فایل هفتم درس ساختمان های داده

- فایل آماده شده برای جلسه ادامه مبحث ساختار درخت و صرفا برای کلاس آنلاین در این ترم می باشد.

همواره سلامت و موفق باشید- اسبقی

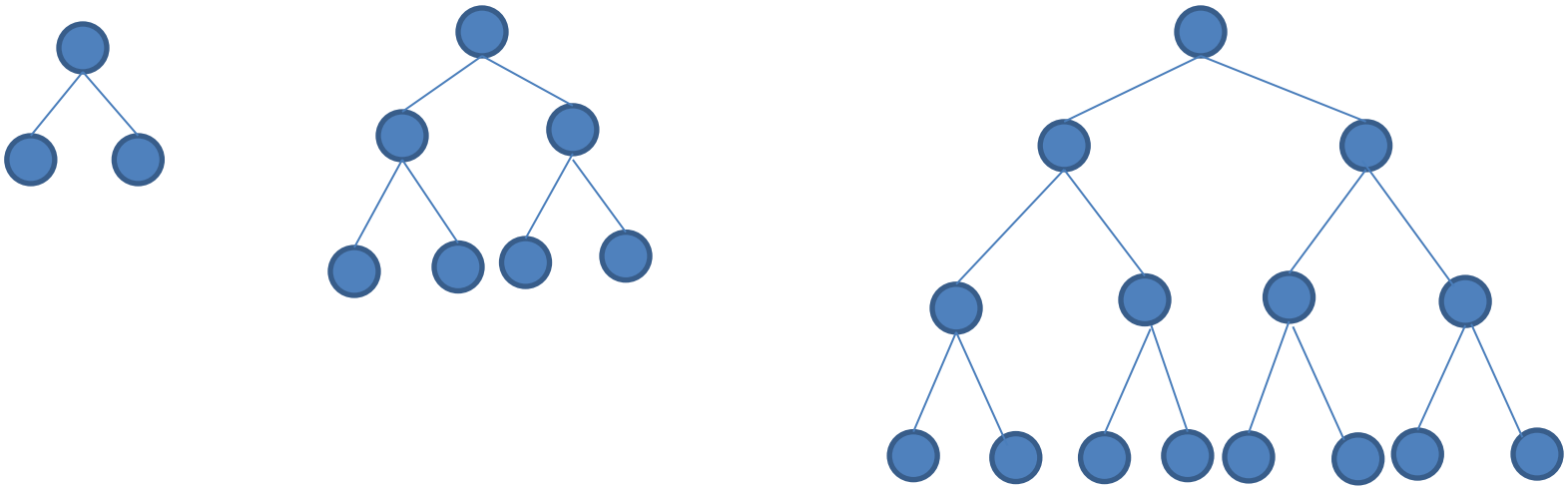
درخت دودویی(binary Tree)

- <u>درخت دودویی پر(Full Binary Tree)</u>

درخت باینری را پر گویند هرگاه:

الف)تمامی گره ها با درجه 2 باشند.

ب)تمامی گره های برگ در یک سطح قرار گرفته باشند.



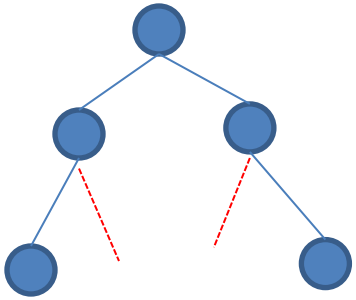
درخت دودویی(binary Tree)

- **درخت دودویی کامل(Complete Binary Tree)**

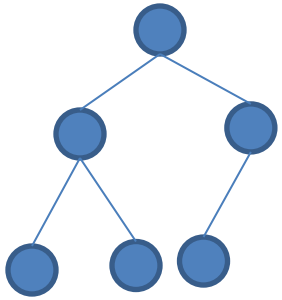
درخت باینری را کامل گویند هر گاه:

الف) تا عمق d – 1 درخت پر باشد.

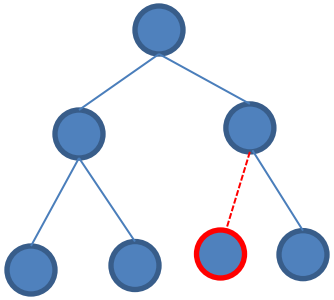
ب)در عمق d، برگها از سمت چپ در درخت به ترتیب اضافه شده باشند.



