阅读：9,748　　作者：解学武

二叉树顺序存储和链式存储的C语言代码实现

通过前一节的学习，了解了树的一些基本知识。二叉树是在树的基础上对本身的结构做了更高的限制：

1. 二叉树本身是有序树。
2. 二叉树中各结点的度最多是 2，可以是 0，1，2。

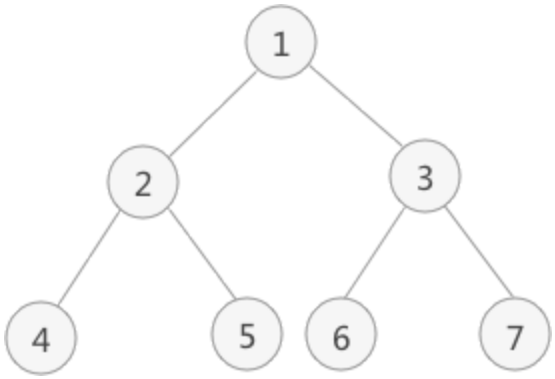
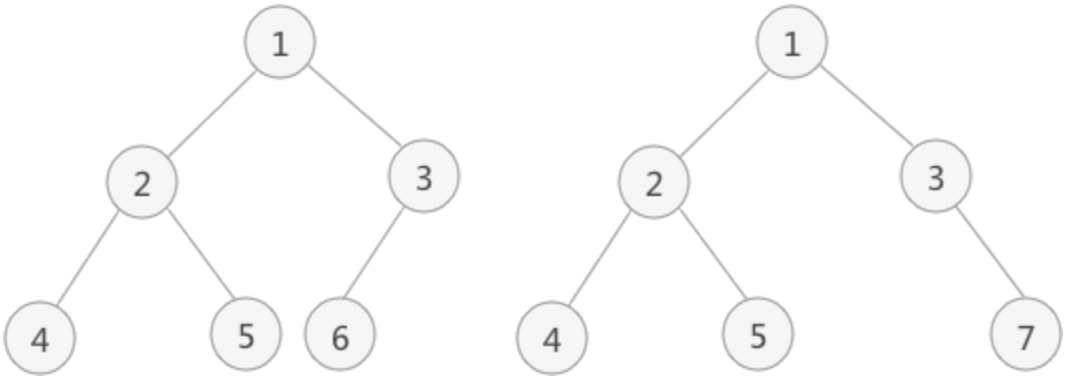


图1 二叉树

满二叉树和完全二叉树

如果二叉树中除了叶子结点，每个结点的度都为 2，那么此二叉树为满二叉树。例如图 1 就是一个满二叉树。

如果二叉树除了最后一层外为满二叉树，最后一层的结点依次从左到右分布，此二叉树为完全二叉树。



（A）　　（B）

图2 完全二叉树

图 2（A）和（B）都是二叉树，但图 2（A）是完全二叉树，（B）由于最后一层不符合从左往右依次分布的要求，所以不是完全二叉树，只是一个普通的二叉树。

二叉树的性质

二叉树有以下几个性质：

1. 二叉树中，第 i 层最多有 $2i-1$ 个结点。
2. 如果二叉树的深度为 K，那么此二叉树最多有 $2K-1$ 个结点。
3. 二叉树中，终端结点数（叶子结点数）为 n_0，度为 2 的结点数为 n_2，则 $n_0 = n_2 + 1$ 。

> 性质 3 的计算方法为：
> 对于一个二叉树来说，除了度为 0 的叶子结点和度为 2 的结点，剩下的就是度为 1 的结点（设为 n_1），那么总结点 $n = n_0 + n_1 + n_2$ 。
> 同时，对于每一个结点来说都是由其父结点分支表示的，假设树中分枝数为 B，那么总结点数 $n = B + 1$ 。而分枝数是可以通过 n_1 和 n_2 表示的: $B = n_1 + 2 * n_2$ 。
> 所以，n 用另外一种方式表示为: $n=n_1+2*n_2+1$ 。
> 两种方式得到的 n 值组成一个方程组，就可以得出 $n_0 = n_2 + 1$ 。

完全二叉树特有的性质

n 个结点的完全二叉树的深度为 $[\log_2 n]+1$ 。

> $[\log_2 n]$表示取小于$\log_2 n$的最大整数。例如，$[\log_2 4] = 2$,而 $[\log_2 5]$ 结果也是 2。

对于任意一个完全二叉树来说，将含有的结点按照层次从左到右依次标号（如图 2（A）），对于任意一个结点 i，有以下几个结论：

- 当 i > 1时，父亲结点为结点 [i / 2] 。（ i = 1时，表示的是根结点，无父亲结点）
- 如果 2*i > n，则结点 i 肯定没有左孩子（为叶子结点）；否则其左孩子是结点 2*i 。
- 如果 2*i +1 > n，则结点 i 肯定没有右孩子；否则右孩子是结点 2*i +1 。

二叉树和完全二叉树的各自所特有的性质，需要熟记，在对其进行存储以及利用二叉树解决问题时，会经常用到。

二叉树的存储结构

二叉树有两种存储结构：顺序存储结构和链式存储结构。

顺序存储

借用数组将二叉树中的数据元素存储起来。此方式只适用于完全二叉树，如果想存储普通二叉树，需要将普通二叉树转化为完全二叉树。

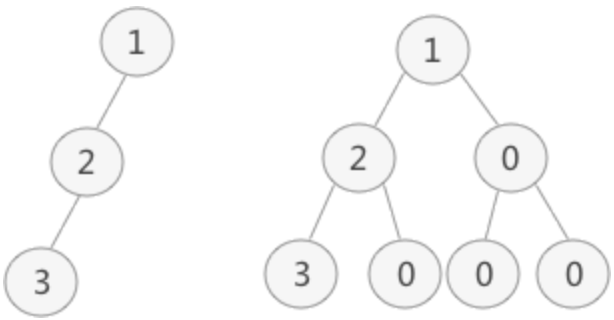
使用数组存储完全二叉树时，从数组的起始地址开始，按层次顺序从左往右依次存储完全二叉树中的结点。当提

