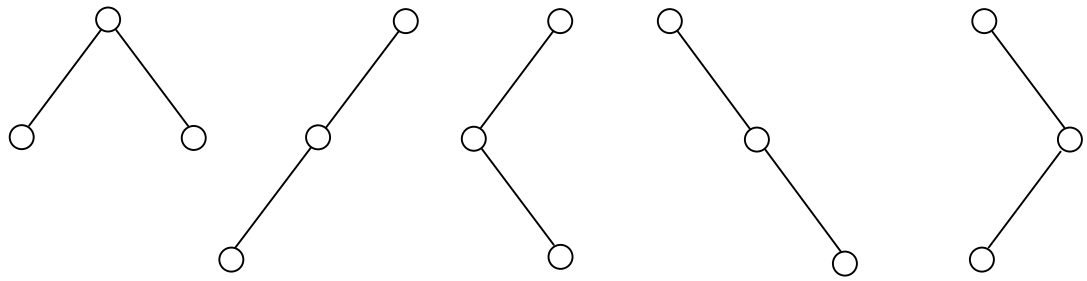
درخت (Tree)

درخت مجموعه‌ای است متناهی از یک یا چند گره که یک گره خاص را بنام ریشه مشخص کرده‌ایم و سایر گره‌ها به مجموعه‌های مجزایی تقسیم می‌شوند که هر مجموعه خود یک درخت است و زیر درخت ریشه نامیده می‌شود. تعداد زیر درخت‌های هر گره درجه آن گره است. فاصله هر گره تا ریشه درخت را سطح آن گره می‌نامند. بزرگترین درجه گره در درخت ، درجه درخت نامیده می‌شود. اگر درجه درخت m باشد درخت را m تایی می‌گویند. به گره‌هایی که درجه آنها صفر است برگ (Leaf) گفته می‌شود. برگها زیر درخت ندارند. برگها را گره‌های خارجی درخت و سایر گره‌ها غیر از برگها را گره‌های داخلی درخت می‌نامند. دو گره که دارای پدر مشترک هستند را گره‌های همزاد گویند. حداکثر سطح یک گره در درخت را ارتفاع (عمق) درخت گویند. پیش فرض سطح ریشه ۱ است.

درخت دودویی طبق تعریف درختی است که درجه آن ۲ باشد یعنی هر گره حداکثر ۲ فرزند داشته باشد. یکی فرزند سمت راست و یکی فرزند سمت چپ که با هم متفاوت (متمایز) هستند.

فرزند سمت چپ است　　فرزند سمت راست است

مثال : با سه گره چند درخت دودویی می‌توان ساخت.

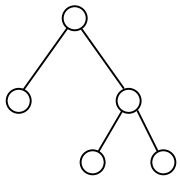


درخت اریب　　درخت اریب

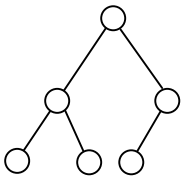
- **درخت Perfect (کاملاً پر)** : درختی است که همه گره‌ها بجز گره‌های سطح آخر (برگها) دارای حداکثر فرزندان بوده (حداکثر درجه درخت) و برگها هم سطح نیز باشند.
- **درخت Complete (کامل)** : درختی است که اگر گره‌های آنرا شماره‌گذاری کنیم ،

شـماره‌هـا بر درخت Perfect متناظرش منطبق باشند. یعنی می‌توان گفت درختی است که اگـر ارتفـاع آن d باشـد تـا ارتفـاع d - 1 ، درخـت Perfect بوده و برگها در سطح d تا حد ممکن در سمت چپ باشند.

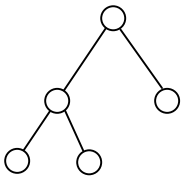
- **درخت Full (پر)** : درختی که در آن گره‌ها یا برگ هستند و یا به تعداد درجه درخت فرزند دارند را درخت full گویند.



Full است
Complete نیست
Perfect نیست



Complete است
Full نیست
Perfect نیست

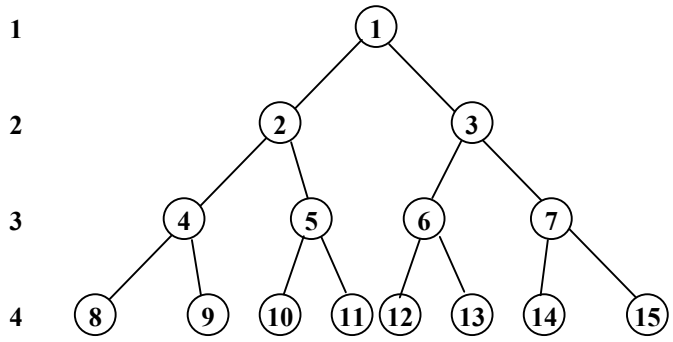


Perfect نیست
Full است
Complete است

- درختی که Perfect باشد حتماً Complete و Full هم هست.
- درختی که Complete است لزوماً Perfect نیست.
- درختی که Complete است لزوماً Full نیست.
- درختی که Full است لزوماً Complete نیست.

