树的应用

二叉树、堆、并查集

树及二叉树

树结构

线性结构：一对一

树结构：一对多

一个结点可以有多个子结点，每个结点有唯一的父结点（根结点例外）。

树的定义（递归）：

树是由n (n >=0)个结点(node)组成的有限集合。

如果n = 0，称为空树； 如果n > 0，则:

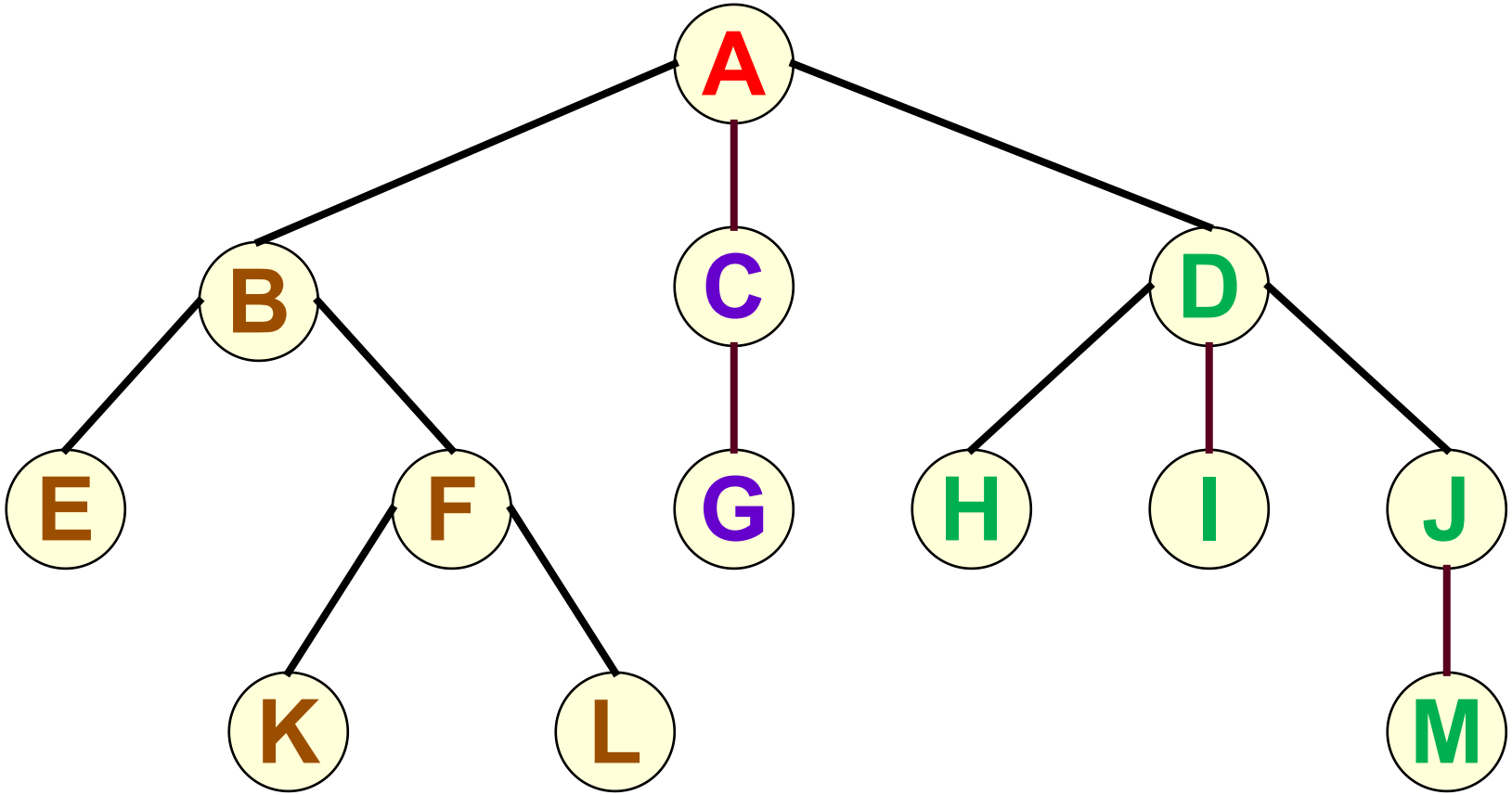
有唯一的一个结点称之为根(root)的结点，它只可以有后继，但没有前驱；

除根结点以外的其它结点划分为m (m >= 0)个互不相交的有限集合$T_0, T_1, \ldots, T_{m-1}$，每个集合本身又是一棵树，并且称之为根的子树(subTree)。

每棵子树的根结点有且仅有一个直接前驱，但可以有0个或多个后继。

编程兔教育

例如：



编程兔教育

基本概念

结点：

父结点、子结点、兄弟结点、堂兄弟、祖先结点、子孙结点、叶结点。

度：

结点的度：结点的子结点个数；

树的度：树中所有结点的度的最大值。

结点层次：约定根结点在第一层，其他结点的层数=父结点的层数 + 1。

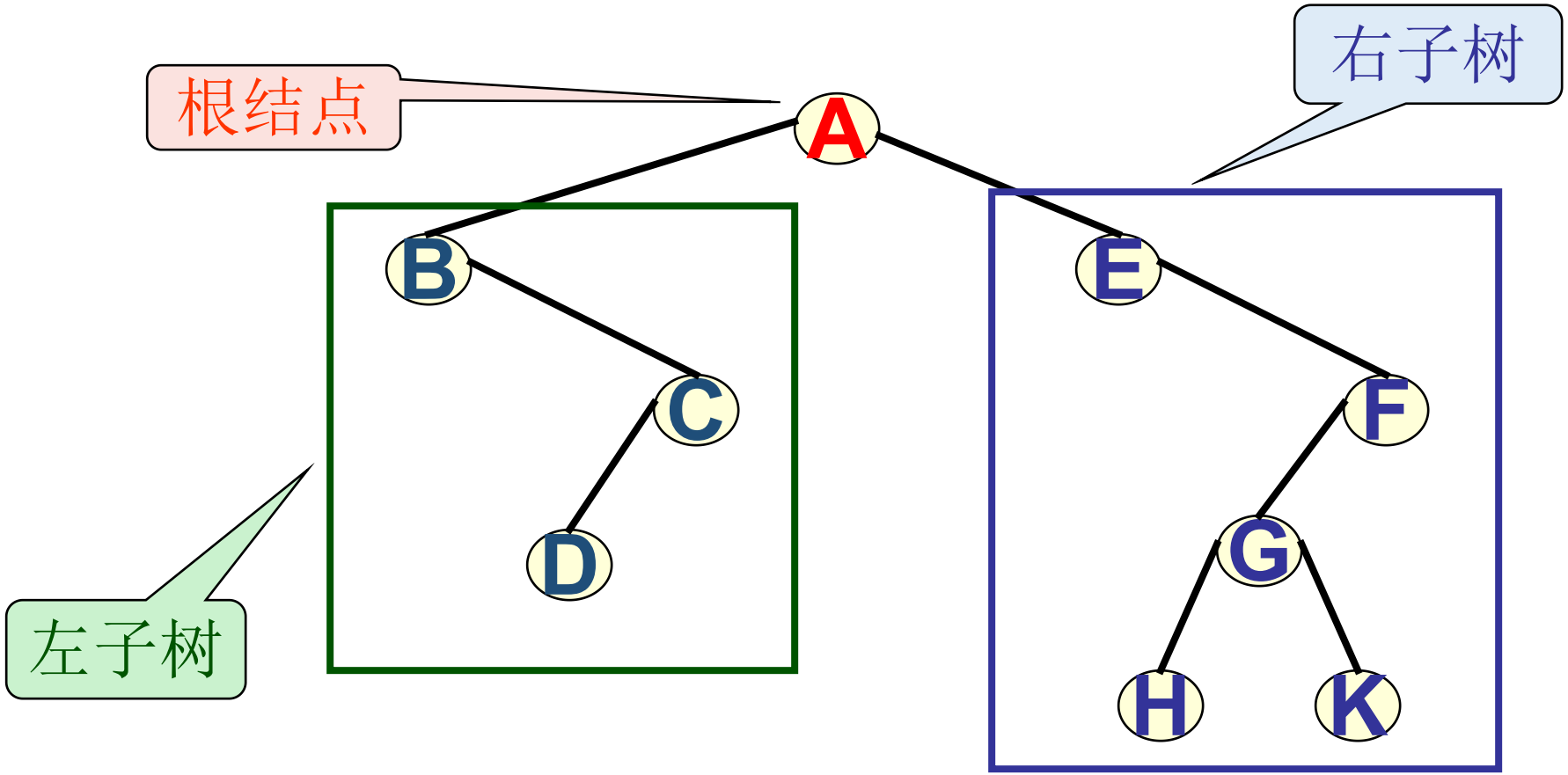
树的高度：树中所有结点的最大层数。

有序树和无序树：若结点的子树有次序排列，且先后次序不能互换，这样的树称为有序树，反之为无序树。

编程免费课

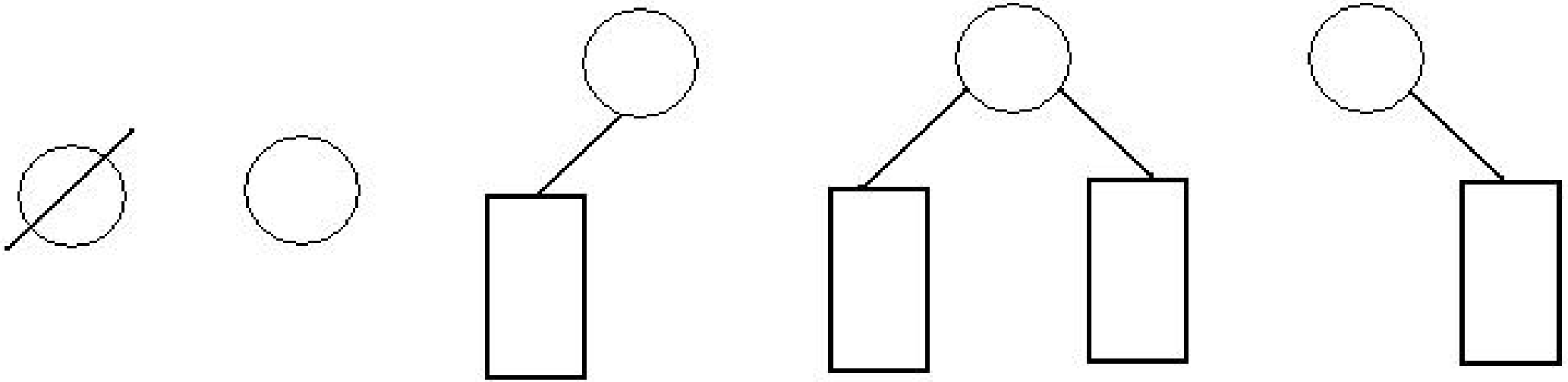
二叉树

度为2的有序树，每个结点最多有两个子结点，分别称为左孩子、右孩子结点。两棵子树分别称为左子树、右子树。



编程兔教育

二叉树有5种基本形态：



编程兔教育

二叉树的性质

1. 在二叉树的第i层上最多有2^{i-1}个结点（i>=1）。

证明：（数学归纳法）

当i=1时，$2^{i-1}=1$显然成立；

现在假设第i-1层时命题成立，即第i-1层上最多有2^{i-2} 个结点。

由于二叉树的每个结点的度最多为2，故在第i层上的最大结点数为第i-1层的2倍，即：$2*2^{i-2}=2^{i-1}$。

编程兔教育

2.深度为k的二叉树至多有2^k-1个结点（k>=1）。

证明：

每层的结点个数都取最大值时，二叉树中的结点个数最多。

由性质1，深度为k的二叉树的结点数至多为：

$2^0+2^1+\ldots+2^{k-1}$

$=2^k-1$

编程兔教育

3.对任意一棵二叉树，如果其叶结点数为n_0，度为2的结点数为n_2，则一定满足：$n_0=n_2+1$。

证明：利用两种方式表示结点总数n。

(1)度为0的结点数n_0、度为1的结点n_1、度为2的结点数n_2之和：

$n=n_0+n_1+n_2$ ……(式子1)

