数据结构:树

Data Structure

主讲教师：杨晓波

E-mail:　248133074@qq.com

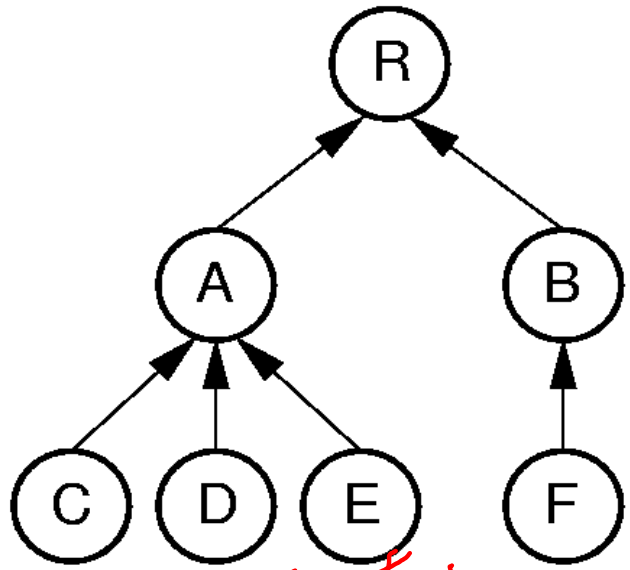
树

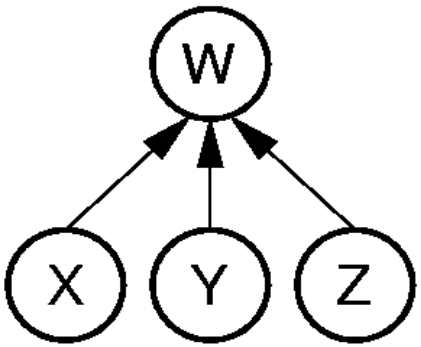
- 父指针表示法（并查集树）
- 树的实现
- 树的顺序表示法

树的实现问题

- 二叉树中每个结点最多有2个孩子结点，因此可以只用两个指针域分别表示左右孩子；
- 而树的孩子有多个，应该如何设计树的结点结构才能既充分表示出信息又节省空间呢?
- **父指针表示法**就是树实现的一种简单方式，对每个结点只保存一个指针域，指向其父结点。

父指针表示法（并查集树）





Parent's Index	/	0	0	1	1	1	2	/	7	7	7
Label	R	A	B	C	D	E	F	W	X	Y	Z
Node Index	0	1	2	3	4	5	6	7	8	9	10

应用场景——Equivalence Class Problem (等价类问题)

The parent pointer representation is good for answering:

- Are two elements in the same tree?

```
// Return TRUE if nodes in different trees
bool Gentree::differ(int a, int b) {
  int root1 = FIND(a);   // Find root for a
  int root2 = FIND(b);   // Find root for b
  return root1 != root2; // Compare roots
}
```

父指针表示法的应用

- 父指针表示法常用于维护由一些不相交子集构成的集合

- 对于不相交集合，希望提供两种基本操作
  - 判断两个结点是否同一个集合；
  - 归并两个集合

因此命名为并查算法（并查集）

- 并查算法用一棵树代表一个集合。如果两个结点在同一棵树中，则认为它们在同一个集合中。

树的并查算法实现（uf.h)

```
// General tree representation for UNION/FIND
class ParPtrTree {
private:
  int* array;                    // Node array
  int size;                      // Size of node array
  int FIND(int) const;           // Find root
public:
  ParPtrTree(int);               // Constructor
  ~ParPtrTree() { delete [] array; } // Destructor
  void UNION(int, int);          // Merge equivalences
  bool differ(int, int);         // True if not in same tree
};
```