取时，根据完全二叉树的第 2 条性质，可以将二叉树进行还原。

例如，存储图 2（A）时，数组中存储为：

根据完全二叉树的第 2 条性质就可以根据数组中的数据重新搭建二叉树的结构。

如果普通二叉树也采取顺序存储的方式，就需要将其转化成完全二叉树，然后再存储，例如：



转化前　　转化后

图 3 普通二叉树转完全二叉树

图 3 中，转化后的二叉树中，数据元素 0 表示此位置没有数据。将转化后的完全二叉树按照层次并从左到右的次序存储到数组中：

由此可见。深度为 K 且只有 K 个结点的单支树（树中不存在度为 2 的结点），需要 `2K-1` 的数组空间，浪费存储空间。所以，顺序存储方式更适用于完全二叉树。

链式存储

采用链式存储结构存储二叉树，就非常容易理解了。根据每个结点的结构，至少需要 3 部分组成：

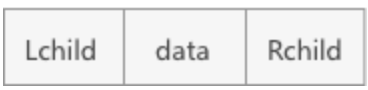


图5 二叉链表结点构成

图 5 中，`Lchild` 代表指向左孩子的指针域；`data` 为数据域；`Rchild` 代表指向右孩子的指针域。使用此种结

点构建的二叉树称为“二叉链表”。

结点结构代码表示：

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

如果程序中需要频繁地访问结点的父结点，就可以使用下面这种结点结构：

图 6 三叉链表结点构成

图 6 中，`Lchild` 指向左孩子；`Rchild` 指向右孩子；`data` 为数据域；`parent` 指向父结点。使用这种结构的结点创建的树称为“三叉链表”。

结点结构代码表示：

```
typedef struct BiTNode{
    TElemType data;//数据域
    struct BiTNode *lchild,*rchild;//左右孩子指针
    struct BiTNode *parent;
}BiTNode,*BiTree;
```

例如，分别用两种结点创建图 3 中的单支树：

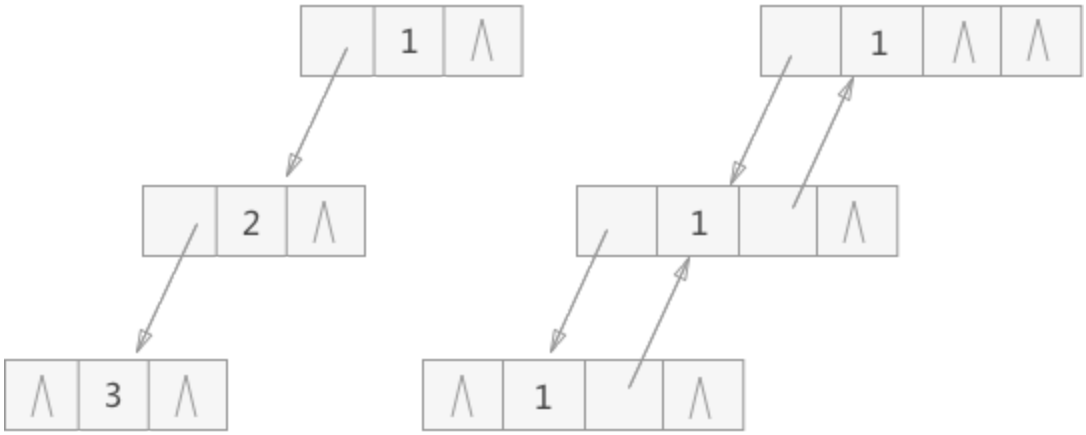


图7 单支树示意图

实现代码（以二叉链表为例）

```
#include <stdio.h>
#include <stdlib.h>
#define TElemType int
```

```
04.
05.  typedef struct BiTNode{
06.      TElemType data;//数据域
07.      struct BiTNode *lchild,*rchild;//左右孩子指针
08.  }BiTNode,*BiTree;
09.
10.  void CreateBiTree(BiTree *T){
11.      *T=(BiTNode*)malloc(sizeof(BiTNode));
12.      (*T)->data=1;
13.      (*T)->lchild=(BiTNode*)malloc(sizeof(BiTNode));
14.      (*T)->rchild=NULL;
15.      (*T)->lchild->data=2;
16.      (*T)->lchild->lchild=(BiTNode*)malloc(sizeof(BiTNode));
17.      (*T)->lchild->rchild=NULL;
18.      (*T)->lchild->lchild->data=3;
19.      (*T)->lchild->lchild->lchild=NULL;
20.      (*T)->lchild->lchild->rchild=NULL;
21.  }
22.  int main() {
23.      BiTree Tree;
24.      CreateBiTree(&Tree);
25.      printf("%d",Tree->lchild->lchild->data);
26.      return 0;
27.  }
```

运行结果：

```
3
```

总结

对于二叉树和完全二叉树的性质，需要学员在理解的情况下进行记忆。有关二叉树存储结构的选择，以及结点结构的选择，要视情况而定，基本上遵循以下两个原则：

1. 如果是普通二叉树，用链式存储结构；如果是完全二叉树，用顺序存储结构。
2. 如果问题中涉及到要访问某结点的父结点，就建立三叉链表；反之，使用二叉链表即可解决问题。