(2)每个度为1的结点有1个子结点，每个度为2 的结点有两个子结点，故二叉树中孩子结点总数是：n_1+2*n_2。

再加上根结点就是结点总数：

$n=n_1+2*n_2+1$ ……(式子2)

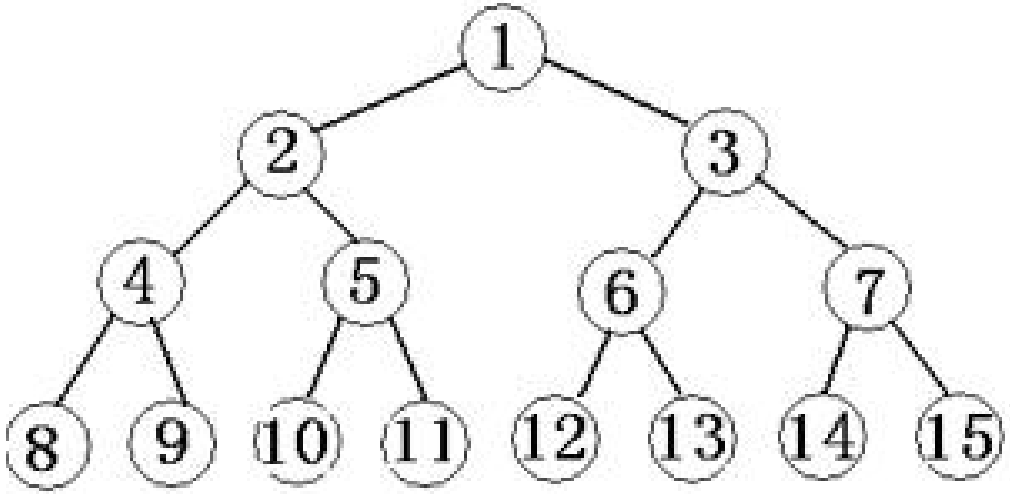
由式子1和式子2得到：$n_0=n_2+1$

编程兔教育

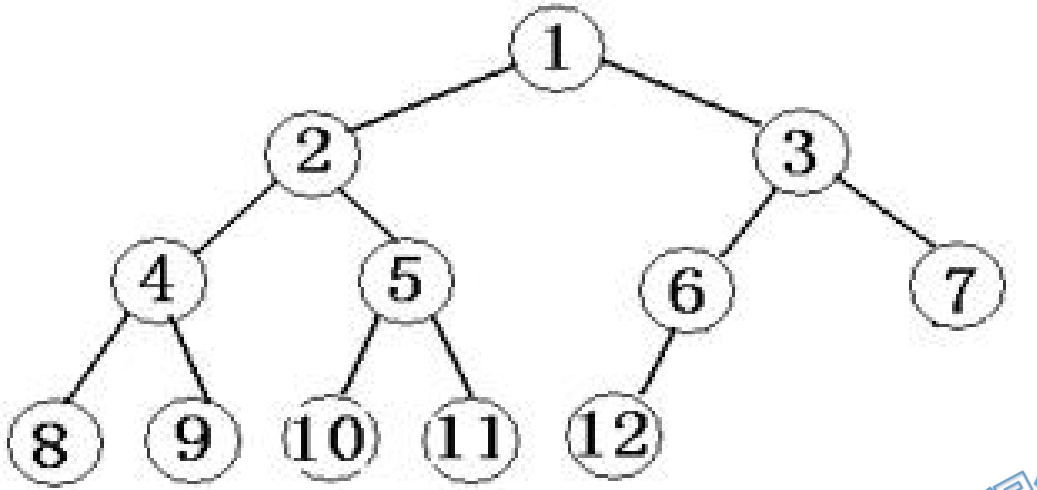
二种特殊的二叉树

一棵深度为k且有2^k-1个结点的二叉树称为<u>满二叉树</u>。

对满二叉树的结点进行连续编号，约定编号从根结点起，自上而下，从左到右，由此引出完全二叉树的定义，深度为k，有n个结点的二叉树，当且仅当其每一个结点都与深度为k的满二叉树中编号从1到n的结点一一对应时，称为<u>完全二叉树</u>。



图A



图B

编程兔教育

4. 具有 n 个结点的完全二叉树的深度为 floor($\log_2 n$) +1 。

证明：设完全二叉树的深度为 k ，根据完全二叉树的定义，前k-1层是满的，所以n> 2^{k-1} -1，而n ≤ 2^k -1，即2^{k-1}≤ n < 2^k，两边取对数得k-1 ≤ $\log_2 n$ < k 。

因为 k 只能是整数，因此， k = floor($\log_2 n$) +1 。

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，无父结点；否则，编号为i/2的结点为其父结点；

(2) 若 2*i>n，则结点i无左孩子，否则，编号为2*i 的结点为其左孩子结点；

(3) 若 2*i+1>n，则结点i无右孩子结点，否则，编号为2*i+1 的结点为其右孩子结点。

编程兔教育

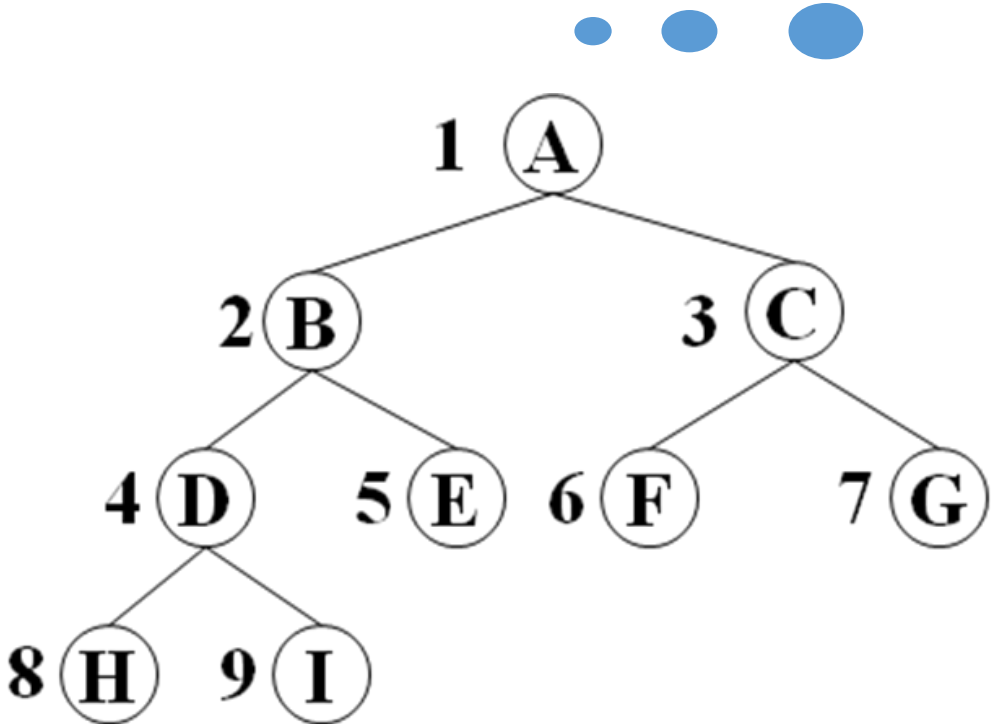
树的存储（二叉树）

因为每个结点最多有两个子结点，可以采用如下的方式存储：

```
struct btree{
    elemtype data;   //结点基本信息
    int fa;          //父结点
    int lc, rc;      //左右孩子结点
}bt[maxn];
```

编程兔教育

序号	结点信息	父结点	左孩子	右孩子
1	A	0	2	3
2	B	1	4	5
3	C	1	6	7
4	D	2	8	9
5	E	2	0	0
6	F	3	0	0
7	G	3	0	0
8	H	4	0	0
9	I	4	0	0



编程兔教育

树的存储（普通树）

父亲孩子表示法:

结点的子结点个数不确定，如果以数组的方式记录每个结点的子结点，需要按照最大结点个数开数组，会造成空间的浪费。

```
struct tree{
    elemtype data;   //结点基本信息
    int fa;                    //父结点
    int child[maxc]; //子结点
}t[maxn];
```

编程兔教育

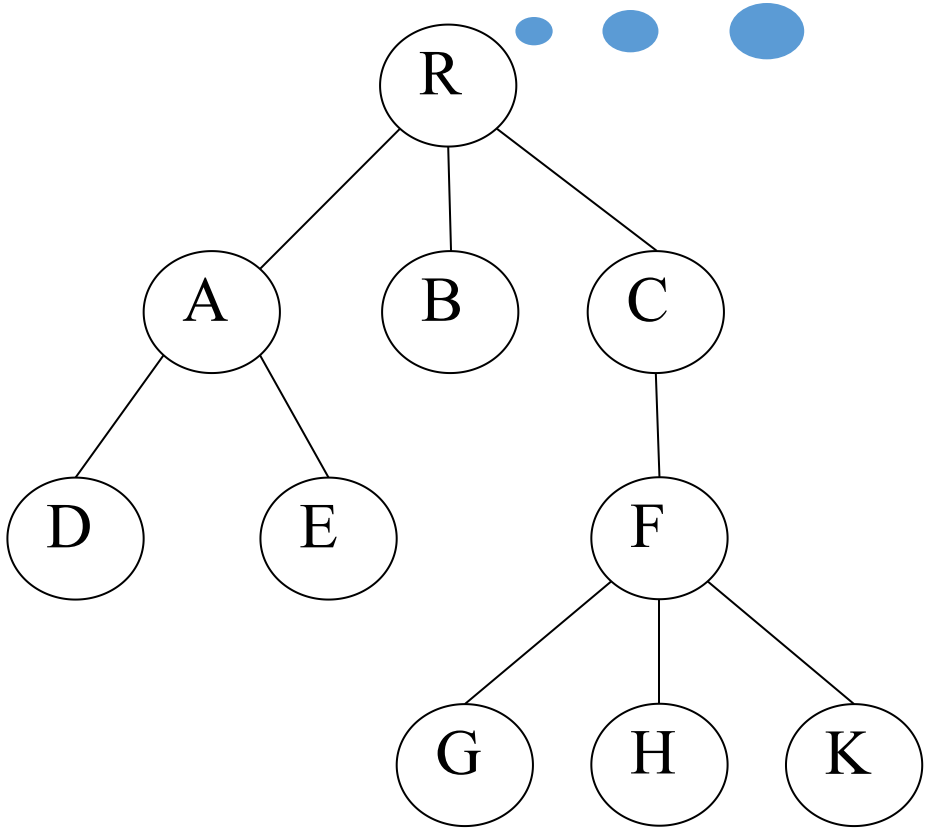
孩子兄弟表示法:

以链表的方式记录子结点信息

```
struct tree{
    elemtype data;   //结点基本信息
    int fa;          //父结点
    int first, next;     //first是第一个子结点，next是下一个兄弟结点
}t[maxn];
```

编程兔教育

序号	结点信息	父结点	第一个子结点	下一个兄弟结点
0	R	-1	3	-1
1	A	0	5	-1
2	B	0	-1	1
3	C	0	6	2
4	D	1	-1	-1
5	E	1	-1	4
6	F	3	9	-1
7	G	6	-1	-1
8	H	6	-1	7
9	K	6	-1	8



