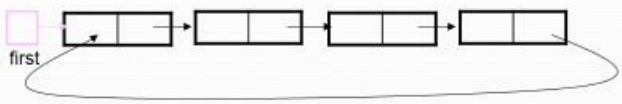
لیست پیوندی حلقوی



در لیست پیوندی حلقوی بهتر است first به انتهای لیست اشاره کند.

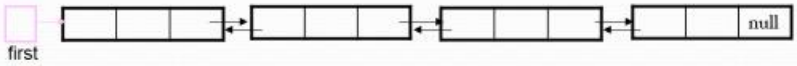
اضافه کردن نود بعد از گره first در لیست

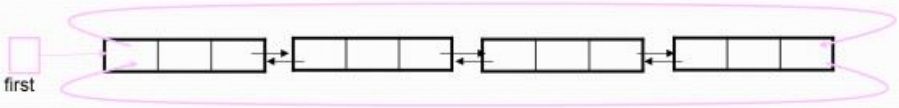
```
node *newptr;
int x;
cin>> x;
newptr = new node;
newptr -> data = x;
if ( first = = NULL)
  newptr -> next = newptr;
else {
   newptr ->next = first -> next;
   first ->next = newptr;
}
first = newptr;
```

لیست دو پیوندی

به منظور دسترسی به گره ماقبل یک گره در لیست تک پیوندی، می بایست از ابتدای لیست پیمایش کرد.

در گره دوپیوندی به سهولت می توان به گره قبلی دسترسی پیدا کرد.





-لیست دو پیوندی

در لیست دو پیوندی سه بخش دارد وجود .

1- بخش data
2- بخش سمت راست که به گره بعدی اشاره می کند.
3- بخش سمت چپ که به گره قبلی اشاره می کند.

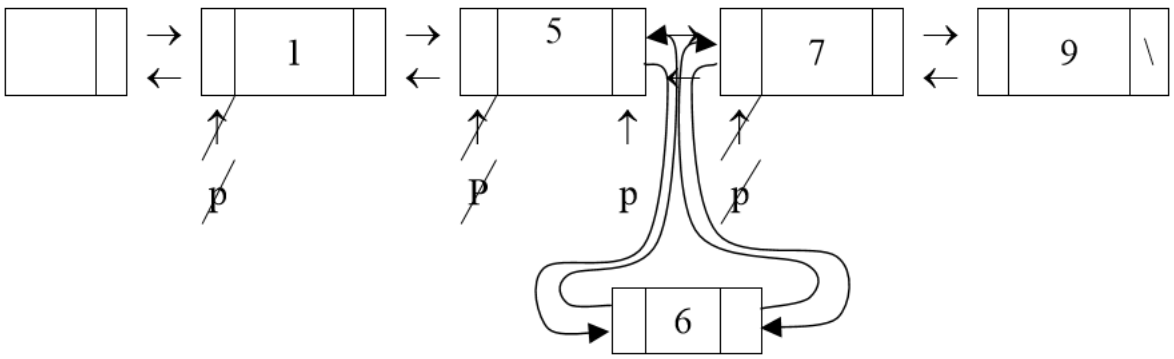
```
struct node {
struct node * left ;
 data ;
struct node * right ; }
node * p , * q ;
```

مراحل اضافه کردن داده به لیست دو پیوندی

1- تشکیل گره جدید با استفاده از (new node)
2- پیدا کردن محل درج گره جدید
3- انتساب مقادیر مورد نظر به بخشهای آدرس چپ و آدرس راست گره های new node و p

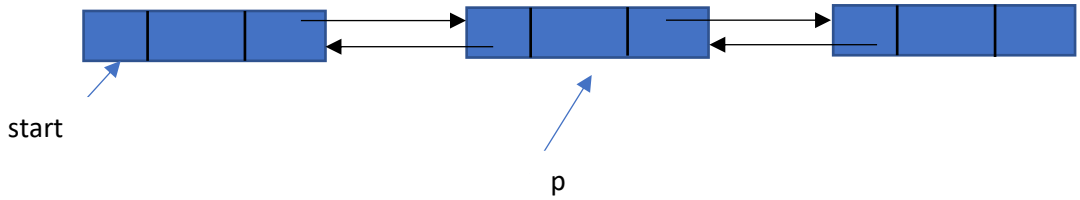
```
void insert (int x , node *start )
{ node * newnode , * p ;
  newnode = new (node);
   newnode → data = x ;
   newnode → right = Null ;
   newnode → left = Null ;
   p = start → right ;
   while ( p → data < x )
       p = p → right ;
    p = p → left ;
   ( p → right ) → left = newnode ;
   newnode → left = p ;
   newnode → right = p → right ;
    p → right = newnode ; }
```

مثال : گره عدد 6 را به لیست زیر اضافه کنید.



