# 41树的存储结构

## 树的存储结构

不好意思哈，这节课又需要大家搞脑子了。对于知识，你理解的越多，需要记住的就越少！

上节课我们简单的介绍了树结构的强大，这节课我们来关心一下如何在内存中安排树这种结构的存放。

说到存储结构，就会想到我们前面章节讲过的顺序存储和链式存储两种基本结构。

对于线性表来说，很直观就可以理解，但对于树这种一对多的结构，我们应该怎么办呢？

要存储树，简单的顺序存储结构和链式存储结构是不能滴！不过如果充分利用它们各自的特点，完全可以间接地来实现。

大家先思考下，如果你是总工程师，让你来设计和规划，你有多少种方法可以实现对树结构的存放？

当然你要考虑到双亲、孩子、兄弟之间的关系。

小甲鱼这里要介绍三种不同的表示法：双亲表示法、孩子表示法、孩子兄弟表示法。

用途：编译语法树 数据库索引 网络管理 操作系统文件系统管理 …

## 双亲表示法

* 双亲表示法，言外之意就是以双亲作为索引的关键词的一种存储方式。
* 我们假设以一组连续空间存储树的结点，同时在每个结点中，附设一个指示其双亲结点在数组中位置的元素。
* 也就是说，每个结点除了知道自己是谁之外，还知道它的粑粑妈妈在哪里。
* 那么我们可以做如下定义：

parent.c

// 树的双亲表示法结点结构定义

#define MAX\_TREE\_SIZE 100

typedef int ElemType;

typedef struct PTNode

{

ElemType data; // 结点数据

int parent; // 双亲位置

}PTNode;

typedef struct

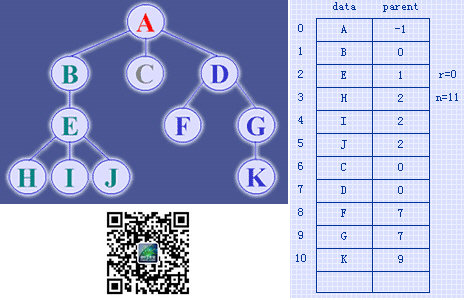
{

PTNode nodes[MAX\_TREE\_SIZE];

int r; // 根的位置

int n; // 结点数目

}PTree;



这样的存储结构，我们可以根据某结点的parent指针找到它的双亲结点，所用的时间复杂度是O(1)，索引到parent的值为-1时，表示找到了树结点的根。

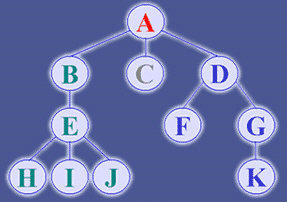
可是，如果我们要知道某结点的孩子是什么？那么不好意思，请遍历整个树结构。

这真是麻烦，能不能改进一下呢？

鱼油们怎么看？

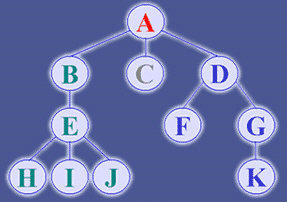
电视机前的小盆友们怎么看？

当然可以，我们只需要稍微改变一下结构即可：



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **data** | **parent** | **child1** | **child2** |
| **0** | A | -1 | 1 | 6 |
| **1** | B | 0 | 2 | -1 |
| **2** | E | 1 | 3 | 4 |
| **3** | H | 2 | -1 | -1 |
| **4** | I | 2 | -1 | -1 |
| **5** | J | 2 | -1 | -1 |
| **6** | C | 0 | -1 | -1 |
| **7** | D | 0 | 8 | 9 |
| **8** | F | 7 | -1 | -1 |
| **9** | G | 7 | 10 | -1 |
| **10** | K | 9 | -1 | -1 |

那现在我们又比较关心它们兄弟之间的关系呢？



|  |  |  |  |
| --- | --- | --- | --- |
|  | **data** | **parent** | **rightSib** |
| **0** | A | -1 | -1 |
| **1** | B | 0 | 6 |
| **2** | E | 1 | -1 |
| **3** | H | 2 | 4 |
| **4** | I | 2 | 5 |
| **5** | J | 2 | -1 |
| **6** | C | 0 | 7 |
| **7** | D | 0 | -1 |
| **8** | F | 7 | 9 |
| **9** | G | 7 | -1 |
| **10** | K | 9 | -1 |