编程兔教育

多叉树转二叉树

用二叉树表示一棵树要比多叉树简单些，而且多叉树都可以转换成唯一的二叉树。

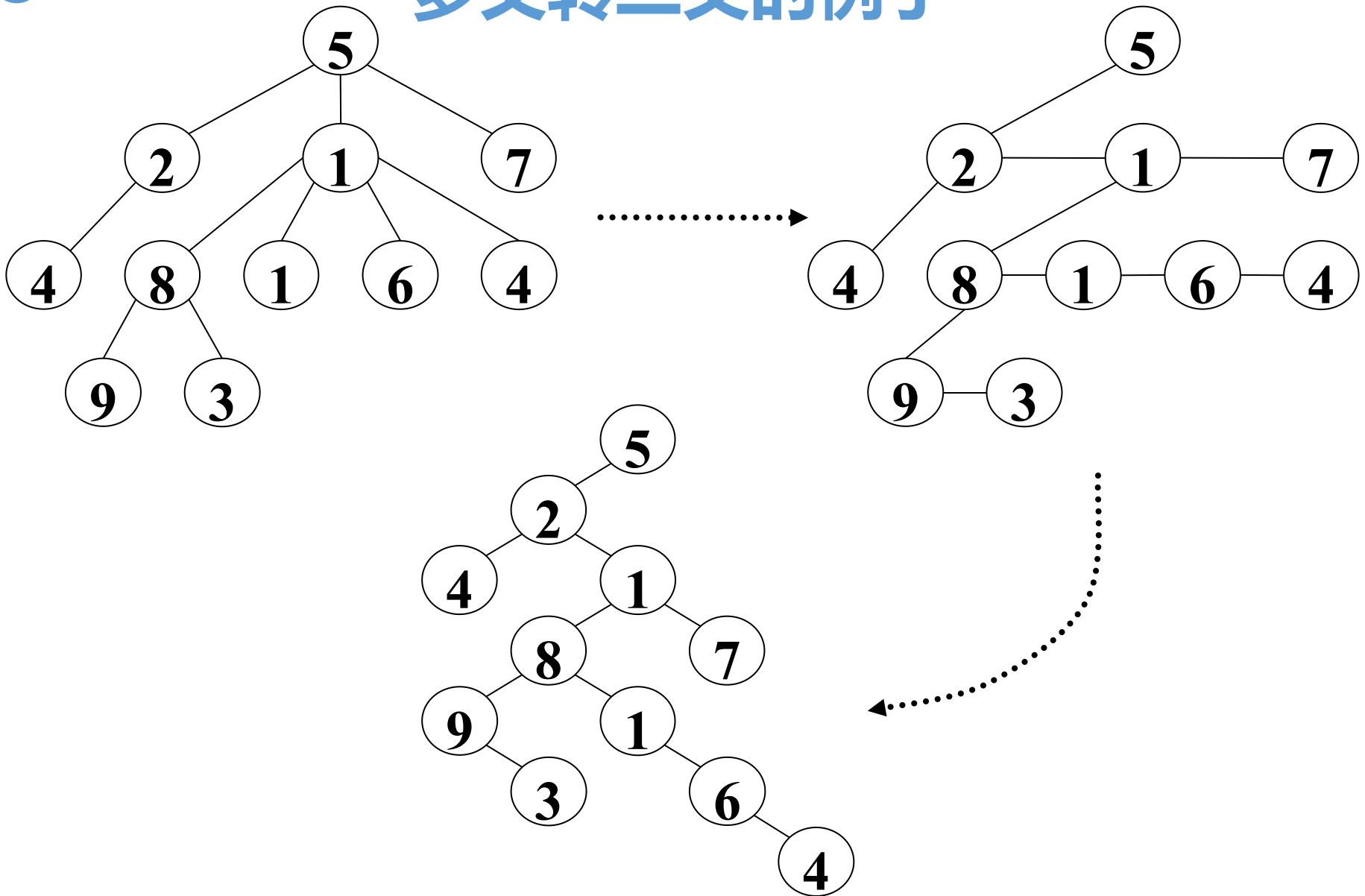
对于多叉树中的每个结点，可以用两个指针分别指向它的第一个子结点和下一个相邻（兄弟）结点。

1.在树中各兄弟（堂兄弟除外）之间加一根连线。

2.对于任一结点，只保留它与最左孩子的连线外，删去它与其余孩子之间的连线。

编程兔教育

多叉转二叉的例子



编程兔教育

二叉树的遍历

在二叉树的应用中，常常要求在树中查找具有某种特征的结点，或者对全部结点逐一进行某种处理。这就是二叉树的遍历问题。

所谓二叉树的遍历是指按一定的规律和次序访问树中的各个结点，而且每个结点仅被访问一次。“访问”的含义很广，可以是对结点作各种处理，如输出结点的信息等。

先（根）序遍历、中（根）序遍历、后（根）序遍历；

按层次遍历。

编程兔教育

先序遍历：

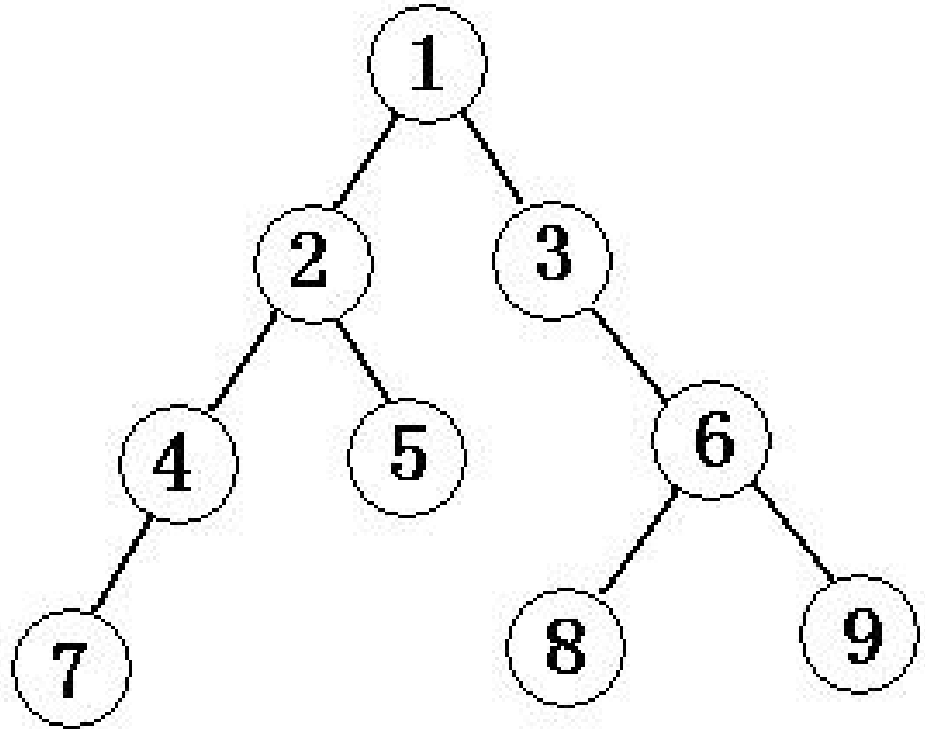
若二叉树为空，结束，否则：

1.访问根结点

2.先序遍历左子树

3.先序遍历右子树

```
void preorder(int k){
    if(k != -1){
            visit(k);     //访问子树根结点
            preorder (bt[k].lc);
            preorder (bt[k].rc);
    }
}
```



编程兔教育

中序遍历：

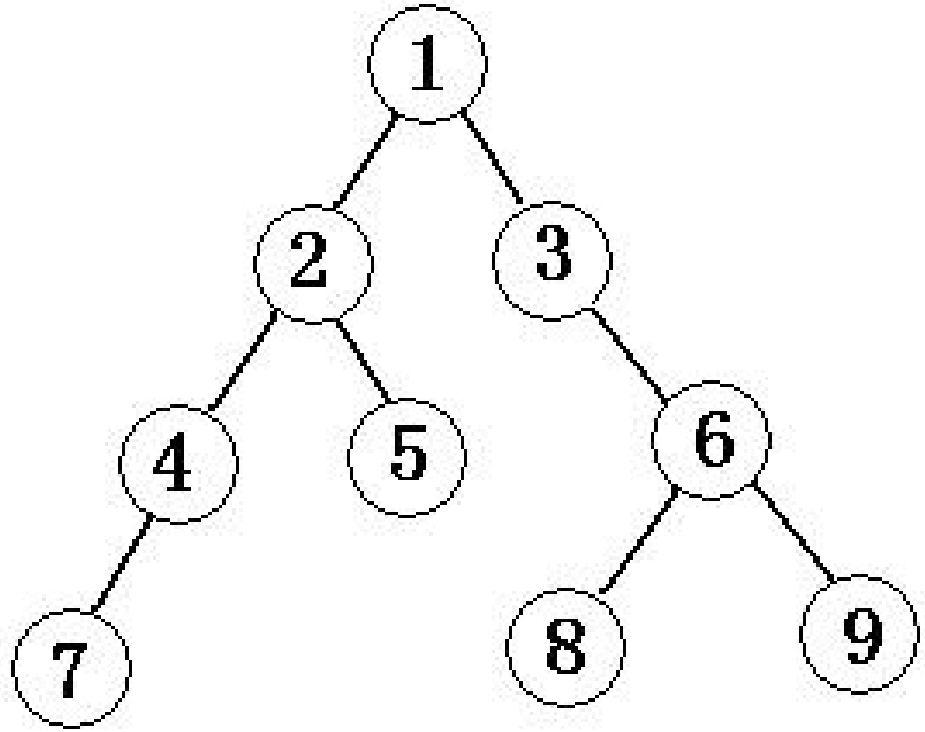
若二叉树为空，结束，否则：

1.中序遍历左子树

2.访问根结点

3.中序遍历右子树

```
void inorder (int k){
    if(k != -1){
            inorder (bt[k].lc);
            visit(k);      //访问子树根结点
    inorder (bt[k].rc);
    }
}
```



编程兔教育

后序遍历：

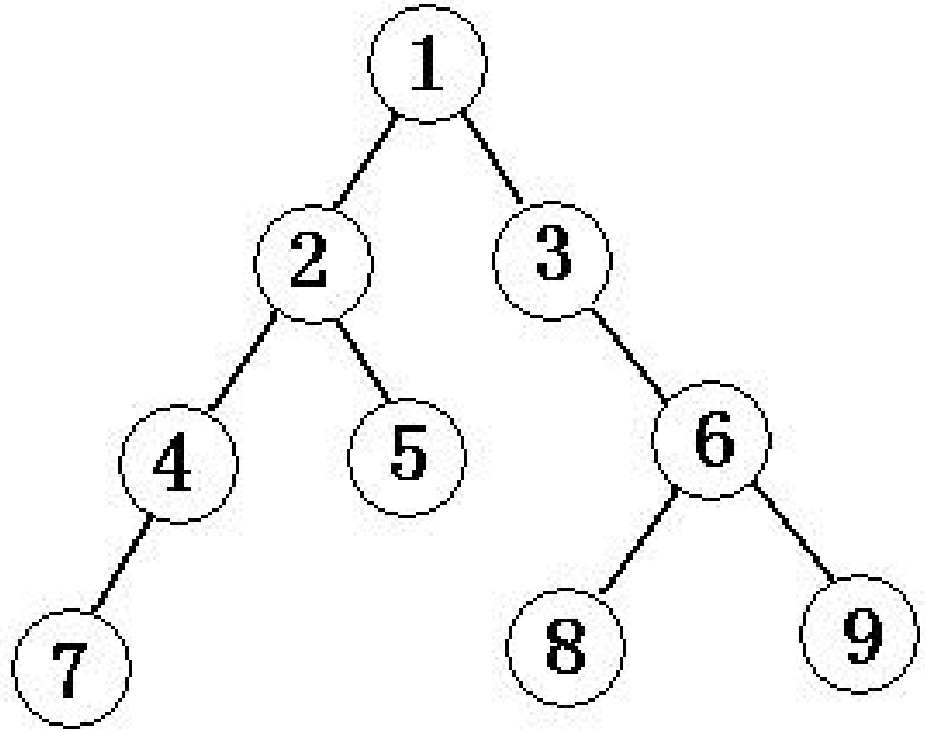
若二叉树为空，结束，否则：

1.后序遍历左子树

2.后序遍历右子树

3.访问根结点

```
void postorder (int k){
    if(k != -1){
            postorder (bt[k].lc);
            postorder (bt[k].rc);
            visit(k);      //访问子树根结点
    }
}
```



