ساختمان داده ها

دانشگاه صنعتی نوشیروانی بابل

دکتر حسام عمران پور

طراحان اسلاید:
زهرا ریحانیان و دانیال علیزاده

حل تمرین :
علی باقری

لینک کانال تلگرام اطلاع رسانی و حل تمرین:
t.me/ds_nit_4011

فصل هفتم : درخت

مدرس : دکتر حسام عمرانپور

فهرست

مفاهیم پایه

درخت مجموعه از عناصر است که گره نام دارنـد.

یکـی از ایـن گـره هـا، در جایگـاه ریشـه درخت قرار گرفته و با رابطه ای پدر - فرزندی ساختار سلسله مراتبی درخت را ایجـاد مـی نماید.

یک گره می تواند حاوی هر داده ای (اعم از عدد، حرف، رشته ای از حروف و یا ساختارهای پیچیده تر) باشد.

هر گره، یک پدر دارد (به جز ریشه) و می تواند یک، دو و یا چنـدین فرزنـد داشـته باشـد.

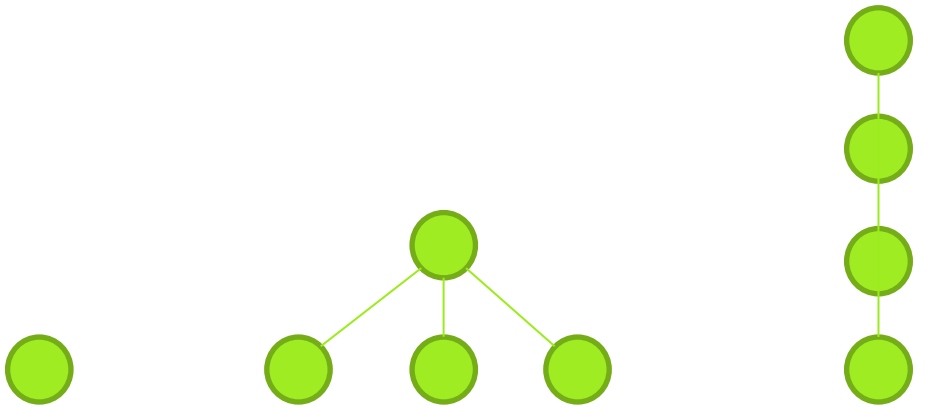
گره ای که فاقد فرزند باشد برگ می نامند.

چند نمونه درخت

(الف) درختی با یک گره.

(ب) درختی با چهار گره.

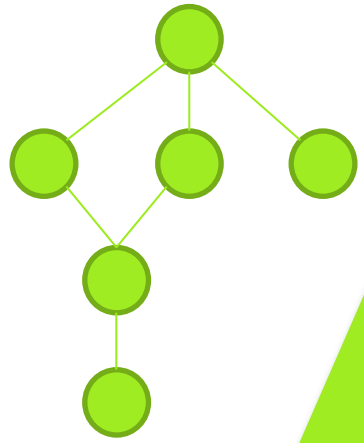
(پ) در این درخت هر گره تنها یک فرزند دارد.



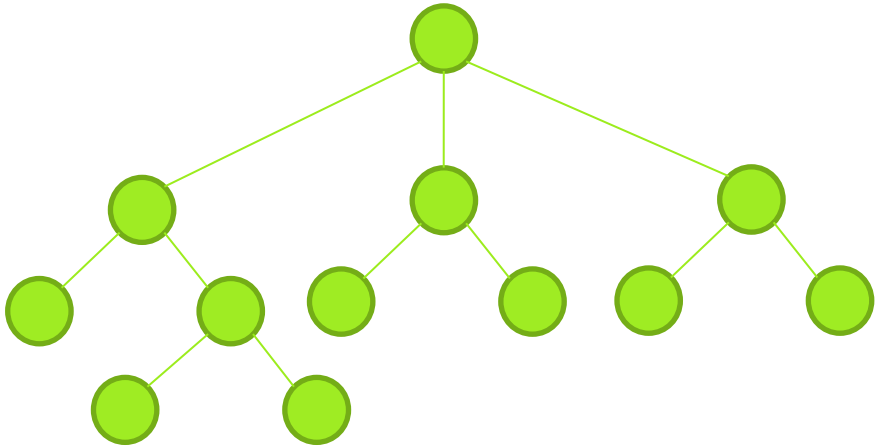
(الف) (ب) (پ)

در درخت هر گره فقط و فقط یک پدر دارد (تنها استثنا ریشه است که فاقد پدر است).

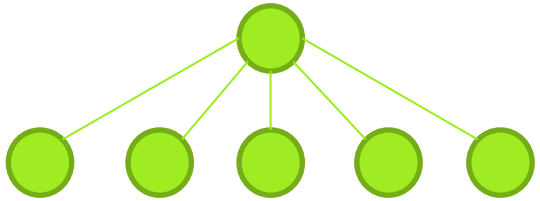
ساختار(ج) درخت نیست.



(ج)

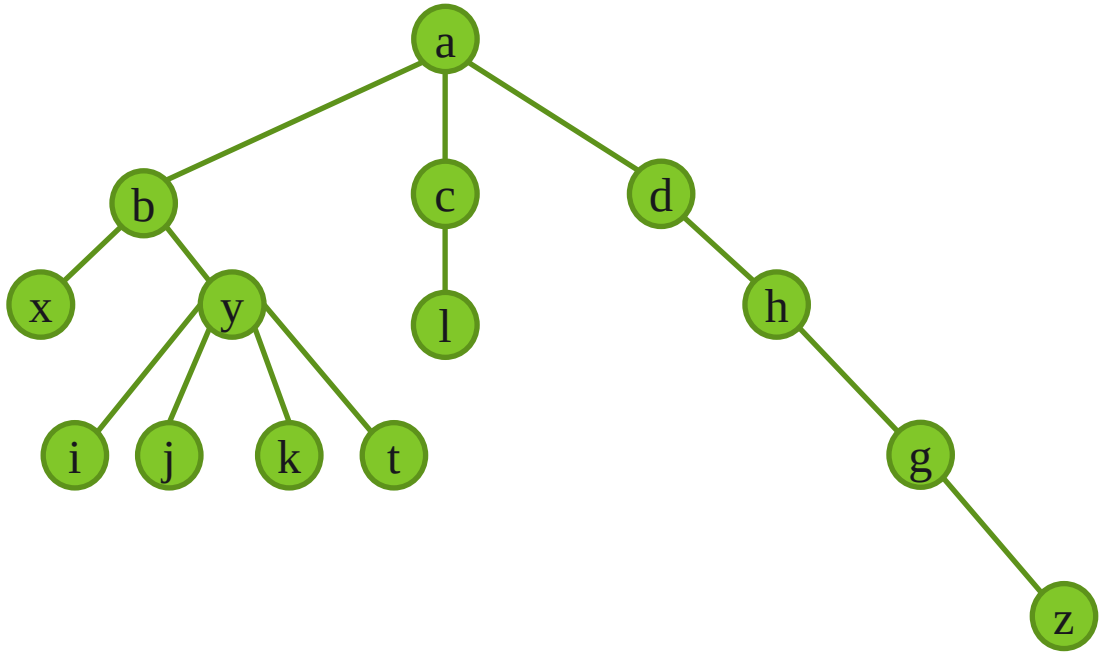


(ث)



(ت)

به شکل زیر دقت کنید. در درخت نمایش داده شده در این شکل، گره ی a، ریشه ی درخت است و دارای ۳ فرزند می باشد (b, c, d) گره ی b که فرزند گره ی a می باشد، خود دارای ۲ فرزند (x, y) می باشد. بیشترین تعداد فرزند، متعلق به گره y است (۴ فرزند: t, k, j, i). گره های l, x, i, j, k, t, z برگ های درخت هستند.



تعریف بازگشتی درخت

1.یک گره به تنهایی یک درخت است و این گره ریشه ی درخت است.

2.فرض کنید r یک گره باشد و T1,T2,…,Tk ، k درخت مجزا باشند، که ریشه های آن ها به ترتیب r1,r2,…,rk است.

در این صورت با قرار دادن r به عنوان پدر گرهِ های r1,r2,…,rk درخت جدیدی ساخته می شود و T1,T2,…,Tk ، زیر درخت َهای ریشه هستند و r1,r2,…,rk فرزندان ریشه می باشند.

نکته: تنها ساختارهای ایجاد شده با استفاده از قوانین ۲ و ۱ درخت هستند.

در زیر خلاصه ای از مفاهیم پایه ای درخت آورده شده است:

هر درخت از یک و یا بیش تر گره تشکیل شده است .
تمامی گره ها یک پدر دارند، به استثنای ریشه که فاقد پدر است.
هر گره می تواند صفر و یا بیش تر فرزند داشته باشد.
گره ای که فاقد فرزند است را برگ می نامند.

درخت با یک گره، تنها حالتی است که در آن، ریشه یک برگ است.

درختی‌که تمامی گره های آن حداکثر یک فرزند دارند، فقط یک برگ خواهد داشت.

تعریف ها

مسیر: اگر $n1,n2,...,nk$ دنباله ای از گره‌های یک درخت باشد به طوری که به ازای $k > i \geq 1$ ، ni پدر $ni+1$ باشد، در آن صورت این دنباله « مسیری از گره ی n1 به nk » نامیده می‌شود؛ و طول این مسیر یکی کم تر از تعداد گره های موجود در آن است.

نکته: هر گره، مسیری به طول صفر به خودش دارد.

مثال: در شکل صفحه ۶، مسیری به طول ۲ از گره ی b به گره ی t، و مسیری به طول ۳ از گره ی a به گره ی g وجود دارد.

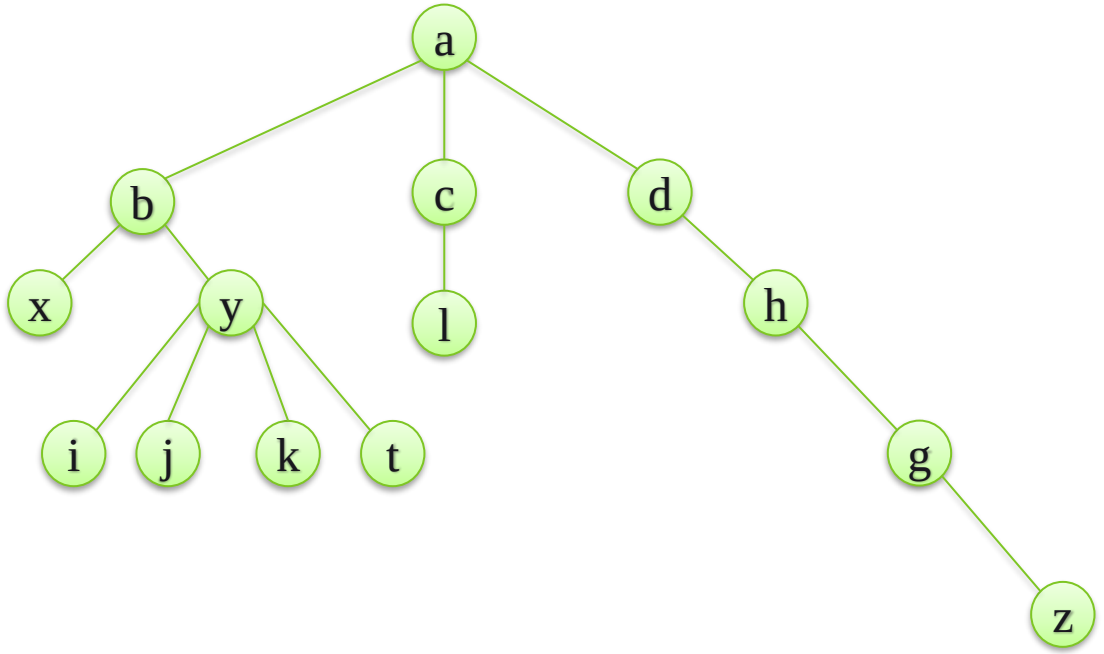
اگر مسیری از گره ی A به گره یB وجود داشته باشد، گفته می شود « A از اجداد B» و «B از نوادگان A» است.

نکته: هر گره از اجداد و نوادگان خود می باشد.

عمق: عمق یک گره، برابر طول مسیری است که از ریشه شـروع شـده و بـه آن گـره خـتم می شود.

? مثال: در شکل زیر، موارد زیر را بدست آورید.

depth(a)
depth(b)
depth(y)
depth(d)
depth(h)
depth(g)
depth(z)
depth(i)
depth(j)
depth(k)



ارتفاع: ارتفاع یک گره، برابر اندازه ی بزرگ ترین مسیری است که از آن گره به برگ وجود دارد.

نکته: ارتفاع درخت برابر است با ارتفاع ریشه ی آن.

مثال: در شکل صفحه قبل، موارد زیر را بدست آورید.

height(l)
height(b)
height(d)
height(z)

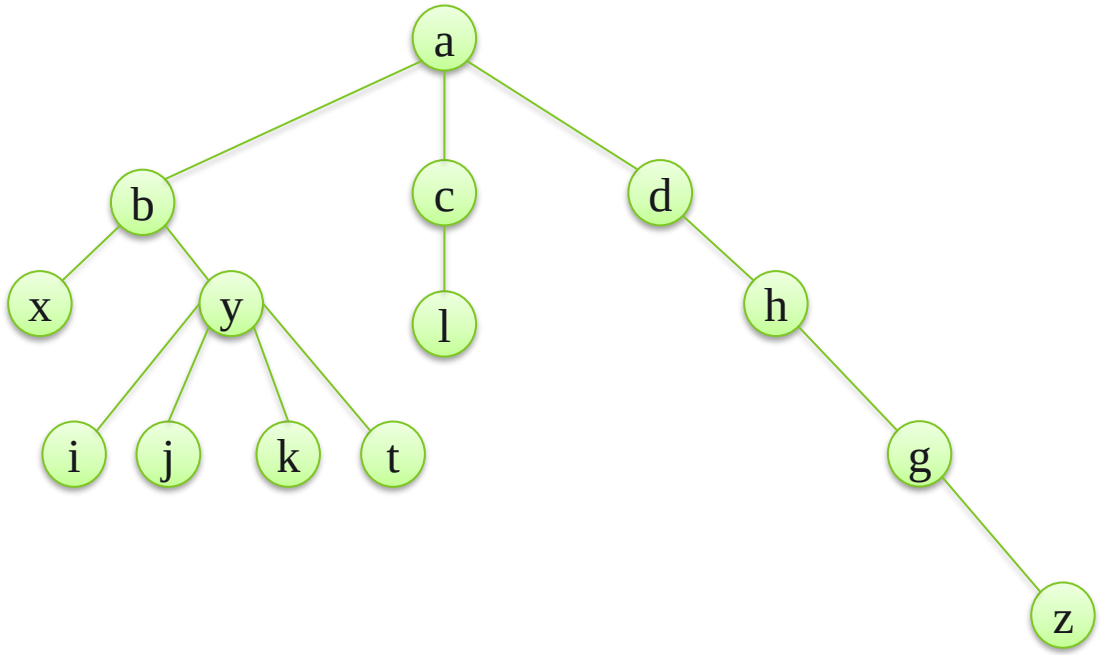
نکته: درخت با یک گره، دارای ارتفاع صفر است.

