Chapter 6. Non-Binary Trees

1

1

Content

6.1 General tree

6.2 Parent Pointer Implementation

6.3 General tree Implementation

6.4 k-ary Trees(k叉树)

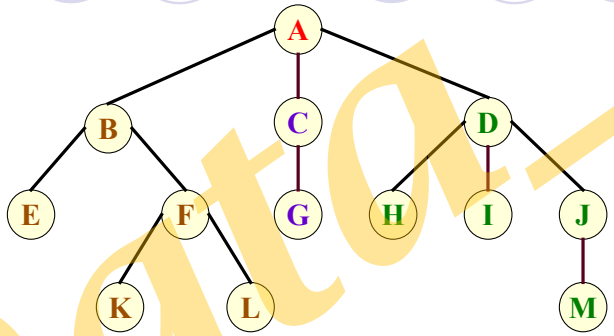
6.5 Converting General tree to a Binary Tree

6.6 Sequential tree Implementations

2

2

6.1 General Trees



3

3

General Tree Node ADT

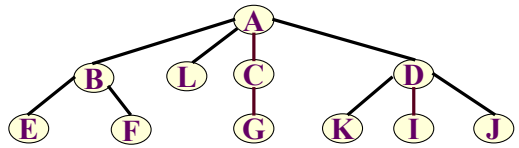
```
template <class Elem> class GTNode {     // General tree node ADT
......
public:
  GTNode(const Elem&);   // Constructor
  ~GTNode();          // Destructor
  Elem value();       // Return value
  bool isLeaf();      // TRUE if is a leaf
  GTNode* parent();   // Return parent
  GTNode* leftmost_child(); // Return First child
  GTNode* right_sibling();  // Return Right sibling
  void setValue(Elem&);     // Set value
  void insert_first(GTNode<Elem>* n);  //insert First child
  void insert_next(GTNode<Elem>* n);  //insert right next sibling
  void remove_first();  // Remove first child
  void remove_next();  // Remove next sibling
};
```

4

4

General Tree Traversal

- 深度优先遍历
  - preRoot traversal(前根遍历): R Cs
  - postRoot traversal(后根遍历): Cs R
- 广度优先遍历/按层遍历



R Cs : A B E F L C G D K I J
Cs R: E F B L G C K I J D A
广度优先遍历: A BLCD EFGKIL

5

5

General Tree Traversal

```
template <class Elem>
void preRoot (GTNode<Elem>* subroot) {
  GTNode<Elem>* temp;
  if (subroot==NULL) return;
  if (subroot->isLeaf()) { cout << "Leaf: "; cout << subroot->value() << "\n"; }
  else
  {
    cout << "Internal: ";
    cout << subroot->value() << "\n";
    for (temp = subroot->leftmost_child(); temp != NULL; temp = temp->right_sibling())
      preRoot(temp);
  }
}
```

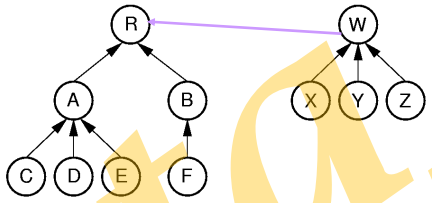
有兴趣的同学可仿照preRoot的代码自己编写postRoot函数

6

6

6.2 Parent pointer Implementation of GT

---simplest, array based



Ex.
FIND(3); 0
UNION(1,8)

Parent's Index	/	0	0	1	1	1	2	0	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

多棵树可保存在同一对数组中

用2个数组，1个整型变量来描述一(多)棵树
- ✓ 1个数组按层存放结点元素值
- ✓ 1个数组存放结点的双亲下标
- ✓ 1个整型变量存放数组的尺寸

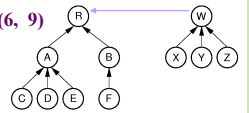
Most often used Operation:
FIND：找到结点所在树的根
UNION: 将两结点所在树合成一棵树

7

7

```
// Parent pointer Implementation of General tree for UNION/FIND
template <class Elem>  Class ParPtrTrees {
private:
  Elem  *data;
  int    *parentIndex;
  int     maxSize;
  int  FIND(int curr) const  {
       while (parentIndex [curr]!=ROOT)    curr = parentIndex[curr];
       return curr;  // At root
    }
public:
  ……
    void UNION(int  a,  int b)  {
         int root1 = FIND(a);     int root2 = FIND(b);
         if (root1 != root2)   parentIndex[root2] = root1;
    }
};
```

UNION(6, 9)



Parent's Index	/	0	0	1	1	1	2	0	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

8

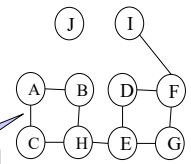
8

An application of Parent pointer Implementation of GT

----Equiv Class(等价类) Processing (1)

等价关系

已知10个元素组成的集合 S={A, B, C, D, E, F, G, H, I, J}，上的连通(相等)关系R为：
{ <A,B>, <A,C>, <B,H>, <C,H>, <A,H>, <H,E>, <E,G>, <D,E>, <D,F>, <E, F>, <F,G>, <F,I> }



