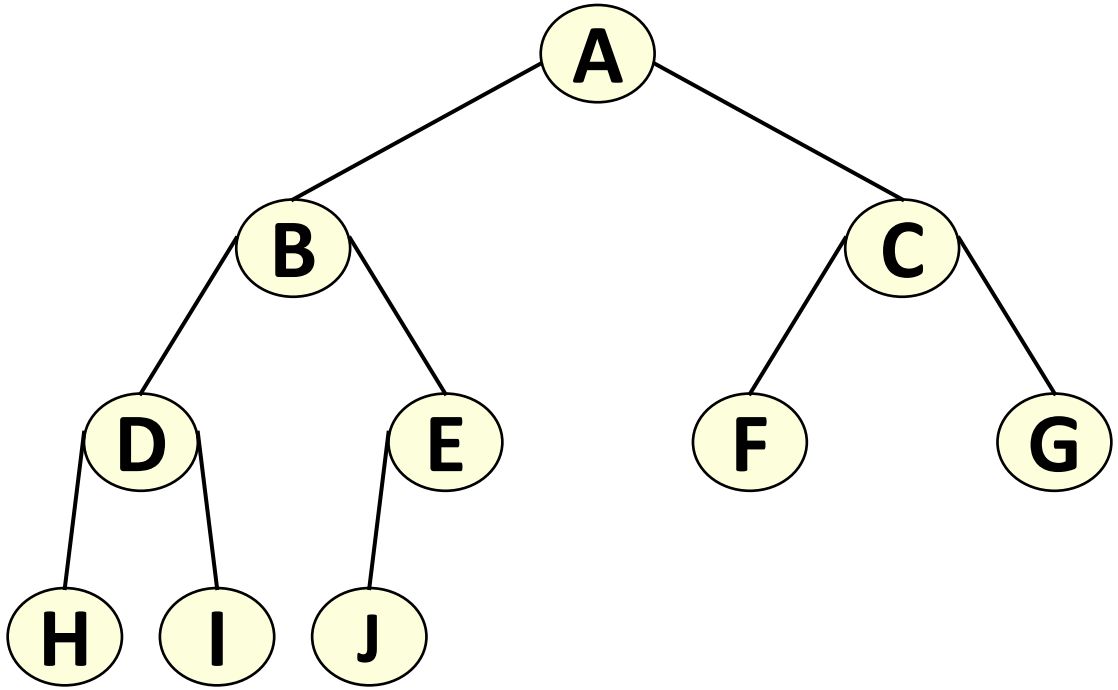
1. 二叉树的顺序存储结构

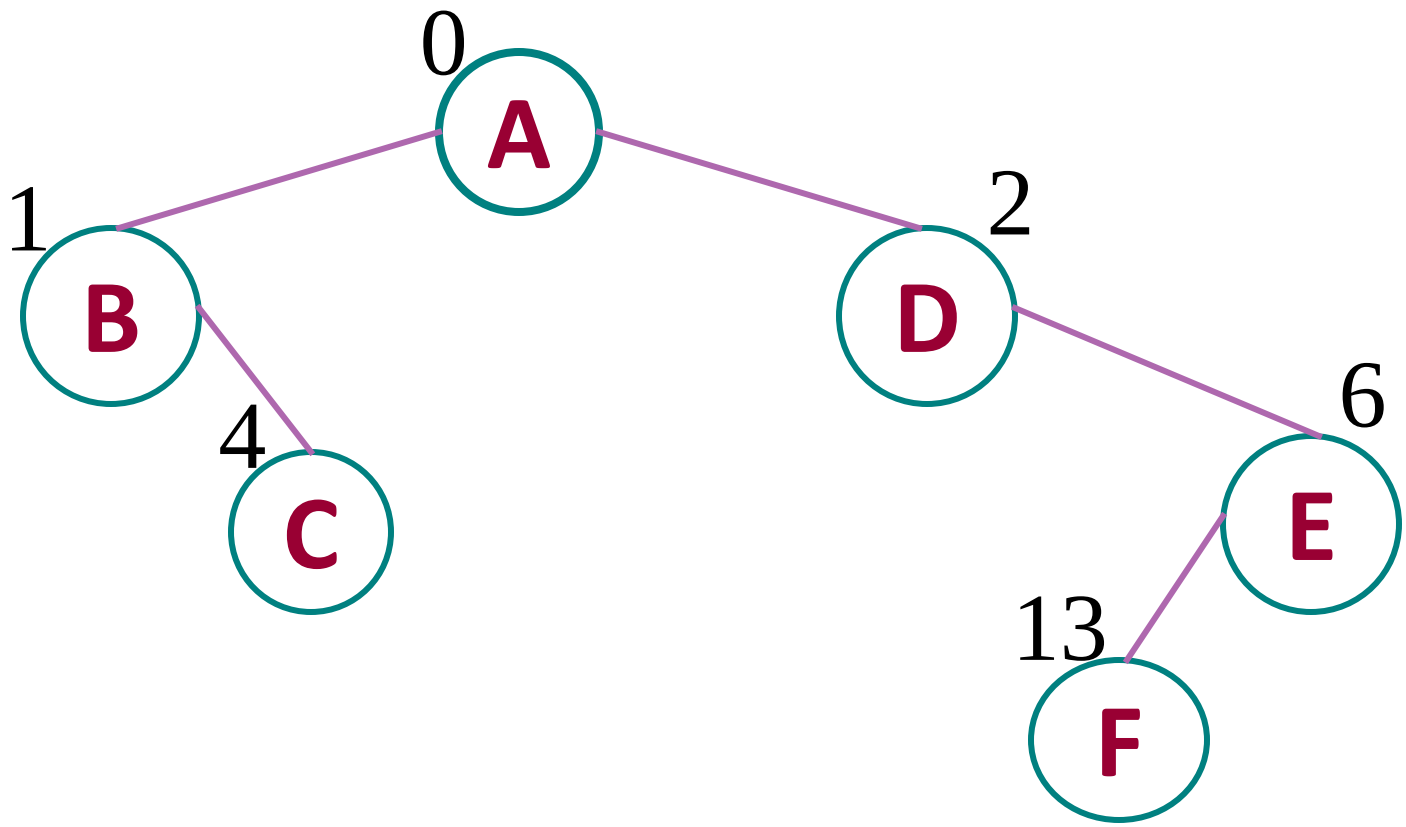
2. 二叉树的链式存储结构

对于完全二叉树：

用一组地址连续的存储单元从根结点开始依次自上而下，并按层次从左到右存储完全二叉树上的各结点元素，即将完全二叉树编号为i的结点元素存储在下标为i数组元素中。

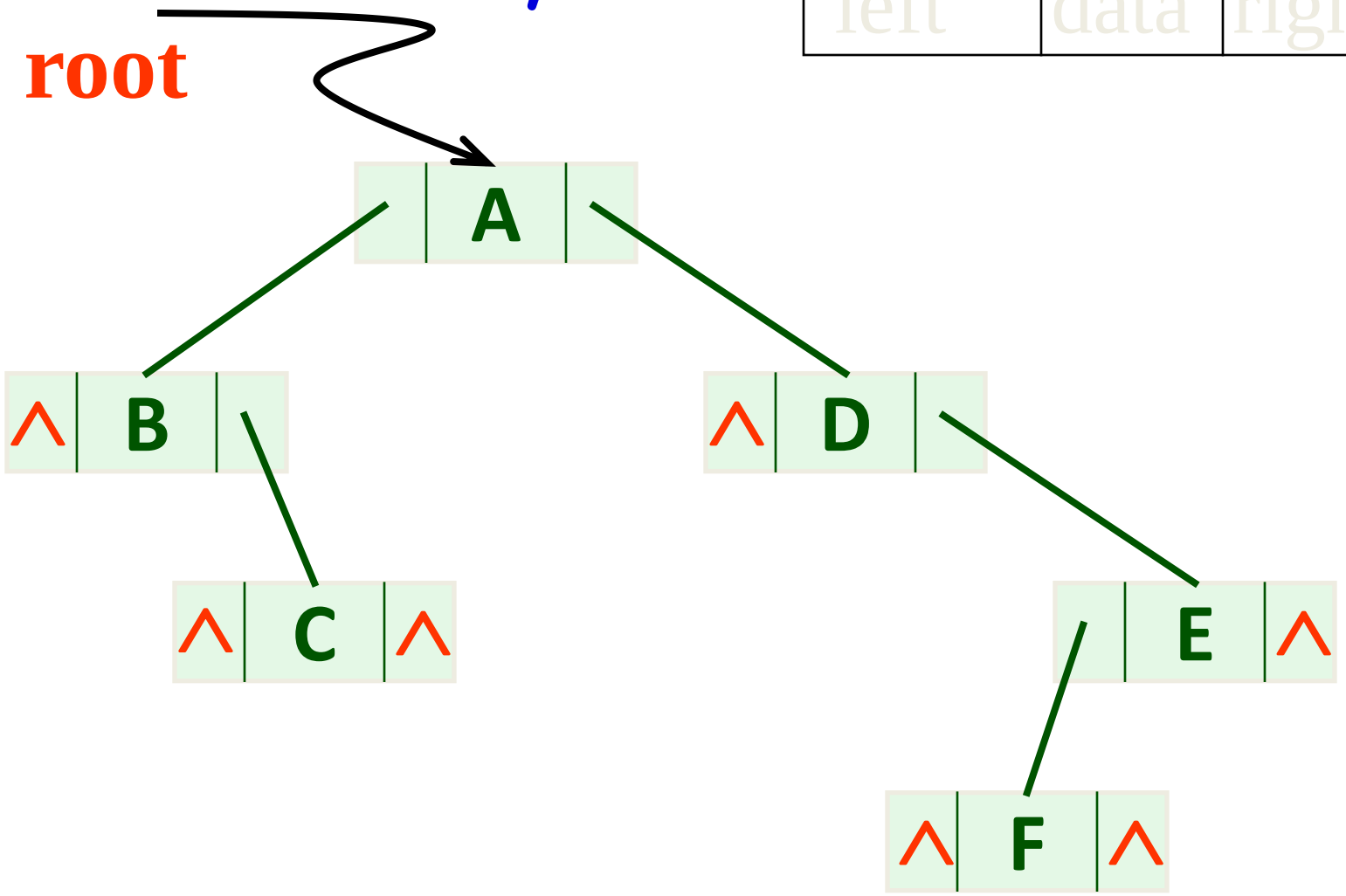


0	1	2	3	4	5	6	7	8	9	10	11	12	13
A	B	C	D	E	F	G	H	I	J				



0	1	2	3	4	5	6	7	8	9	10	11	12	13
A	B	D		C		E							F

Linked Binary Tree



Linked Binary Tree Specifications（链式二叉树规格说明）

- Binary node class：（二叉树结点类）

```
template <class Entry>
struct Binary_node {
    // data members:
    Entry data;
    Binary_node<Entry> *left;
    Binary_node<Entry> *right;
    // constructors:
    Binary_node( );
    Binary_node(const Entry &x);
};
```

Linked Binary Tree Specifications
（链式二叉树规格说明）

- Binary Tree Class

```
template <class Entry>
class Binary_tree {
public:
  // Add methods here.
protected:
  // Add auxiliary function prototypes here.
