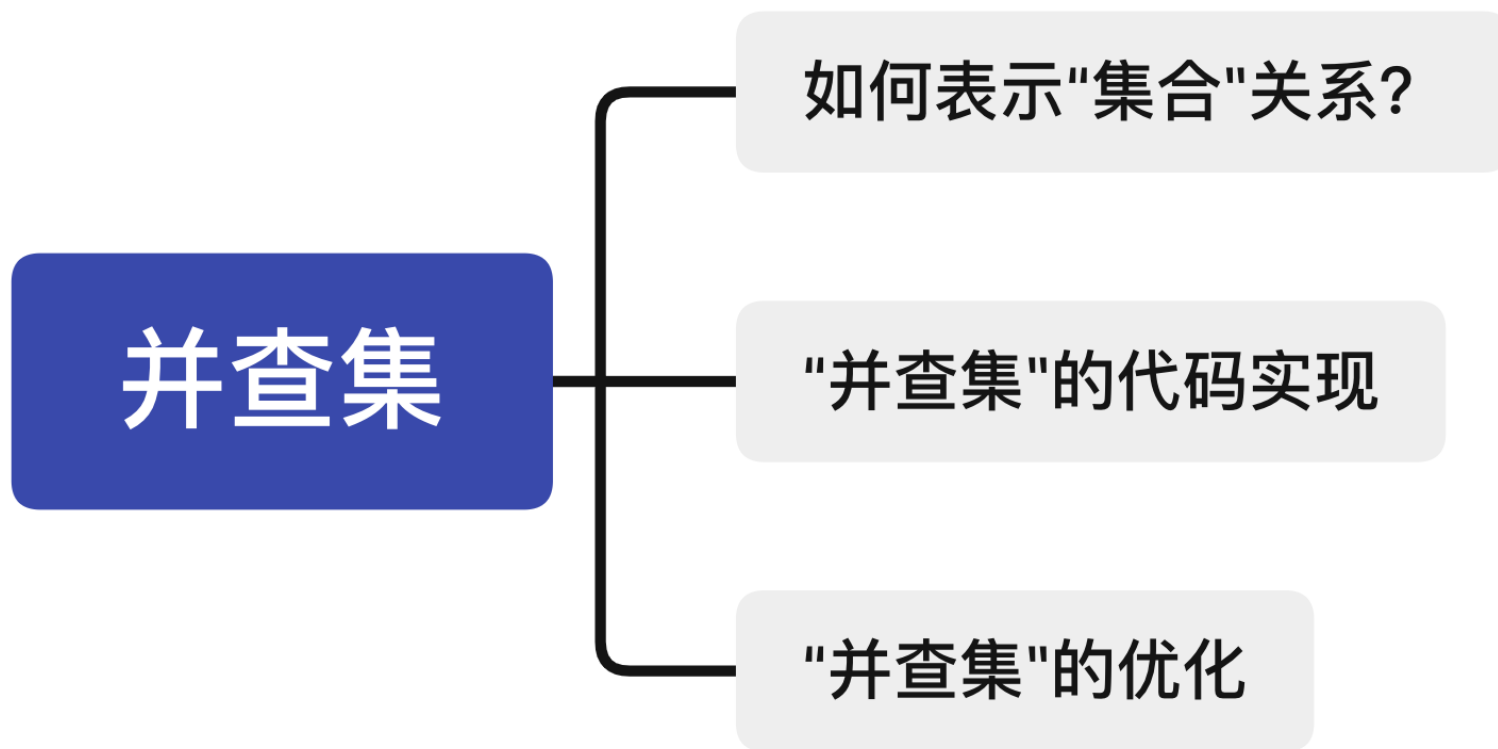


本节内容

并查集

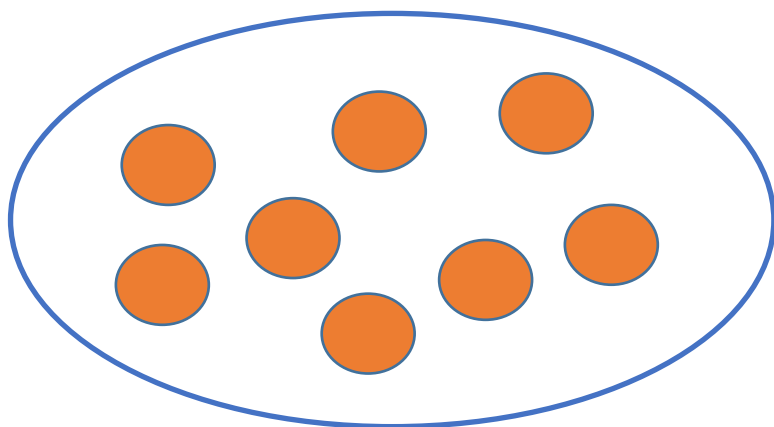
知识总览



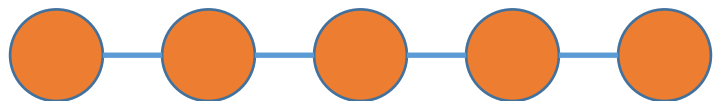
漏网之鱼：逻辑结构——“集合”



