فصل پنجم : درخت

اهداف

- آشنایی با درخت
- درخت های دودویی
- پیمایش درختان
- هرم
- جنگل

فصل پنجم : درختان

درخت

ساختار درختی یعنی مجموعه داده های سازماندهی شده ای که عناصر اطلاعاتی شان به وسیله انشعاباتی با هم رابطه داشته باشند.

مفهوم درخت

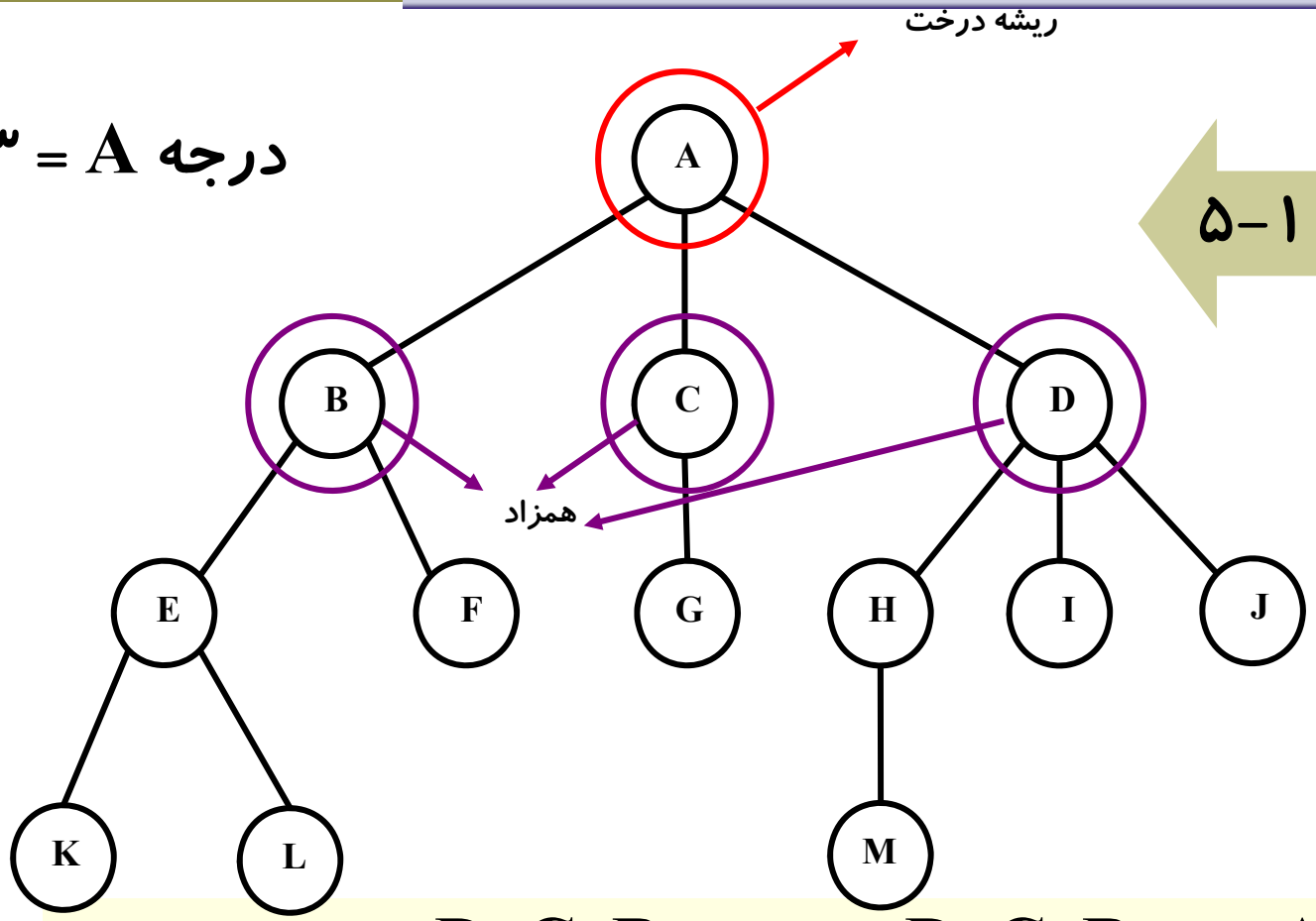
درخت مجموعه محدودی از یک یا چند گره به صورت زیر می باشد :

- دارای گره خاصی به نام ریشه باشد.
- بقیه گره ها به $n \geq 0$ مجموعه مجزا $T_1,....T_n$ تقسیم شده که هر یک از این مجموعه ها خود یک درخت هستند. $T_1,....T_n$ زیر درختان ریشه نامیده می شوند.

تعریف

- به عنصر حاوی اطلاعات و انشعابات به دیگر عناصر، گره اطلاق می شود.
- تعداد زیر درختهای یک گره، درجه آن نامیده میشود
- گره ای که درجه آن صفر باشد، برگ یا گره پایانی نامیده میشود و در مقابل بقیه گره‌ها گره های غیرپایانی نامیده میشوند.
- گره ریشه را در هر زیر درخت گره پدر(والد) و به گره‌های متصل شده به آن گره فرزند میگویند
- فرزندانی که پدر یکسان دارند برادر (یا همزاد) نامیده میشوند

مثالی از یک درخت



درجه A = ۳

شکل ۱-۵

A والد B، C، D است . B، C، D همزادند.

اصطلاحات درخت ها

اجداد گره : اجداد یک گره ، گره هایی هستند که در مسیر طی شده از ریشه تا آن گره وجود دارند.

سطح گره : سطح یک گره بدین صورت تعریف می شود که ریشه در سطح یک قرار می گیرد. برای تمامی گره های بعدی ، سطح گره برابر است با سطح والد به اضافه یک .

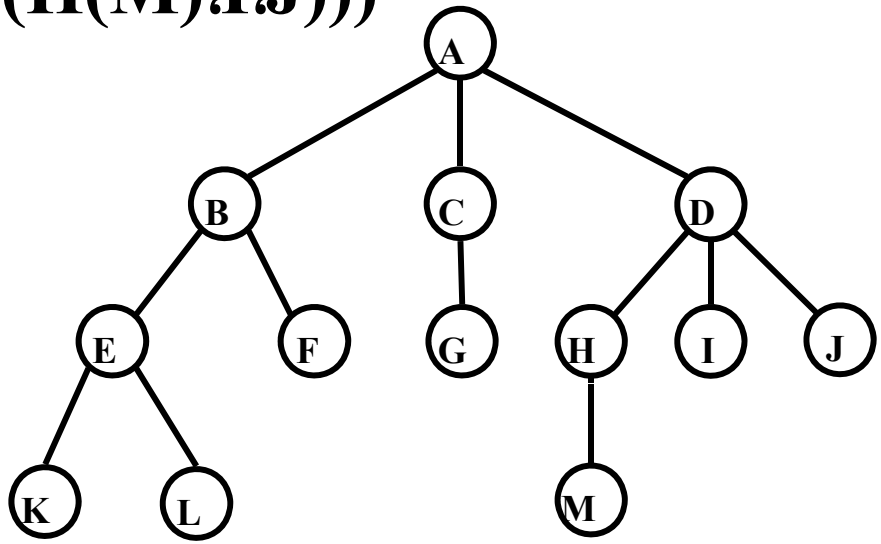
ارتفاع درخت : ارتفاع یا عمق یک درخت به بیشترین سطح گره های آن درخت گفته می شود.

نمایش لیست

یک راه نمایش درخت ، استفاده از لیست است .

شکل ۱-۵ را می توان به صورت زیر نشان داد :

(A(B(E(K ،L)،F)،C(G)،D(H(M)،I،J)))



تمرین

- درخت اسلاید قبل را با استفاده از لیستهای عمومی نمایش دهید.

- اگر بخواهیم برای نگهداری یک درخت ساختمان داده ای مانند نمایش معمول (هر گره آدرس گره های فرزند خود را داشته باشد) تعریف کنیم مشکل زیر بوجود می‌آید:
  - درجه تمام درختها یکی نمیباشد لذا مجبور خواهیم شد برای هر گره تعداد متغیری از اشاره گرها داشته باشیم. با این وجود، نوشتن الگوریتم برای گره‌هایی که طول متغیر دارند دشوار خواهد شد
- برای حل مشکل باید از گره هایی با طول ثابت استفاده کنیم.

استفاده از گره با طول ثابت

- اگر بیشترین درجه درخت k باشد بنابراین هر گره باید توانایی نگهداری k اشاره گر را داشته باشد و ساختمان داده ای به شکل زیر باید تعریف کنیم.

data	link 1	link 2	…	Link k

قضیه

- اگر T درختی از درجه k با n گره باشد و هر گره آن مانند شکل قبل طول ثابتی داشته باشد آنگاه n(k-1)+1 از nk فیلد بچه آن (اشاره گرهای به فرزند) برابر 0 (NULL) است.

- اثبات:

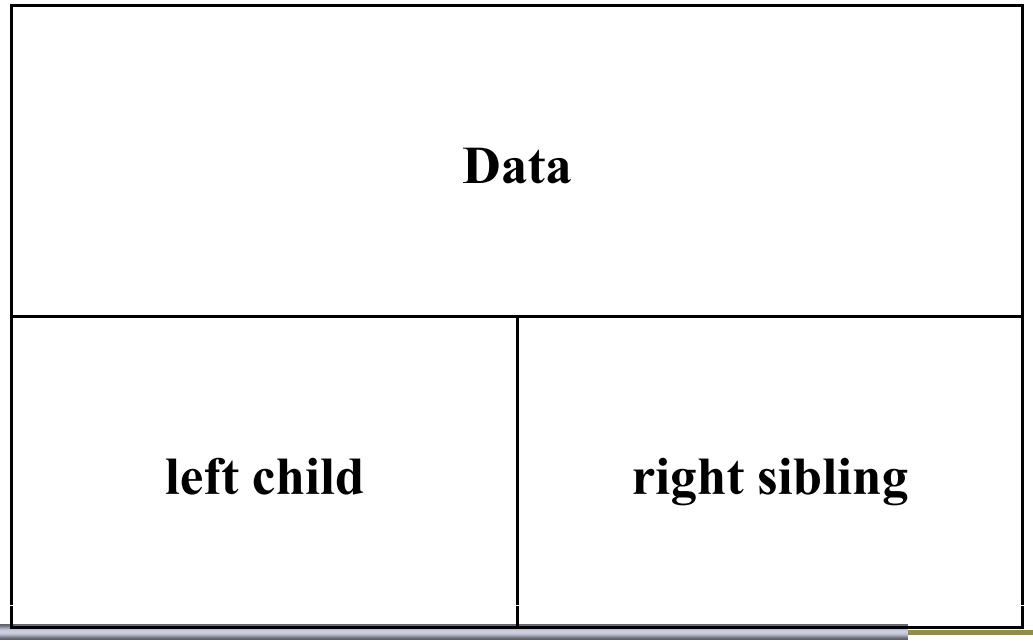
- از آنجا که هر فیلد بچه غیر صفر به یک گره اشاره میکند و برای هر گره غیر از ریشه یک اشاره گر وجود دارد از اینرو تعداد فیلدهای بچه غیرصفر برای درختی با n گره دقیقاً برابر n-1 است. تعداد کل فیلدهای بچه برای درختی با n گره از درجه k برابر nk میباشد. لذا:

- تعداد اشاره گرهای بچه NULL : $nk-(n-1)=n(k-1)+1$

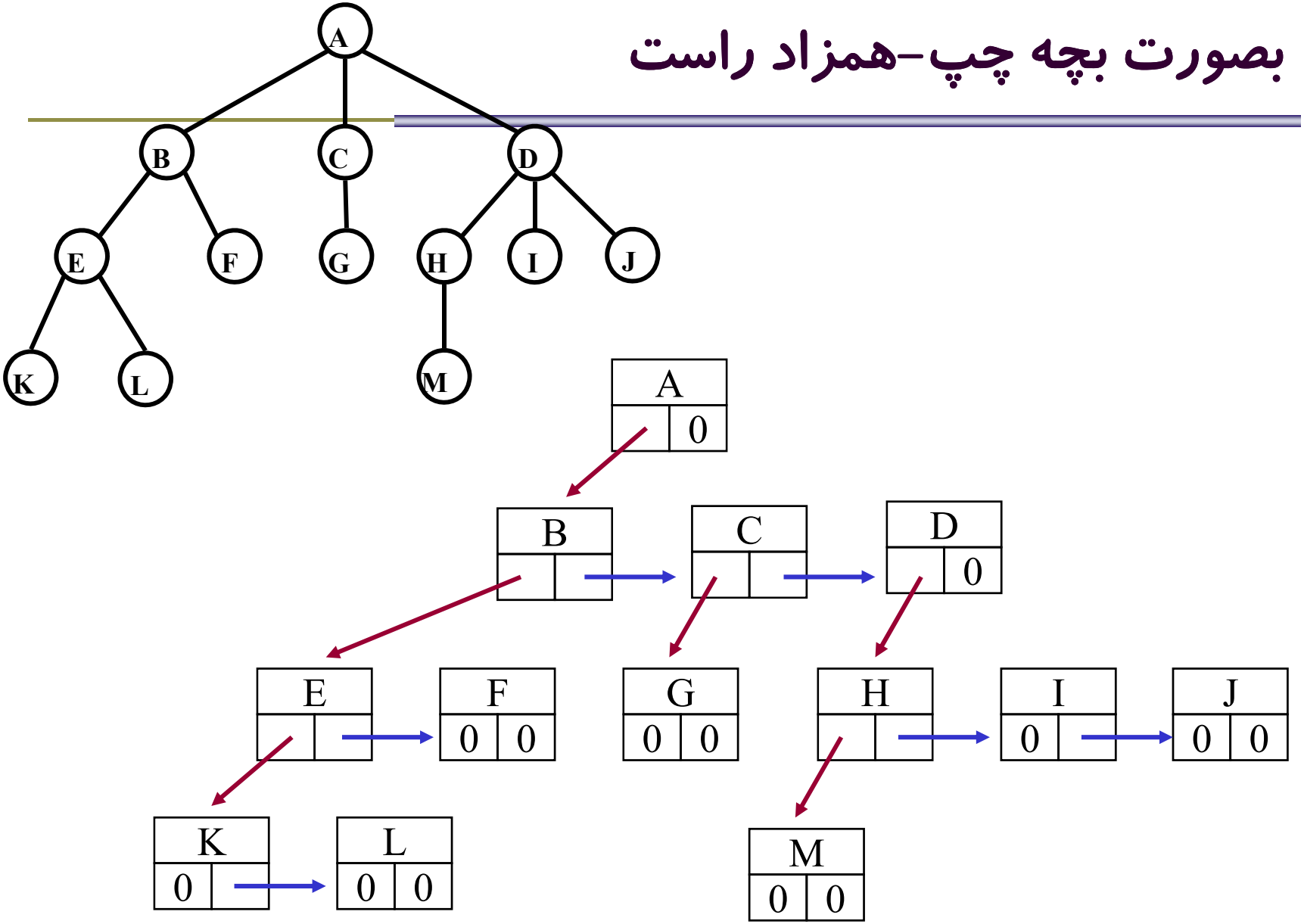
- برای درختان دو نمایش خاص که از گره‌هایی با طول ثابت استفاده میکند ارائه میکنیم.
- الف) نمایش درخت بصورت بچه چپ-همزاد راست
- ب) نمایش درخت بصورت یک درخت درجه ۲.

نمایش دودویی یک درخت

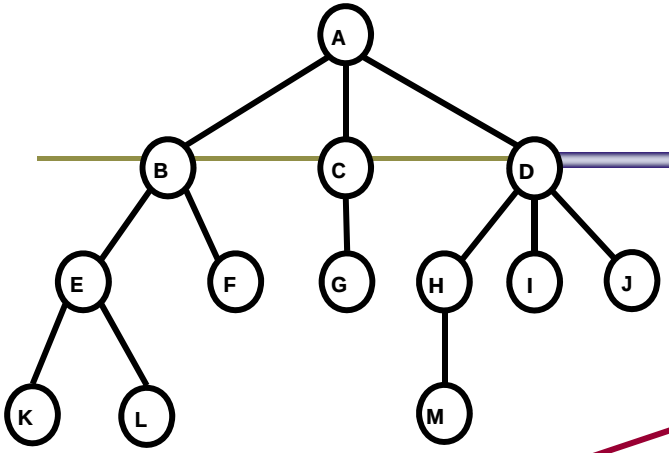
برای نمایش درختان دودویی ، دقیقا نیاز به دو اتصال یا اشاره گر به ازای هر گره است.

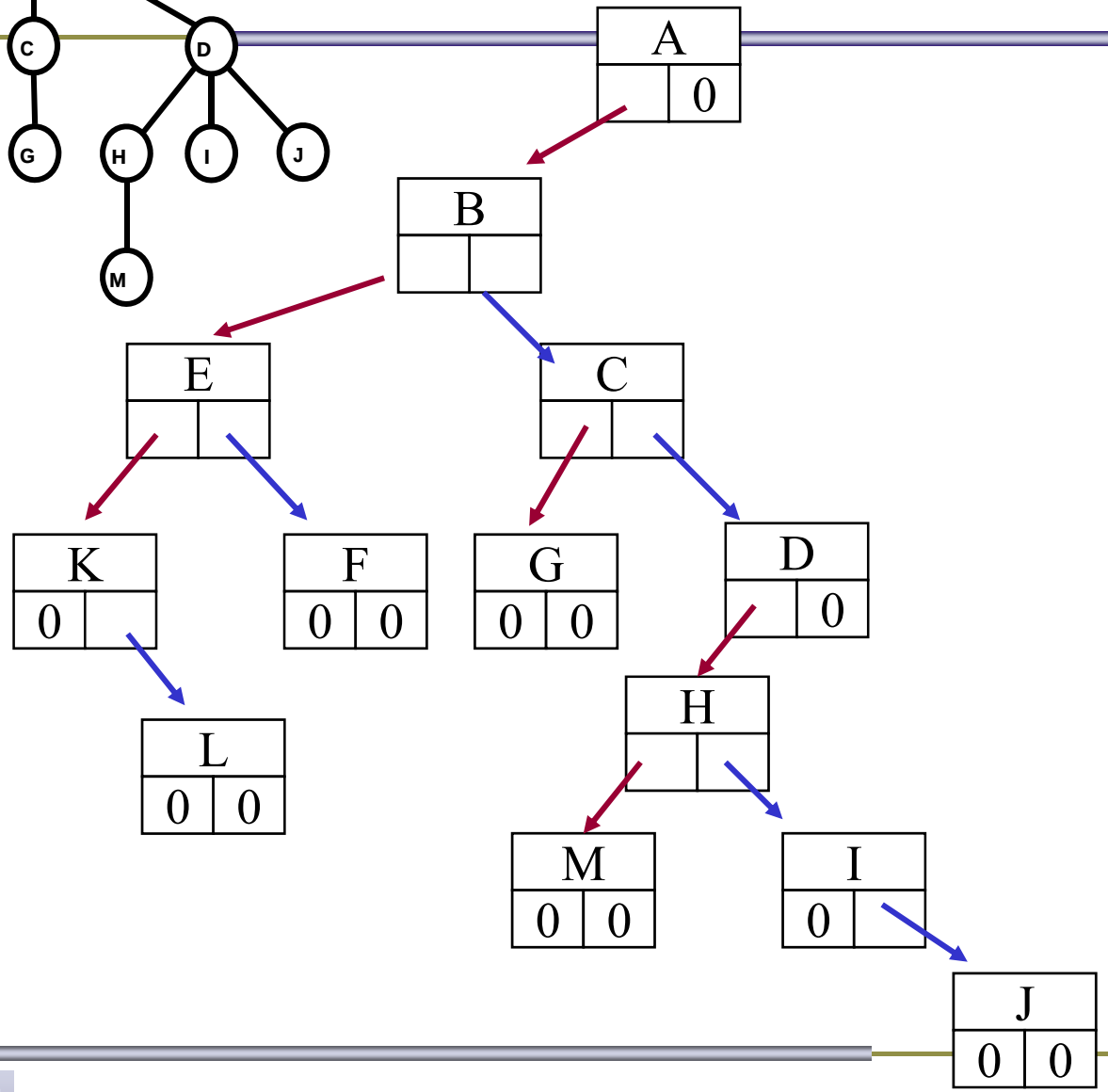


بصورت بچه چپ-همزاد راست



بصورت درخت درجه ۲





۲-۵ درخت های دودویی

تعریف :

یک درخت دودویی یا تهی است یا حاوی مجموعه ای محدود از گره ها شامل یک ریشه و دو زیر درخت دودویی است. این درخت ها زیر درخت های چپ و راست نامیده می شوند.

- **مشخصه اصلی یک درخت دودویی بدین شکل است که هر گره آن حداکثر دو انشعاب دارد یعنی گره هایی که درجه ای بیشتر از دو نداشته باشند.**
- **برای درخت های دودویی زیر درخت سمت چپ و راست با یکدیگر متمایز است.**

۲-۵ ساختار درخت دودویی

structure *Binary_tree* (abbreviated *BinTree*) is

objects : a finite set of nodes either empty or consisting of

a root node ،left *Binary_tree* ،and right *Binary_tree.*

functions :

for all bt ،bt1 ،bt2 *BinTree* ،item element

BinTree Create()

Boolean IsEmpty (bt)

BinTree MakeBT(bt1 ،item ،bt 2)

BinTree Lchild(bt)

element Data(bt)

BinTree Rchild(bt)

۲-۵ تفاوت درخت عادی با درخت دودویی

- در هیچ درخت عادی صفر گره وجود ندارد ، اما درخت دودویی تهی وجود دارد.
- در یک درخت دودویی ترتیب فرزندان دارای اهمیت بوده در حالی که در درخت عادی به این صورت نیست.

۲-۵ خواص درختان دودویی

حداکثر تعداد گره ها

- حداکثر تعداد گره ها در سطح iام یک درخت دودویی 2^{i-1} است.
- حداکثر تعداد گره ها در یک درخت دودویی به عمق k ، $2^{k}-1$ است.