- حداکثر تعداد گره‌ها در یک درخت دودویی به ارتفاع d براب با $2^d - 1$ خواهد بود.
- حداثر تعداد برگها در یک درخت دودویی به ارتفاع d برابر با 2^{d-1} خواهد بود.
- حداکثر تعداد گره‌های غیر برگ یک درخت با ارتفاع d برابر با $2^{d-1} - 1$ خواهد بود.
- درختی دودویی با n گره دارای ارتفاع Log_2^{n-1} خواهد بود.

سؤال : درخت دودویی زیر را در نظر بگیرید به سؤالهای آن پاسخ دهید.



۱- حداکثر چند برگ وجود دارد؟

حداکثر تعداد برگها از رابطه 2^{d-1} بدست می‌آید. اگر بعنوان مثال ارتفاع 4 را در نظر بگیریم داریم :

حداکثر تعداد برگهای موجود در این درخت $2^{d-1} = 2^{4-1} = 2^3 = 8$

۲- حداکثر چند گره غیر از برگ داریم؟ (گره‌های داخلی)

حداکثر گره‌های داخلی از رابطه $2^{d-1} - 1$ بدست می‌آید. باز هم در ارتفاع 4 داریم :

$$2^{d-1} - 1 = 2^{4-1} - 1 = 2^3 - 1 = 8 - 1 = 7$$

۳- حداکثر چند گره وجود دارد؟

حداکثر تعداد گره‌ها از رابطه $2^d - 1$ بدست آمده و بعنوان مثال باز هم در ارتفاع 4 داریم :

$$2^d - 1 = 2^4 - 1 = 16 - 1 = 15$$

۴- چند درخت کامل متمایز به ارتفاع d داریم؟

حداکثر تعداد درخت کامل متمایز نیز از همان رابطه تعداد برگها بدست می‌آید. پس در ارتفاع 4 داریم :

$$2^{d-1} = 2^{4-1} = 2^3 = 8$$

سؤال : اگر n تا گره داشته باشیم :

الف) حداکثر عمق چقدر است؟

حداکثر عمق برابر با n خواهد بود. در این حالت درخت بصورت کاملاً اریب خواهد بود. یعنی تمام فرزندان از یک سمت (چپ یا راست) رشد می‌کنند.

ب) حداقل عمق چقدر است؟

حداقل ارتفاع یک درخت دودویی با n گره از رابطه زیر بدست می‌آید :

$$\left[\mathrm{Log}_2^n\right] + 1 \Rightarrow \left[\mathrm{Log}_2^8\right] + 1 = 3 + 1 = 4$$

نکته : همه روابط گفته شده برای درخت‌های m تایی نیز قابل تعمیم است.

اگر n_0 تعداد برگهای در یک درخت دودویی و n_2 تعداد گره‌های دو فرزندی باشند رابطه $n_0 = n_2 + 1$ برقرار است.

روشهای پیمایش درخت

۱- آرایه

برای نمایش درخت‌های دودویی می‌توان از آرایه‌ها استفاده کرد. بدین منظور به تعداد گره‌های درخت کامل متناظر با درخت مفروض برای یک آرایه حافظه نیاز داریم . در آنصورت داریم :

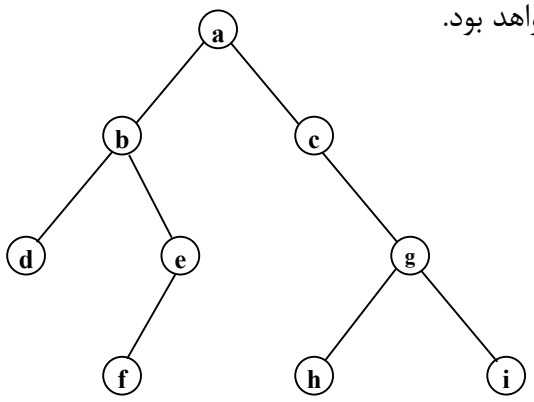
- ریشه در خانه اول آرایه قرار می‌گیرد.
- فرزند سمت چپ گره‌ای با اندیس i در آرایه درون خانه $2i$ قرار می‌گیرد که $2i \le n$ خواهد بود. اگر $2i > n$ بود یعنی گره i فرزند سمت چپ ندارد.

فرزند سمت چپ = $2i$

- فرزند سمت راست گره‌ای با اندیس i در آرایه درون خانه $2i + 1$ آرایه قرار می‌گیرد که $2i+1 \le n$ خواهد بود. اگر $2i+1 > n$ باشد یعنی گره i فرزند سمت راست ندارد.

فرزند سمت راست = $2i + 1$

در نمایش درخت‌های دودویی بوسیله آرایه‌ها اگر درخت کامل نباشد اتلاف حافظه خواهیم داشت ولی اگر درخت کامل باشد روش خوبی خواهد بود.



