6.2 二叉树

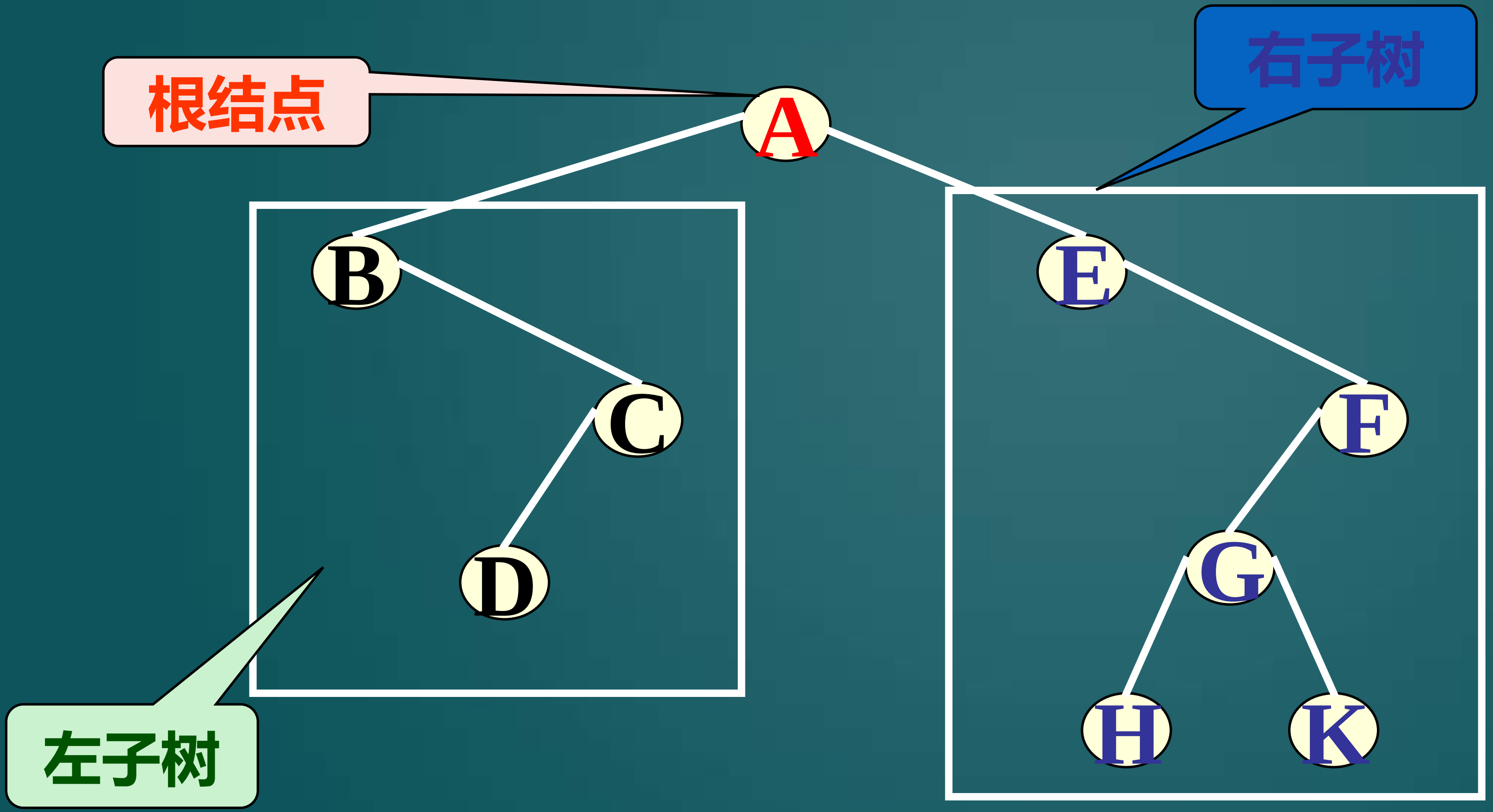
6.2.1 二叉树的定义

6.2.2 二叉树的性质

6.2.3 二叉树的存储结构

6.2.1 二叉树的定义

二叉树或为**空树**，或是由一个**根结点**加上**两棵**分别称为**左子树**和右子树的、**互不交的二叉树**组成。



ADT BinaryTree{

数据对象 D： D是具有相同特性的数据元素的集合。

数据关系 R：

若D= Ø，则R= Ø，称BinaryTree为空二叉树。否则，

R={H}，H是如下二元关系:

(1) 在D中存在唯一的称为根的数据元素root；它在H下无前驱

(2) 若D- {root} ≠Ø，则存在D- {root}= { D_l, D_r}, 且D_l∩ D_r= Ø;

