فصل پنجم : درخت(Tree)

اهداف

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

مفهوم درخت

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

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

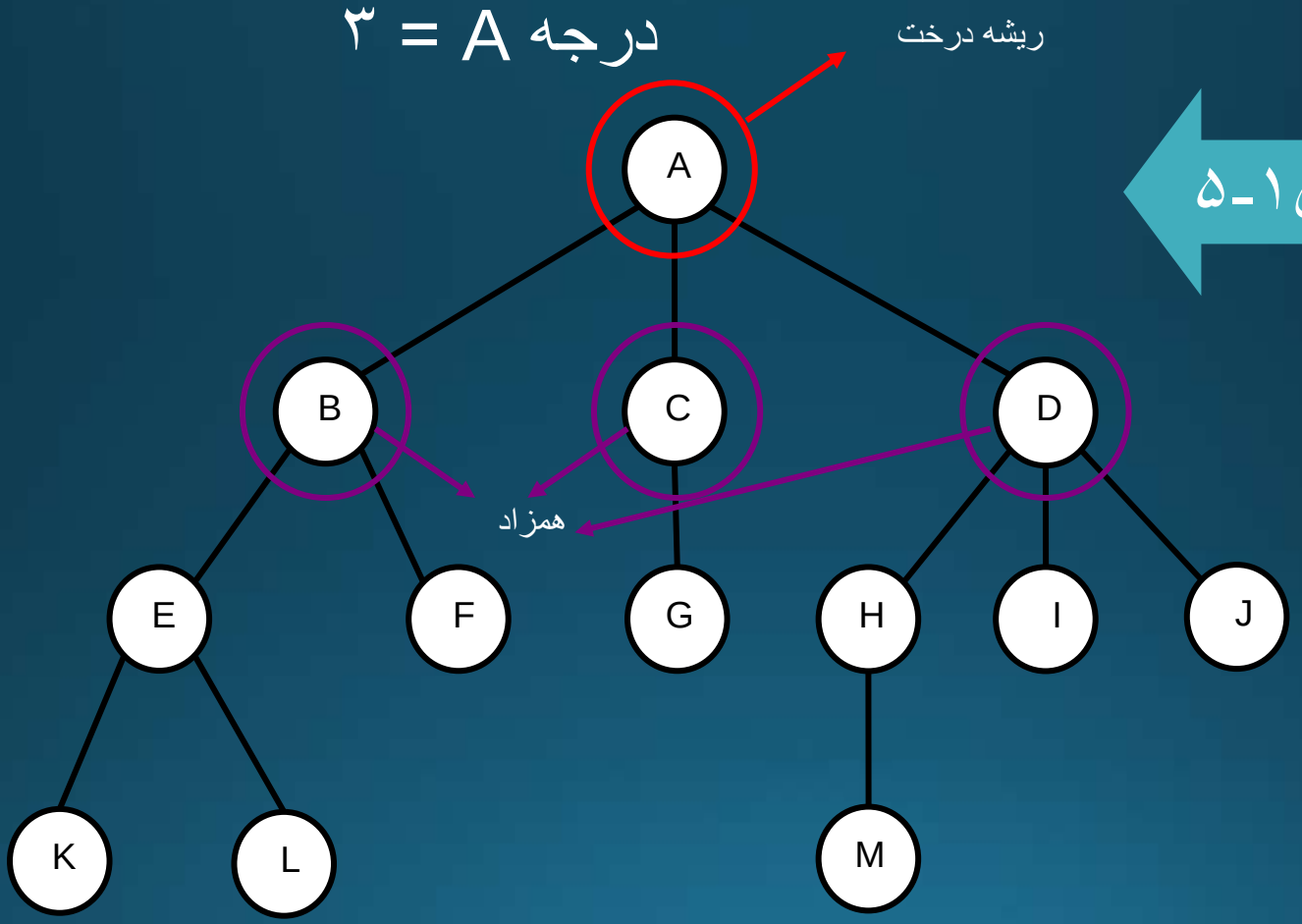
  - نکته: عبارت «مجموعه مجزا» از هرگونه اتصال زیردرخت ها به هم جلوگیری می کند یعنی در درخت حلقه نداریم

تعریف

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

مثالی از یک درخت

برنامه ۱-۵



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.

یال و مسیر : خطی که از یک گره به گره بعدی رسم می شود **یال** و دنباله ای از یال های متوالی **مسیر** نامیده می شود.

شاخه: مسیری که به یک برگ ختم می شود **شاخه** نام دارد.

اصطلاحات درخت ها

درخت مرتب: درختی که ترتیب زیردرخت ها در آن مهم باشد.

درخت های مشابه: درخت های T1 و T2 مشابه هستند اگر دارای ساختمان یکسان باشند. به بیان دیگر شکل مساوی داشته باشند.

درخت Kتایی: درختی که تعداد فرزندان هر گره در آن حداکثر K باشد.

درخت متوازن: درختی که اختلاف سطح برگ های آن حداکثر یک باشد.

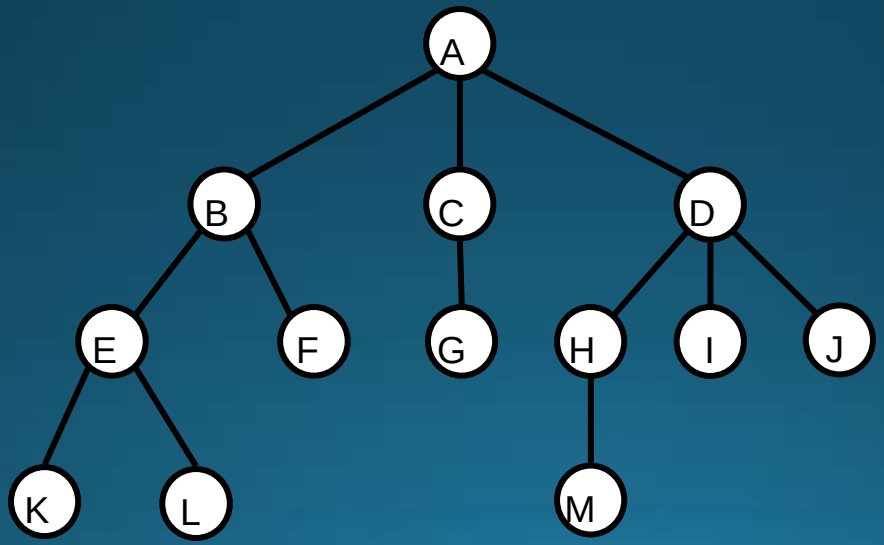
درخت کاملا متوازن: درختی که اختلاف سطح برگ های آن صفر باشد.

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)))



درخت های دودویی(Binary)

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.

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)