حذف از لیست دوپیوندی

```
void delete ( int x , node * start ) {
   node *p ;
   p = start  →right ;
  while ( p && p → data < x )
      p = p → right ;
 if ( p = = Null | | p→  data > x ) return " داده در لیست نبوده است " ;
 ( p → Left ) → right = p → right ;
 ( p → right ) →Left = p→ Left ;
 delete ( p ) ; }
```



درخت (tree):

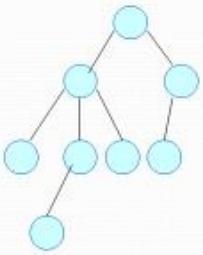
مجموعه محدودی از یک یا چند گره که دارای شرایط زیر می باشند:

۱- دارای گره ای به نام ریشه است.

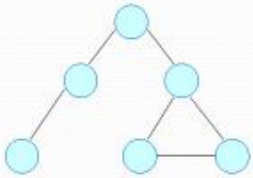
۲- بقیه گره ها به مجموعه هایی مجزا تقسیم شوند که هر یک از این مجموعه ها خود یک درخت هستند.

این شرط که مجموعه ها از هم مجزا باشند از هر گونه اتصال زیر درختها به هم جلوگیری می کند.

در درخت حلقه نداریم.



درخت



درخت نیست

گره برگ : گره ای که هیچ انشعابی از آن خارج نمی شود.

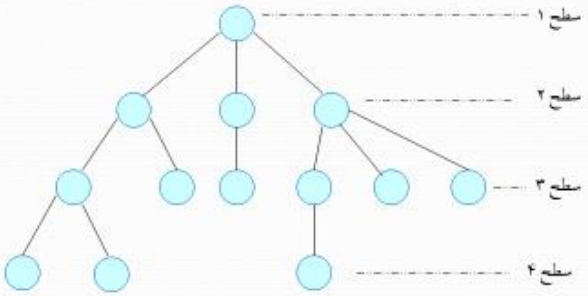
درخت
- درخت دودویی ← درختی که از هر گره آن حداکثر دو پیوند خارج می شود
- درخت عمومی ← درختی که دودویی نباشد، درخت عمومی است.

درجه یک گره: تعداد زیر درختهای یک گره درجه آن نام دارد.

درجه یک درخت: حداکثر درجه گره های آن درخت می باشد.

سطح گره: ریشه در سطح یک قرار می گیرد. سایر گره ها بر اساس تعداد کمانی که از ریشه فاصله دارند شماره سطح می گیرند.

عمق درخت: بیشترین سطح گره های آن درخت

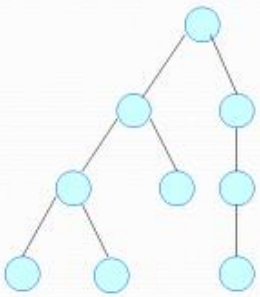


عمق درخت : ۴

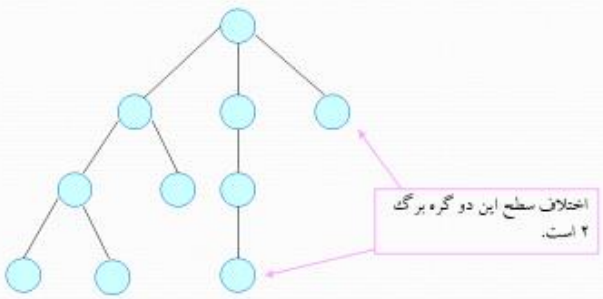
درجه درخت : ۳

درخت متوازن:

درختی است که اختلاف سطح برگهای آن حداکثر یک باشد. اگر اختلاف سطح برگهای آن صفر باشد، کاملا متوازن است.



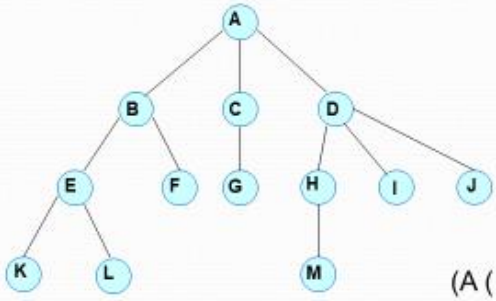
درخت متوازن



درخت متوازن نیست

نمایش درخت:

از فرم پرانتزی استفاده می شود. در این فرم ابتدا اطلاعات ریشه و سپس داخل پرانتز، اطلاعات فرزندان آن گره به ترتیب از چپ به راست ذکر می گردد.



(A (B (E (K,L) ,F), C (G), D(H(M), I, J)))

درخت دودویی

درخت دودویی یا تهی است یا حاوی مجموعه ای از گره ها شامل یک ریشه و دو زیر درخت دودویی است.
زیر درخت چپ و راست

تفاوت درخت عادی و دودویی

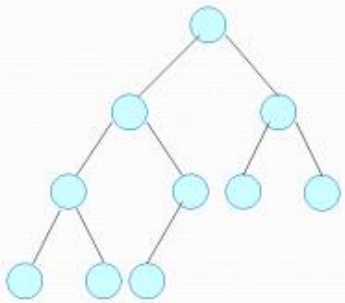
۱- در درخت عادی ترتیب زیر درختان مهم نیست ولی در درخت دودویی ترتیب مهم است.