(3) 若D_l ≠Ø ，则D_l中存在惟一的元素x_l，<root，x_l>∈H，且存在D_l上的关系H_l∈H；若D_r ≠Ø ，则D_r中存在惟一的元素x_r，<root，x_r>∈H，且存在D_r上的关系H_r∈H；

H={<root，x_l>，<root，x_r>，H_l , H_R}；

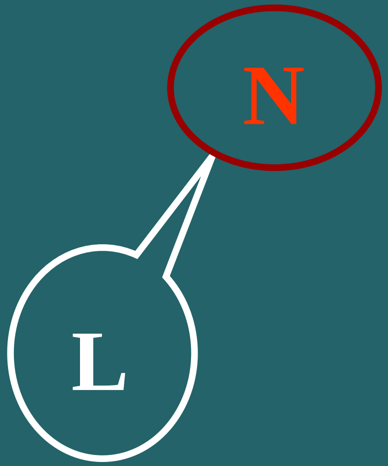
（4）(D_l，{H_l})是一棵符合本定义的二叉树，称为根的左子树，(D_r，{H_r})是一棵符合本定义的二叉树，称为根的右子树，

二叉树的五种基本形态：

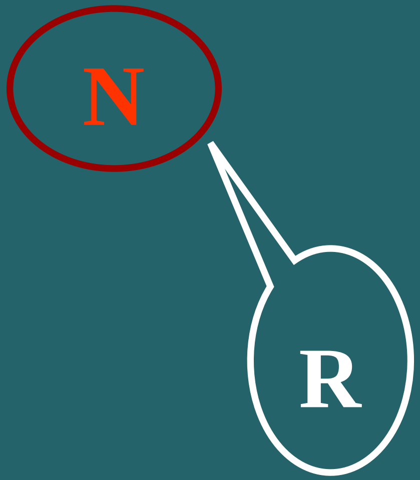
空树

只含根结点

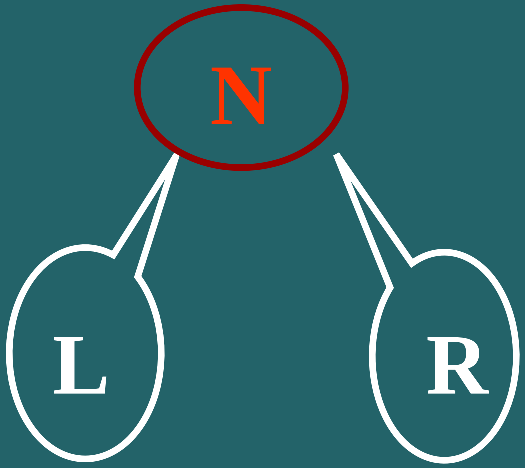
右子树为空树



左子树为空树



左右子树均不为空树



二叉树的主要基本操作：

Root(T);   Value(T, e);   Parent(T, e);

LeftChild(T, e);   RightChild(T, e);

LeftSibling(T, e);   RightSibling(T, e);

BiTreeEmpty(T);   BiTreeDepth(T);

PreOrderTraverse(T, Visit());

InOrderTraverse(T, Visit());

PostOrderTraverse(T, Visit());

LevelOrderTraverse(T, Visit());

InitBiTree(&T);

Assign(T, &e, value);

CreateBiTree(&T, definition);

InsertChild(T, p, LR, c);

ClearBiTree(&T);

DestroyBiTree(&T);

DeleteChild(T, p, LR);