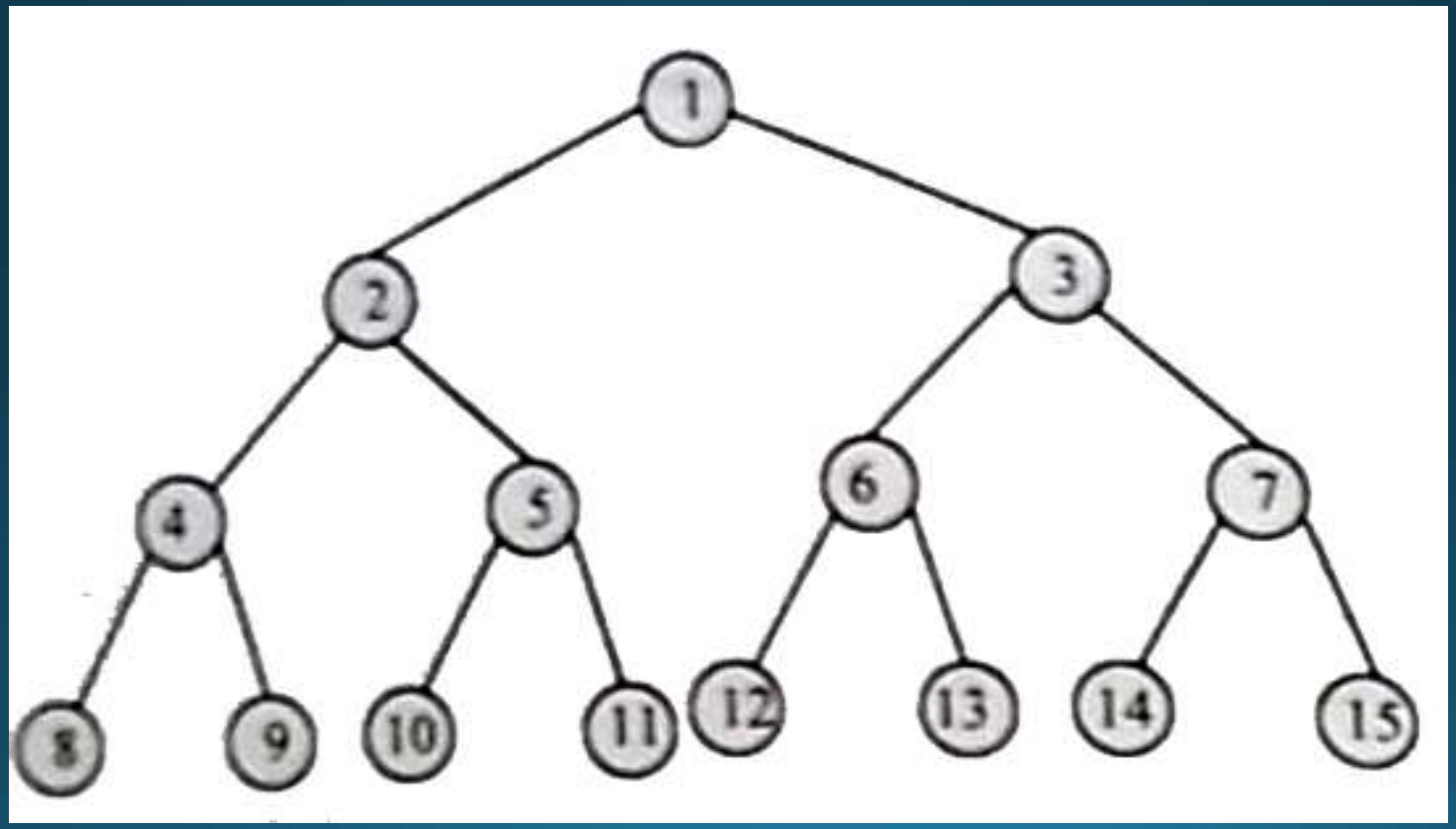
BinTree Rchild(bt)

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.

انواع درخت دودویی

درخت پر: درختی دودویی که همه گره ها به جز گره های سطح آخر دقیقا دو فرزند داشته باشند.

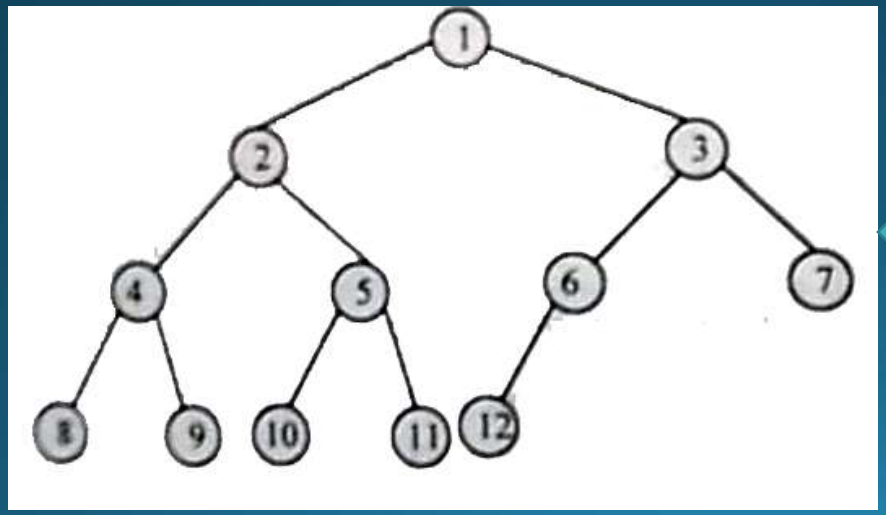


انواع درخت دودویی

درخت کامل: درخت T را کامل گویند اگر تمام سطح های آن به جز احتمالا آخرین سطح حداکثر تعداد گره ممکن را داشته باشند.همچنین تمام گره های آخرین سطح در سمت چپ ترین مکان باشند.

نکته: درخت پر هم کامل است و هم کاملا متوازن.

درخت Kتایی کامل: درختی که تعداد فرزندان هر گره به جز برگ ها، دقیقا K باشد.

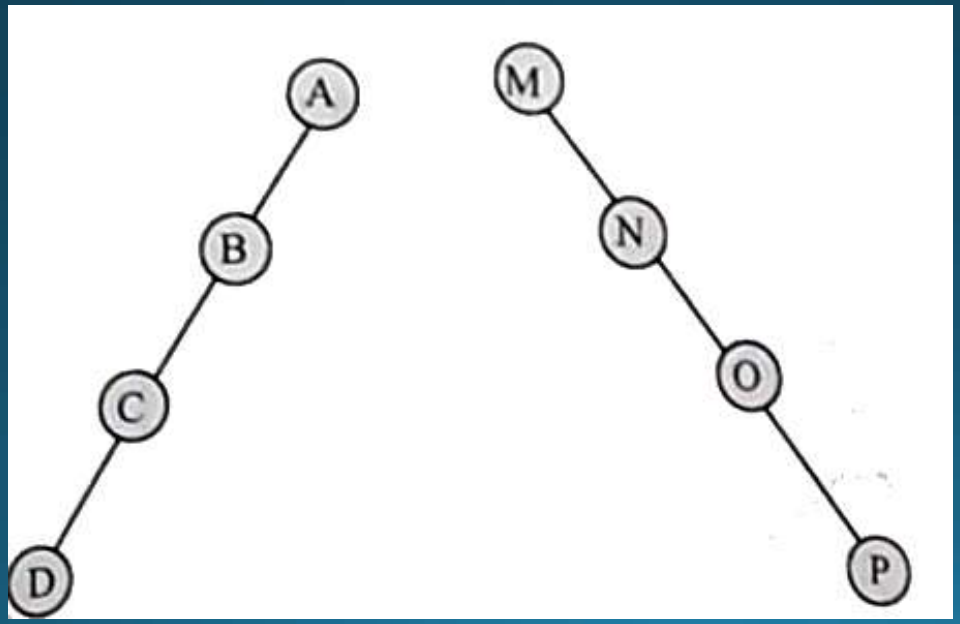


درخت دودویی کامل

انواع درخت دودویی

درخت مورب:

- درخت مورب چپ: هر گره، فرزند چپ والد خود باشد.
- درخت مورب راست: هر گره، فرزند راست والد خود باشد.



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.

