درخت، برای هر گره یک رشته متناظر ساخته می‌شود که برابر با حروف یال‌های مسیر رشته تا آن راس است؛ بنابراین اگر برای آن گره یک نشانه (وابسته به شرایط) وجود داشته، می‌گوییم که رشته مربوطه در درخت وجود دارد.

یک ترای با کلیدهای:  
inn, in, i, ten, ted, tea, to, A



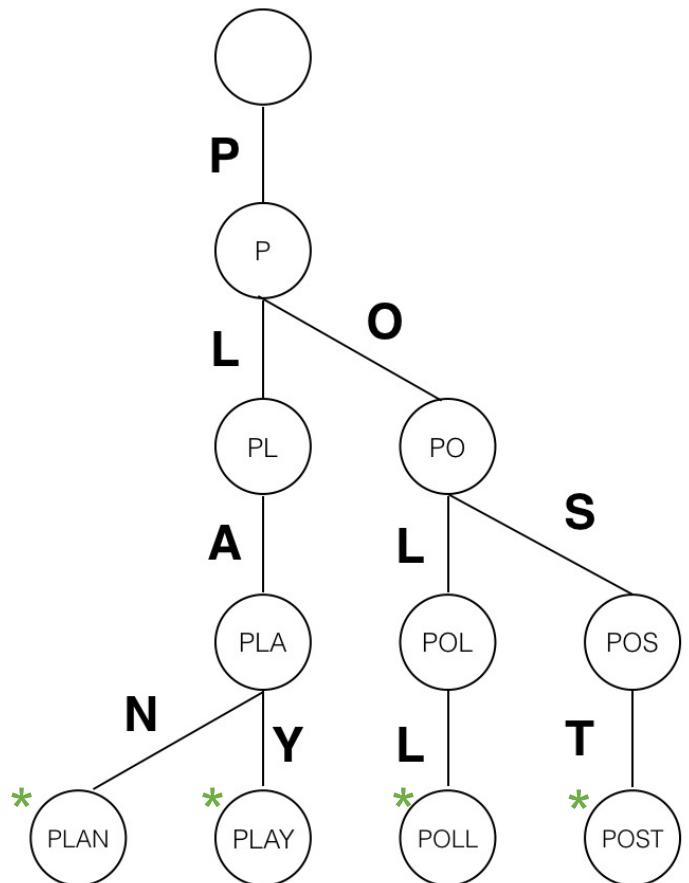
برای مثال در این درخت، نشانه راس tea برابر با ۳ است، اما گره نشانه‌ای ندارد.

# درخت پیشوندی



- گره ریشه یک رشته خالی است.
- موقعیت گره در درخت نشان دهنده کلید مربوط به آن است.
- تمام فرزندان یک گره پیشوند مشترکی دارند که این پیشوند در گره مربوطه ذخیره می‌شود.
- عموماً همه گره‌ها مشخص کننده کلیدها نیستند.
- فقط برگ‌ها و بعضی از گره‌های داخلی با کلیدها مرتبط می‌شوند.
- گره‌های حاوی کلید به نحوی علامت‌گذاری می‌شوند تا تمام کلیدها مشخص شوند.
- البته یک ترای الزاماً شامل رشته‌های کاراکتری نمی‌باشد، بلکه حتی برای جایگشت‌های عددی و مواردی از این قبیل هم می‌توان از ترای استفاده کرد.

# درخت پیشوندی



- گره ریشه یک رشته خالی است.
- موقعیت گره در درخت نشان دهنده کلید مربوط به آن است.
- تمام فرزندان یک گره پیشوند مشترکی دارند که این پیشوند در گره مربوطه ذخیره می‌شود.
- عموماً همه گره‌ها مشخص کننده کلیدها نیستند.
- فقط برگ‌ها و بعضی از گره‌های داخلی با کلیدها مرتبط می‌شوند.
- گره‌های حاوی کلید به نحوی علامت‌گذاری می‌شوند تا تمام کلیدها مشخص شوند.

البته برای بهبود حافظه، در گره‌ها فقط نشانه‌ها ذخیره می‌شوند و کلید‌ها در طی جستجو ساخته می‌شوند.