具体内容见讲义 P121-P123

6.2.2 二叉树的性质

性质 1 ：

在二叉树的第 i 层上至多有 2^{i-1} 个结点 $(i \geq 1)$。

用归纳法证明：

归纳基： $i = 1$ 层时，只有一个根结点：

$$2^{i-1} = 2^0 = 1\text{；}$$

归纳假设： 假设对所有的 j，$1 \leq j < i$，命题成立；
即第 j 层上至多有 2^{j-1} 个结点。

归纳证明： 二叉树上每个结点至多有两棵子树，
则第 i 层的结点数 $= 2^{i-2} \times 2 = 2^{i-1}$ 。

归纳假设

性质 2 ：

深度为 **k** 的二叉树上至多含 $\mathbf{2^k-1}$ 个结点（k≥1）。

证明：

基于上一条性质，深度为 *k* 的二叉树上的结点数至多为

$$2^0+2^1+\cdots\cdots+2^{k-1}=2^k-1。$$

性质 3 ：

对任何一棵二叉树，若它含有n_0个叶子结点、n_2个度为 2 的结点，则必存在关系式：$n_0 = n_2+1$。

证明：

设 二叉树上结点总数为**n，则** $n = n_0 + n_1 + n_2$，

其中n_1为度为1的结点数。

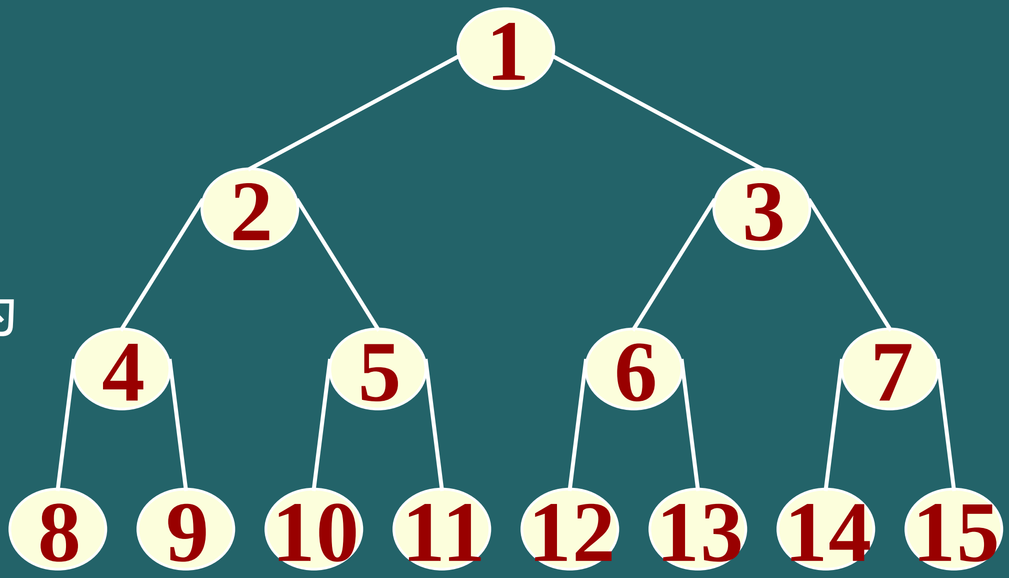
又 二叉树上分支总数 $b = n_1+2n_2$

而 $b = n-1 = n_0 + n_1 + n_2 - 1$

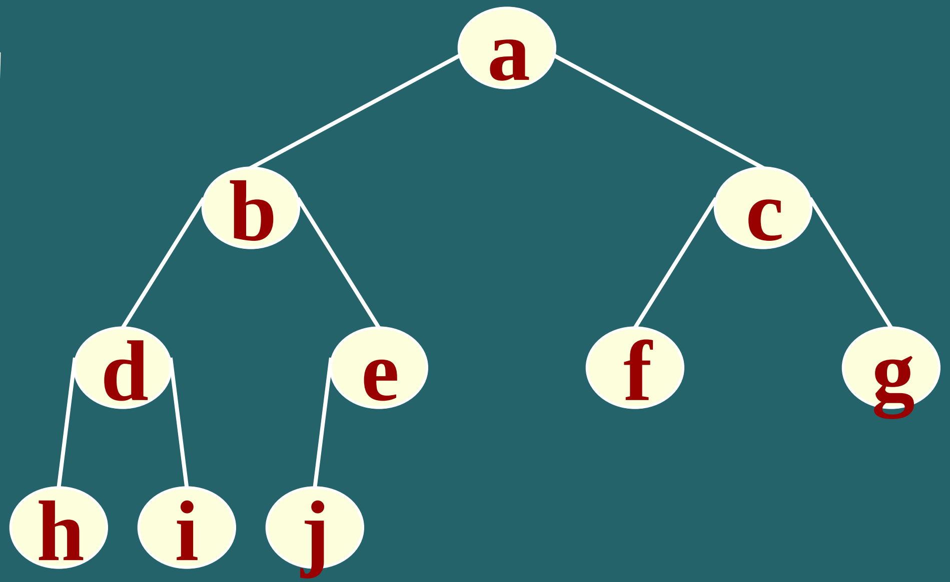
由此， $n_0 = n_2 + 1$ 。

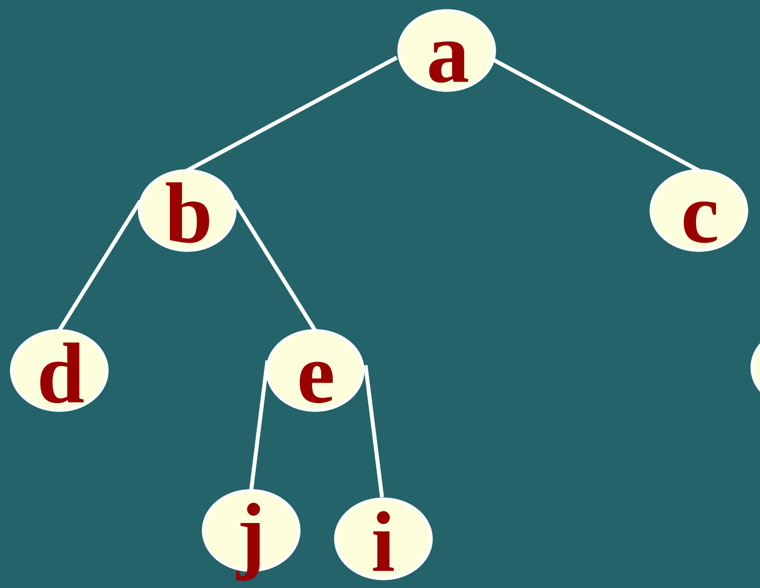
两类特殊的二叉树：

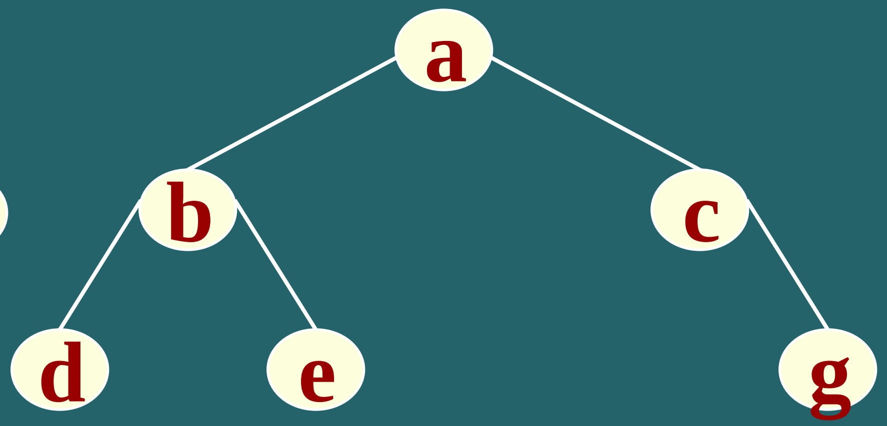
满二叉树：指的是深度为**k**且含有**2^k-1**个结点的二叉树。



完全二叉树：树中所含的**n** 个结点和满二叉树中**编号为 1 至 n 的结点**一一对应。







