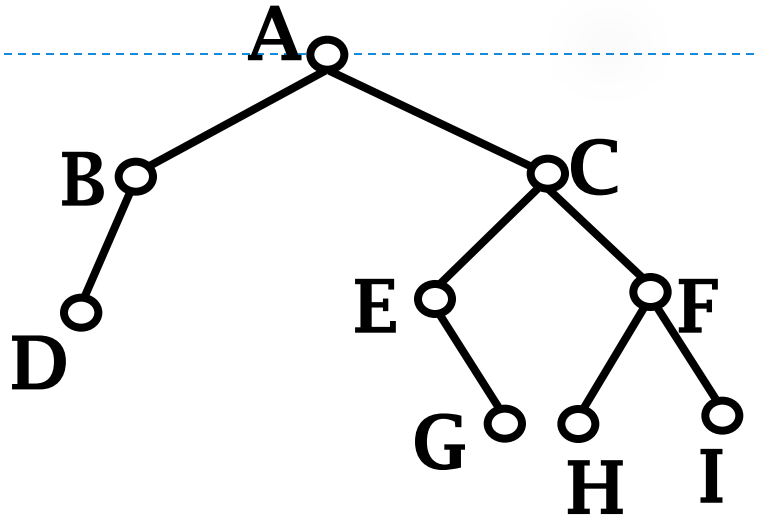
数据结构与算法(五)

张铭 主讲

采用教材：张铭，王腾蛟，赵海燕 编写
高等教育出版社，2008. 6 （“十一五”国家级规划教材）

http://www.jpk.pku.edu.cn/pkujpk/course/sjjg

第五章 二叉树



- 二叉树的概念
- 二叉树的抽象数据类型
  - 深度优先搜索
  - 宽度优先搜索
- 二叉树的存储结构
- 二叉搜索树
- 堆与优先队列
- Huffman树及其应用