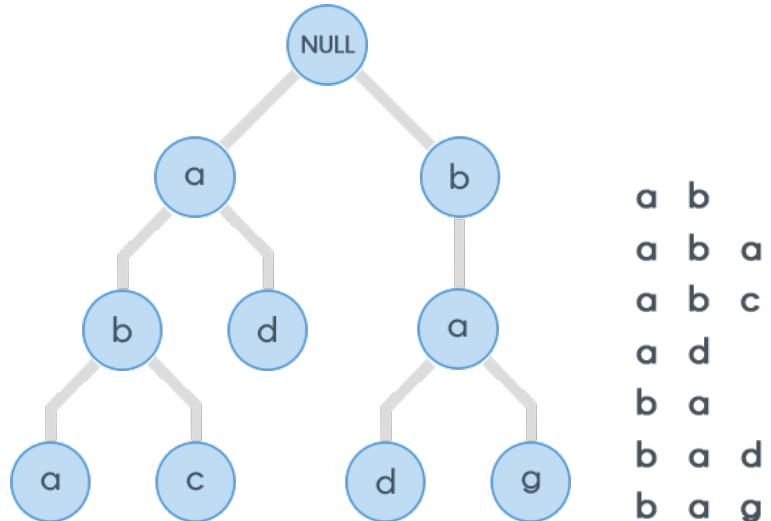
# کاربرد درخت پیشوندی

- برای ذخیره‌سازی و دسترسی سریع به رشته‌های یک لغتنامه کاربرد فراوانی دارد.
- برای برخی سیستم‌های تکمیل خودکار رشته (autocomplete) استفاده می‌شود.
- می‌تواند در بعضی شرایط به عنوان جایگزینی برای جدول‌های درهم‌سازی استفاده بشود.



# کاربرد درخت پیشوندی

- از ترای برای مرتبسازی داده‌ها در زمان خطی استفاده می‌شود. مثلاً اگر مجموعه‌ای از رشته‌ها را در ترای درج کنیم و نمایش پیش‌ترتیب آن را چاپ کنیم، رشته‌ها به ترتیب لغتنامه‌ای مرتب می‌شوند.
- همچنین اگر ترای را با مجموعه‌ای از اعداد بسازیم، جستجوی سطح اول اعداد را به ترتیب می‌پیماید.



# درخت هافمن

همانطور که می دانید، در کامپیوتر برای هر کاراکتری، یک کد (با استاندارد اسکی) بین 0 تا 255 در نظر گرفته می شود. که به هفت بیت فضا نیاز دارد.

مثلا برای نمایش حرف A از عدد 65 استفاده می شود. که در مبنای 2 به صورت 1000001 درخواهد آمد و در نتیجه برای ذخیره شدن به 7 بیت فضا نیاز دارد. که در این صورت رشته AAAAaaaa که متشکل از 8 حرف A است به فضایی معادل  $8 \times 8 = 64$  بیت یا به بیان ساده تر 7 بایت نیاز دارد.

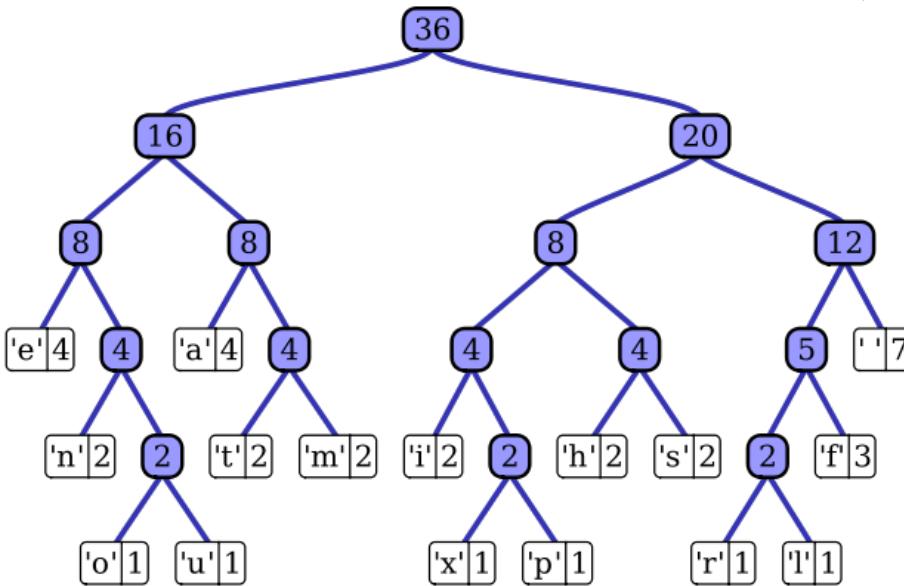
در استاندارد اسکی 254 کاراکتر مجاز دیگر به جز A وجود دارد. اما در رشته AAAAaaaa فقط یک نوع کاراکتر بکار رفته!