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

فرزند سمت چپ = $2i$

فرزند سمت راست = $2i + 1$

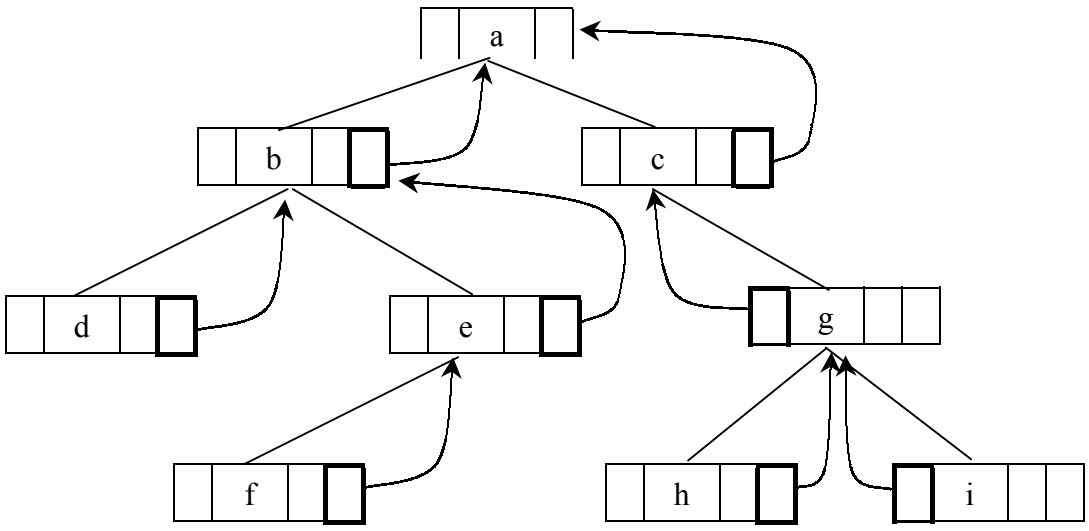
$\Rightarrow$ پدر هر گره i = $\frac{i}{2}$

۲- لیست‌های پیوندی

برای نمایش درخت‌ها به روش لیست پیوندی باید گره‌هایی با ساختار زیر تعریف کنیم.

هر گره یک فرزند سمت چپ و یک فرزند سمت راست و یک داده دارد. در ساختار تعریف شده مشخص کردن پدر هر گره به سادگی امکان پذیر نیست. برای رفع این مشکل می‌توان در ساختار هر گره یک فیلد جدید به نام Parent که به پدر آن گره اشاره می‌کند تعریف نمود.

```
Struct Treetype
{
        int data ;
        struct Treetype * Lchild ;
        struct Treetype * Rchild ;
        struct Treetype * Parent ;
}
typedef struct Treetype Tree ;
```

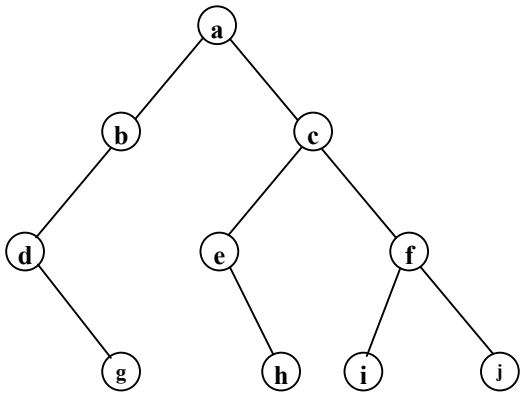


روشهای پیمایش درخت

۱- پیمایش اول عمق (عمقی)

سه روش پیمایش عمقی به شرح ذیل می‌باشد :

		سوم	دوم	اول
1.	(پیش ترتیب) Preorder	فرزند سمت راست	فرزند سمت چپ	ریشه
2.	(پس ترتیب) Postorder	ریشه	فرزند سمت راست	فرزند سمت چپ
3.	(میان ترتیب) Inorder	فرزند سمت راست	ریشه	فرزند سمت چپ



Preorder = a b d g c e h f i j

Postorder = g d b h e i j f c a

Inorder = d g b a e h c i f j

الگوریتم پیمایش عمقی به روش Inorder

```
Void  Inorder  ( Tree * T )
{
      if ( T ! = Null )
      {
            Inorder ( T → Lchild ) ;
            C out << T → data ;
            Inorder ( T → Rchild );
      }
}
```

الگوریتم پیمایش عمقی به روش Preorder

```
Void  Preorder  ( Tree * T )
{
        if ( T ! = Null )
        {
                C out << T → data ;
                Preorder  ( T → Lchild ) ;
                Preorder  ( T → Rchild );
        }
}
```