درخت کامل نیست



درخت کامل است



درخت کامل نیست

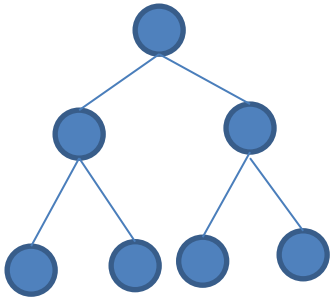
نکته: هر درخت پری، یک درخت کامل است ولی برعکس آن صحیح نیست.

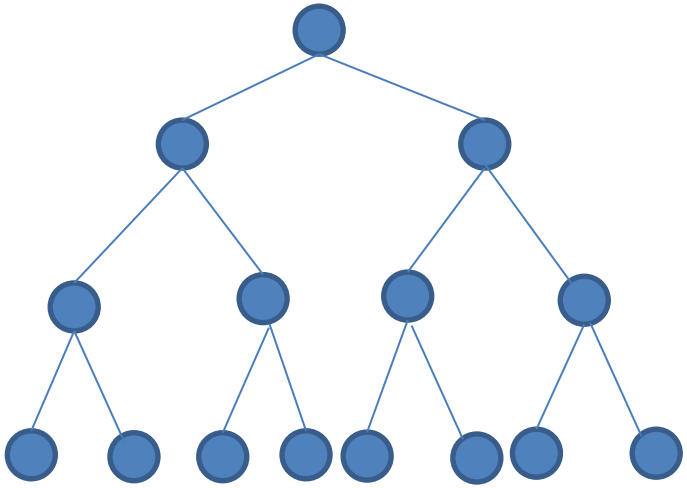
چند قضیه مقدماتی در مورد درخت ها

- حداکثر تعداد گره های یک درخت دودویی با عمق k برابر است با:

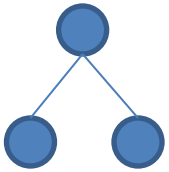
$n = 2^k - 1$

K = 3
$2^k - 1 = 7$
n = 7





K = 4
$2^k - 1 = 15$
n = 15



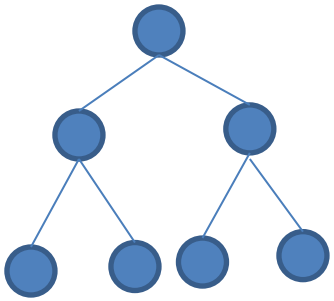
K = 2
$2^k - 1 = 3$
n = 3

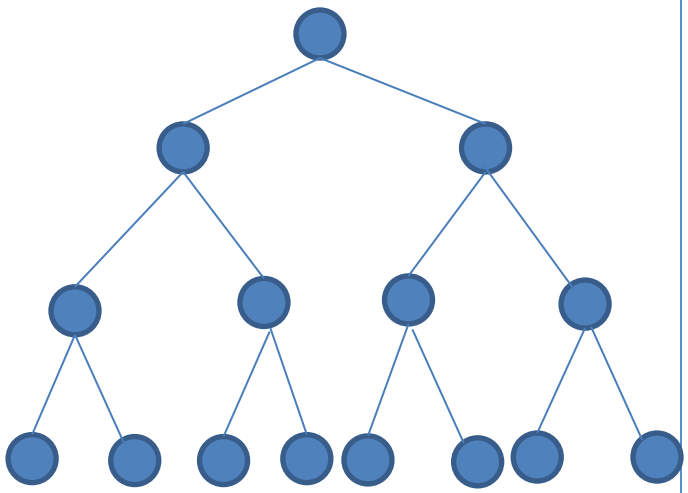
چند قضیه مقدماتی در مورد درخت ها

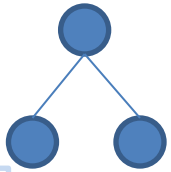
- عمق یک درخت دودویی پر با n گره برابر است با:

$$k = \log(n+1)$$

K = log(7+1)
2^k= 8
K = 3







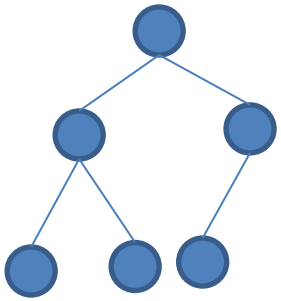
K = log(3+1)
2^k= 4
K = 2

K = log(15+1)
2^k= 16
K = 4

چند قضیه مقدماتی در مورد درخت ها

- عمق یک درخت دودویی کامل با n گره برابر است با:

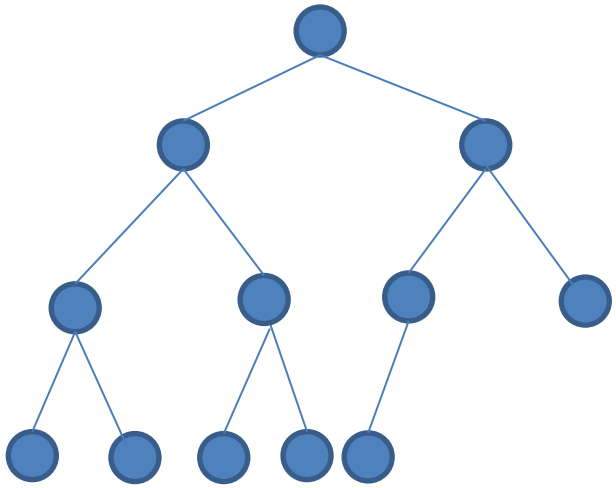
$$k = \lceil \log(n+1) \rceil$$



K = log(6+1)
Log(4)<log(7)<log(8)
2 < ? < 3

K = ⌈log (6+1)⌉

K = 3



K = log(12+1)
Log(8)<log(13)<log(16)
3 < ? < 4

چند قضیه مقدماتی در مورد درخت ها

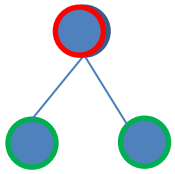
- در درخت های دودویی همواره داریم:

$n_0 = n_2 + 1$

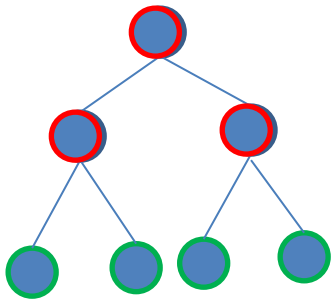
n_0: تعداد گره های با درجه 0(برگ)

n_2: تعداد گره های با درجه 2

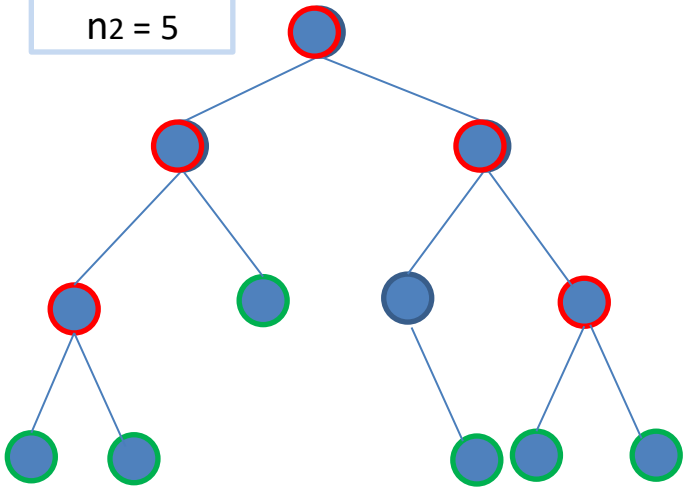
$n_0 = 2$
$n_2 = 1$



$n_0 = 4$
$n_2 = 3$



$n_0 = 6$
$n_2 = 5$



پیاده سازی درخت

- <u>پیاده سازی درخت:</u>

در پیاده سازی درخت نیز، بایستی ساختار سلسله مراتبی آن حفظ گردد.