非完全二叉树

性质 4 ：

具有 n 个结点的完全二叉树的**深度**为$\lfloor \log_2 n \rfloor + 1$。

证明：

设完全二叉树的深度为 k

则根据第二条性质得 ：

$2^{k-1}-1 < n \le 2^k-1$ 或 $2^{k-1} \le n < 2^k$

即 $k-1 \le \log_2 n < k$

因为 k 只能是整数，因此，$k = \lfloor \log_2 n \rfloor + 1$ 。

性质 5 ：

若对含 n 个结点的完全二叉树从上到下且从左至右进行 1 至 n 的编号，则对完全二叉树中任意一个编号为 i 的结点：

（1）若 i=1，则该结点是二叉树的根，无双亲，

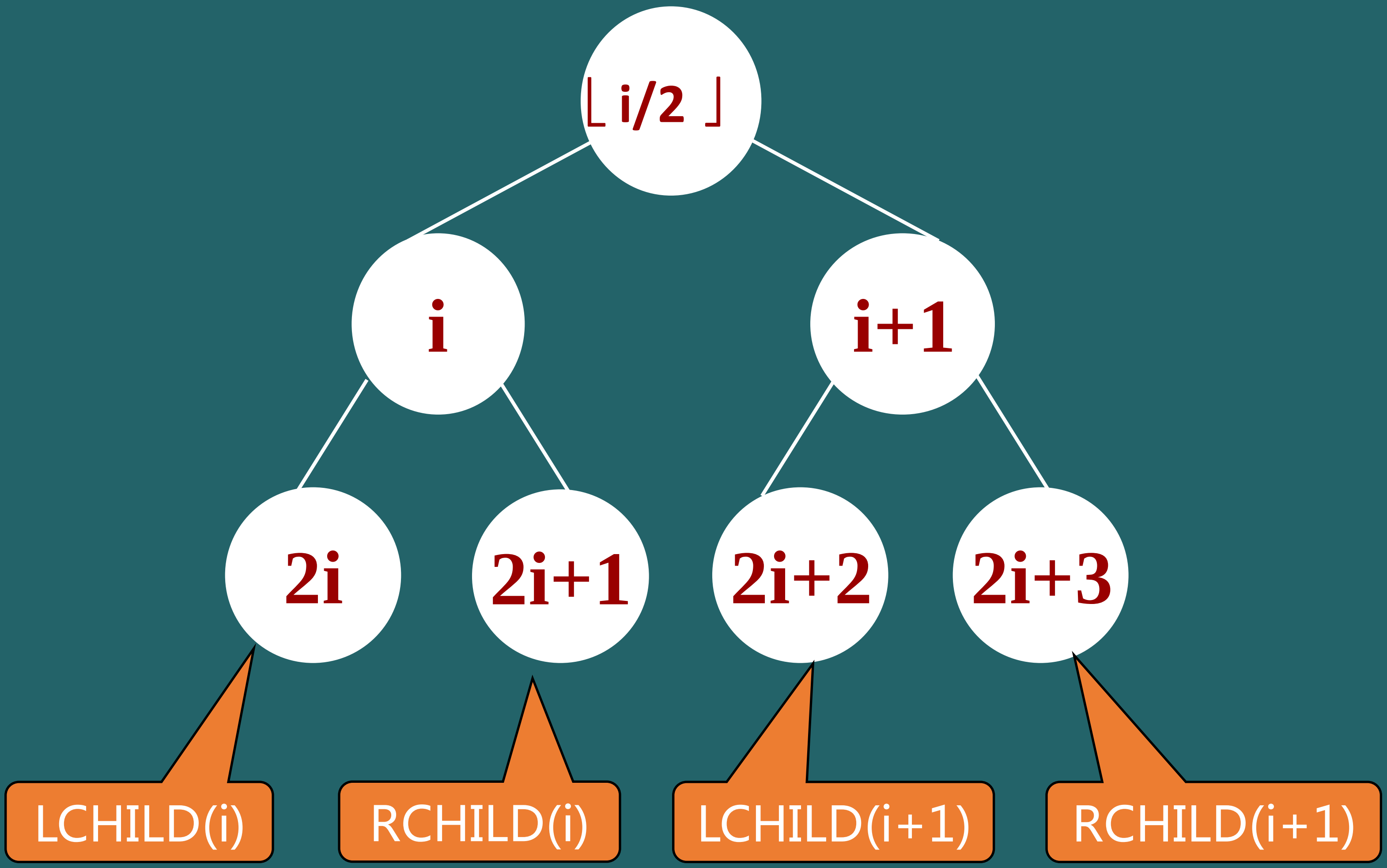
否则，编号为 $\lfloor i/2 \rfloor$ 的结点为其**双亲**结点；

