به نام خدا

ساختمان داده ها

جلسه سیزدهم

دانشگاه صنعتی همدان

گروه مهندسی کامپیوتر

نیم سال دوم 98-1397

درختها

Trees

مطالب

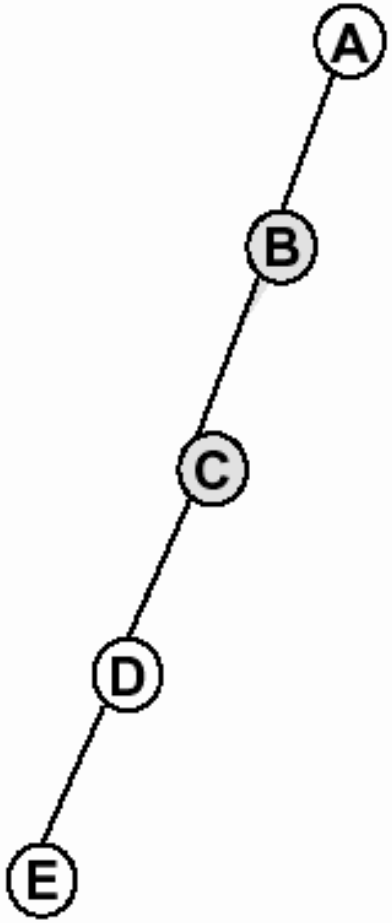
- مقدمه و تعاریف
- نمایش درختان
- درختان دودویی
  - ADT درختان دودویی
  - خواص درختان دودیی
  - نمایش درختان دودویی
  - نمایش درختان با لیست ها
  - پیمایش درختان دودویی
  - اعمال روی درختان دودیی
  - درختان دودویی نخ کشی شده
- هرمها (Heaps)
- درختان جستجو
- درختان انتخاب
- جنگلها

درختان دودیی

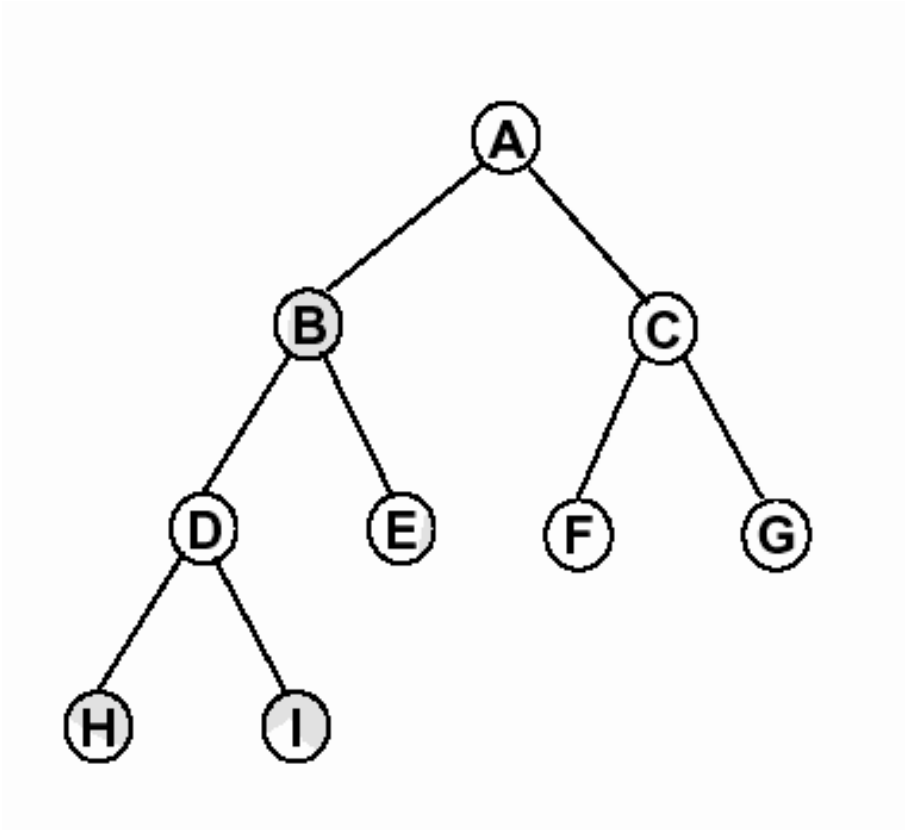
- درخت دودویی:
  - در این درخت درجه هر نود حداکثر دو است.
  - ترتیب نودها اهمیت دارد.
  - ممکن است دارای صفر نود باشد.

