به نام خدا

ساختمان داده ها

جلسه چهاردهم

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

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)

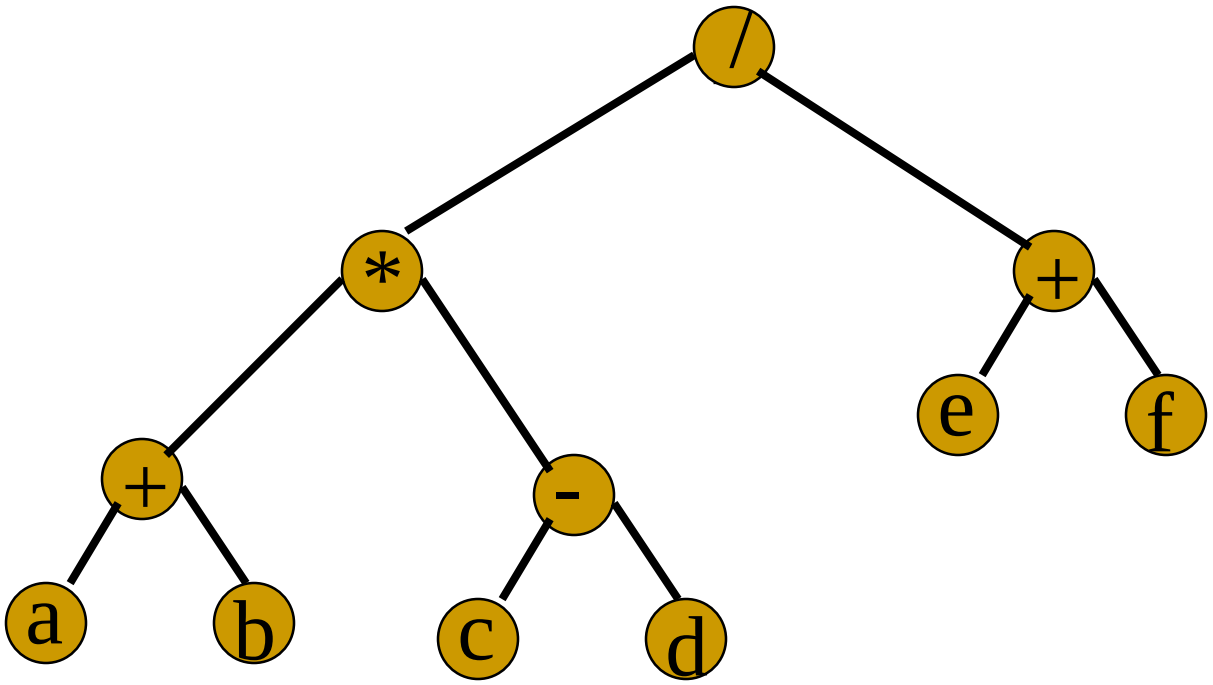
- شش حالت امکان پذیر است:
  - LVR, LRV, VLR, VRL, RVL, RLV

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

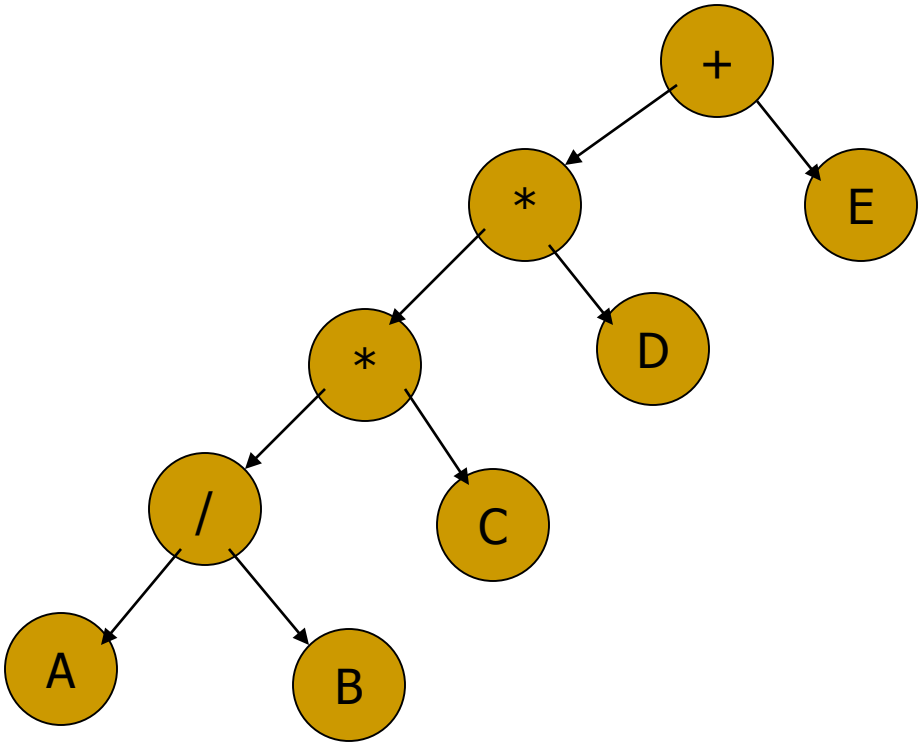
LVR	LRV	VLR
Inorder	Postorder	Preorder
میان ترتیب	پس ترتیب	پیش ترتیب