می توان این کد را به طور قراردادی به کد کوتاهتری (مثلا 1) تغییر دهیم. در این صورت رشته فوق در فضایی به طول  $8 \times 1 = 8$  بیت یا به بیان ساده تر 1 بایت قابل ذخیره سازی است.

# درخت هافمن

کد	تکرار	حرف
111	7	space
010	4	a
000	4	e
1101	3	f
1010	2	h
1000	2	i
0111	2	m
0010	2	n
1011	2	s
0110	2	t
11001	1	l
00110	1	o
10011	1	p
11000	1	r
00111	1	u
10010	1	x

در تئوری اطلاعات، کدگذاری نوع مشخصی از کد بهینه است که کاربردی فراوان در فشردهسازی بی‌اتلاف اطلاعات دارد.



درخت هافمن، ایجاد شده از تعداد تکرار حرف‌های جمله:  
this is an example of a huffman tree  
رمز کردن این جمله به کمک این کد، به 135 بیت نیاز دارد.

# تعریف مسئله

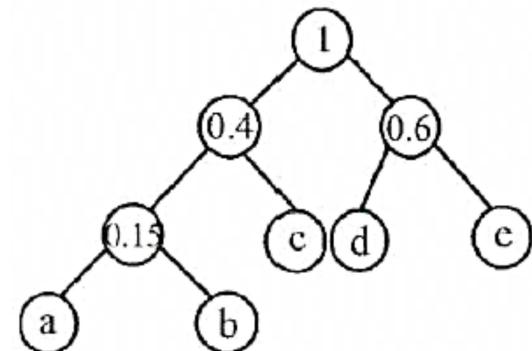
داده‌های مسئله:

مجموعه‌ای از نمادها (حروف و کاراکترها) به همراه وزن هر کدام؛  
وزن هر حرف غالباً همان تعداد تکرار آن در فایل منبع است.

خواسته:

یافتن کد دودویی بدون پیشوند با کمترین میانگین برای طول کد کل مجموعه.

حروف	a	b	c	d	e
فراوانی	0.05	0.1	0.25	0.28	0.32



# درخت هافمن

- .i. چگالی هر کاراکتر را محاسبه میکنیم (تعداد دفعات حضور کاراکتر در متن مورد نظر).
- .ii. دو کاراکتر با کمترین میزان تکرار (چگالی) را انتخاب میکنیم.
- .iii. کاراکتر های مرحله ۲ را با کاراکتر جدیدی که دارای چگالی برابر با مجموع چگالی دو کاراکتر فوق است جایگزین میکنیم.
- .iv. زمانی که فقط یک کاراکتر باقی مانده باشد، به مرحله ۲ میرویم.
- .v. از عملیات فوق یک درخت حاصل می شود، بر روی این درخت هر مسیر به سمت چپ با ۰ و هر مسیر به سمت راست با ۱ وزن دهی میشود.
- .vi. کد هر کاراکتر با کنار هم گذاشتن وزن ها از ریشه تا آن کاراکتر به دست می آید.