۲-۵ خواص درختان دودویی

رابطه بین تعداد گره های برگ و گره های درجه ۲

برای هر درخت دودویی غیر تهی مانند T ، اگر n_0 تعداد گره های پایانی و n_2 تعداد گره های درجه ۲ باشد ، آنگاه خواهیم داشت :

$$n_0 = n_2 + 1$$

۲-۵ خواص درختان دودویی

- یک درخت دودویی پر به عمق k، یک درخت دودویی پر است مشروط به اینکه 2^k-1 گره داشته باشد.
- یک درخت دودویی با n گره و عمق k کامل است ، اگر و تنها اگر گره هایش مطابق با گره های شماره گذاری شده در یک درخت دودویی پر به عمق k باشد.
- ارتفاع یک درخت دودویی کامل با n گره برابر $\lceil \log_2(n+1) \rceil$ است

۲-۵ نمایش درخت دودویی

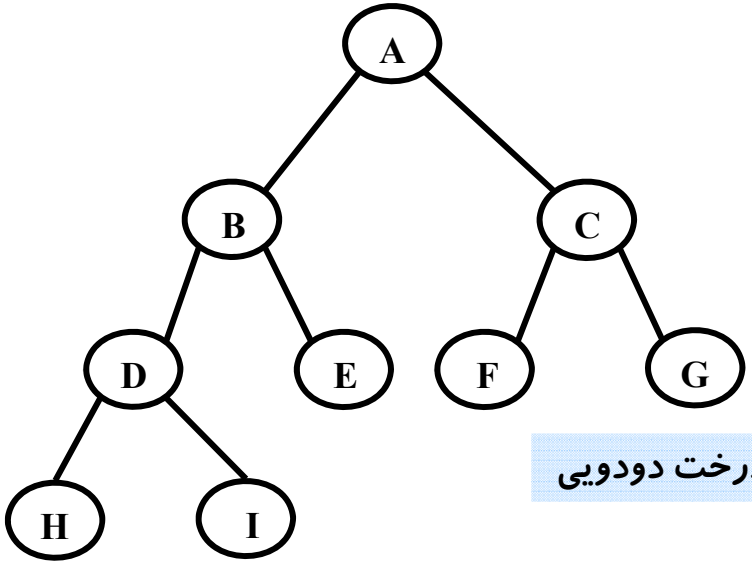
نمایش درخت دودویی به دو صورت است :

- نمایش آرایه
- نمایش لیست پیوندی

۲-۵ نمایش آرایه

شیوه شماره گذاری ارایه شده در شکل زیر ، اولین نمایش یک درخت دودویی در حافظه را مطرح و پیشنهاد می کند . از آنجایی که گره ها از 1 تا n شماره گذاری شده اند ، یک آرایه یک بعدی می تواند برای ذخیره سازی گره ها استفاده شود .

۲-۵ نمایش آرایه



نمایش آرایه ای درخت دودویی

0	-
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H

۲-۵ نمایش آرایه

اگر یک درخت دودویی کامل با n گره (یعنی $\text{عمق} = [\log_2 n] + 1$) به ترتیب بالا تعریف شده باشد ، آنگاه برای هر گره با اندیس i و $1 \le i \le n$، داریم:

- (۱) اگر i≠1 ، آنگاه پدر i در [i/2] است . اگر i=1 ، i ریشه است و پدری نخواهد داشت.
- (۲) اگر 2i≤n ، آنگاه فرزند چپ i در 2i است. اگر 2i>n ، آنگاه i فرزند چپ ندارد.
- (۳) اگر 2i+1≤n ، آنگاه فرزند راست i در 2i+1 است. اگر 2i+1>n ، آنگاه i فرزند راست ندارد

۲-۵ نمایش آرایه

در بدترین حالت ، یک درخت مورب به عمق **k** ، به $2^k - 1$ محل و موقعیت نیاز دارد که از این مقدار، فقط **k** محل اشغال می شود.

۲- ۵ نمایش لیست پیوندی

اگرچه نمایش ترتیبی (آرایه ای) برای درختان دودویی کامل مناسب به نظر می رسد ، ما برای بسیاری از درختان دیگر باعث اتلاف حافظه میشود به علاوه ، این روش از نارسایی های موجود در نمایش ترتیبی نیز برخوردار است. درج یا حذف گره های یک درخت ، مستلزم جابه جایی گره هاست که خود باعث تغییر شماره سطح گره ها می شود. این مسایل می تواند با به کارگیری نمایش پیوندی به آسانی حل شود.

۲-۵ نمایش لیست پیوندی

با این روش هر گره سه فیلد خواهد داشت :

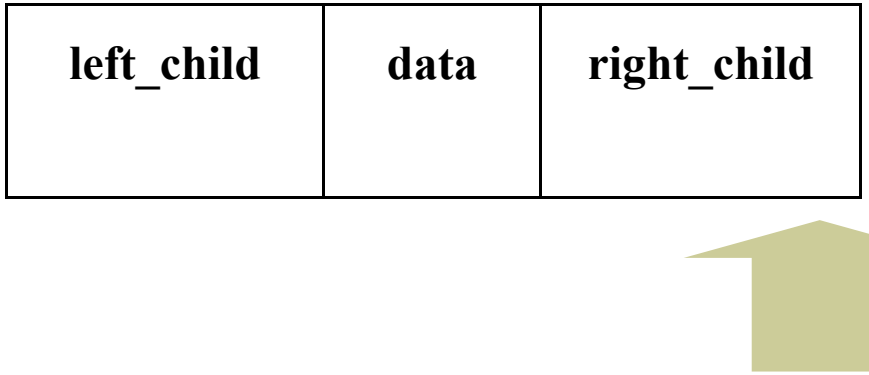
left_child، data، right_child

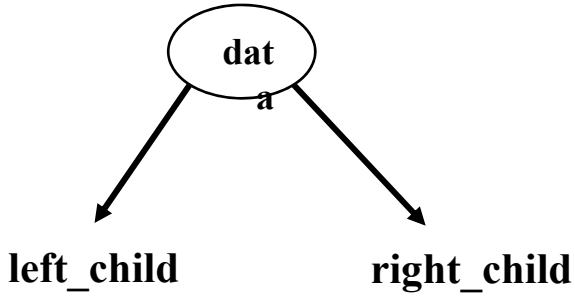
که در زبان C به شرح زیر تعریف می شوند :

```
typedef  struct  node  *tree_pointer ;
typedef  struct  node  {
int  data ;
tree_pointer  left_child  ،right_child ;
};
```

```
class Tree;
class TreeNode
{
friend class Tree;
private:
    char data;
     TreeNode *LeftChild;
     TreeNode *RightChild;
};
class Tree
{
public:
     Tree();
     Tree(const Tree& t);
private:
     TreeNode *root;
};
```

۵-۲ نمایش لیست پیوندی





نمایش یک گره درخت دودویی

۳-۵ پیمایش درخت دودویی

به هنگام پیمایش یک درخت دودویی ، با هر گره و زیردرختانش به طرز مشابهی رفتار کنیم. اگر R، V، L به ترتیب حرکت به چپ ، ملاقات کردن یک گره (برای مثال ، چاپ فیلد داده آن گره) و حرکت به راست باشد، آنگاه شش ترکیب ممکن برای پیمایش یک درخت خواهیم داشت :

RLV، RVL، VRL، VLR، LRV، LVR

۳-۵ پیمایش درخت دودویی

مثال

اگر تنها حالتی را انتخاب کنیم که ابتدا به سمت چپ و بعد به سمت راست برود ، تنها سه ترکیب VLR، LRV، LVR خواهیم داشت. این سه حالت را با توجه به موقعیت V نسبت به L و R به ترتیب preordcr، postorder، inorder می نامیم.

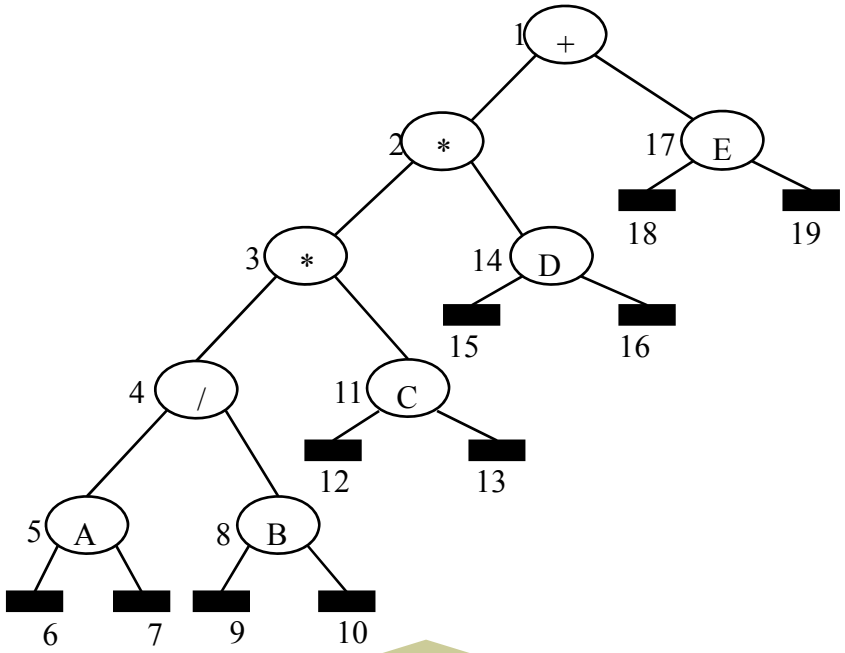
۳-۵ پیمایش درخت دودویی

- در پیمایش postorder ، یک گره موقعی ملاقات و چاپ می شود که زیردرختان چپ و راست آن قبلا ملاقات شده باشند.

- در پیمایش preorder ، یک گره قبل از پیمایش زیردرختان چپ و راست ، ملاقات می گردد.

۳-۵ پیمایش درخت دودویی

درخت زیر حاوی یک عبارت ریاضی است : A/B*C*D*+E



درخت دودویی برای یک عبارت محاسباتی

۳-۵ پیمایش Inorder

هنگامی که این پیمایش انتخاب می شود ، حرکت به سمت پایین به طرف چپ انجام می شود و این عمل تا آخرین گره صورت می گیرد سپس می توان گره را بازیابی کرد و بعد به سمت راست رفته و به همین ترتیب کار را ادامه پیدا می کند.

این متناظر با شکل infix یک عبارت است.

۵-۳ پیمایش Inorderیک درخت دودویی

```
Vide  inorder (tree_pointer   ptr )
/*  inorder  tree   traversal */
{
    if  (ptr)  {
            inorder  ( ptr -> left_child );
            printf  (“ % d”  ،ptr -> data );
            inorder  (ptr -> right_child);
    }
}
```

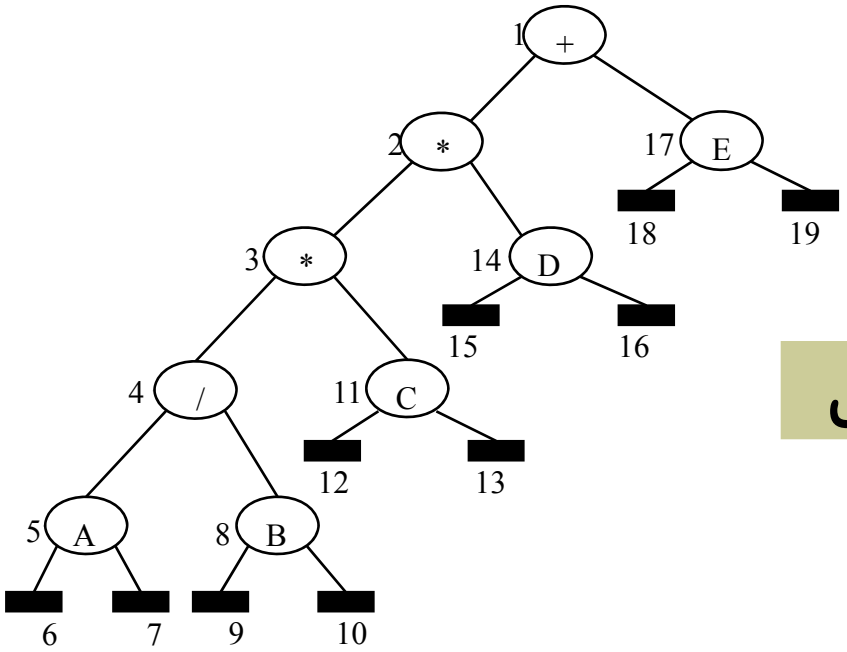
۵-۳ پیمایش Preorder

تابع preorder حاوی دستورات لازم برای شکل دوم پیمایش است.