- تعریف رسمی:
  - یک درخت دودویی مجموعه محدودی از نودها است که یا خالی است یا شامل ریشه و دو زیر درخت متمایز دودویی است که به آنها زیر درختان سمت راست و سمت چپ گفته می شود.

درختان دودیی



خطي - نامتعادل



کامل – متعادل تر

درختان دودیی

- خواص جالب درخت دودویی:

  - حداکثر تعداد گره ها در سطح i برابر 2^{i-1} است.
  - حداکثر تعداد گره ها در درخت باینری با ارتفاع k ، برابر $2^k - 1$ است.
  - برای هر درخت غیر تهی مانند T اگر n_0 تعداد گره های درجه صفر و n_2 تعداد کره های درجه ۲ باشد انگاه $n_0=n_2+1$
  - یک درخت باینری پر با عمق K ، دارای $2^k - 1$ نود هست

درختان دودیی

- حداکثر تعداد نودها در سطح i برابر 2^{i-1} است.

- اثبات با استفاده از استقراء:
  - حالت پایه:
    - Level 1 = $2^{1-1} = 2^0 = 1$
  - برای یک $i > 1$
    - اگر سطح i-1 دارای 2^{i-2} نود باشد.
  - برای i :
    - هرنود در سطح i-1 ، حداکثر دو فرزند در سطح i دارد. پس تعداد نودهای سطح i حداکثر برابر $2^{i-1} * 2 = 2^i$ است.

درختان دودیی

- حداکثر تعداد نودها در درخت باینری با ارتفاع k ، برابر $2^k - 1$ است.

  می توان ماکسیمم تعداد نودهای هر سطح را با هم جمع زد.

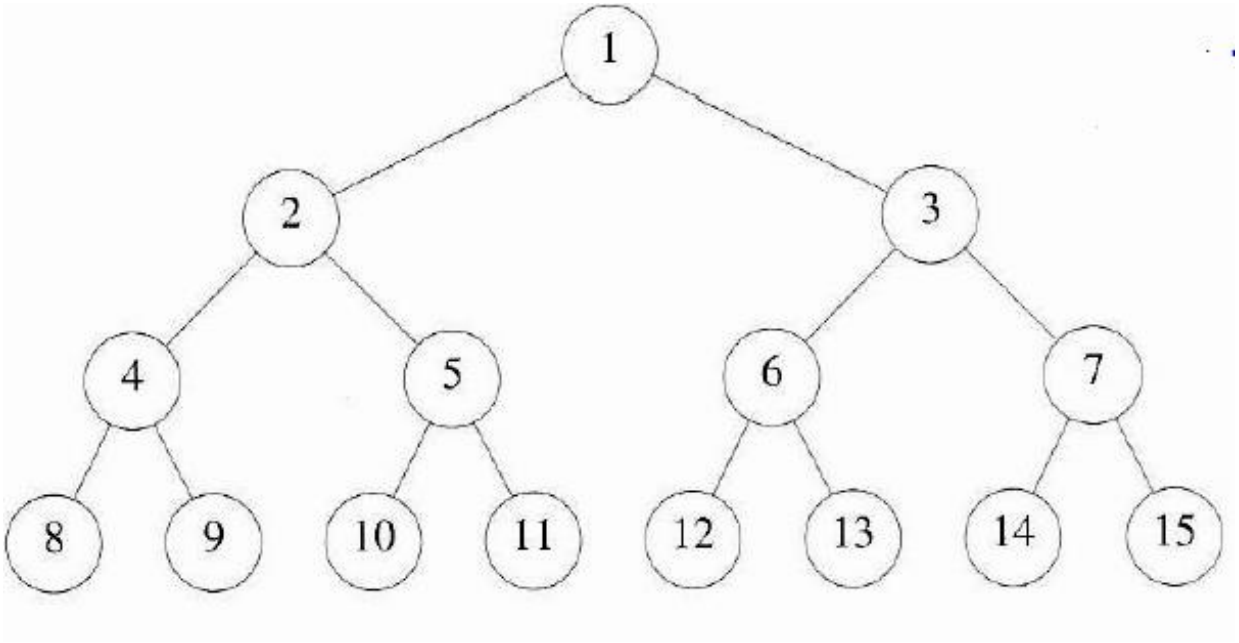
  باید حاصل جمع $1+2+4+\ldots+2^{k-1}$ را بدست آوریم.