存储结构的设计是一个非常灵活的过程，只要你愿意，你可以设计出任何你想要的奇葩！

一个存储结构设计得是否合理，取决于基于该存储结构的运算是否适合、是否方便，时间复杂度好不好等等。

不要拘泥于你所学过的有限的数据类型，要把思维放开些，放开些，放开些！

当你的思维放开了，世界就变小了！

当你的思维放开了，知识就变少了！

# 42树的存储结构2

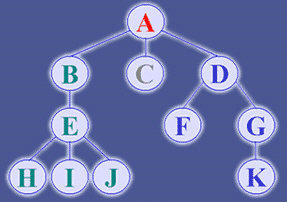
## 孩子表示法

我们这次换个角度来考虑，由于树中每个结点可能有多棵子树，可以考虑用多重链表来实现。

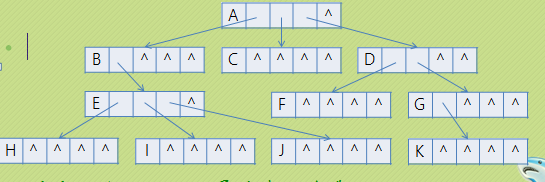
就像我们虽然有计划生育，但我们还是无法确保每个家庭只养育一个孩子的冲动，那么对于子树的不确定性也是如此。

1. 右图中，树的度为（ 3 ）

2. 如果我们用“孩子表示法”，聪明的鱼油可以想出多少种可行方案？

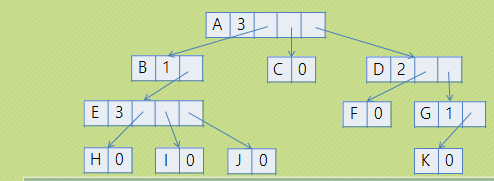


这里我们不限制大家的答案，小甲鱼给出三个参考的方案，先来看下方案一：根据树的度，声明足够空间存放子树指针的结点。



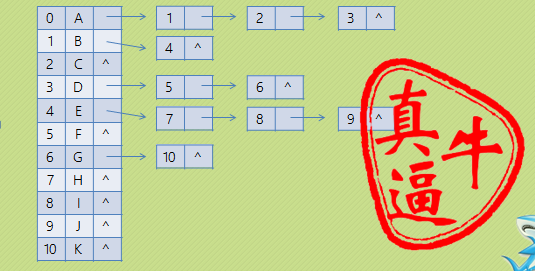
缺点十分明显，就是造成了浪费！

针对方案一的缺点，我们有了方案二(me：记录子节点的个数)：



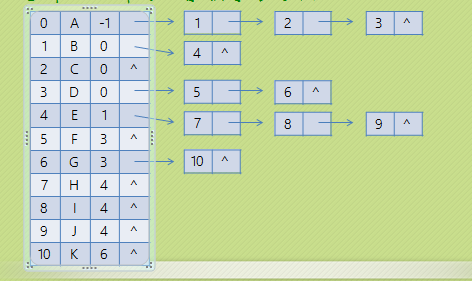
这样我们就克服了浪费这个概念，我们从此走上了节俭的社会主义道路！但每个结点的度的值不同，初始化和维护起来难度巨大吧？

难倒没有更好的了？请看下边架构：



双亲孩子表示法

那只找到孩子找不到双亲貌似还不够完善，那么我们合并上一讲的双亲孩子表示法（me：在上面的基础上，存储父节点的指针）：



Parent\_child.c

#define MAX\_TREE\_SIZE 100

typedef char ElemType;

// 孩子结点

typedef struct CTNode

{

int child; // 孩子结点的下标

struct CTNode \*next; // 指向下一个孩子结点的指针

} \*ChildPtr;

// 表头结构

typedef struct

{

ElemType data; // 存放在树中的结点的数据

int parent; // 存放双亲的下标

ChildPtr firstchild; // 指向第一个孩子的指针

} CTBox;

// 树结构

typedef struct

{

CTBox nodes[MAX\_TREE\_SIZE]; // 结点数组

int r, n;

}

# 43二叉树

## 二叉树的定义

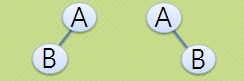
世上树有万千种，唯有二叉课上讲。这里的二叉是二叉树，因为二叉树使用的范围最广，最具有代表意义，因此我们重点讨论二叉树。