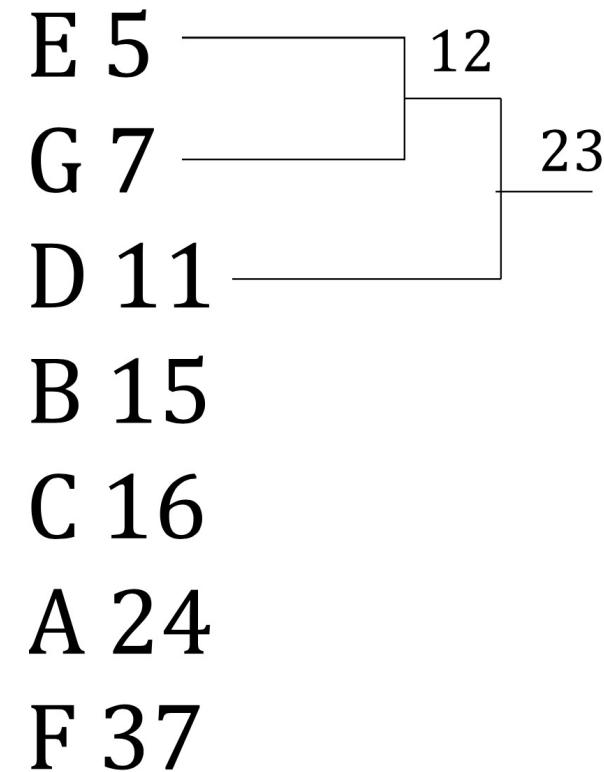
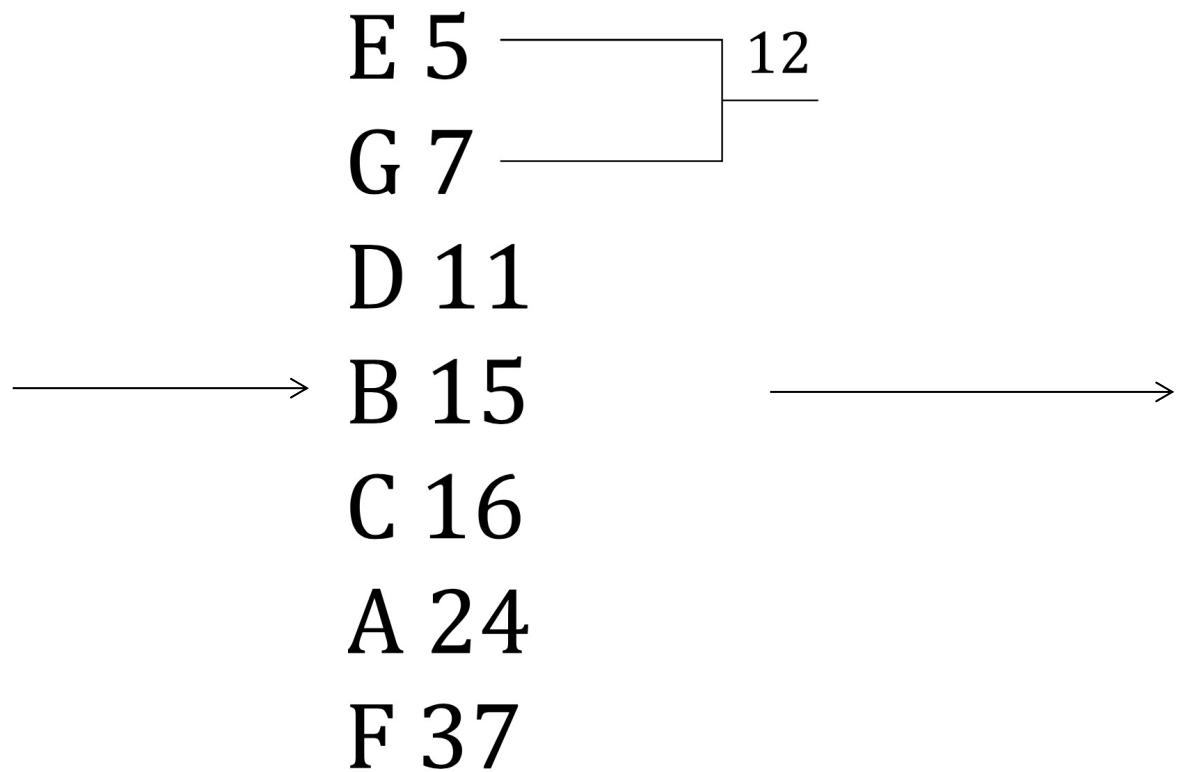
# مراحل کدگذاری

