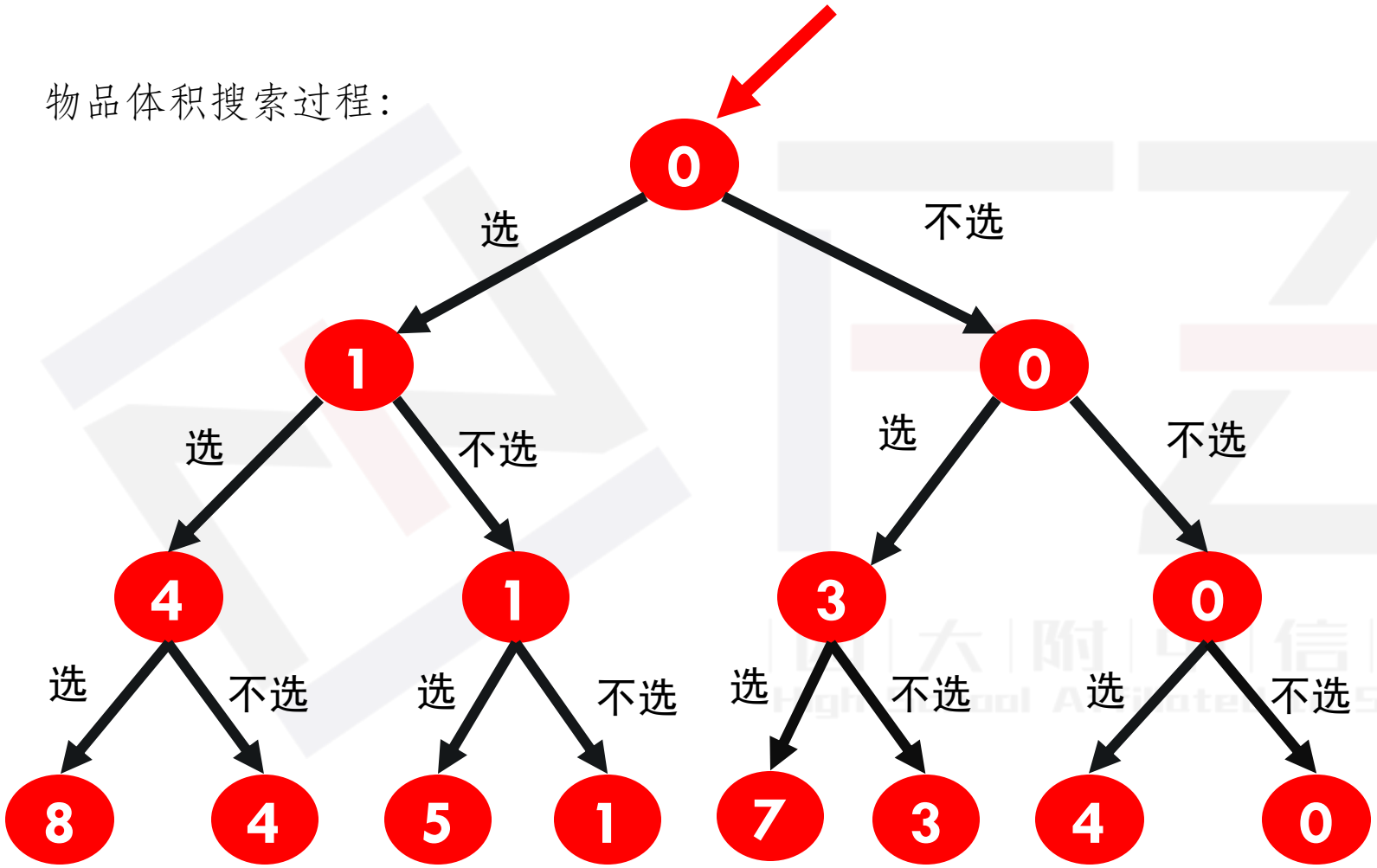
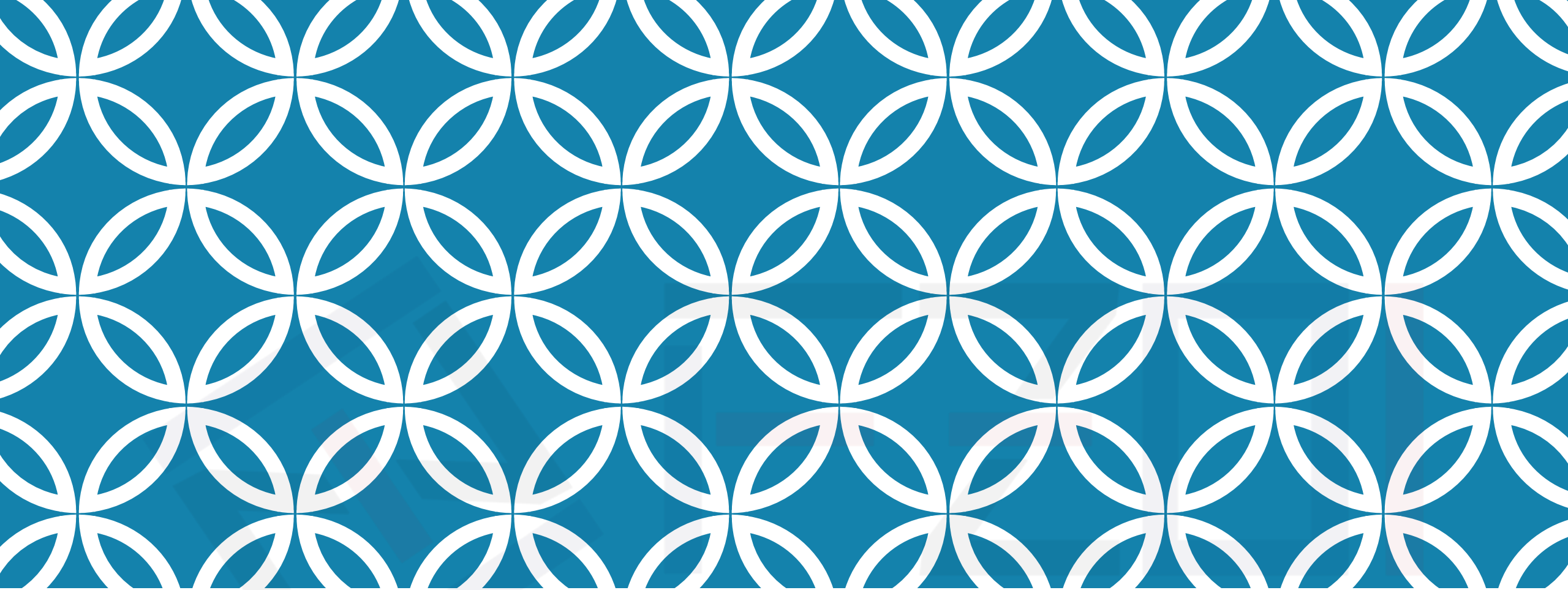
还记得DFS搜索的过程吗?

物品体积搜索过程:



DFS搜索树

树形结构



树与二叉树

QYC

树(TREE)

什么是树?