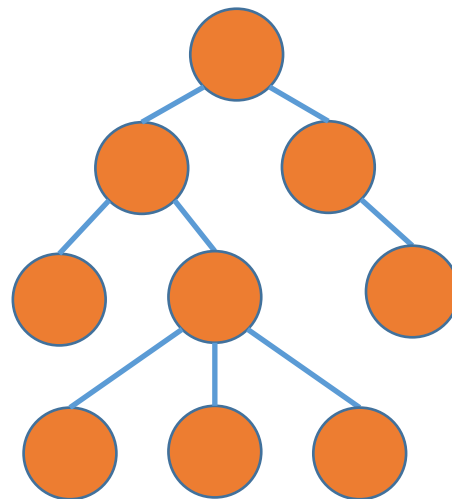
逻辑结构——数据元素之间的逻辑关系是什么？



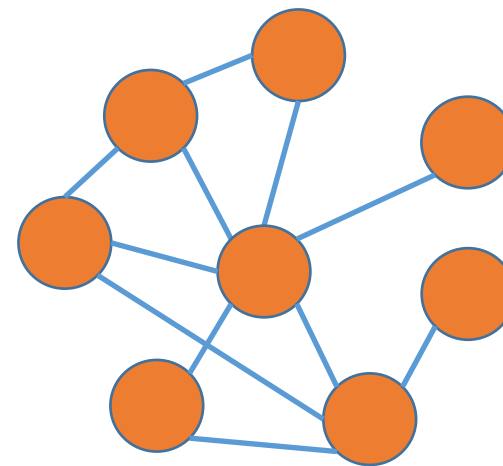
集合



线性结构

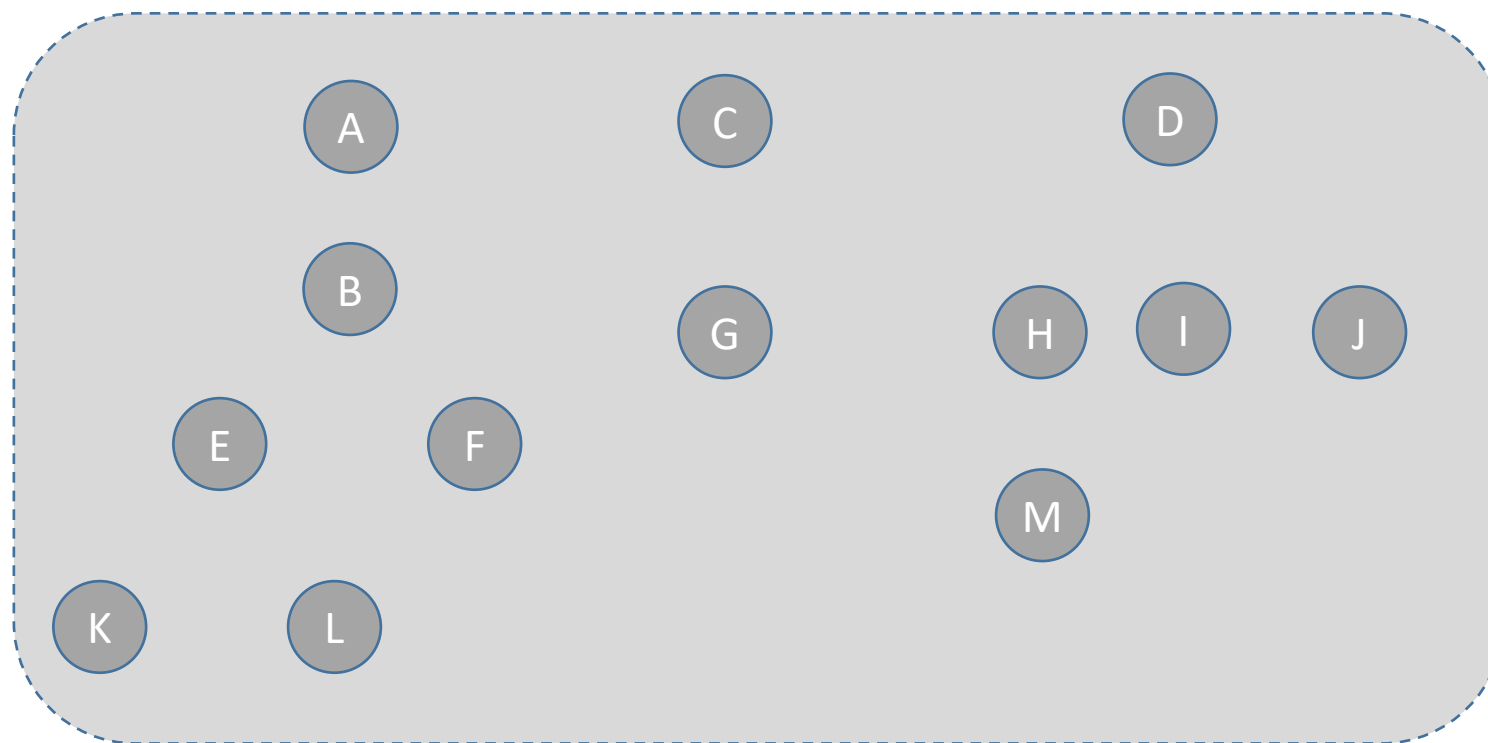


树形结构



图结构

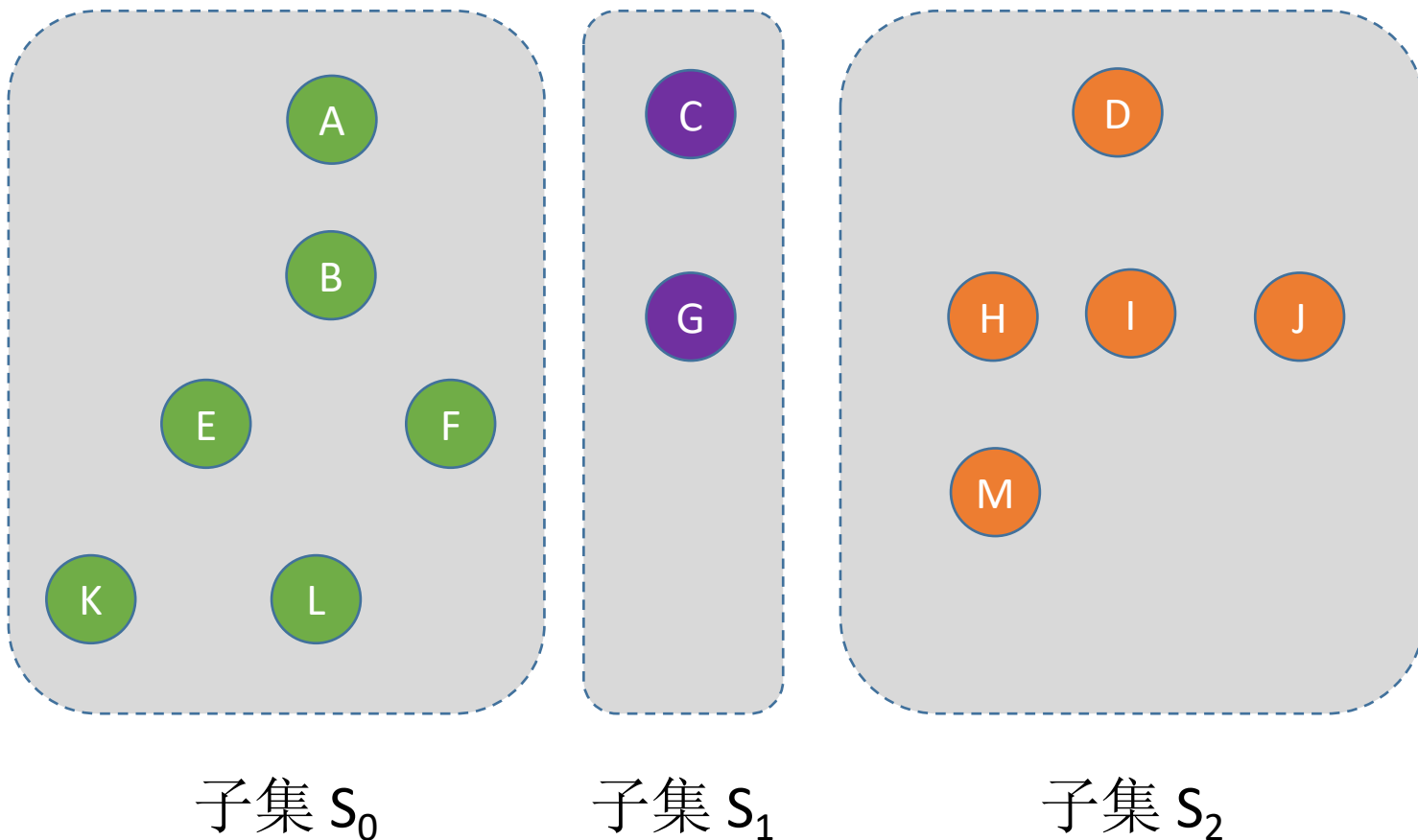
逻辑结构——“集合”



所有元素的全集 S

逻辑结构——“集合”

将各个元素划分为若干个互不相交的子集

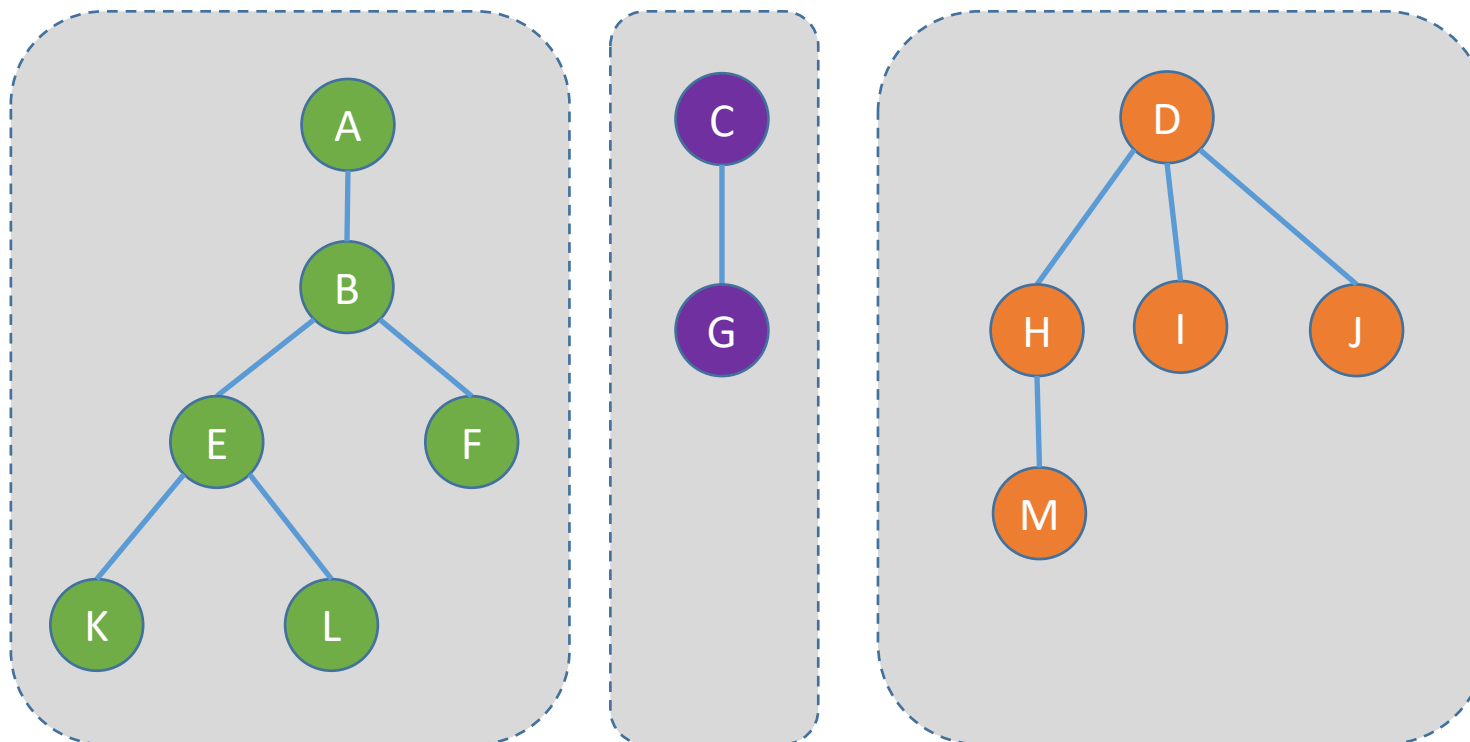


怎么用代码表示这种逻辑关系???