编程兔教育

例题：新二叉树

【题目描述】
输入一串完全二叉树，用前序遍历打出。
【输入格式】
第一行为二叉树的结点数n(n<=26)。
后面n行，每一个字母为结点，后两个字母分别为其左右儿子。
空结点用*表示
【输出格式】
前序排列的完全二叉树
【输入样例】
6
abc
bdi
cj*
d**
i**
j**
【输出样例】：
abdicj

编程兔教育

```
#include <cstdio>
const int maxl=60;
struct node{int ll,rr;}a[maxl];
int n;char ss[3];bool v[maxl];
void pre(int k){
        printf("%c",k+'a'-1);
        if(a[k].ll) pre(a[k].ll);
        if(a[k].rr) pre(a[k].rr);
}
int main(){
        scanf("%d",&n);
        for(int i=1;i<=n;i++){
                scanf("%s",ss);
                if(ss[1]!='*'){
                        a[ss[0]-'a'+1].ll=ss[1]-'a'+1;v[ss[1]-'a'+1]=true;}
                if(ss[2]!='*'){
                        a[ss[0]-'a'+1].rr=ss[2]-'a'+1;v[ss[2]-'a'+1]=true;}

        }
        int root;
        for(root=1;v[root];root++);
        pre(root); printf("\n");
        }
```

【输入样例】

```
6
abc
bdi
cj*
d**
i**
j**
```

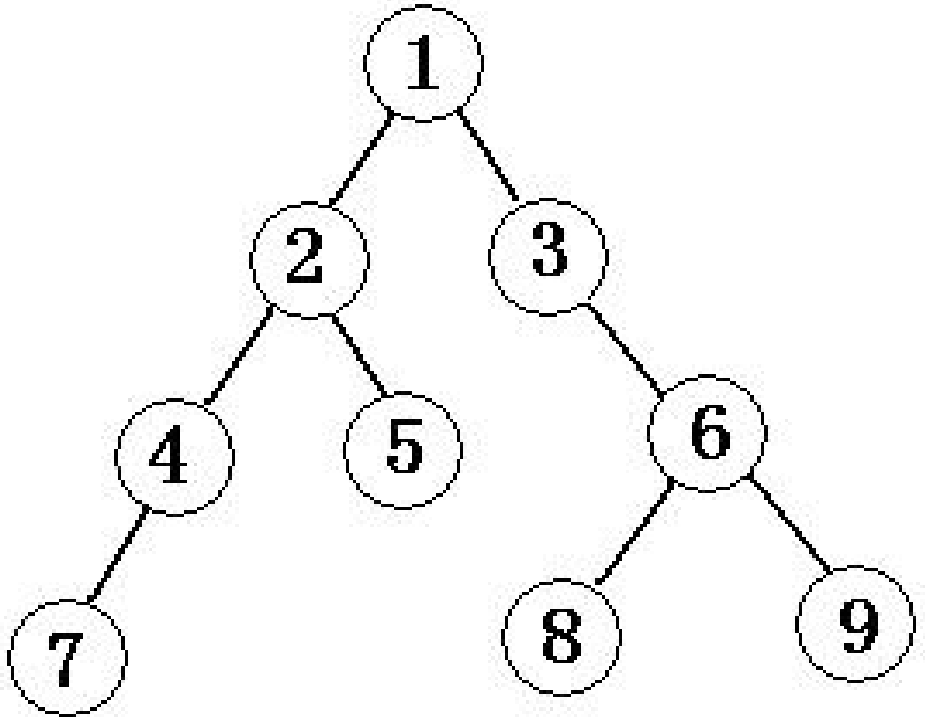
【输出样例】：

```
abdicj
```

编程兔教育

非递归的中序遍历：利用栈来实现。

```
void inorder (int k){
    int stack[maxn], top=0;
    while (k!=-1 || top){
        while (k!=-1){
            stack[top++]=k;
            k=bt[k].lc;
        }
        if (top){
            visit(stack[--top]);
            k=bt[stack[top]].rc;
        }
    }
}
```



编程兔教育

按层次遍历：

先访问根结点，

再依次访问根结点的所有子结点，

然后依次访问根结点的子结点的子结点。

广度优先搜索（基于队列的应用）

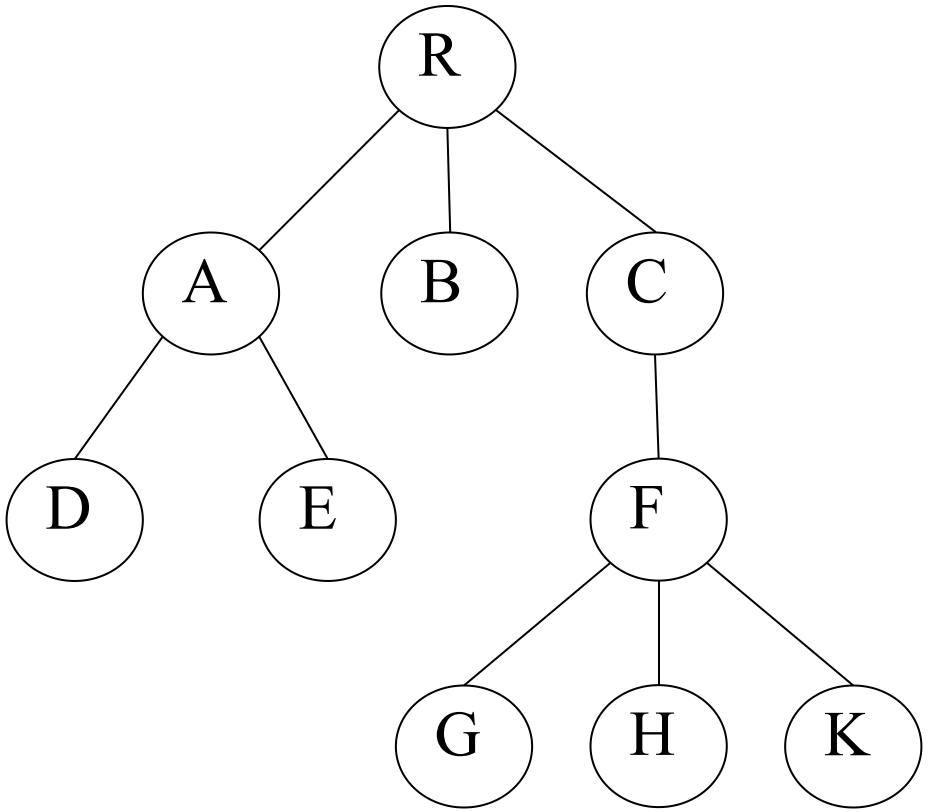
编程兔教育

树的遍历（普通树）：

先（根）序遍历：先访问根结点，再从左到右先序遍历各棵子树。

后（根）序遍历：先从左到右后序遍历各棵子树，再访问根结点。

按层次遍历。



编程兔教育

例题：P1030 求先序排列

【题目描述】

给出一棵二叉树的中序与后序排列。求出它的先序排列。（约定树结点用不同的大写字母表示，长度<=8）。

【输入格式】

2行，均为大写字母组成的字符串，表示一棵二叉树的中序与后序排列。

【输出格式】

1行，表示一棵二叉树的先序。

【输入样例】

BADC

BDCA

【输出样例】

ABCD

编程兔教育

算法分析：

中序：BADC

后序：BDCA

A为根结点，B、DC分别为左右子树的中序序列；

B、DC分别为左右子树的后序序列。

递归处理：中序为B，后序为B的子树；

递归处理：中序为DC，后序为DC的子树；

编程兔教育

```
#include <cstdio>
#include <string>
#include <iostream>
using namespace std;
const int maxl=10;
string mid,post;
void solve(int lmid,int rmid,int lpost,int rpost){          //[left,right)
        printf("%c",post[rpost-1]);
        int p=mid.find(post[rpost-1]);
        if(p>lmid) solve(lmid,p,lpost,lpost+p-lmid);
        if(p+1<rmid) solve(p+1,rmid,lpost+p-lmid,rpost-1);
}
int main(){
        cin>>mid>>post;
        solve(0,mid.size(),0,post.size()); printf("\n");
        return 0;
}
```

编程兔教育

二叉树的应用——哈夫曼树

例：P1090 合并果子

【题目描述】

在一个果园里，多多已经将所有的果子打了下来，而且按果子的不同种类分成了不同的堆。多多决定把所有的果子合成一堆。

每一次合并，多多可以把两堆果子合并到一起，消耗的体力等于两堆果子的重量之和。可以看出，所有的果子经过n-1次合并之后，就只剩下一堆了。多多在合并果子时总共消耗的体力等于每次合并所耗体力之和。

因为还要花大力气把这些果子搬回家，所以多多在合并果子时要尽可能地节省体力。假定每个果子重量都为1，并且已知果子的种类数和每种果子的数目，你的任务是设计出合并的次序方案，使多多耗费的体力最少，并输出这个最小的体力耗费值。

例如有3种果子，数目依次为1，2，9。可以先将1、2堆合并，新堆数目为3，耗费体力为3。接着，将新堆与原先的第三堆合并，又得到新的堆，数目为12，耗费体力为12。所以多多总共耗费体力=3+12=15。可以证明15为最小的体力耗费值。

编程兔教育

【输入格式】
输入文件fruit.in包括两行，第一行是一个整数n(1< = n<=10000)，表示果子的种类数。第二行包含n个整数，用空格分隔，第i个整数a_i(1< = a_i<=20000)是第i种果子的数目。
【输出格式】
输出文件fruit.out包括一行，这一行只包含一个整数，也就是最小的体力耗费值。输入数据保证这个值小于2^{31}。
【输入样例】
3
1 2 9
【说明】
对于30%的数据，保证有n<=1000；对于50%的数据，保证有n<=5000； 对于全部的数据，保证有n<=10000。
【输出样例】
15