  Binary_node<Entry> *root;
};
```

Linked Binary Tree Specifications
（链式二叉树规格说明）

```
template <class Entry>
class Binary_tree {
public:
    Binary_tree( );
    bool empty( ) const;
    void preorder(void (*visit)(Entry &));
    void inorder(void (*visit)(Entry &));
    void postorder(void (*visit)(Entry &));
    int size( ) const;
    void clear( );
    int height( ) const;
    void insert(const Entry &);
```

Linked Binary Tree Specifications
（链式二叉树规格说明）

```
    Binary_tree (const Binary tree<Entry> &original);
    Binary_tree & operator = (const Binary_tree<Entry>
      &original);
    ~Binary_tree( );
    .....
protected:
// Add auxiliary function prototypes here.
    Binary_node<Entry> *root;
};
```

Linked Binary Tree Specifications
（链式二叉树规格说明）

- **Constructor:（初始化）**

```
template <class Entry>
Binary_tree<Entry> :: Binary_tree( )
/* Post: An empty binary tree has been created. */
{
        root = NULL;
}
```

Linked Binary Tree Specifications
（链式二叉树规格说明）

- Empty:（判空）

```
template <class Entry>
bool Binary_tree<Entry> :: empty( ) const
/* Post: A result of true is returned if the binary tree is
         empty. Otherwise, false is returned. */
{
        return root == NULL;
}
```

Linked Binary Tree Specifications
（链式二叉树规格说明）

– Inorder traversal:（中序遍历）

```
template <class Entry>
void Binary_tree<Entry> :: inorder(void (*visit)(Entry &))
/* Post: The tree has been been traversed in inorder sequence.
Uses: The function recursive_inorder */
{
recursive_inorder(root, visit);
}
```

Linked Binary Tree Specifications
（链式二叉树规格说明）

– Most Binary tree methods described by recursive processes can be implemented by calling an auxiliary （辅助的）recursive function that applies to subtrees.