بر اساس این پیمایش ، گره را ابتدا بازیابی و ملاقات نموده و سپس انشعابات چپ را دنبال و تمام گره ها را بازیابی می کنیم. این فرآیند ادامه پیدا می کند تا به یک گره تهی برسیم. در این نقطه ، به نزدیکترین جدی که دارای یک فرزند راست باشد مراجعه و با این گره شروع خواهیم نمود.

۳-۵ پیمایش Preorder

با پیمایش preorder گره های درخت زیر خروجی به شکل زیر خواهند داشت :



خروجی

+ * * / A B C D E

این به شکل یک عبارت prefix است.

۵-۳ پیمایش Preorder یک درخت دودویی

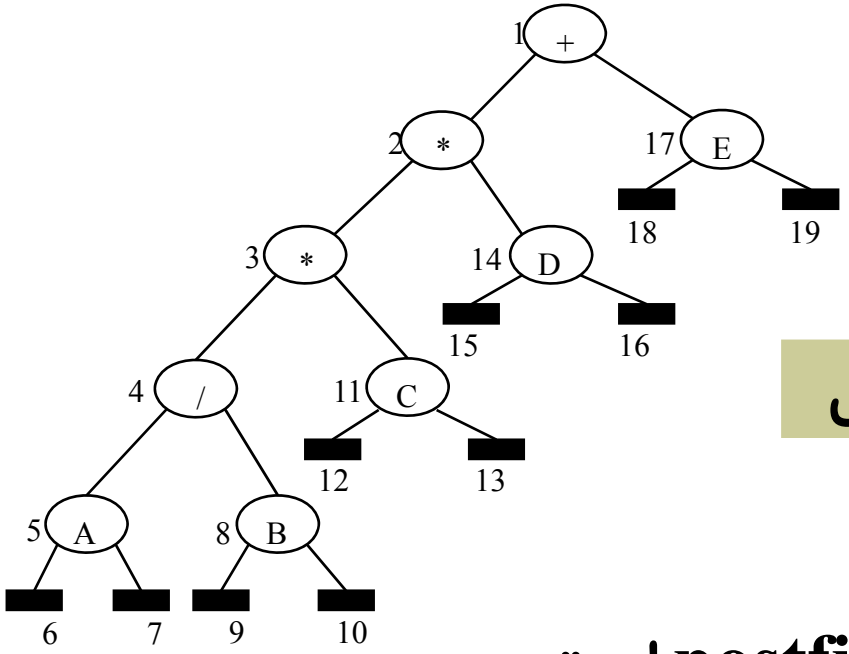
```
Vide  preorder (tree_pointer   ptr )
/*  preorder  tree   traversal */
{
   if  (ptr)  {
         printf  (“ % d”  ،ptr -> data );
         preorder  ( ptr -> left_child );
         preorder  (ptr -> right_child);
   }
}
```

۳-۵ پیمایش postorder

این پیمایش دو فرزند یک گره را قبل از بازیابی آن گره ملاقات و چاپ می کند. این مساله بدین مفهوم است که فرزندان یک گره قبل از خود آن گره بازیابی می گردد.

۳-۵ پیمایش postorder

خروجی حاصل از پیمایش postorder شکل زیر به صورت زیر است :



خروجی

A B/ C* D* E +

این خروجی مانند یک عبارت postfix است.

۵-۳ پیمایش inorder غیربازگشتی

```
Void  iter_pointer (tree_pointer  node )
{
        int  top = -1 ;   /* initialize  stack */
        tree_pointer   stack [MAX_STACK_SIZE] ;
        for ( ; ; ) {
                for (; node ; node = ->left_child)
                add ( &top  ،node ); /* add  to  stack  */
        node = delete (&top); /*delete  from stack */
                if (! Node)  break ; /* empty  stack*/
                printf  (“ % d”  ،node-> data ) ;
                node = node -> right_child;
        }
}
```

۳-۵ پیمایش inorder غیربازگشتی

تحلیل inorder2 : فرض کنید تعداد گره های درخت n باشد ، اگر عمل iter_inorder را در نظر بگیریم ، مشاهده می شود که هر گره درخت فقط یک بار در پشته قرار گرفته و یا از آن خارج می شود. بنابراین اگر تعداد گره های درخت n باشد ، پیچیدگی زمان تابع برابر با O(n) می باشد. حافظه مورد نیاز برابر با عمق درخت است که مساوی با O(n) می باشد.

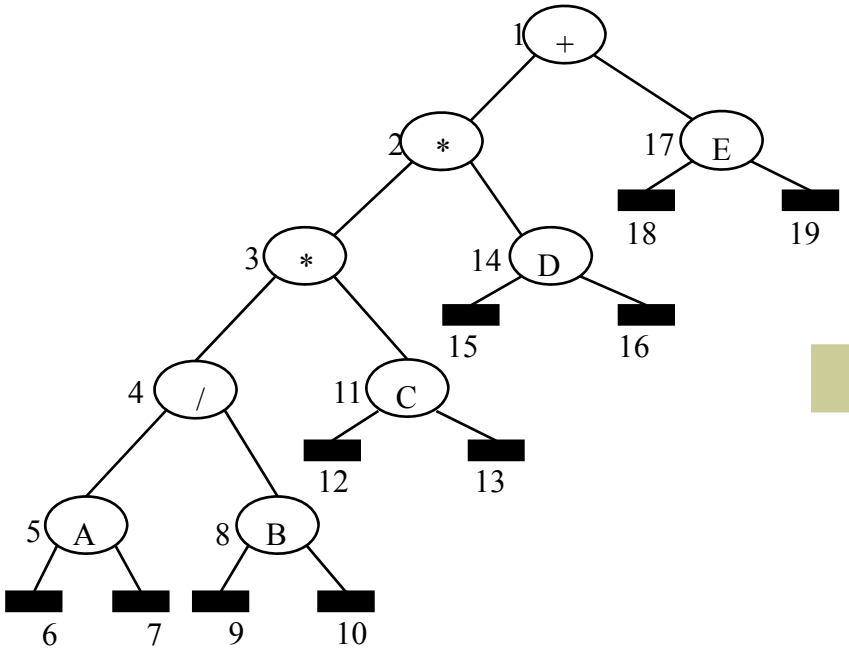
۳-۵ پیمایش ترتیب سطحی

پیمایش های inorder، preorder، postorder چه به صورت بازگشت پذیری نوشته یا به صورت غیربازگشتی ، همگی نیازمند پشته می باشند.

این پیمایش ، ترتیب سطحی ، ابتدا ریشه را بازیابی ، سپس فرزند چپ ریشه و به دنبال آن فرزند راست ریشه بازیابی می گردد. این شیوه با بازیابی از گره منتهی الیه سمت چپ به سمت راست هر سطح جدید تکرار می گردد. این پیمایش از صف استفاده می کند.

۳-۵ پیمایش ترتیب سطحی

پیمایش ترتیب سطحی درخت زیر به صورت زیر است :



+ *E *D / C A B

۴-۵ اعمال مفید بر روی درختان دودویی

۱- کپی کردن درختان دودویی

۲- تعیین برابری و تساوی دو درخت

۳- مساله Satisfiability

۵-۵ درختان نخی دودویی

تعداد اتصالات تهی در یک درخت دودویی بیشتر از تعداد اشاره گرهای غیرتهی است.

در یک درخت دودویی تعداد $n + 1$ اتصال از کل اتصالات آن یعنی ، $2n$ تهی است. یک راه برای به کارگیری این اتصالات توسط پرلین و تورنتن پیشنهاد شد. راه حل این بود که از اتصالات تهی برای ارتباط با دیگر گره های یک درخت استفاده شود که در این صورت درخت را درخت نخی می نامند.

۵-۵ درختان نخی دودویی

برای ایجاد اتصالات نخی از قوانین زیر استفاده می شود :

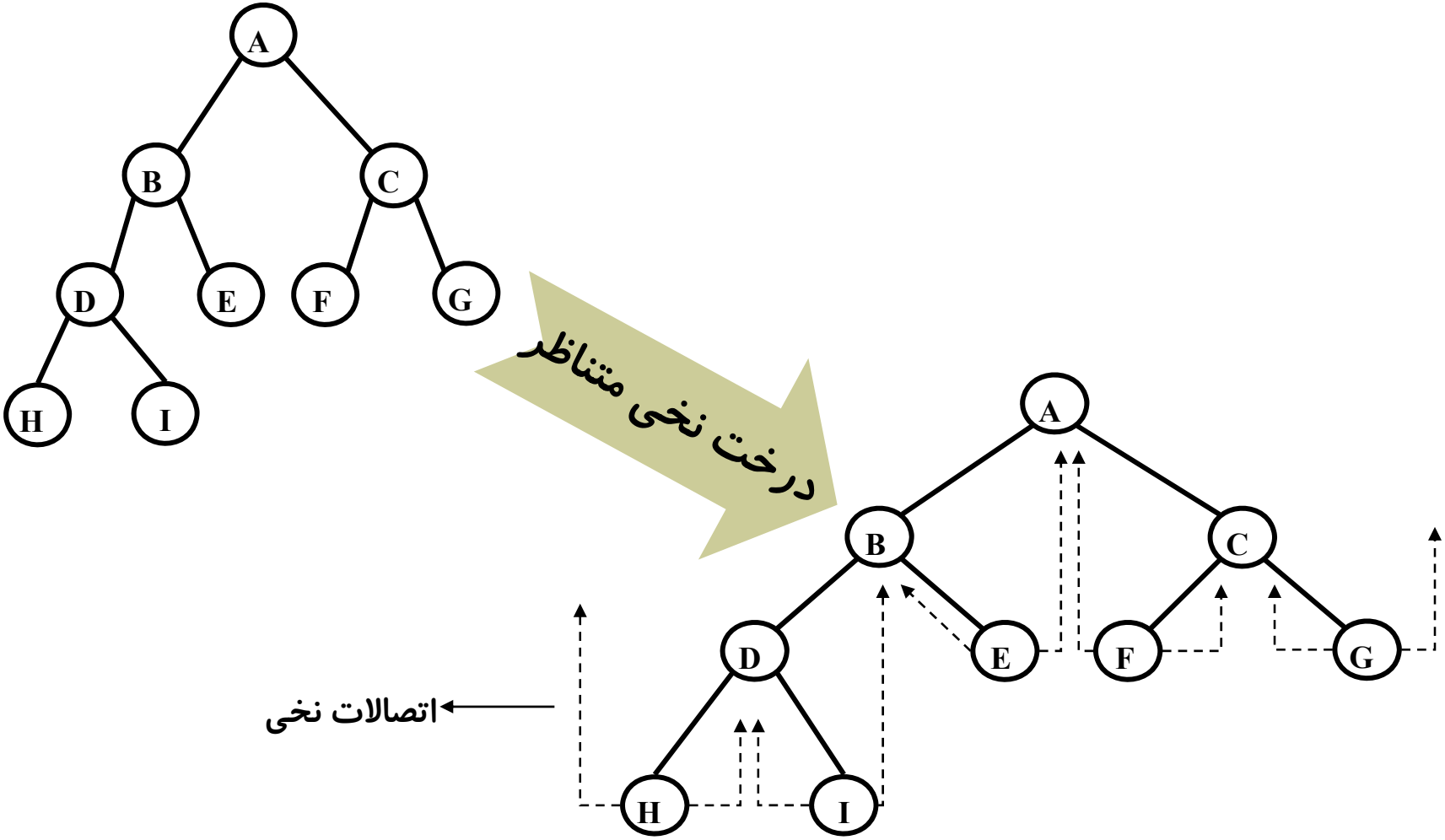
1) اگر ptr-> left_child تهی باشد ، آن را طوری تغییر می دهیم که به گره ای که در پیمایش inorder قبل از ptr قرار دارد ، اشاره کند.