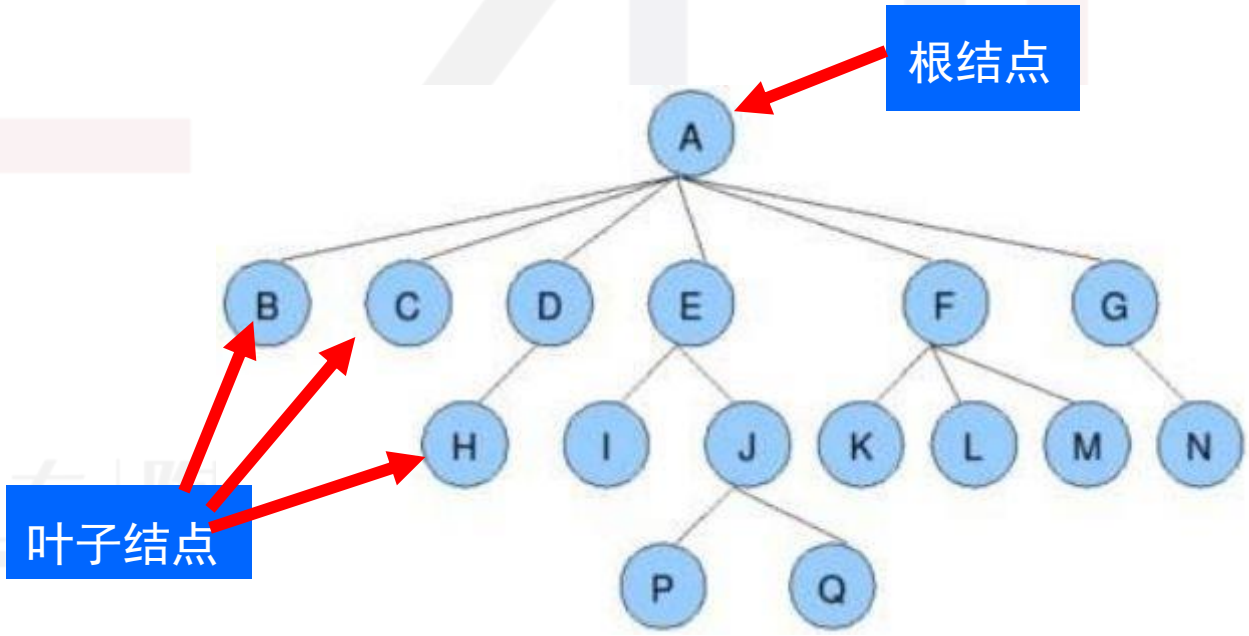
树是一种非线性数据结构，可以很好的描述分支好层次特性的数据集合。

每个元素称为结点。

没有前驱的结点称为根结点

除了根结点，每个元素**有且仅有一个前驱**

没有后继的结点称为叶子结点



树的相关名词

结点——包含一个数据元素

结点的度——结点含有的子树个数

树的度——树中所有结点度的最大值

根结点——只有后继，没有前驱

叶结点——度为零的结点

分支结点——除叶结点以外的结点

子树——树的一棵分支

孩子结点——含有的子树的根节点

双亲节点或父节点——若一个节点含有子节点，则这个节点称为其子节点的父节点

A的度为6

树的度为6

根结点为A

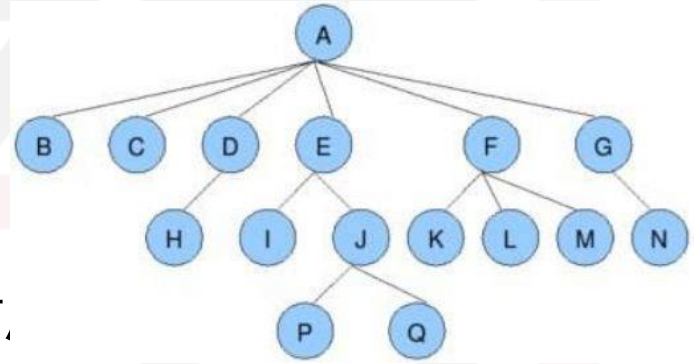
B, C, H, …为叶结点

D, E, J, …为分支结点

E的子树为I和JPQ

E的孩子结点为I，J

E的父结点为A



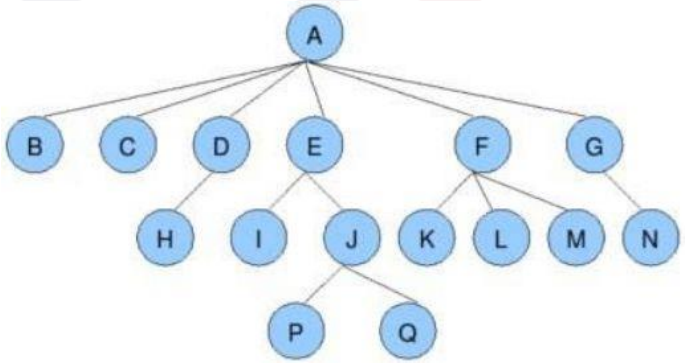
树的相关名词

子孙结点——某结点为根的子树上的所有结点都是它的子孙

E的子孙结点：I, J, P, Q

祖先结点——从根结点到某结点的路径上的任意结点都是该结点的祖先

P的祖先结点：J, E, A



兄弟结点——有共同的父结点的结点。

I，J为兄弟结点

堂兄弟结点——双亲在同一层

B, C, D为堂兄弟结点

层数——结点的层次，根为第一层

E的层次是2，J是3，P是4

深度（高度）——结点的最大层次

该树的深度为4

路径——从根到该结点所经分支和结点。

A到K的路径：A-F-K

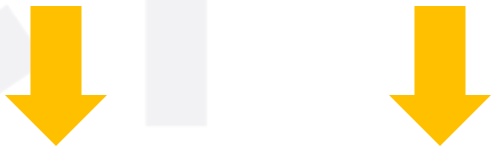
森林——m棵互不相交树的集合

树的存储

方法一： 父亲表示法

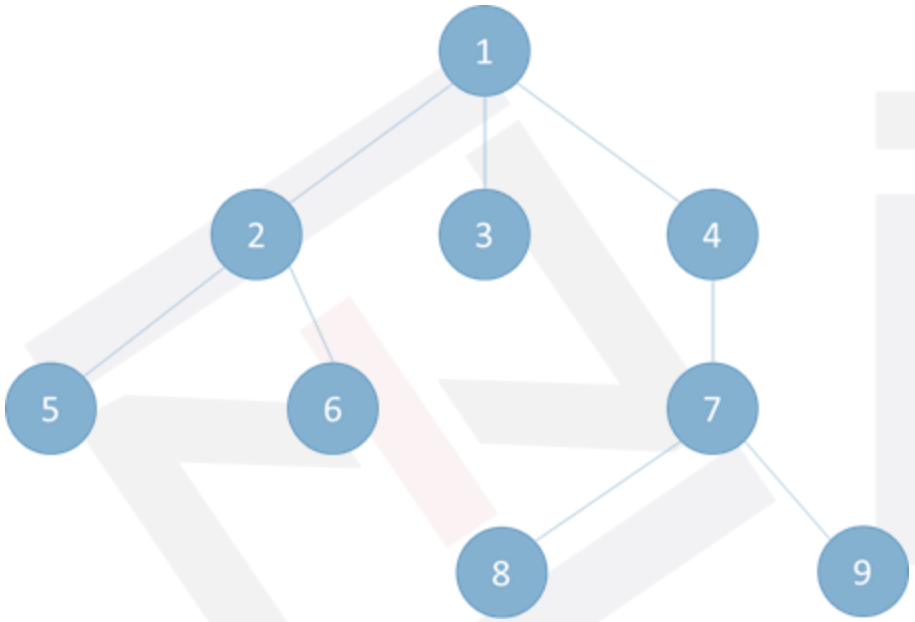
将每个结点的父亲记录下来

int data[MAX]，parent[MAX]



第i个结点的信息　第i个结点的父亲

树的存储



下标	data	parent
1	1	0
2	2	1
3	3	1
4	4	1
5	5	2
6	6	2
7	7	4
8	8	7
9	9	7

优点：

利用了树中除根结点外每个结点都有唯一的父结点这个性质。很容易找到树根。

缺点：

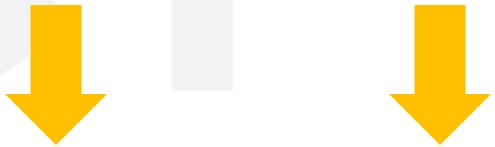
找孩子时需要遍历整个线性表。

树的存储

方法二： 孩子表示法

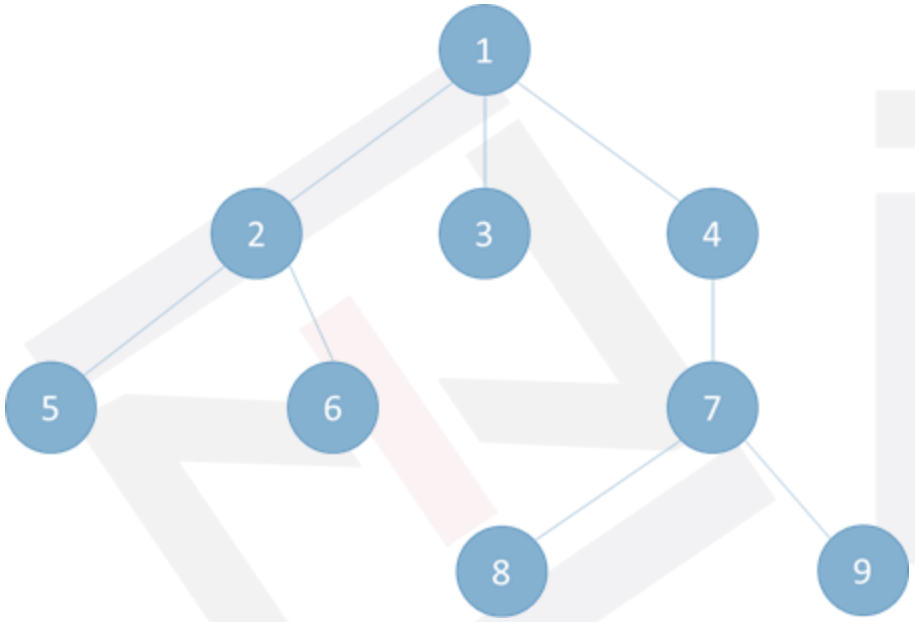
将每个结点的孩子记录下来

```
int M=5;      //树的度
int data[MAX],child[MAX][M];
```



