摸一摸数据结构（树）

作者：stillwarter
原文链接：https://ld246.com/article/1630664226772
来源网站：链滴

树型结构是一类重要的非线性数据结构。以树与二叉树最常用，直观上树是分支关系定义的层次结构并且在计算机领域也有广泛应用，比如编译程序和数据库系统。

树的定义以及基本用语

定义

树是 n(n>=0) 的有限集。在任意一棵树非空树中

1. 有且仅有一个特定根（root）的结点

2. 当 n>1 时，其余结点可分为 m(m>0) 个互不相交的有限集，其每一个结合本身又是一棵树我们称为**子树**

有点抽象，我用表再尝试叙述一下

根结点 root	**1**
表一	1.1（2）
	1.2（3）
	1.3（4）
	1.11（5）
	1.12（6）
	1.1（7）

1	**2**	**3**	**4**	**5**	**6**	**7**
1	1.1	1.2	1.3	1.11	1.12	1.31

左边是树的层次表示，1 表示根结点，1.1 表示其父结点为 1；1.11 表示其父节点为 1.1；这样就表一个树的层次结构（可以画出树状图更为直观）。

右边是树的一维数组表示，也是计算机存储树的一种方式。在猫猫的理解里，数据的存储无非是连续存储或者链式存储。所以树是一种逻辑上的结构，我们通过赋予数据一些联系来实现我们想要的功能。

基本术语

1. 结点：一个数据元素及其若干指向其子树的分支。

2. 结点的度，树的度：结点所拥有的子树的棵树称为结点的度；树中结点度最大值称为树的度。

以表一为例，结点 1（根结点）的度为 3（其子树为 1.1,1.2,1.3）；同时这个树的结点也是 3。

3. 叶子结点，非叶子结点：度为 0，称 **叶子结点或终端结点**；反之为非...或**分支结点**；除了根节外，分支结点又称为内部结点。

以表一为例，结点 1.11，1.12 是叶子结点（终端结点），1.1 为非叶子结点（分支结点，有分支，非结点也是内部结点）。

4. 孩子兄弟和双亲结点：

一个结点的子树的根称为该结点的**孩子结点**或子结点；响应的该结点是其孩子结点的**双亲结点**或父结。同一双亲结点所有子结点互称为**兄弟结点**。

5. 层次堂兄弟结点：

规定树中根节点的层次为 1，其余结点的层次等于其双亲结点的层次加一。若某结点再第 n 层，则其

结点再第 n+1 层，双亲结点再同一层上的所有结点互称为**堂兄弟结点**。

6. 结点的层次路径，祖先和子孙：

从根结点开始，到达某结点 p 所经过的所有结点成为结点 p 的**层次路径**（有且只有一条）。

结点 p 的层次路径上的所有结点（p 除外）称为 p 的祖先。

以某一结点为根的子树中的任意结点称为该结点的子孙结点。

7. 树的深度：树中结点的最大层次值，又称为树的高，表一的树的深度为 3。

8. 有序树和无序树：对于一棵树，若其中每一个结点的子树（若有）具有一定的次序，则该树称有序树，反之无序树。（二叉树就是有序树，分左右）

9. 补充-高度的另一定义：

    1. 深度：层次数，从上往下
    2. 高度：规定叶子结点的高度为 1，其双亲结点的高度等于它的高度加一，从下往上。
    3. 树的高度：等于根结点的高度，即根结点所有子女高度的最大值加一。
    4. 结点的深度和结点的高度是不同的。
    5. 树的高度和深度是相等的。

10. 森林：是 m（m 大于 0）棵树互不相交的树的集合。若将一棵树的根结点删除，剩余的子树构成了森林。

11. 树的表示形式（图示请自行百度）

    1. 倒悬树。最常用的表现形式（树状图）
    2. 嵌套集合。是一些集合的集体，对任何两个集合，或者不相交，或者一个集合包含另一个集。（类似我们离散中的集合图状表示）
    3. 广义表形式（（A(b,c),D(e,f)））
    4. 凹入法表示形式（类似数的编目）

树的概念与术语有点多，这也侧面说明其重要性与复杂性。接下来根据这些概念思考

1. 所有结点的度之和是多少?

以表一为例，根结点 1 度是 3，1.1 的度是 2,1.2 没有度，1.3 度为 1。其度之和为 3+2+1=6，其结有 7 个。那么结点为 n 个呢?