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 تعدادگره های پایانی و n_2 تعداد گره های درجه ۲ باشد ، آنگاه خواهیم داشت :

$$n_0 = n_2 + 1$$

خواص درختان دودویی

- یک درخت دودویی پر به عمق k، یک درخت دودویی پر است مشروط به اینکه 2k-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

نمایش درخت دودویی

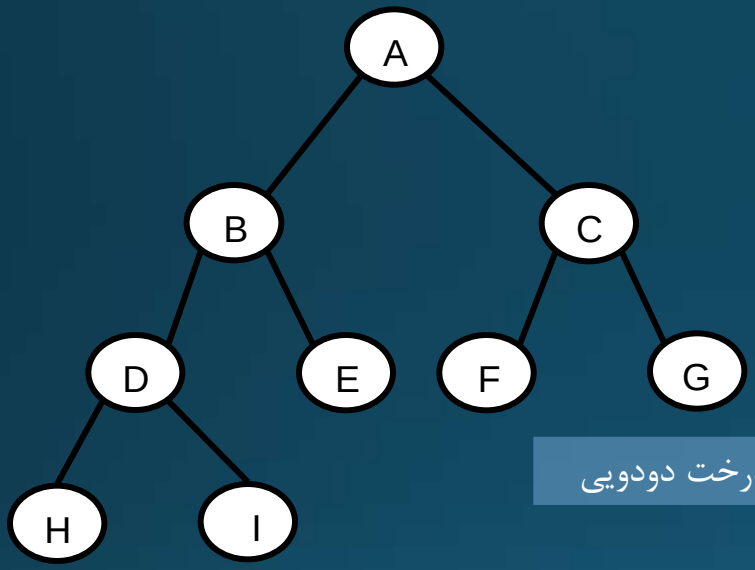
نمایش درخت دودویی به دو صورت است :

- نمایش آرایه
- نمایش لیست پیوندی

نمایش آرایه

- شیوه شماره گذاری ارایه شده در شکل، 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 .

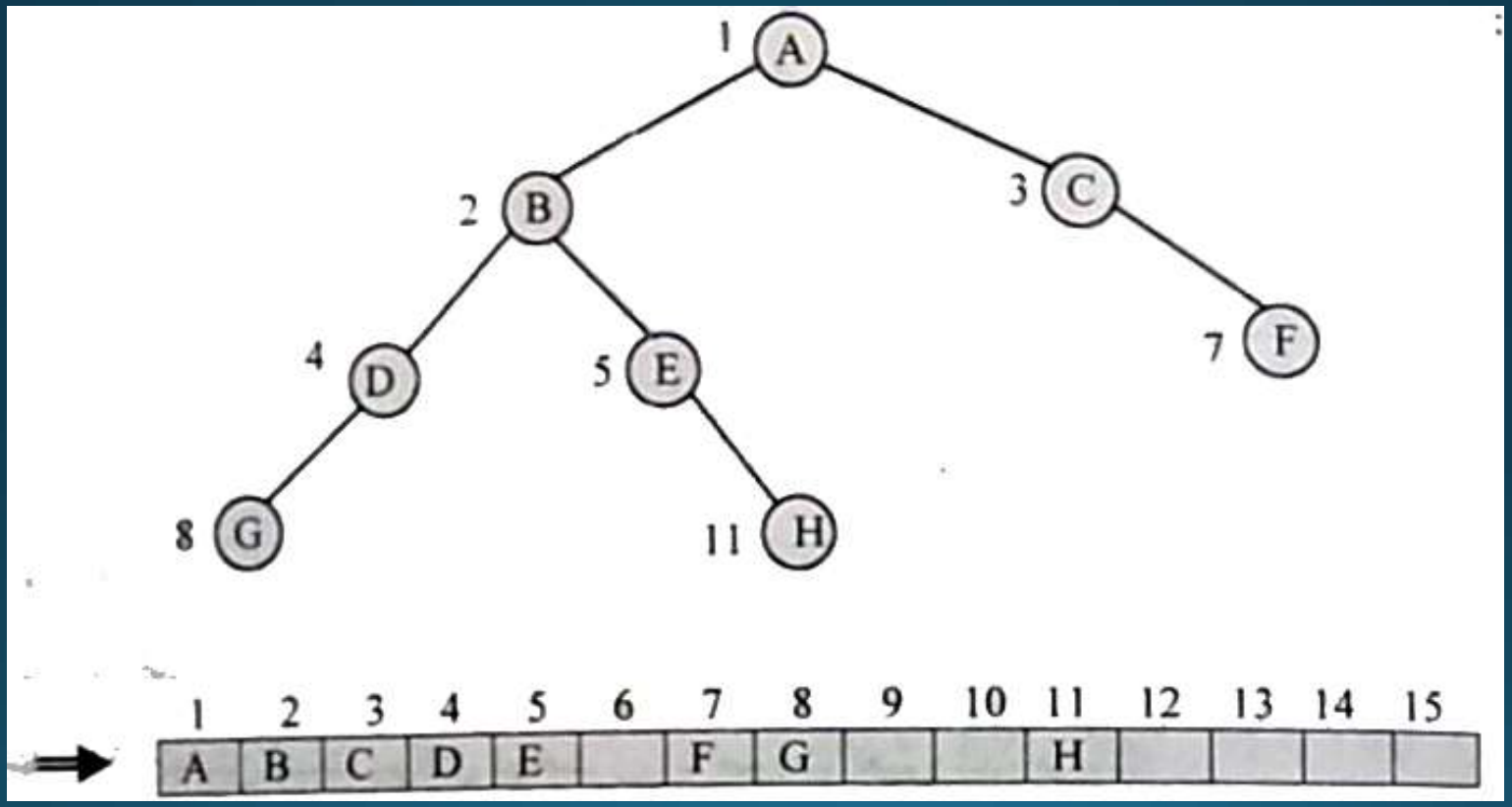
نمایش آرایه



نمایش آرایه ای درخت دودویی

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

نمایش آرایه



نمایش آرایه

اگر یک درخت دودویی کامل با n گره (یعنی $=[\log_2 n]+1$ عمق) 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) اگر2i≤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

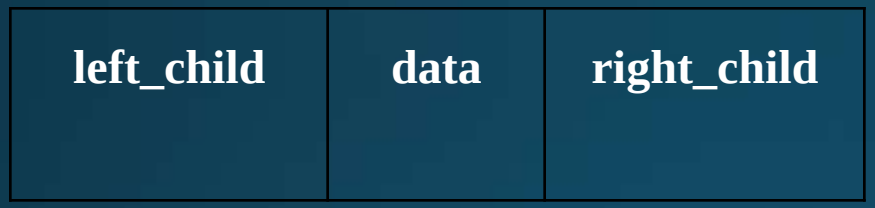
نمایش آرایه

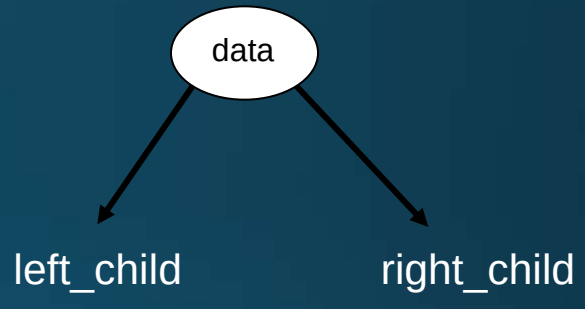
در بدترین حالت ، یک درخت مورب به عمق k ، به $2^k - 1$ محل و موقعیت نیاز دارد که از این مقدار، فقط k محل اشغال می شود.

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.