الگوریتم پیمایش عمقی به روش Postorder

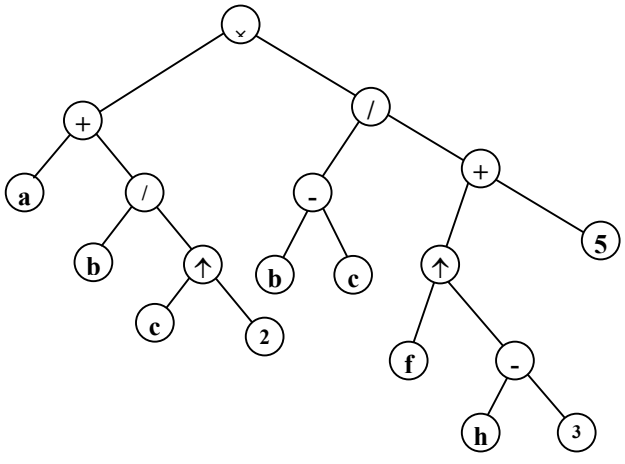
```
Void  Postorder ( Tree * T )
{
        if ( T ! = Null )
        {
                Postorder ( T → Lchild ) ;
                Postorder ( T → Rchild );
                C out << T → data ;
        }
}
```

درخت متناظر با عبارت infix

عبارت infix زیر را در نظر می‌گیریم.

$$\left(a+b/\left(c\uparrow 2\right)\times\left(b-c\right)/\left(f\uparrow\left(h-3\right)+5\right)\right)$$

هر عبارت infix یک درخت دودویی دارد.



$$Inorder = a + b / c \uparrow 2 \times b - c / f \uparrow h - 3 + 5$$

این همان عبارت infix بدون در نظر گرفتن پرانتزها است.

$$Preorder = \times + a / b \uparrow c2 / - bc + \uparrow f - h35$$

این همان عبارت prefix بدون در نظر گرفتن پرانتزها است.

$$Postorder = abc2 \uparrow / + bc - fh3 - \uparrow 5 + / \times$$

این همان عبارت postfix بدون در نظر گرفتن پرانتزها است.

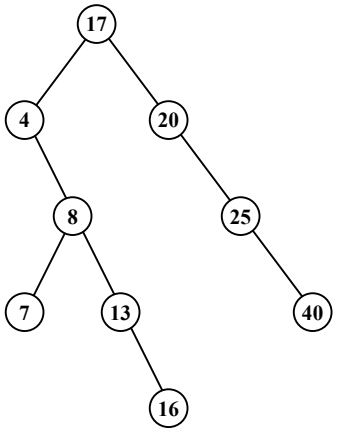
۲- پیمایش اول سطح (سطحی)

```
Void Levelorder ( Tree * T )
{
    while ( T )
    {
        C out << T → data ;
        if ( T → Lchild ) addqueue ( T → Lchild ) ;
        if ( T → Rchild ) addqueue ( T → Rchild ) ;
        T = delqueue ( ) ;
    }
}
```

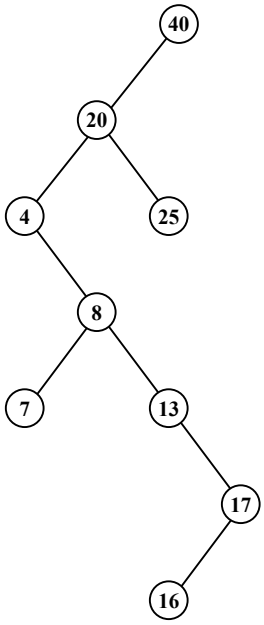
درخت جستجوی دودویی BST (Binary Search Tree)

ساختمان داده‌هایی که تا کنون بررسی شده‌اند هر یک دارای نقاط ضعفی هستند. مثلاً درج در آرایه مرتب مستلزم شیفت دادن داده‌ها و در نتیجه کندتر شدن الگوریتم است. پیمایش‌های مختلف روی لیست‌های پیوندی نیز بصورت خطی انجام می‌شود که هزینه انجام اعمال را بالا می‌برد. درخت‌های جستجوی دودویی راهکاری پیشنهاد می‌کنند که هزینه انجام اعمال اصلی مانند حذف ، اضافه و جستجو با زمان متوسط بهتری انجام می‌شود. این زمان برابر است با ارتفاع درخت که از Log_2^n تا n متغیر است. ترتیب ورود عناصر یا کلیدها برای تشکیل درخت BST از آنها کاملاً مؤثر است. کلیدهای یکسان با ترتیب متفاوت ، درخت‌های BST متفاوتی ایجاد می‌کنند.

17 20 25 40 4 8 7 13 16



40 20 4 8 13 7 17 25 16



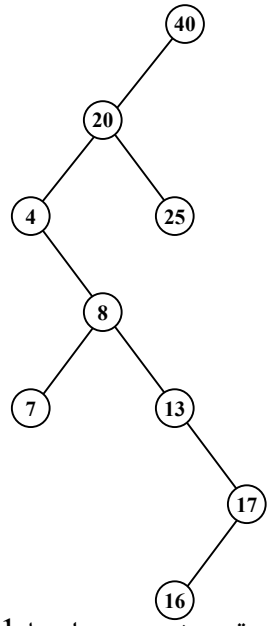
اگـر درخـت BST را بصـورت inorder پـیمایش کنـیم در خروجـی لیسـت مرتب صعودی خواهیم داشت.