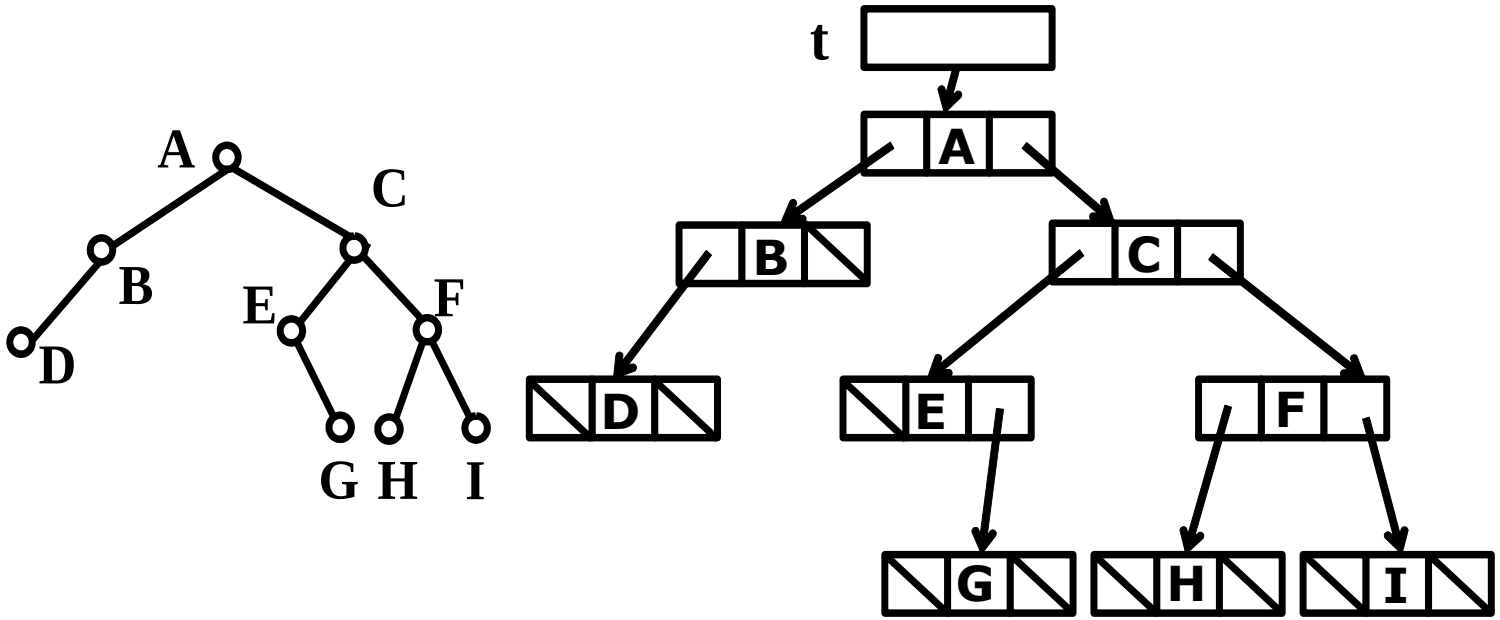
二叉树的链式存储结构

二叉树的各结点随机地存储在内存空间中，结点之间的逻辑关系用指针来链接。

- 二叉链表
  - 指针 left 和 right，分别指向结点的左孩子和右孩子

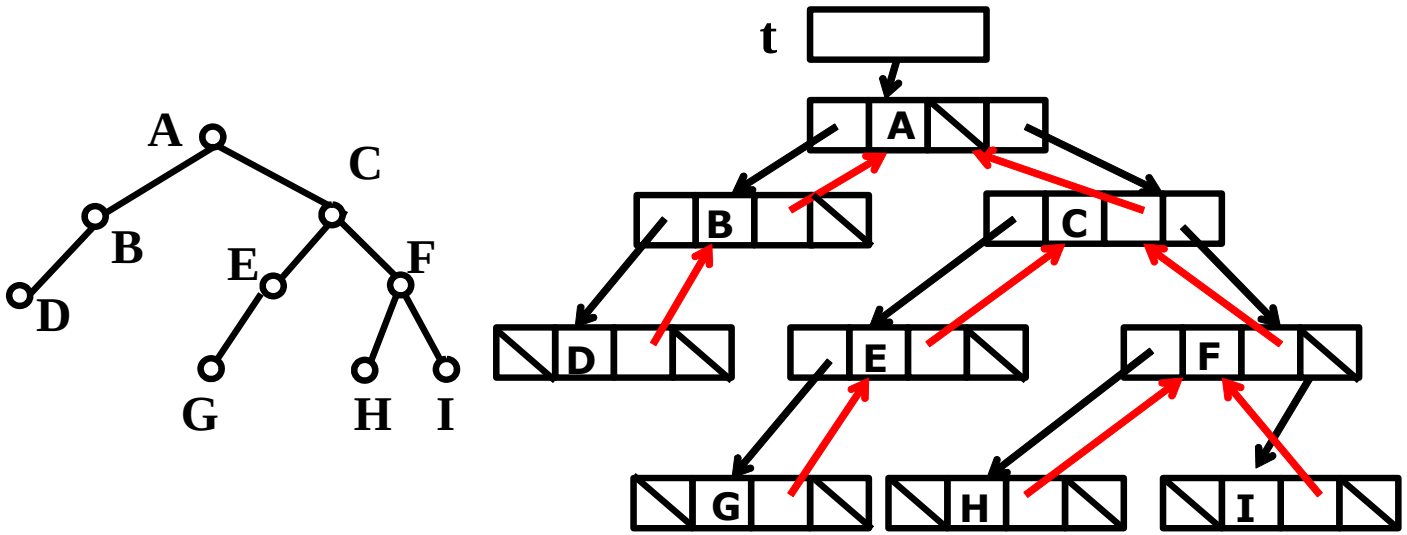
- 三叉链表
  - 指针 left 和 right，分别指向结点的左孩子和右孩子
  - 增加一个父指针

二叉链表



“三叉链表”

- 指向父母的指针parent，“向上”能力



BinaryTreeNode类中增加两个私有数据成员

```
private:
    BinaryTreeNode<T> *left;              // 指向左子树的指针
    BinaryTreeNode<T> *right;             // 指向右子树的指针
```

```
template <class T> class BinaryTreeNode  {
friend class BinaryTree<T>;              // 声明二叉树类为友元类
private:
   T  info;                              // 二叉树结点数据域
public:
   BinaryTreeNode();                     // 缺省构造函数
   BinaryTreeNode(const T& ele);         // 给定数据的构造
   BinaryTreeNode(const T& ele, BinaryTreeNode<T> *l,
                  BinaryTreeNode<T> *r); // 子树构造结点
.... }
```

