



数据结构与算法（六）

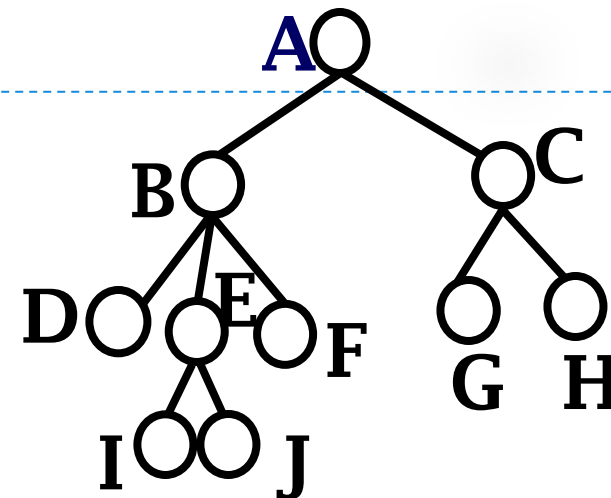
张铭 主讲

采用教材：张铭，王腾蛟，赵海燕 编写
高等教育出版社，2008.6（“十一五”国家级规划教材）

<http://www.jpku.pku.edu.cn/pkujpku/course/sjjg>

第6章 树

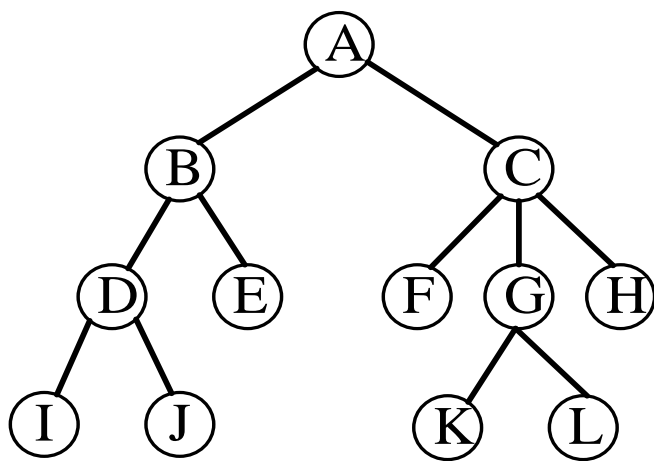
- 树的定义和基本术语
- 树的链式存储结构
 - “子结点表”表示方法
 - 静态“左孩子/右兄弟”表示法
 - 动态表示法
 - 动态“左孩子/右兄弟”表示法
 - 父指针表示法及其在并查集中的应用
- 树的顺序存储结构
- K叉树



6.2 树的链式存储结构

父指针表示法

- 只需要知道父结点的应用
- 只需要保存一个指向其父结点的指针域，称为 **父指针** (parent pointer)表示法
- 用数组存储树结点，同时并在每个结点中附设一个指针指示其父结点的位置



结点索引 值	0	1	2	3	4	5	6	7	8	9	10	11
父结点索引	A	B	C	D	E	F	G	H	I	J	K	L
		0	0	1	1	2	2	2	3	3	6	6

6.2 树的链式存储结构

父指针表示法：算法

- 查询结点的根
 - 从一个结点出发找出一条向上延伸到达根的祖先路径
 - $O(k)$, k 为树高
- 判断两个结点是否在同一棵树
 - 两个结点根结点相同，它们一定在同一棵树中
 - 如果其根结点不同，那么两个结点就不在同一棵树中