هزینه ساخت یک درخت دودویی BST از یک آرایه n تایی ورودی (نامرتب) $n \times Log_2^n$ است.

الگوریتم جستجو در یک درخت دودویی BST

```
int  BSTSearch ( node * T , int * x )
{
        int founded = 0 ;
        if ( T )
        {
                if ( T → data < x )
                        founded = BSTSearch ( T → Right , x ) ;
                else if ( T → data > x )
                        founded = BSTSearch ( T → Left , x ) ;
                else founded = 1 ;
        }
        return founded ;
}
```

مثال : می‌خواهیم ببینیم عدد ۱۳ در درخت زیر وجود دارد یا خیر؟



(13) ← T	x = 13	F = ~~0~~ = 1
(8) ← T	x = 13	F = ~~0~~ = BSTSearch (13 , 13)
(4) ← T	x = 13	F = ~~0~~ = BSTSearch (8 , 13)
(20) ← T	x = 13	F = ~~0~~ = BSTSearch (4 , 13)
(40) ← T	x = 13	F = ~~0~~ = BSTSearch (20 , 13)

خروجی = founded = 1

وقتی خروجی برابر با 1 باشد یعنی داده پیدا شده و در صورتیکه 0 باشد یعنی داده پیدا نشده است.

الگوریتم اضافه کردن داده به درخت دودویی BST

```
Void  insertBST ( node * T , int x )
{
        node * p , * q , * S ;
        p = new (node) ; p → data = x ;
        p → Right = Null ; p → Left = Null ;
        S = T ;
        While ( S && S → data != x )
        {
                if ( S → data > x ) { q = S ; S = S → Left ; }
                else if ( S → data < x ) { q = S ; S = S → Right ; }
        }
        if ( !S ) if ( q → data > x ) q → Left = p ;
                else q → Right = p ;
}
```

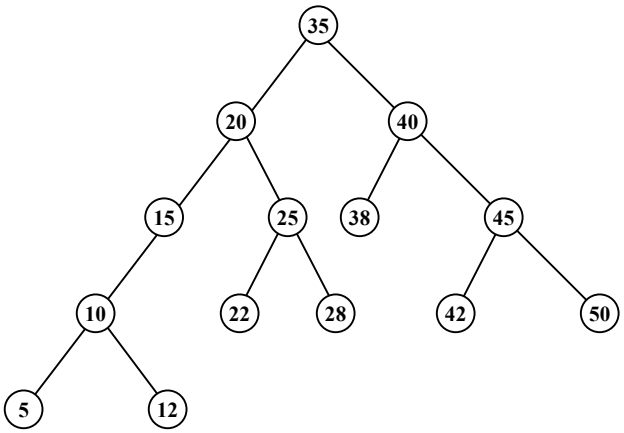
حذف

بـرای حـذف یک گره از درخت جستجو دودویی ابتدا باید آن گره را در درخت BST پیدا کنیم. حال یکی از وضعیتهای زیر رخ می‌دهد :

۱- اگـر گـره مـورد نظـر برگ باشد حذف می‌شود یعنی حافظه گرفته شده برای گره آزاد شده و اشاره‌گر پدرش Null می‌شود.

۲- اگـر گـره حـذف شدنی فقط یک فرزند داشته باشد فرزند آن گره جایگزین گره حذف شدنی می‌گردد و یا می‌توان مورد بعدی را انجام داد.

۳- اگـر گـره دارای دو فـرزند باشد یک قدم به راست و سپس آنقدر به چپ می‌رویم تا به Null برسـیم و یـا بـرعکس یک قدم به چپ و سپس آنقدر به راست می‌رویم تا به Null برسیم. با دنـبال کردن هر یک از حالات فوق گره آخر را جایگزین گره حذف شدنی کرده و حافظه آنرا آزاد می‌کنیم.

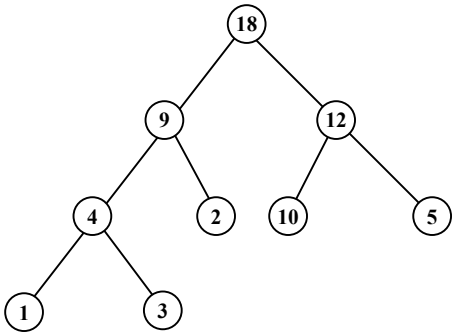


- اگـر بخواهـیم گره شماره ۴۰ را حذف کنیم هم می‌توانیم گره شماره ۳۸ و هم می‌توانیم گره شماره ۴۲ را جایگزین آن کنیم.
- اگـر بخواهـیم گره شماره ۲۰ را حذف کنیم هم می‌توانیم گره شماره ۱۵ و هم می‌توانیم گره شماره ۲۲ را جایگزین آن کنیم.
- اگـر بخواهـیم گره شماره ۱۵ را حذف کنیم هم می‌توانیم گره شماره ۱۰ و هم می‌توانیم گره شماره ۱۲ را جایگزین آن کنیم.