递归框架寻找父结点——注意返回

```
template<class T>
BinaryTreeNode<T>* BinaryTree<T>::
Parent(BinaryTreeNode<T> *rt, BinaryTreeNode<T> *current)  {
    BinaryTreeNode<T> *tmp,
    if (rt == NULL) return(NULL);
    if (current == rt ->leftchild() || current == rt->rightchild())
        return rt; // 如果孩子是current则返回parent
    if ((tmp =Parent(rt- >leftchild(), current) != NULL)
        return tmp;
    if ((tmp =Parent(rt- > rightchild(), current) != NULL)
        return tmp;
    return NULL;
}
```

思考

- 该算法是什么框架？
- 该算法是什么序遍历？
- 可以怎样改进？
  - 可以用非递归吗？
  - 可以用BFS吗？
- 怎样从这个算法出发，寻找兄弟结点

非递归框架找父结点

```
BinaryTreeNode<T>* BinaryTree<T>::Parent(BinaryTreeNode<T> *current) {
    using std::stack;                                   // 使用STL中的栈
    stack<BinaryTreeNode<T>* > aStack;
    BinaryTreeNode<T> *pointer = root;
    aStack.push(NULL);                                  // 栈底监视哨
    while (pointer) {                                   // 或者!aStack.empty()
        if (current == pointer->leftchild() || current == pointer->rightchild())
            return pointer;                 // 如果pointer的孩子是current则返回parent
        if (pointer->rightchild() != NULL)              // 非空右孩子入栈
            aStack.push(pointer->rightchild());
        if (pointer->leftchild() != NULL)
            pointer = pointer->leftchild();             // 左路下降
        else {                                          // 左子树访问完毕，转向访问右子树
            pointer=aStack.top(); aStack.pop();   // 获得栈顶元素，并退栈
} } }
```

空间开销分析

- 存储密度 α (≤1) 表示数据结构存储的效率

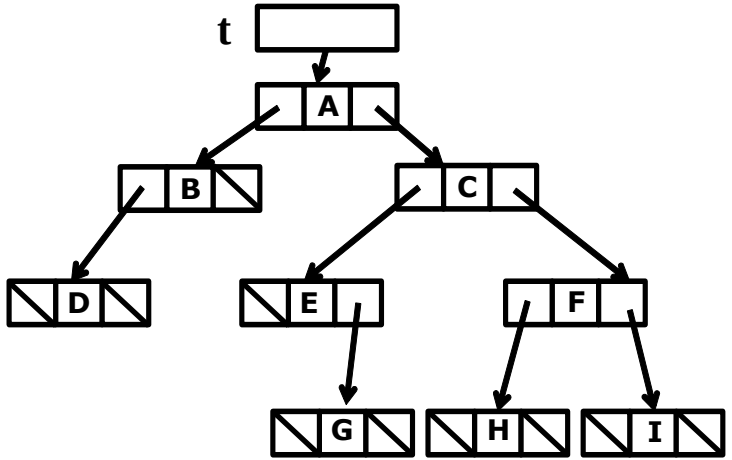
$$\alpha(\text{存储密度}) = \frac{\text{数据本身存储量}}{\text{整个结构占用的存储总量}}$$

- 结构性开销 γ = 1- α

空间开销分析

根据满二叉树定理：一半的指针是空的

- 每个结点存两个指针、一个数据域
  - 总空间 $(2p + d)n$
  - 结构性开销：$2pn$
  - 如果 $p = d$, 则结构性开销 $2p/(2p + d) = 2/3$