سطح: سطح یک گره، برابر عمق گره به علاوه ی یک است.

$$Level(n_i) = depth(n_i) + 1$$

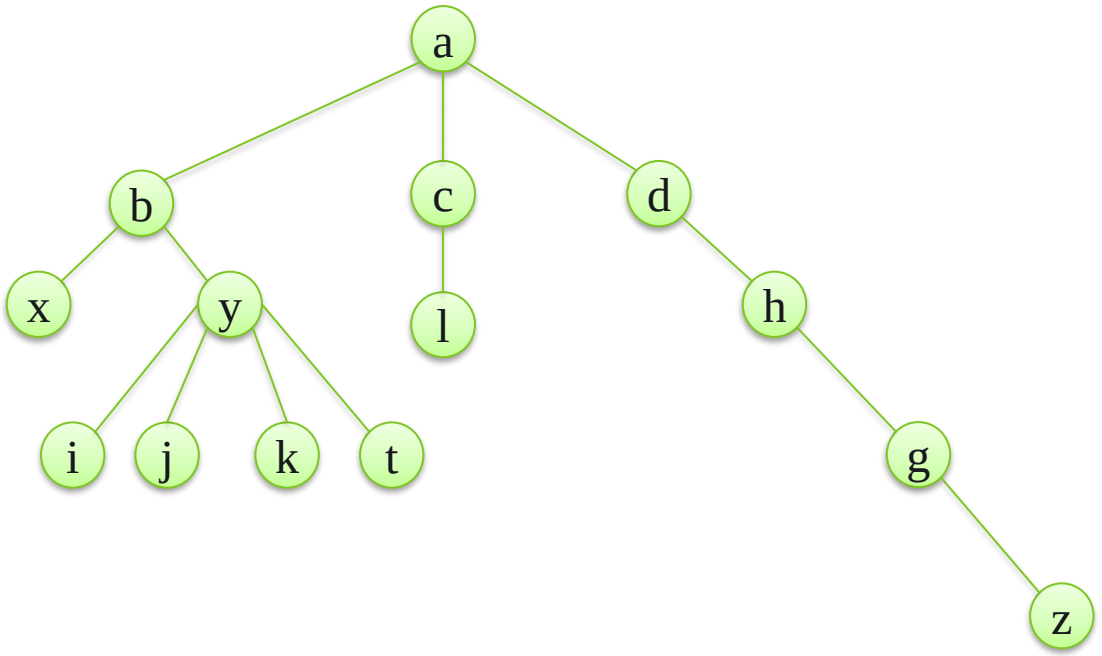
? مثال: در شکل زیر، موارد خواسته شده را بدست آورید.

level(a)
level(i)
level(g)
level(z)
level(x)
level(l)
level(y)
level(h)



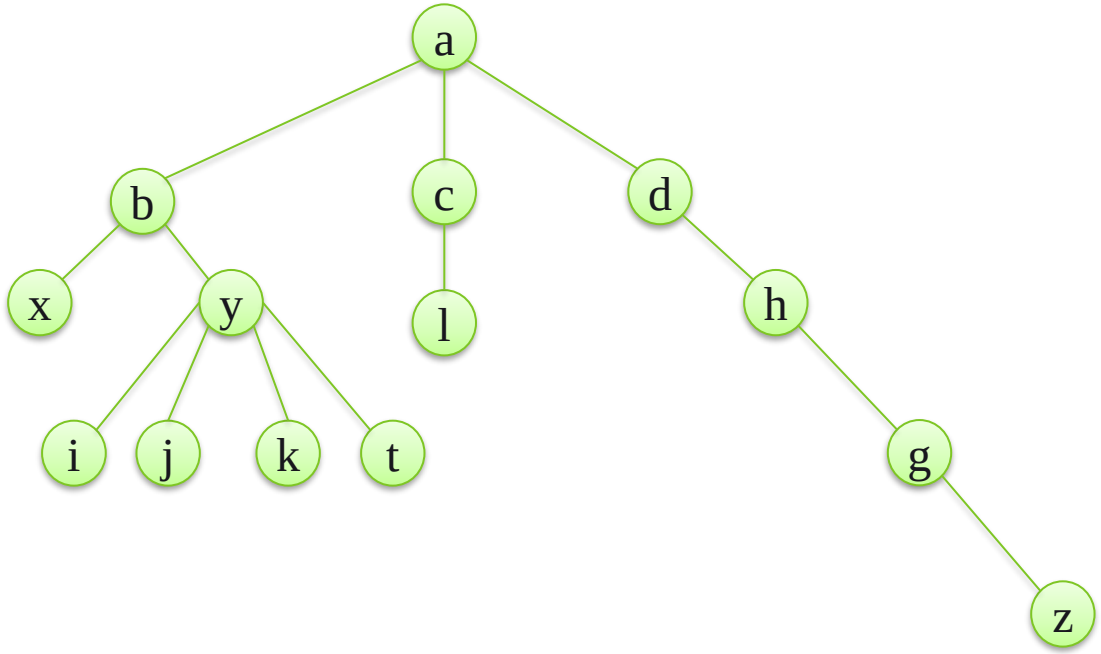
زیردرخت: در یک درخت، هر گره همراه با تمامی فرزندانش، یک زیردرخت را تشکیل می دهد.

در شکل زیر، زیردرخت b دارای ۷ گره است و ارتفاع این زیردرخت برابر ۲ می باشد. در این زیردرخت، عمق گره های t, k, j, i برابر ۲ است.



? در شکل زیر، چند زیردرخت وجود دارد؟ (خود درخت را جزو زیردرخت ها شمارش نکنید.)

پاسخ این سوال بسیار ساده است و تنها کافی است تعداد گره های درخت را بشمارید.
این درخت ۱۴ گره دارد که با در نظر نگرفتن ریشه، ۱۳ گره باقی خواهد ماند. پس این درخت دارای ۱۳ زیردرخت (بدون شمارش خود درخت اصلی) است. توجه داشته باشید که هر برگ به تنهایی، زیردرختی است به ارتفاع صفر که تنها دارای یک گره است.



درجه: تعداد فرزندان یک گره را درجه ی آن گره می نامند.

در شکل صفحه قبل، درجه ی گره های t, g, b, a به ترتیب ۳، ۲، ۱ و صفر است.

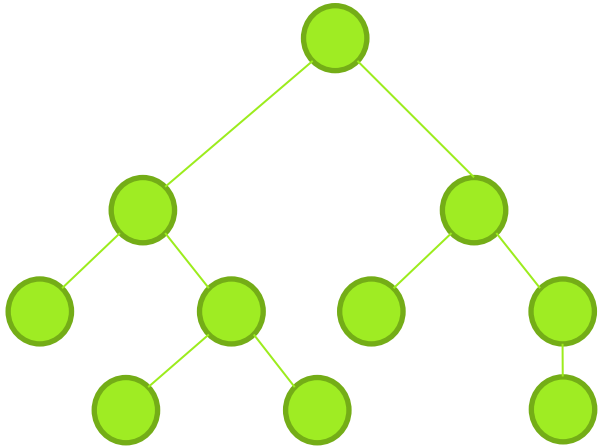
چپ ترین فرزند: چپ ترین فرزند هرگره، فرزندی از آن گره است که در نمایش درخت چپ‌تر از دیگر گره ها رسم می شود.

? مثال: در شکل صفحه قبل، چپ ترین فرزند گره های زیر را بدست آورید.

LMC(a)
LMC(g)
LMC(d)
LMC(y)
LMC(LMC(a))
LMC(b)

درخت درجه ی k (درخت k تایی): درختی که درجه ی تمامی گره های آن حداکثر k باشد.

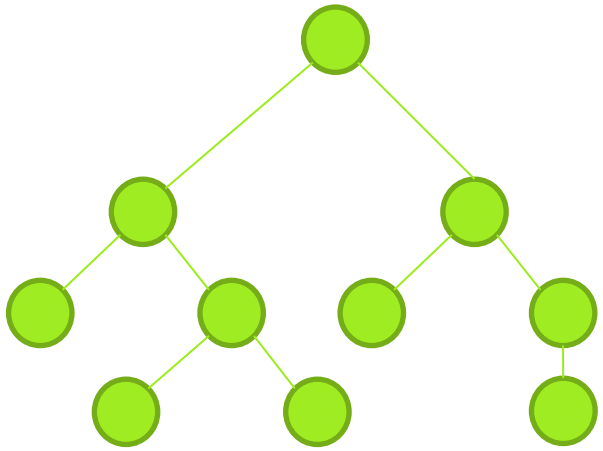
در شکل زیر دو درخت از درجه های یک و دو دیده می شود.



(الف)

(ب)

نکته: اگر درختی از درجه k باشد، می توان آن درخت را از درجه ای بزرگ تر از k نیـز در نظر گرفت، زیرا درجه ی درخت به صورت حداکثر فرزندان یک گره تعریف می شود. بـه طـور مثال، درخت (الف) در شکل زیر ، می تواند از درجه ی ۳ ، ۴ و حتی بیش تر باشد؛ و یا درخـت (ب) می تواند درختی از درجه ی ۲باشد که هیچ یک از گره های آن، دارای ۲ فرزنـد نیسـتند ولی این قابلیت بالقوه برای تمامی گره های آن وجود دارد.



(الف)

(ب)

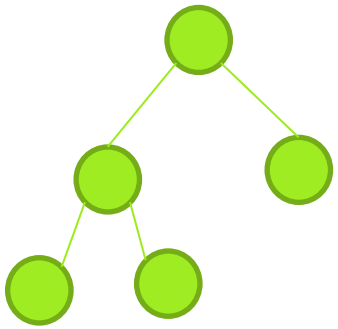
درخت کامل k تایی: درخت درجه k که تمامی گره های غیر برگ آن، دقیقا k فرزند داشته باشند.

درخت متوازن: درختی‌که تفاوت سطح برگ های آن حداکثر برابر یک باشد.

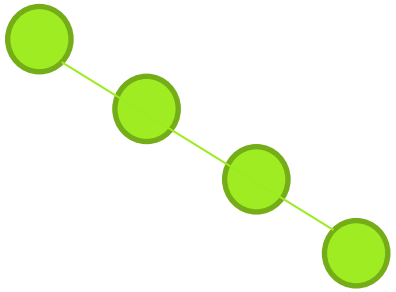
درخت کاملاً متوازن: درختی‌که تفاوت سطح برگ های آن صفر باشد و به عبارتی تمام برگ های آن در یک سطح قرار داشته باشند.

نکته: یک درخت کاملاً متوازن، درختی متوازن است.

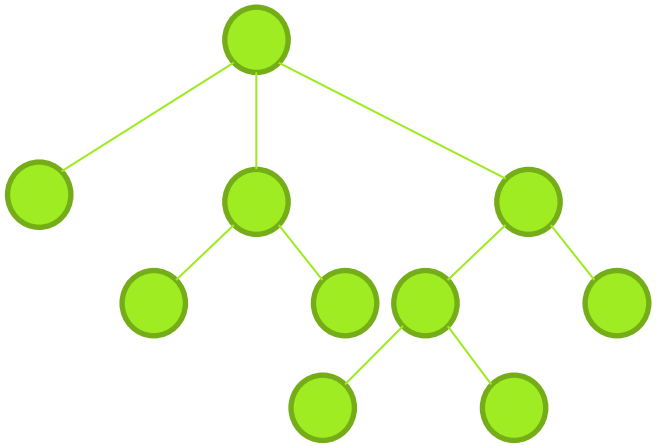
? سوال: در شکل های زیر، کدام درخت ها کاملاً متوازن و کدام یک متوازن هستند؟



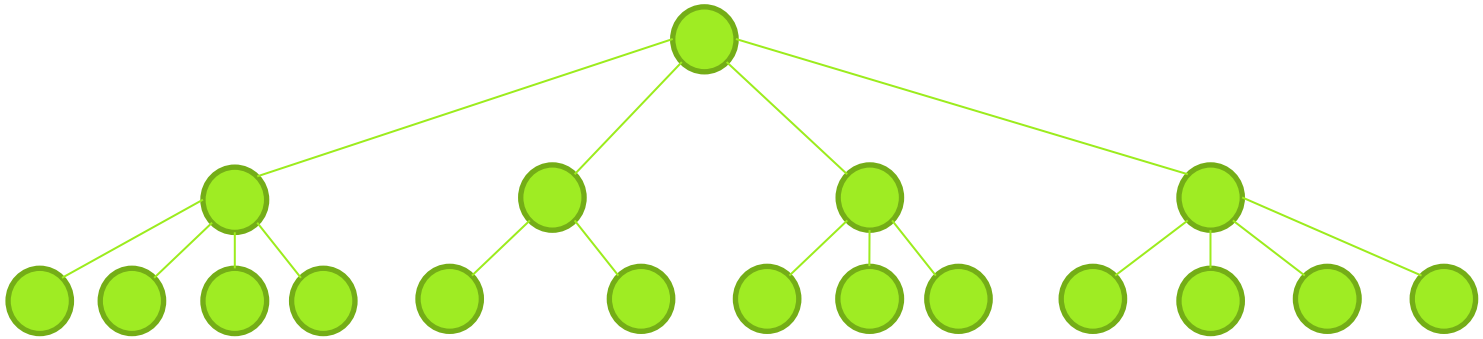
(الف)



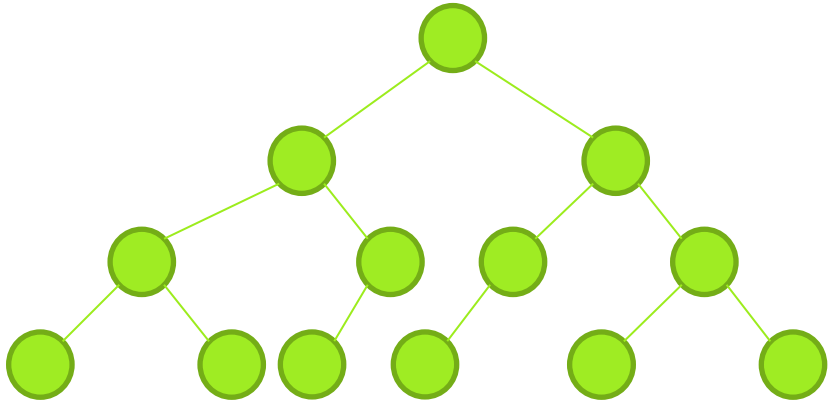
(ب)



(پ)



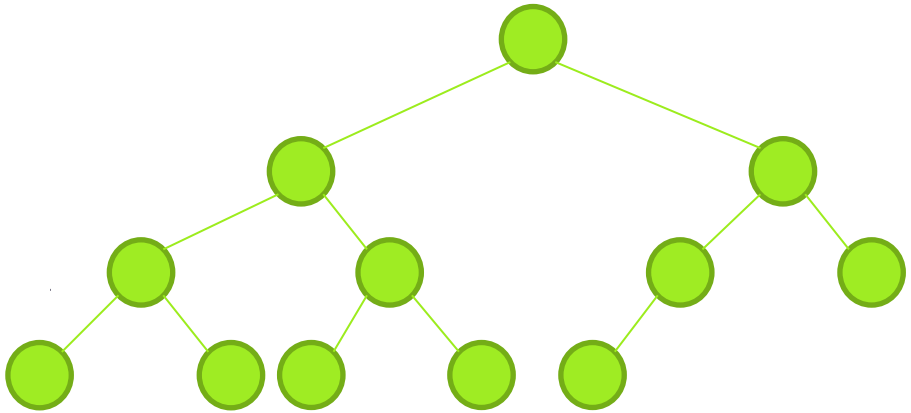
(ت)



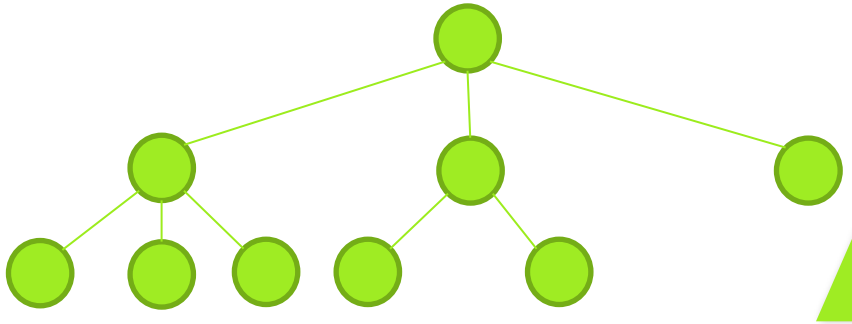
(ث)

درخت تکمیل k تایی: یک درخت کامل k تایی که متوازن است و هر سطح آن از چپ به راست پر می‌شود و پیش از پر شدن یک سطح، برگی در سطح بعد قرار نمی‌گیرد.