حرف	تعداد تکرار
A	24
B	15
C	16
D	11
E	5
F	37
G	7

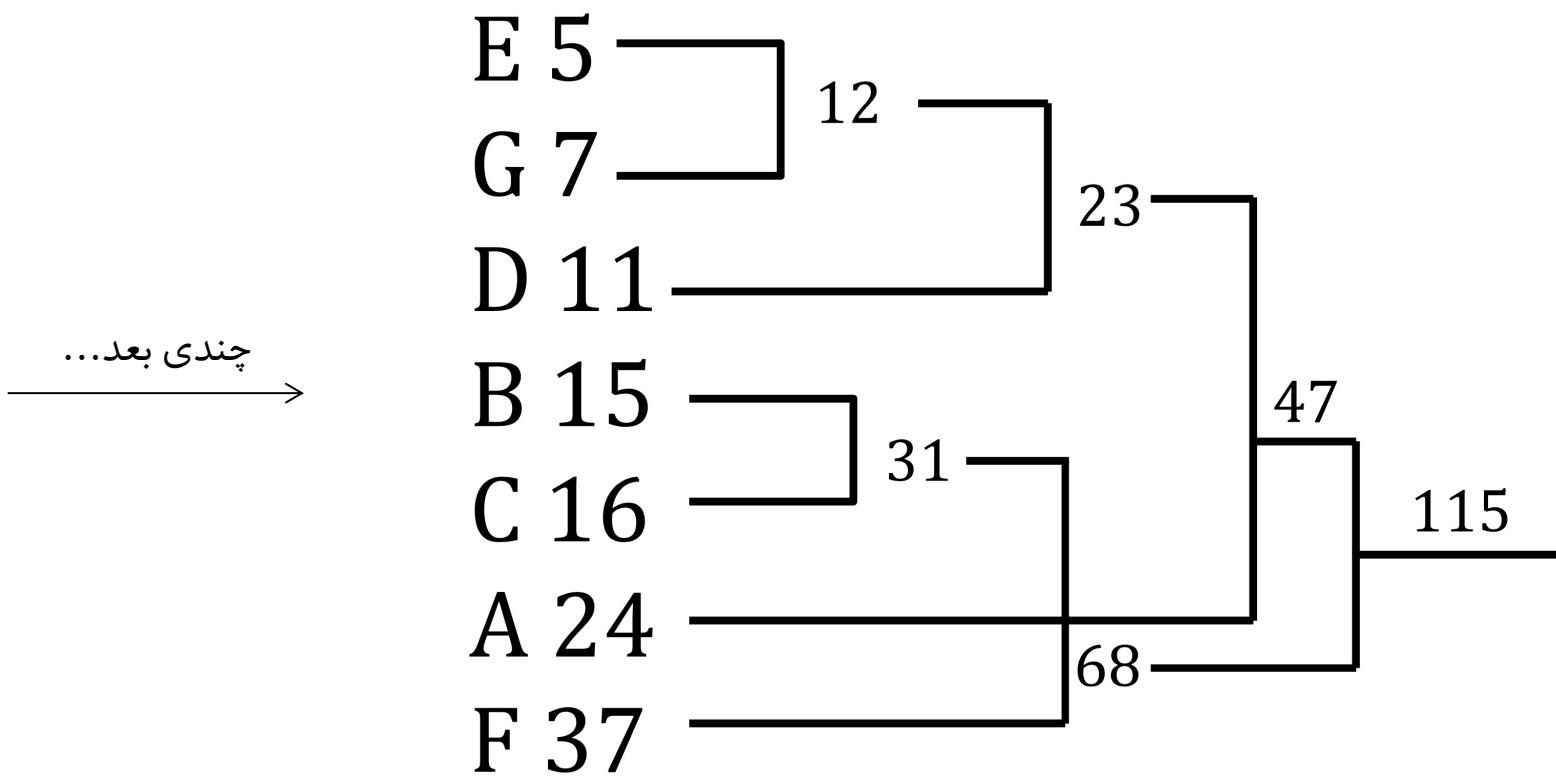
مرتب سازی بر اساس تعداد تکرار

E 5  
G 7  
D 11  
B 15  
C 16  