空间开销分析

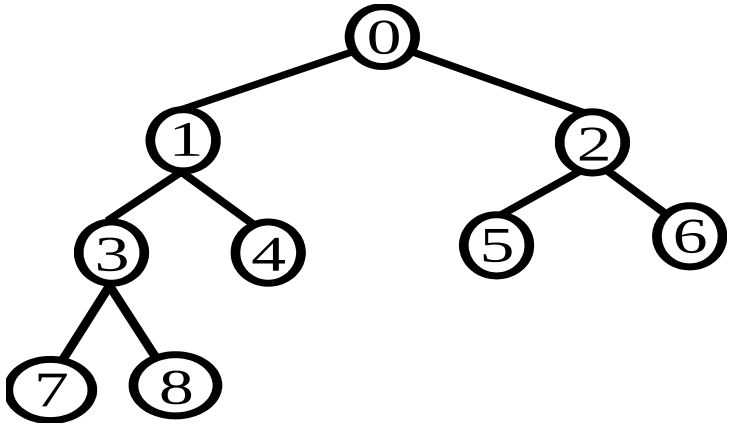
- C++ 可以用两种方法来实现不同的分支与叶结点：
  - 用union联合类型定义
  - 使用C++的子类来分别实现分支结点与叶结点，
    并采用虚函数isLeaf来区别分支结点与叶结点
- 早期节省内存资源
  - 利用结点指针的一个空闲位（一个bit）来标记结点所属的类型
  - 利用指向叶的指针或者叶中的指针域来存储该叶结点的值

完全二叉树的顺序存储结构

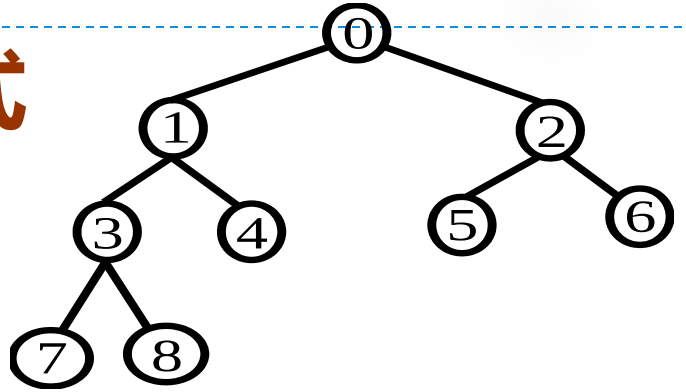
- 顺序方法存储二叉树
  - 把结点按一定的顺序存储到一片连续的存储单元
  - 使结点在序列中的位置反映出相应的结构信息
- 存储结构上是线性的

0	1	2	3	4	5	6	7	8

- 逻辑结构上它仍然是二叉树形结构



完全二叉树的下标公式

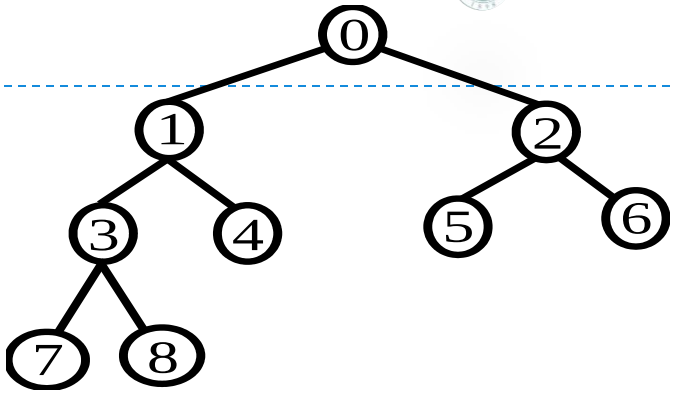


- 从结点的编号就可以推知

其父母、孩子、兄弟的编号

- 当 2i+1<n 时，结点 i 的左孩子是结点 2i+1，
  否则结点i没有左孩子
- 当 2i+2<n 时，结点 i 的右孩子是结点 2i+2，
  否则结点i没有右孩子

完全二叉树的下标公式



- 当 0<i<n 时，结点 i 的父亲是结点 ⌊(i-1)/2⌋
- 当 i 为偶数且 0<i<n 时，结点 i 的左兄弟是结点 i-1，
  否则结点 i 没有左兄弟
- 当 i 为奇数且 i+1<n 时，结点i的右兄弟是结点 i+1，
  否则结点i没有右兄弟

思考

- 用三叉链的存储形式修改二叉树的相应算法。特别注意插入和删除结点，维护父指针信息。
- 完全三叉树的下标公式？

数据结构与算法

谢谢聆听

国家精品课“数据结构与算法”
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

张铭，王腾蛟，赵海燕
高等教育出版社，2008. 6。“十一五”国家级规划教材