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

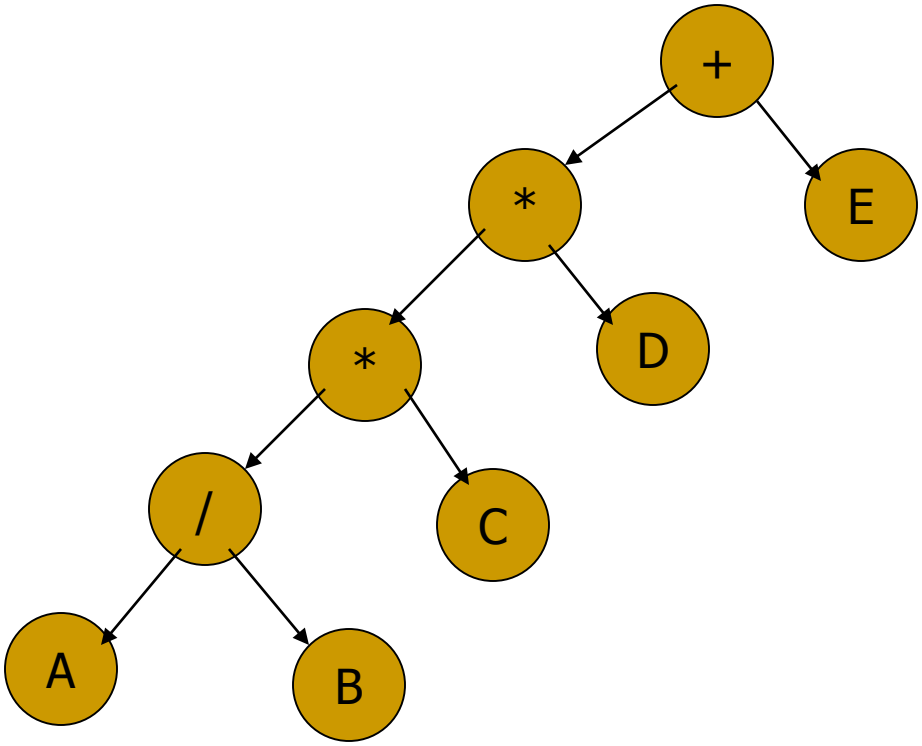
پیمایش درختها

- پیاده سازي 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

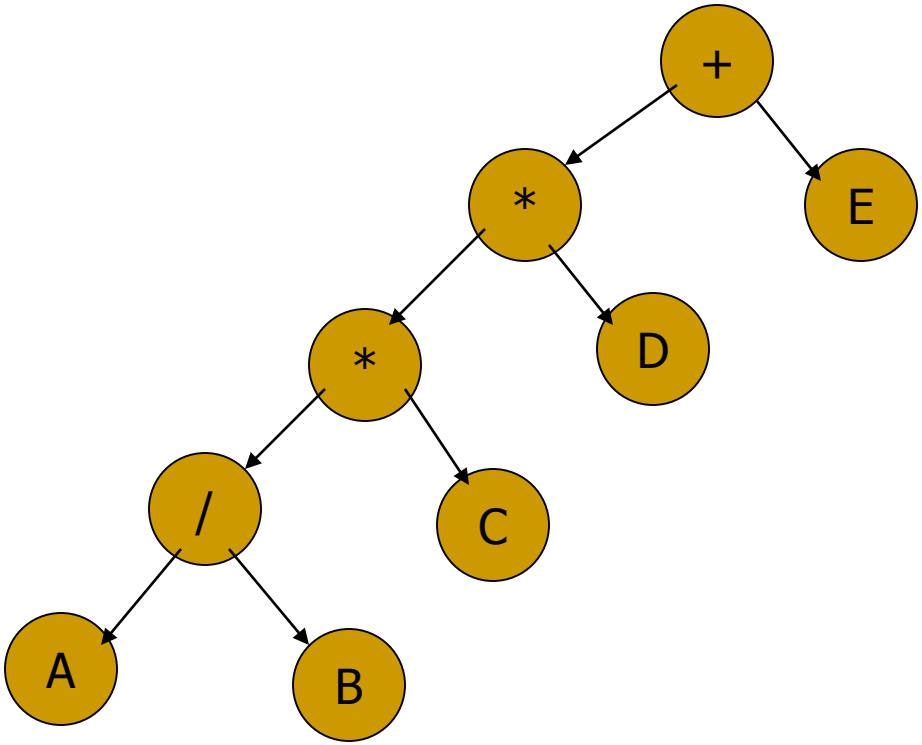
پیمایش درختها

- پیاده سازي 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 سازیPreorder :

```
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 ؟

کلاس تکرار کننده inorder