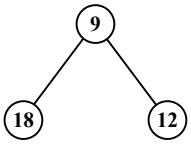
درخت heap (کپه)

درختـی اسـت دودویـی کامل (Complete) که تعداد موجود در هر گره از مقدار موجود در گره‌های فـرزندانش کوچکـتر نباشـد. ایـن کـپه ، کـپه بیشـترین (maxheap) اسـت. در صورتیکه در درخت دودویـی کامل مقدار هر گره از مقدار گره فرزندانش بیشتر نباشد کپه کمترین (minheap) خواهیم داشت.

maxheap



minheap



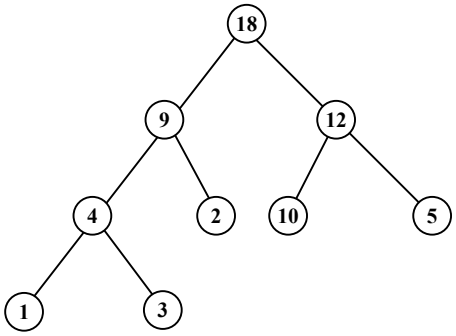
در maxheap بزرگترین عنصر را می‌توان با مرتبه زمانی $O(1)$ بدست آورد (یعنی بدون محاسبه زیرا ریشه درخت بزرگترین عنصر است) و متقابلاً در minheap کمترین عنصر را می‌توان با مرتبه زمانی $O(1)$ بدست آورد (در این حالت نیز کمترین عنصر همان ریشه درخت است).

افزودن داده به درخت heap

برای افزودن داده جدید به درخت heap داده جدید را در انتهای لیست آرایه قرار می‌دهیم (توجه اینکه درخت heap را درون آرایه نگهداری می‌کنیم). روال زیر تا رسیدن به ابتدای آرایه و یا بر قرار بودن شرط درخت heap انجام می‌شود :

- ✓ داده موجود در خانه i (برای اولین بار آخرین عنصر آرایه) با پدر خویش در خانه $\frac{i}{2}$ مقایسه می‌شود. در صورت جابجایی مجدداً این مقایسه برای خانه جدید ($\frac{i}{2}$) انجام می‌گردد. این روش را افزودن به طریقه درج در heap می‌نامند.
- ✓ درج در درخت heap برای هر عنصر با مرتبه زمانی Log_2^n انجام می‌شود.

درخت زیر را در نظر بگیرید :



آرایه این درخت به شکل زیر خواهد شد.

1	2	3	4	5	6	7	8	9
18	9	12	4	2	10	5	1	3

حال می‌خواهیم داده شماره 8 را به این آرایه اضافه کنیم و درخت heap نیز برقرار باشد.

1	2	3	4	5	6	7	8	9	10
18	9	12	4	~~2~~	10	5	1	3	~~8~~
18	9	12	4	8	10	5	1	3	2

حال می‌خواهیم ابتدا داده شماره 7 و سپس داده شماره 25 را به آرایه اضافه کنیم.

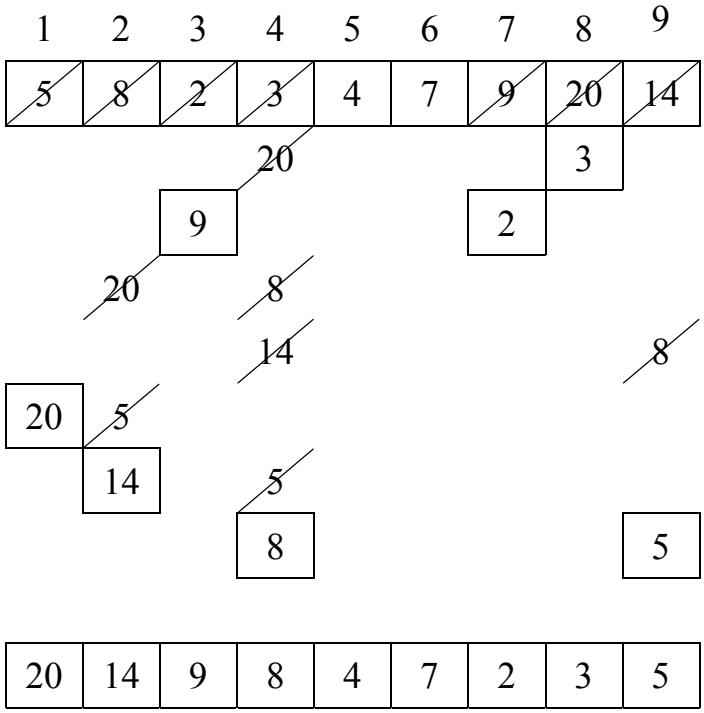
1	2	3	4	5	6	7	8	9	10	11	12
18	9	12	4	8	~~10~~	5	1	3	2	7	~~25~~
18	9	~~12~~	4	8	~~25~~	5	1	3	2	7	10
~~18~~	9	~~25~~	4	8	12	5	1	3	2	7	10
25	9	18	4	8	12	5	1	3	2	7	10