回顾：森林

森林。森林是 m ($m \geq 0$) 棵互不相交的树的集合



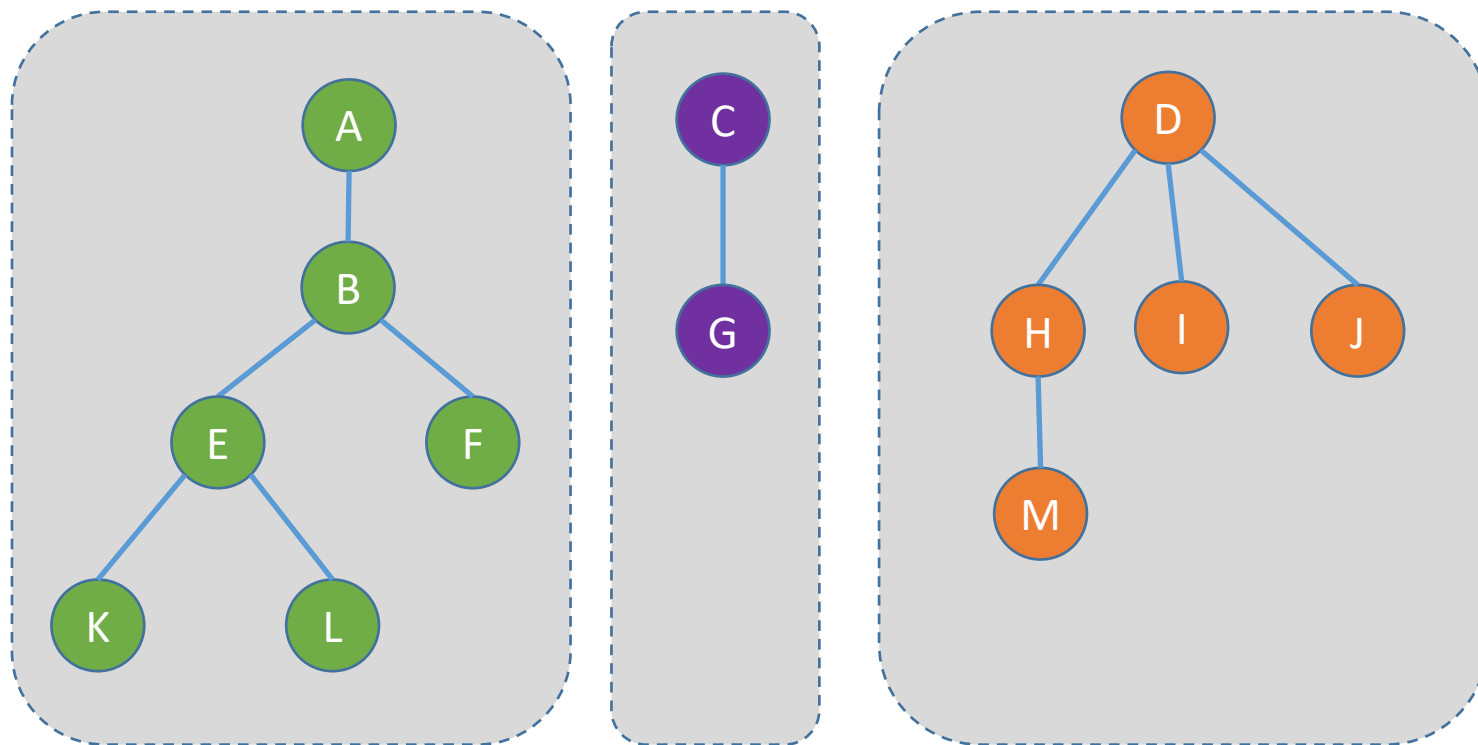
同一子集中的
各个元素，组
织成一棵树



灵稽一动

三棵树组成的森林

用互不相交的树，表示多个“集合”



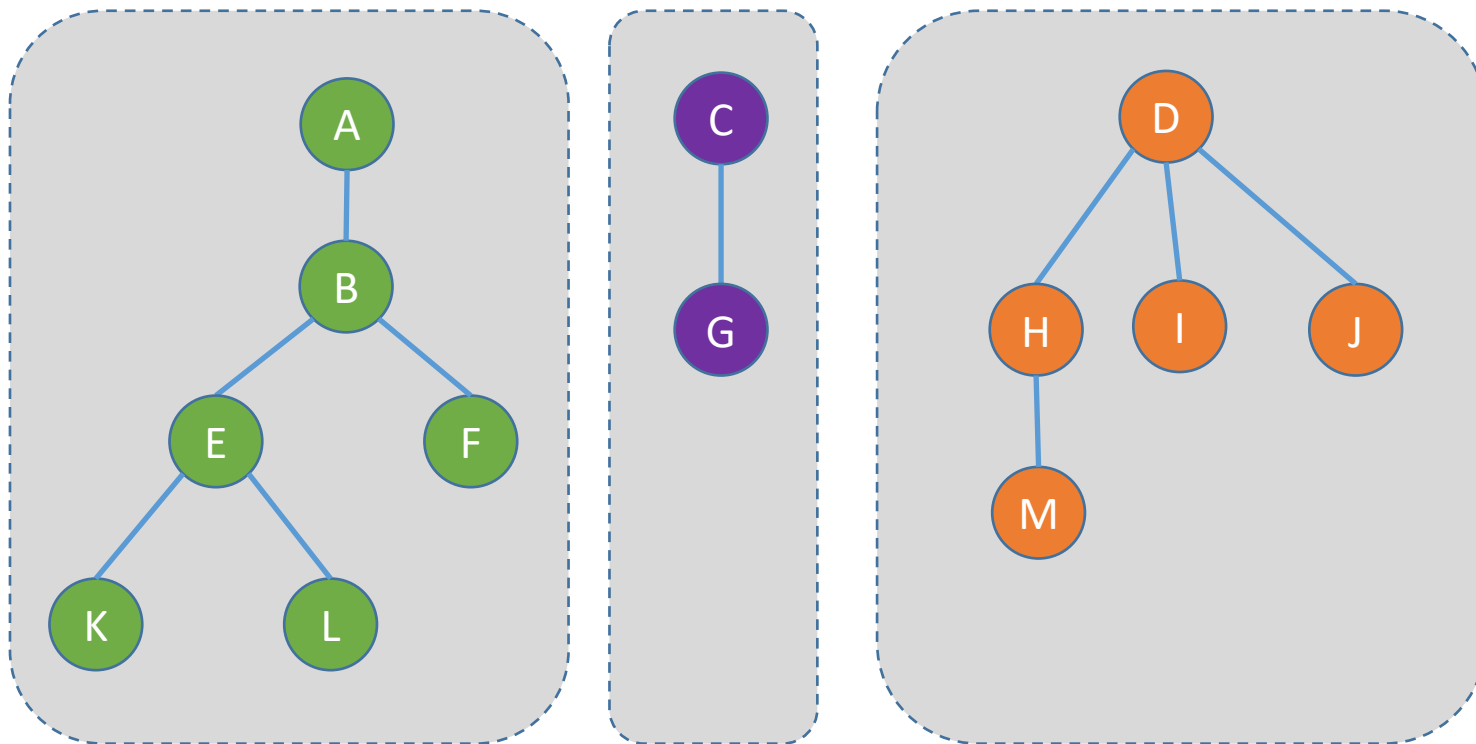
如何“查”到一个元素到底属于哪一个集合？
—— 从指定元素出发，一路向北，找到根节点