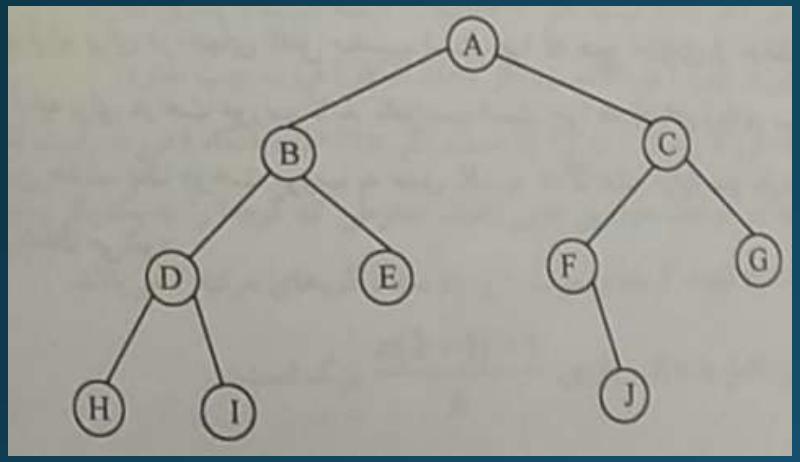
نمایش لیست پیوندی

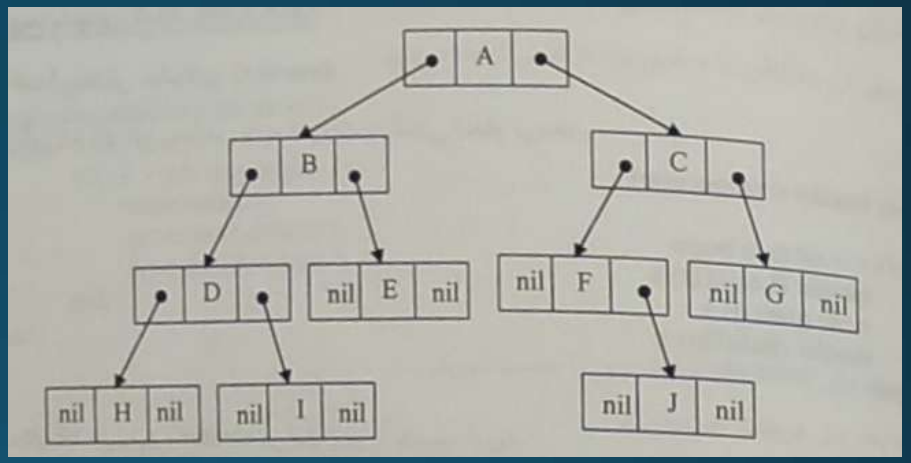




نمایش یک گره درخت دودویی

نمایش لیست پیوندی





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 :

LVR ،LRV ،VLR ،VRL ،RVL ،RLV

پیمایش درخت دودویی

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.

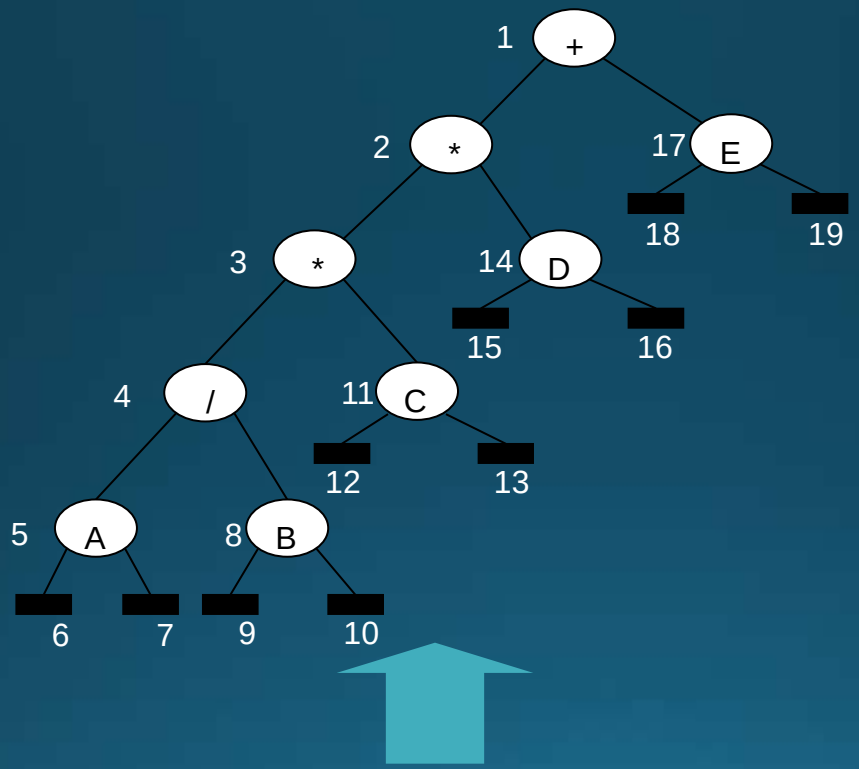
پیمایش درخت دودویی

- 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.

پیمایش درخت دودویی

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



درخت دودویی برای یک عبارت محاسباتی

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 ادامه پیدا می کند.

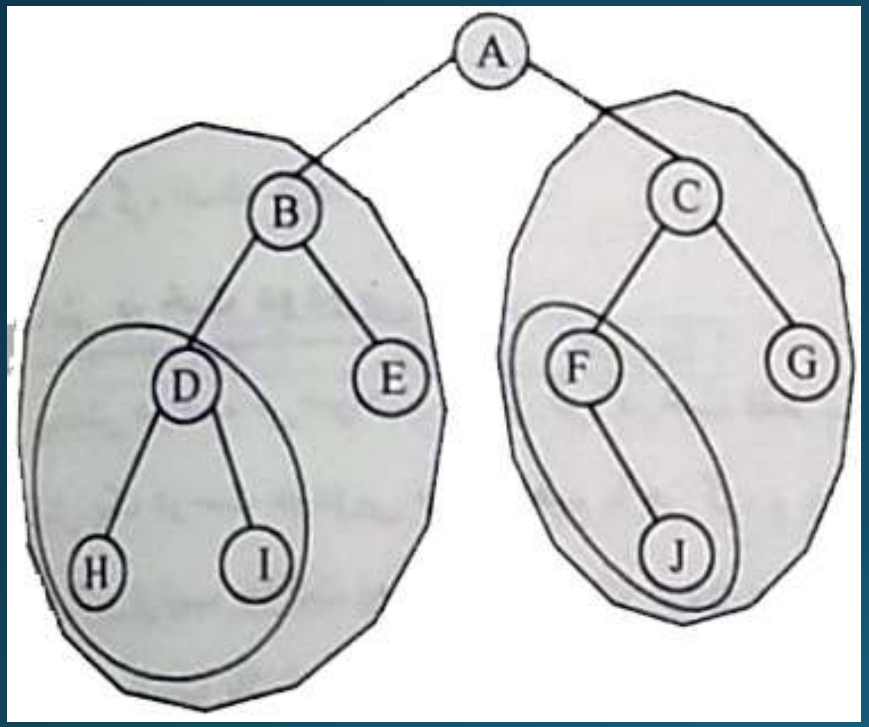
این متناظر با شکل infix یک عبارت است.

پیمایش Inorderیک درخت دودویی

برنامه ۲-۵

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

پیمایش Inorder یک درخت دودویی



H D I B E A F J C G

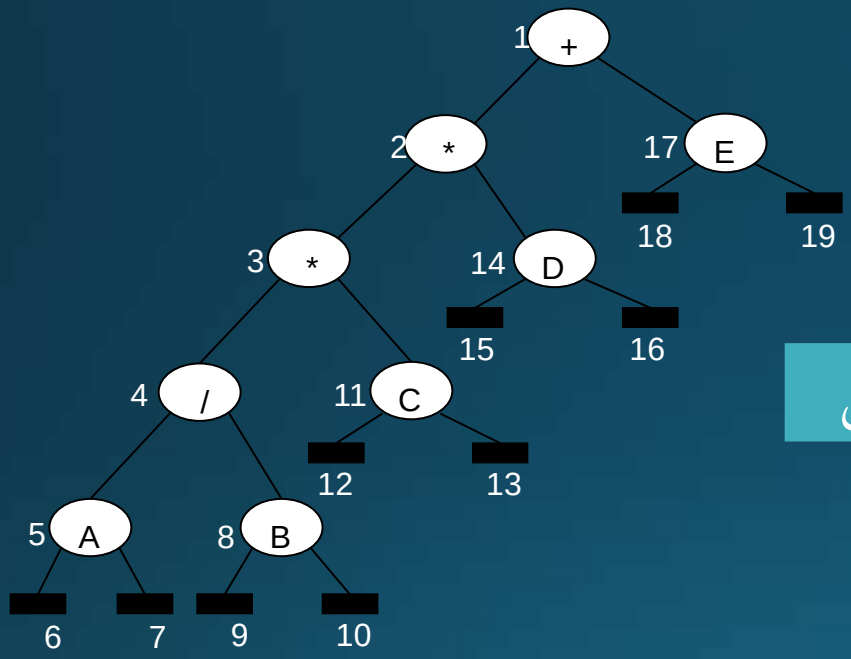
پیمایش Preorder

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.