Linked Binary Tree Specifications
（链式二叉树规格说明）

```
template <class Entry>
void
  Binary_tree<Entry> ::recursive_inorder(Binary_node
  <Entry> *sub_root,void (*visit)(Entry &))
{
  if (sub_root != NULL)
  {
        recursive_inorder(sub_ root->left, visit);
        (*visit)(sub_root->data);
        recursive_inorder(sub_root->right, visit);
  }
```

```
template  <class Entry>
int
Binary_Tree<Entry>::recursive_leafsize(Binary_node
<Entry> *sub_root){
if (sub_root==NULL)
        return 0;
if (sub_root->left==NULL && sub_root-
>right==NULL)
return 1;
return recursiveleaf_size(sub_root-
>left)+recursiveleaf_size(sub_root->right);
}
```

统计二叉树高度的算法

```
template <class Entry>
int Binary_tree<Entry> :: recursive_height(Binary_node<Entry>
*sub_root) const
/* Post: The height of the subtree rooted at sub_root is returned. */
{
    if (sub_root == NULL) return 0;
    int l = recursive_height(sub_root->left);
    int r = recursive_height(sub_root->right);
    if (l > r) return 1 + l;
    else return 1 +r;
}
```

拷贝构造函数

```
template <class Entry>
Binary_node<Entry> *Binary_tree<Entry> :: recursive_copy(
Binary_node<Entry> *sub_root)
/* Post: The subtree rooted at sub_root is copied, and a pointer
to the root of the new copy is returned. */
{
    if (sub_root == NULL) return NULL;
    Binary_node<Entry> *temp = new
    Binary_node<Entry>(sub_root->data);
    x= recursive_copy(sub_root->left);
    temp->left=x;
    temp->right = recursive_copy(sub_root->right);
    return temp;
}
```

二叉树清空

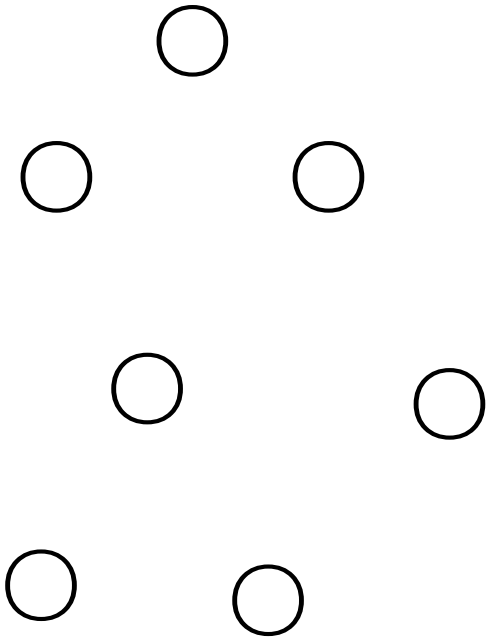
```
template <class Entry>
void Binary_tree<Entry> ::clear()
{
    root=recursive_clear(root);
}
template <class Entry>
Binary_node<Entry>   *
    Binary_tree<Entry>::recursive_clear(Binary_node<Entry>   * sub_root)
{
    if (sub_root != NULL)
    {
            sub_root->left=recursive_clear(sub_root->left);
            sub_root->right=recursive_clear(sub_root->right);
            delete sub_root;
    }
    return NULL;
}
```

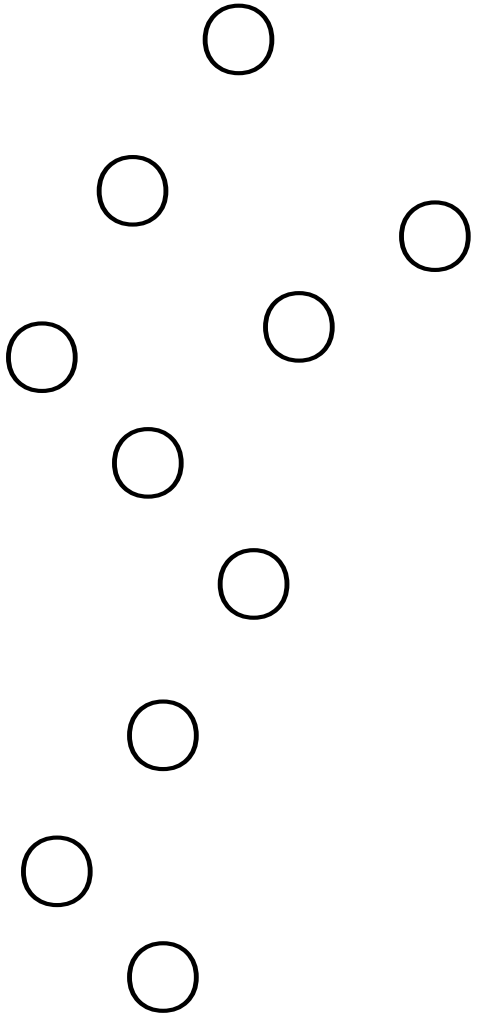
二叉树清空

```
template <class Entry>
void Binary_tree<Entry> ::clear()
{
    recursive_clear(root);
}
template <class Entry>
void Binary_tree<Entry>::recursive_clear(Binary_node<Entry>   * &sub_root)
{
    if (sub_root != NULL)
    {
            recursive_clear(sub_root->left);
            recursive_clear(sub_root->right);
            delete sub_root;
            sub_root=NULL;
    }
}
```