دو نمونه از درخت‌های تکمیل در شکل زیر دیده می‌شود.



(الف)

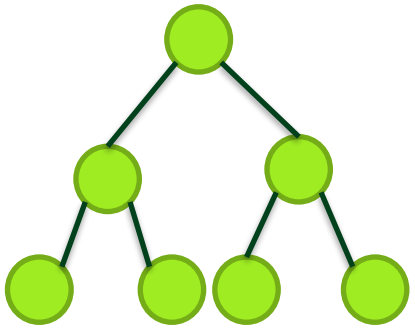


(ب)

درخت مرتب: درختی که در آن ترتیب فرزندان مهم است . در چنین درختی می‌توان فرزندان یک گره را بر اساس ترتیب قرارگیری، شماره‌بندی کرد و با جابه‌جا نمودن ترتیب فرزندان، درخت دیگری به دست آورد.

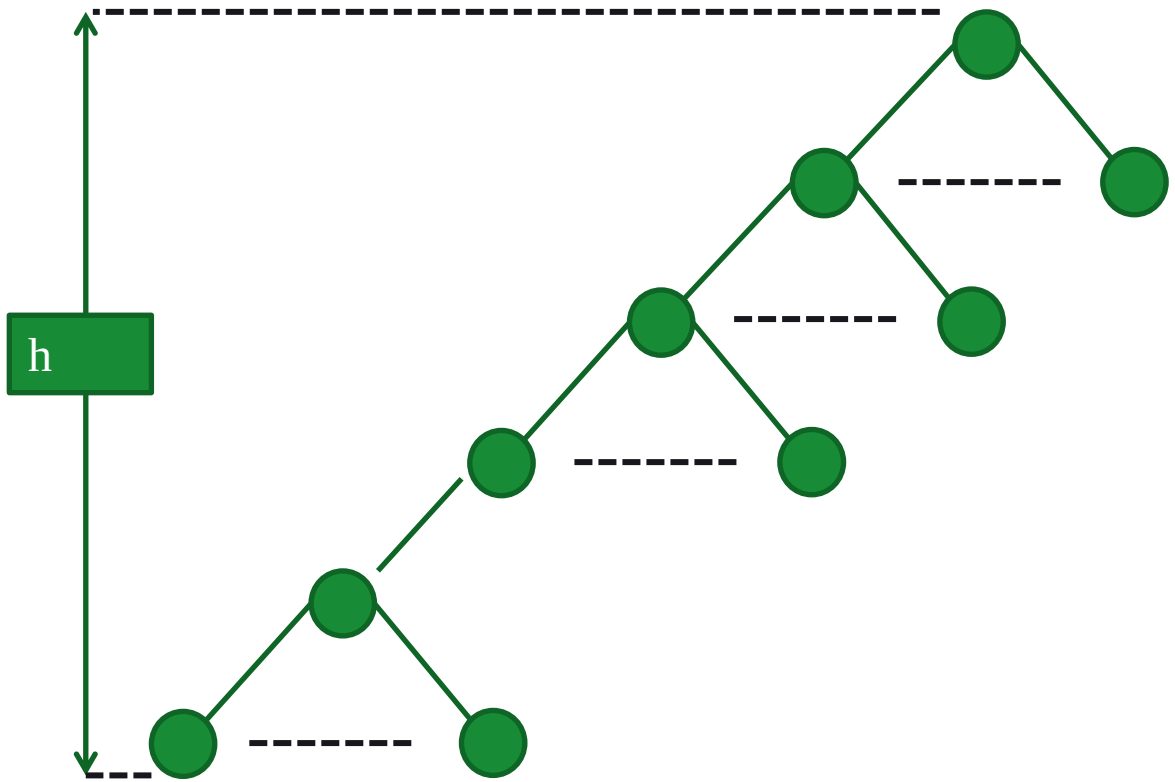
درخت پر: یک درخت کامل k تایی کاملاً متوازن است .

1. تمامی برگ‌های آن در یک سطح است.
2. تمامی گره‌های غیر برگ آن دقیقاً k فرزند دارد.



? در یک درخت کامل k تایی با ارتفاع h حداقل و حداکثر تعداد گره های درخت چقدر است؟

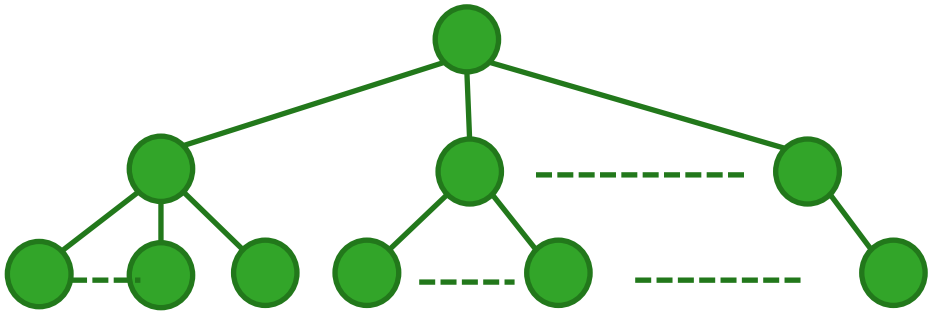
✔ حداقل تعداد گره ها، در حالتی است که در هر سطح فقط یک گره دارای فرزند باشد، مانند شکل زیر :



در این صورت در هر سطح (جز سطح ریشه) فقط k گره وجود خواهد داشت و در نتیجه k*h + 1 حداقل تعداد گره های یک درخت کامل k تایی با ارتفاع h است.

حداکثر تعداد گره ها در حالتی وجود دارد که درخت پر باشد. در این حالت تعداد گره های موجود در هر سطح یک تصاعد هندسی ایجاد می کنند :

$n_1 = k^0 = 1$
$n_2 = k^1$
$n_3 = k^2$
$n_4 = k^3$
....
$n_{h+1} = k^h$



حداکثر تعداد گره $1 + k + k^2 + k^3 + \cdots + k^h$

? تعداد گره های غیر برگ یک درخت پر k تایی با ارتفاع h چقدر است؟

پاسخ:

تعداد کل گره ها : $\frac{k^{h+1}-1}{k-1}$

تعداد برگ : k^h

تعداد گره های غیر برگ : $\frac{k^{h+1}-1}{k-1} - k^h = \frac{k^h-1}{k-1}$

باید توجه داشت که به ازای تمامی مقادیر k (به جز $k = 1$) : $k^h > \frac{k^h-1}{k-1}$.
در نتیجه همیشه برگ های یک درخت پر k تایی، بیش از نیمی از گره های آن را تشکیل می دهد.

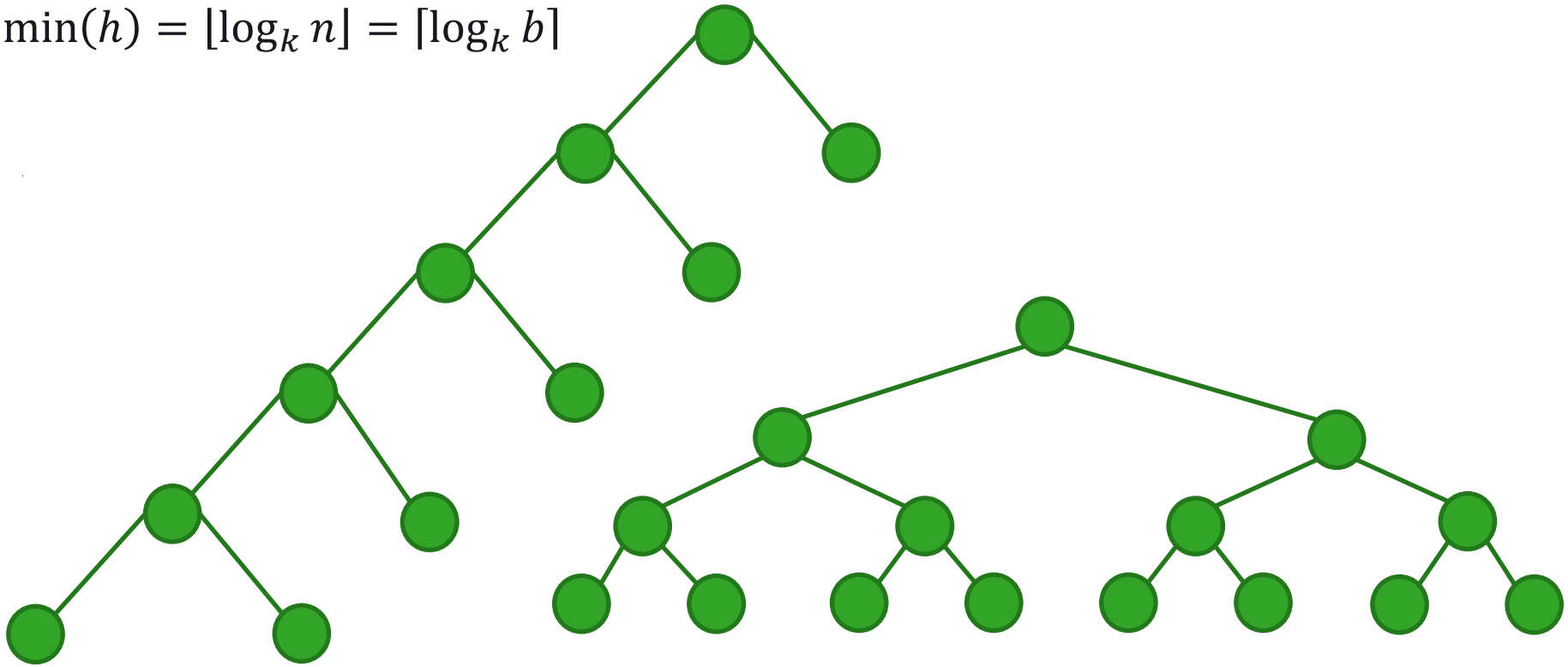
? حداقل و حداکثر ارتفاع یک درخت کامل k تایی با n گره، چقدر است؟

✔ حداکثر ارتفاع درخت زمانی حاصل می شود که در هر سطح فقط یک گره دارای فرزند باشد، مانند شکل زیر. در این حالت در هر سطح تنها k گره قرار می گیرد.

$$\max(h) = \left\lfloor \frac{n}{k} \right\rfloor$$

و حداقل ارتفاع درخت در حالتی رخ می دهد که درخت متوازن باشد.

$$\min(h) = \lfloor \log_k n \rfloor = \lceil \log_k b \rceil$$



چگونگی نمایش درخت

۱. نمایش با پرانتزگذاری:

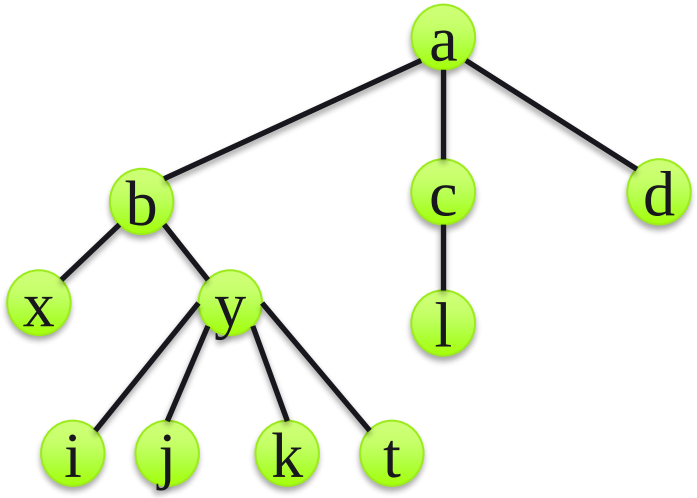
- فرزندان هرگره، درون پرانتز بعد از نام آن نوشته می‌شود.

مثلا: اگر گـره a ، دارای دو فرزنـدb و c باشد، آن را به صورت a(b,c) نمایش می‌دهند.

حال اگرگره b نیز دارای ۳ فرزنـد x، y و z باشد، نمایش این درخت به شکل a(b(x,y,z),c) خواهد شد.

در مجموعه‌ی درختـان مرتـب، درخـت a(b,c) بـا درخـت a(c,b) متفـاوت اسـت، در صورتی که در مجموعه‌ی درختان غیرمرتب، این دو درخت یکی هستند.

? درخت زیر را به صورت پرانتزگذاری بنویسید.



پاسخ:

a(b(x,y(i,j,k,t)),c(l),d)

۲. استفاده از لیست پیوندی

درختان از نظر تعداد فرزندان هر گره، به دو دسته تقسیم می‌شوند:

۱. درخت‌هایی‌که حداکثر تعداد فرزندان گره های آن مشخص است. (درخت k تایی یا درخت از درجه‌ی k)
۲. درخت‌هایی‌که تعداد فرزندان گره های آن نامشخص و نامحدود است.