  این جمع برابر است با: $2^k - 1$

- برای هر درخت غیر تهی مانند T اگر n_0 تعداد گره های درجه صفر و n_2 تعداد کره های درجه ۲ باشد انگاه $n_0=n_2+1$

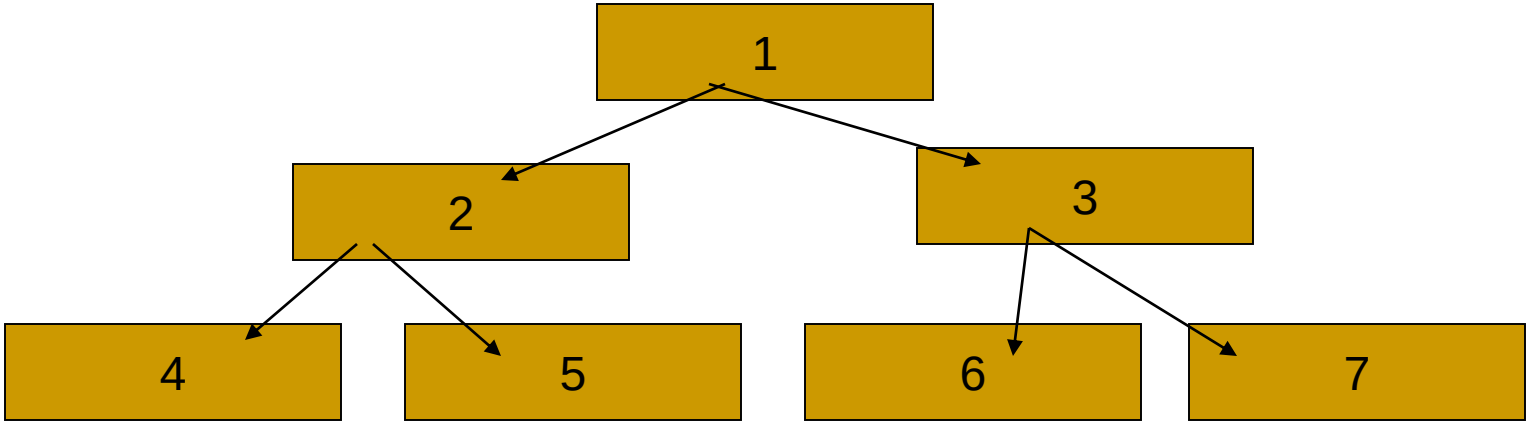
- $n=n_0+n_1+n_2$
- $n=B+1$
- $B=n_1+2n_2$
- $n=B+1=n_1+2n_2+1$
- $n_0+n_1+n_2=n_1+2n_2+1$

- یک درخت باینری پر با عمق K ، دارای $2^k - 1$ نود هست
- یک درخت دودیی به عمق k کامل است اگر شماره گذاری گره های ان مطابق با شماره گذاری یک درخت دودویی پر به عمق k باشد.
- ارتفاع یک درخت دودویی کامل با n نود برابر است با: $(\log_2(n)+1)$



نمایش درختهای دودویی

- پیاده سازی آرایه ای



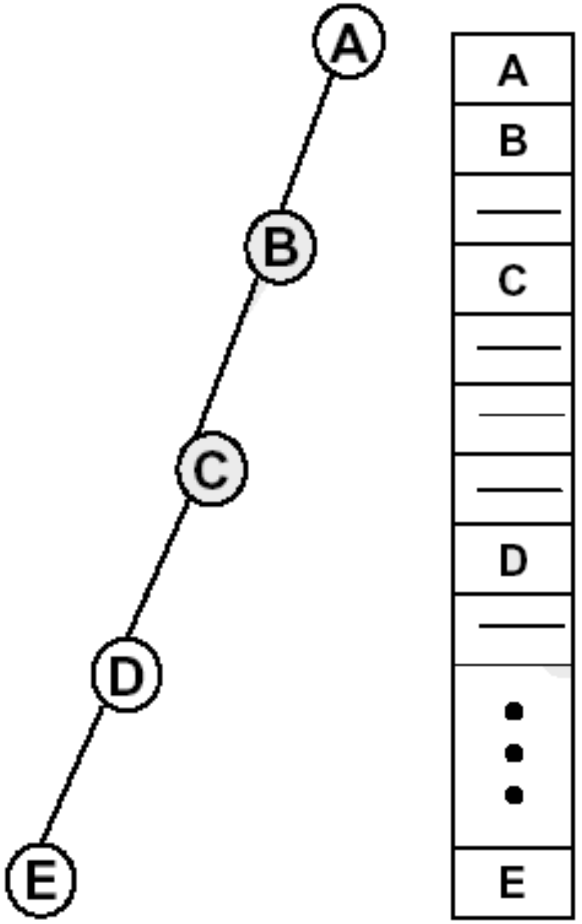
- هر نود با یک عنصر از آرایه متناظر می گردد