编程兔教育

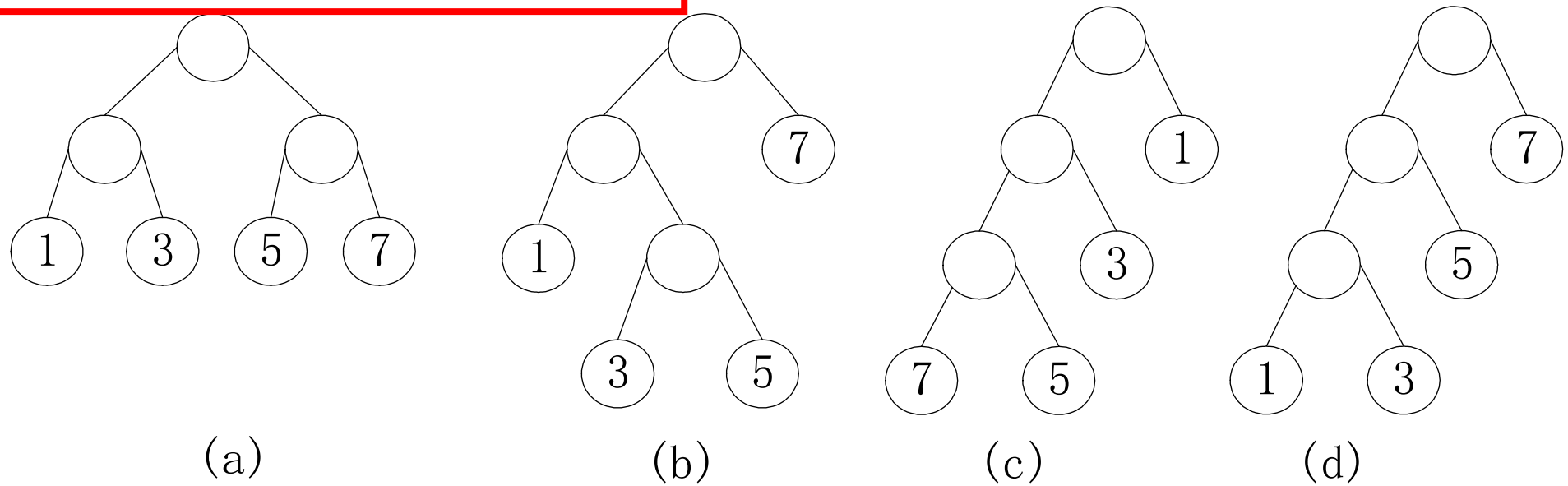
哈夫曼（Huffman）树又称最优二叉树或最优搜索树，是一种带权路径长度最短的二叉树。树的带权路径长度定义为树中所有叶子结点的带权路径长度之和，$WPL = \Sigma w_k l_k$ (对所有叶子结点)。

(a) 1×2+3×2+5×2+7×2=32

(b) 1×2+3×3+5×3+7×1=33

(c) 7×3+5×3+3×2+1×1=43

(d) 1×3+3×3+5×2+7×1=29



编程兔教育

构造哈夫曼树：

1)以权值分别为$W_1,W_2,\ldots,W_n$的n个结点，构成n棵二叉树$T_1,T_2,\ldots,T_n$，并组成森林$F=\{T_1,T_2,\ldots,T_n\}$，其中每棵二叉树T_i只有一个权值为W_i的根结点；

2) 在F中选取两棵根结点权值最小的树作为左右子树构造一棵新二叉树，新二叉树根结点权值为左右子树上根结点的权值之和；

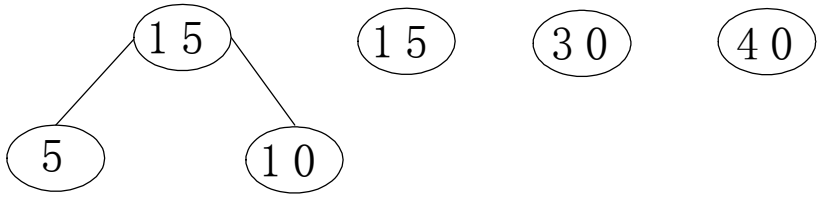
3) 从F中删除这两棵二叉树，同时将新二叉树加入到F中;

4) 重复2)、3)直到F中只含一棵二叉树为止，这棵二叉树就是Huffman树。

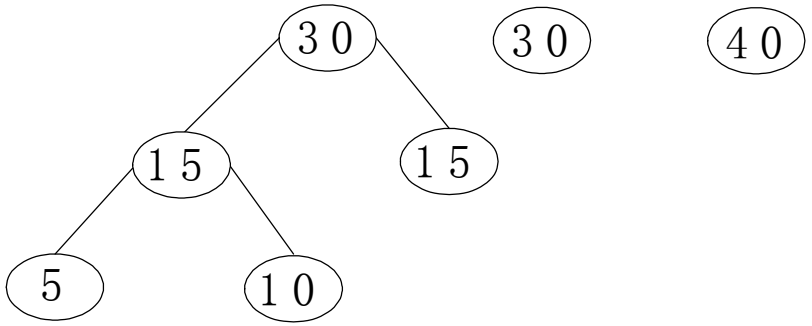
编程兔教育

- 给定权值集合{5，15，40，30，10}构造哈夫曼树的过程如图所示，其中最优的带权路径长度为：(5 + 10)×4 + 15×3 + 30×2 + 40 = 205。
- 可以看出，哈夫曼树的结点的度数为0或2，没有度为1的结点。

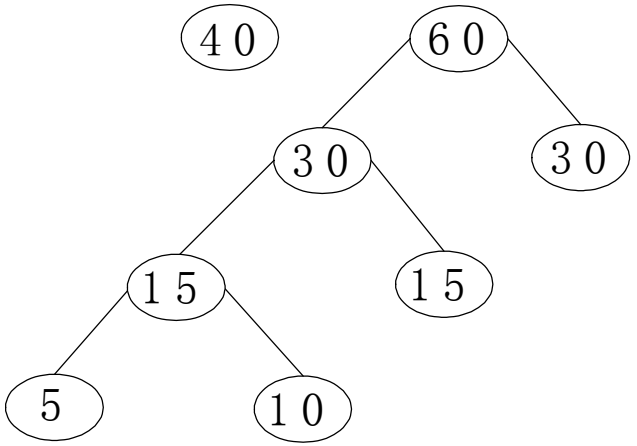
（a）第 一 步



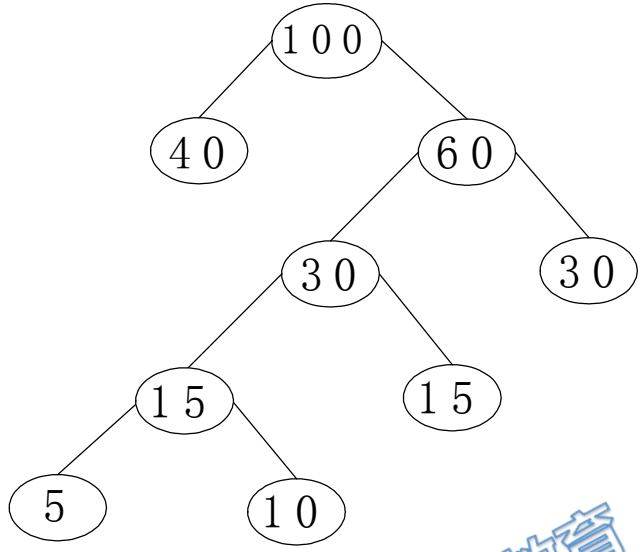
（b）第 二 步



（c）第 三 步



（d）第 四 步



（e）第 五 步

编程兔教育

哈夫曼树的算法实现

算法的核心步骤：

从若干棵二叉树中，选取根结点权值最小的两棵二叉树；

把这两棵二叉树合并成一棵新的二叉树，加入到森林中，并删除原来的两棵二叉树。

朴素的算法：

对森林中的二叉树按根结点的权值排序，可以找出最小的两棵二叉树；

但加入新的二叉树，并删除原来的两棵二叉树之后，森林中的二叉树不再有序，下一次仍需排序。O(n * n * logn)