A 24  
F 37

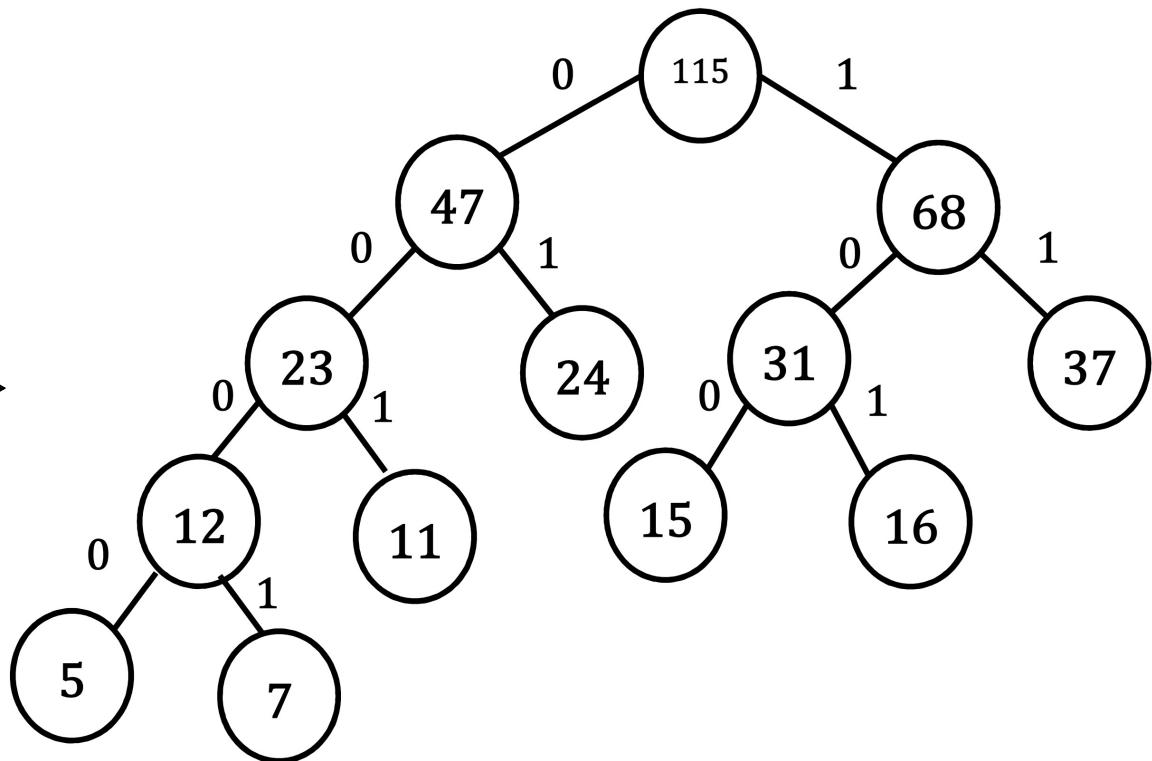
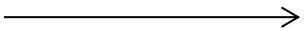
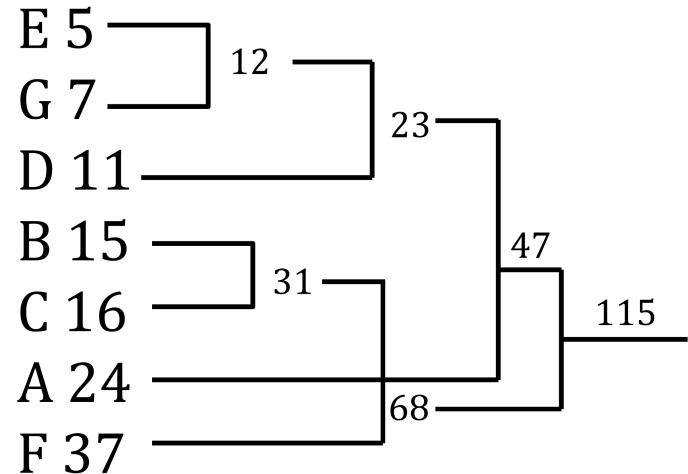
# مراحل کدگذاری