每个节点父结点的索引

树的并查算法实现

```
ParPtrTree::ParPtrTree(int sz) { // Constructor
  size = sz;
  array = new int[sz];           // Create node array
  for(int i=0; i<sz; i++) array[i] = ROOT;
}

// Return True if nodes are in different trees
bool ParPtrTree::differ(int a, int b) {
  int root1 = FIND(a);           // Find root of node a
  int root2 = FIND(b);           // Find root of node b
  return root1 != root2;         // Compare roots
}
```

空，不可能出现的值

树的并查算法实现

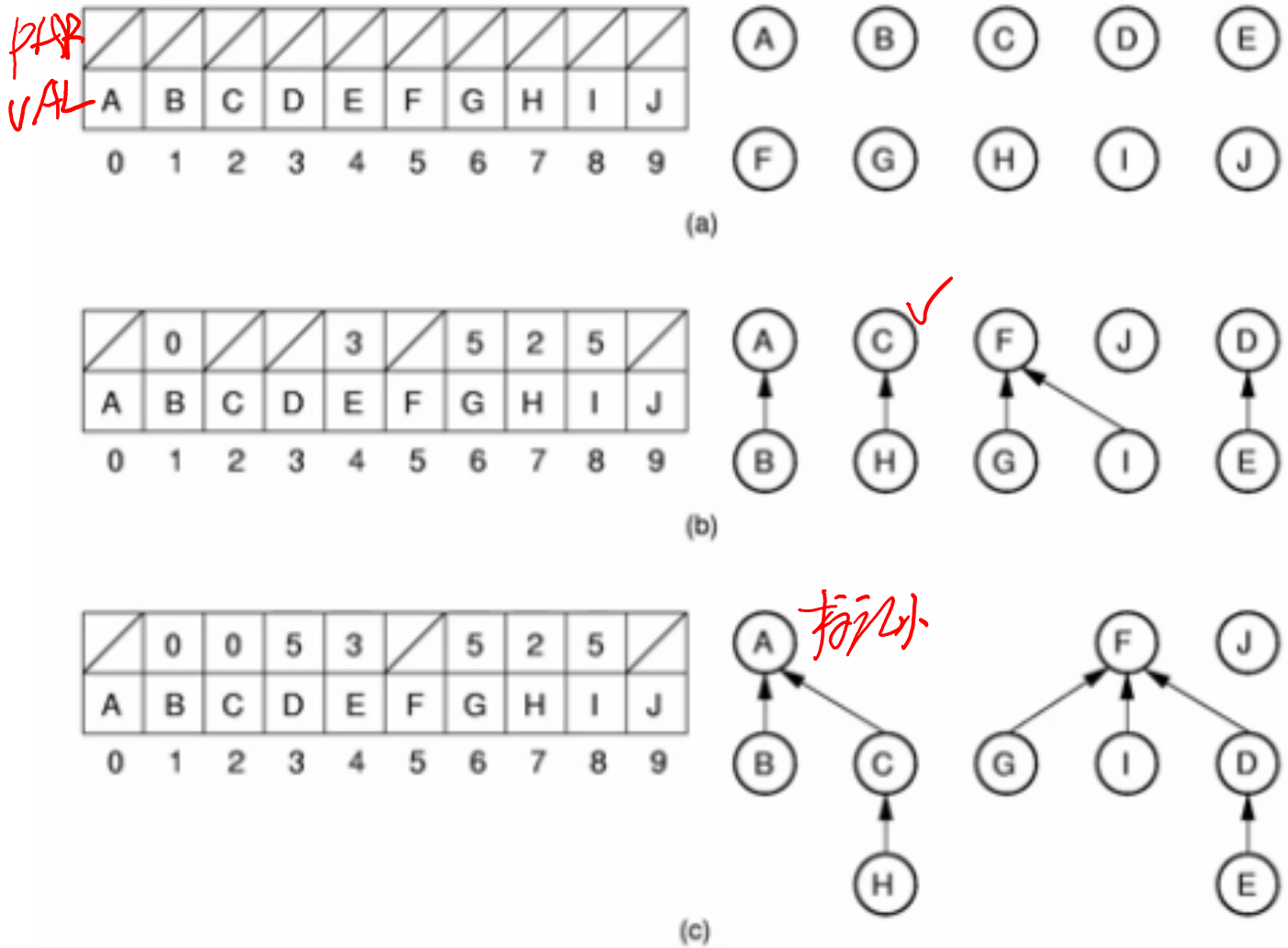
- void ParPtrTree::UNION(int a, int b) { // Merge subtrees
- int root1 = FIND(a); // Find root of node a
- int root2 = FIND(b); // Find root of node b
- if (root1 != root2) array[root2] = root1; // Merge
- }
- // FIND with path compression
- int ParPtrTree::FIND(int curr) const {
- if (array[curr] == ROOT) return curr; // At root
- **array[curr] = FIND(array[curr]);** find返回根节点
- **//通过递归调用实现该结点至根结点的沿路结点的压缩，使得沿路结点的父结点都变成了根结点，从而缩短了这些结点到根结点的路径长度。**
- return array[curr];
- }
- //把当前结点所有祖先结点的父指针都指向根结点