编程免费教育

算法一：

把森林中的n棵二叉树排序，作为一个队列；

合并形成的新二叉树放到另一个队列中(升序)；

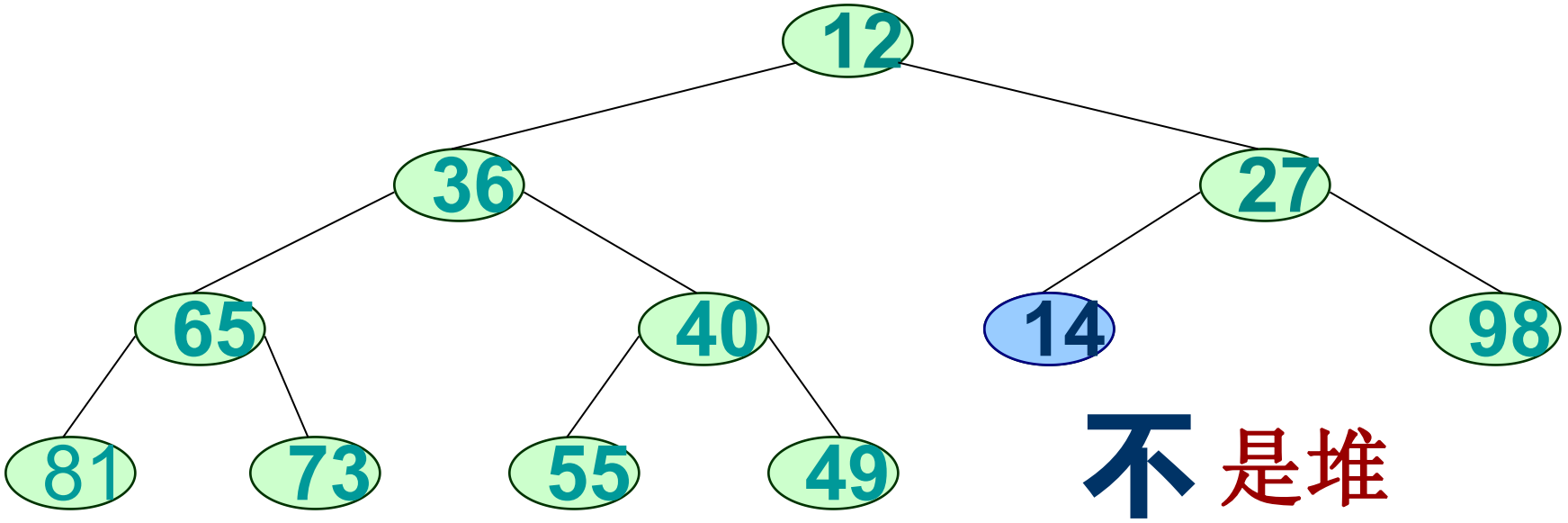
每次选取最小的二叉树时，只需比较两个队列的队首元素。

编程兔教育

算法二(二叉堆)：

堆是一棵完全二叉树，对于每一个非叶子结点，它的权值都不大于（或不小于）左右孩子的权值，这样的堆称之为小根堆（大根堆）。

对于小（大）根堆，根结点的权值最小（大）的。

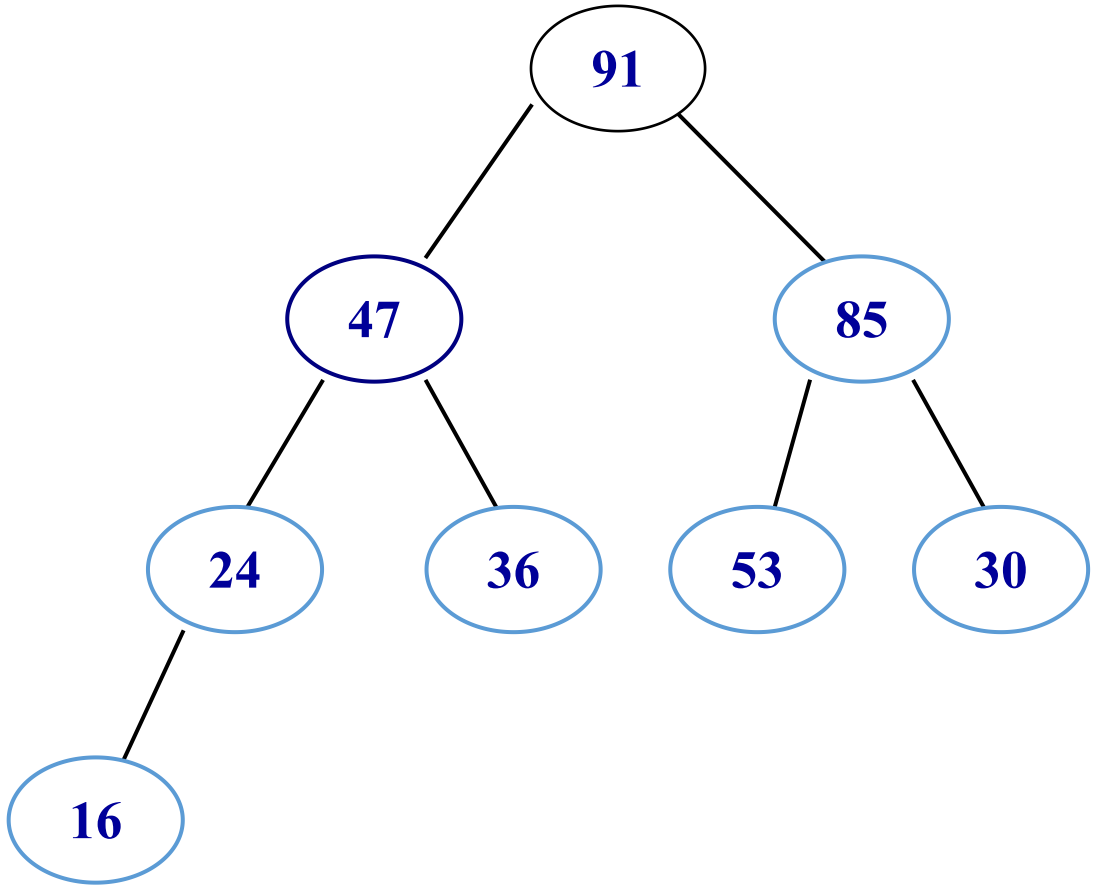


编程兔教育

堆

堆

- 堆是一棵完全二叉树
- 不仅仅是完全二叉树



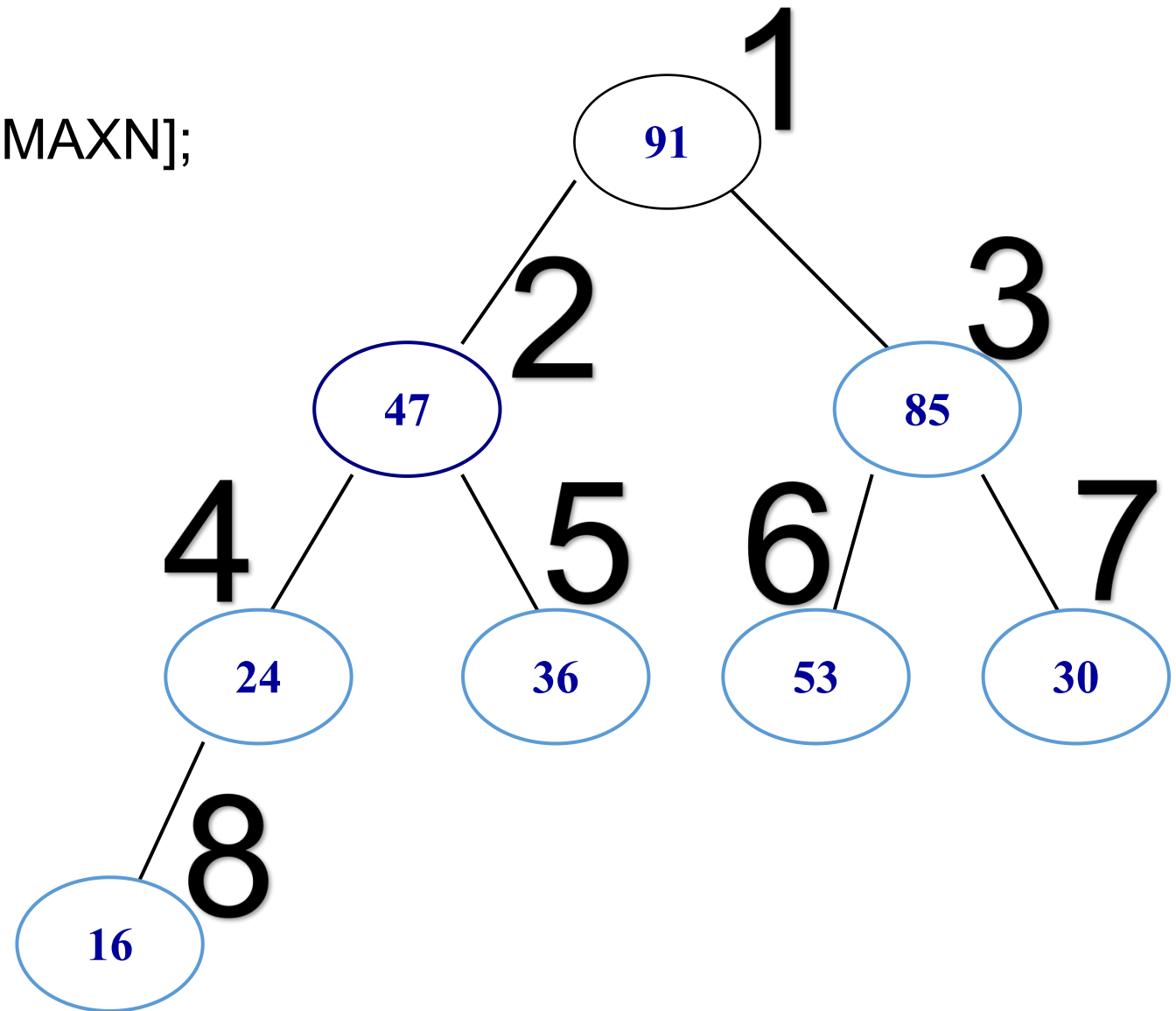
编程兔教育

- 大根堆
  - 每个结点的权值都比儿子的权值大
- 小根堆
  - 每个结点的权值都比儿子的权值小
- 因为堆的性质，大根堆的根结点一定权值最大，小根堆的根结点一定权值最小

编程兔教育

堆的存储

- int heap[MAXN];

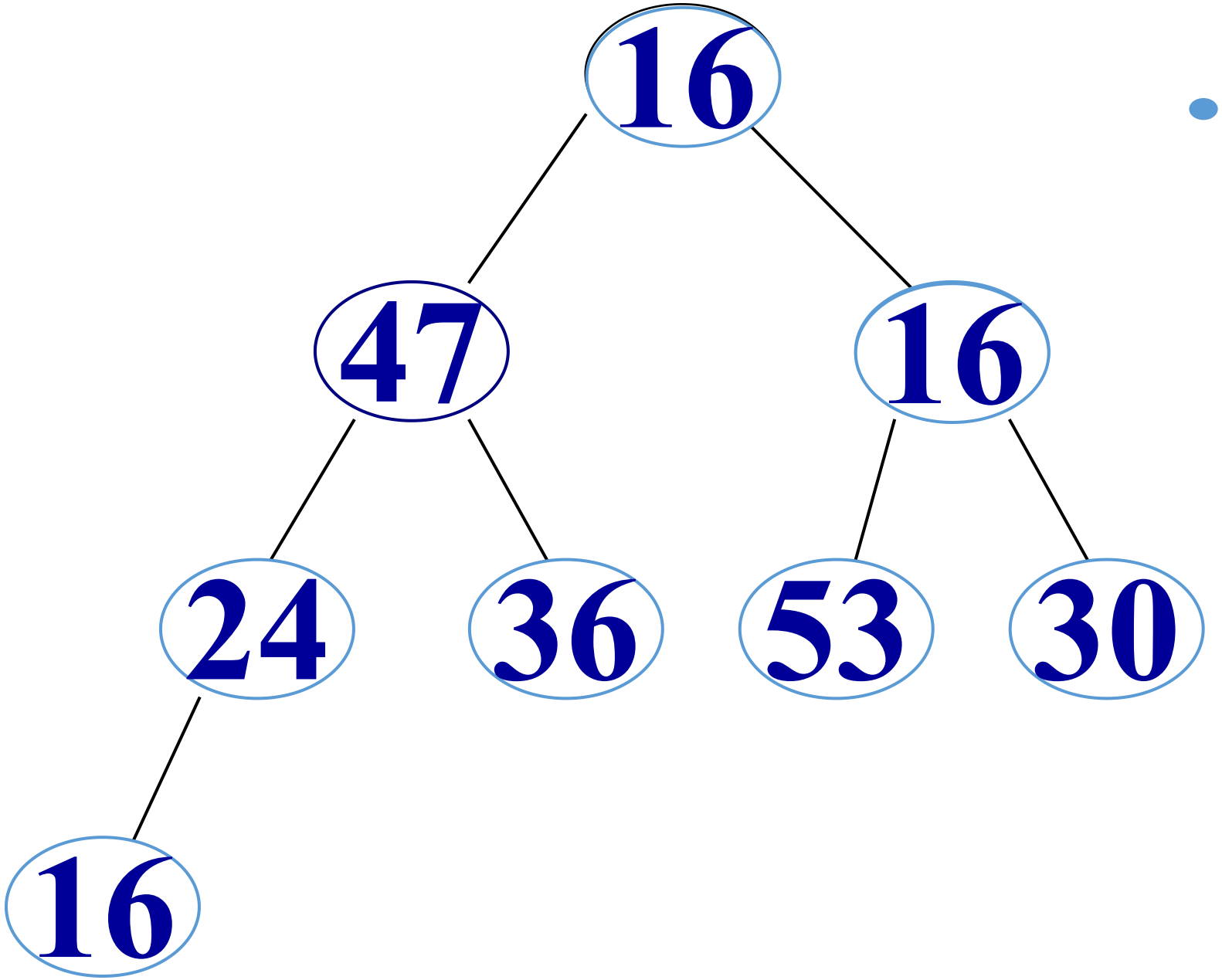


编程兔教育

向下调整（筛选）

- 所谓“**筛选**”指的是，对一棵左/右子树均为堆的完全二叉树，“**调整**”**根结点**使整个二叉树也成为一个堆。
- 假设有一个堆heap[1..n]，我现在要删除第一个元素（最大的元素），如何维护堆的性质不变？
- 将根结点删除，把最后一个点移动到根结点上来。如果当前点比某一个儿子小，就与儿子交换（如果比两个儿子都小，与最大的儿子交换），以此类推，直到没有儿子或者比儿子大。

编程兔教育



编程兔教育

向下调整（筛选）

```
void heap_adjust(int x) {
    while (x*2<=tot) {
        int j=x*2;
        if (x*2+1<=tot&&heap[x*2+1]>heap[x*2])
            j++;
        if (heap[j]>heap[x])
            swap(heap[x],heap[j]),x=j;
        else
            break;
    }
}
```

编程兔教育

堆的初始化（建堆）

- 现在我们有一个数组a[1..n]，如何把它初始化成一个堆？
- 从编号大的点，到编号小的点，依次进行“筛选”操作。

编程兔教育

堆的初始化（建堆）

- for (int i = tot / 2; i >= 1; i--)
  - heap_adjust(i);
- 为什么是对的？
  - 每次对一个点做调整的时候，它的子树一定都已经是合法的了
  - 对1节点进行调整后，整个堆就是合法的了