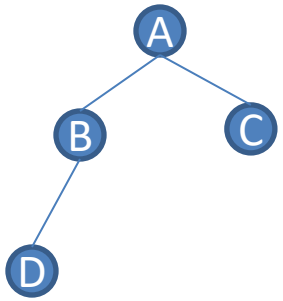
- پیاده سازی درخت با استفاده از آرایه

مناسب برای ذخیره درخت های کامل

- پیاده سازی درخت با استفاده از لیست پیوندی

پیاده سازی درخت با استفاده از آرایه

درخت زیر را در نظر بگیرید:



- <u>قواعد پیاده سازی درخت با آرایه:</u>

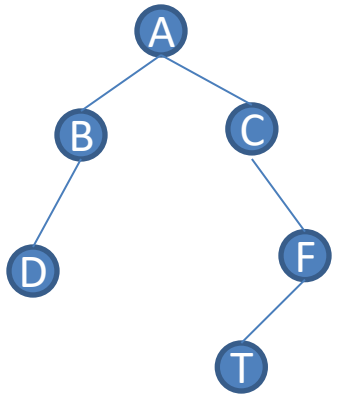
- گره ریشه در اولین محل آرایه، در این مثال در اندیس 1 تعریف می شود.
- فرزند چپ گره موجود دراندیس i در اندیس 2* i قرار می گیرد.
- فرزند راست گره موجود در اندیس i در اندیس 1+ 2* i قرار می گیرد.
- گره پدر گره موجود در اندیس i در اندیس ⌊ i *2 ⌋ قرار می گیرد.

- i = 1 ⟹ i*2 = 2 i*2+1 = 3
- i = 2 ⟹ i*2 = 4 i*2+1 = 5
- i = 3 ⟹ ⌊ i/2 ⌋ = 1
- i = 4 ⟹ ⌊ i/2 ⌋ = 2

1	2	3	4
A	B	C	D

پیاده سازی درخت با استفاده از آرایه

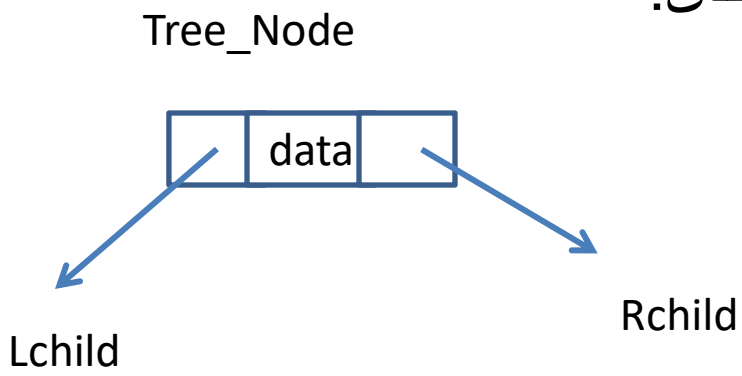
- i = 1 ⟹ i*2 = 2 i*2+1 = 3
- i = 2 ⟹ i*2 = 4 i*2+1 = 5
- i = 3 ⟹ i*2 = 6 i*2+1 = 7
- i = 4 ⟹ i*2 = 8 i*2+1 = 9
- .....
- i = 7 ⟹ i*2 = 14 i*2+1 = 15
- i = 7 ⟹ ⌊i/2⌋ = 3
- i = 14 ⟹ ⌊i/2⌋ = 7