root2父结点为root1

设第二棵树的根结点为第1棵树的结点

及

等价类处理的例子 (1)



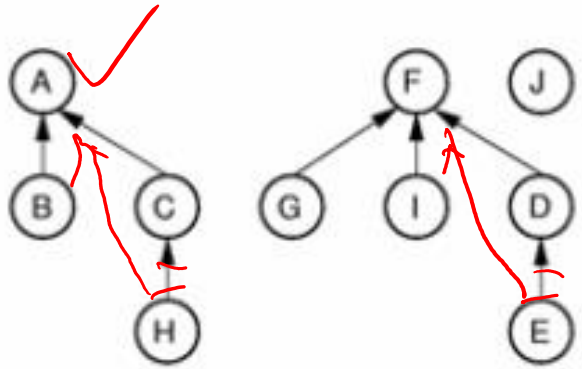
方法：判断2个等价类对应的结点现在在同一棵树中吗？如不在，则合并两棵树。

对（A,B）(C,H）,(G,F),(D,E)和(I,F)五个等价类的处理

对（H,A）和(E,G)等价类的处理

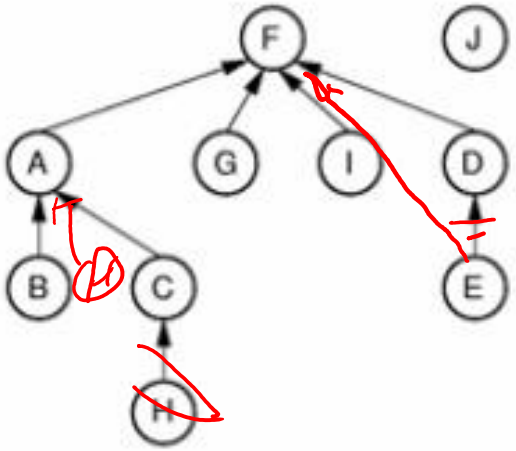
等价类处理的例子(2)

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



(c)

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

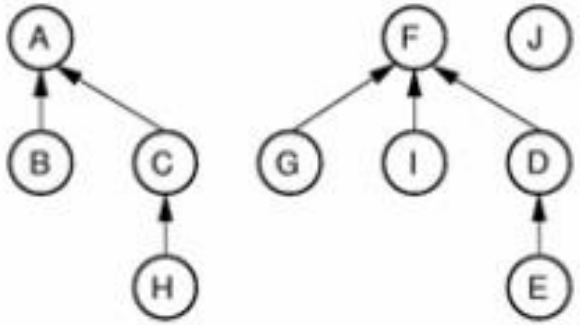


(d)