（2）若 2i>n，则该结点无左孩子，

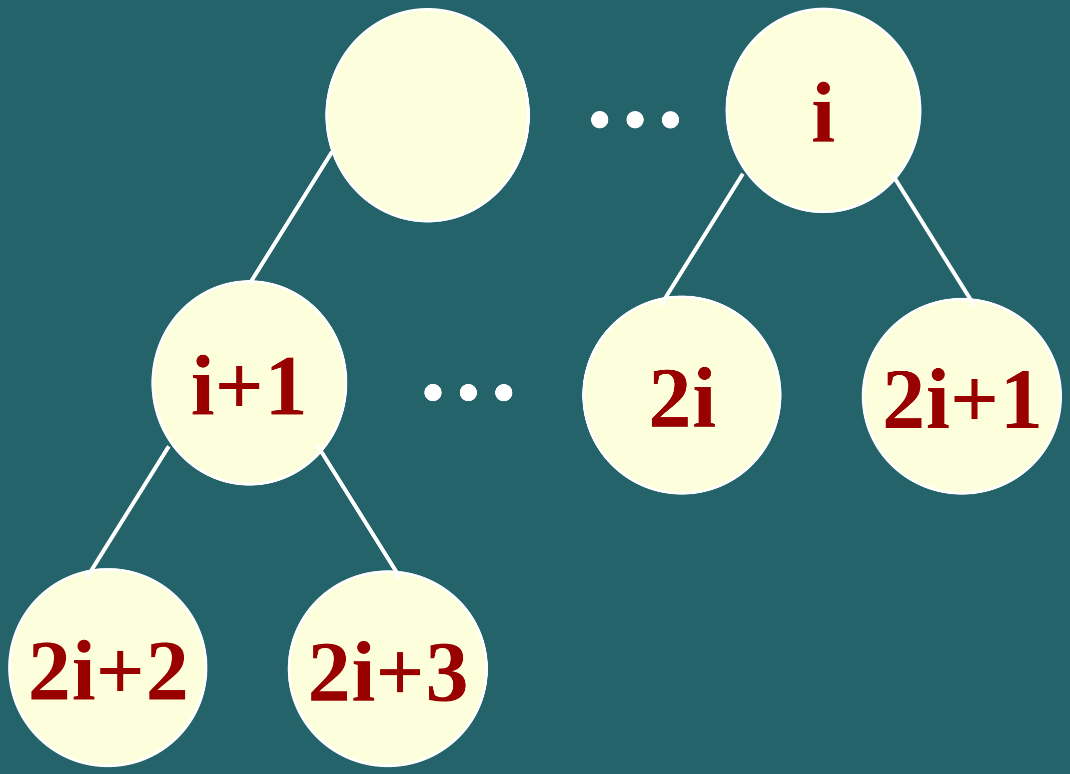
否则，编号为 2i 的结点为其**左孩子**结点；

（3）若 2i+1>n，则该结点无右孩子结点，

否则，编号为2i+1 的结点为其**右孩子**结点。



(a) 结点i和i+1在同一层上；



(b）结点i和i+1不在同一层上；

图6.5 完全二叉树中结点i和i+1的左、右孩子

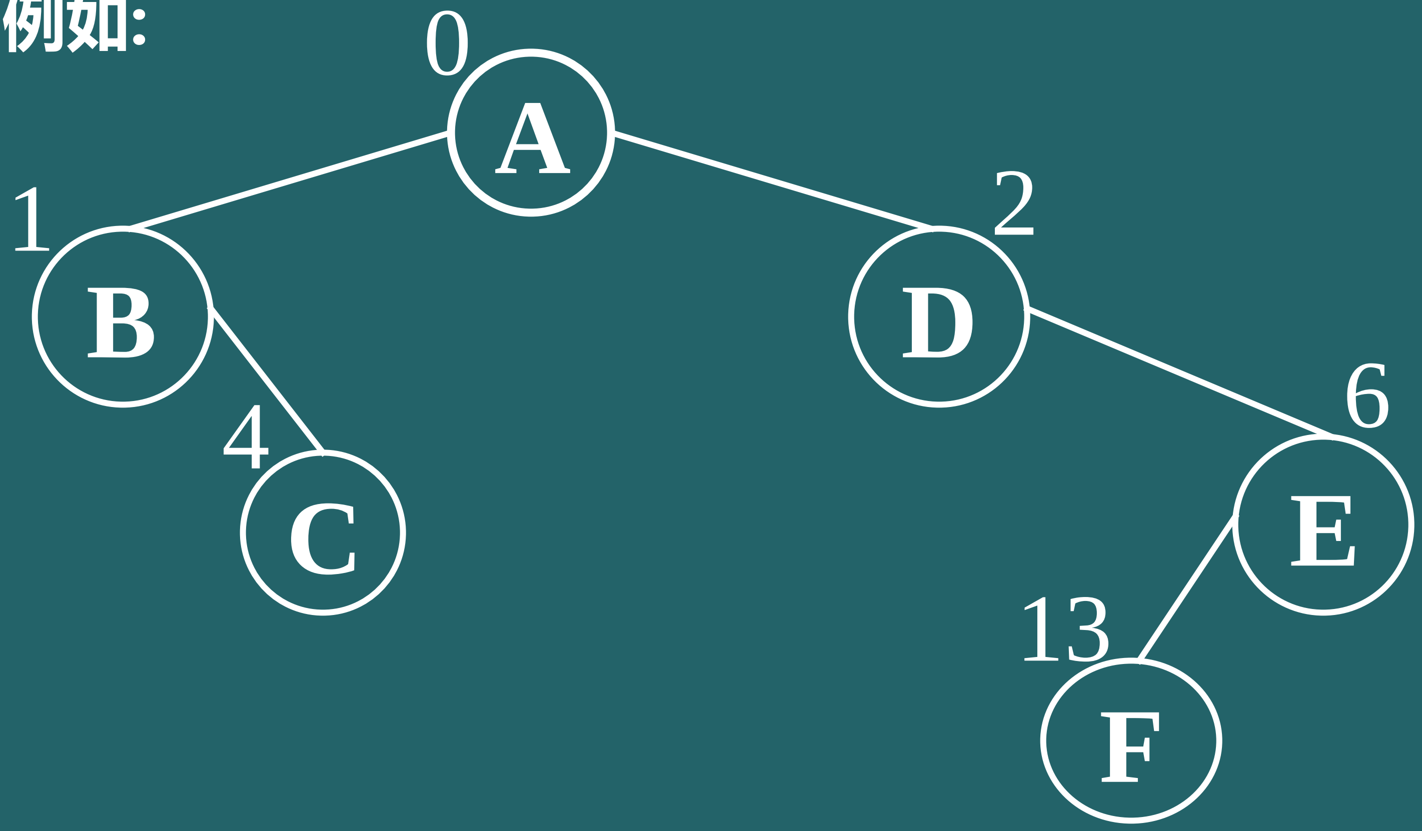
6.2.3 二叉树的存储结构

一、 二叉树的顺序存储表示

```
#define  MAX_TREE_SIZE  100
              // 二叉树的最大结点数
typedef  TElemType SqBiTree[MAX_TREE_SIZE];
              // 0号单元存储根结点
SqBiTree bt;
```