اگر درخت را عادی تصور کنید این دو درخت یکسانند

۲- درخت عادی تهی نداریم.

درخت کامل

درختی که تمام سطح های آن بجز احتمالا آخرین سطح حداکثر تعداد گره ها را داشته باشد. همچنین گره های آخرین سطح در سمت چپ موجود باشند.

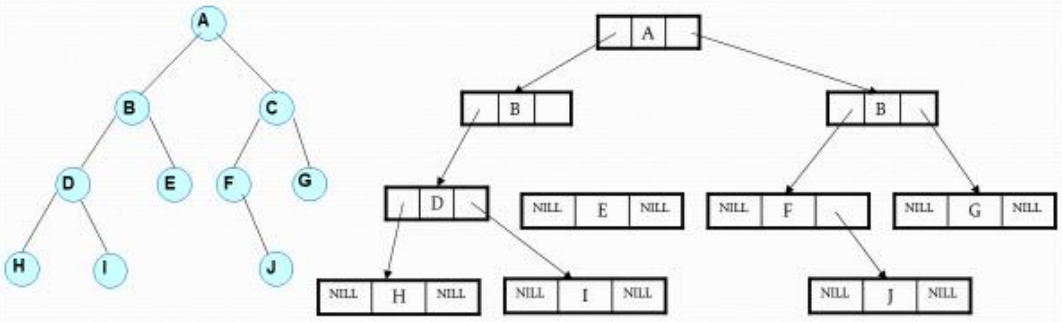


درخت پر

درختی که هم کامل و هم کاملا متوازن باشد.

ذخیره سازی درخت با استفاده از لیست پیوندی

Lchild	data	Rchild



```
class node{
 friend class tree;
 node *left;
 int data;
 node *right;
};
//**********************
class tree{
  node *root;
 public:
  tree();
  ~tree();
  ...
 };
```

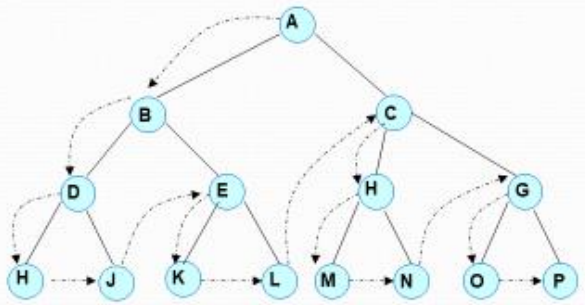
از آنجائیکه توابع کلاس tree باید به متغیرهای خصوصی کلاس node دسترسی داشته باشند، باید به عنوان دوست کلاس معرفی گردد.

پیمایش درخت دودویی

در پیمایش درخت دودویی می خواهیم به هر گره درخت فقط یکبار دستیابی داشته باشیم. در پیمایش درخت دودویی با هر گره و زیر درختانش به طرز مشابهی رفتار می شود.

VLR = preorder = prefix

اول ریشه دوم زیر درخت چپ سوم زیر درخت راست

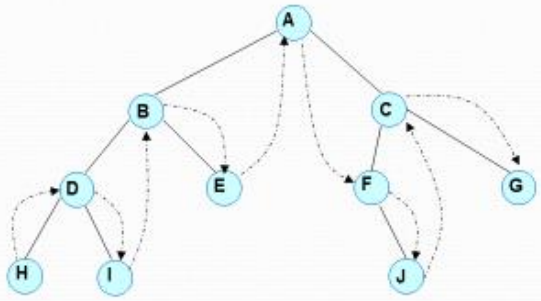


A B D H J E K L C H M N G O P

```
void tree::preorder(tree *s)
 {
  if(s)
   {
    cout<<s->data<<" ";
    preorder(s->left);
    preorder(s->right);
   }
 }
```

LVR = inorder =infix

اول زیر درخت چپ دوم ریشه سوم زیر درخت راست

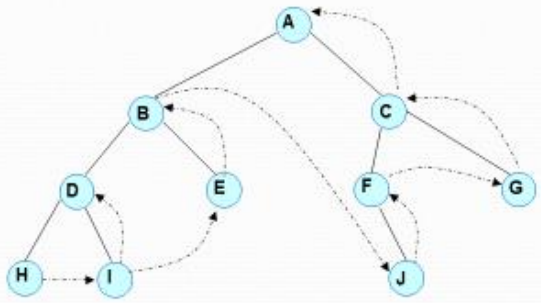


H D I B E A F J C G

```
void tree::inorder(tree *s)
 {
  if(s)
   {
    inorder(s->left);
    cout<<s->data<<" ";
    inorder(s->right);
   }
 }
```

LRV = postorder =postfix

اول زیر درخت چپ دوم زیر درخت راست سوم ریشه



H I D E B J F G C A

```
void  tree::postorder(tree *s)
 {
  if(s)
   {
    postorder(s->left);
    postorder(s->right);
   cout<<s->data<<" ";
   }
 }
```