پیمایش Preorder

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



خروجی

+ * * / A B C D E

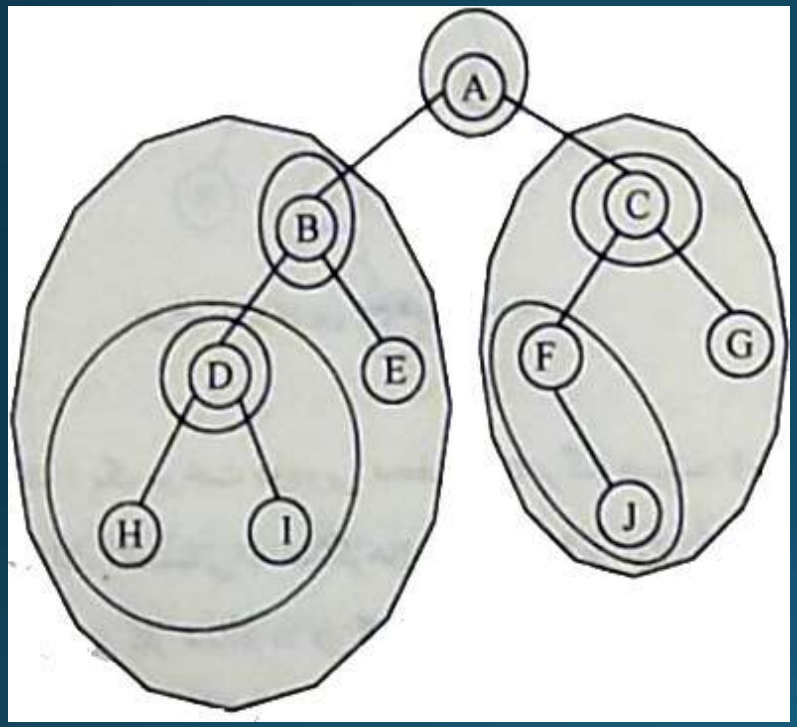
این به شکل یک عبارت prefix است.

پیمایش Preorder یک درخت دودویی

برنامه ۲-۵

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

پیمایش Preorder یک درخت دودویی



A B D H I E C F J G

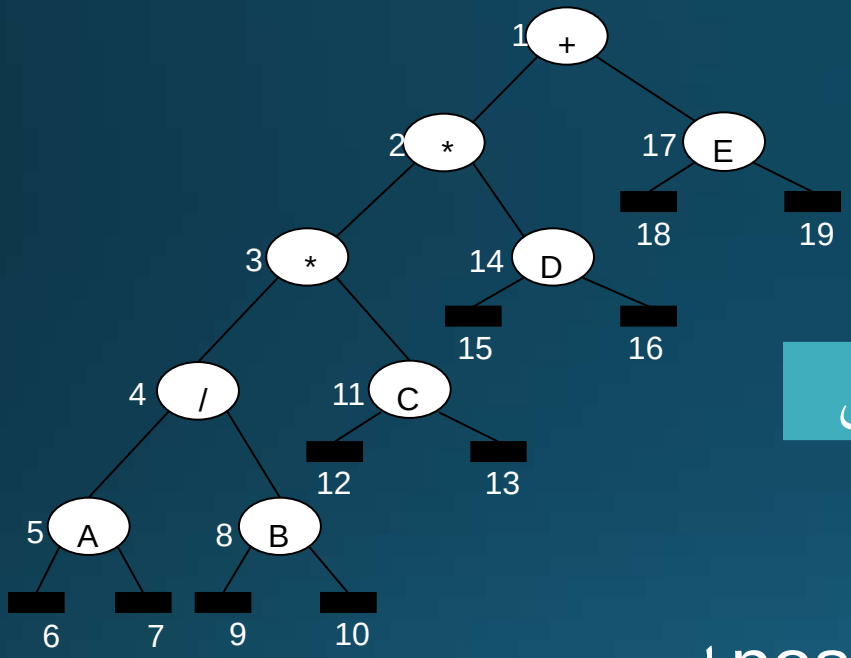
پیمایش postorder

برنامه ۲-۵

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.

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.

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;
        }
}
```

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.

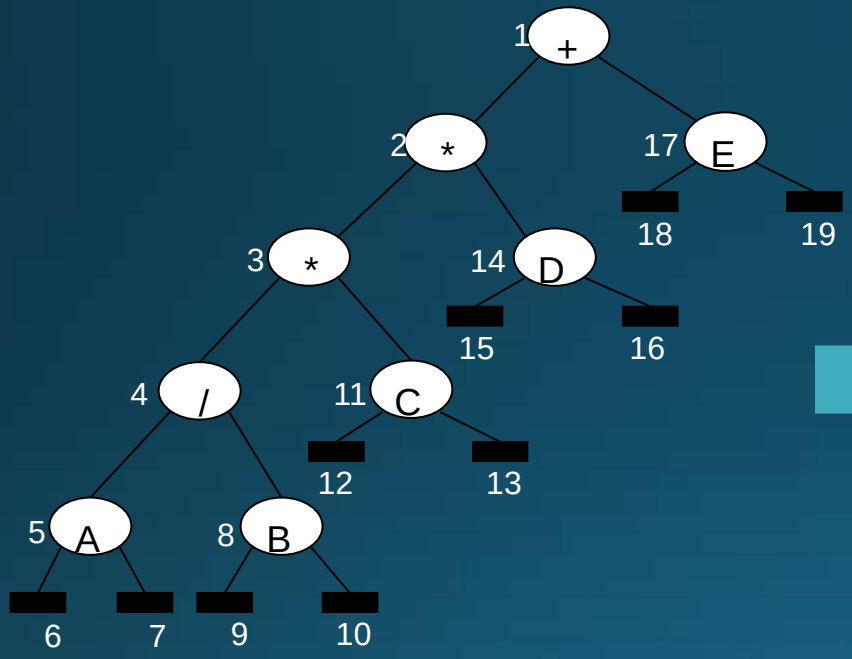
پیمایش ترتیب سطحی

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.

پیمایش ترتیب سطحی

پیمایش ترتیب سطحی درخت زیر به صورت زیر است :



+ *E *D / C A B

اعمال مفید بر روی درختان دودویی

۱- کپی کردن درختان دودویی

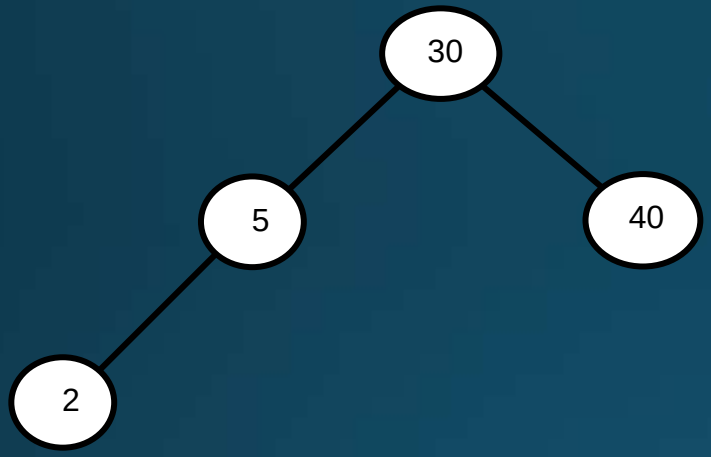
۲- تعیین برابری و تساوی دو درخت

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 فرد هستند.
- 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.

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 می توان از قالب ها استفاده نمود.

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.

تحليل search

اگر h ارتفاع يا عمق يک درخت جستجوي دودويي باشد ، عمل جستجو را در مدت O(h) انجام مي شود.

الگوریتم جستجو در BST بصورت بازگشتی

برنامه ۳-۵