1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	C	D	-	-	F	-	-	-	-	-	-	T

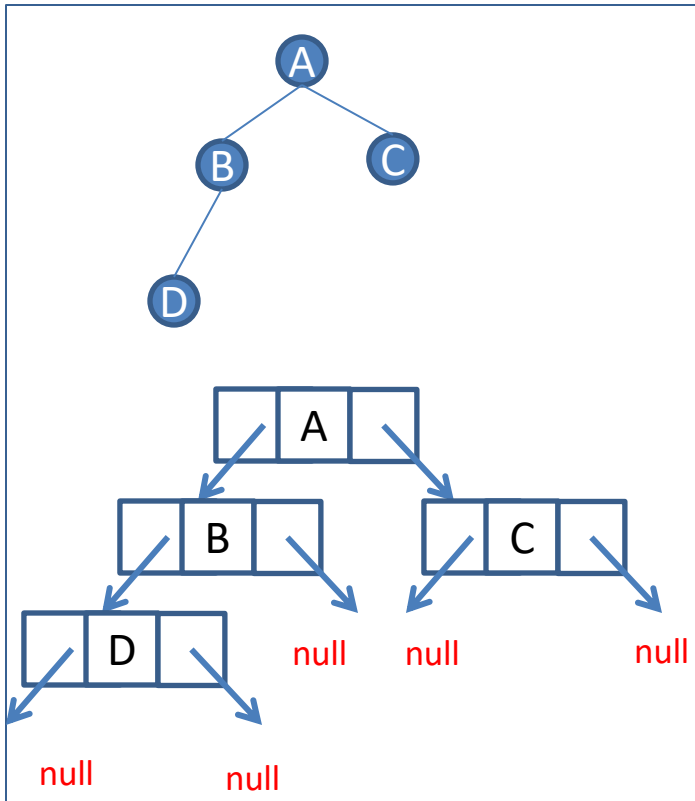
پیاده سازی درخت با استفاده از لیست پیوندی

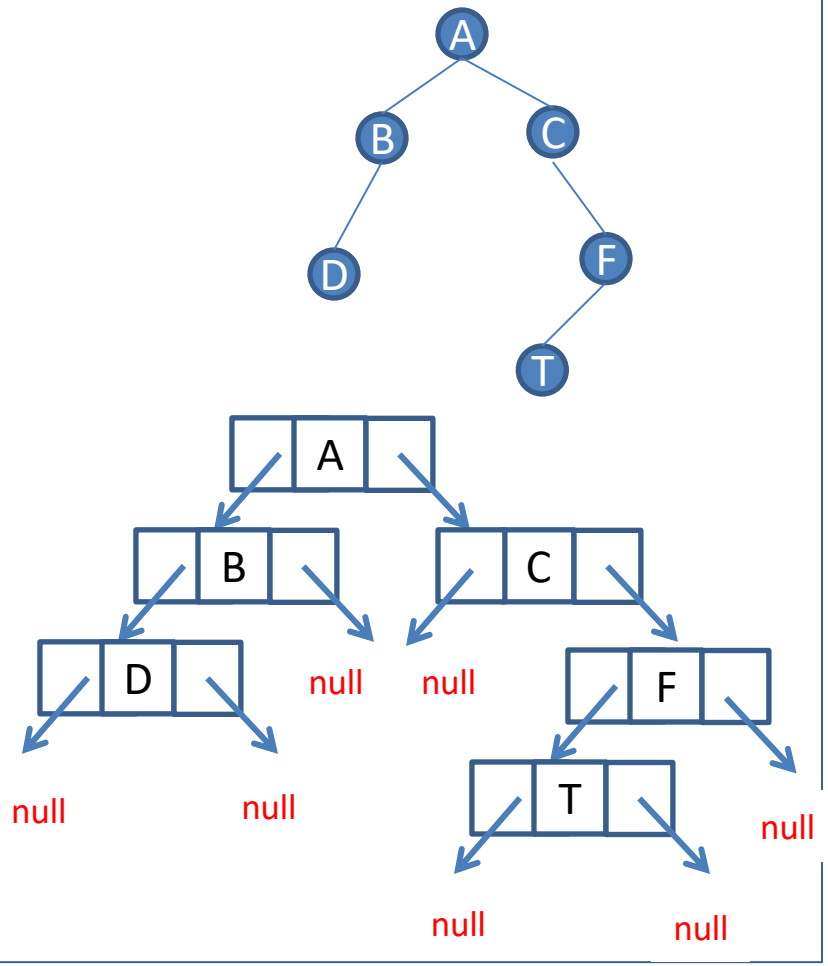
- در پیاده سازی درخت دودویی با لیست پیوندی، نیاز به لیست های دو پیوندی برای اشاره به فرزندان چپ و راست وجود دارد.
- گره درخت در پیاده سازی با لیست پیوندی:




```
struct   Tree_Node{
Char     data;
Tree_Node      *Rchild;
Tree_Node      *Lchild;};
```