درخت‌های گروه دوم را تنها می‌توان به صورت درخت عمومی نمایش داد . (در مورد درخت‌های گروه اول نیز قابل استفاده است)

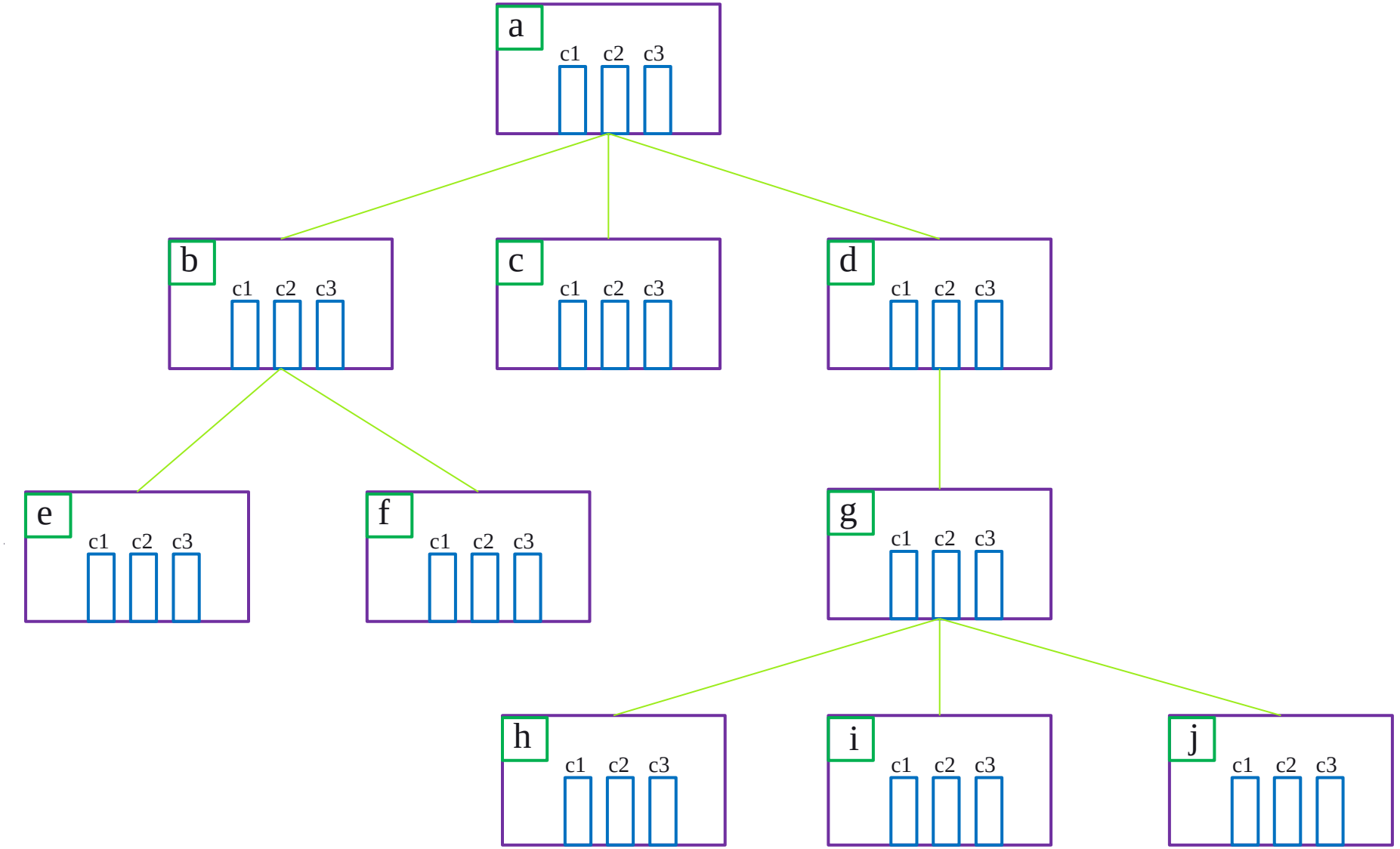
نمایش درخت با درجه ی محدود

در این روش در هر گره‌ی درخت، به تعداد حداکثر فرزندان ممکن برای هر گره (درجه ی درخت)، اشاره‌گر وجود دارد و هنگامی که گره‌ی B فرزند گره‌ی A باشد، یکی از اشاره‌گرهای گره‌ی A به گره‌ی B اشاره خواهد کرد.

در شکل صفحه بعد درختی با ۱۰ گره را می توان مشاهده نمود که هرگره ی آن حداکثر می‌تواند ۳ فرزند داشته باشد.

در نتیجه، در هر گره ۳ اشاره‌گر C۱ ، C۲ و C۳ به منظور اشاره به فرزندهای اول، دوم و سوم (در صورت وجود) قرار داده شده است.

- **مزیت** این طراحی سادگی پیاده سازی و سرعت پیمایش و انجام عملیات در درخت است .
- ولی **عیبی** که در آن دیده می‌شود، بدون استفاده ماندن تعداد زیادی اشاره‌گر است .


a
c1 c2 c3
b
c1 c2 c3
c
c1 c2 c3
d
c1 c2 c3
e
c1 c2 c3
f
c1 c2 c3
g
c1 c2 c3
h
c1 c2 c3
i
c1 c2 c3
j
c1 c2 c3

? ثابت کنید در صورت نمایش یک درخت k تایی با استفاده از این طراحی، بیش از نیمی از اشاره گر ها بدون استفاده باقی می مانند.

پاسخ:

تعداد کل اشاره گرها $= (leaf + nonLeaf) \times k$

تعداد اشاره گرهای اشغال شده $= (leaf + nonLeaf) - 1$

تعداد اشاره گرهای بلا استفاده = تعداد کل اشاره گرها − تعداد اشاره گرهای اشغال شده

$$= [(leaf + nonLeaf) \times k - (leaf + nonLeaf - 1)]$$

$$= (leaf + nonLeaf) \times (k - 1) + 1$$

تعداد اشاره گرهای بلا استفاده > تعداد اشاره گرهای اشغال شده $(k \geq 2)$

- عیب دیگر این طراحی، لزوم تغییر ساختار برای پشتیبانی درخت‌های مختلف از درجات متفاوت است.

- بدین معنی که برای استفاده از چنین ساختاری به منظور نمایش درخت ۴تایی، باید ساختار گره ها را تغییر داده و در هر گره ۴ اشاره‌گر قرار داد.

- این تغییر سبب ایجاد تغییر (هرچند جزئی) در تمامی الگوریتم‌های مورد استفاده در درخت می شود.

نمایش درخت عمومی با درخت دودویی

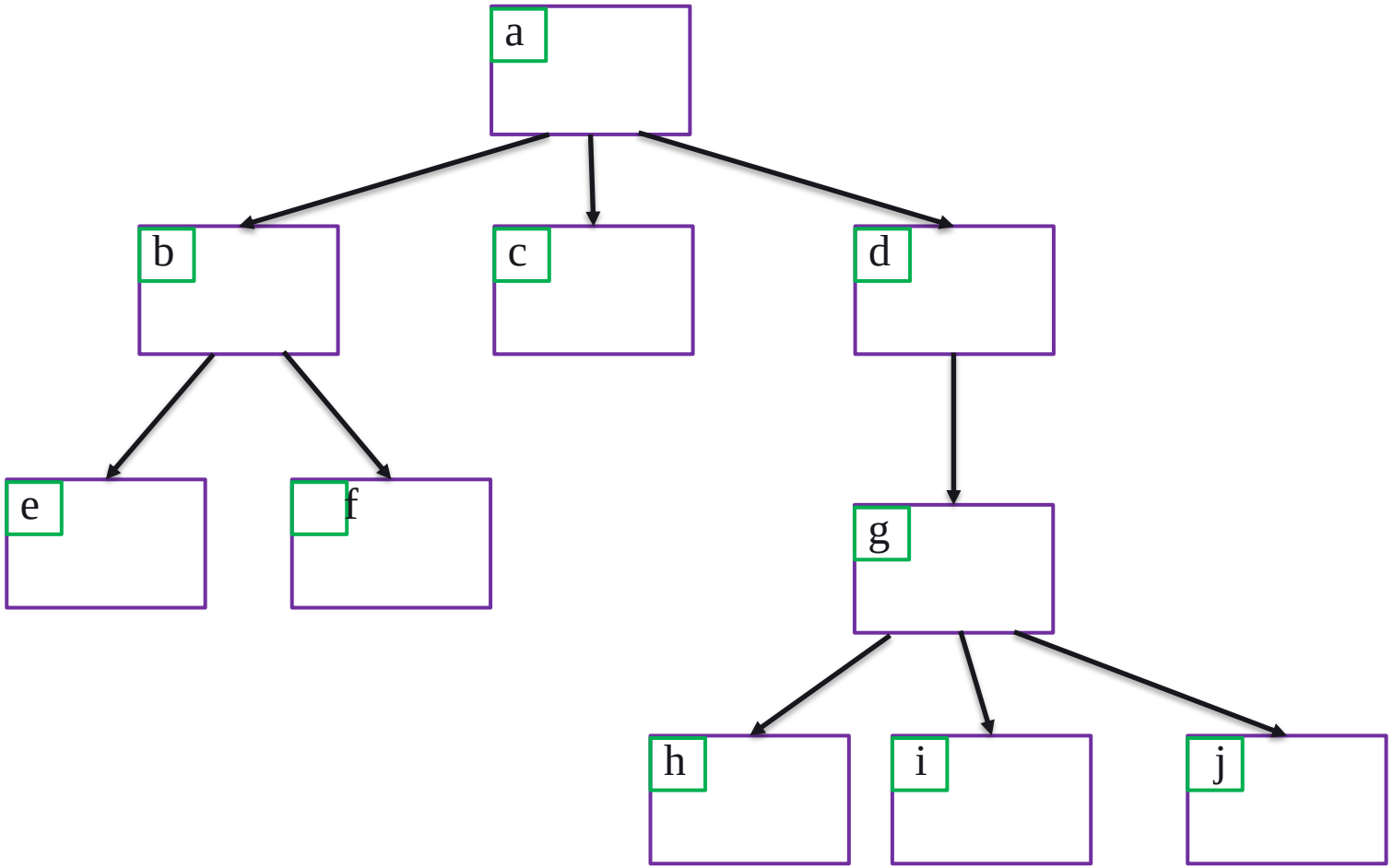
در این روش، هر گره تنها ۲ اشاره گر خواهد داشت:

- اشاره‌گرL برای اشاره به چپ فرزند ترین
- اشاره‌گر R به منظور اشاره به برادر سمت راست

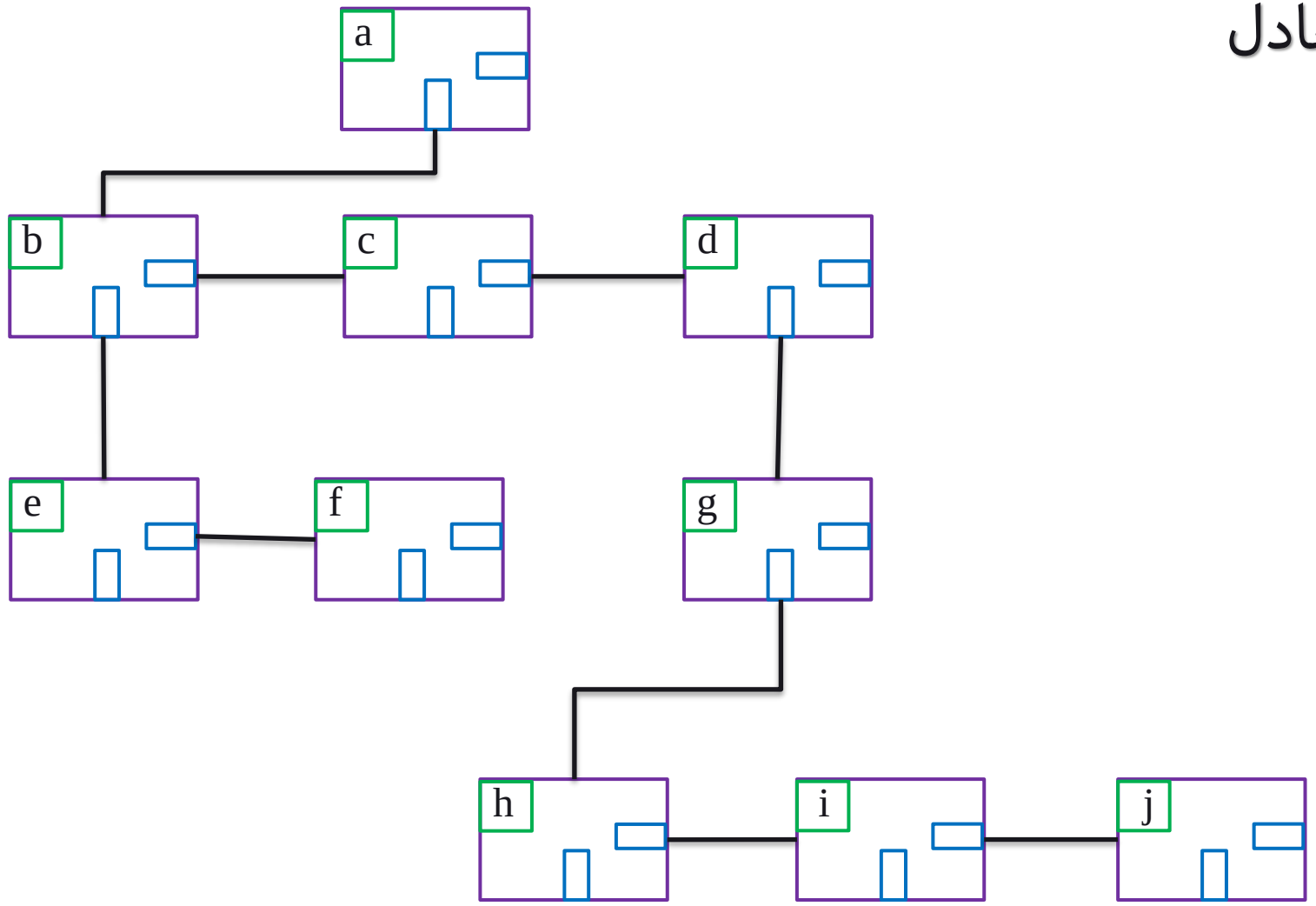
معایب این طراحی:

1. دشواری پیاده سازی ساختار و الگوریتم
2. پیچیدگی زمانی ضعف

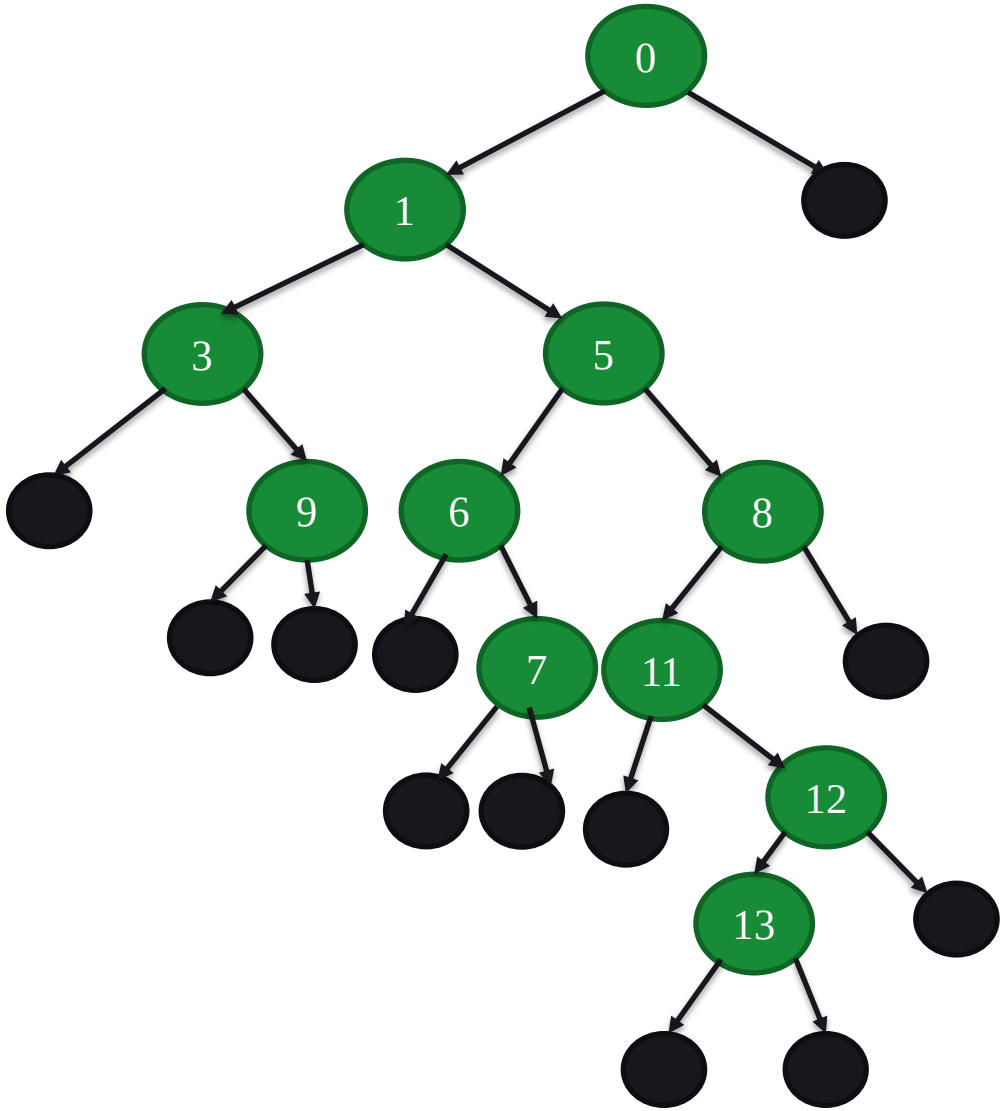
مثال : درخت عمومی



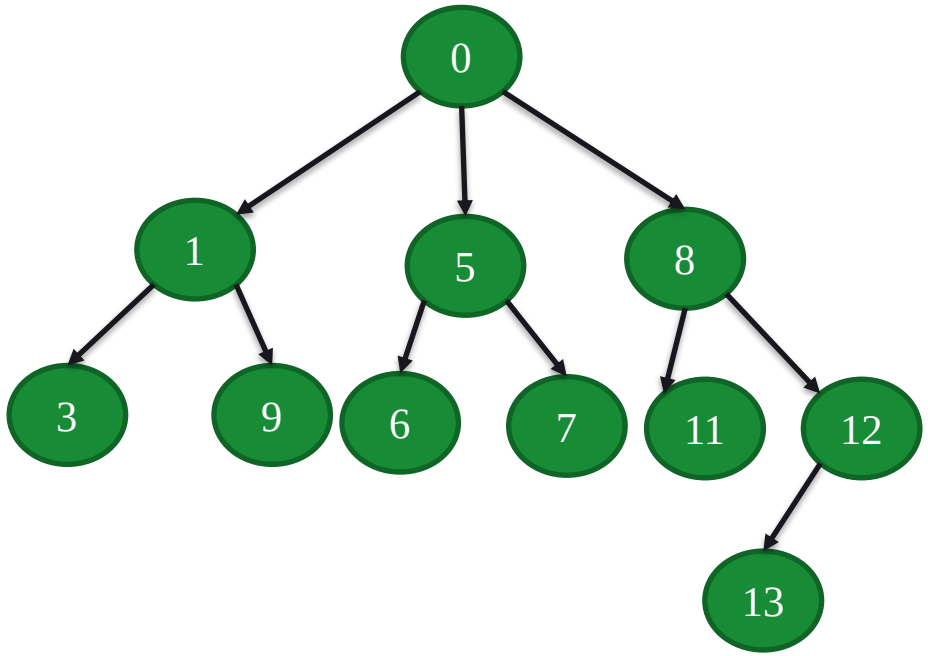
درخت دودویی معادل



? درخت زیر یک درخت عمومی بوده که تبدیل به دودویی شده است. درخت عمومی اولیه را بدست آورید.

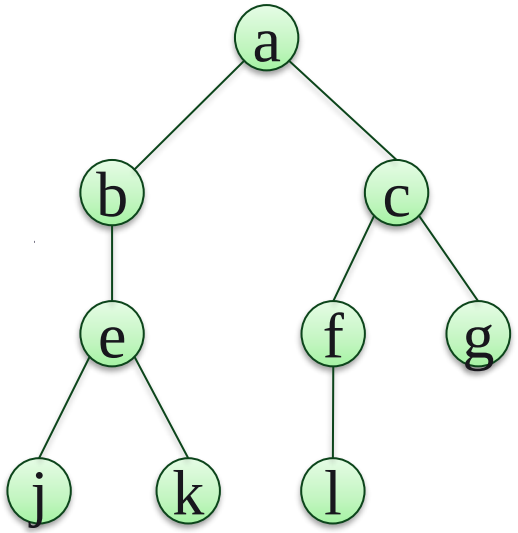


L : اشاره گر چپ ترین فرزند
R : اشاره گر به برادر سمت راست
دایره های خالی برابر null هستند.


0
1
5
8
3
9
6
7
11
12
13

نمایش درخت با استفاده از آرایه

- ✓ این پیاده سازی بیش‌تر در درخت های درجه ۲ مورد استفاده قرار می‌گیرد.
- ✓ در این روش اندیس هـای آرایـه از یـک آغـاز می شود و ریشـه ی درخت در این خانه قرار داشته باشد.
- ✓ دو فرزند گـره ی i، در خانـه هـای i ۲ و ۱ + i ۲ آرایـه قـرار می گیرند.
- ✓ NOE تعداد عناصر درخت.



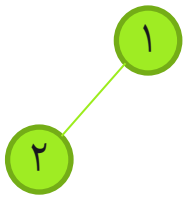
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
a	b	c	NULL	e	f	g	NULL	NULL	j	k	l					

NoE

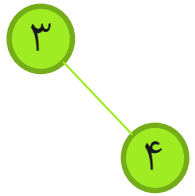
درخت دودویی

درختی از درجه‌ی ۲ است به طوری که فرزند چپ و فرزند راست هر گره، از لحاظ موقعیت قرارگیری با یکدیگر متفاوت هستند.

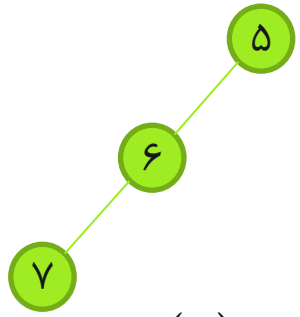
- درخت دودویی، درختی مرتب است.
- نکته: درخت (الف) و (ب) متمایز هستن.
- نکته: مفاهیم فرزند چپ با چپ‌ترین فرزند یکی نیست .



(الف)

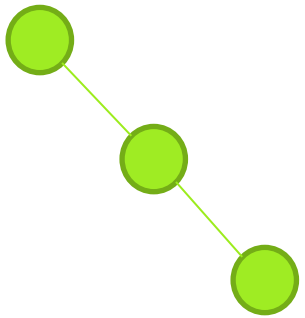


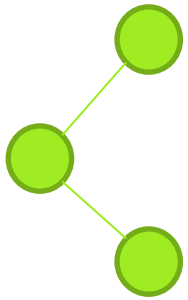
(ب)

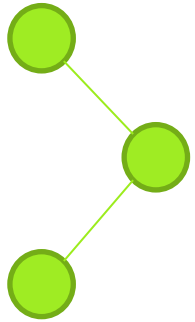


(پ)

RC(3)= 4
LC(3)= Null
LMC(3)= 4


(ت)


(ث)


(ج)

پیاده‌سازی درخت دودویی پیوندی

نوع داده ی انتزاعی درخت دودویی پیوندی

```
Type TREE = Record
{
        Data : ElementType;
        RC : ^TREE;
        LC : ^TREE;
}
Type Position = ^TREE;
Const NULL = 0;

INSERT( ref T : TREE ; x : ElementType );
DELETE( ref T : TREE ; Node : Position );
PARENT( T : TREE ; Node : Position ) : Position
LMC( T : TREE ) : Position
SIZE( T : TREE ) : Integer
EMPTY( T : TREE ) : Boolean
MAKENULL( ret T : TREE );
HEIGHT( T : TREE ) : Integer
```

عملیات مربوط به درخت دودویی پیوندی

```
SIZE ( T : TREE ) : Integer
{
        if T == NULL then
                return 0;

        return 1 + SIZE(T^.RC) + SIZE(T^.LC);
}


LMC ( T : TREE ) : Position
{
        if T^.LC != NULL then
                return T^.LC;
        else
                return T^.RC;
}
```

```
EMPTY ( T : TREE ) : Boolean
{
        if T != NULL then
                return FALSE;

        return TRUE;
}
```

```
MAKENULL ( ref T : TREE )
{
        if T == NULL then
                return;

        MAKENULL(T^.RC);
        MAKENULL(T^.LC);
        dispose(T);          //Free allocated memort
}
```

```
HEIGHT( T : TREE ) : Integer
{
        if T == NULL then
                return 0;

        return 1+ max( HEIGHT(T^.LC), HEIGHT(T^.RC));
}

PARENT( T : TREE ; Node : Position ) : Position
{
        if T == NULL or T == Node then
                return NULL;

        if Node == T^.RC or Node == T^.LC then
                return T;

        var temp : Position;
        temp = PARENT( T^.LC, Node);
        if temp == NULL then
                temp = PARENT( T^.RC, Node);
        return temp;
}
```

تابع	پیچیدگی زمانی
LMC	$O(1)$
PARENT	$O(n)$
SIZE	$\Theta(n)$
EMPTY	$O(1)$
MAKENULL	$\Theta(n)$
HEIGHT	$\Theta(n)$

پیاده سازی درخت عمومی پیوندی به وسیله ی درخت دودویی معادل

بدنـــه‌ی توابـــع EMPTY، SIZE و MAKENULL دقیقاً مشابه درخت دودویی پیوندی است.

نکته: به طور کلی هر درخت عمومی را می‌توان به عنوان یک درخت دودویی در نظر گرفت؛ ولی تنها در صورتی درخت دودویی معادل یک درخت عمومی اسـت کـه ریشـه‌ی آن فرزنـد راست نداشته باشد.

```
LMC( T : TREE ) : Position
{
        return T^.LC;
}
```

```
PARENT( T : TREE ; Node : Position ) : Position
{
        if T == NULL or T == Node then
                return NULL;

        var p, q : TREE;
        p = T^.LC;
        while p != NULL do
        {
                if p == Node then
                        return tree;
                p = p^.RC;
        }
        p = T^.LC;
        while p != NULL do
        {
                q = PARENT( p, Node );
                if q != NULL then
                        return q;
                p = p^.RC;
        }
        return NULL;
}
```

پیاده‌سازی درخت دودویی با آرایه

1) ریشه در خانه‌ی شمارهی یک آرایه به نام Elements قرار دارد.
2) اگر عنصری در خانهی i آرایه‌ی Elements قرار داشته باشد : فرزند چپ آن در صورت وجود در خانه خانه‌ی ۲i و فرزند راست آن در صورت وجود در خانه ی ۱ + ۲i قرار خواهـد داشت.
3) اگر عنصری در خانه‌ی j آرایه‌ی Elements باشد، پدر آن در خانه‌ی $\left\lfloor \frac{j}{2} \right\rfloor$ قرار دارد.
4) در صـــورتی کـــه خانه ی i فاقـــد گـــره ای از درخـــت باشـــد، در آن مقـــدار NULL قـــرار می‌گیرد.
5) NOE نشان دهنده گره ای از درخت با بزرگ ترین اندیس اسـت و گـره ای بعـد از آن وجود ندارد.

نکته: از ایرادات این طراحی، محدود بودن تعداد گره های قابل ذخیره سازی بـه دلیـل ایسـتا بودن ساختار آرایه است.

```
Type TREE = Record
{
        Elements : ElementType[MAX];
        NOE : Position;
}

Type Position = Integer;



LC ( p : Position ) : Position
{
        return p * 2;
}

RC ( p : Position ) : Position
{
        return p * 2 + 1;
}
```

? برای ذخیره سازی درخت دودویی با n گره در آرایه به روش گفته شده، در بهترین حالت و بدترین حالت نیاز به آرایه با چه طولی است؟

✔ بهترین حالت زمانی رخ می دهد که در درخت تکمیل باشد. در این حالت آرایه ای به طول n کافی است.
بدترین حالت زمانی رخ می دهد که درخت، اریب به راست باشد. در این حالت خواهیم داشت :

index(i) ← اندیس گره ی i ام درخت :

$$\text{index}(1) = 1$$
$$\text{index}(2) = 3$$
$$\text{index}(3) = 7$$
.
.
.
$$\text{index}(n) = 2^n - 1$$

پس در بدترین حالت به آرایه ای با $2^n - 1$ خانه نیاز است که $2^n - 1 - n$ خانه ی آن NULL بوده و تنها n خانه ی آن مقدار دارد.

پیاده‌سازی درخت عمومی با آرایه

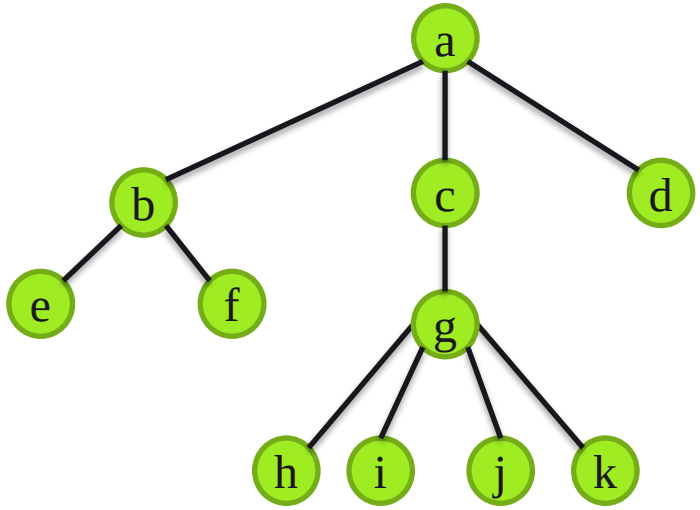
درپیاده سازی درخت عمومی با استفاده از آرایه،به دو ارایه نیاز است:

1. **Elements**: داده‌های موجود درگره‌های درخت را ذخیره می‌کند.
2. **Parents**: آرایه‌ای که اندیس پدر هر گره را مشخص می‌نماید.

قوانین زیر در مورد این طراحی برقرار است:

1. ریشه‌ی درخت درخانه‌ی شماره ی یک آرایـه‌ی Elements قـرار مـی‌گیـرد و در خانـه‌ی شماره ی یک آرایه‌ی Parent، مقدار صفر قرار می گیرد که نشان دهنده ی ریشه بودن این عنصر است(Parents[۱]=NULL).
2. اگر پدر گره ی i در خانه‌ی j باشد آنگاه ، parents[i] = j.
3. اگر گره ی موجود در خانه‌ی j، پدر گره ی موجود در خانه‌ی i باشد، آنگاه i>j.
4. اگر گره ی i برادر سمت راست گره ی j باشد، آنگاه i>j.
5. NOE نشان دهنده ی اندیس گره ای از درخت با بزرگ ترین اندیس در آرایه است.