2. 度为 m 的数中第 i 层至多有多少个结点（i>=1）？

3. 高度为 h 的 m 叉树至多有多少结点?

4. 具有 n 个结点的 m 叉树的最小高度?

好吧猫猫无法将详细的证明过程写出来，这些问题就是树的性质

树的性质

1. 树中的结点数等于所有结点的度树之和加 1。
2. 度为 m 的树中第 i 层上至多有 m^{i-1} 个结点(i>=1)。
3. 高度为 h 的 m 叉树至多有 $(m^h-1)/(m-1)$ 个结点。
4. 具有 n 个结点的 m 叉树的最小高度为 $[log_m(n(m-1)+1)]$。

二叉树的概念以及实现

二叉树的概念

定义

二叉树（binary tree）是另一种树型结构，其特点是每个结点至多有 2 个子树（即二叉树的度不能大 2），并且二叉树的子树有左右之分，其次序不能任意颠倒。

这里我用表来表示一个简单的二叉树

表二	**1**	** **
	1.1	1.2
	1.11,1.12	1.21,1.22

由二叉树的定义可知，有五种基本形式是满足二叉树的

1. 空树，空树是树，也是二叉树。
2. 只有根结点的树。
3. 以表二为例，若一棵树只有结点 1 和 1.1，那么这是右子树为空的二叉树。
4. 以表二为例，若一棵树只有结点 1 和 1.2，那么这是左子树为空的二叉树。
5. 以表二为例，若一棵树只有结点 1，1.1 和 1.2，那么是左右子树均有的二叉树。

除此之外，二叉树也有满二叉树和完全二叉树的说法。

1. 满二叉树：一棵深度为 k 且有 $2^{k}-1$ 个结点（最多结点）的二叉树
2. 完全二叉树：若深度为 k 由 n 个结点的二叉树，当且仅当其每一个结点都与深度为 k 的满二叉树编号从 1~n 一一对应，则为完全二叉树。（完全二叉树就是其结点序号对应满二叉树是一样的。）

二叉树的性质

1. 非空二叉树中第 i 层至多有 2^{i-1} 个结点（i>=1）。用数学归纳法证明。

假设我们有一个 i（大于 1） 层的二叉树，那么在这层上至多有 $2^{(i-1)}$。

2. 深度为 k 的二叉树最多有 2^k-1 个结点（k 大于等于 1）

由性质一可知，第 i 层的结点数至多有 2^{(i-1)}。

所以总的结点数：$2^0+2^1+.....2^{k-1}=2^k-1$。

3. 对任何一棵二叉树，其叶子结点的个数为 n_0，度为 2 的结点数为 n_2，则 n_0=n_2+1

理解：

假设一个二叉树叶子结点数为 n_0（度为 0），度为 1 的结点数为 n_1，度为 2 的结点数为 n_2；

所以我们能推出结点总数：N=n_0+n_1+n_2；二叉树有分支数 B（比如根结点后面跟的线就可以看分支，实际上就是结点的度）；

分支数 B 的和等于度之和（不局限与二叉树）；

由于树的根结点是没有前驱的，所以一个二叉树的结点总数与分支数存在关系 B=N-1；

由此可以推出 n_0+n_1+n_2-1=2n_2+n_1；所以得到结论 n_0=n_2+1。

4. n 个结点的完全二叉树的深度为 log2^n+1（向下取整）

假设完全二叉树的深度为 k，则根据性质 2 有 $2^{k-1}-1<n<=2^k-1或2^{k-1}<=n<2^k$，取对就可以得到性质 4。

log_{2^{n+1}} 也对

5. 对一棵有 n 个结点的完全二叉树按层序自左到右进行编号，则对于编号为 i（1<=i<=n）的结：

1. 若 i 等于 1：则是二叉树的根。若 i》1，则其双亲结点编号是 i。
2. 若 2i》n：则结点 i 是叶子结点；若不大于，则 i 结点的孩子结点编号为 2i。
3. 若 2i+1》n：则结点 i 无右孩子；否则其右孩子编号为 2i+1

证明是数学归纳法。

理解二叉树的性质对于我们如何实现有指导意义。

二叉树的实现

ps：因为二叉树的顺序没有链式常用，所以我们不做测试，而链式有时间补（鸽）

顺序二叉树结构

```
#include <stdio.h>
#include <math.h>
#include "Status.c"
#include "Scanf.c"

#define maxtreesize 100

typedef char telemtypesq;//假设顺序二叉树元素均为字符
typedef telemtypesq sqbitree[maxtreesize];//0号单元存储根结点

typedef{
    int level;//结点所在层
    int order;//结点再本层序号
}position;
```