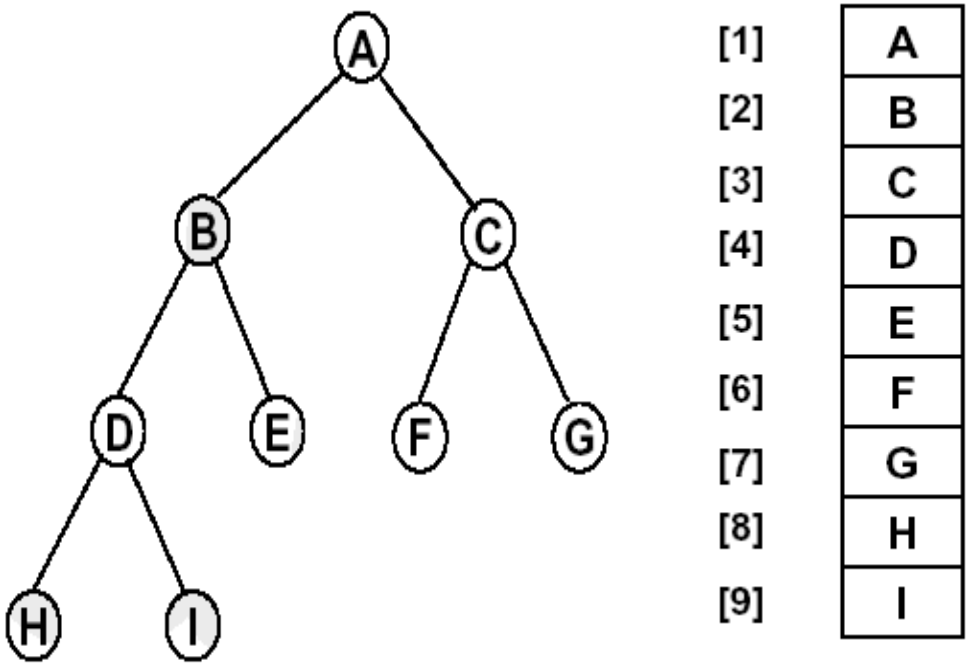
نمایش ارایه ها با ارایه

- پیاده سازی آرایه ای:
  - parent(i) در محل i/2 قرار دارد (برای i!=1).
  - فرزند سمت چپ i در محل 2i و فرزند سمت راست در محل 2i+1 قرار دارد.
  - اگر 2i>n باشد، i فرزند سمت چپ ندارد.
  - اگر 2i+1>n باشد، i فرزند سمت راست ندارد.

Array Index: 0 1 2 3 4 5 6 7

خالي	Data1	Data2	Data3	Data4	Data5	Data6	Data7


A
B
C
D
E
A
B
C
D
E


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

نمایش درختها با ارایه

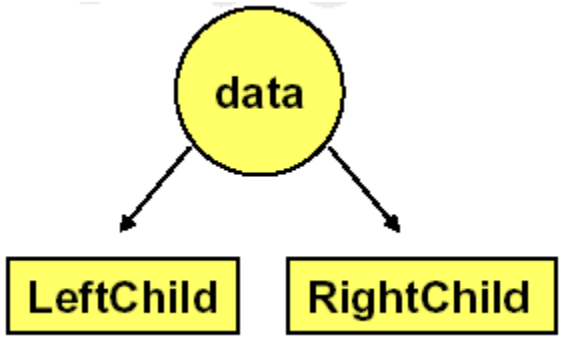
- نمایش آرایه ای بهترین راه حل نیست.
  - اندازه حافظه ثابت است:
    - براحتی قابل گسترش نیست.
    - اگر درخت بالانس نباشد، حافظه هدر می رود.

- راه بهتر: استفاده از ایده لینک پیوندی.
  - می شود از راه حل استفاده از همزاد دوری کرد، چون تعداد فرزندان ثابت است و فقط به دو اشاره گر نیازمندیم.