2) اگر ptr-> right_child تهی باشد ، آن را طوری تغییر می دهیم که به گره ای که در پیمایش inorder بعد از ptr قرار دارد ، اشاره کند.

۵-۵ درختان نخی دودویی

هنگامی که درختی را در حافظه نمایش می دهیم ، بایستی بتوانیم بین اتصالات نخی و واقعی تفاوتی قایل شویم. این کار را با افزودن دو فیلد اضافی به هر گره انجام می دهیم که آنها را left_thread ،right_thread می نامیم.

۵-۵ درختان نخی دودویی



۵-۵ پیمایش inorder یک درخت نخی دودویی

برای هر گره مانند ptr ، در یک درخت دودویی ، چنانچه ptr->right_thread = TRUE باشد ، طبق تعریف گره بعدی ptr در پیمایش inorder ، ptr->right_child می باشد . در غیر این صورت گره بعدی ptr، با پایین رفتن روی مسیر فرزندان چپ ptr از طرف فرزند سمت راست ptr تا وقتی که به گره ای با وضعیت left_thread = TRUE برسیم ، تعیین می شود .

۵-۵ پیمایش inorder یک درخت نخی دودویی

تابع insucc بدون استفاده از پشته ، گره بعدی در پیمایش inorder را در یک درخت نخی دودویی پیدا می کند.

برای پیمایش inorder می توانیم با فراخوانی مکرر insucc تمام گره ها را بازیابی کنیم .

۵-۵ پیمایش inorder یک درخت نخی دودویی

تابع insucc :

```
threaded_pointer  insucc (threaded_pointer tree)
{
/* find  the  inorder  sucassor  of  tree  in  a threaded
binary  tree */
        threaded_pointer  temp ;
        temp = tree -> right_child ;
        if  (! Tree -> right_thread)
                while  (! temp -> left_thread)
                        temp = temp -> left_child ;
        return  temp ;
}
```

پیدا نمودن گره بعد ، یک گره خاص در پیمایش inorder

۵-۵ پیمایش inorder یک درخت نخی دودویی

```
Void  tinorder (threaded_pointer  tree)
{
/* traverse  the  threaded  binary  tree  inorder */
        threaded_pointer  temp = tree ;
        for ( ; ; )   {
                temp = insucc (temp) ;
                if  (temp = tree )  break ;
                printf (“ % 3c”  ،temp -> data ) ;
        }
}
```

۵-۵ درج یک گره به داخل درخت نخی دودویی

فرض کنید دارای گرهی به نام parent هستیم که دارای زیردرخت راست تهی می باشد ، آنگاه مایل هستیم child را به عنوان فرزند راست parent درج کنیم . برای انجام این کار باید :

(1) parent->right_thread را برابر FLASH قرار دهید.

(2) child-> left_thread و child-> right_thread را برابر TRUE قرار دهید .

(3) child-> left_child را طوری تنظیم کنید که به paren اشاره کند.

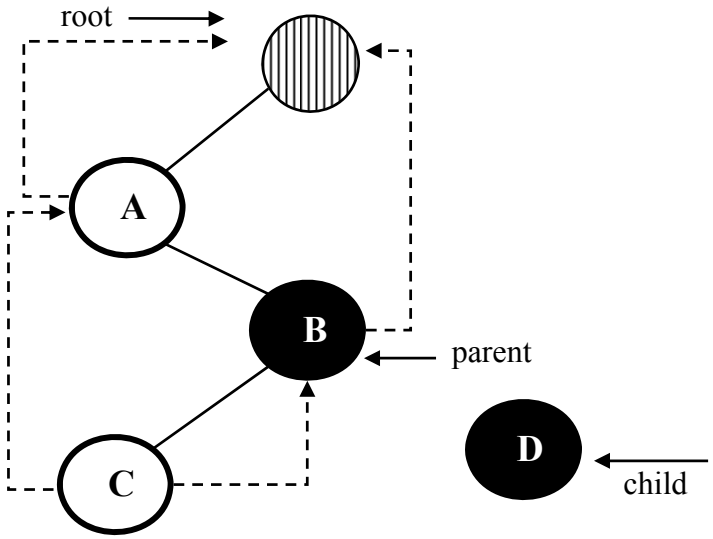
(4) child-> right_child را برابر parent->right_child قرار دهید.

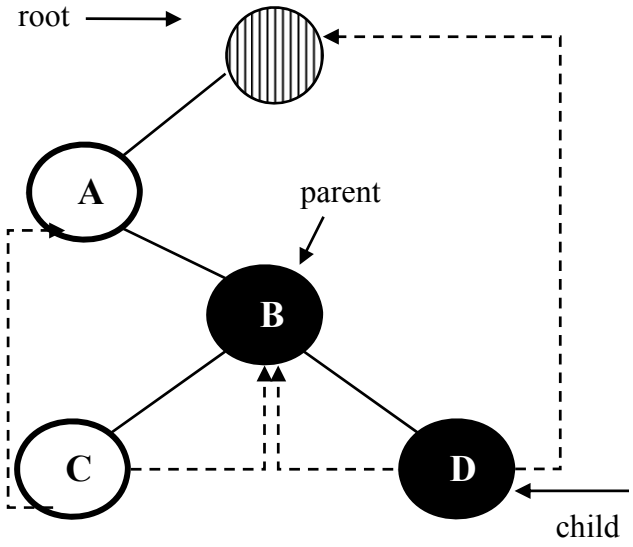
(5) parent->right_child را طوری تنظیم کنید که به child اشاره کند.

۵-۵ درج یک گره به داخل درخت نخی دودویی

مثال

در شکل زیر گره D را به عنوان فرزند راست گره B جایگذاری می کنیم :





۵-۶ نوع داده مجرد (ADT) هرم

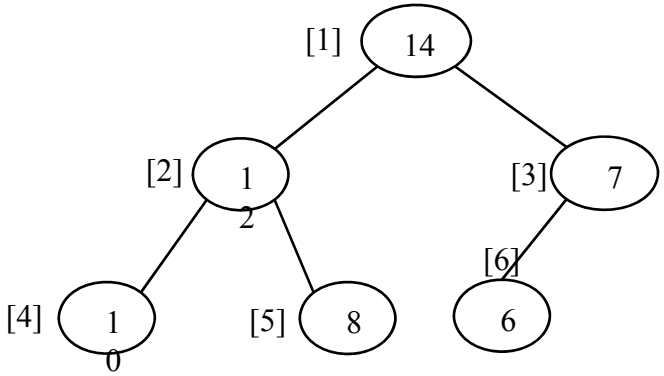
max tree درختی است که مقدار کلید هر گره آن کمتر از مقادیر کلیدهای فرزندانش نباشد.

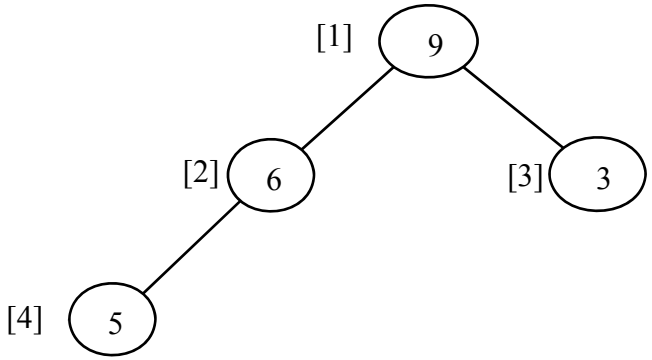
max heap یک درخت دودویی کامل است که یک max tree نیز می باشد.

min tree درختی است که مقدار کلید هر گره آن بیشتر از مقادیر کلیدهای فرزندانش نباشد.

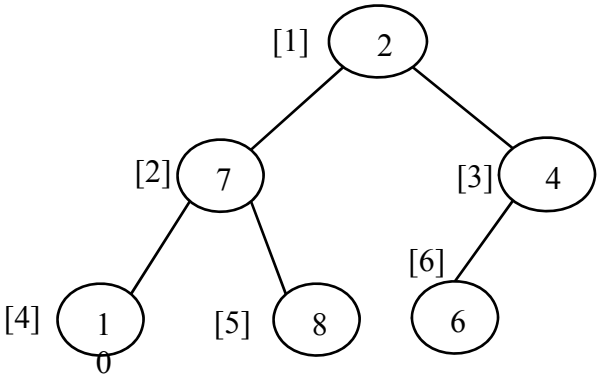
min heap یک درخت دودویی کامل است که در واقع یک min tree می باشد.

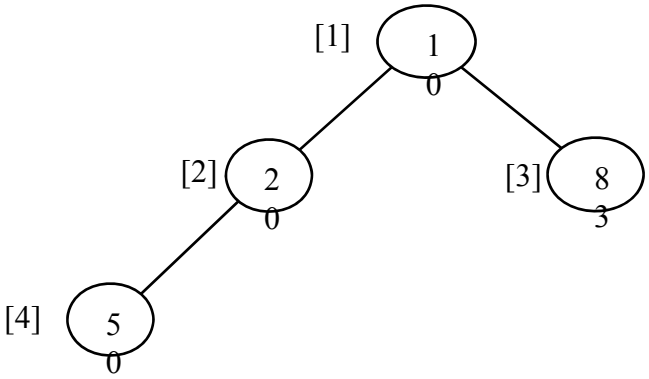
۶-۵ مثال از max heap و min heap





مثال از max heap





مثال از min heap

۶-۵ اعمال اساسی بر روی heap

- ایجاد یک هرم(heap) تهی
- جایگذاری عنصر جدید به هرم (heap)
- حذف بزرگترین عنصر از هرم (heap)

۶-۵ صف اولویت

غالبا هرم ها برای پیاده سازی **صف اولویت ها** استفاده می شوند.