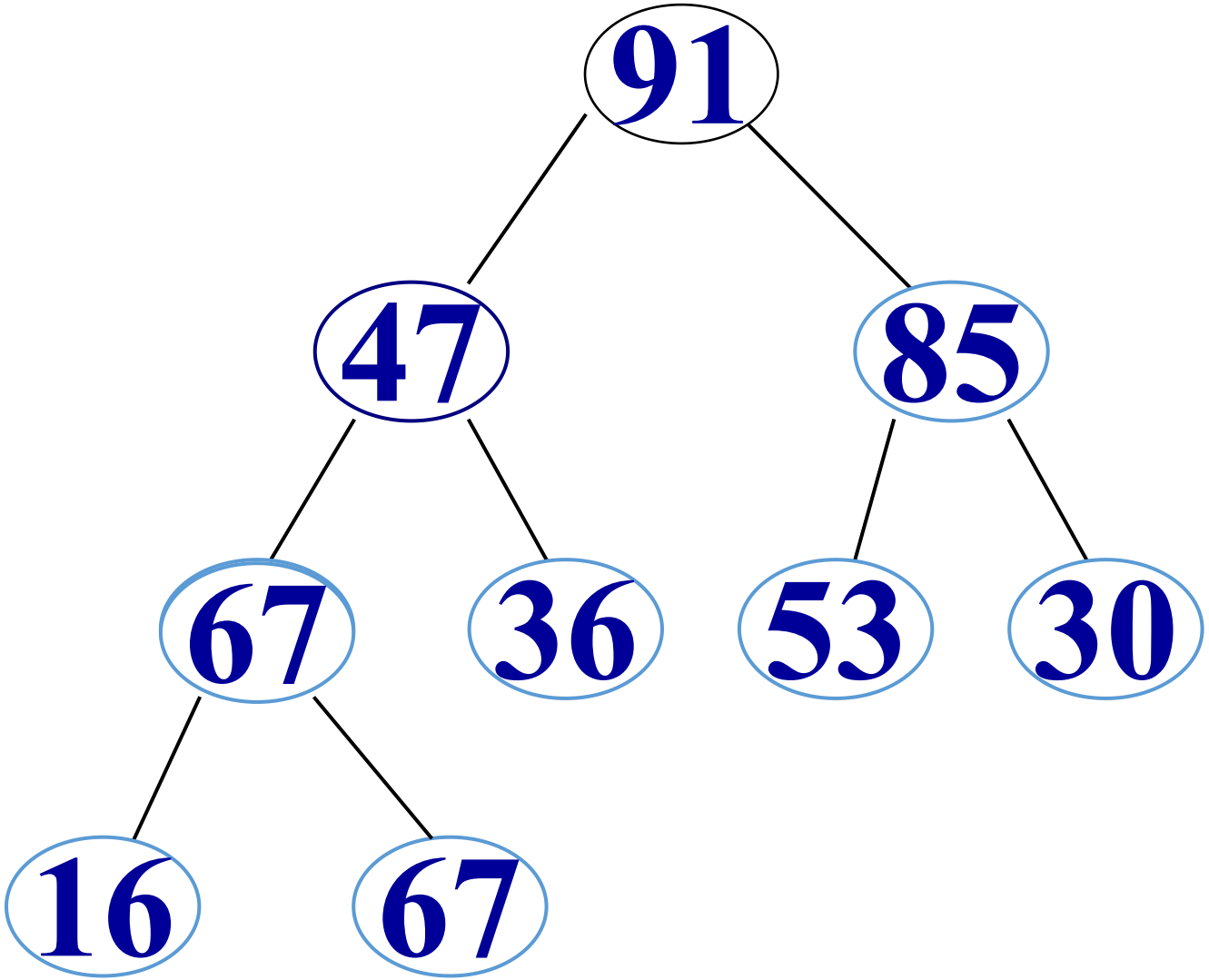
编程兔教育

加入新的点（向上调整）

- 如何在堆中加入新的结点？
- 如果现在堆中共有tot个结点，我们把新的结点放在位置tot+1上。
- 运用和“筛选”类似的方法，我们不停地比较当前点和**父亲**，如果当前点比父亲大，则交换，直到交换到根或者不再比父亲大为止。

编程兔教育

加入新的点（向上调整）



编程兔教育

加入新的点（向上调整）

- 正确性？
  - 除了新的点到根路径上的这些点，其余点都满足比父亲小
  - 经过调整后，路径上的点也合法了

编程兔教育

小根堆的基本操作：

取堆中元素的最小值：h[1]

删除元素：一般是删除堆顶，用最后一个元素覆盖堆顶，堆中元素个数减1；

添加元素：加在完全二叉树的最后，堆中元素个数加1。

删除或添加元素都可能会破坏堆的性质，需要进行相应的调整。

编程兔教育

```
//添加元素后，该元素可能需要向上调整。
void up(int k){
        while (k>1){
                if (h[k/2]>h[k]){
                        swap(h[k/2], h[k]);
                        k/= 2;
                }
                else return;
        }
}
//删除堆顶，最后一个元素来到堆顶后，该元素可能需要向下调整（筛选）：down(1)
void down(int k){
        while (2*k<= tot){ //tot:堆中的元素个数
                int j = 2 * k;
                if (j<tot && h[j+1]<h[j]) j++;
                if (h[j]<h[k]){
                        swap(h[j], h[k]);
                        k = j;
                }else return;
        }
}
```


编程兔教育

```
int main(){
        scanf("%d", &n);
        //元素进堆，每次都向上调整
        for(int i=1; i<=n; i++){
                scanf("%d", &h[i]); tot++;
                up(tot);
        }
        //n-1次合并
        for(int i=1; i<n; i++){
                tmp = h[1];  //堆中的最小元素
                h[1] = h[tot--]; down(1);
                tmp += h[1]; //再取堆中的最小元素
                h[1] = h[tot--]; down(1);
                ans += tmp;  //本次合并的代价
                //合并所得的新元素进堆
                h[++tot] = tmp; up(tot);
        }
        printf("%d\n", ans);
        return 0;
}
```

编程兔教育

优先队列

优先队列定义： priority_queue<Type, Container, Functional>
Type 就是数据类型，Container 就是容器类型（Container必须是用数组实现的容器，比如vector,deque等等，但不能用 list。STL里面默认用的是vector），Functional 就是比较函数，当需要用自定义的数据类型时才需要传入这三个参数，使用基本数据类型时，只需要传入数据类型，默认是大顶堆。

```
priority_queue <int, vector<int>, greater<int> > q; //升序队列
priority_queue <int, vector<int>, less<int> >q; //降序队列
```

头文件：

```
#include <queue>
```

基本操作：

empty()	如果队列为空，则返回真
pop()	删除对头元素，删除第一个元素
push()	加入一个元素
size()	返回优先队列中拥有的元素个数
top()	返回优先队列对头元素，即优先级最高的元素

在默认的优先队列中，优先级高的先出队。在默认的int型中先出队的为较大的数。

编程兔教育

```
#include <iostream>
#include <cstdio>
#include <queue>
using namespace std;
int n;
priority_queue< int, vector<int>, greater<int> > h;//优先队列
int main(){
        int i,x,y,sum=0;
        cin>>n;
        for (i=1;i<=n;i++) {
                cin>>x;
                h.push(x);
        }
        for (i=1;i<n;i++){
                x=h.top();h.pop();
                y=h.top();h.pop();
                sum+=x+y;
                h.push(x+y);
        }
        cout<<sum;
        return 0;
}
```

编程兔教育

结构体优先队列

使用C++标准模板库STL，自定义类型

```
#include <cstdio>
#include <queue>
using namespace std;
const int maxn=10010;
struct node{
        int v;  //元素的值
        //运算符重载。元素的值大的，优先级低。小根堆
        bool operator<(const struct node x)const{
                return v>x.v;
        }
}cur;
priority_queue<struct node> pq;//优先队列
int main(){
        int n,sum=0;
        scanf("%d",&n);
        for(int i=0;i<n;i++){scanf("%d",&cur.v);pq.push(cur);}
        for(int i=1;i<n;i++){
                cur=pq.top(); pq.pop(); //取优先队列的队首元素，即最小值
                cur.v+=pq.top().v; pq.pop();//再取新的队首元素，即当前的最小值
                sum+=cur.v; pq.push(cur); //合并之后的新元素进入优先队列
        }
        printf("%d\n",sum);
        return 0;
}
```

编程兔教育

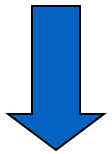
堆排序

- 即利用堆的特性进行排序的一种排序方法。
- 步骤：
- 1. 建立初始大顶堆，即将整个序列调整为堆
- 2. 输出根结点，将堆顶元素和最后元素交换，从堆中删除原来的堆顶元素
- 3. 重新调整为大顶堆
- 重复第2、3步，直到所有的元素都被删除。

编程兔教育

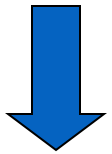
例如：

{ 40, 55, 49, 73, 12, 27, 98, 81, 64, 36 }



建大顶堆

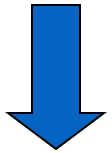
{ 98, 81, 49, 73, 36, 27, 40, 55, 64, 12 }