初始化，清空与销毁

```
void initbitreesq(sqbitree T){
    int i;
    for(i=0;i<maxtreesize;i++) T[i]='\0';//这里用空字符填充
};

void clearbitreesq(sqbitree T){
    int i;
    for(i=0;i<maxtreesize;i++) T[i]='\0';//这里用空字符填充
};
void destroybitreesq(sqbitree T){
    //无，顺序二叉树销毁操作无
```

```
};
```

其函数内代码都是一样的，因为我们初始化一个顺序二叉树在计算机里是用数组，并且赋值都是字符的空字符，而清空数组元素恰好是对数组数据清空。

判空

```
Status bitreeemptysq(sqbitree T){
    return T[0]=='\0' ? TRUE : FALSE;
};
```

通过 T[0]就可以知道二叉树是否为空

创建

```
Status createbitreesq(FILE *fp,sqbitree T){
    char ch;
    int i=0;
    while(Scanf(fp,"%c",&ch)==1&&ch!='\n'){
        if(ch=='^') T[i++]='\0';
        else T[i++]=ch;
    }
    return OK;
};//按层序序列构造二叉树
Status createbitreepresq(FILE *fp,sqbitree T,int i){
    char ch;
    Scanf(fp,"%c",&ch);
    if(ch=='^') T[i]='\0';
    else{
        T[i]=ch;
        createbitreepresq(fp,T,2*i+1);
        createbitreepresq(fp,T,2*i+2);
    }
};//按先序序列构造二叉树
```

层序序列构造：这里是用 scanf 函数来扫描文本填充到数组（二叉树）里面

先序序列构造：扫描文本后将文本中的 ^ 替换为空字符（空格），因为在 c 里面读取到空字符会停扫描。

树的长度，深度和根结点

```
int bitreelensq(sqbitree T){
    int len;
    for(len=maxtreesize;len-1>=0;lem--){
        if(T[len-1]!='\0') break;
    }
    return len;
};

int bitreedepsq(sqbitree T){
    int level=0;
    while((int)pow(2.level)-1<bitreelensq(T)) level++;
```

```
    return level;
};

Status rootsq(sqbitree T,telemtypesq *e){
    if(bitreeemptysq(T)) return ERROR;
    *e=T[1];
    return OK;
};
```

树的长度并不是树的规模，对树进行遍历，若对应下标的值不为空，则跳出循环输出长度值。

树的深度：根据二叉树的性质（深度为 k 的二叉树最多有 2^k-1 个结点（k 大于等于 1）），我可以通过比较结点数来得到对应的深度。

树的根结点：判断树是否为空，然后直接输出根结点。

返回结点值

```
telemtypesq valuesq(sqbitree T,position s){
    int i=(int)pow(2,s.level-1)+s.order-2;
    return T[i];
};//返回树中某位置的结点值
```

传入树 T 与结点 S，把其树位次信息转为数组信息。，然后返回对应值就可以了。

为某位置结点赋值

```
Status assignsq(sqbitree T,position s,telemtypesq value){
    int i=(int)pow(2,s.level-1)+s.order-2;

    if(value=='\0' && (T[2*i+1]!='\0' ||T(2*i+2)!='\0'))
        return ERROR;
    else if(value!='\0' && T[(i+1)/2-1]=='\0')
        return ERROR;
    else
        T[i]=value;
    return OK;
};//为树中某位置的结点赋值
```