如何判断两个元素是否属于同一个集合？
—— 分别查到两个元素的根，判断根节点是否相同即可



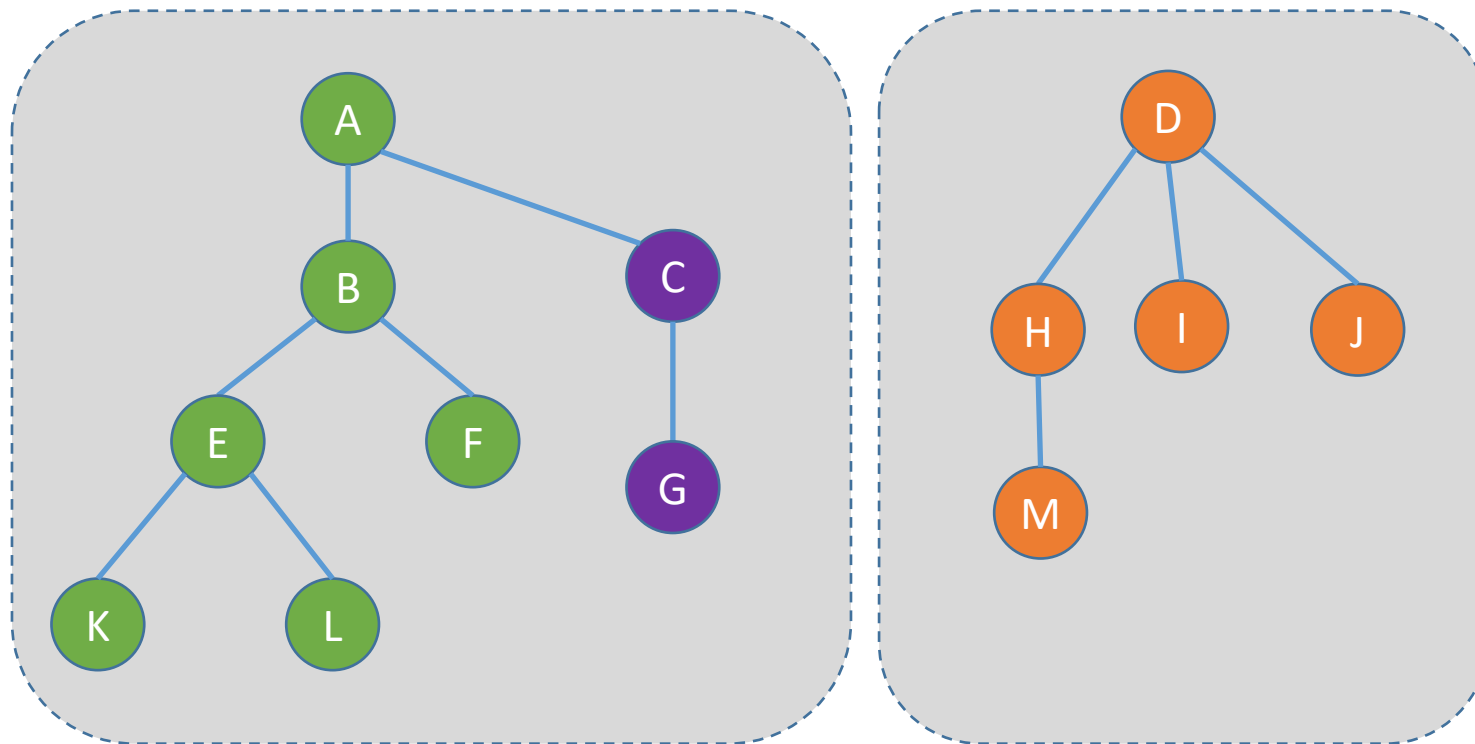
我一路向北
离开有你的季节

用互不相交的树，表示多个“集合”

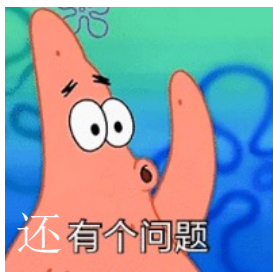


如何把两个集合“并”为一个集合？

用互不相交的树，表示多个“集合”



应采用什么样的存储结构？



如何把两个集合“并”为一个集合？

—— 让一棵树成为另一棵树的子树即可

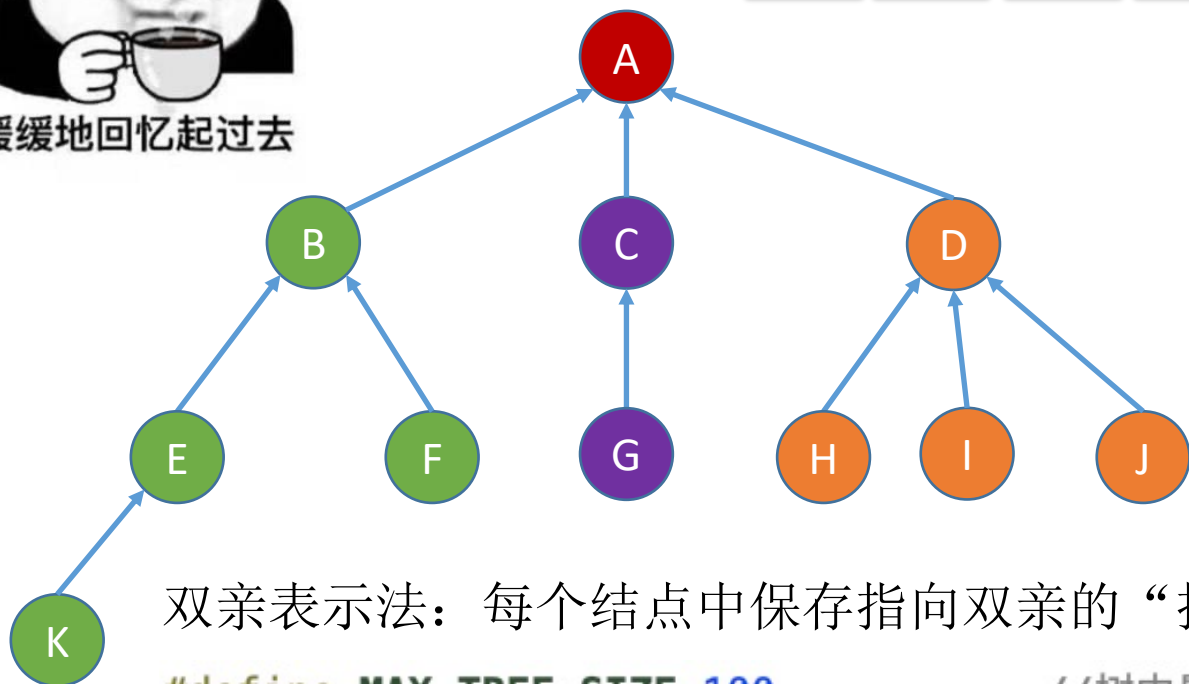


欲言又止 稍加思考



缓缓地回忆起过去

回忆：树的存储——双亲表示法



双亲表示法：每个结点中保存指向双亲的“指针”

```
#define MAX_TREE_SIZE 100
```

```
typedef struct{
```

```
    ElemType data;
```

```
    int parent;
```

```
}PTNode;
```

```
typedef struct{
```

```
    PTNode nodes[MAX_TREE_SIZE];
```

```
    int n;
```

```
}PTree;
```