ساختار کلاس برای درختان دودویی به صورت پیوندی

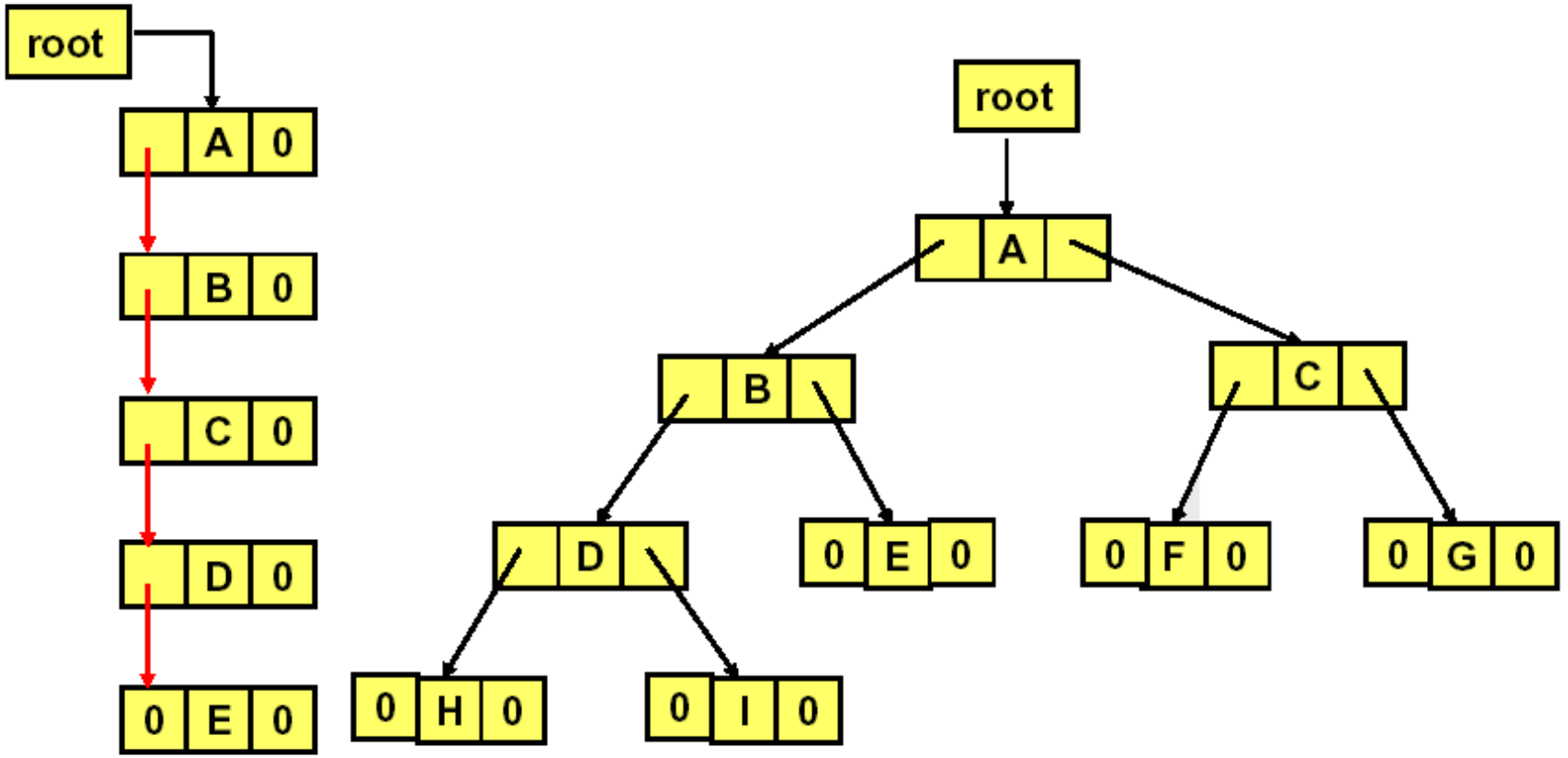
```
class Tree;
class TreeNode
{
    friend class Tree;
    private:
        char data;
        TreeNode* leftChild;
        TreeNode* rightChild;
};

class Tree
{
    public:
        // public member methods
    private:
        TreeNode* root;
};
```



مشکل این روش ؟

نمایش درختهای دودویی با لیست پیوندی



نمایش درختهای دودویی با لیست پیوندی

- تنها مشکل نمایش پیوندی این است که به پدر دسترسی نداریم.