二叉树（Binary Tree）是n（n>=0）个结点的有限集合，该集合或者为空集（空二叉树），或者由一个根结点和两棵互不相交的、分别称为根结点的左子树和右子树的二叉树组成。

这个定义显然是递归形式的，所以咱看上去有点晕，因为自古有“神使用递归，人使用迭代！

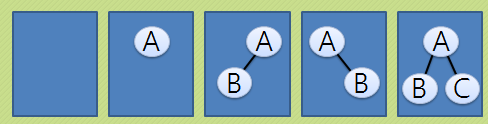
## 二叉树的特点

* 每个结点最多有两棵子树，所以二叉树中不存在度大于2的结点。（注意：不是都需要两棵子树，而是最多可以是两棵，没有子树或者有一棵子树也都是可以的。）
* 左子树和右子树是有顺序的，次序不能颠倒。
* 即使树中某结点只有一棵子树，也要区分它是左子树还是右子树，下面是完全不同的二叉树：



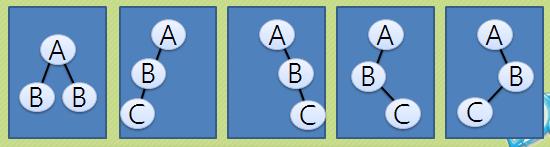
## 二叉树的五种基本形态

* 空二叉树
* 只有一个根结点
* 根结点只有左子树
* 根结点只有右子树
* 根结点既有左子树又有右子树



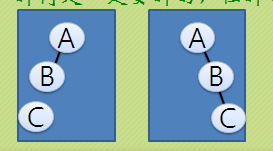
## 二叉树很二

* 小甲鱼为什么这么说呢？因为若只从形态上来考虑，拥有三个结点的普通树只有两种情况：两层或者三层。
* 但对于很二的二叉树来说，由于要区分左右，所以就演变成五种形态：



## 特殊二叉树

* 因为他很二，所以他也很特殊。小甲鱼接下来再介绍一下一些特殊的二叉树，虽然暂时你可能不能理解它们的用处，但我们有必要先了解一下。
* 斜树
  + 顾名思义，斜树是一定要斜的，但斜也要斜得有范儿，例如：



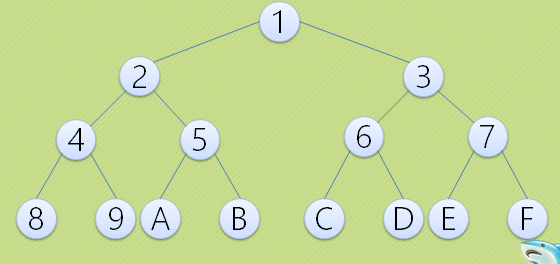
## 满二叉树

坡坡有云：“人有悲欢离合，月有阴晴圆缺，此事古难全。但愿人长久，千里共长娟。”意思就是说完美的那是理想，不完美的才是人生。

但是对于二叉树来说，是否存在完美呢？有滴，那就是满二叉树啦。

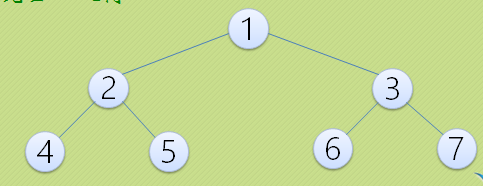
在一棵二叉树中，如果所有分支结点都存在左子树和右子树，并且所有叶子都在同一层上，这样的二叉树称为满二叉树。

不理解？No pic you say a J8....