//树中最多结点数

//树的结点定义

//数据元素

//双亲位置域

//树的类型定义

//双亲表示

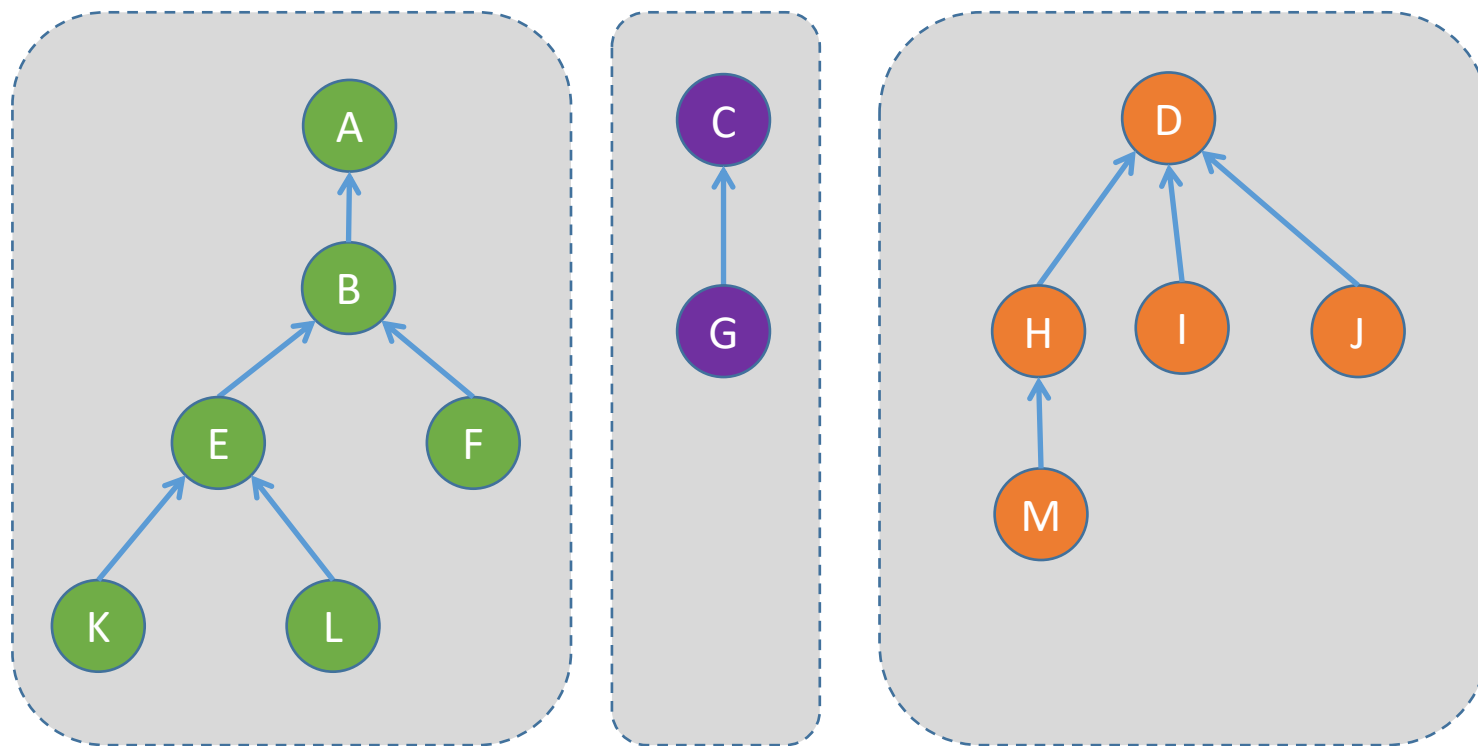
//结点数

	data	parent
0	A	-1
1	B	0
2	C	0
3	D	0
4	E	1
5	F	1
6	G	2
7	H	3
8	I	3
9	J	3
10	K	4
11		
12		
13		

根节点
parent=-1

parent=4 表
示父节点的
数组下标为4

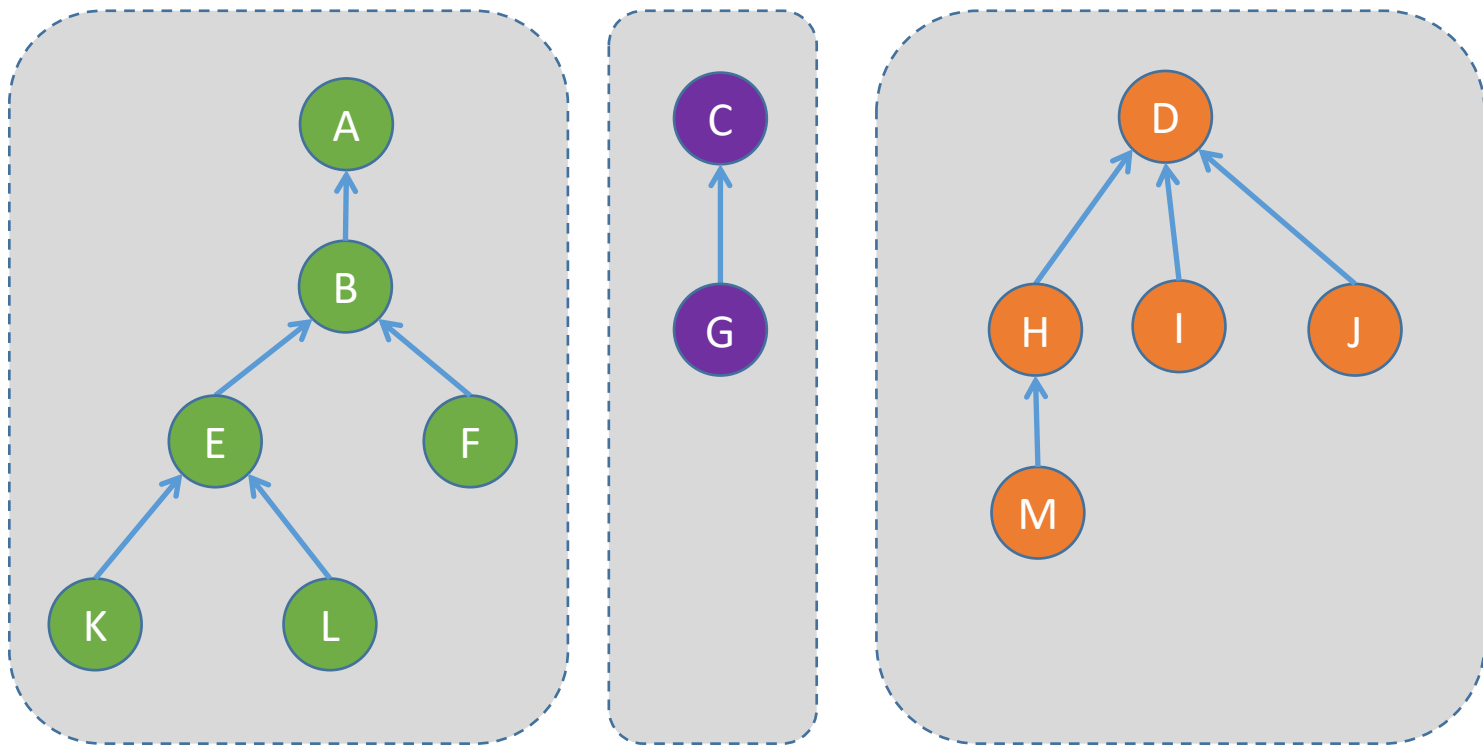
“并查集”的存储结构



数据元素	A	B	C	D	E	F	G	H	I	J	K	L	M
数组下标	0	1	2	3	4	5	6	7	8	9	10	11	12
S[]	-1	0	-1	-1	1	1	2	3	3	3	4	4	7

用一个数组 S[] 即可表示“集合”关系

“并查集”的基本操作



集合的两个基本操作——“并”和“查”