对（E,H）等价类的处理

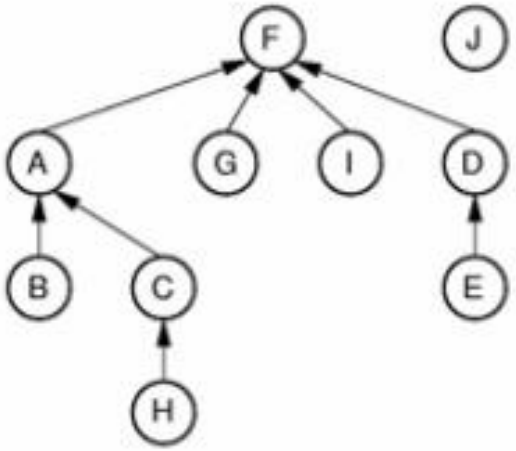
等价类处理的例子(2)

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



(c)

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



(d)

教材图6.7提供了不做路径压缩的
对（E,H）等价类的处理

不带路径压缩的find方法

```
// FIND root without path compression
int ParPtrTree::FIND(int curr) const {
  while（array[curr]!=ROOT） curr= array[curr];
  return curr;
}
```

合并规则

重量权衡合并规则

重量权衡合并规则（Weighted union rule）：两树归并时，将结点较少树的根结点指向结点较多树的根结点.

可以降低树的高度.

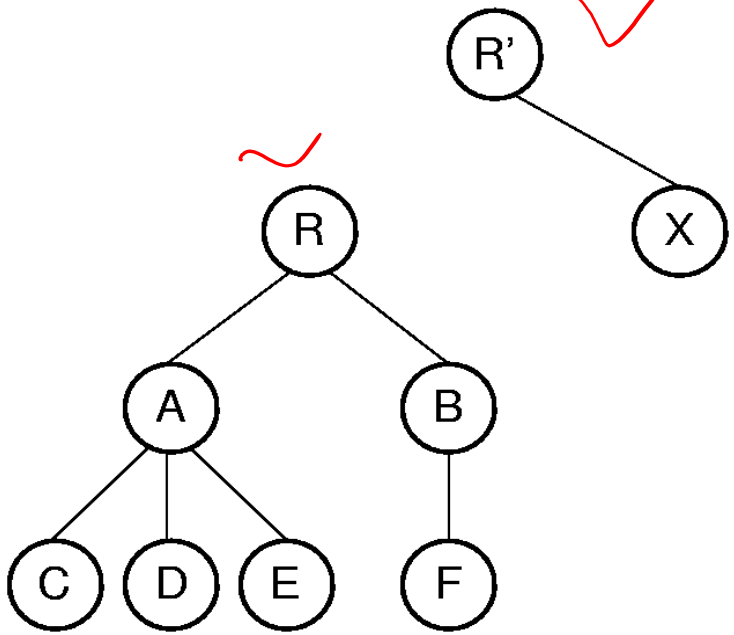
可以把树的整体深度限制在O(log n)

6.3 树的实现

- 子结点表表示法（见慕课）
- **左子结点/右兄弟结点表示法**
- 动态结点表示法（见慕课）
  - 子结点数+相应的子结点指针（子结点数目固定）
- 动态结点表示法（见慕课）
  - 子结点指针链表（子结点数目不固定）

树的实现

——Leftmost Child/Right Sibling (1)