第i个结点的信息　第i个结点的孩子结点

树的存储



结点下标i	1	2
data[i]	1	2
child[i][MAX]	2 3 4 …	5 6 … …

Q:
可以发现某些点可能会有许多孩子结点，能否优化存储？

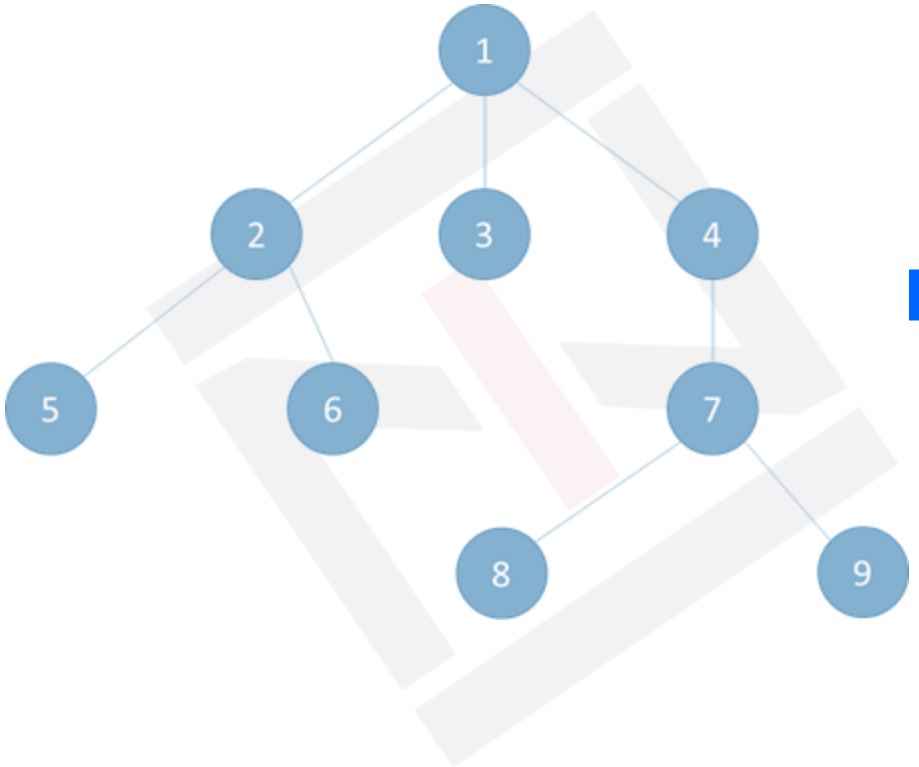
优点：

寻找孩子方便，利于从上往下遍历整棵树

缺点：

找父亲困难，只能从根（父）结点遍历到子结点，不能从某个子结点返回到它的父结点。并且存在一定空间的浪费

存储优化—链表



下标i	data[i]	child[]
1	1	→ 2 → 3 → 4
2	2	→ 5 → 6
3	3	^
4	4	→ 7
5	5	^
6	6	^
7	7	→ 8 → 9
8	8	^
9	9	^

树的孩子链
与图论里的邻接表类似

树的存储

方法三： 兄弟孩子表示法

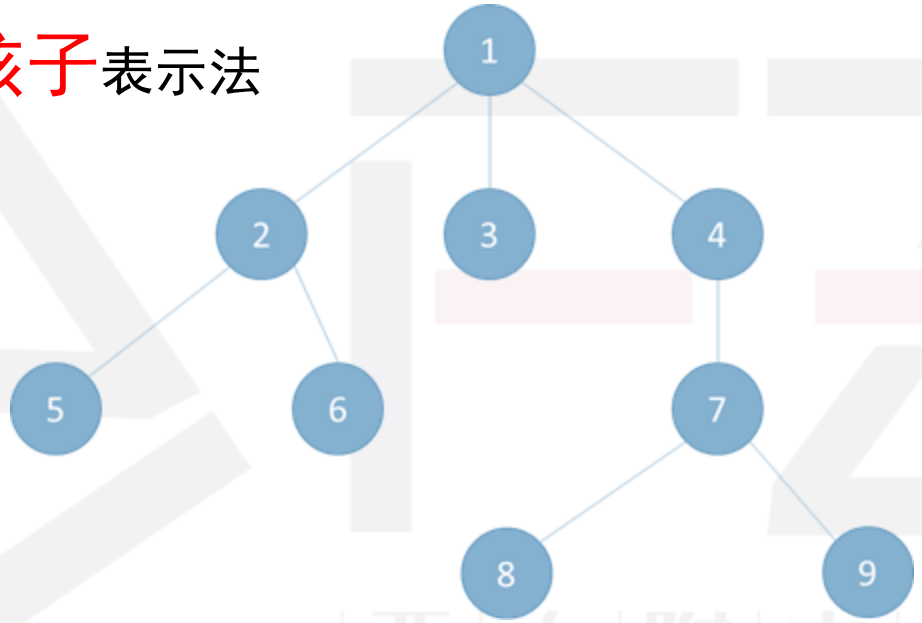
将每个结点的孩子记录下来

int data[MAX] ,firstchild[MAX], firstbro[MAX];

第i个结点的信息　第i个结点的第一个孩子　第i个结点的第一个兄弟

树的存储

方法三： **兄弟孩子**表示法



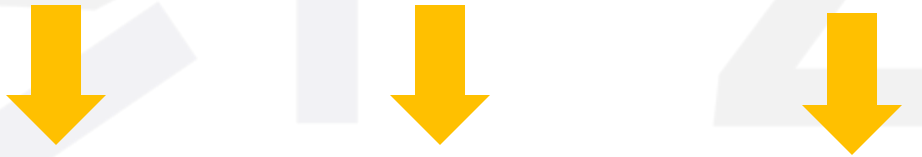
下标	1	2	3	4	5	6	7	8	9
child	2	5	0	7	0	0	8	0	0
firstbro	0	3	4	0	6	0	0	9	0