پیاده سازی درخت با استفاده از لیست پیوندی





پیمایش درخت های دودویی

- منظور از پیمایش هر ساختار داده ای، ملاقات تمام عناصر آن ساختار داده ای است، فقط و فقط یکبار(نه بیشتر و نه کمتر)
- پیمایش عمقی
  - پیمایش پیشوندی(preorder)(DLR/DRL)
  - پیمایش میانوندی(inorder)(LDR/RDL)
  - پیمایش پسوندی(postorder)(LRD/RLD)
- پیمایش سطحی(levelorder)

پیمایش های درخت های دودویی

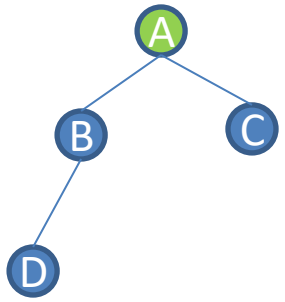
- برای درخت های زیر پیمایش های درخت را استخراج نمایید.

Preorder: A B D C

Inorder:D B A C

Postorder:D B C A

Levelorder:A B C D

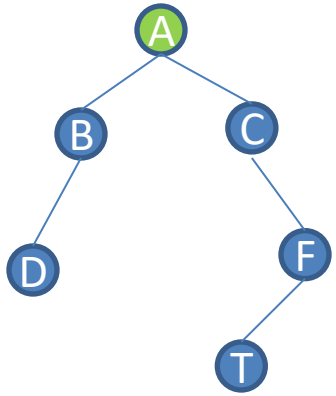


Preorder: A B D C F T

Inorder:D B A C T F

Postorder:D B T F C A

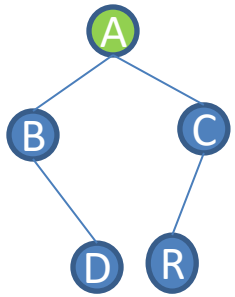
Levelorder:A B C D F T



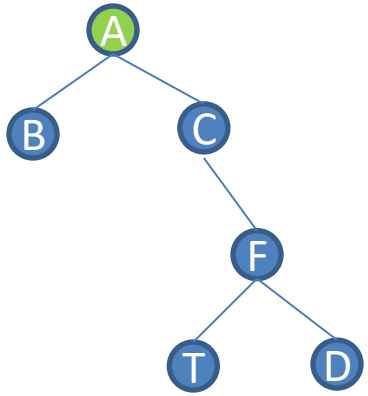
پیمایش های درخت های دودویی

- برای درخت های زیر پیمایش های درخت را استخراج نمایید.

Preorder: A B D C R
Inorder: B D A R C
Postorder:D B R C A
Levelorder:A B C D R



Preorder: A B C F T D
Inorder: B A C T F D
Postorder:B T D F C A
Levelorder:A B C F T D



پیمایش های درخت های دودویی

- پیاده سازی بازگشتی پیمایش های عمقی

```
Preorder(Tree_Node  *root)
{
if (root!=null)
{    cout<<root->data;
     Preorder(root->Lchild);
     Preorder(root->Rchild);
}}
```

```
Inorder(Tree_Node  *root)
{
if (root!=null)
{    Inorder(root->Lchild);
     cout<<root->data;
     Inorder(root->Rchild);
}}
```

```
Postorder(Tree_Node  *root)
{
if (root!=null)
{    Postorder(root->Lchild);
     Postorder(root->Rchild);
     cout<<root->data;
}}
```

پیمایش های درخت های دودویی

- پیاده سازی غیر بازگشتی پیمایش های عمقی(پیمایش میانوند)- در اینحالت بایستی از ساختار پشته، برای پیاده سازی پیمایش های عمقی استفاده نمود.

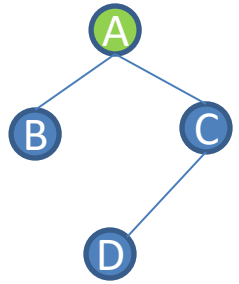
```
Inorder(Tree_Node  *root){
  if (root!=null)
   {  stack    s;
      Tree_Node   *temp = root;
     while(true)
      {
         while( temp!=null)
           { push(s, temp);
            temp = temp->Lchild;}
         if (s  isn't  empty)
           { temp = pop(s);
             cout<<temp->data;
             temp = temp->Rchild;
             }
         else
              break;
      }
}}
```

بعنوان تمرین، الگوریتم غیربازگشتی پیمایش های پیشوندی و پسوندی را بنویسید.