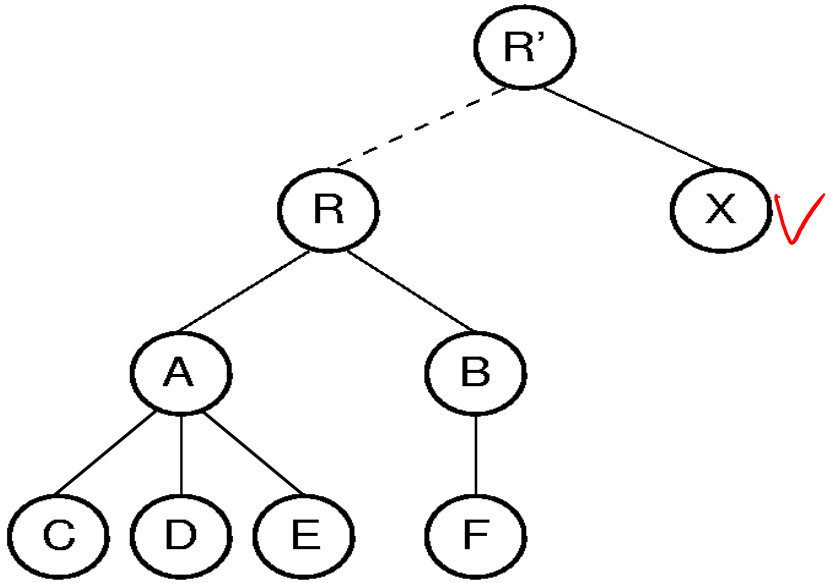


	Left	Val	Par	Right
1		R		
3		A	0	2
6		B	0	
		C	1	4
		D	1	5
		E	1	
		F	2	
8		R'		
		X	7	

左子结点/右兄弟表示法在教材上实现为数组，使用静态指针，空间效率高，且每个结点的空间长度固定

left：指向第一个孩子结点的指针或索引
right：指向第一个右兄弟结点的指针或索引
val和par含义同父指针表示法

Leftmost Child/Right Sibling (2)



树的归并

	Left	Val	Par	Right
1		R	7	8
3		A	0	2
6		B	0	
		C	1	4
		D	1	5
		E	1	
		F	2	
0		R'	-1	
		X	7	

只调整了3个指针：
R'的左孩子、R的父指针和右兄弟

树的顺序表示法

- 通常把结点的值按照它们在先根遍历中出现的顺序存储起来，描述树形状的充足信息同时也被存储起来。
- 目的：存储一系列结点的值，其中包含尽可能少、但是对于重建树结构必不可少的信息
- 优点：节省空间，因为无需存储指针
- 缺点：只允许顺序查找
- 应用：输入树数据、高效率磁盘存储、序列化树结构以便于在分布式环境中传输