树的存储

方法四： 父亲孩子表示法

将每个结点的孩子记录下来

int data[MAX] ,child[MAX], parent[MAX]；



第i个结点的信息　第i个结点的孩子　第i个结点的父亲

例题：找树根和孩子

【题目描述】给定一棵树，输出树的根root，孩子最多的结点max以及他的孩子。

【输入】
第一行：n（结点个数<=1000），m（边数<=200）。
以下m行；每行两个结点x和y，表示y是x的孩子(x,y<=1000)。

【输出】
第一行：树根：root。
第二行：孩子最多的结点max。
第三行：max的孩子。

【样例输入】
8 7
4 1
4 2
1 3
1 5
2 6
2 7
2 8

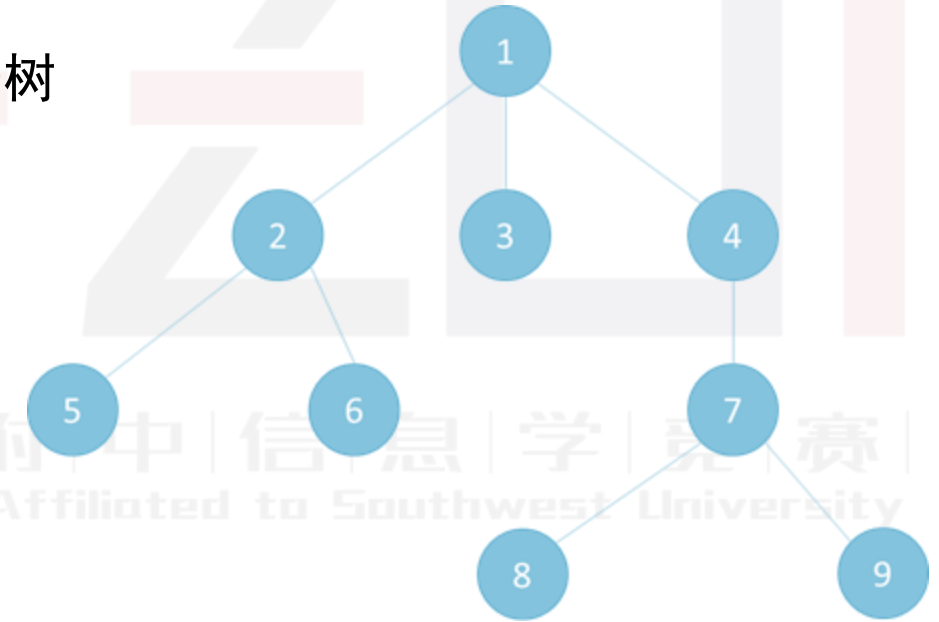
树的遍历

1. 先序遍历

先访问根结点，再从左到右按照先序思想遍历各棵子树

如右图结果为：

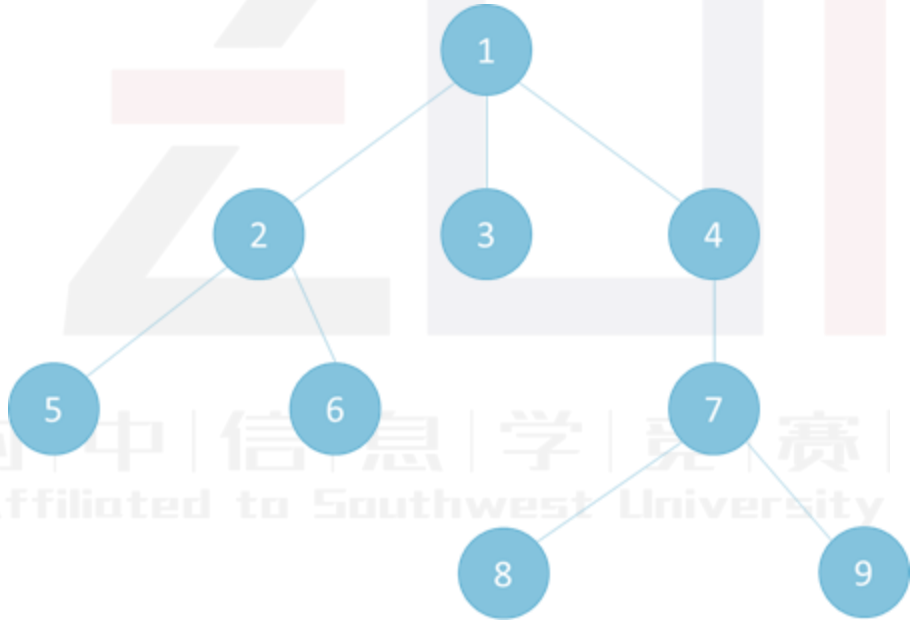
125634789



树的遍历

2. 后序遍历

先从左到右遍历各棵子树，再访问根结点。



如图结果为：

562389741

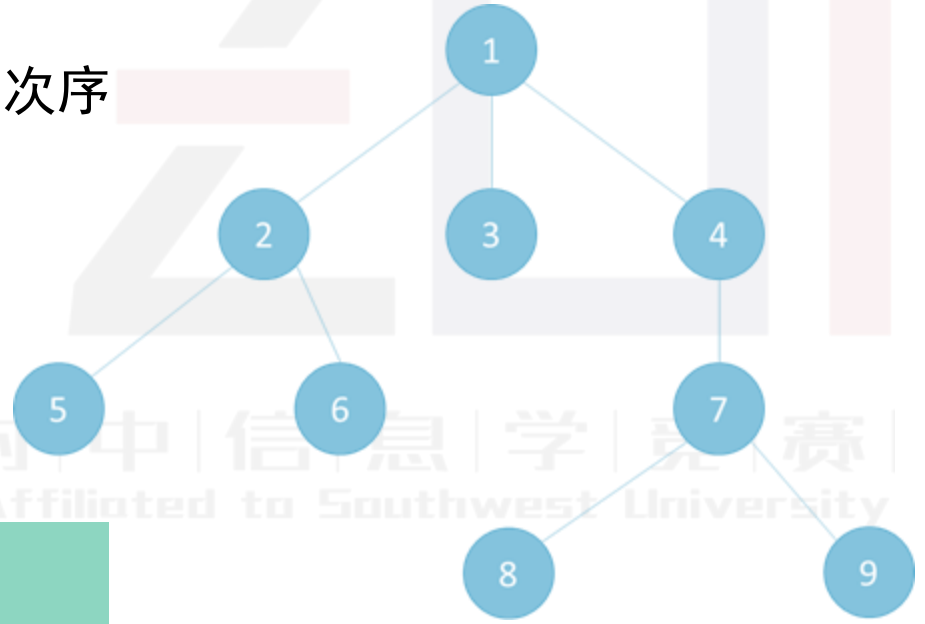
树的遍历

3. 层序(次)遍历

按层次从小到大逐个访问，同一层次按照从左到右的次序

如图结果为：

123456789

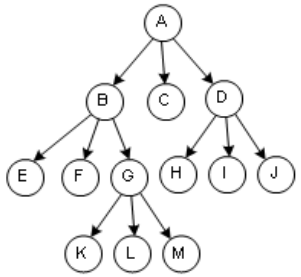


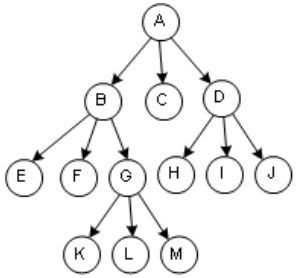
注意：
该遍历方式使用较多
思考：适合使用什么方式存储树?
孩子表示法

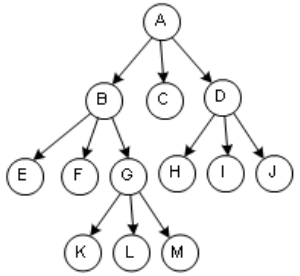
森林

题目若给出多组树形关系

就应当建立多棵树，多棵树就组成了“森林”







二叉树(BINARY TREE)

重点

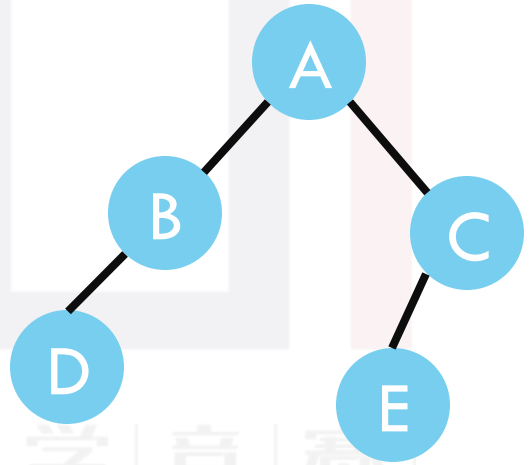
什么是二叉树?

有一句话曾说：“如果计算机世界有树，那树的枝条的必定分二叉”

虽然树可以分三叉、四叉...n叉，但是二叉树由于其特殊的性质，特别适合程序设计。

什么是二叉树?