```
class InorderIterator {
  const Tree& t;
  Stack<TreeNode *> s;
  TreeNode *Cur;
public:
  char* Next();
  InorderIterator(const Tree& tree)
    :t(tree), Cur(tree.root)  // s(DefultSize)
  { }
};
```

```
char* InorderIterator::Next() {
  while(Cur) {
    s.Add(Cur);
    cur = cur->LeftChild;
  }
  if(! s.IsEmpty()) {
    s.Delete(Cur);
    char& tmp = Cur->data;
    Cur = Cur->RightChild;
    return &tmp;
  }
  else return 0; // no more elements
}
```

پیمایش سطحی درخت

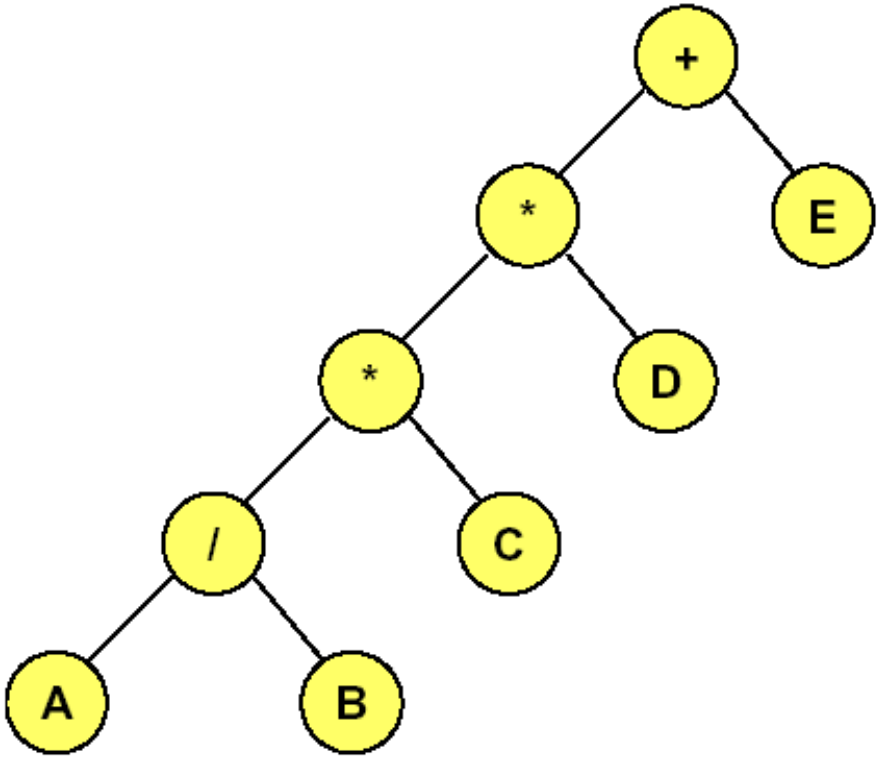
- برای پیمایش درختان چه به صورت بازگشتی و غیر بازگشتی و به ترتیب میانی، پسوندی و پیشوندی از پشته استفاده می شود.

- حال می خواهیم درختها را به صورت سطح به سطح پیمایش کنیم.
- برای انجام این کار از صف باید استفاده کنیم.
  - هنگام پیمایش سطح i گره های ان سطح را در صف قرار می دهیم.
  - سپس فرزندان سطح i را پیمایش کرده و دوبار انها را در صف قرار می دهیم.
  - این کار تا خالی شدن صف ادامه پیدا می کند.

تابع پیمایش سطحی درخت

```
void Tree::levelorder() {
  Queue<TreeNode *> q;
  TreeNode *Cur = root;
  while(Cur) {
    cout << Cur->data;
    if(Cur->LeftChild)
      q.Add(Cur->LeftChild);
    if(Cur->RightChild)
      q.Add(Cur->RightChild);

    q.Delete(Cur);  // delete from the head
  }
}
```



چند مثال درباره اعمال روی درختها

- کپی کردن یک درخت در درخت دیگر

```
Tree::Tree(const Tree& s) {
  root = copy(s.root);
}

TreeNodes* Tree::copy(TreeNode *orig) {
  if(orig) {
    TreeNode *tmp = new TreeNode;
    tmp->data = orig->data;
    tmp->LeftChild = copy(orig->LeftChild);
    tmp->RightChild = copy(orig->RightChild);
    return tmp;
  }
  return 0; // an empty binary tree
}
```