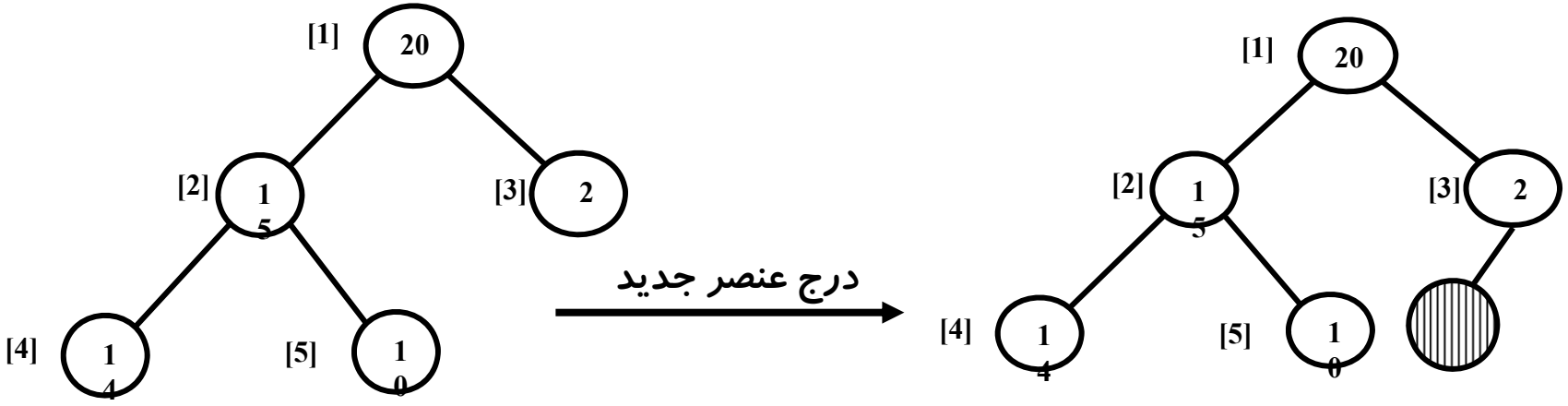
در صف اولویت ها عنصری که دارای بالاترین (یا پایین ترین) اولویت هست ، حذف می شود.

- آرایه ساده ترین نمایش برای یک صف اولویت می باشد.

۶-۵ نمایش های صف اولویت

Representation	Insertion	Deletion
Unordered array	Θ(1)	Θ(n)
Unordered linked list	Θ(1)	Θ(n
Stored array	O(n)	Θ(1)
Stored linked list	O(n)	Θ(1)
Max heap	$O(\log_2 n)$	$O(\log_2 n)$

۶-۵ درج عناصر به داخل یک Max Heap



الف – درخت heap قبل از درج

ب – محل اولیه گره جدید

اضافه کردن گره جدید در هر موقعیت دیگری ،تعریف heap را نقض می کند زیرا نتیجه یک درخت دودویی کامل نخواهد بود.

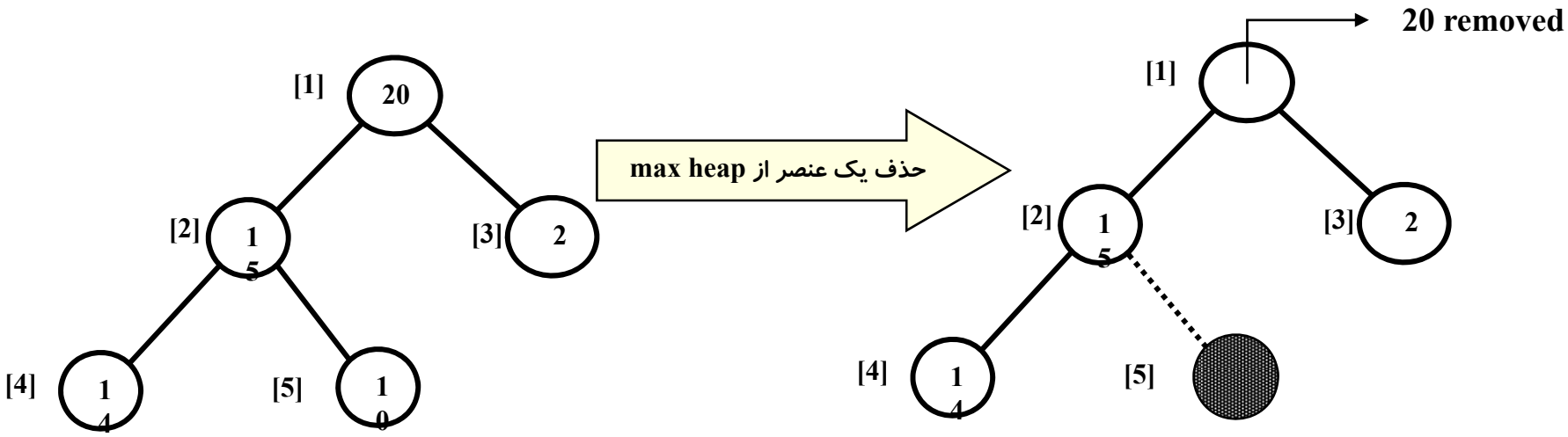
۶-۵ درج عنصر به یک Max heap

۶-۵ تحلیل تابع insert_max_heap

- از آنجا که heap یک درخت کامل با n عنصر می باشد ، دارای ارتفاع $\log_2(n)$ می باشد. این بدین معنی می باشد که حلقه while به میزان $\log_2(n)$ تکرار شود. بنابراین پیچیدگی تابع درج برابر $\log_2(n)$ می باشد.

۶-۵ حذف عنصری از Max Heap

هنگامی که عنصری از max heap حذف می شود ، آن را از ریشه درخت heap می گیریم.



۶-۵ حذف عنصری از Max Heap

۶-۵ تحلیل تابع delete_max_heap

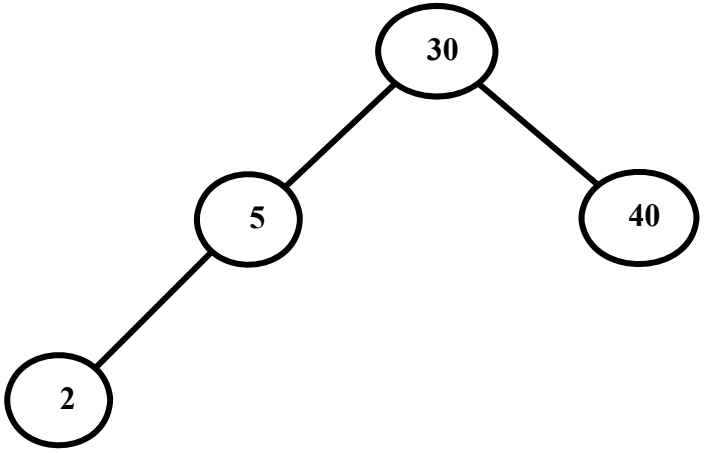
- پیچیدگی حذف برابر $O(\log_2 n)$ می باشد.
- زمان حذف یک عنصر دلخواه از درخت heap با n عنصر ، برابر O(n) می باشد.

۷-۵ درختان جستجوی دودویی

یک درخت جستجوی یک درخت دودویی است که ممکن است تهی باشد. اگر درخت تهی نباشد خصوصیات زیر را برآورده می کند :

- هر عنصر دارای یک کلید است و دو عنصر نباید دارای کلید یکسان باشند ، در واقع کلیدها منحصر به فردند.
- کلیدهای واقع در زیردرخت غیرتهی چپ باید کمتر از مقدار کلید واقع در ریشه زیردرخت راست باشد.
- کلیدهای واقع در زیردرخت غیرتهی راست باید بزرگتر از مقدار کلید واقع در ریشه زیردرخت چپ باشد.
- زیردرختان چپ و راست نیز خود درختان جستجوی دودویی میباشند.

۵-۷ درختان جستجوی دودویی



مثال

```
class BSTNode
{
        friend class BST;
private:
        BSTNode *left;
        int key;
        BSTNode *right;
public:
        int GetKey();
};

class BST
{
public:
        BST();
        bool Insert(int key);
        BSTNode* Search(int key);
        void DeleteNode(BSTNode* pNode,bool bRight);
private:
        BSTNode *root;
};
```

در کلاس نوشته شده فرض شده که هر گره تنها یک مقدار کلید دارد ولی در طراحی پیشرفته تر میتوان از قالبها استفاده نمود.

۵-۷ جستجوی یک درخت دودویی

فرض کنید خواسته باشیم دنبال عنصری با کلید key بگردیم. ابتدا از ریشه (root) شروع می کنیم ، اگر ریشه تهی باشد ، درخت جستجو فاقد هر عنصری بوده و جستجو ناموفق خواهد بود. در غیر این صورت keyرا با با مقدار کلید ریشه مقایسه کرده :

- اگر key کمتر از مقدار کلید ریشه باشد ، هیچ عنصری در زیردرخت راست وجود ندارد که دارای کلیدی برابر key باشد ، بنابراین زیردرخت چپ ریشه را جستجو می کنیم.
- اگر key بزرگتر از مقدار کلید ریشه باشد ، زیردرخت راست را جستجو می کنیم.

۵-۷ تحلیل search

اگر h ارتفاع یا عمق یک درخت جستجوی دودویی باشد ، عمل جستجو را در مدت O(h) انجام می شود.

الگوریتم جستجو در BST بصورت بازگشتی

```
BSTNode* BST::Search(int key)
{
        return Search(root , key);
}
BSTNode* BST::Search(BSTNode* p, int key)
{
        if  (!p)
                return 0;
        if(key ==p→key)
                return p;
        if(key <p→key)
                return Search(p→left , key);
        else
                return Search(p→right , key);
}
```

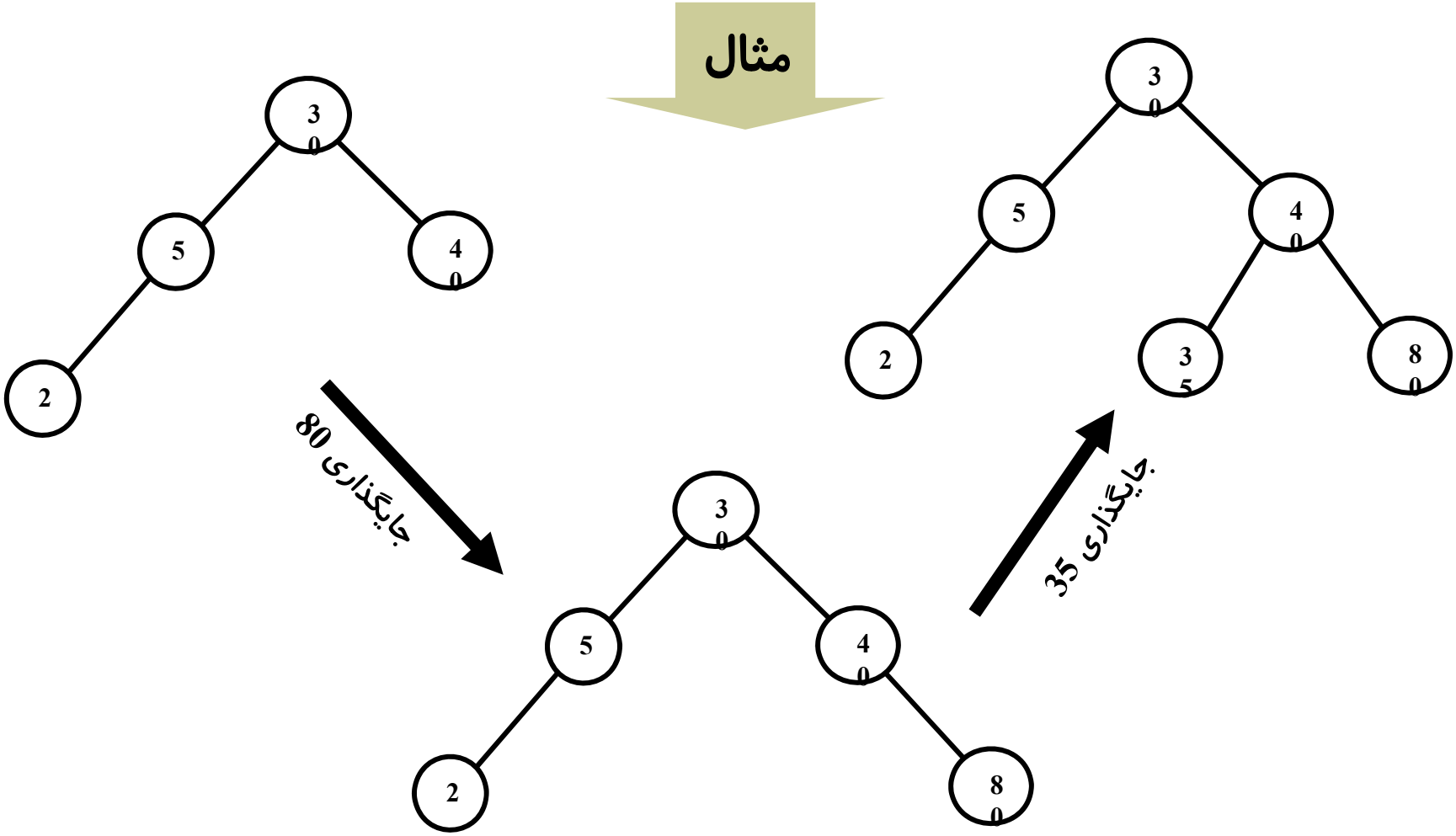
الگوریتم جستجو در BST بصورت غیر بازگشتی

```
BSTNode* BST::Search(int key)
{
      BSTNode *p=root;
      while(p!=NULL)
      {
            if(key < p→key)
                  p = p→left;
            else if(key > p→key)
                  p = p→right;
            else
                  return p;
      }
      return 0;
}
```