传入二叉树 T 和结点 s，用 value 赋值。

首先转为数组序列表示，然后判断，若该结点的子树不为空，那么不能替换；若替换值不是空，且其结点不存在，则不能替换（因为替换后一个变成一个没有父结点的结点）。

判断之后进行替换。

输出对应结点数值的父母结点，兄弟结点，孩子结点

```
telemtypesq parentsq(sqbitree T,telemtypesq e){
    int i;
    if(T[0]!='\0'){
        for(i=0;i<maxtreesize;i++){
            if(T[i]==e) return T[(i+1/2-1)];
```

```
        }
    }
    return '\0';
};

telemtypesq lchildsq(sqbitree T,telemtypesq e){
    int i;
    if(T[0]=='\0') return '\0';

    for(i=0;i<maxtreesize;i++){
        if(T[i]==e) return T[2*i+1];
    }
    return '\0';
};

telemtypesq rchildsq(sqbitree T,telemtypesq e){
    int i;
    if(T[0]=='\0') return '\0';

    for(i=0;i<maxtreesize;i++){
        if(T[i]==e) return T[2*i+1];
    }
    return '\0';
};

telemtypesq lsibsq(sqbitree T,telemtypesq e){
    int i;
    if(T[0]=='\0') return '\0';

    for(i=0;i<maxtreesize;i++){
        if(i%2==0 && T[i]==e) return T[i-1];
    }
    return '\0';
};

telemtypesq rsibsq(sqbitree T,telemtypesq e){
    int i;
    if(T[0]=='\0') return '\0';

    for(i=0;i<maxtreesize;i++){
        if(i%2==0 && T[i]==e) return T[i+1];
    }
    return '\0';
};
```

算法思路都是先遍历树，找到对应结点的下标值，然后根据二叉树的特性去输出。

遍历二叉树

```
void levelordertrasq(sqbitree T,void(Visit)(telemtypesq),int i){
    int i;
    int len=bitreelensq(T);

    for(i=0;i<len;i++){
```

```
        if(T[i]!='\0') Visit(T[i]);
    }
};

void preordertrasq(sqbitree T,void(Visit)(telemtypesq),int i){
    if(T[i]!=0){
        Visit(T[i]);
        preordertrasq(T,Visit,2*i+1);
        preordertrasq(T,Visit,2*i+2);
    }
};

void inordertrasq(sqbitree T,void(Visit)(telemtypesq),int i){
    if(T[i]!=0){
        preordertrasq(T,Visit,2*i+1);
        Visit(T[i]);
        preordertrasq(T,Visit,2*i+2);
    }
};

void postordertrasq(sqbitree T,void(Visit)(telemtypesq),int i){
    if(T[i]!=0){
        preordertrasq(T,Visit,2*i+1);
        preordertrasq(T,Visit,2*i+2);
        Visit(T[i]);
    }
};
```

遍历二叉树有好几种方法

1. 层序遍历：我们通过树的长度去访问树中每一个数据
2. 先序遍历：通过根左右的顺序去访问
3. 中序遍历：通过左根右的顺序去访问
4. 后序遍历：通过左右根的顺序去访问

数据的输出

```
void printsq(sqbitreesq T){
    int i,j,level;
    char tmp[maxtreesize][maxtreesize]={};
    level=bitreedepsq(T);

    for(i=1;i<level;i++)
        for(j=1;j<=(int)pow(2,i-1);i++)
            tmp[i-1][(int)pow(2,level-1)+(j-1)*(int)pow(2.level-i+1)-1]=T[(int)pow(2,i-1)-1+j-1;

    for(i=0;i<level;i++){
        for(j=0;j<2*(int)pow(2,level-1)-1;j++){
            if(tmp[i][j]!='\0') printf("%c",tmp[i][j]);
            else printf(" ");
        }
        printf("\n");
```

```
    }
}
```

传入树 T。函数内设变量存储树的深度，根据深度对应的结点值去填充设置的二位数组 tmp

（其中的数据关系式猫猫不想看了，头大）

其他

其实除了这个简单的顺序二叉树，还有许多其他的不同的结构的二叉树。比如链式存储的二叉树，其一般用链式二叉树更多，因为可以保证空间不会过多浪费。

还有树，森林的实现，都是坑（鸽）。

树与二叉树的应用

1. 二叉查找树：也叫二叉排序树，是根据二叉树的特性进行数据查找的一种算法。

2. 平衡二叉树：为了避免树的高度增长，降低二叉排序树的性能，我们要保证左，右子树高度差绝对不超过 1，将这样的二叉树称为平衡二叉树。

3. 哈夫曼树：在许多应用中，树中结点被赋予一个表示某种意义的数值，称为权。从树的根到任意结的路径长度（经过的边数）与该结点上权值的乘积，称为该结点的带权路径长度。

在含有 n 个带权也结点的二叉树中，其带权路径长度最小的二叉树称为哈夫曼树。

鸽

思考

1. 对于任意一棵高度为 5 且 10 个结点的二叉树，若用顺序存储方式，每个结点占据一个存储单元，其存储单元数量至少是多少?

2. 一个具有 1025 个结点的二叉树的高度为多少?

ps: 摸一摸数据结构(目录)