用一组地址连续的存储单元依次自上而下、自左至右存储完全二叉树上的节点元素，即将完全二叉树上编号为i的结点元素存储在如上定义的一维数组中下标为i-1的分量中。

例如:



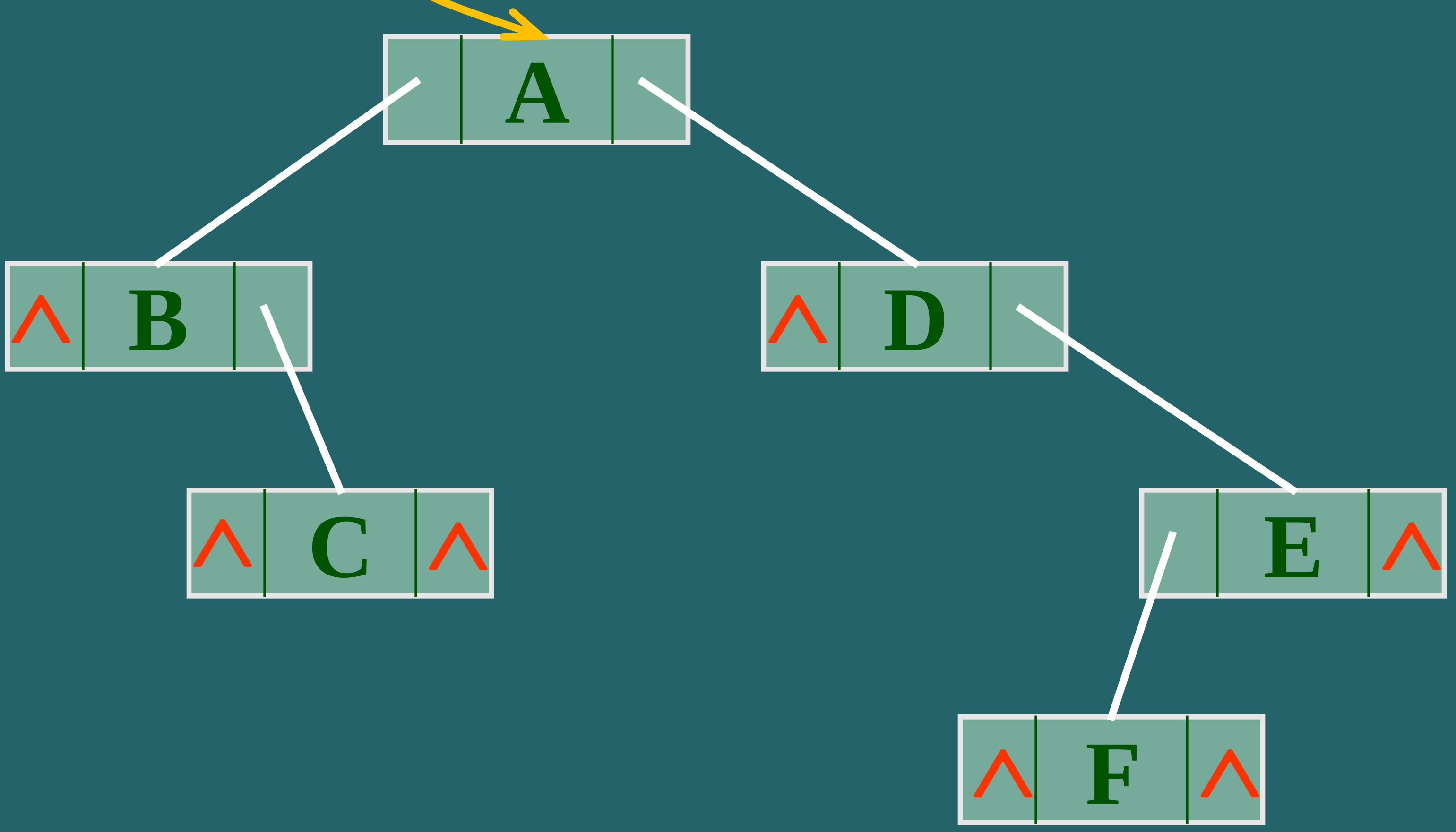
0	1	2	3	4	5	6	7	8	9	10	11	12	13
A	B	D		C		E							F