树的顺序表示法

- 问题?：怎么建立存储内容，怎么恢复成森林或二叉树

先讨论二叉树的顺序表示法

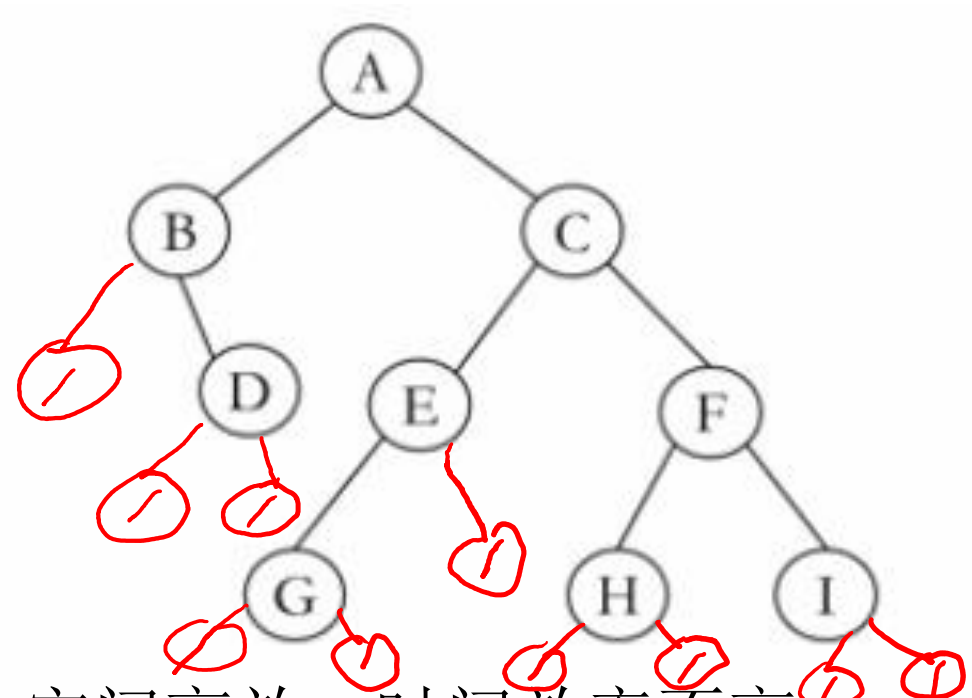
（1）先根遍历序列表示法

把树的结点值按照先根遍历序列的顺序列出，把所有非空结点看成分支结点，只有空指针NULL才被当做叶结点。

树的顺序表示法

例6.5 对于下图中的二叉树，相应的顺序表示结点如下（假定"/"表示空指针NULL）：

AB/D//CEG///FH//I//



空间高效，时间效率不高

如何重构树？
按照先根遍历序列，可以确定A是树根；B紧跟其后，必是左孩子的根结点，其后跟/，说明B的左孩子为空，/后为D，说明D为B的右孩子，而D后跟//，说明它是叶结点，此后出现的C是A的右孩子；

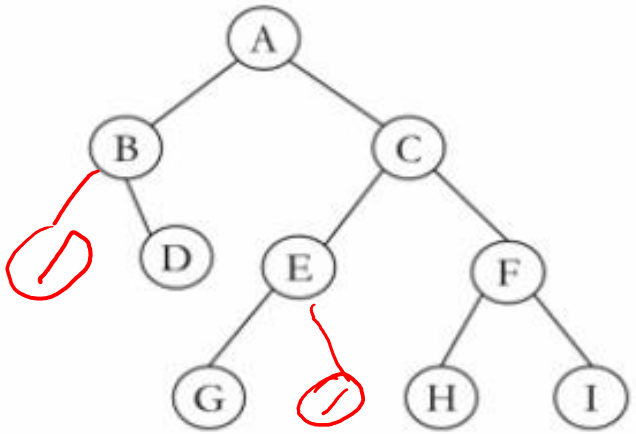
树的顺序表示法

- 应用中的问题：
- 为了找到A的右孩子，需要把其左子树都查找一遍。

- 空间开销分析
  - 比使用指针高效
  - 根据满二叉树定理，其中空指针数目=n+1，结构性开销占了1/2

带标记的先根序列表示法

- 显式地在每个结点后标识出它是叶结点还是分支结点
  - 分支结点加标记（'），而叶结点不加任何标记
  - 分支结点的空子结点以“/”表示，而叶结点的空子结点不加表示。
  - 因为不需要存储叶子结点的空指针，所需开销更小
  - 结点中需要存储标记位的空间



例6.6 上图可以表示为：

A‘B’/DC'E'G/F‘HI

使用标记位向量的先根序列表示法

- 存储标记位的另一种方式是提供一个单独的位向量来表示各个结点的状态。
- 树中每个结点对应于位向量中的一位
  - 1表示分支结点，0表示叶结点

- 例6.7 图6.17中的位向量（包括结点B和E的空节点）如下：
- 11001100100

树的顺序表示法

- 用顺序表示法存储树不仅要给出一个结点是分支结点还是叶节点，还必须给出有多少个子结点的信息。
- 一种替代的方法是给出一个结点的子结点表结束的位置，如：用特殊标记"）"来标明子结点表的结束。

树的顺序表示法

- 例6.8 对于下图中的树，结点表为：RAC)D)E))BF)))

FBR 的子结点结束

- 所有叶节点后面都跟着一个“）”。如果一个叶节点是其父结点的最后一个子结点，则其后将有两个以上连续的“）”，如下图中的E和F。