```
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);
}
```

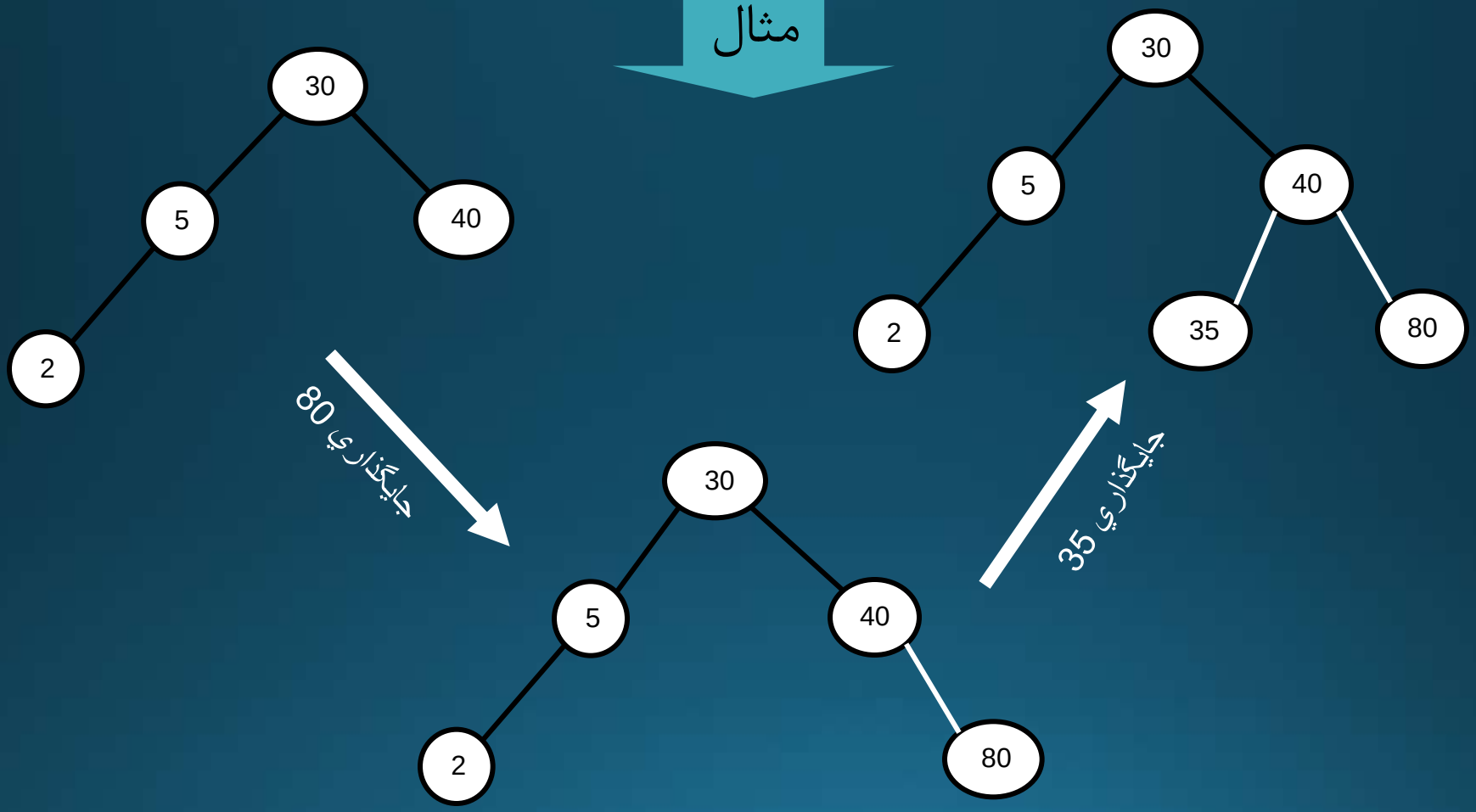
الگوریتم جستجو در BST – غیر بازگشتی

```
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;
}
```

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.

درج عنصری به داخل درخت جتجوی دودویی



تحلیل insert_node

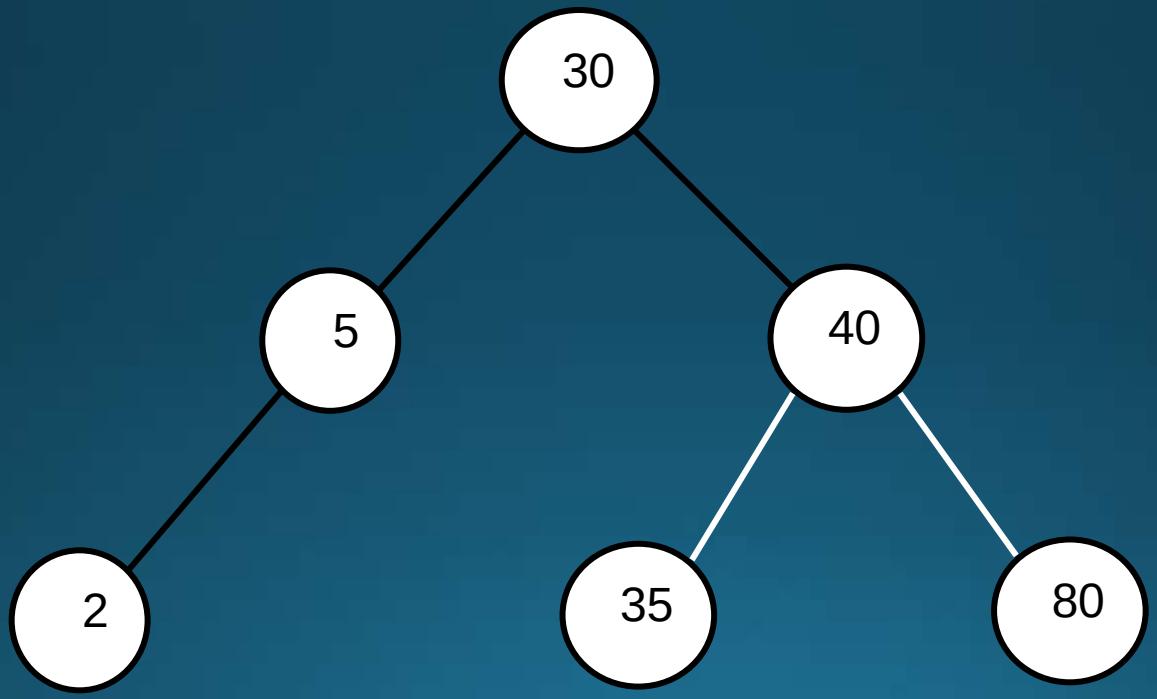
زمان لازم برای جستجوی num در یک درخت برابر O(h) می باشد به نحوی که h برابر با عمق یا ارتفاع درخت است. بقیه الگوریتم نیاز به زمان Θ(1) دارد. بنابراین زمان کل مورد نیاز insert_node برابر با Θ(h) می باشد.

برنامه ۳-۵

```
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;
}
```

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 :



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);
  }
}
```