- اشکالی ندارد:
  - بسیاری از الگوریتمها نیازی به دانستن پدر ندارند.
  - اگر هم مهم باشد، وقتی که درخت را پیمایش می کنیم، یک اشاره گر به پدر را نگهداری می کنیم.
  - اگر خیلی مهم باشد، می توان به تعریف نود اشاره گر پدر را اضافه نمود.

پیمایش درختها

- تمام عملگرهای که روی درخت باینری کار می کنند نیازمند حرکت روی درخت هستند:
  - دیدن نودها
  - اضافه کردن نود
  - حذف یک نود
  - محاسبه ارتفاع بصورت غیر بازگشتی
- لذا داشتن یک مکانیسم برای پیمایش درخت بسیار مفید خواهد بود.

پیمایش درختها

- پیمایش درخت
  - هر نود فقط یک بار دیده شود.
  - تمام نودها دیده شوند.
  - یک یا چند عملگر روی درخت اجرا شود:
    - چاپ داده
    - جمع با حاصل جمع
    - چک کردن حداکثر ارتفاع
  - هر پیمایش یک ترتیب خطی از همه نودها را تولید خواهد کرد.

پیمایش درختها

- فرض کنید برای پیمایش از حروف زیر استفاده کنیم:
  - L یعنی حرکت به فرزند سمت چپ
  - R یعنی حرکت به فرزند سمت راست
  - V یعنی دیدن نود (یا انجام عمل مورد نظر)

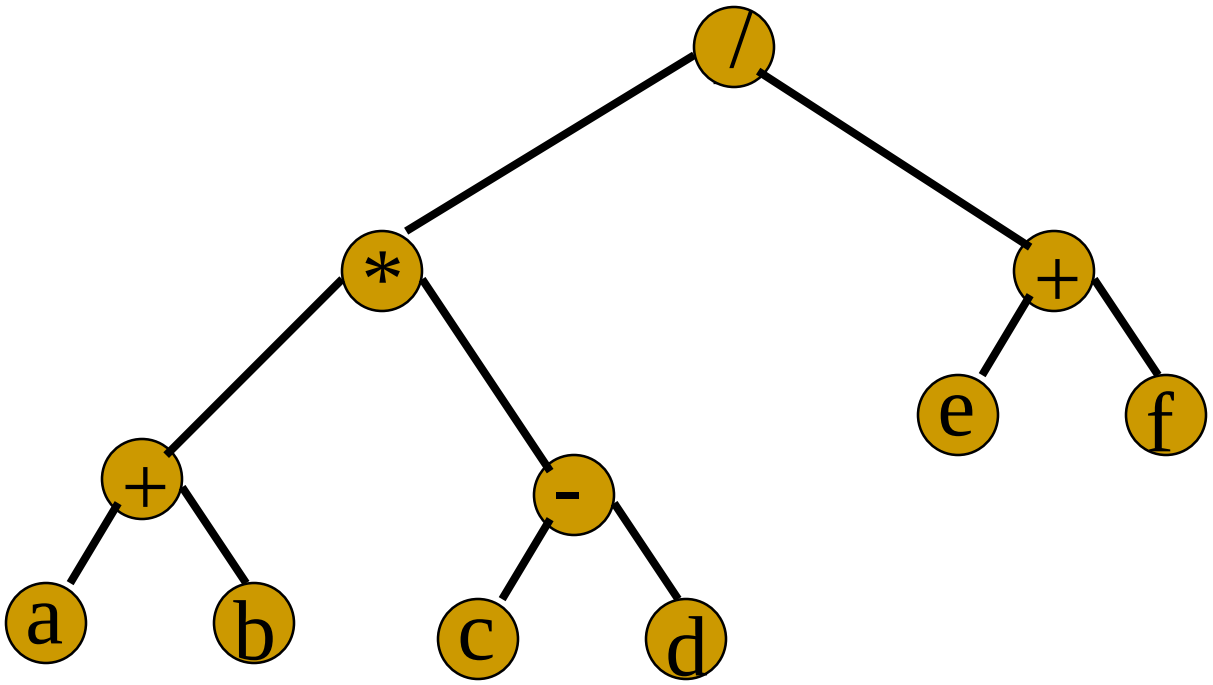
- شش حالت امکان پذیر است:
  - LVR, LRV, VLR, VRL, RVL, RLV

- ما فقط حالتهایی را که L قبل از R آمده است را مورد توجه قرار می دهیم:

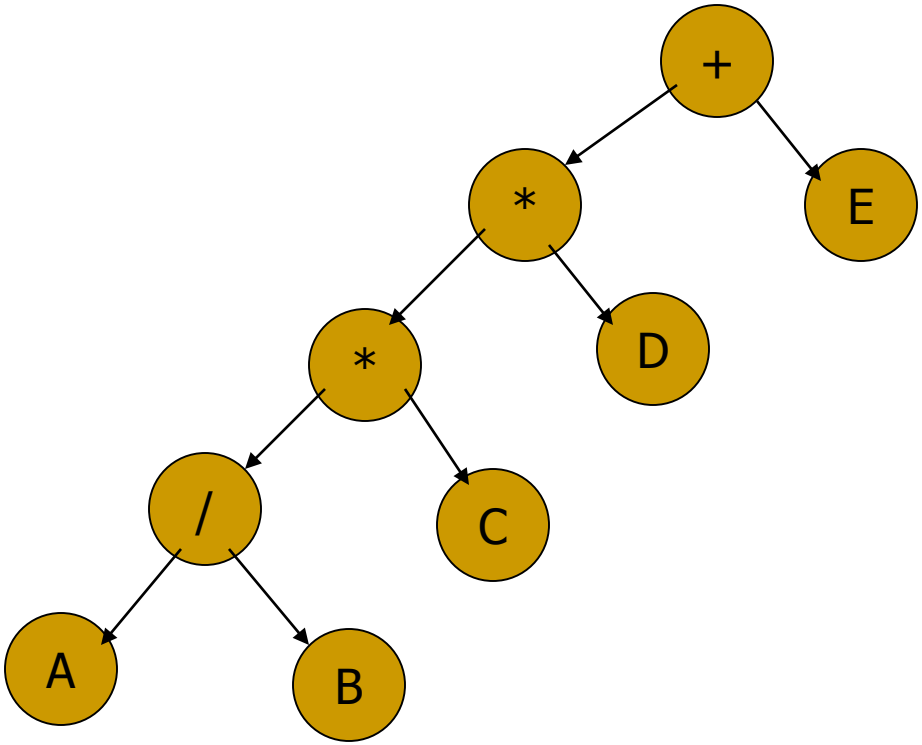
LVR	LRV	VLR
Inorder	Postorder	Preorder
میان ترتیب	پس ترتیب	پیش ترتیب

نمايش يك عبارت با درخت دودويي

- (a + b) * (c – d) / (e + f)



پیمایش درختها



Inorder: LVR

A / B * C * D + E

عبارت میانوندی
دیدن سمت چپ پیش از پدر

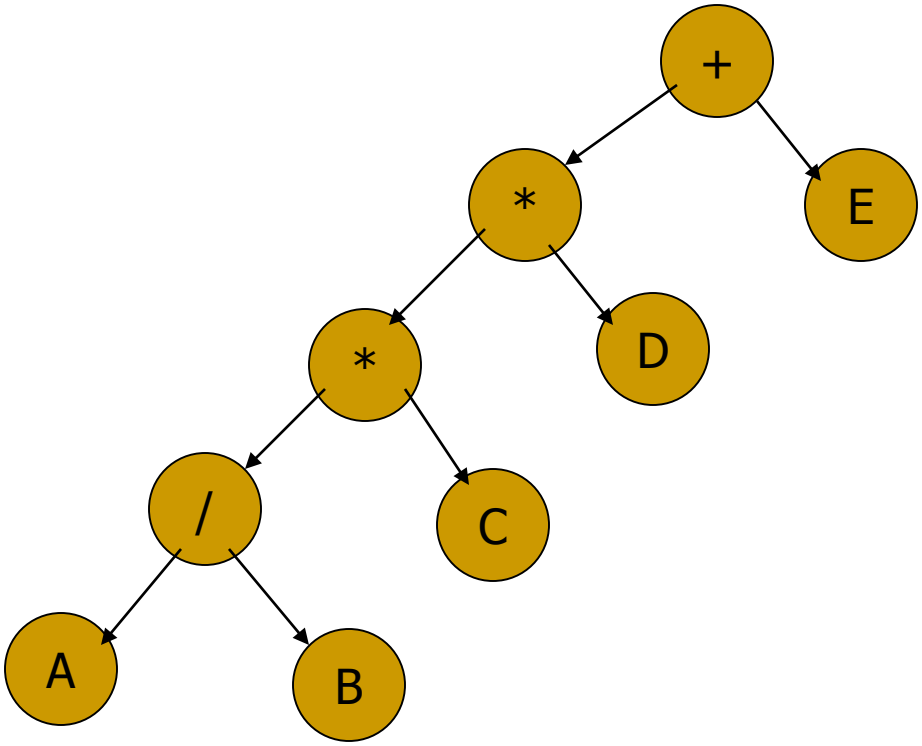
پیمایش درختها

- پیاده سازی Inorder:

```
void  Tree::inorder()
{
    inorder(root);
}

Void Tree::inorder(TreeNode* node)
{
    if (node)
    {
        inorder(node->leftChild);
        cout << node->data;
        inorder(node->rightChild);
    }
}
```