Find —— “查”操作：确定一个指定元素所属集合

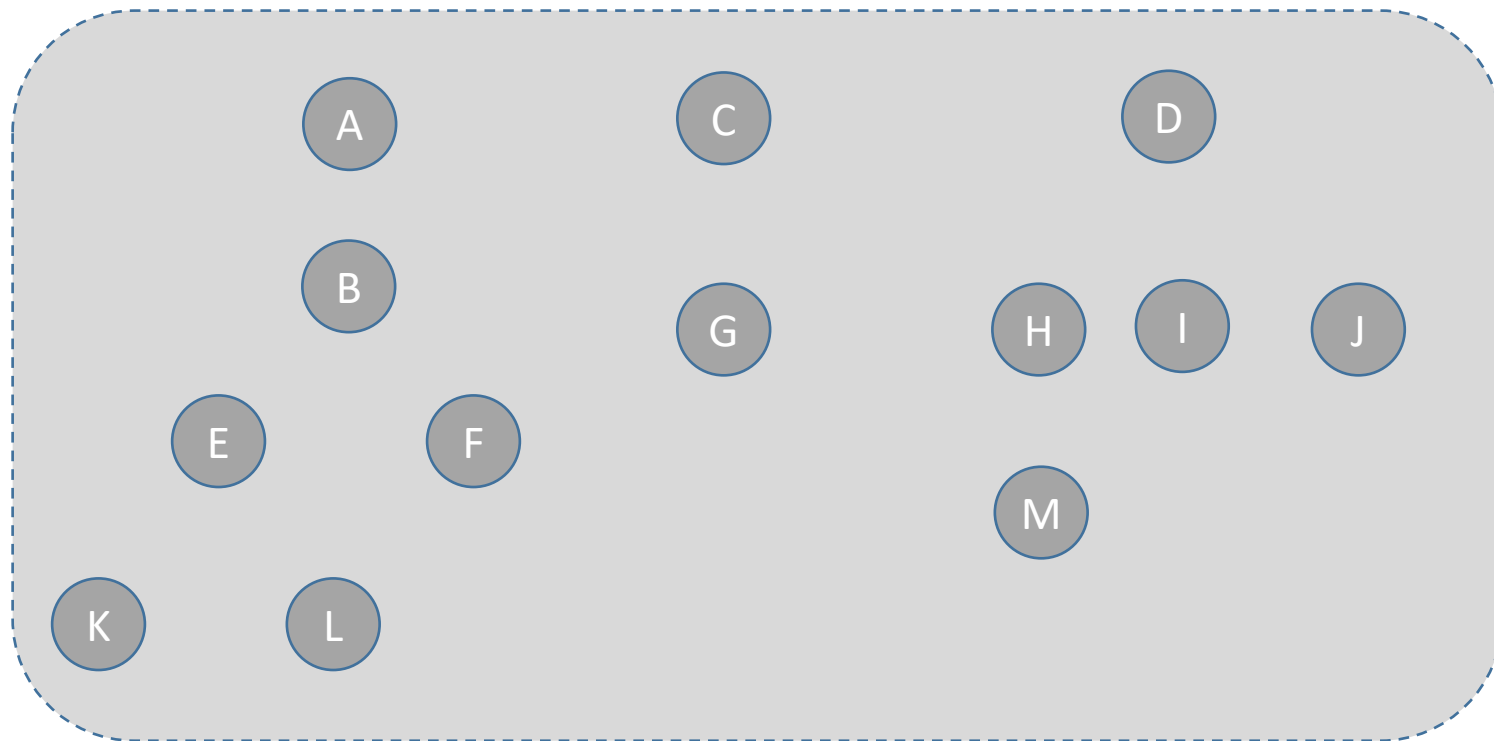
Union —— “并”操作：将两个不想交的集合合并为一个

注：并查集（Disjoint Set）是逻辑结构——集合的一种具体实现，只进行“并”和“查”两种基本操作

数据元素	A	B	C	D	E	F	G	H	I	J	K	L	M
数组下标	0	1	2	3	4	5	6	7	8	9	10	11	12
S[]	-1	0	-1	-1	1	1	2	3	3	3	4	4	7

用一个数组 S[] 即可表示“集合”关系

“并查集”的代码实现——初始化



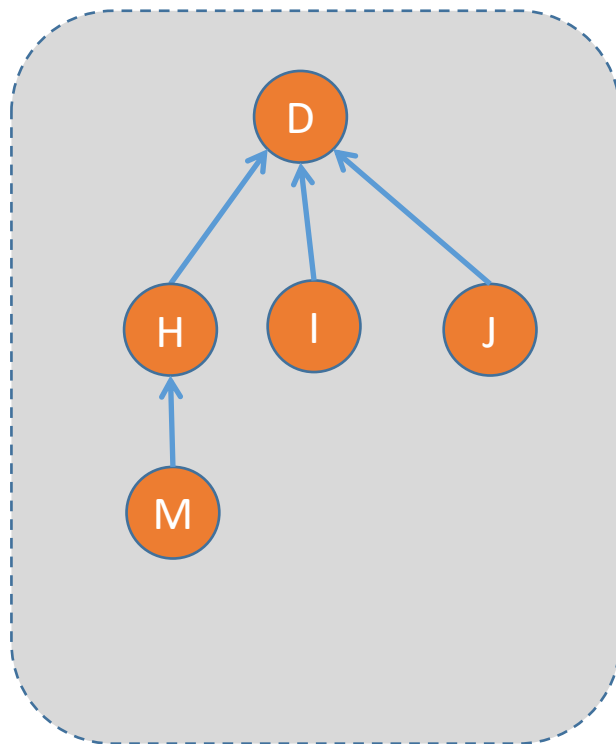
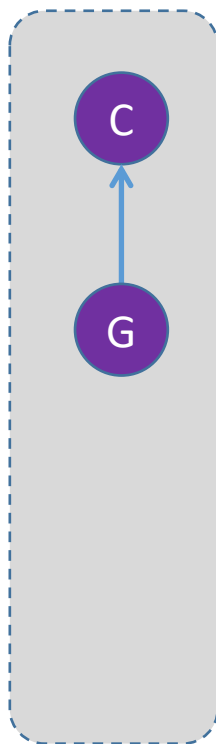
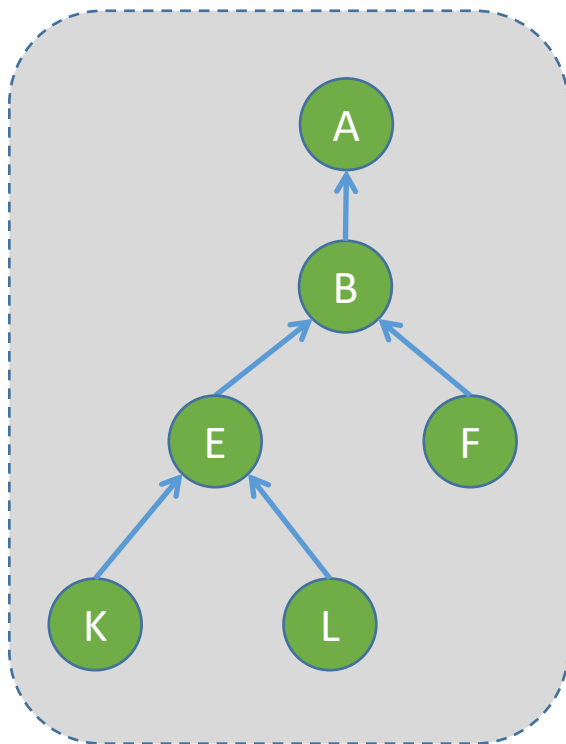
```
#define SIZE 13
int UFSets[SIZE];    //集合元素数组

//初始化并查集
void Initial(int S[]){
    for(int i=0;i<SIZE;i++)
        S[i]=-1;
}
```

数据元素	A	B	C	D	E	F	G	H	I	J	K	L	M
数组下标	0	1	2	3	4	5	6	7	8	9	10	11	12
S[]	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

用一个数组 S[] 即可表示“集合”关系

“并查集”的代码实现——并、查



```
//Find “查”操作，找x所属集合（返回x所属根结点）
int Find(int S[],int x){
    while(S[x]>=0)           //循环寻找x的根
        x=S[x];
    return x;                //根的s[]小于0
}

//Union “并”操作，将两个集合合并为一个
void Union(int S[],int Root1,int Root2){
    //要求Root1与Root2是不同的集合
    if(Root1==Root2) return;
    //将根Root2连接到另一根Root1下面
    S[Root2]=Root1;
}
```

数据元素	A	B	C	D	E	F	G	H	I	J	K	L	M
数组下标	0	1	2	3	4	5	6	7	8	9	10	11	12
S[]	-1	0	-1	-1	1	1	2	3	3	3	4	4	7

用一个数组 S[] 即可表示“集合”关系

时间复杂度分析

```
#define SIZE 13
int UFsets[SIZE];    //集合元素数组

//初始化并查集
void Initial(int S[]){
    for(int i=0;i<SIZE;i++)
        S[i]=-1;
}

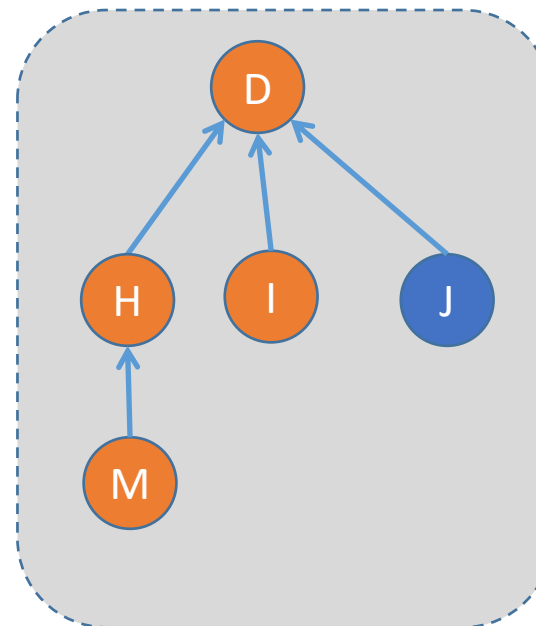
//Find “查”操作，找x所属集合（返回x所属根结点）
int Find(int S[],int x){
    while(S[x]>=0)    //循环寻找x的根
        x=S[x];
    return x;        //根的s[]小于0
}

//Union “并”操作，将两个集合合并为一个
void Union(int S[],int Root1,int Root2){
    //要求Root1与Root2是不同的集合
    if(Root1==Root2) return;
    //将根Root2连接到另一根Root1下面
    S[Root2]=Root1;
}
```