۷-۵ درج عنصری به داخل درخت جستجوی دودویی

برای درج عنصر جدیدی به نام key ، ابتدا باید مشخص نمود که آیا کلید با عناصر موجود متفاوت است یا خیر. برای انجام این کار باید درخت را جستجو کرد، اگرجستجو ناموفق باشد ، عنصر را در محلی که جستجو خاتمه پیدا نموده است ، درج می کنیم.

۷-۵ درج عنصری به داخل درخت جتجوی دودویی



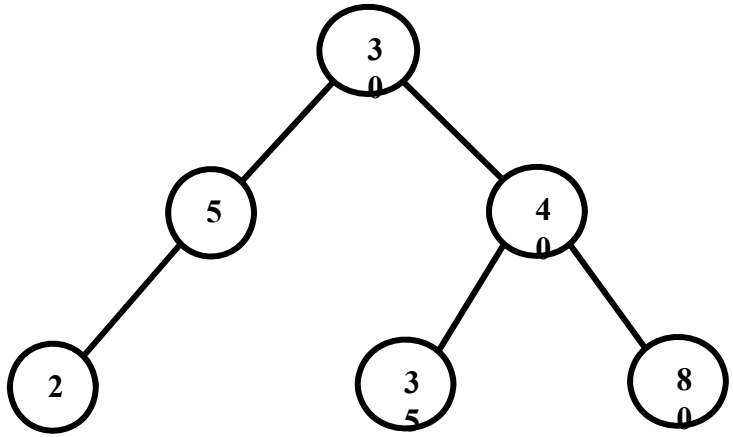
۵-۷ تحلیل insert_node

زمان لازم برای جستجوی num در یک درخت برابر $O(h)$ می باشد به نحوی که h برابر با عمق یا ارتفاع درخت است. بقیه الگوریتم نیاز به زمان $\Theta(1)$ دارد. بنابراین زمان کل مورد نیاز insert_node برابر با $\Theta(h)$ می باشد.

```
bool BST::Insert(int key)
{
        BSTNode *p=root;
        BSTNode *q;
        while(p!=NULL)
        {
                q = p;
                if(key < p→key)
                        p = p→left;
                else if(key > p→key)
                        p = p→right;
                else
                        return false;   //Error for duplicate key
        }
        p = new BSTNode (key);
        if (!root)
        {
                root = p;
                return true;
        }
        if(key < q→key)
                q→left  =  p;
        else
                q→right = p;
        return true;
}
```

۷-۵ حذف عنصری از درخت جستجوی دودویی

برای حذف ۳۵ از درخت زیر باید فیلد فرزند چپ والد این گروه را برابر NULL قرار داده و گره را آزاد نمود :



۵-۷ حذف عنصری از درخت جستجوی دودویی

زمانی که یک گره برگ با دو فرزند حذف می شوند ، گره را با بزرگترین عنصر در زیر درخت چپ و یا کوچکترین عنصر در زیردرخت راست آن گره جایگزین و تعویض کرد.

عمل حذف در زمان $O(h)$ انجام می گیرد ، به نحوی که h عمق درخت می باشد.

```
void DeleteNode (BSTNode* pNode,bool bRight)
{
   BSTNode *CurNode = pNode→left;
   if (bRight)
   CurNode = pNode→right;
   if( !CurNode→right  && !CurNode→ left)
   {
     delete CurNode;
     return;
   }
   if(CurNode→right  && !CurNode→ left)
   {
      if (bRight)
        pNode→right = CurNode→right;
     else
        pNode→left = CurNode→right;
    delete CurNode;
    return;
  }
  if(!CurNode→right  && CurNode→ left)
  {
     if (bRight)
       pNode→right = CurNode→left;
    else
      pNode→left = CurNode→left;
    delete CurNode;
    return;
  }
```

```
if(CurNode→right  && CurNode→ left)
  {

      BSTNode *tmp = CurNode→left;
      pNode = CurNode;
      bRight = false;
      while (tmp→right)
       {
            pNode = tmp;
            bRight = true;
            tmp = tmp→right;
       }
       CurNode→key = tmp→key;
       DeleteNode(pNode , bRight);
  }
}
```

۵-۷ درختان جستجوی متعادل

درختان جستجو با بیشترین عمق $O(\log_2 n)$ ، درختان جستجوی متعادل نامیده می شوند.

درختان جستجوی متعادلی وجود دارند که عمل جستجو ، درج و حذف را در زمان O(h) ممکن می سازند از جمله درختان red_black، 2-3، AVL

۸-۵ درختان انتخابی

فرض کنید دارای k مجموعه و رشته مرتب شده ای از عناصر هستیم که باید در یک رشته واحد ادغام شوند. هر دنباله یا ترتیب شامل تعدادی رکورد به ترتیب غیرنزولی و فیلد مشخصی به نام key می باشد.

- یک دنباله مرتب/*جرا (run)* نامیده می شود.
- فرض کنید که n ، تعداد رکوردها در k اجرا باشد، عمل ادغام می تواند با تکرار رکورد با کوچکترین کلید انجام شود.

۸-۵ درختان انتخابی

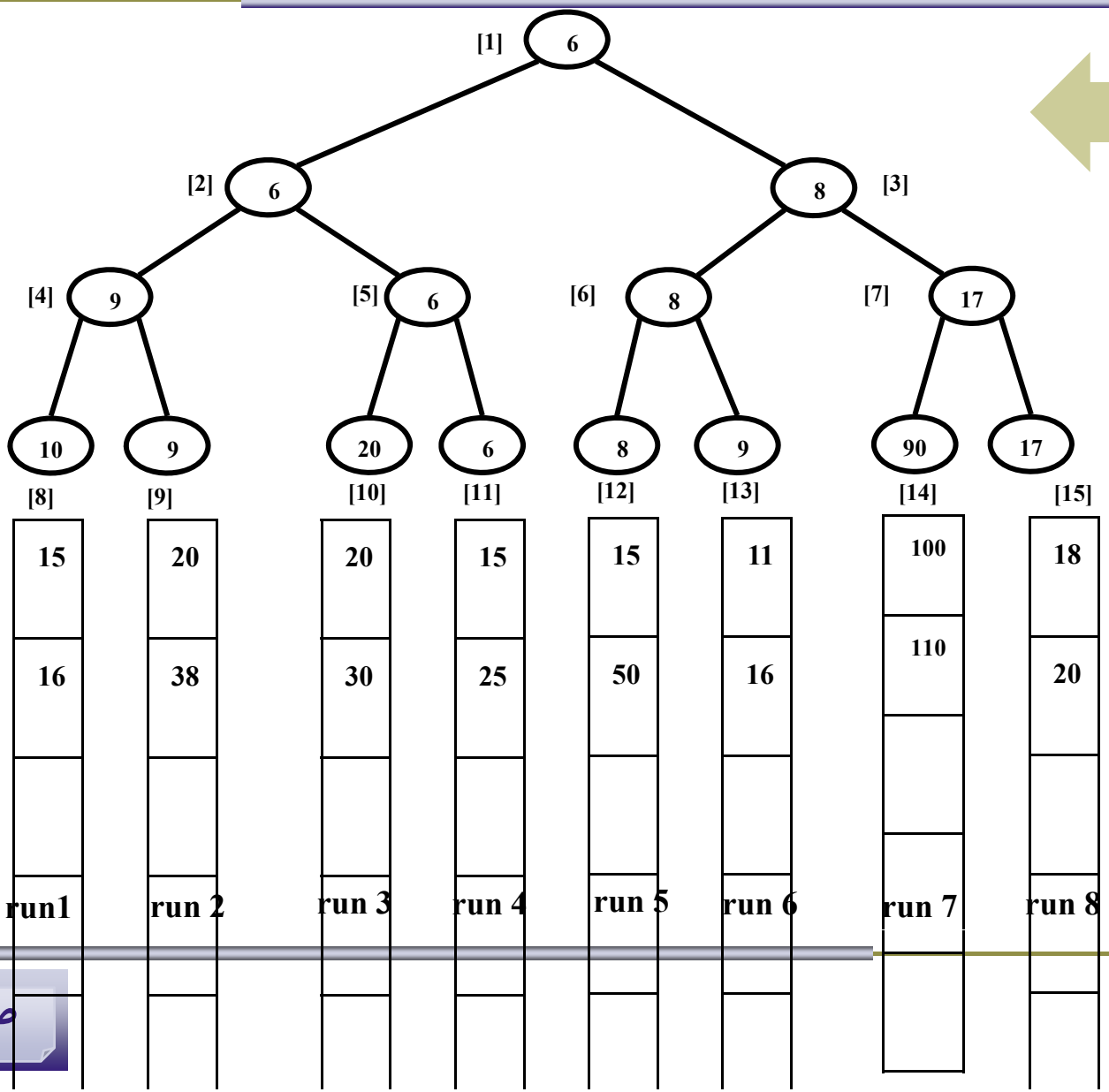
- کوچکترین کلید باید ازبین k امکان موجود پیدا شود و می تواند رکوردی قبل از هر k اجرا (k-runs) باشد.
- بهترین روش برای ادغام k اجرا (k-runs) ، نیازمند k-1 مقایسه برای تعیین و انتقال رکورد بعدی به خروجی می باشد.
- به ازای k>2 ، می توانیم با استفاده از ایده درخت انتخابی ، تعداد مقایسه های لازم جهت تعیین کوچکترین عنصر را کاهش دهیم.

۸-۵ درختان انتخابی

یک درخت انتخابی ، یک درخت دودویی است که هر گره آن کوچکتر از دو فرزند خود می باشد بنابراین ، گره ریشه نشان دهنده کوچکترین گره در درخت می باشد.