6.2 树的链式存储结构

并查集

并查集 是一种特殊的集合，由一些不相交子集构成，合并查集的基本操作是：

- Find : 查询结点所在集合
 - Union : 归并两个集合
-
- 并查集是重要的抽象数据类型
 - 应用于求解等价类等等问题

6.2 树的链式存储结构

等价关系

- 一个具有 n 个元素的集合 S ，另有一个定义在集合 S 上的 r 个关系的关系集合 R 。 x, y, z 表示集合中的元素
- 若关系 R 是一个 **等价关系**，当且仅当如下条件为真时成立：
 - (a) 对于所有的 x ，有 $(x, x) \in R$ （即关系是**自反**的）
 - (b) 当且仅当 $(x, y) \in R$ 时 $(y, x) \in R$ （即关系是**对称**的）
 - (c) 若 $(x, y) \in R$ 且 $(y, z) \in R$ ，则有 $(x, z) \in R$ （即关系是**传递**的）
- 如果 $(x, y) \in R$ ，则元素 x 和 y 是等价的



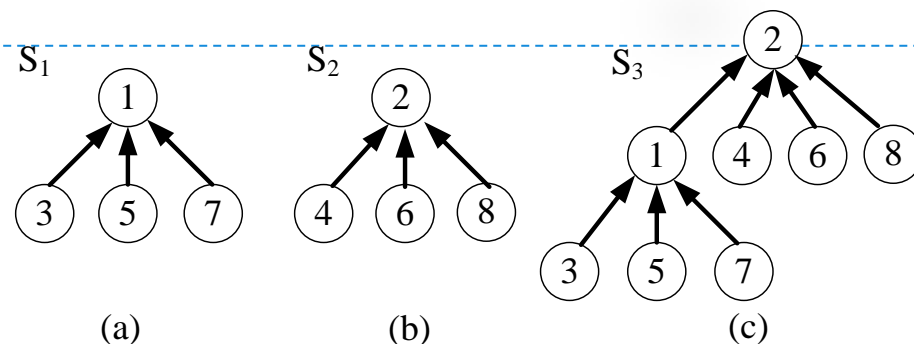
等价类(equivalence classes)

- 等价类是指相互等价的元素所组成的最大集合。所谓最大，就是指不存在类以外的元素，与类内部的元素等价
- 由 $x \in S$ 生成的一个R等价类
 - $[x]_R = \{y \mid y \in S \wedge xRy\}$
 - R将S划分成为r个不相交的划分 S_1, S_2, \dots, S_r ，这些集合的并为S



用树来表示等价类的并查

- 用一棵树代表一个集合
 - 集合用父结点代替
 - 若两个结点在同一棵树中，则它们处于同一个集合
- 树的实现
 - 存储在静态指针数组中
 - 结点中仅需保存父指针信息



6.2 树的链式存储结构

UNION/FIND算法示例(1)

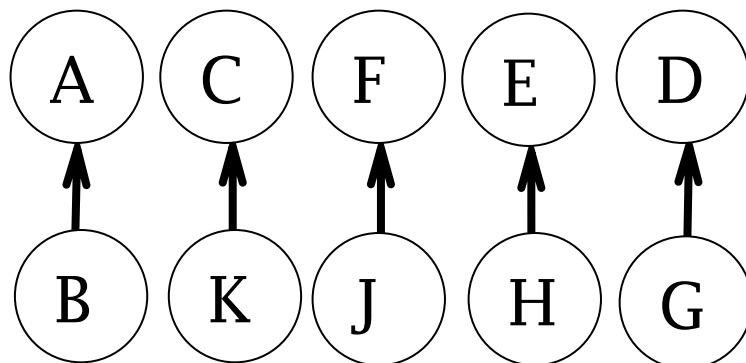
对这5个等价对进行处理 (A,B)、

(C,K)、(J,F)、(H,E)、(D,G)

	0		2				4	5	6
A	B	C	K	D	E	F	G	H	J

0 1 2 3 4 5 6 7 8 9

(A,B)(C,K)(J,F)(E,H)(D,G)