最坏时间复杂度：
 $O(n)$

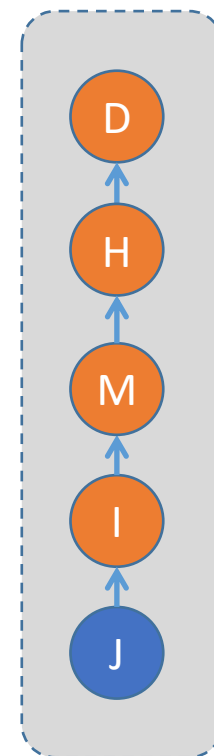
时间复杂度：
 $O(1)$

找到J所属的集合



较好的情况

高度
 $h=n$



最坏的情况

若结点数为 n ，Find 最坏时间复杂度为 $O(n)$

Union 操作的优化

```
#define SIZE 13
int UFSets[SIZE];    //集合元素数组

//初始化并查集
void Initial(int S[]){
    for(int i=0;i<SIZE;i++)
        S[i]=-1;
}

//Find “查”操作，找x所属集合（返回x所属根结点）
int Find(int S[],int x){
    while(S[x]>=0)    //循环寻找x的根
        x=S[x];
    return x;        //根的s[]小于0
}

//Union “并”操作，将两个集合合并为一个
void Union(int S[],int Root1,int Root2){
    //要求Root1与Root2是不同的集合
    if(Root1==Root2) return;
    //将根Root2连接到另一根Root1下面
    S[Root2]=Root1;
}
```

最坏时间复杂度：
 $O(n)$

时间复杂度：
 $O(1)$

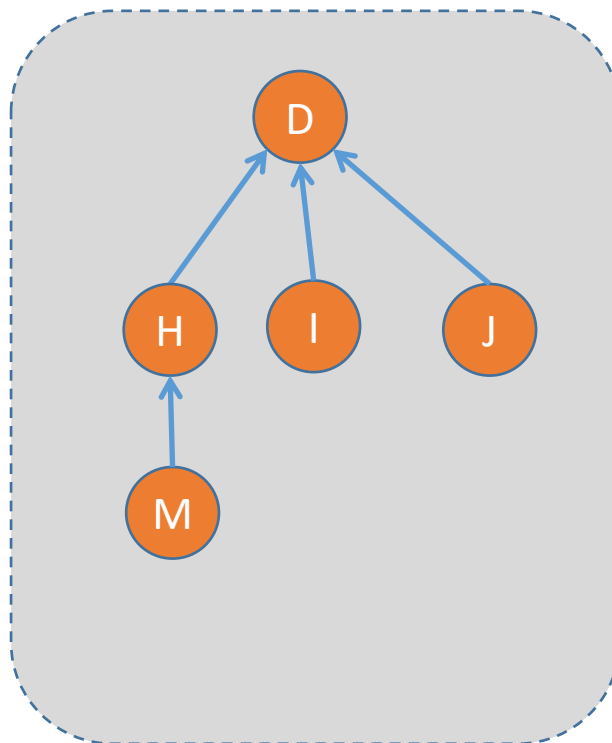
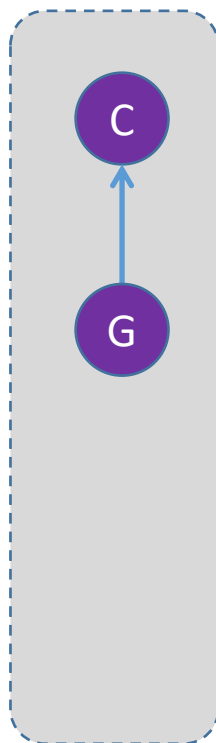
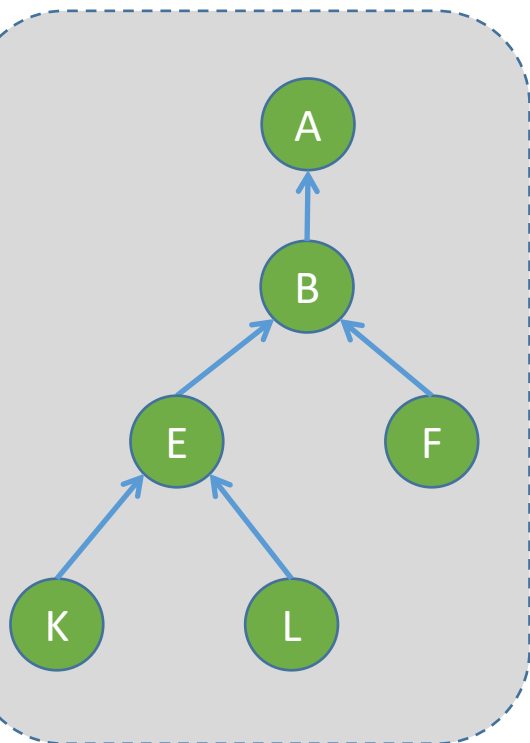
好主意



优化思路：在每次Union操作构建树的时候，尽可能让树不长高高

- ①用根节点的绝对值表示树的结点总数
- ②Union操作，让小树合并到大树

Union 操作的优化



//Union “并”操作，小树合并到大树

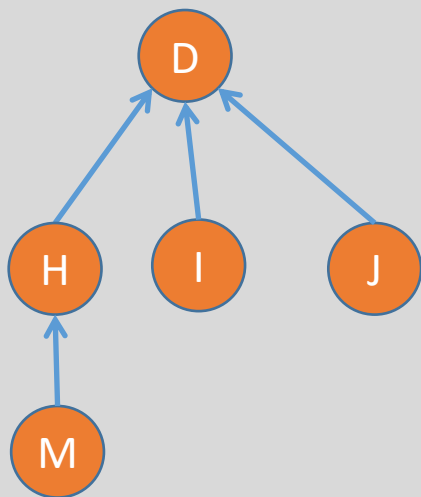
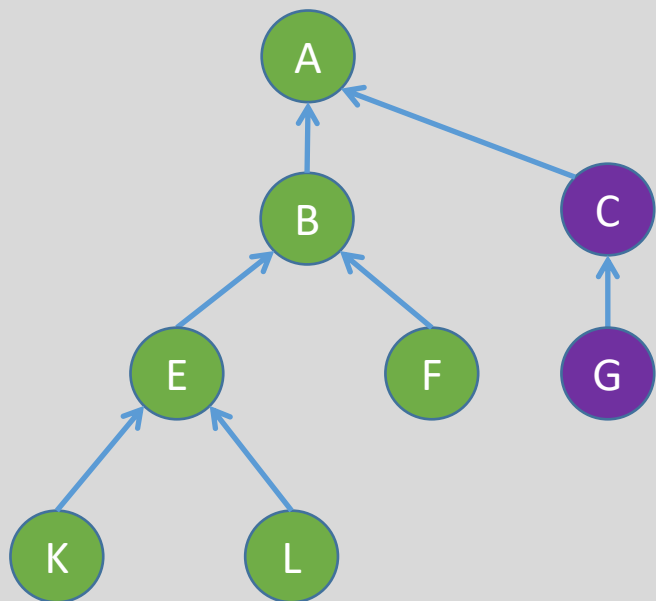
```
void Union(int S[],int Root1,int Root2){  
    if(Root1==Root2) return;  
    if(S[Root2]>S[Root1]) { //Root2结点数更少  
        S[Root1] += S[Root2]; //累加结点总数  
        S[Root2]=Root1; //小树合并到大树  
    } else {  
        S[Root2] += S[Root1]; //累加结点总数  
        S[Root1]=Root2; //小树合并到大树  
    }  
}
```