برنامه ۴-۵

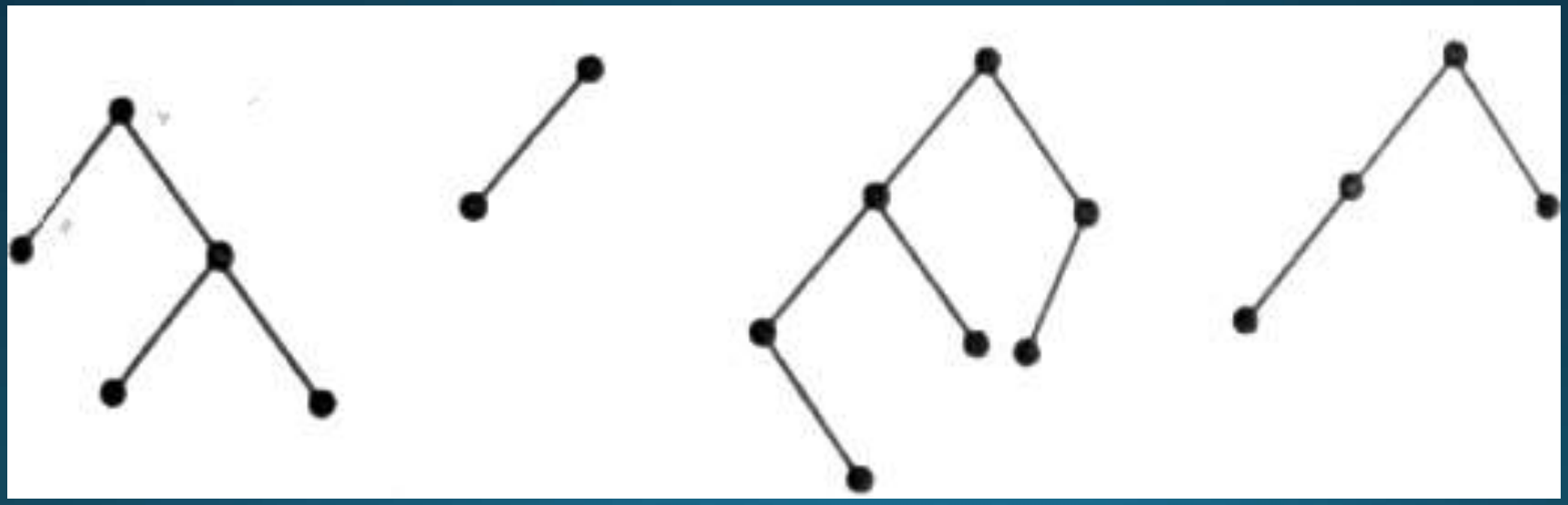
درختان جستجوی متعادل

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 درختAVL

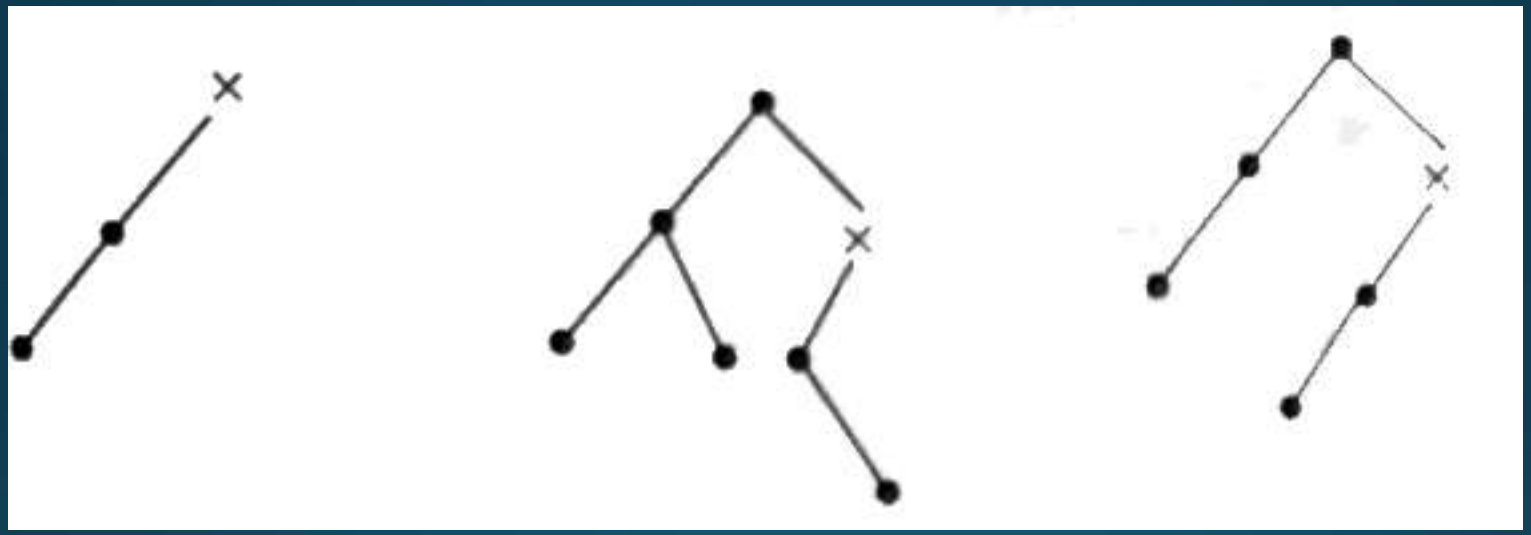
درخت AVL

تعریف: درخت دودویی جستجویی است که ارتفاع آن متوازن باشد. درخت با ارتفاع متوازن درختی است که حداکثر اختلاف ارتفاع دو زیردرخت چپ و راست هر گره آن یک باشد.مانند درخت های زیر:



درخت AVL

درخت های زیر AVL نیستند:

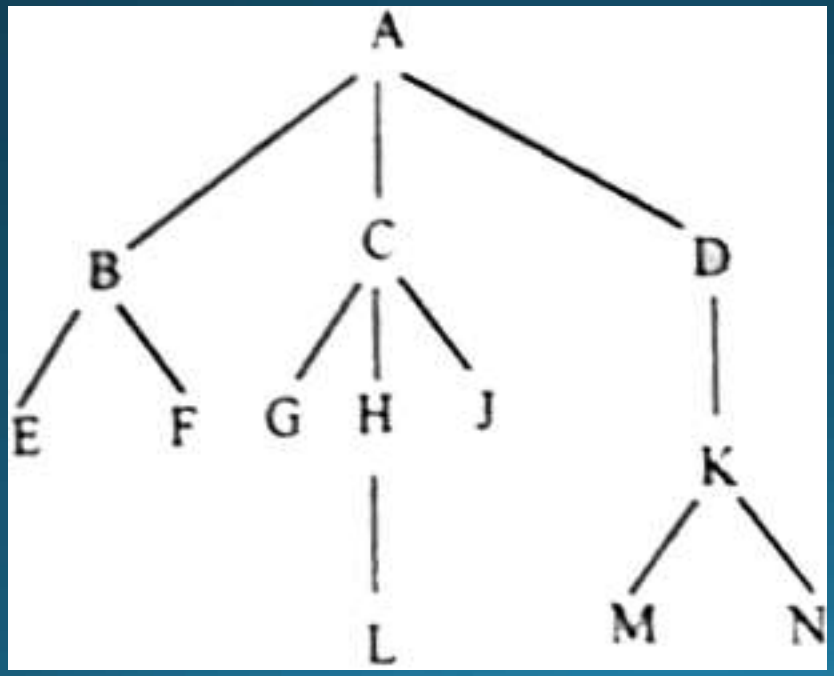


نکته:

- حداکثر ارتفاع یک درخ AVL با n گره 1.44Log(n) می باشد.
- اگر N(h) حداقل تعداد گره های موردنیاز برای ساخت یک درخت AVL باشد آنگاه: N(h) = N(h-1) + N(h-1) + 1

درخت عمومی (General)

تعریف: درخت k تایی است که در آن فقط یک گره به نام ریشه با درجه ورودی صفر وجود دارد و سایر گره ها دارای یک کمان ورودی هستند. برای سهولت فرض می شود درخت عمومی مرتب است یعنی گره های فرزند ترتیب دارند.