二叉树（binary tree）是n(n>=0)个结点的有限集，它或为空树(n=0)，或由一个根结点和两棵分别称为左子树和右子树的互不相交的二叉树构成。

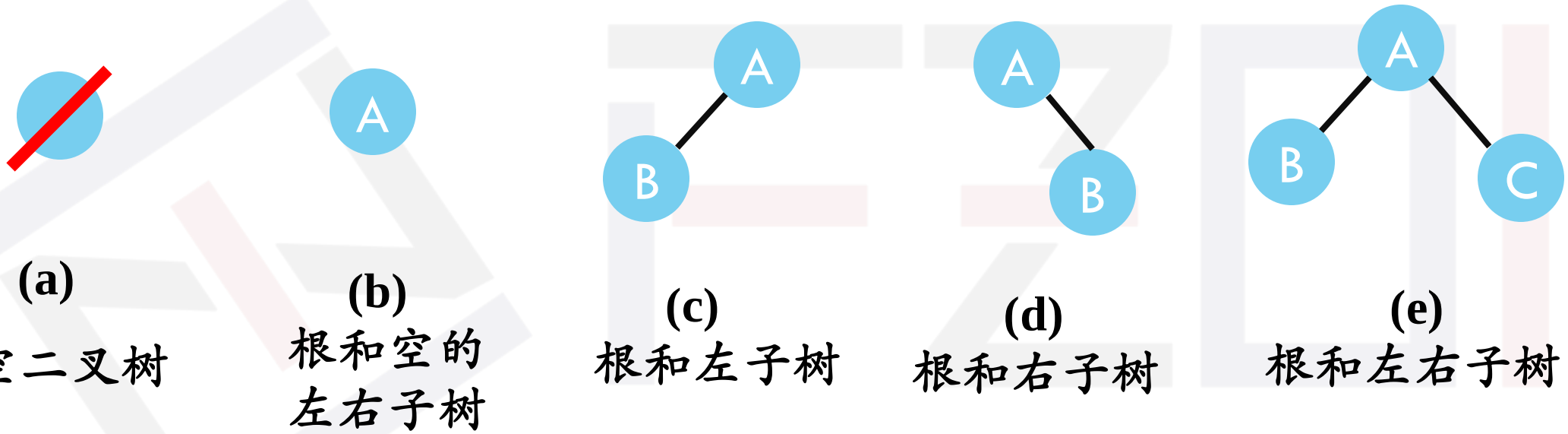


特点

1. 每个结点至多有2棵子树，即度<=2
2. 二叉树的子树有序，且有左、右之分，且其次序不能任意颠倒

二叉树本身是递归定义的。

二叉树



(a)
空二叉树

(b)
根和空的
左右子树

(c)
根和左子树

(d)
根和右子树

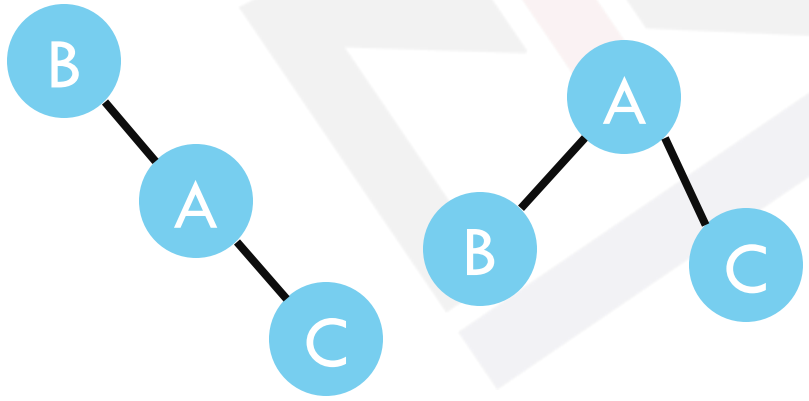
(e)
根和左右子树

注意：树不区分左右子树，而二叉树严格区分左右子树

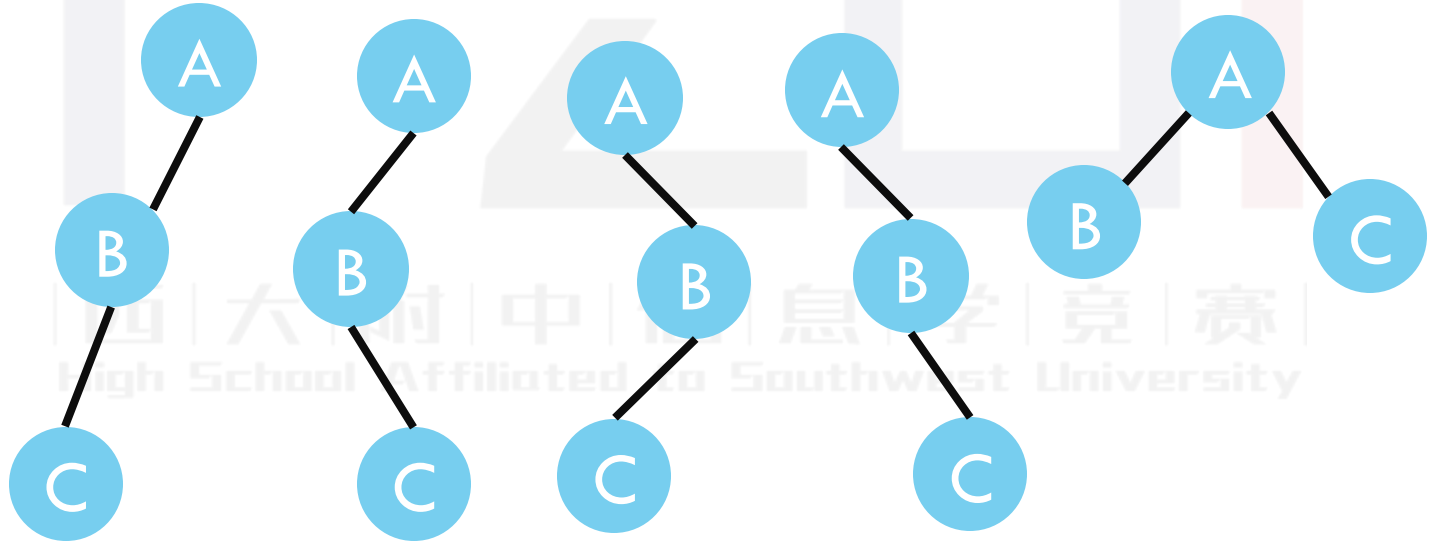
树与二叉树

3个结点构成的树可能有多少形态，3个结点构成的二叉树呢？

三个结点的树



三个结点的二叉树



二叉树性质

性质1　在二叉树的第i层上至多有2^{i-1}个结点(i>=1)

用数学归纳法证明

① 当i=1时，仅根结点，$2^{i-1}=2^0=1$,成立

② 设i=j时成立，即j层上至多有2^{j-1}个结
由于二叉树结点的度最大为2，故在第j+1层上最大结点数为第j层上最大结点数的2倍，即$2\times2^{j-1}=2^j$。故i=j+1时成立

由1、2得证

二叉树性质

性质2 一棵深度为k的二叉树中，最多具有2^k-1个结点

证明：由性质1，第i层上的结点数最多为2^{i-1}，k层的二叉树的结点数最多为

$$\sum_{i=1}^{k} 2^{i-1} = 2^k - 1$$

二叉树性质

性质3 任何一棵二叉树，如果其叶子结点数为n0，度为2的结点数为n2，则n0＝n2＋1。

设结点数为n， $n = n_0 + n_1 + n_2$ （ n_1为度为1的结点数)

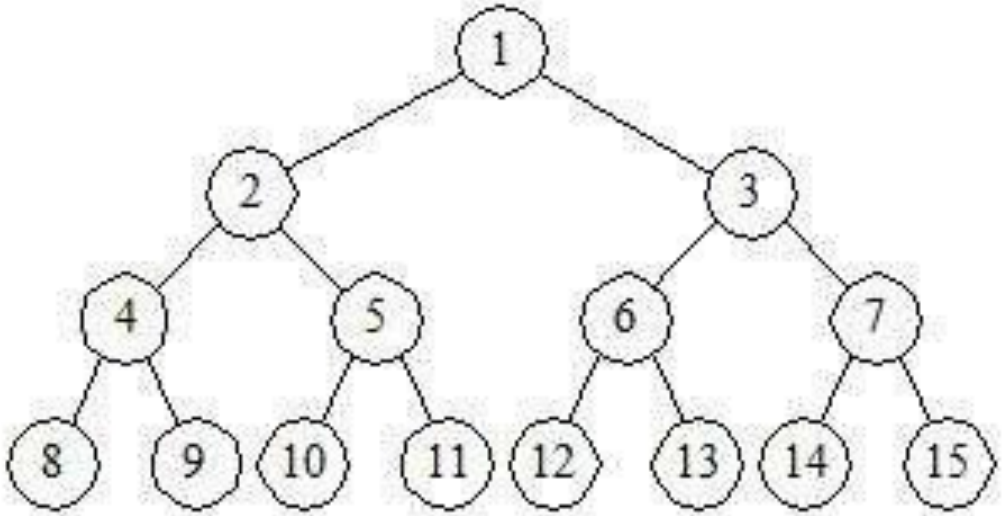
另一方面，度为1的结点有一个孩子n_1，为2的结点有2*n_2个孩子根结点没有孩子，总数也可以表示：

$$n = n_1 + 2 * n_2 + 1$$

得证，$n_0 = n_2 + 1$

满二叉树

二叉树中每一层的结点数都达到最大

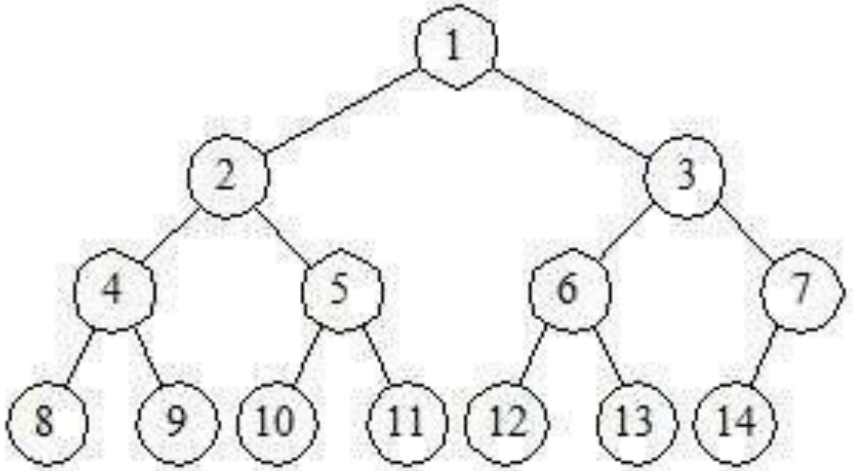


(a) 满二叉树

完全二叉树

除最后一层外，其余各层都是满的；

最后一层或者是满的，或者是**右边缺少连续**的若干结点。



(b) 完全二叉树

完全二叉树的性质

具有n个结点的完全二叉树的深度为$\mathbf{floor}(\log_2 n)+1$

设二叉树的深度为k，结点数为n

$2^{k-1}-1 < n <= 2^k-1$（据性质2及完全二叉树的定义）

$2^{k-1} <= n < 2^k$

取对数得：$k-1 <= \log_2 n < k$

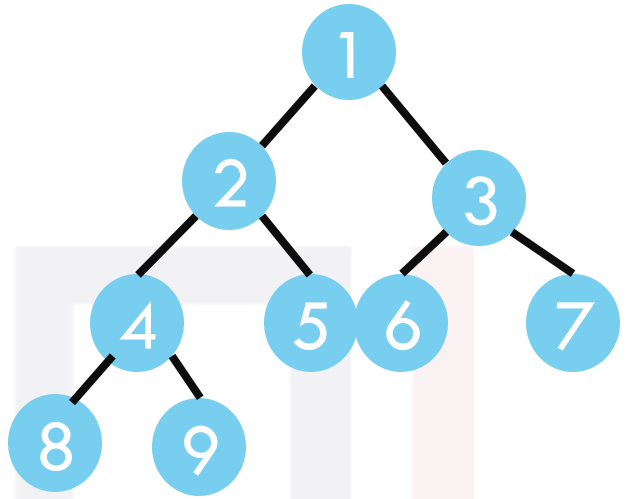
$\mathbf{floor}(\log_2 n)=k-1$

得证：$k=\mathbf{floor}(\log_2 n)+1$

完全二叉树的性质

对有n个结点的完全二叉树，按顺序（1~n）对结点进行编号。结点i满足以下性质

完全二叉树的性质



1）如果i=1，则结点i是根

如果i>1，则其双亲parent(i)是结点i/2

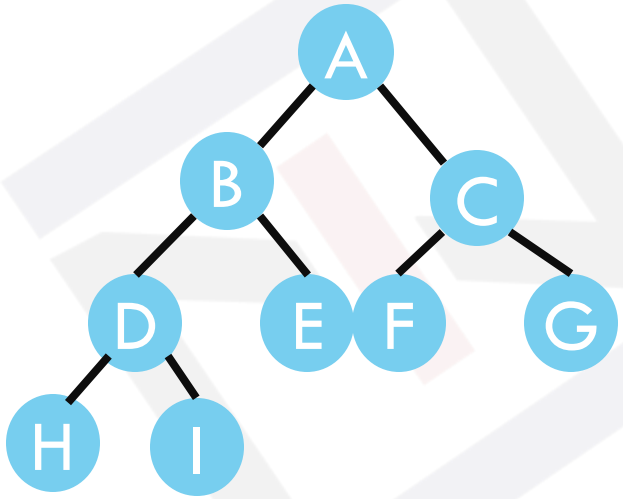
2）如果2i>n，则结点i为叶子，否则其左孩子是结点2i

3）如果2i+1>n，则结点i无右孩子，否则其右孩子是结点2i+1

Q:我们是否可以利用这个性质来方便地储存它?