数据元素
数组下标

	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12
S[]	-6	0	-2	-5	1	1	2	3	3	3	4	4	7

- ①用根节点的绝对值表示树的结点总数
- ②Union操作，让小树合并到大树

Union 操作的优化



//Union “并”操作，小树合并到大树

```
void Union(int S[],int Root1,int Root2){  
    if(Root1==Root2) return;  
    if(S[Root2]>S[Root1]) { //Root2结点数更少  
        S[Root1] += S[Root2]; //累加结点总数  
        S[Root2]=Root1; //小树合并到大树  
    } else {  
        S[Root2] += S[Root1]; //累加结点总数  
        S[Root1]=Root2; //小树合并到大树  
    }  
}
```

数据元素

数组下标

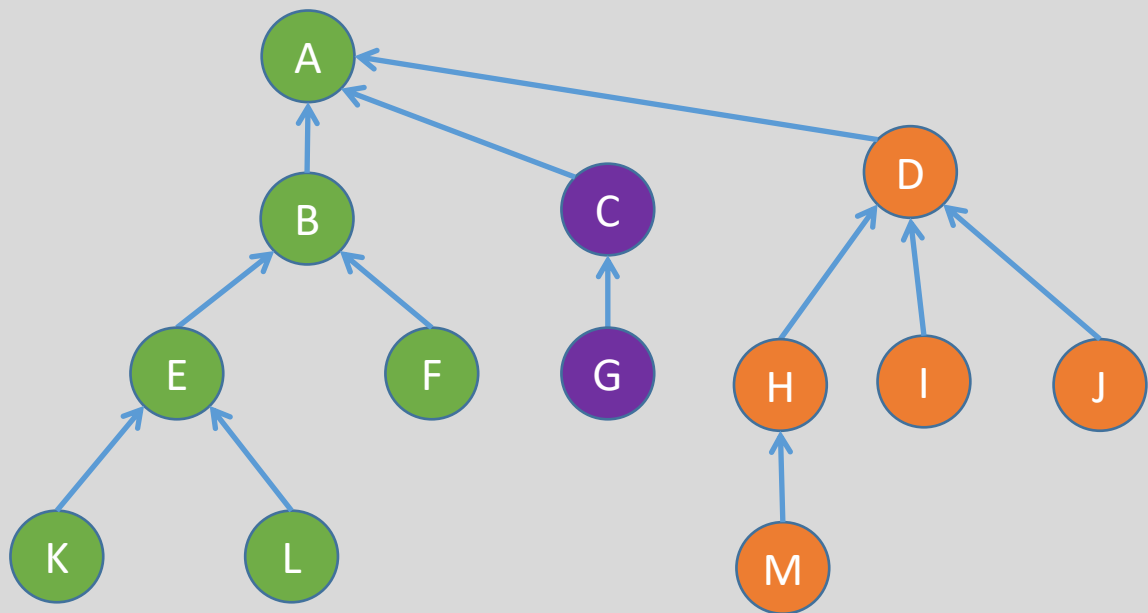
S[]

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
-8	0	0	-5	1	1	2	3	3	3	4	4	7

①用根节点的绝对值表示树的结点总数

②Union操作，让小树合并到大树

Union 操作的优化



//Union “并”操作，小树合并到大树

```
void Union(int S[],int Root1,int Root2){  
    if(Root1==Root2) return;  
    if(S[Root2]>S[Root1]) { //Root2结点数更少  
        S[Root1] += S[Root2]; //累加结点总数  
        S[Root2]=Root1; //小树合并到大树  
    } else {  
        S[Root2] += S[Root1]; //累加结点总数  
        S[Root1]=Root2; //小树合并到大树  
    }  
}
```

数据元素

数组下标

	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	1	2	3	4	5	6	7	8	9	10	11	12
S[]	-13	0	0	0	1	1	2	3	3	3	4	4	7

①用根节点的绝对值表示树的结点总数

②Union操作，让小树合并到大树

Union 操作的优化

```
#define SIZE 13
int UFSets[SIZE];    //集合元素数组

//初始化并查集
void Initial(int S[]){
    for(int i=0;i<SIZE;i++){
        S[i]=-1;
    }

    //Find “查”操作，找x所属集合（返回x所属根结点）
    int Find(int S[],int x){
        while(S[x]>=0)    //循环寻找x的根
            x=S[x];
        return x;        //根的s[]小于0
    }
}
```

```
//Union “并”操作，将两个集合合并为一个
void Union(int S[],int Root1,int Root2){
    //要求Root1与Root2是不同的集合
    if(Root1==Root2) return;
    //将根Root2连接到另一根Root1下面
    S[Root2]=Root1;
}
```

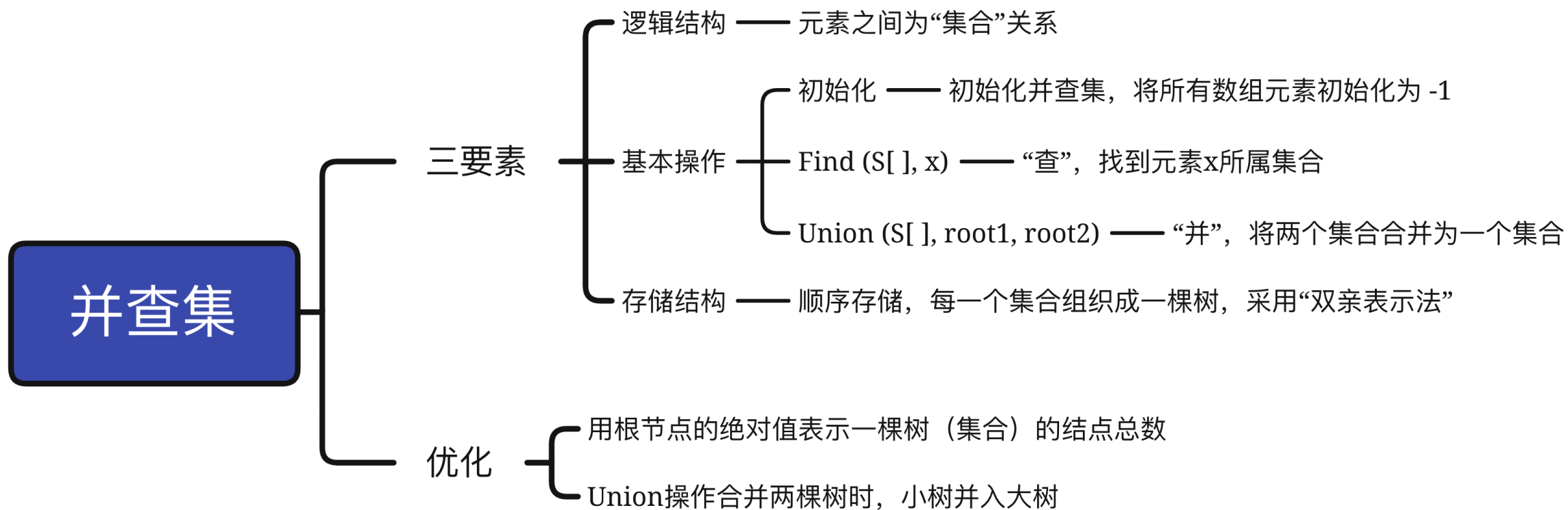
Union操作优化后，
Find 操作最坏时间
复杂度： $O(\log_2 n)$

该方法构造的树高不超过 $\lfloor \log_2 n \rfloor + 1$

优化

```
//Union “并”操作，小树合并到大树
void Union(int S[],int Root1,int Root2){
    if(Root1==Root2) return;
    if(S[Root2]>S[Root1]) { //Root2结点数更少
        S[Root1] += S[Root2]; //累加结点总数
        S[Root2]=Root1; //小树合并到大树
    } else {
        S[Root2] += S[Root1]; //累加结点总数
        S[Root1]=Root2; //小树合并到大树
    }
}
```

知识回顾与重要考点



$$\text{树高} \leq \lfloor \log_2 n \rfloor + 1$$

$$\text{Find} \rightarrow O(\log_2 n)$$

欢迎大家对本节视频进行评价~



学员评分：5.5.2_1 并查集

扫一扫二维码打开或分享给好友



— 腾讯文档 —

可多人实时在线编辑，权限安全可控



公众号：王道在线



b站：王道计算机教育



抖音：王道计算机考研