برخلاف پیاده سازی درخت دودویی با آرایه که آرایه‌ای منحصر به فرد را نتیجه می‌دهد، در این مورد آرایه منحصربه فرد نیست و بسته به ترتیب درج عناصر، محل گره‌ها در آرایه متفاوت خواهد بود.



آرایه های زیر ، هر دو معادل یک درخت عمومی هستند.

	1	2	3	4	5	6	7	8	9	10	11	12
Elements	a	b	c	d	e	f	g	h	i	j	K	
Parents	0	1	1	1	2	2	3	7	7	7	7	

NoE

	1	2	3	4	5	6	7	8	9	10	11	12
Elements	a	b	e	f	c	g	h	i	d	j	k	
Parents	0	1	2	2	1	5	6	6	1	6	6	

NoE

عملیات مربوط به درخت عمومی آرایه ای

- الگوریتم های این ساختار به دلیل تفاوت های عمده ی آن با سایر طراحی ها، تا حـدی اختصاصـی است.

- ابتدا نوع داده ی انتزاعی و ساختمان داده ی مربوط به درخت عمومی آرایه ای بیان می شود و در ادامه عملیات مربوط به آن مطرح خواهد شد.

```
Type GTREE = Record
{
        Elements, Parents : ElementType[MAX];
        NOE : Position;
}

Type Position = Integer;
```

```
INSERT( ref T : GTREE ; x : ElementType );
DELETE( ref T : GTREE ; Node : Position );
LMC( T : GTREE ; Node : Position ) : Position
PARENT( T : GTREE ; Node : Position ) : Position
CREATE( ref T : GTREE  );
EMPTY( T : GTREE ) : Boolean
MAKENULL( ref T : GTREE);
RS( T : GTREE ; Node : Position ) : Position
JOIN( Root : ElementType ; A, B : GTREE ) : GTREE
```

```
LMC( T : GTREE ; Node : Position ) : Position
{
        for I = Node + 1 to T.NOE do
                if T.Parents[i] == Node then
                        return i;

        return NULL;
}

PARENT( T : GTREE ; Node : Position ) : Position
{
        return T.Parents[Node];
}

RS( T : GTREE ; Node : Position ) : Position
{
        for i = Node + 1 to T.NOE do
                if T.Parents[i] == T.Parents[Node] then
                        return i;

        return NULL;
}
```

```
JOIN( Root : ElementType ; A, B : GTREE ) : GTREE
{
        var T : GTREE;
        T.Elements[1] = Root;
        T.Parents[1] = NULL;
        for i = 1 to A.NOE do
        {
                T.Elements[i+1] = A.Elements[i];
                T.Parents[i+1] = A.Parents[i] + 1;
        }
        for i = 1 to B.NOE do
        {
                T.Elements[i+1+ A.NOE] = B.Elements[i];
                T.Parents[i+1+ A.NOE] = B.Parents[i] + A.NOE + 1;
        }

        T.Parents[A.NOE + 2] = 1;
        T.NOE = A.NOE + B.NOE + 1;

        return T;
}
```

- تابع JOIN پیاده سازی شده در اسلاید قبل دو درخت عمومی آرایه ای را با یکدیگر ادغام می کند،
بدین صورت که گره ی Root را به عنوان ریشه ی درخت جدید قـرار داده و درخت های عمومی A و B را به در جایگاه فرزندان این گره قرار می دهد .

- تابع RS برای یک گره ی مشخص فرزند بعدی پدرش را باز می‌گرداند .
به عبارت دیگـر، ایـن تابع گره ی برادر یک گره را به عنوان خروجی می دهد.

- تابع LMC نیز چپ‌ترین گره ی فرزند یک گره ی ورودی را باز می‌گرداند.

پیمایش درخت

- **پیمایش درخت عبارت است از ارائه یک ترتیب خطی شامل همه‌ی عناصر درخـت.**

- در پیمـایش درخت تمامی گره های آن یک بار ملاقات می شوند، ولی ترتیب ملاقات گره ها، بسـتگی بـه نـوع پیمایش دارد و در هر نوع پیمایش، لیست خطی متفاوتی را نتیجه می‌دهد.

- نکته: پیمایش یک درخت، یکتا نیست، به این معنی که ممکن اسـت نتیجـه‌ی پیمـایش دو درخت متفاوت، یکسان باشد.

- اگر V به معنی ملاقات گره، R به معنی حرکت به فرزند راسـت و L بـه معنـی حرکـت بـه فرزند گره ی چپ باشد، با ترکیب این ۳ حرف، ۶ نوع پیمـایش مختلـف مـی‌تـوان تولیـد کـرد. (RVL، LVR، VLR ، VRL، RLV،LRV)

- حال اگر فقط پیمایش هایی در نظر گرفته شوند که L پیش از R آمده است (با لویت از چپ به راست) تنها ۳ پیمایش بـه دسـت خواهـد آمـد : LVR ،LRV، VLRکه به ترتیب به‌آن ها پیمایش میان ترتیب ، پس ترتیب و پیش ترتیب گفته می شود.

پیمایش میان ترتیب

تعریف بازگشتی: اگر T درختی با ریشه ی r بوده وT1,T2,…,TK به ترتیب از چپ به راسـت زیردرخت‌های آن باشند، پیمایش‌میان ترتیب درخت T به صورت زیر خواهد بود:

$$T_1 r T_2 T_3 \dots T_k$$

که در آن Ti پیمایش **میان ترتیب** زیردرخت i ام است.

اگر درخت T دودویی باشد، این پیمایش عبارت خواهد بود از :

$$LrR$$

که در آن L پیمایش میان ترتیب زیردرخت چـپ و R پیمـایش میـان ترتیـب زیردرخـت راسـت است .

```
Inorder ( T : TREE )
{
        if T == NULL then
                return;
        Inorder( T^.LC );
        Write( T^.Data);
        Inorder( T^.RC );
}
```

پیمایش پیش ترتیب

تعریف بازگشتی: اگر T درختی با ریشه ی r بوده و T1,T2,…,TK به ترتیب از چپ به راسـت زیردرخت‌های آن باشند، پیمایش پیش ترتیب درخت T به صورت زیر خواهد بود:

$$rT_1T_2T_3 \dots T_k$$

که در آن Ti پیمایش **پیش ترتیب** زیردرخت iام است.

اگر درخت T دودویی باشد، این پیمایش عبارت خواهد بود از :

$$rLR$$

که در آن L پیمایش پیش ترتیب زیردرخت چـپ و R پیمـایش پیش ترتیـب زیردرخـت راسـت است .

```
Preorder ( T : TREE )
{
        if T == NULL then
                return;
        Write( T^.Data );
        Preorder( T^.LC );
        Preorder( T^.RC );
}
```

پیمایش پس ترتیب

تعریف بازگشتی: اگر T درختی با ریشه ی r بوده و T1,T2,…,TK به ترتیب از چپ به راست زیردرخت‌های آن باشند، پیمایش پس‌ترتیب درخت T به صورت زیر خواهد بود:

$$T_1 T_2 T_3 \dots T_k r$$

که در آن Ti پیمایش **پس ترتیب** زیردرخت iام است.

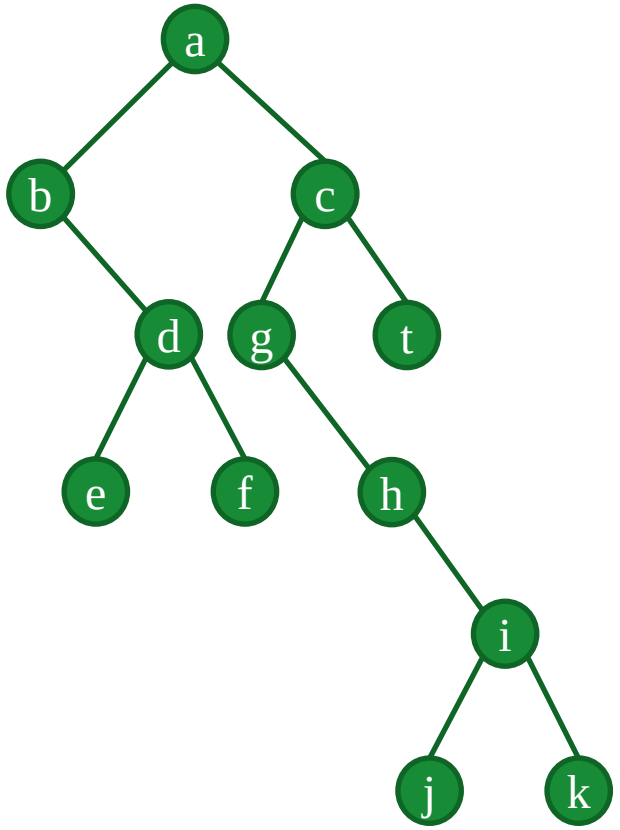
اگر درخت T دودویی باشد، این پیمایش عبارت خواهد بود از :

$$LRr$$

که در آن L پیمایش پس ترتیب زیردرخت چپ و R پیمایش پس ترتیب زیردرخت راست است .

```
Postorder ( T : TREE )
{
        if T == NULL then
                return;
        Postorder( T^.LC );
        Postorder( T^.RC );
        Write( T^.Data );
}
```

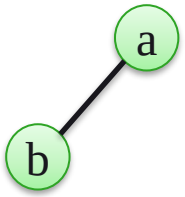
مثال



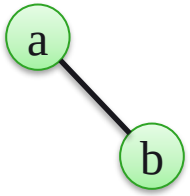
$$Inorder(T) = b\ e\ d\ f\ a\ g\ h\ j\ i\ k\ c\ t$$
$$Preorder(T) = a\ b\ d\ e\ f\ c\ g\ h\ i\ j\ k\ t$$
$$Postorder(T) = e\ f\ d\ b\ j\ k\ i\ h\ g\ t\ c\ a$$

نکته: اگر یک درخت عمومی به صورت درخت دودویی معادل نمایش داده شود، پیمایش پیش ترتیب آن ها مشابه خواهد بود، ولی این امر در مورد سایر پیمایش ها (میان ترتیب و پس ترتیب) صدق نمی کند.

نکته : پیمایش میان ترتیب یک درخت عمومی از درجه ۲ و یک درخت دودویی با گره های یکسان، الزاما برابر نخواهد بود (شکل زیر). این تفاوت به دلیل تفاوت در تعریف چپ ترین فرزند و فرزند چپ در دو درخت است که در مباحث قبلی شرع داده شد.



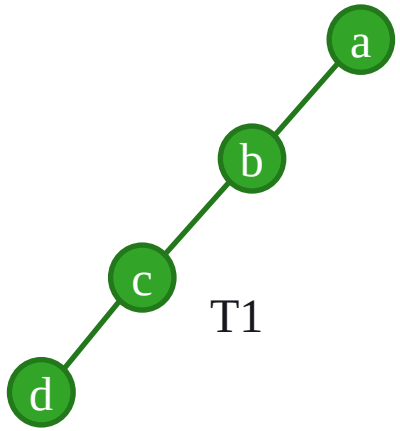
$$General\ Tree \rightarrow inorder(T) = b\ a$$
$$Binary\ Tree \rightarrow inorder(T) = b\ a$$



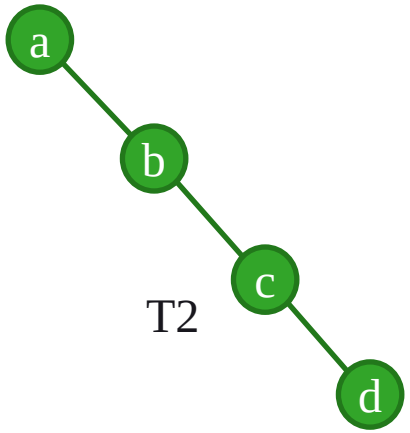
$$General\ Tree \rightarrow inorder(T) = b\ a$$
$$Binary\ Tree \rightarrow inorder(T) = a\ b$$

? کدام یک از پیمایش های میان ترتیب، پیش ترتیب، پس ترتیب برای دو درخت دودویی مورب خطی راست و مورب خطی چپ با گره های یکسان در هر سطح، یکسان خواهد بود؟

همانطور که از پیمایش های درخت های شکل زیر مشخص است، در درخت مورب خطی راست، پیمایش های پیش ترتیب و میان ترتیب، نتیجه ی یکسانی خواهند داشت. همین امر در درخت مورب خطی چپ و پیمایش های پس ترتیب و میان ترتیب نیز صادق است.



T1



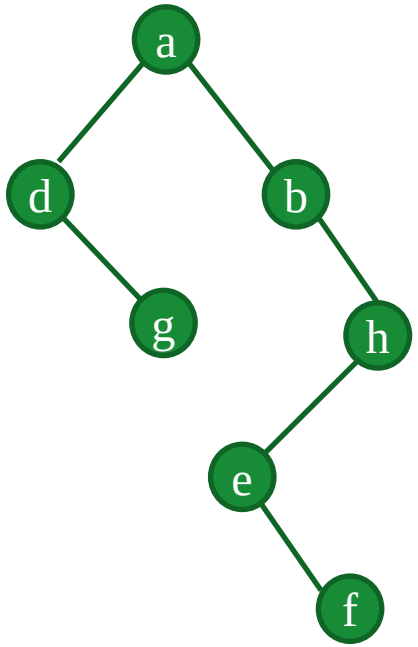
T2

$$preorder(T_2) = a\ b\ c\ d$$
$$inorder(T_2) = a\ b\ c\ d$$

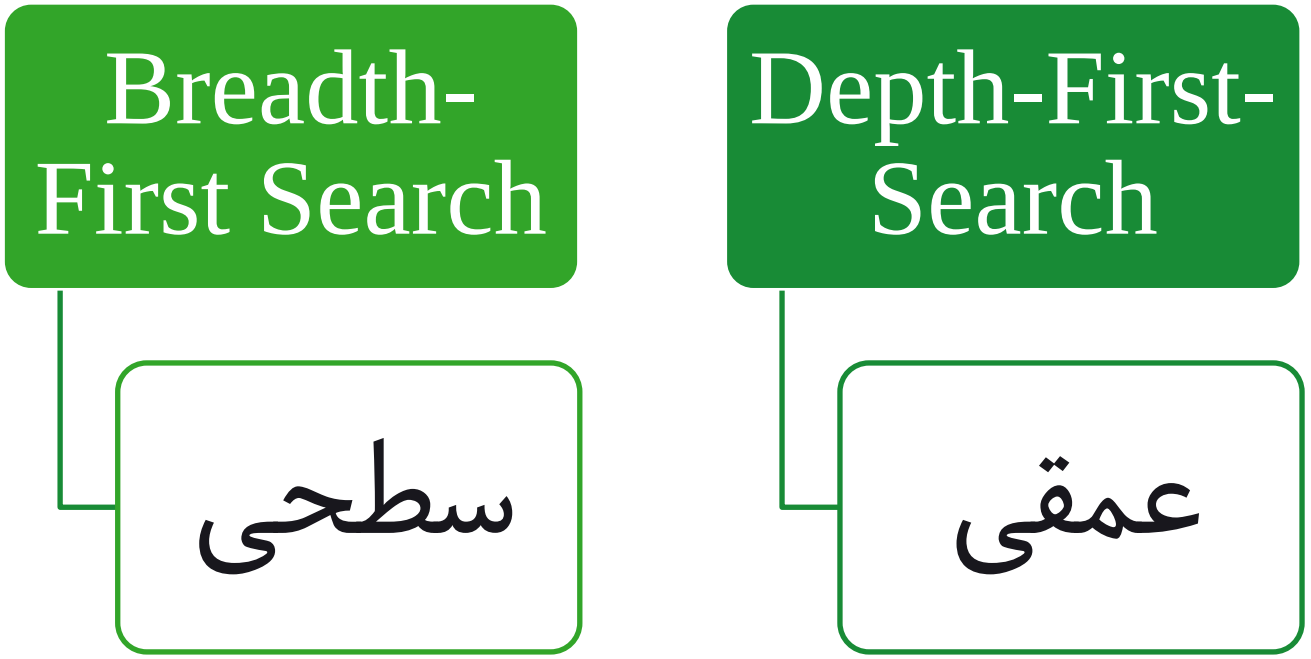
$$postorder(T_1) = d\ c\ b\ a$$
$$inorder(T_1) = d\ c\ b\ a$$

? درخت معادل را رسم کنید.