完全二叉树的性质

完全二叉树可用一维数组存储



1	2	3	4	5	6	7	8	9	10
A	B	C	D	E	F	G	H	I	

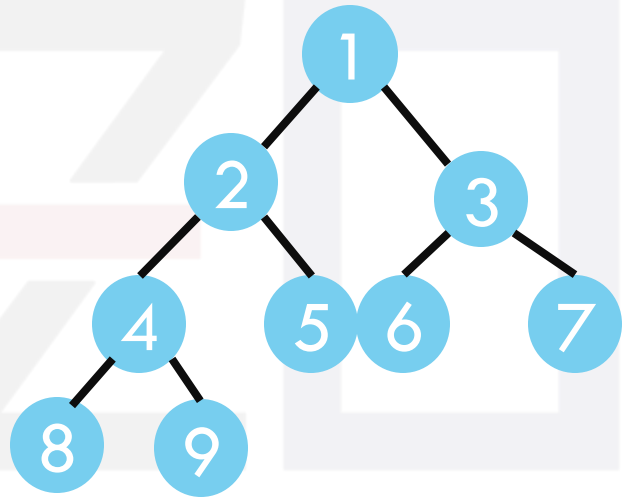
由于编号就代表位置，所以完全化建立二叉树：

```
for(i=1;i<=n;i++)
 cin>>tree[i];
```

完全二叉树的性质

tree[i] —— 编号i的结点

由下标可推出结点间的层次关系



tree[1]：根

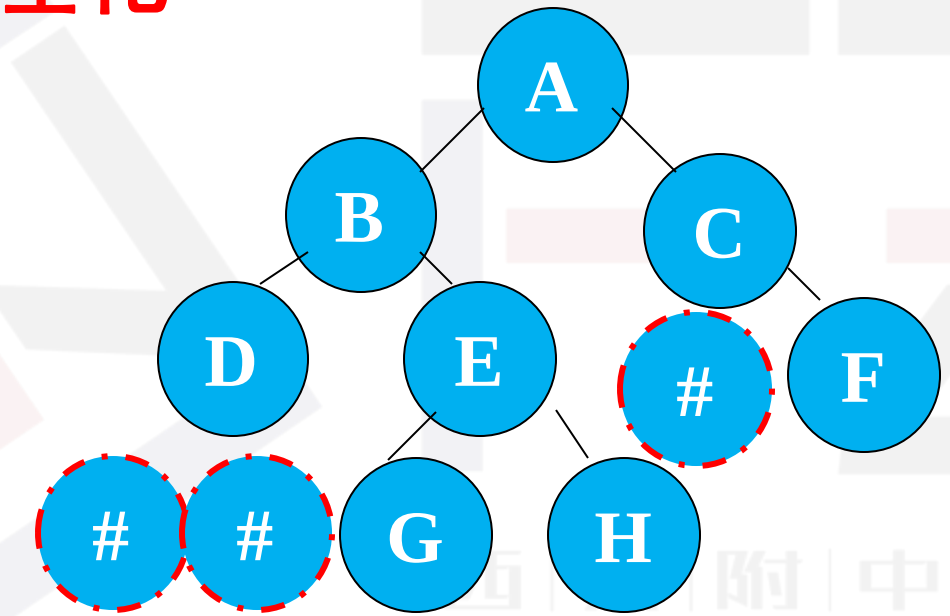
tree[i]的双亲：tree[i/2]

tree[i]的左孩子： tree[2*i]

tree[i]的右孩子： tree[2*i+1]

数组顺序存储

方案一：完全化



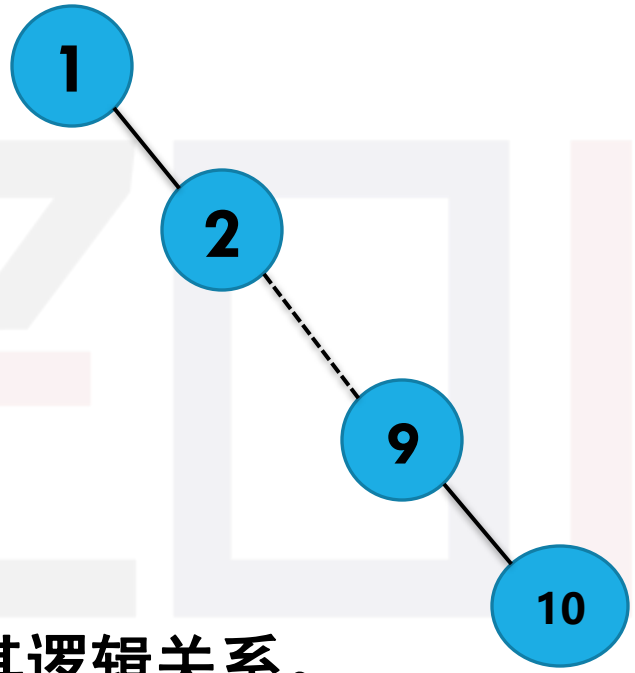
A	B	C	D	E	#	F	#	#	G	H	
1	2	3	4	5	6	7	8	9	10	11	

完全化存储

思考：这样一棵有10个结点的二叉树需要多少个单位的存储空间?

$2^{10}-1$

其中$2^{10}-1$ – 10个填充字符



- **优点：** 直接利用元素在数组中的位置(下标)表示其逻辑关系。
- **缺点：** 若不是完全二叉树, 则会浪费空间。
- 因此，完全化适合于(接近)完全二叉树

数组顺序存储

方案二：孩子表示法

记录孩子节点

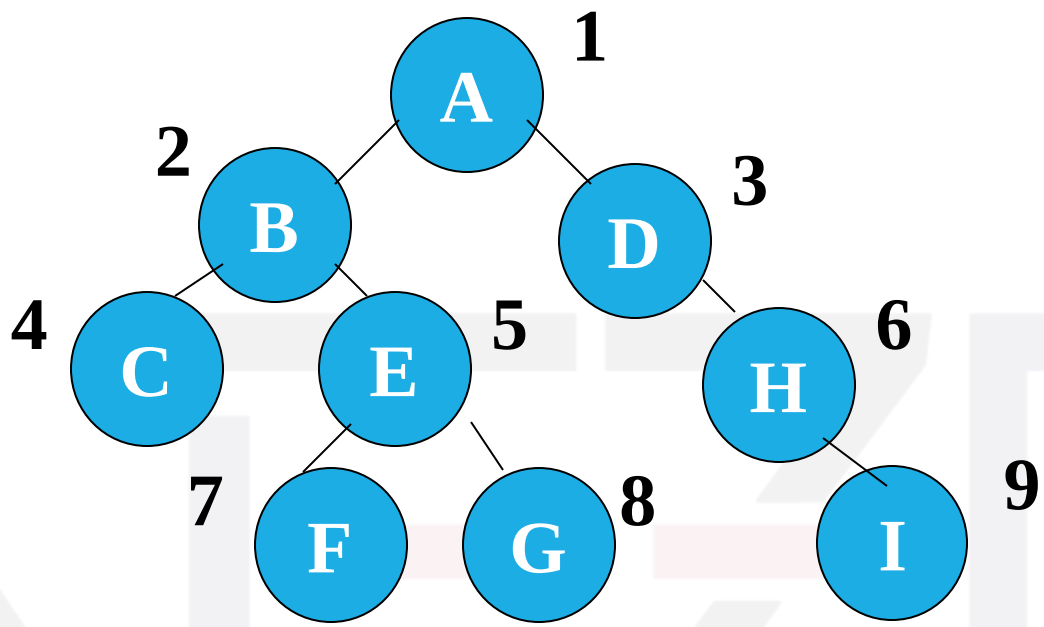
```
int data[MAX] ,lchild[MAX], rchild[MAX];
```

方案三：孩子父亲表示法

记录孩子与父亲节点

```
int data[MAX] ,lchild[MAX], rchild[MAX],parent[MAX]
```

数组存储



	1	2	3	4	5	6	7	8	9	
Data	**A**	**B**	**D**	**C**	**E**	**H**	**F**	**G**	**I**	**...**
lchild	**2**	**4**	**0**	**0**	**7**	**0**	**0**	**0**	**0**	**...**
rchild	**3**	**5**	**6**	**0**	**8**	**9**	**0**	**0**	**0**	**...**