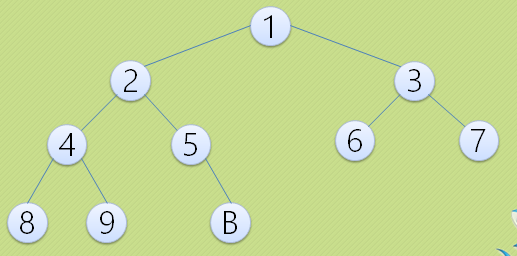
## 满二叉树的特点有：

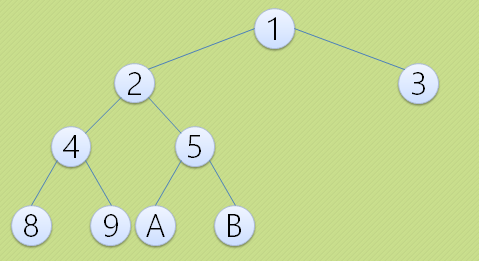
* + 叶子只能出现在最下一层。
  + 非叶子结点的度一定是2。
  + 在同样深度的二叉树中，满二叉树的结点个数一定最多，同时叶子也是最多。
* 满二叉树和完全二叉树历年都是一个重大考点，因为考生很容易混淆两者，但如果只是为了深入学习编程，那么只需要理解即可。记得我们曾说过：理解越多，需要记住的就越少！
* 对一棵具有n个结点的二叉树按层序编号，如果编号为i(1<=i<=n)的结点与同样深度的满二叉树中编号为i的结点位置完全相同，则这棵二叉树称为完全二叉树。

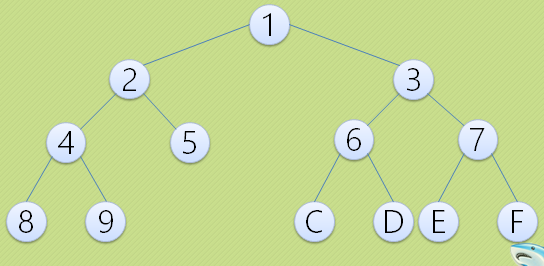


## 完全二叉树的特点有：

* + 叶子结点只能出现在最下**两**层。
  + 最下层的叶子一定集中在左部连续位置。
  + 倒数第二层，若有叶子结点，一定都在右部连续位置。
  + 如果结点度为1，则该结点只有左孩子。
  + 同样结点树的二叉树，完全二叉树的深度最小。
* 注意：满二叉树一定是完全二叉树，但完全二叉树不一定是满二叉树。
* 以下这些都**不是**完全二叉树：







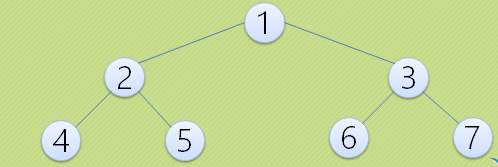
## 二叉树的性质

* 作为课后扩展作业，在不谷哥无度娘的条件下，自己完成：通过观察二叉树的图片，总结出你认为的二叉树应该具有的性质。例如：在二叉树的第i层上至多有2^(i-1)个结点。

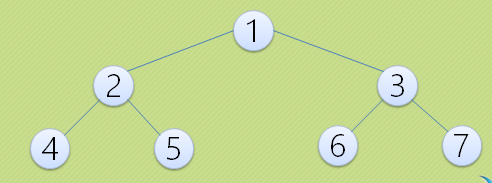
# 44二叉树2

## 二叉树的性质

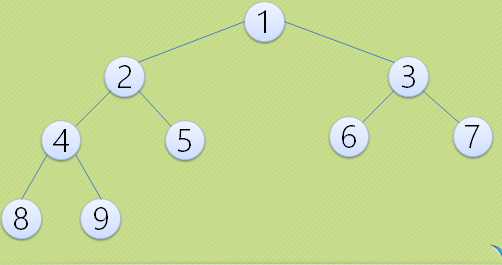
* 二叉树的性质一：在二叉树的第i层上至多有2^(i-1)个结点(i>=1)
  + 这个性质其实很好记忆，考试的时候懂得画出二叉树的图便可以推出