Binary Tree Traversal



Postorder: LRV

A B / C * D * E +

عبارت پسوندی

دیدن سمت چپ و راست پیش از پدر

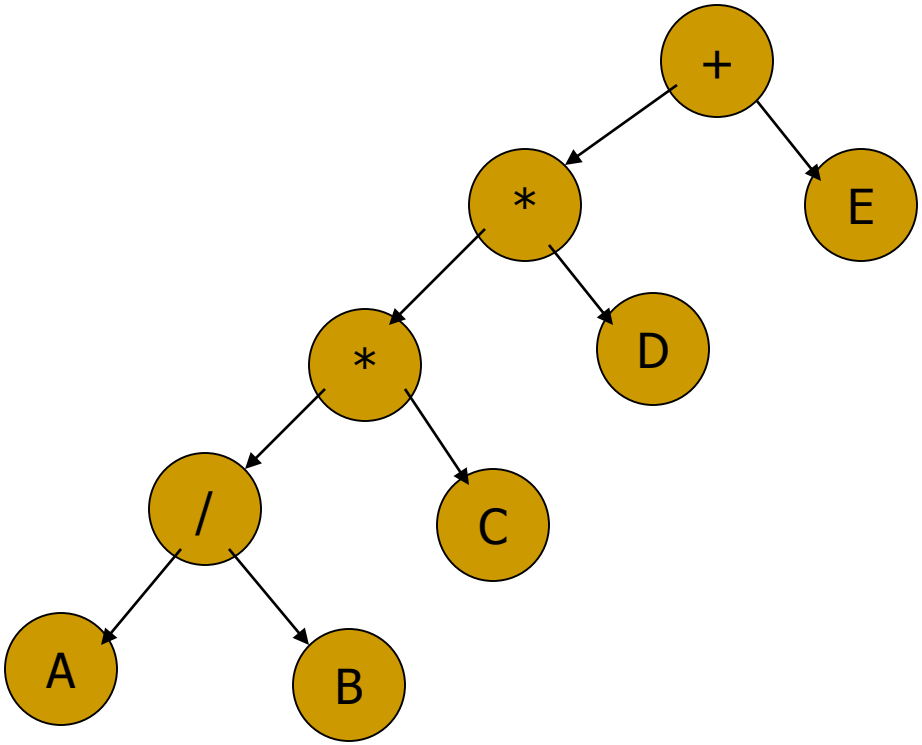
پیمایش درختها

- پیاده سازی postorder:

```
void Tree::postorder()
{
    postorder(root);
}

void Tree::postorder(TreeNode* node)
{
    if (node)
    {
        postorder(node->leftChild);
        postorder(node->rightChild);
        cout << node->data;
    }
}
```

پیمایش درختها



Preorder: VLR

+ * * / A B C D E

عبارت پیشوندی
دیدن پدر پیش از فرزندان

پیمایش درختها

- پیاده سازی Preorder :

```
void Tree::preorder()
{
    preorder(root);
}

Void Tree::preorder(TreeNode* node)
{
    if (node)
    {
        cout << node->data;
        preorder(node->leftChild);
        preorder(node->rightChild);
    }
}
```

پیمایش غیر بازگشتی

```
1 void Tree ::NonrecInorder ()
2 // nonrecursive inorder traversal using a stack
3 {
4   Stack <TreeNode *> s; // declare and initialize stack
5   TreeNode *CurrentNode = root;
6   while(1) {
7       while (CurrentNode) { // move down LeftChild fields
8           s.Add (CurrentNode); // add to stack
9           CurrentNode = CurrentNode →LeftChild;
10      }
11      if (! s.IsEmpty ()) { // stack is not empty
12          CurrentNode = *s.Delete (CurrentNode); // delete from stack
13          cout << CurrentNode →data << endl;
14          CurrentNode = Curr tNode →RightChild;
15      }
16      else break;
17  }
18 }
```

پیچیدگی زمانی ؟