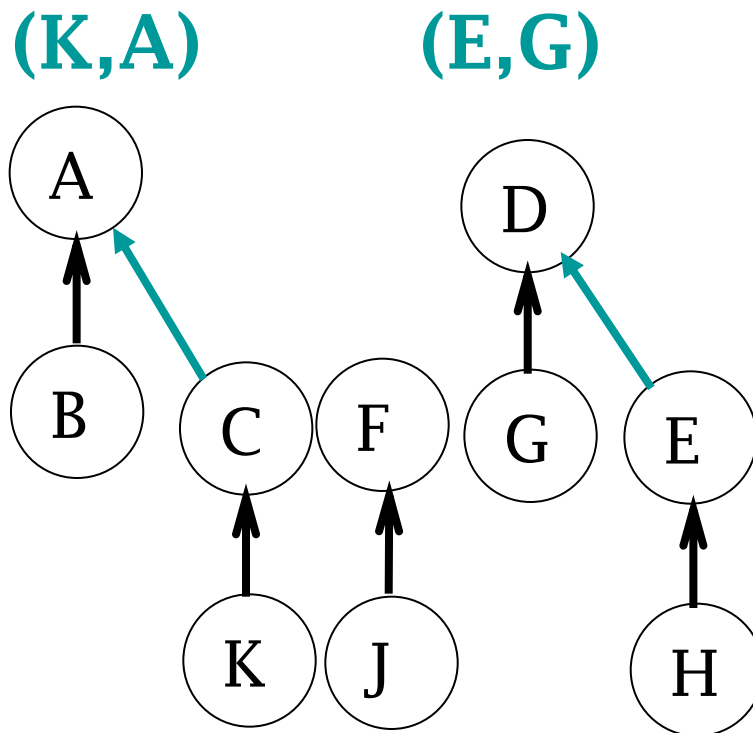
6.2 树的链式存储结构

UNION/FIND算法示例(1)

然后对两个等价对 (K, A) 和 (E, G) 进行处理

K所在树的根为C, A自己是根结点, $A \neq C$, 所以两棵树要合并

	0	0	2		4		4	5	6
A	B	C	K	D	E	F	G	H	J
0	1	2	3	4	5	6	7	8	9





树的父指针表示与Union/Find算法实现

```
template<class T>
class ParTreeNode {                                //树结点定义
private:
    Tvalue;                                        //结点的值
    ParTreeNode<T>* parent;                       //父结点指针
    int nCount;                                    //集合中总结点个数
public:
    ParTreeNode();                                //构造函数
    virtual ~ParTreeNode(){};                     //析构函数
    TgetValue();                                  //返回结点的值
    void setValue(const T& val);                   //设置结点的值
    ParTreeNode<T>* getParent();                   //返回父结点指针
    void setParent(ParTreeNode<T>* par);           //设置父指针
    int getCount();                                //返回结点数目
    void setCount(const int count);                 //设置结点数目
};
```



树的父指针表示与Union/Find算法实现

```
template<class T>
class ParTree {                                // 树定义
public:
    ParTreeNode<T>* array;                     // 存储树结点的数组
    int Size;                                  // 数组大小
    ParTreeNode<T>*
    Find(ParTreeNode<T>* node) const;         // 查找node结点的根结点
    ParTree(const int size);                  // 构造函数
    virtual ~ParTree();                      // 析构函数
    void Union(int i,int j);                  // 把下标为i , j的结点合并成一棵子树
    bool Different(int i,int j);              // 判定下标为i , j的结点是否在一棵树中
};
```



树的父指针表示与Union/Find算法实现

```
template <class T>
ParTreeNode<T>*
ParTree<T>::Find(ParTreeNode<T>* node) const
{
    ParTreeNode<T>* pointer=node;
    while ( pointer->getParent() != NULL )
        pointer=pointer->getParent();
    return pointer;
}
```