遍历二叉树

遍历二叉树

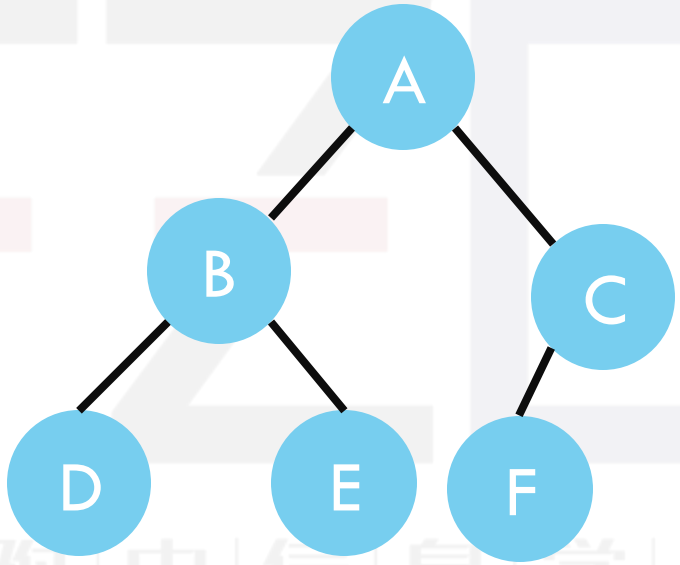
1. 先序遍历

若二叉树为空，则空操作；否则：

（1）访问根节点；

（2）先序遍历左子树；

（3）先序遍历右子树；



如图结果为：

ABDECF

遍历二叉树

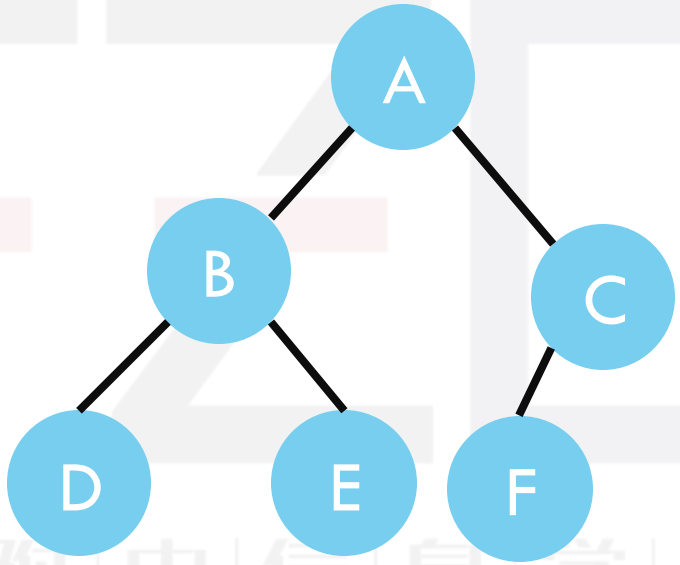
2. 中序遍历

若二叉树为空，则空操作；否则：

（1）中序遍历左子树；

（2）访问根节点；

（3）中序遍历右子树；



如图结果为：

DBEAFC

遍历二叉树

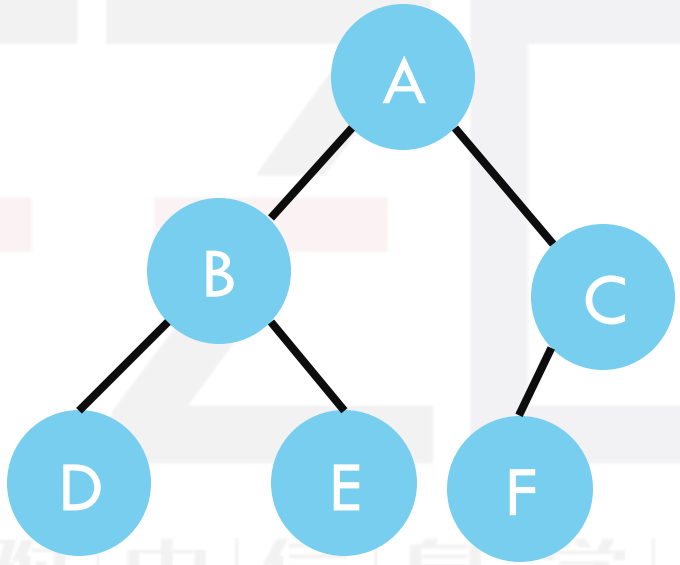
3. 后序遍历

若二叉树为空，则空操作；否则：

（1）后序遍历左子树；

（2）后序遍历右子树；

（3）访问根节点；



如图结果为：

DEBFCA

三类遍历程序代码

```
void preorder(int root){ //先根序

    if(a[root]==-1)return ;
    cout<<a[root]<<" ";
    preorder(root*2);
    preorder(root*2+1);

}
```

```
void lastorder(int root){ //后根序
    if(a[root]==-1)return ;
    lastorder(root*2);
    lastorder(root*2+1);
    cout<<a[root]<<" ";

}
```

```
void inorder(int root){ //中根序

    if(a[root]==-1)return ;
    inorder(root*2);
    cout<<a[root]<<" ";
    inorder(root*2+1);

}
```

```
int main(){
    memset(a,-1,sizeof(a));
   int n=5;
   for(int i=1;i<=n;i++) a[i]=i;  //完全化建立
   //调用函数
   return 0;
}
```

提问

某些题目会给出先、中、后、层次遍历的两种，让你建立对应的二叉树，求出其他的遍历结果

问题：已知先序和后序，是否能构建一棵唯一的二叉树?