叶子结点删除

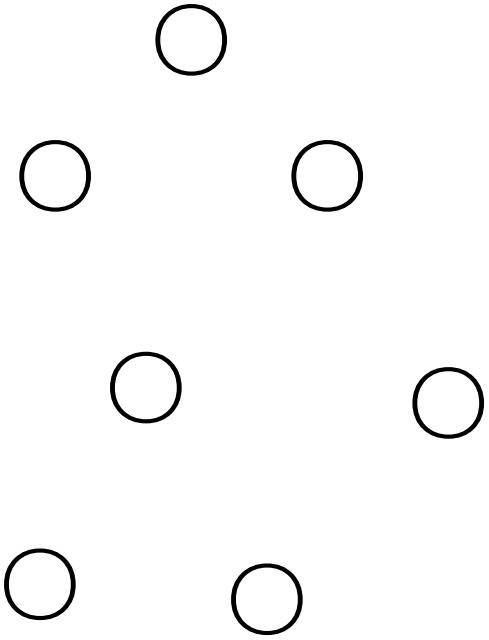
```
template <class Entry>
void Binary_tree<Entry>:: deleteleaf()
{recursive_deleteleaf(root);
}
template <class Entry>
void Binary_tree<Entry>:: recursive_deleteleaf(Binary_node<Entry>*
    &sub_root){
    if(sub_root==NULL)
            return;
    if (sub_root->left==NULL && sub_root->right==NULL)
    {delete sub_root;sub_root=NULL;return;}

    recursive_deleteleaf(sub_root->left);
    recursive_deleteleaf(sub_root->right);

    }
```



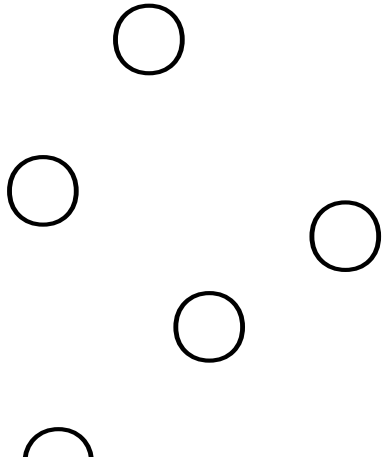


Level traversal

1. 设置一个队列
2. 非空root指针入队
3. 在队列非空时{
   ① 出队p
   ② 访问p->data
   ③ 将p的非空左右孩子入队
4. }



Non_recursive Inorder traversal



- 1、初始化空栈S，p=root;
- 2、当p不为空时，执行循环：
  - p入S栈
  - p=p->left;
- 3、当S栈非空时，
  - 出栈p
  - 访问p所指结点，
  - p=p->right,
  - 如果p为空，继续出栈（循环至3）；
  - 如果p不为空，循环至2
- 4、否则结束

Non_recursive inorder traversal

1. 设置一个空栈S
2. 设置活动指针p，初值p=root
3. 当栈S不空时或p不空时，执行循环{
   ① 当p不为NULL时重复循环：p入栈，p=p->left;
   ② 当S不空时，出栈栈顶p，访问p->data
   ③ p=p->right

}

```
template <class Entry>
void Binary_tree<Entry>:: inorder(void
(*visit)(Entry &)){
if (root != NULL) {
Stack S;
Binary_node<Entry>*p= root;
while (!S.Empty() ||p!=NULL) {
   while (p!= null){
       S.push(p);p=p->left;}
   if (!S.Empty()) {
      S.pop(p);
      visit(p.data);
      p=p.right;}
}}
}
```

1、二叉树的逻辑定义
2、二叉树的三种遍历序列
3、二叉树遍历与表达式树
4、二叉树的链式实现（二叉链表）
5、二叉链表下的递归算法
总体思路：
将二叉树分成三部分。
运用递归特性。
如：前序、中序、后序遍历、求结点数、深度等的算法。
有一个公共方法，呈现给客户程序。
有一个私有辅助递归函数。
6、层次遍历算法
7、前序、中序非递归算法

更多算法

- 以两个序列创建二叉链表
- 统计度为1的结点数
- 统计度为2的结点数
- 统计二叉树的宽度
- 计算二叉树中指定结点p所在层次
- 计算二叉树中最大元素值
- 删除度为1的结点数
- ……