树的父指针表示与Union/Find算法实现

```
template<class T>
void ParTree<T>::Union(int i,int j) {
    ParTreeNode<T>* pointeri = Find(&array[i]);    //找到结点i的根
    ParTreeNode<T>* pointerj = Find(&array[j]);    //找到结点j的根
    if (pointeri != pointerj) {
        if(pointeri->getCount() >= pointerj->getCount()) {
            pointerj->setParent(pointeri);
            pointeri->setCount(pointeri->getCount() +
                               pointerj->getCount());
        }
        else {
            pointeri->setParent(pointerj);
            pointerj->setCount(pointeri->getCount() +
                               pointerj->getCount());
        }
    }
}
```

6.2 树的链式存储结构

UNION/FIND算法示例(2)

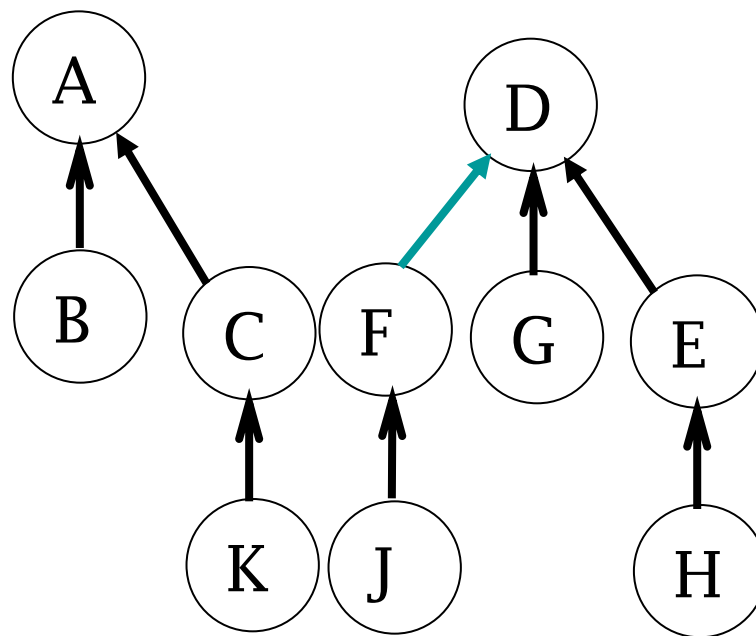
最后使用加权合并规则处理等价对 (H,J)

H所在树的根为E，而J的根结点为F， $E \neq F$ ，所以两颗树要合并

依据加权合并规则，以F为根的树结点数少，故将F指向D

(H,J)