不能

输出后序遍历

【问题描述】

输入一棵二叉树的先序和中序遍历序列，输出其后序遍历序列。

【输入格式】

共两行，第一行一个字符串，表示树的先序遍

历，第二行一个字符串，表示树的中序遍历。树的结点一律用小写字母表示

【输出格式】

仅一行，表示树的后序遍历序列。

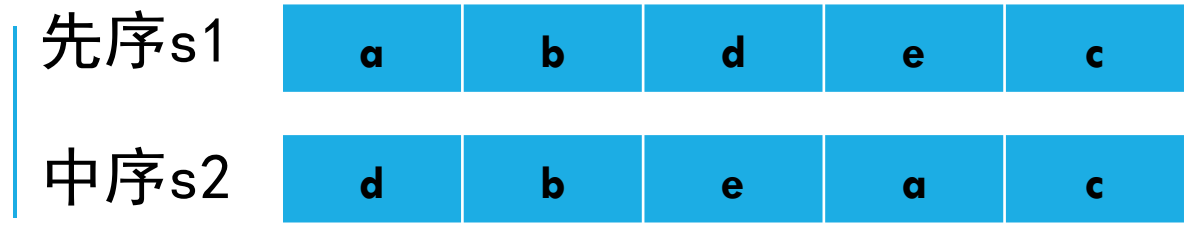
【样例输入】

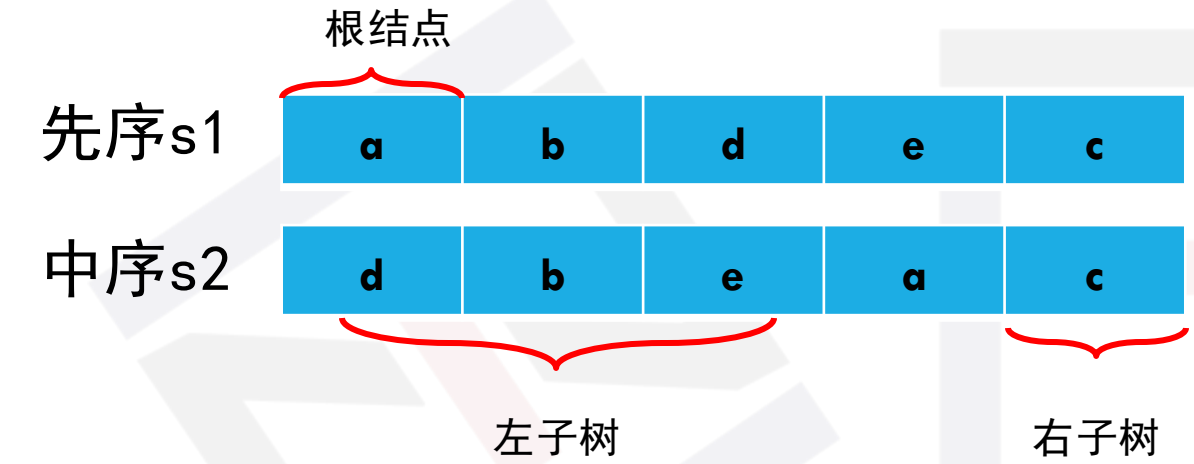
abdec

dbeac

【样例输出】

debca

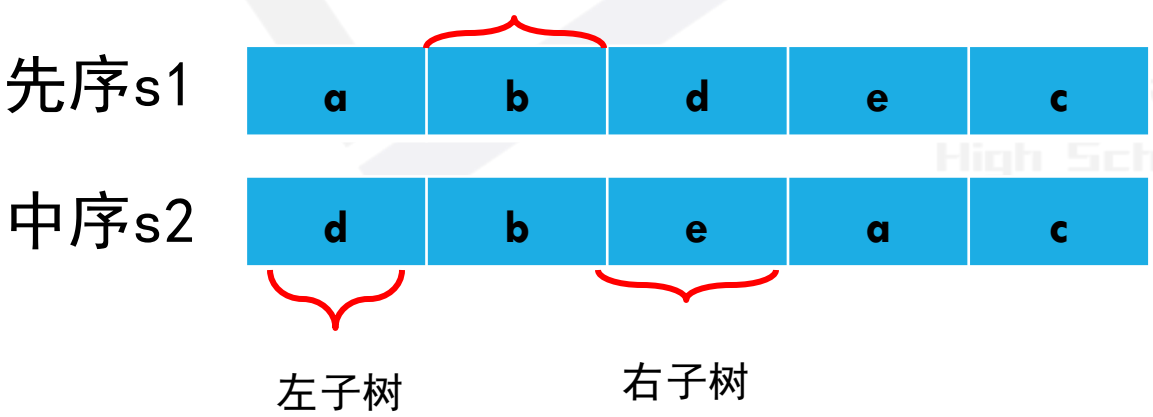




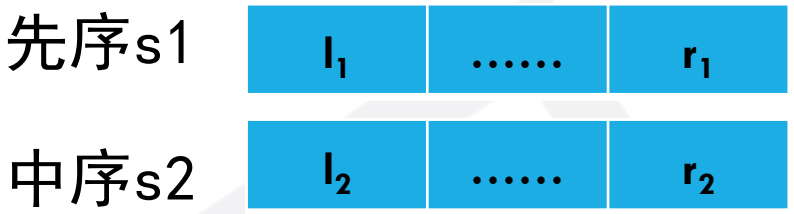
先序的第一个节点就是根节点，

中序中找到根节点的位置，根节点之前是其左子树，之后是右子树

按此顺序，依次在左子树部分遍历，右子树部分遍历

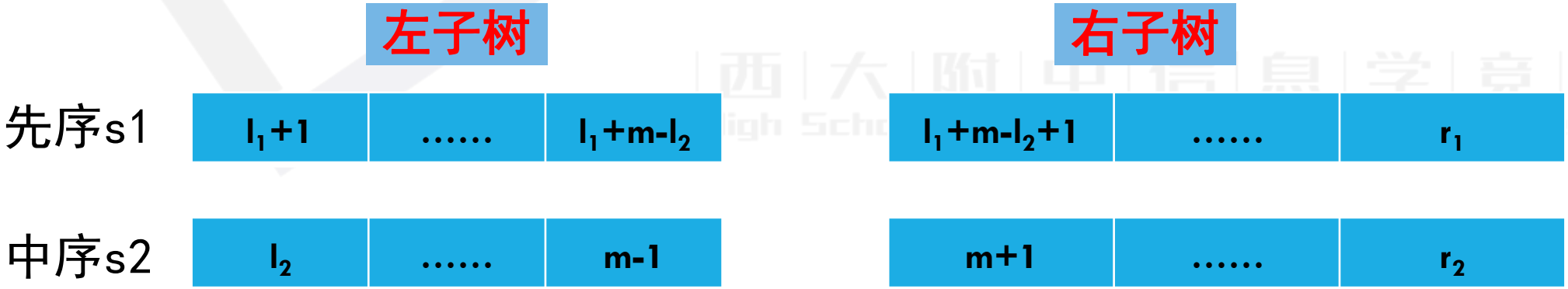


……



解题过程：

- 先序的第一个结点为根
- 在s2中找到结点l_1，位置为m
- 根的左子树在中序的位置为l_2~m-1，右子树为m+1~r_2
- 先序根结点后紧跟着为左子树，之后为右子树
- 因此，中序的l_2~m-1总共（m-l_2）个结点，对应先序的l_1+1开始的（m-l_2）结点，即l_1+1~l_1+m-l_2
- 中序的m+1~r_2对应先序的 l_1+m+l_2+1~r_1
- 左右子树遍历完毕，输出根结点s1[l_1]



递归代码

计算s1[l1…r1]为先序，s2[l2…r2]为中序的后续遍历

```
void calc(int l1, int r1, int l2, int r2){
   if(l1>r1) return;
   int m =s2中s1[l1]的位置;
   calc(l1 + 1, l1 + m - l2, l2, m - 1);
   calc(l1 + m - l2 + 1, r1, m + 1, r2);
   printf(“%c”,s1[l1]);
}
```

输出层序遍历

题目描述
给定一棵二叉树的后序遍历和中序遍历，请你输出其层序遍历的序列。这里假设键值都是互不相等的正整数。
输入格式:
输入第一行给出一个正整数N（N<=30），是二叉树中结点的个数。
第二行给出其后序遍历序列。
第三行给出其中序遍历序列。数字间以空格分隔。
输出格式:
在一行中输出该树的层序遍历的序列。数字间以1个空格分隔，行首尾不得有多余空格。

输入样例:
7
2 3 1 5 7 6 4
1 2 3 4 5 6 7
输出样例:
4 1 6 3 5 7 2

输出层序遍历

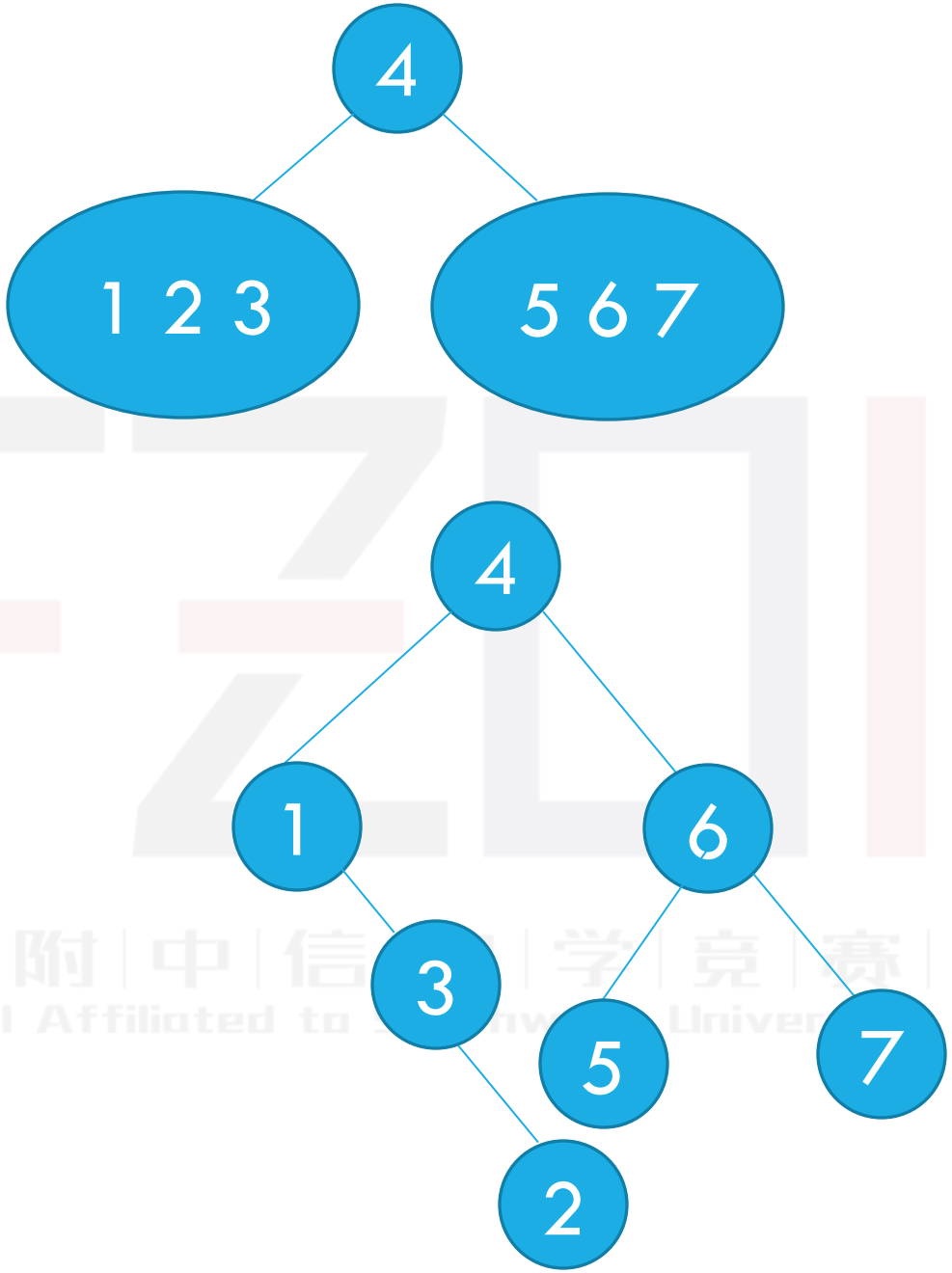
思路：
1.从后往前，在后序中找到根结点root
2.找到root在中序里的位置下标pos
3.在中序里，找到左子树和右子树的范围
4.照此过程，递归建树

2 3 1 5 7 6 4

1 2 3 4 5 6 7

左子树　　右子树

1 2 3 4 5 6 7



参考代码—建树部分

```
//结点信息定义
struct node
{
    int data,l,r;
}tree[maxn];
int last[maxn],in[maxn],cnt;
```

```
int build(int last[],int mid[],int n) {  //建树

    int root=last[n]; //每次从后序序列中拿出最后一个元素作为根结点
    if(n<=0){
        return -1;
    }
    if(n==1){
        tree[++cnt].data=mid[1];
        tree[cnt].l=tree[cnt].r=-1;
        return cnt;
    }
    tree[++cnt].data=root;
    int nex=cnt;
    int pos;
    for(pos=1;pos<=n;pos++){  //每次在中序序列中找当前在后序中拿出来的根结点对应的在中序中的位置
        if(mid[pos]==root)
            break;
    }
    tree[nex].l=build(last,mid,pos-1);  //递归找左子树
    tree[nex].r=build(last+pos-1,mid+pos,n-pos);  //递归找右子树
     //注：找右子树是对应的子树的序列为当前的右半段，即last+pos-1和mid+pos，数组名+一个数x，代表原数组的x位置~最后的那部分
    return nex;
}
```

参考代码—其他部分

```
void bfs() {  //层次遍历
    queue<int>q;
    q.push(1);
    while(!q.empty()){
        int node=q.front();
        q.pop();
        if(node!=1){
            cout<<" ";
        }
        cout<<tree[node].data;
        if(tree[node].l!=-1){
            q.push(tree[node].l);
        }
        if(tree[node].r!=-1){
            q.push(tree[node].r);
        }
    }
}
```

```
int mian(){
    int n;
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>last[i];
    }
    for(int i=1;i<=n;i++){
        cin>>in[i];
    }
    build(last,in,n);
    bfs();
    return 0;
}
```

总结

数据结构反映了一种构建数据、解决问题的思维方法，
有些时候其实并不需要建立对应的二叉树，
运用二叉树的思想也能够解决问题。

THANKS

QYC