پیمایش های درخت های دودویی

```
Preorder(Tree_Node  *root)
{
if (root!=null)
{    cout<<root->data;
     Preorder(root->Lchild);
     Preorder(root->Rchild);
}}
```

preorder: A B C D




```
Preorder(Tree_Node  *A)
{
if (A!=null)
{    cout<<A->data;
     Preorder(B);
     Preorder(C);}}
```

```
Preorder(Tree_Node  *B)
{
if (B!=null)
{    cout<<B->data;
     Preorder(null);
     Preorder(null);}}
```

```
Preorder(Tree_Node  *C)
{
if (C!=null)
{    cout<<C->data;
     Preorder(D);
     Preorder(null);}}
```

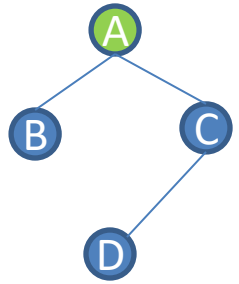
```
Preorder(Tree_Node  *D)
{
if (D!=null)
{    cout<<D->data;
     Preorder(null);
     Preorder(null);}}
```

1 2 3 4 5 6 7 8

پیمایش های درخت های دودویی

```
Postorder(Tree_Node  *root)
{
if (root!=null)
{    Postorder(root->Lchild);
     Postorder(root->Rchild);
     cout<<root->data;
}}
```

postorder: B D C A




```
Postorder(Tree_Node  *B)
{
if (B!=null)
{    Postorder(null);
     Postorder(null);
     cout<<B->data;}}}
```

```
Postorder(Tree_Node  *A)
{
if (A!=null)
{    Postorder(B);
     Postorder(C);
     cout<<A->data;}}}
```

```
Postorder(Tree_Node  *C)
{
if (C!=null)
{    Postorder(D);
     Postorder(null);
     cout<<C->data;}}}
```

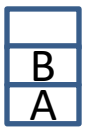
```
Postorder(Tree_Node  *D)
{
if (D!=null)
{    Postorder(null);
     Postorder(null);
     cout<<D->data;}}}
```

1 2 3 4 5 6 7 8

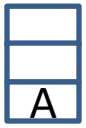
پیمایش های درخت های دودویی

```
Inorder(Tree_Node  *root){
  if (root!=null)
   {  stack    s;
     Tree_Node   *Temp = root;
    while(true)
     {
        while( Temp !=null)
          { push(s, Temp);
           Temp = Temp->Lchild;}
        if (s  isn't  empty)
           { Temp = pop(s);
             cout<<Temp->data;
             Temp = Temp->Rchild;
             }
        else
            break;
     }
}}
```

Temp = A
Temp = B
Temp = null



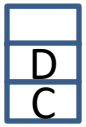
Temp = B
Temp = null



Temp = A



Temp = C
Temp = D
Temp = null

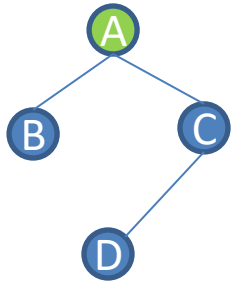


Temp = D
Temp = null



Temp = C
Temp = null





inorder: B A D C

پیمایش سطحی(level order)

- پیاده سازی پیمایش سطحی با استفاده از ساختار صف:

```
levelorder(Tree_Node  *root){
  if (root!=null)
   {  Queue    q;
      Tree_Node   *Temp = root;
     while(Temp!=null)
      {
         cout<<Temp->data;
         if (Temp->Lchild!= null)
            Add_q(q,Temp->Lchild);
          if (Temp->Rchild!= null)
            Add_q(q,Temp->Rchild);
              Temp = Remove_q();
      }
}
```

Temp = A



Temp = B



Temp = C



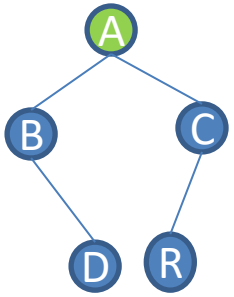
Temp = D



Temp = R



Temp = null



levelorder: ABCDR