	0	0	2		4	4	4	5	6
A	B	C	K	D	E	F	G	H	J
0	1	2	3	4	5	6	7	8	9

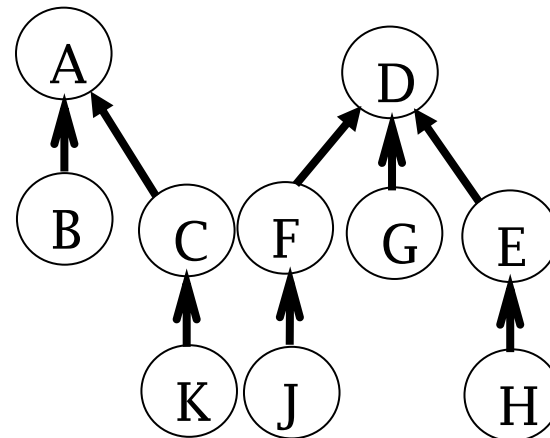


6.2 树的链式存储结构

路径压缩

• 查找X

- 设X最终到达根R
- 顺着由X到R的路径把每个结点的父指针域均设置为直接指向R

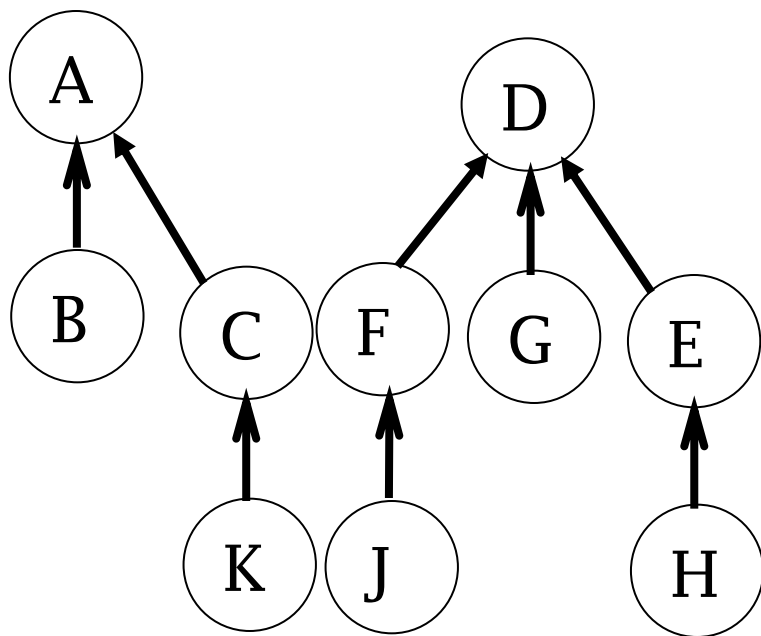


• 产生极浅树

6.2 树的链式存储结构

UNION/FIND算法示例(3)

使用路径压缩规则处理Find(H)

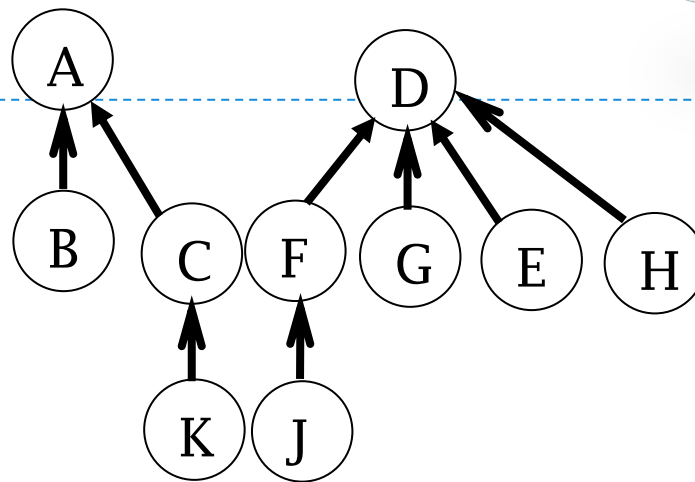


	0	0	2		4	4	4	4	6
A	B	C	K	D	E	F	G	H	J
0	1	2	3	4	5	6	7	8	9

6.2 树的链式存储结构

路径压缩

```
template <class T>
ParTreeNode<T>*
ParTree<T>::FindPC(ParTreeNode<T>* node) const
{
    if (node->getParent() == NULL)
        return node;
    node->setParent(FindPC(node->getParent()));
    return node->getParent();
}
```





路径压缩使Find开销接近于常数

- 权重 + 路径压缩
- 对 n 个结点进行 n 次Find操作的开销为 $O(n\alpha(n))$ ，约为 $\Theta(n\log^*n)$
 - $\alpha(n)$ 是单变量Ackermann函数的逆，它是一个增长速度比 $\log n$ 慢得多但又不是常数的函数
 - \log^*n 是在 $n = \log n \leq 1$ 之前要进行的对 n 取对数操作的次数
 - $\log^*65536 = 4$ (4次log操作)
- Find至多需要一系列 n 个Find操作的开销非常接近于 $\Theta(n)$
 - 在实际应用中， $\alpha(n)$ 往往小于4

6.2 树的链式存储结构

思考

- 可否使用动态指针方式实现父指针表示法？
- 查阅各种并查集权重和路径压缩优化方法，并讨论各种方法的异同和优劣



数据结构与算法

谢谢聆听

国家精品课“数据结构与算法”

<http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/>

张铭，王腾蛟，赵海燕

高等教育出版社，2008.6。“十一五”国家级规划教材