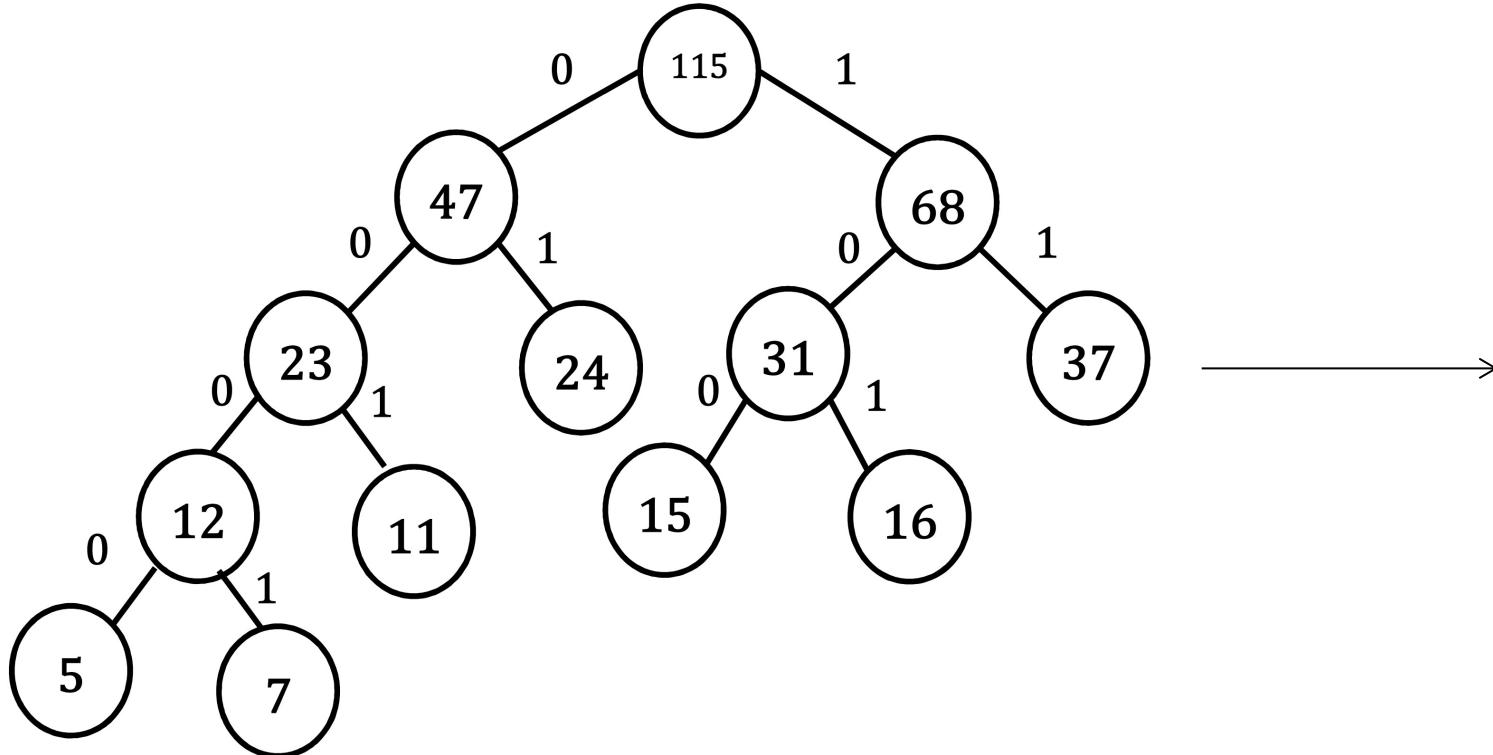
# مراحل کدگذاری



# مراحل کدگذاری



# مراحل کدگذاری



حرف	تعداد تکرار	مقدار عددی
A	24	01
B	15	100
C	16	101
D	11	001
E	5	0000
F	37	11
G	7	0001

# مقایسه

حروف	تعداد تکرار، در حالت عادی	تعداد تکرار در درخت هافمن پس از فشرده سازی
A	24	2
B	15	3
C	16	3
D	11	3
E	5	4
F	37	2
G	7	4
	115	21