چون داده شماره 7 بر سر جای خود درست قرار گرفته است پس آنرا دست نمی‌زنیم و فقط داده شماره 25 را آنقدر جابجا کرده تا درخت heap برقرار باشد.

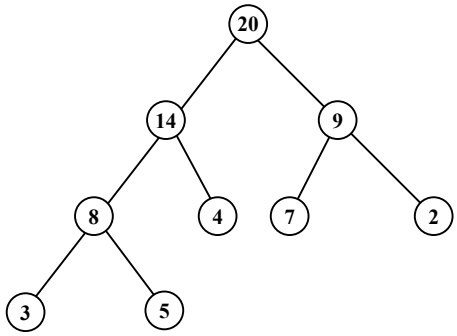
روش ساخت درخت دودویی heap به روش جوان‌ترین پدر

در ایجاد کپه به روش جوان‌ترین پدر ابتدا همه n عنصر ورودی را در یک آرایه قرار دهید. سپس از پائین درخت شروع کرده ، هر پدر و فرزندانش را بصورت کپه تنظیم می‌کنیم و به سمت بالا (ریشه) حرکت می‌کنیم. همینطور که به سمت ریشه می‌رویم زیردرخت‌ها بصورت کپه درآمده‌اند. در این روش چون برگها خودبخود به تنهایی یک heap هستند باید از جوان‌ترین پدر شروع کنیم که اگر عناصر آرایه i تا باشند از عنصر $\frac{i}{2}$ شروع می‌کنیم.

مثال : درخت زیر را به روش جوان‌ترین درخت به صورت درخت دودویی heap درآورید.



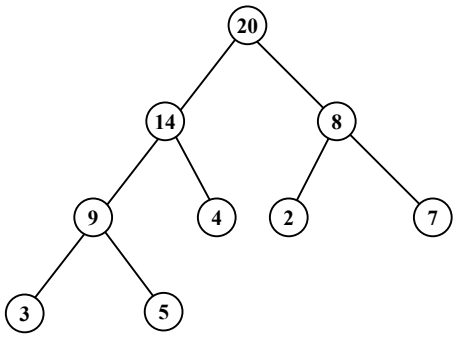
1	2	3	4	5	6	7	8	9
20	14	9	8	4	7	2	3	5



مثال : آرایه زیر را به روش درج بصورت درخت heap بنویسید.

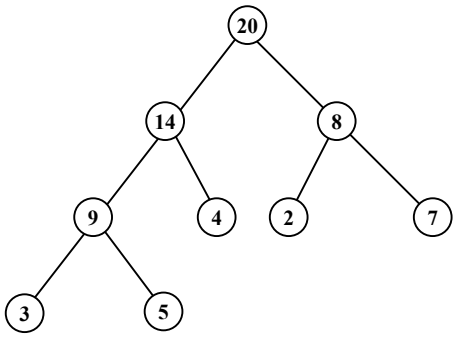
1	2	3	4	5	6	7	8	9
5	8	2	3	4	7	9	20	14

8	5	2	3	4	7			
8	5	7	3	4	2	9		
8	5	9	3	4	2	7	20	
8	5	9	20	4	2	7	3	14
9	5	8	20	4	2	7	3	14
9	20	8	5	4	2	7	3	14
9	20	8	14	4	2	7	3	5
20	9	8	14	4	2	7	3	5
20	14	8	9	4	2	7	3	5



حذف یک عنصر

برای حذف یک عنصر از درخت heap ، ریشه درخت خارج شده و داده آخر به جای آن قرار می‌گیرد. سپس از ابتدای آرایه (i =) شروع می‌کنیم و بین 2i و 2i + 1 عنصر ماکزیمم را در صورت لزوم با عنصر i عوض می‌کنیم. به همین ترتیب جابجایی انجام گرفته تا زمانیکه به انتهای آرایه برسیم (یک گره فرزندی نداشته باشد).



1	2	3	4	5	6	7	8	9
~~20~~	14	9	8	4	7	2	3	~~5~~
~~5~~	~~14~~	9	8	4	7	2	3	
14	~~5~~	9	~~8~~	4	7	2	3	
14	8	9	5	4	7	2	3	
14	8	9	5	4	7	2	3	

در این مثال عنصر 20 را حذف کرده و عنصر 5 را جایگزین کرده و بقیه مراحل ساخت درخت heap را انجام داده‌ایم.