۸-۵ درختان انتخابی

مثال



۸-۵ درختان انتخابی

زمان تجدید ساختار **درخت برابر** $O(\log_2 k)$ **می باشد.**

زمان لازم برای ادغام **تمام n رکورد برابر** $O(n \log_2 k)$ **می باشد.**

زمان کل لازم جهت ادغام **k اجرا (run) برابر** $O(n \log_2 k)$ **می باشد.**

۹-۵ جنگل ها

جنگل مجموعه $n \geq 0$ درخت مجزا می باشد.

اگر ریشه درخت را حذف کنیم ، آنگاه دارای یک جنگل خواهیم بود.

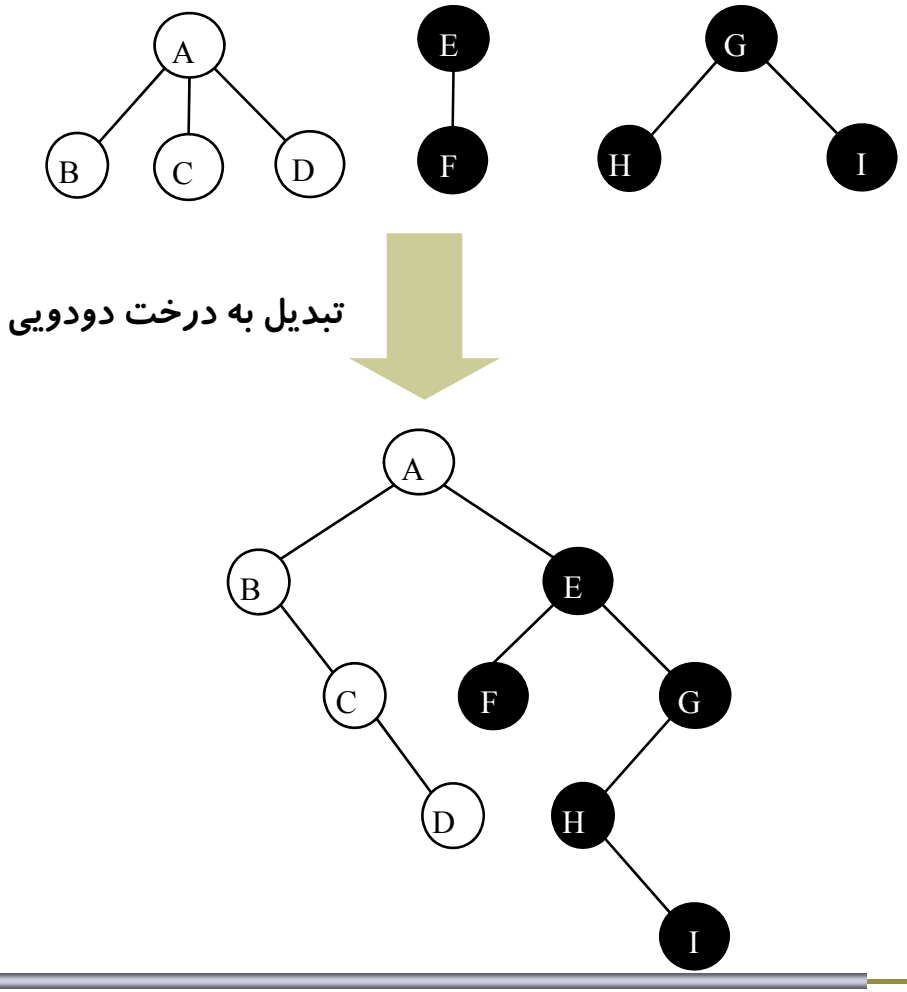
۹-۵ تبدیل جنگل به یک درخت دودویی

برای تبدیل این جنگل به یک درخت دودویی واحد :

ابتدا نمایش دودویی هر یک از درختان جنگل را به دست می آوریم

سپس تمام درختان دودویی را از طریق فیلد همزاد گره ریشه به یکدیگر متصل می کنیم.

۹-۵ تبدیل جنگل به یک درخت دودویی



۹-۵ تبدیل جنگل به یک درخت دودویی

اگر $T_1,\ldots,T_n$ جنگلی از درختان باشد ، آنگاه درخت دودویی متناظر با این جنگل یعنی $B(T_1,\ldots,T_n)$:

(1) اگر **n=0** باشد ، تهی خواهد بود((T_1)).

(2) دارای ریشه ای برابر با ریشه T_{11} ، دارای زیردرخت چپی برابر با $B(T_{12},\ldots,T_{1m})$ می باشد ، به نحوی که زیردرختان ریشه $B(T_2,\ldots,T_n)$ می باشند و در نهایت دارای زیردرخت راست می باشد.

۹-۵ پیمایش جنگل

پیمایش های postorder، inorder، preorder متناظر درخت دودویی T یک جنگل F دارای یک تناظر طبیعی با پیمایش های F می باشند.

۹-۵ پیمایش جنگل

پیمایش preorder مربوط به T معادل با بازیابی گره های F در درخت preorder می باشد:

1) اگر F تهی باشد ، برگردید.

2) ریشه درخت اول F را بازیابی کنید.

3) زیردرخت ، درخت اول را به صورت preorder پیمایش کنید.

4) سایر درختان F را به صورت preorder پیمایش کنید.

۹-۵ پیمایش جنگل

پیمایش inorder مربوط به T معادل گره های F در درخت inorder است که به صورت زیر تعریف می شود :

1) اگر F تهی باشد ، برگردید.
2) ریشه درخت ، درخت اول را به صورت inorder پیمایش کنید.
3) ریشه درخت اول را بازیابی کنید.
4) سایر درختان را به صورت inorder پیمایش کنید.

۹-۵ پیمایش جنگل

هیچ گونه معادل طبیعی برای پیمایش postorder درخت دودویی متناظر یک جنگل وجود ندارد . پیمایش postorder یک جنگل F را به صورت زیر بیان ی کنیم :

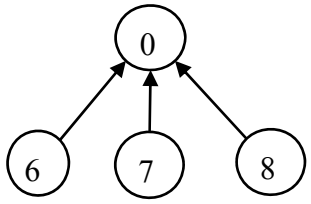
۱) اگر F تهی باشد ، برگردید.
۲) زیردرخت ، اولین درخت F را به صورت postorder پیمایش کنید.
۳) سایر درختان باقی مانده F را به صورت postorder پیمایش کنید.
۴) ریشه اولین درخت F را بازیابی کنید

۱۰-۵ نمایش مجموعه

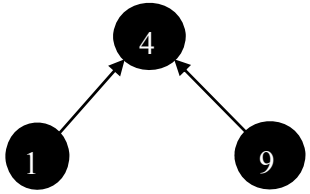
مثال

ده عضو از 0 تا 9 که به سه مجموعه مجزا از هم تفکیک شده باشند ، می توانند بدین صورت باشند :

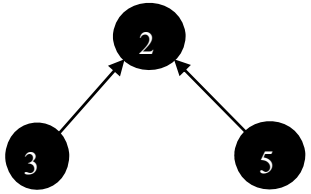
$$S_3 = \{2,3,5,\}, S_2 = \{1,4,9\}, S_1 = \{0,6,7,8\}$$



S_1



S_2



S_3

۵-۱۰ نمایش مجموعه

در هر مجموعه بر خلاف معمول که اشاره گرها به از والد به فرزندان در نظر گرفته می شدند، در اینجا اشاره گرها از فرزندان به والد تنظیم می شوند و یا در حقیقت گره ها با رابطه پدری اتصال یافته اند.

۱۰-۵ اعمال روی مجموعه ها

حداقل اعمالی که بر روی مجموعه انجام می شود ، به شرح زیر است :

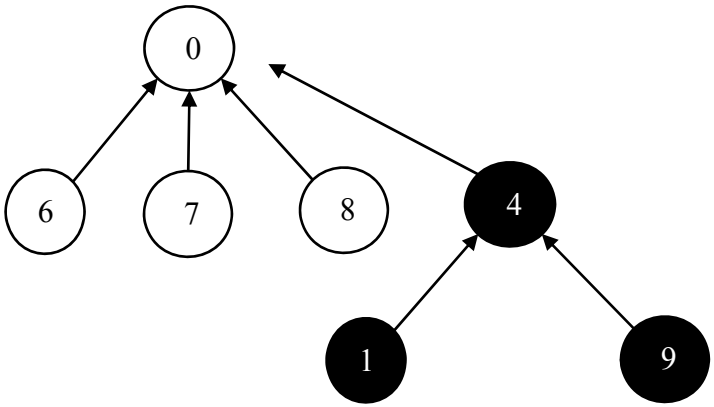
1)اجتماع مجموعه مجزا (union) : اگر S_i و S_j دو مجموعه مجزا باشند ، انگاه اجتماع آنها به صورت زیر تعریف می شود :

$S_i US_j =$ {همه اعضا به صورت x که x یا عضو S_i باشد یا عضو Sj}

۲) find(i)(پیداکردن i) : مجموعه ای که i عضو آن است را پیدا کنید

۱۰-۵ اعمال روی مجموعه ها

مثال



$$S_1 U S_2$$

۱۰-۵ قانون Weighting برای Union(i، j)

تعریف

قانون Weighting برای Union(i، j) : اگر تعدا گره ها در درخت i کمتر از تعداد گره ها در درخت j باشد ، j را والد i ، در غیر این صورت i را والد j قرار می دهیم.

۱۰-۵ پیاده سازی قانون Weighting

برای پیاده سازی قانون Weighting باید بدانیم که در هر درخت چند گره وجود دارد. این کار را بدین ترتیب انجام می دهیم که در ریشه هر درخت یک فیلد count قرار می دهیم. اگر i یک گره ریشه باشد ، آنگاه در count[i] تعداد گره های آن درخت خواهد بود. Count به صورت یک عدد منفی در فیلد parent گذاشته می شود. زمانی که قانون Weighting را بیان می کنیم ، عمل اجتماع به این صورت را union2 می نامیم.

۵-۱۰ مجموعه ها

اصل موضوعی : **فرض کنید T یک درخت با n گره باشد که توسط union2 ایجاد شده باشد ، در این صورت هیچ گرهی در T ، سطحی بیشتر از $[\log_2 n]+1$ نخواهد داشت.**

اگر در یک درخت n عضو وجو.د داشته باشد ، بیشترین زمان برای یافتن یک عضو به صورت $O(\log_2 n)$ خواهد بود.

اگر ترکیبی از n-1 عمل union و m عمل find داشته باشیم ، در بدترین حالت ممکن ، زمان به صورت $O(n+m\log_2 n)$درمی آید.

تعریف (قانون تخریب) : **اگر j گرهی روی مسیر از i تا ریشه خود باشد ، آنگاه j را فرزند ریشه قرار دهید.**

۱۱-۵ شمارش درختان دودویی

تعداد درختان دودویی مجزا :

تعداد درختان دودویی با n گره ، تعداد جایگشت های از 1 تا n با استفاده از یک پشته و بالاخره تعداد راههای ضرب n+1 ماتریس ، باهم مساوی اند.

۵-۱۱ تعداد درختان دودویی مجزا

برای به دست آوردن تعداد درختان مجزا با n گره از تابع زیر استفاده می کنیم :

$$B(x) = \sum_{i \geq 0} b_i x^i$$

تابع فوق تولیدکننده تعداد درختان دودویی است.

که b_n برابر است با :

$$b_n = \frac{1}{n+1}\begin{bmatrix} 2n \\ n \end{bmatrix}$$

در نتیجه داریم :

$$b_n = O\left(\frac{4^n}{n^{\frac{3}{2}}}\right)$$