二、二叉树的链式存储表示

1. 二叉链表

结点结构:

lchild	data	rchild



C **语言的类型描述如下:**

```
typedef struct BiTNode { // 结点结构
    TElemType    data;
    struct BiTNode   *lchild, *rchild;
                    // 左右孩子指针
} BiTNode, *BiTree;
```

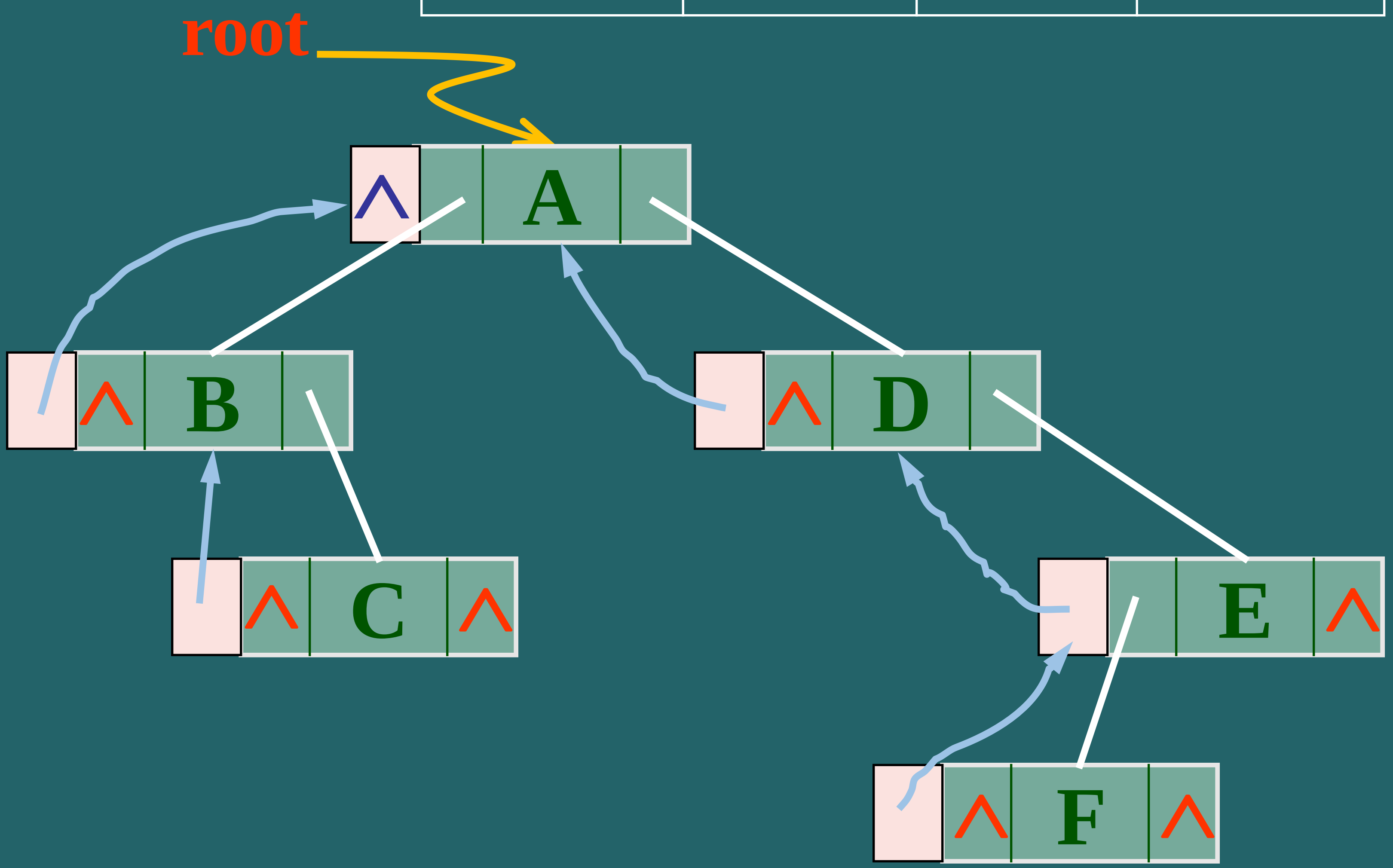
结点结构:

lchild	data	**r**child

2．三叉链表

结点结构:

Parent	lchild	data	rchild



C **语言的类型描述如下:**

```
typedef struct TriTNode { // 结点结构
TElemType      data;
struct TriTNode  *lchild, *rchild;
                // 左右孩子指针
struct TriTNode  *parent;  //双亲指针
} TriTNode, *TriTree;
```

结点结构:

parent	**lchild**	**data**	**rchild**