交换 98 和 12

{ 12, 81, 49, 73, 36, 27, 40, 55, 64, 98 }

经过筛选



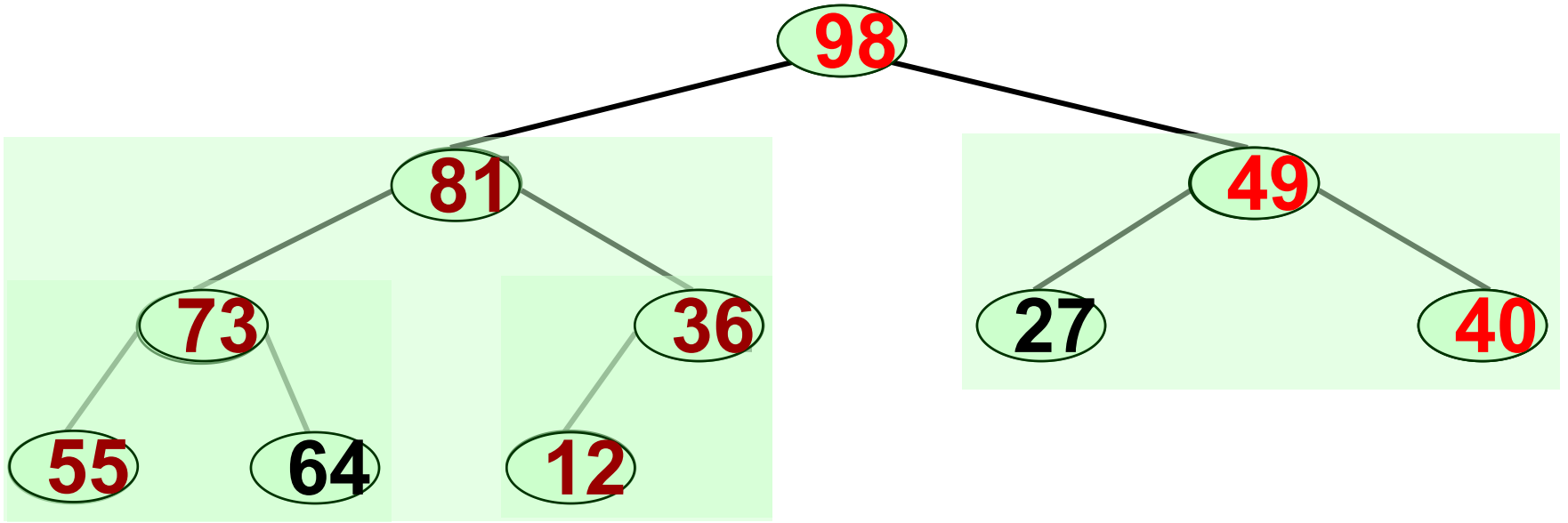
重新调整为大顶堆

{ 81, 73, 49, 64, 36, 27, 40, 55, 12, 98 }

编程兔教育

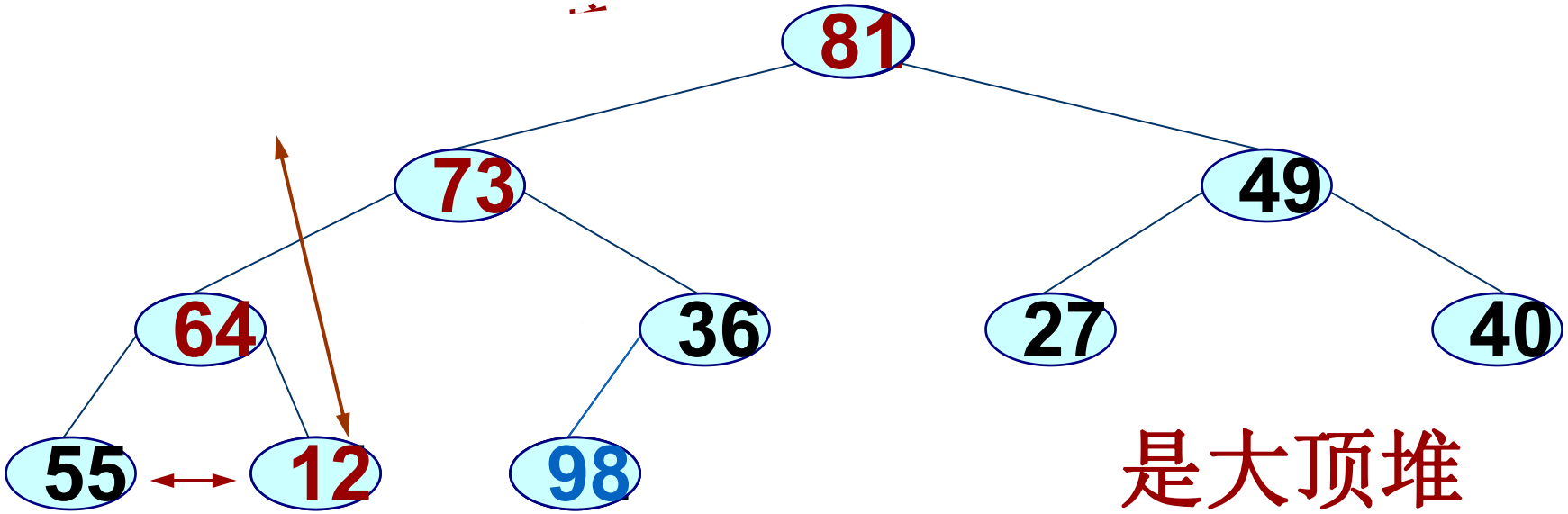
建堆是一个从下往上进行“筛选”的过程。

例如: 排序之前的关键字序列为



现在，左/右子树都已经调整为堆，最后只要调整根结点，使整个二叉树是个“堆”即可。

编程兔教育



但在 98 和 12 进行互换之后，它就**不**是堆了，

因此，需要对它进行“筛选”。

编程兔教育

- 哈夫曼编码：
  - 哈夫曼树的应用很广，哈夫曼编码就是哈夫曼树在电讯通信中的应用之一。
  - 在电报通信中，电文是以二进制的0，1序列传送的。在发送端需要将电文中的字符转换成0，1序列（编码）发送，在接收端又需要把接收到的0，1序列还原成相应的字符序列（译码）。
  - 最简单的二进制编码方式是等长编码。假定需传送的电文是ACCBDDD，在电文中仅使用A，B，C，D 4种字符，则只需用两个字符便可分辨。可依次对其编码为：00，01，10，11。上述需发送的的电文是“00101001111111”。译码员可按两位一组进行译码，恢复原来的电文。
  - 这种编码方式下，电文的总长度：2 * 7 = 14，不一定最短。

编程兔教育

ACCBDDD

出现的频度：A、B：1　　C：2　　D：3

A:000

B:001

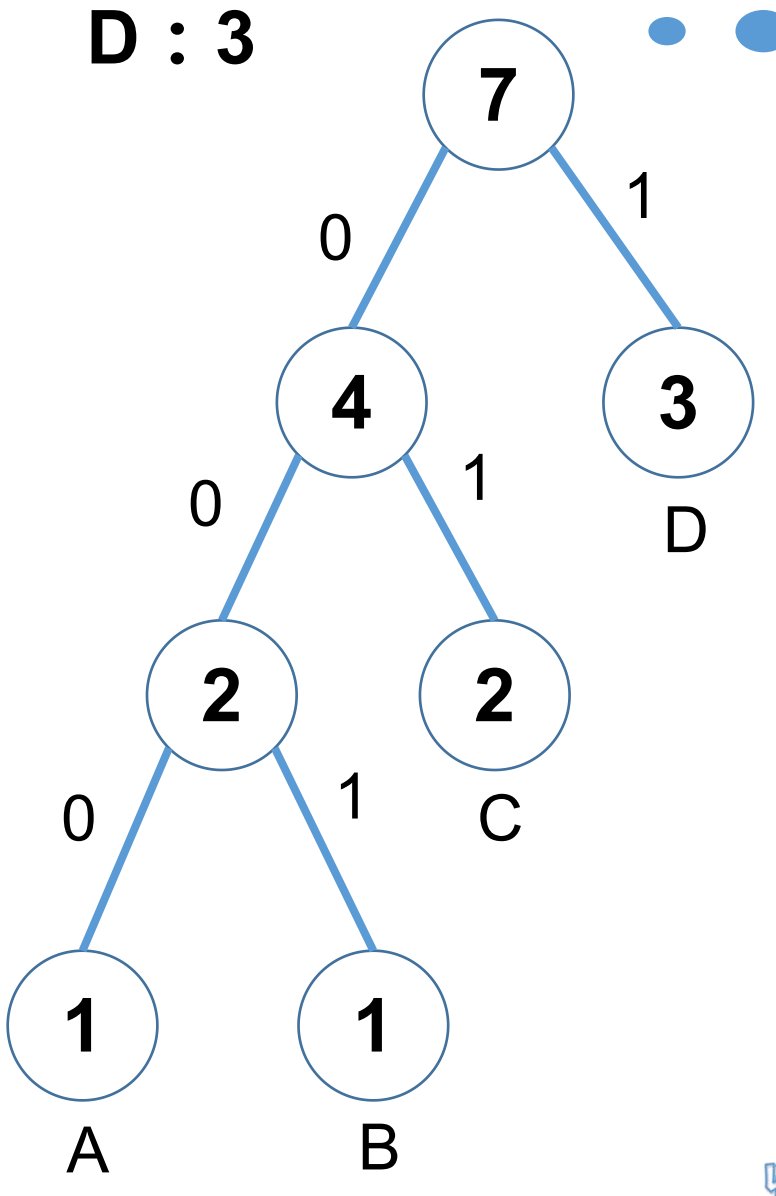
C:01

D:1

电文总长度

= 3 + 3 + 2 * 2 + 3 * 1

= 13



编程兔教育

并查集

树的应用——并查集

并查集是一种多叉树，用于处理一些不相交集合的合并及查询问题。

- **常用操作：**
  - **初始化：每个结点单独作为一个集合。**
  - **查询：求元素所在的集合的代表元素，即根结点。**
  - **合并：将两个元素所在的集合，合并为一个集合。**
  - **合并之前，应先判断两个元素是否属于同一集合，用上面的“查询”来实现。**

编程兔教育

例：P1551 亲戚

【题目背景】

若某个家族人员过于庞大，要判断两个是否是亲戚，确实还很不容易，现在给出某个亲戚关系图，求任意给出的两个人是否具有亲戚关系。

【题目描述】

规定：x和y是亲戚，y和z是亲戚，那么x和z也是亲戚。如果x,y是亲戚，那么x的亲戚都是y的亲戚，y的亲戚也都是x的亲戚。

【输入输出格式】

输入格式：

第一行：三个整数n,m,p，（n<=5000,m<=5000,p<=5000），分别表示有n个人，m个亲戚关系，询问p对亲戚关系。

以下m行：每行两个数Mi，Mj，1<=Mi，Mj<=N，表示Mi和Mj具有亲戚关系。

接下来p行：每行两个数Pi，Pj，询问Pi和Pj是否具有亲戚关系。

输出格式：

P行，每行一个'Yes'或'No'。表示第i个询问的答案为"具有"或"不具有"亲戚关系。

【输入输出样例】

输入样例#1：	输出样例#1：
6 5 3	Yes
1 2	Yes
1 5	No
3 4	
5 2	
1 3	
1 4	
2 3	
5 6	

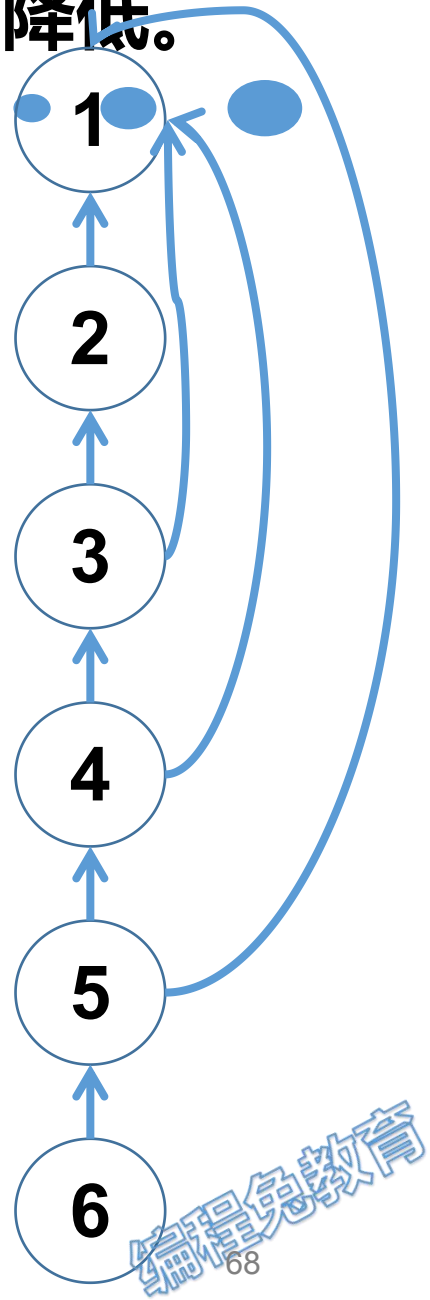
编程兔教育

```
int f[maxn];
//初始化：f[i] = 0;或f[i] = i;都可以

//查询，递归实现
int findf(int k){
  if(f[k] == 0) return k;  //初始化f[k] = 0
  //if(f[k] == k) return k;  //初始化f[k] = k
  return findf(f[k]);
}

//查询，非递归实现
int findf(int k){
  while (f[k] !=0 ) k=f[k]; //初始化f[k] = 0
  return k;
}

//合并
void union(int x, int y){
  int fx = findf(x), fy = findf(y);
  if(fx != fy) f[fx] = fy;
}
```

编程兔教育

- 朴素的并查操作可能使得多叉树退化成线性，从而导致效率的降低。
- 改进措施：路径压缩
- int findf(int k){
- if(f[k] == 0) return k;
- int ans = findf(f[k]);
- f[k] = ans;
- return ans;
- }

```
int findf(int k){
        return f[k] == 0 ? k : f[k] = findf(f[k]);
}
```



编程兔教育

```
#include <cstdio>
const int maxn=5010;
int n,m,p,x,y;
int f[maxn];
int findf(int k){
        return f[k]==0 ? k : f[k] = findf(f[k]);
}
int main(){
        scanf("%d%d%d",&n,&m,&p);
        for(int i=0;i<m;i++){
                scanf("%d%d",&x,&y);
                int fx=findf(x),fy=findf(y);
                if(fx!=fy) f[fx]=fy;
        }
        for(int i=0;i<p;i++){
                scanf("%d%d",&x,&y);
                int fx=findf(x),fy=findf(y);
                if(fx==fy) printf("Yes\n");
                else printf("No\n");
        }
        return 0;
}
```

編程兔教育

Thanks for listening !