* 二叉树的性质二：深度为k的二叉树至多有2^k-1个结点(k>=1)(me：满二叉树为2^k-1)
  + 这里一定要看清楚哦，是2^k再-1，老方法理解：



* 二叉树的性质三：对任何一棵二叉树T，如果其终端结点数为n0，度为2的结点数为n2，则n0=n2+1
  + 这个就比较困难了，需要推导获得
  + 首先我们再假设度为1的结点数为n1，则二叉树T的结点总数n=n0+n1+n2
  + 其次我们发现连接数总是等于总结点数n-1，并且等于n1+2\*n2
  + 所以n-1=n1+2\*n2
  + 所以n0+n1+n2-1=n1+n2+n2
  + 最后n0=n2+1
* 二叉树的性质四：具有n个结点的完全二叉树的深度为⌊log₂n⌋+1
  + 涉及到一些数学知识了，专门针对要考试的学生，只为学编程的路过即可
  + 由满二叉树的定义结合性质二我们知道，深度为k的满二叉树的结点树n一定是2^k-1
  + 那么对于满二叉树我们可以通过n=2^k-1倒推得到满二叉树的深度为k=log₂(n+1)
  + 由于完全二叉树前边我们已经提到，它的叶子结点只会出现在最下面的两层，我们可以同样如下推导
  + 那么对于倒数第二层的满二叉树我们同样很容易回推出它的结点数为n=2^(k-1)-1
  + 所以完全二叉树的结点数的取值范围是：2^(k-1)-1 < n <= 2^k-1
  + 由于n是整数，n <= 2^k-1可以看成n < 2^k
  + 同理2^(k-1)-1 < n可以看成2^(k-1) <= n
  + 所以2^(k-1) <= n < 2^k
  + 不等式两边同时取对数，得到k-1<=log₂n<k
  + 由于k是深度，必须取整，所以k=⌊log₂n⌋+1
* 二叉树的性质五：如果对一棵有n个结点的完全二叉树(其深度为⌊log₂n⌋+1)的结点按层序编号，对任一结点i(1<=i<=n)有以下性质：
  + 如果i = 1，则结点 i 是二叉树的根，无双亲；如果i > 1，则其双亲是结点⌊i/2⌋
  + 如果2i > n，则结点 i 无做左孩子(结点 i 为叶子结点)；否则其左孩子是结点2i
  + 如果2i+1 > n，则结点 i 无右孩子；否则其右孩子是结点2i+1
* 文字描述太折腾，直接看图听小甲鱼分析：



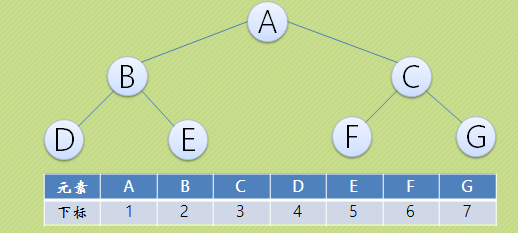
# 45二叉树的存数结构

## 二叉树的存储结构

* 树结构在计算机中的存储形式很多，可谓天马行空任你创造，只要能够按照要求完成任务即可。
* 在前边的演示中，我们发觉很难单单只用顺序存储结构或者链式存储结构来存放。
* 但是二叉树是一种特殊的树，由于它的特殊性，使得用顺序存储结构或链式存储结构都能够简单实现。
* 二叉树的顺序存储结构就是用一维数组存储二叉树中的各个结点，并且结点的存储位置能体现结点之间的逻辑关系。

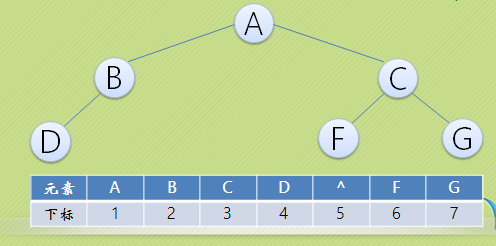
这一点儿都不难：No pic you say a J8!

## 二叉树的顺序存储结构

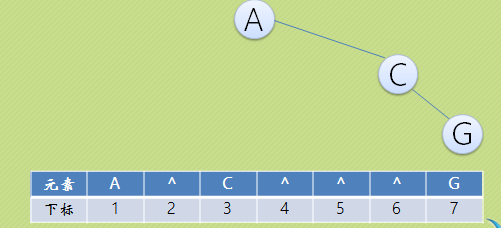


这下看出完全二叉树的优越性来了吧？由于他的严格定义，在数组直接能表现出逻辑结构。

* 当然对于一般的二叉树，尽管层序编号不能反映逻辑关系，但是也可以按照完全二叉树编号方式修改一下，把不存在的结点用“^”代替即可。



* 但是考虑到一种极端的情况，回顾一下斜树，如果是一个又斜树，那么会变成这样。。。。。。



## 二叉链表

* 既然顺序存储方式的适用性不强，那么我们就要考虑链式存储结构啦。二叉树的存储按照国际惯例来说一般也是采用链式存储结构。
* 二叉树每个结点最多有两个孩子，所以为它设计一个数据域和两个指针域是比较自然的想法，我们称这样的链表叫做二叉链表。

|  |  |  |
| --- | --- | --- |
| **lchild** | **data** | **rchild** |

* 以下是我们的二叉链表的结点结构定义代码：

typedef struct BiTNode

{

ElemType data;

struct BiTNode \*lchild, \*rchild;

} BiTNode, \*BiTree;