سؤال : تابع f چه کاری انجام می‌دهد؟

```
int f (node * T )
{
        int L , r ;
        if ( T )
        {
                L = f ( T → Left ) ;
                R = f ( T → Right ) ;
                if  L > r  return  L + 1 ;
                else return  r + 1 ;
        }
        return  0 ;
}
```

جواب : ارتفاع درخت را نشان می‌دهد.

سؤال : تابع f چه کاری انجام می‌دهد؟

```
node  * f ( node * T )
{
        node  * r , * s , * q ;
        q = Null ;
        if ( T )
        {
                r = f ( T → Left ) ;
                s = f ( T → Right ) ;
                q = New ( node ) ;
                q → Left = r ;
                q → Right = s ;
                q → data = T → data ;
        }
        return  q ;
}
```

جواب : از یک کپی تهیه می‌کند.

سؤال : تابع g چه کاری انجام می‌دهد؟

```
int  g ( node * T )
{
        if ( T )
        {
                if ( ! T → Left ) && ( ! T → Right )
                        return  1 ;
                else
                        return ( g ( T → Left ) + g ( T → Righe ) + 1 ) ;
        }
        return  0 ;
}
```

جواب : تعداد گره‌ها را چاپ می‌کند.

سؤال : تابع h چه کاری انجام می‌دهد؟

```
int  h ( node * T )
{
        if ( T )
        {
                if ( T → Left = = Null ) && ( T → Right = = Null )
                        return  1 ;
                else
                        return ( h ( T → Left ) + h ( T → Righe ) ) ;
        }
        return  0 ;
}
```

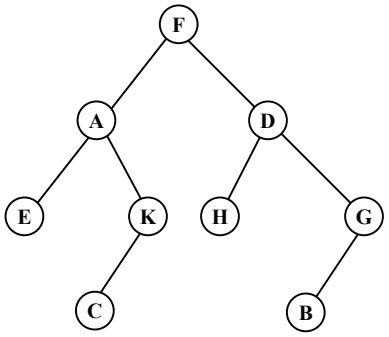
جواب : تعداد برگها را چاپ می‌کند.

ســؤال : پـیمایش Inorder و Preorder زیـر را داریـم. درخت دودویی آنرا تشکیل داده و پیمایش Postorder آنرا بنویسید.

Inorder : E A C K F H D B G

Preorder : F A E K C D H G B

Postorder : E C K A H B G D F

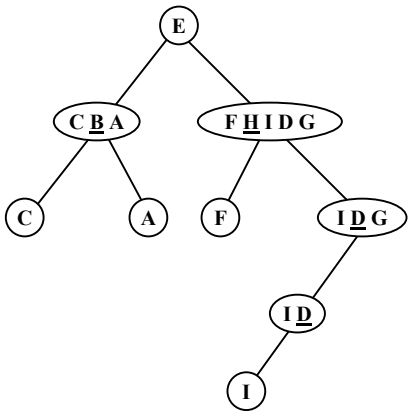


ســؤال : پـیمایش Inorder و Postorder زیر را داریم. درخت دودویی آنرا تشکیل داده و پیمایش Preorder آنرا بنویسید.

Inorder : C B A E F H I D G

Postorder : C A B F I D G H E

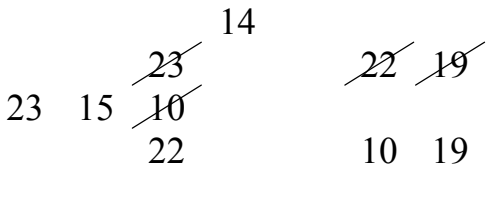
Preorder : E B C A H F G D I



سؤال : درخت maxheap آرایه زیر را از دو روش درج و جوان‌ترین پدر بدست آورید.

1	2	3	4	5	6	7	8
~~10~~	15	~~22~~	~~4~~	11	~~23~~	~~19~~	~~14~~

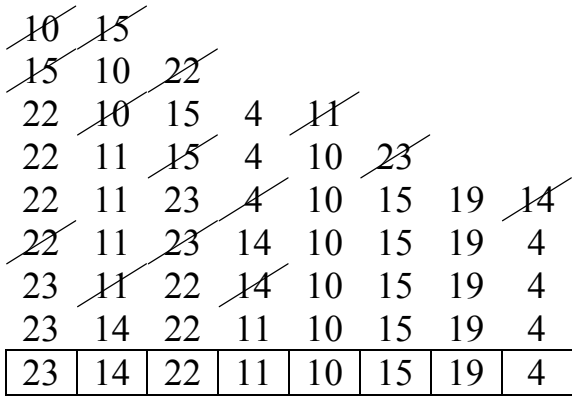
به روش جوان‌ترین پدر



23	15	22	14	11	10	19	4

1	2	3	4	5	6	7	8
10	15	22	4	11	23	19	14

به روش درج



23	14	22	11	10	15	19	4