جنگل ها

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

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

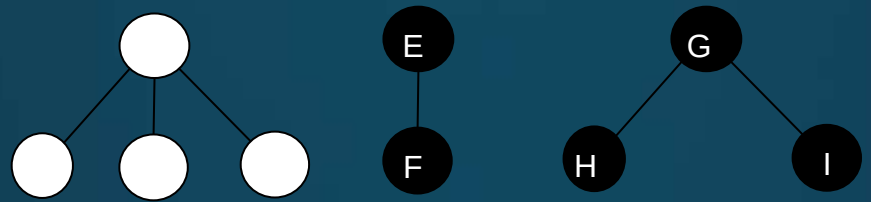
تبدیل جنگل به یک درخت دودویی

برای تبدیل این جنگل به یک درخت دودویی واحد :

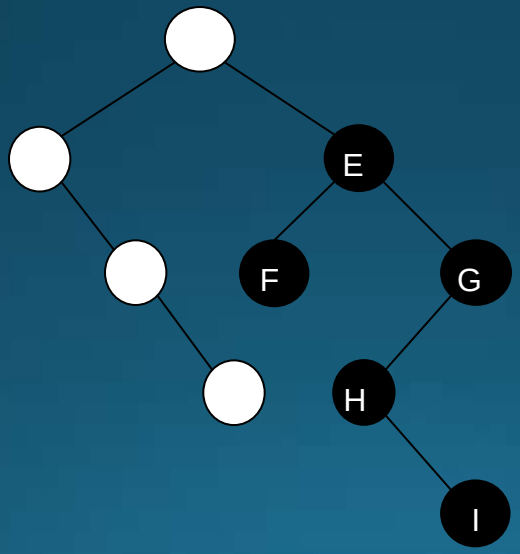
ابتدا نمایش دودویی هر یک از درختان جنگل را به دست می آوریم

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

تبدیل جنگل به یک درخت دودویی



تبدیل به درخت دودویی



پیمایش جنگل

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

نوع داده مجرد (ADT) هرم

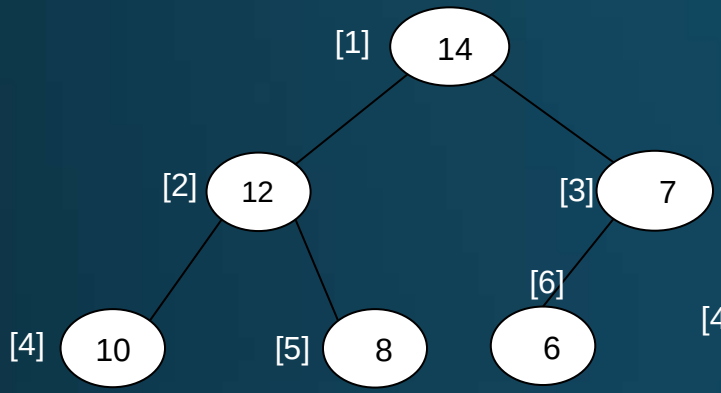
max tree درختی است که مقدار کلید هر گره آن کمتر از مقادیر کلیدهای فرزندانش نباشد.

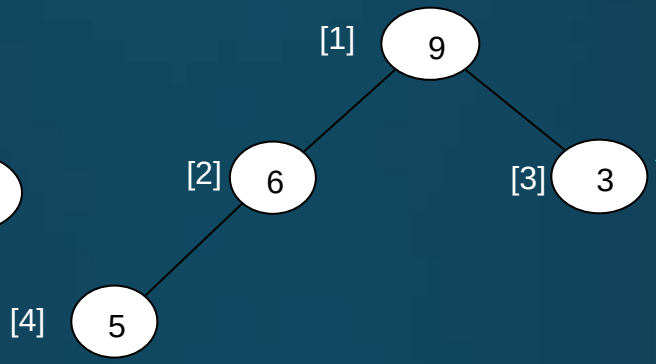
max heap یک درخت دودویی کامل است که یک max tree نیز می باشد.

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

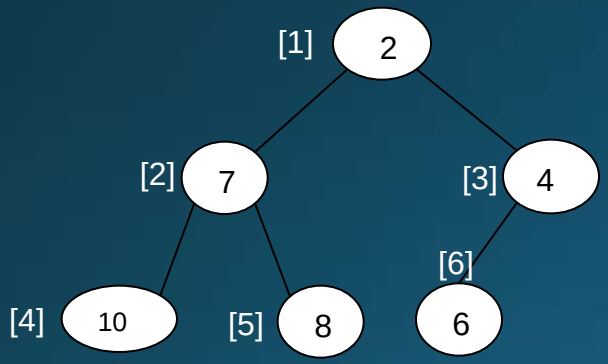
min heap یک درخت دودویی کامل است که در واقع یک min tree می باشد.

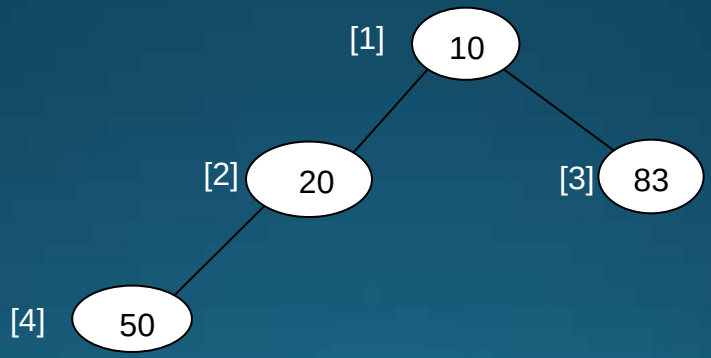
مثال از max heap و min heap





مثال از max heap





مثال از min heap

اعمال اساسی بر روی heap

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

صف اولویت

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

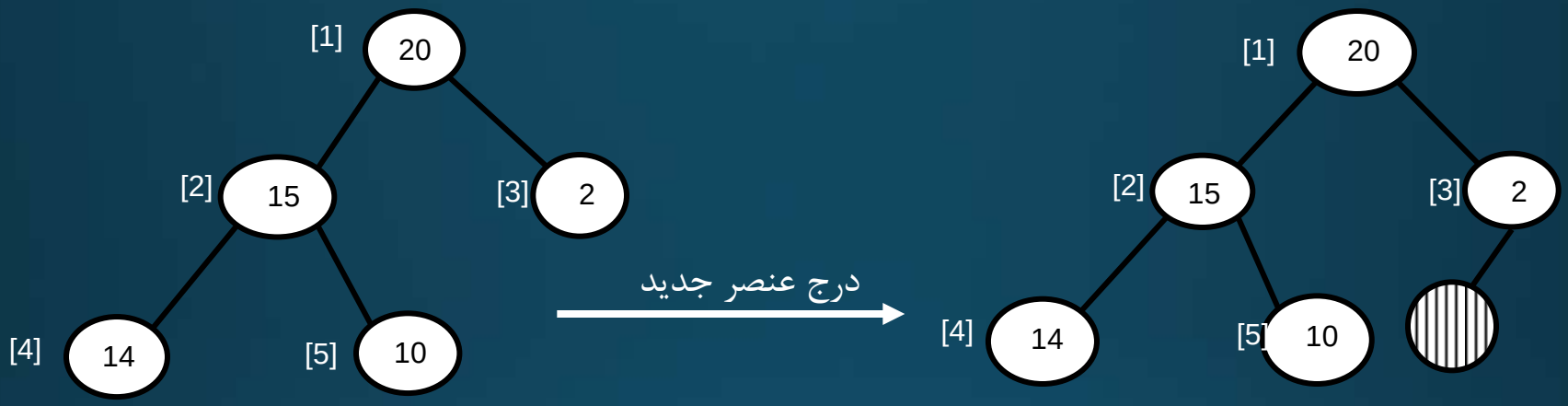
در صف اولویت ها عنصری که دارای بالاترین (یا پایین ترین) اولویت هست ، حذف می شود.

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

نمایش های صف اولویت

نوع نمایش	درج	حذف
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	$O(\log_2 n)$	$O(\log_2 n)$

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.

درج عنصر به یک Max heap

برنامه ۵-۵

```
void Heaparr::insert(int da) {
    size++;
    int* tmp = new int[size];

    for (int i = 0; i < size - 1; i++) {
        tmp[i] = maxHeap[i];
    }

    tmp[size - 1] = da;
    delete[] maxHeap;
    maxHeap = tmp;
}
```

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.

حذف عنصری از Max Heap

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

حذف عنصری از Max Heap

```
void deleteMax (int arr[],int n)
{
   int temp;
   arr[0]=arr[n-1];
   n--;
   for( int i = n/2 -1 ; i >= 0 ; i-- ){
      maxheapify( arr,n,i);
   }
}
```

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.