Inorder : d g a b e f h
Preorder : a d g b h e f



جستجو (از دیدگاه پیمایش)



جستجوی عمقی

مراحل این پیمایش به صورت زیر است :

1. گره ی جاری به عنوان ریشه در نظر گرفته می شود.
2. گره ی جاری ملاقات می شود.
3. اگر گره ی جاری، فرزند ملاقات نشده ای دارد، از بین آن ها فرزندی که اولویت بیش تری دارد و قبلاً ملاقات نشده است ملاقات شده و به عنوان رأس جاری در نظر گرفته می شود. مرحلهٔ ۳ تکرار می‌شود .
4. اگر تمامی فرزندان رأس جاری ملاقات شده اند، رأس ملاقات شده قبل از رأس جاری به عنوان رأس جاری در نظر گرفته می شود. پرش به مرحلهٔ ۳ .
5. اگر امکان انجام مرحله ی ۴ نباشد، پایان الگوریتم.

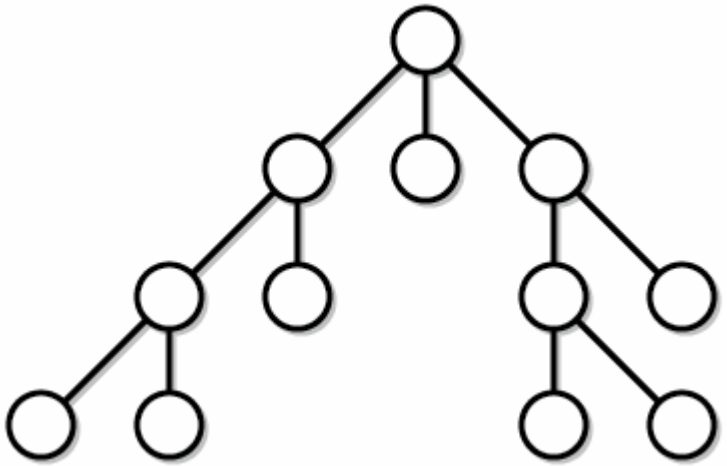
```
NR-DFS ( T : TREE )
{
        if T == NULL then
                return;
        var Current : Node;
        var S : Stack;
        S.PUSH(S, T);
        while EMPTY(S) == FALSE do
        {
                Current = POP(S);
                Write ( Current^.Data );
                PUSH (S, all children of current);
        }
}
```

```
DFS ( T : TREE )
{
        if T == NULL then
                return;

        Write ( T^.Data );

        while ( there exists a not visited child of T ) do
        {
                t = highest priority child of T and not visited;

                DFS(t);
        }
}
```

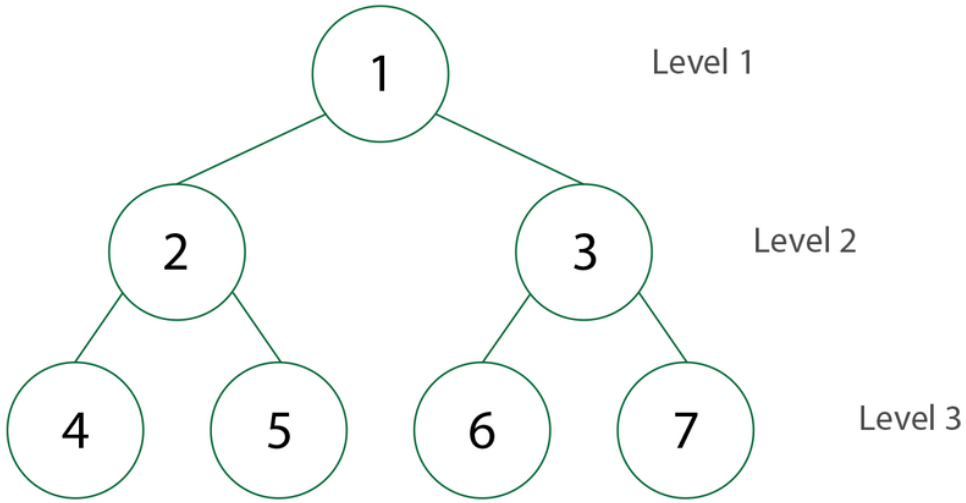


جستجوی سطحی

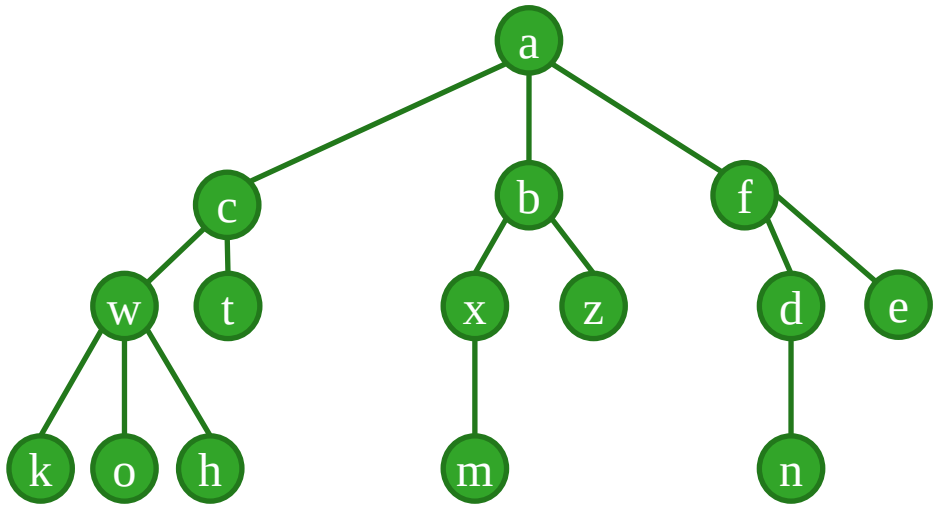
1. گره ی جاری برابر ریشه قرارگرفته و آن را ملاقات می‌کند.
2. تمامی فرزندان گره ی جاری به ترتیب اولویت ملاقات می شوند.
3. گره ی جاری برابر اولین‌گره ی ملاقات شده ای قرار داده می‌شود که فرزندانش ملاقات نشده اند. مرحله ی ۳ تکرار می‌شود.
4. اگر امکان انجام مرحله ی ۳ نباشد، پایان الگوریتم.

نکته: در جستجوی سطحی اگر اولویت فرزندان از چپ به راست باشد، ترتیب ملاقات گره ها معادل با پیمایش سطح ترتیب خواهد بود.

```
BFS ( T : TREE )
{
        if T == NULL then
                return;
        var Current : Node;
        var Q : QUEUE;
        ENQUEUE (Q, n);
        while EMPTY(Q) == FALSE do
        {
                Current = DEQUEUE (Q);
                Write( Current^.Data );
                ENQUEUE (Q, all children of current);
        }
}
```



جستجوی عمقی در درختی که اولویت گره ها از چپ به راست است، مشابه پیمایش پیش ترتیب عمل می کند.



DFS(T) = a b x m z c t w h k o f d n e
BFS(T) = a b c f x z t w d e m h k o n
Level Order(T) = a c b f w t x z d e k o h m n

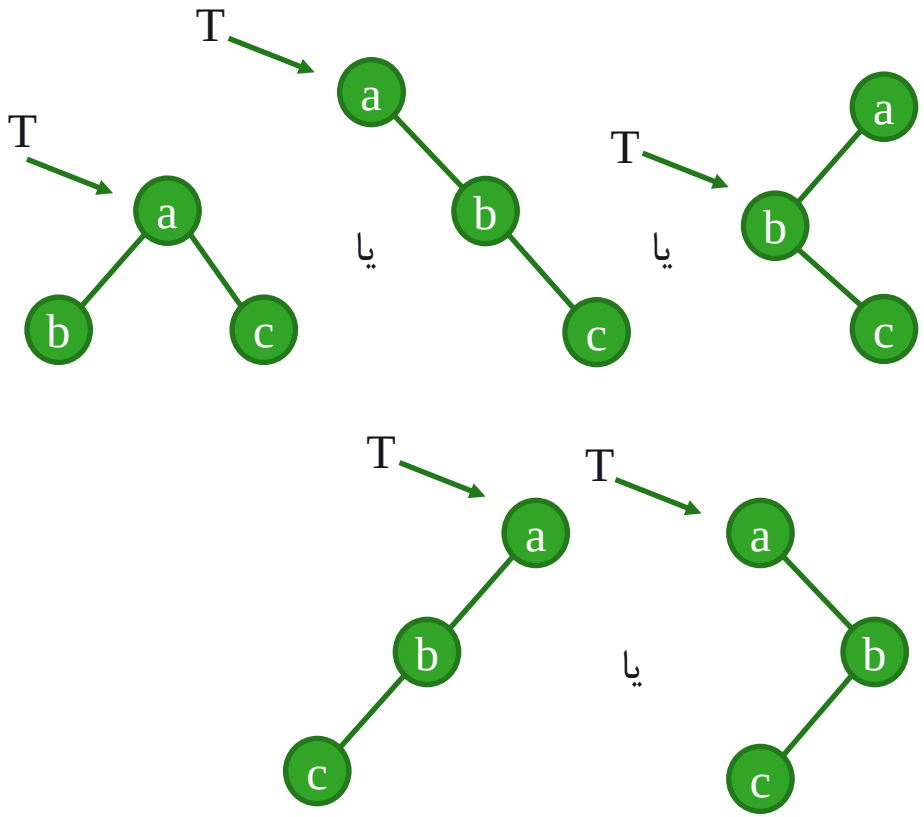
پیمایش سطح ترتیب

- ✓ در پیمایش سطح ترتیب ، گره های درخت به صورت سطر به سطر از چپ به راست ملاقات می شوند.
- ✓ با داشتن هم زمان برخی از پیمایش‌ها، در صورتی که درجه‌ی درخت برابر دو باشد، می‌توان آن را به صورت یکتا بازسازی نمود.

نکته: در صورتی که درخت از درجه ای بیش از دو باشد، به طور کلی امکان بازسازی یکتای درخت با استفاده از پیمایش‌های آن وجود ندارد، حتی اگر تمامی پیمایش های درخت مذکور موجود باشد.

شرط‌های لازم برای بازسازی:

1.درخت باید از درجه‌ی دو باشد.
2.داده های موجود در گره ها باید یکتا باشند.
3. باید بتوان با استفاده از پیمایش ها، ریشه ی درخت را به دست آورد.
4.باید بتوان با استفاده از پیمایش ها، به گره‌های مربوط زیر درخت‌های چپ و راست درخت را به دست آورد.



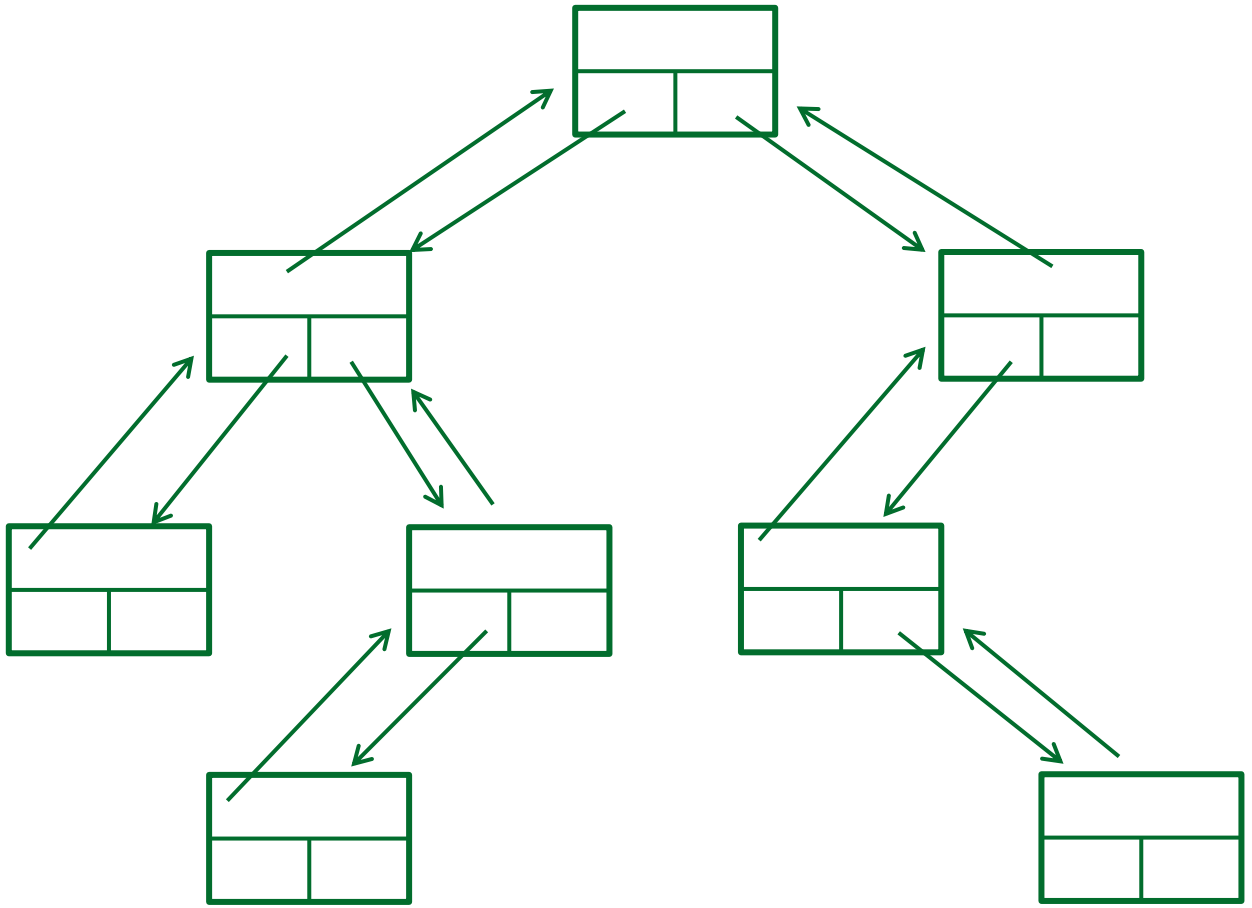
Preorder (T) = a b c

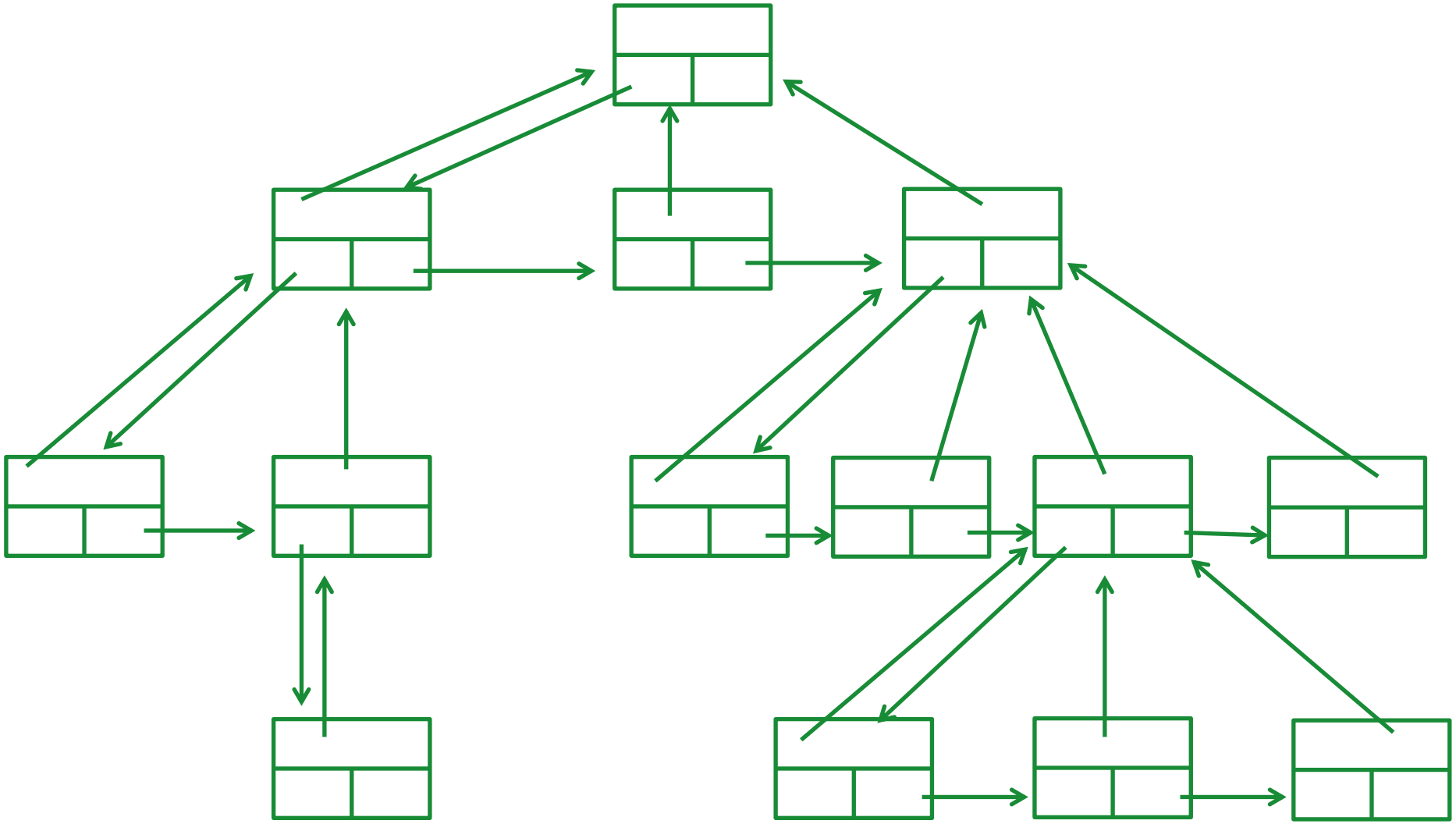
طراحی بهینه درخت‌های ساخته شده با اشاره گر

- ✓ در طراحی یک درخت، گاهی می‌توان در هر گره اشاره‌گرهای اضافی قرار داد تا ثابت زمانی الگوریتم ها کاهش یابد.
- ✓ همچنین می توان با استفاده از اشاره گرهای بدون استفاده ی موجود در برگ ها علاوه بر بهبود ثابت زمانی، کارایی را نیز افزایش داد .

اشاره‌گر به پدر

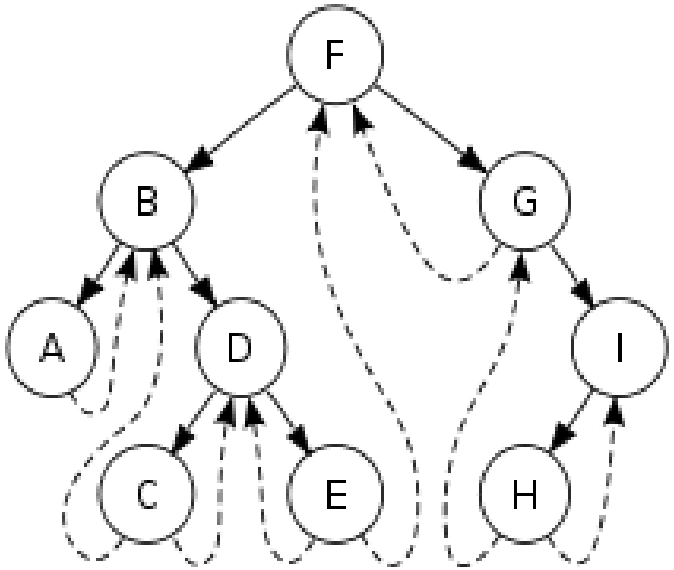
برای دسترسی به گره پدر می توان مرتبه زمانی $O(n)$ را به $O(1)$ تغییر داد با استفاده از اشاره گر هر گره را به پدر متصل کنیم.





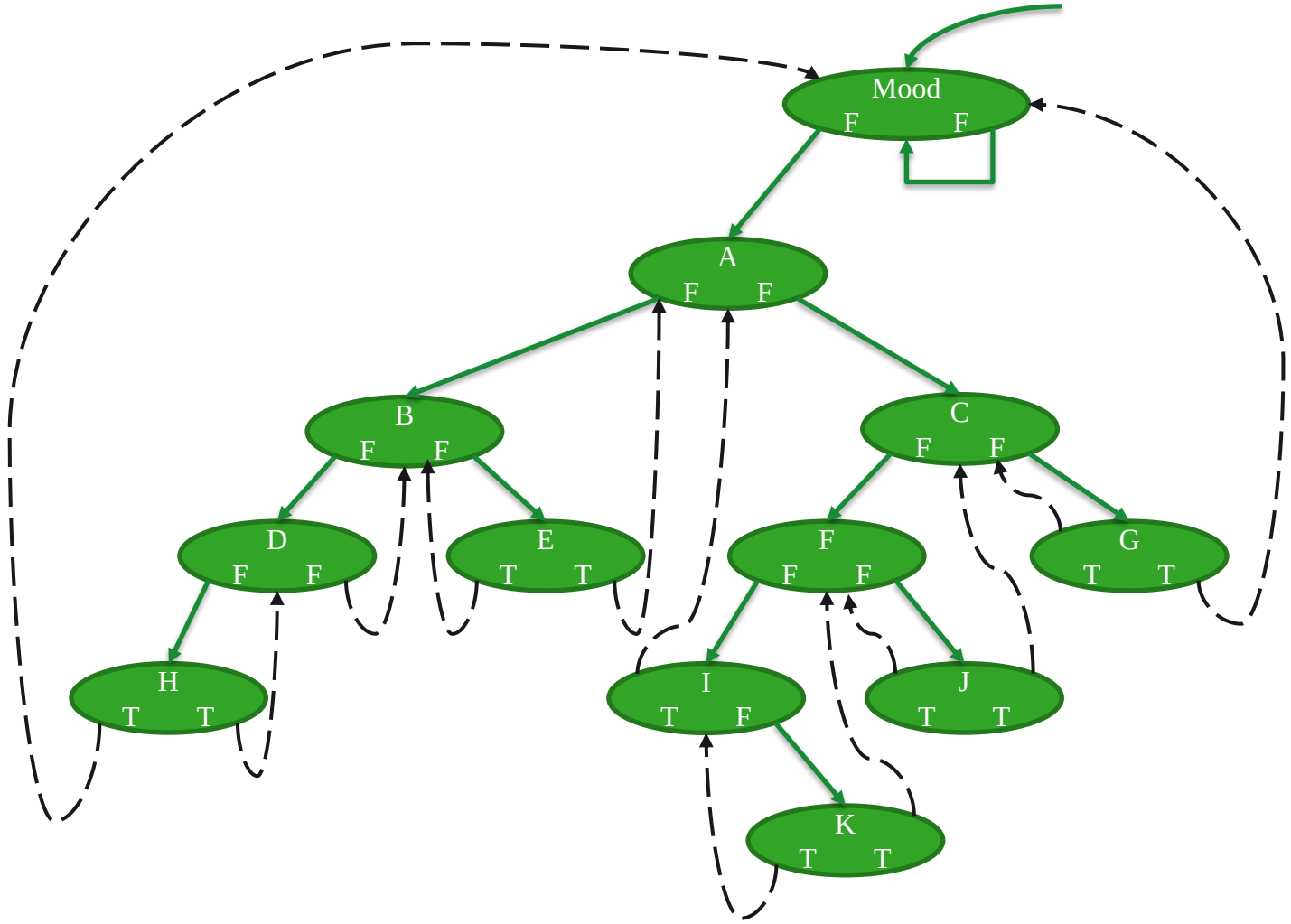
نخ‌کشی درخت دودویی

- ✓ در یک درخت دودویی با n گره، ۲n اشاره‌گر وجود دارد که تعداد n +۱ عـدد از آن‌هـا بـدون اسـتفاده هسـتند .
- ✓ در ایـده ی نخ کشی، با استفاده از اشاره گرهای بدون استفاده، ساختمان داده ای بهینه‌تر طراحی می شود .



در این طراحی:

- ✓ هر اشاره‌گر بدون استفاده ی RC، به گره ای که در پیمایش میان‌ترتیـب بعـد از گـره ی جاری ملاقات می شود، اشاره می کند.
- ✓ هر اشاره گر بدون استفاده ی LC، به گره ای که در پیمایش میان‌ترتیب پـیش از گـره ی جاری ملاقات می شود، اشاره می کند


Mood
F F
A
F F
B
F F
C
F F
D
F F
E
T T
F
F F
G
T T
H
T T
I
T F
J
T T
K
T T

حال با استفاده از درخت نخ‌کشی شده، می توان پیمایش میان‌ترتیب را به صورت غیربازگشتی انجام داد .

```
ThreadedInorder ( T : TREE )
{
        var p : Position;
        p = NEXT(T);
        while p != Root do
        {
                Write ( p^.Data );
                p = NEXT(p);
        }
}
```

تابع NEXT گره ی بعدی گره ی جاری را در پیمایش میان‌ترتیب باز می‌گرداند .

```
NEXT ( Node : Position ) : Position
{
        var p : Position;
        p = Node^.RC;
        if Node^.RightIsThread == FALSE then
                while p^.LeftIsThread == FALSE do
                        p = p^.LC;
        return p;
}
```

برای تشخیص نخ بودن یک بیت برای هر فرزند cellType اضافه می کنیم. اگر 1 بود نخ است و اگر 0 بود نخ نیست.

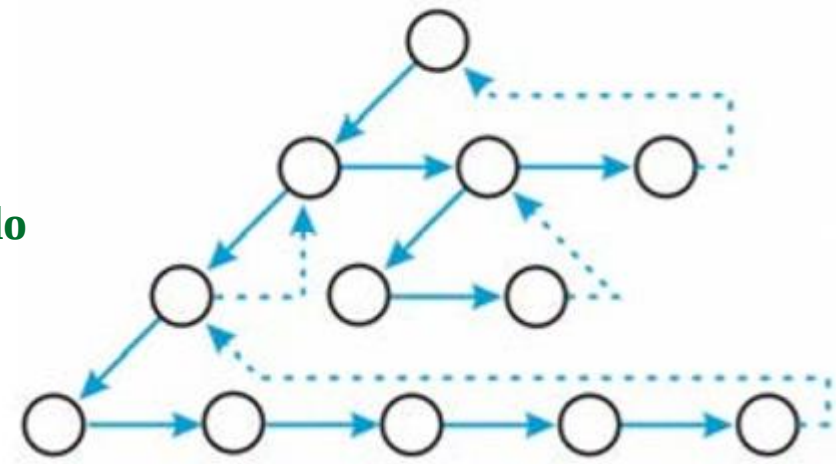
RightIsThread
LeftIsThread

نخ‌کشی درخت عمومی

با این نخ کشی، الگوریتم یافتن پدر یک گره، به صورت شبه کد زیر در می‌آید و پیچیـدگی زمانی آن از مرتبه‌ی O(n) تبدیل به O(k) می شود کـه k حـداکثر تعـداد فرزنـدان هـر گـره است.

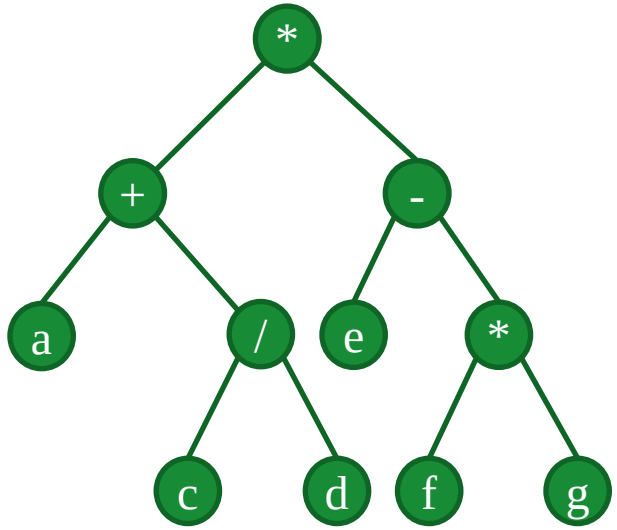
```
PARENT( T : TREE )
{
        var p : Position;
        p = T;
        while p^.RightIsThread == FALSE do
                p = p^.RC;

        return p^.RC;
}
```



درخت عبارت

درخت عبارت، درختی دودویی است که در آن برگ‌ها حاوی عملوند و گـره‌هـای میـانی حـاوی عملگر هستند.



$$Inorder(T) = a + c/d * e - f * g$$
$$Preorder(T) = * + a/cd - e * fg$$
$$Postorder(T) = a\ c\ d/ + e\ f\ g * - *$$

پایان