省略环和传递边

要解决的问题：
对S中元素进行聚类（求R对应的分划）

用计算机编程实现
输入： 1） 集合S； 2） S上的连通关系R
输出：所有等价类(分划块)：
1）等价类个数； 2） 每个等价类中的元素

R关系图的简化表示，根据自反性去掉了环，根据传递性去掉了传递边，根据对称性省掉了箭头

[A]={A, B, C, D, E, F, G, H, I}
[J]={J}

9

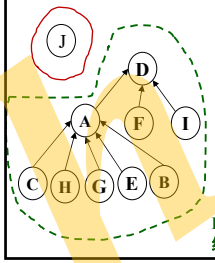
9

An application of Union/Find

----Equiv Class (等价类) Processing (2)

已知集合 S={A, B, C, D, E, F, G, H, I, J}，及S上连通关系R

{<A,B>, <A,C>, <B,H>, <C,H>, <A,H>, <H,E>, <E,G>, <D,E>, <D,F>, <E, F>, <F,G>, <F,I> }

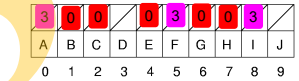


Step1：将S的每个元素作为根节点构建一个包含10棵树的森林

ParPtrTrees：

Step2：依次输入R中元素，并据此执行union操作；直到用完所有信息

<A,B>，UNION(0,1);
<A,C>，UNION(0,2);
<B,H>，UNION(1,7);
<C,H>，UNION(2,7);
<A,H>，UNION(0,7);
<H,E>，UNION(7,4);
<E,G>，UNION(4,6);
<D,E>，UNION(3,4);
<D,F>，UNION(3,5);



<E,F>，UNION(4,5);
<F,G>，UNION(5,6);
<F,I>，UNION(5,8);

聚类完成,done！

ParPtrTrees中树的棵树，即为等价类的个数，同一棵树中的元素组成的集合，即为一个等价类

10

10

Want to keep the height of united tree smaller under low cost

Weighted union rule

Rule1: Join the tree with fewer nodes to the tree with more nodes.

Rule2: Find root 的同时，set root 作为当前结点的父亲

Path Compression Rule

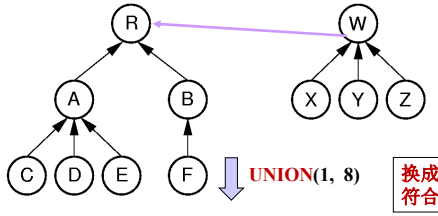
11

11

Weighted union rule

---Join the tree with fewer nodes to the tree with more nodes.

思考：如何代码化此规则？



UNION(1, 8)

换成UNION(8, 1)符合此规则吗？

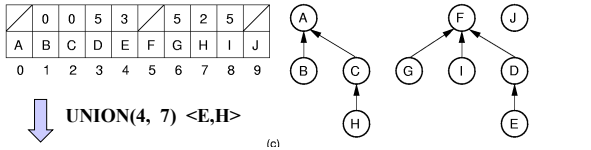
Node Index	0	1	2	3	4	5	6	7	8	9	10
Label	R	A	B	C	D	E	F	W	X	Y	Z
Parent's Index	/	0	0	1	1	1	2	0	7	7	7

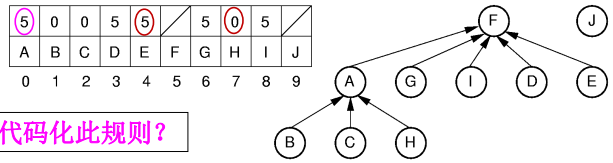
12

12

Path Compression Rule

-- Find root 的同时，将root作为当前结点的父亲





如何代码化此规则？

```
int  FIND(int curr) const {
     if (parentIndex[curr] == ROOT) return curr;
     return  parentIndex[curr] = FIND(parentIndex[curr]);
}
```

13

13

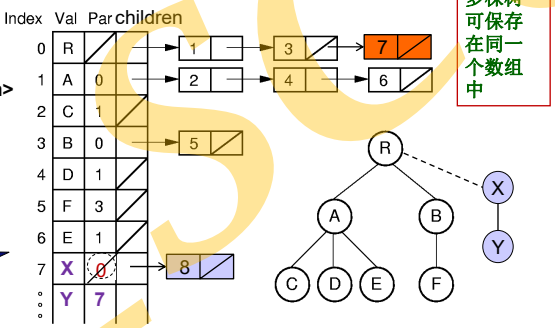
6.3 General tree Implementation

1） Lists of Children(孩子链表表示法）

基于数组(元素为结构体)

```
Template<class Elem>
class  TNode1 {
private:
   Elem  val;
   int  parent;
   LList<int>  *child;
public:
……
};
```

结点



多棵树可保存在同一个数组中

when two trees share the same array, Union is simply

Easy for finding parant and children

but difficult for finding right_sibling: 先找双亲，再从双亲的孩子链找兄弟

14

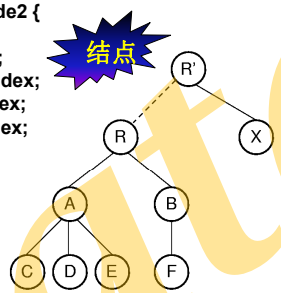
14

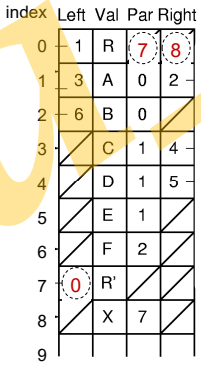
2）Leftmost Child/Right Sibling(左孩子/右兄弟表示法)

基于数组(元素为结构体)

```
Template<class Elem>
class  TNode2 {
private:
   Elem val;
   int  parIndex;
   int  lcIndex;
   int  rsIndex;
public:
……
};
```

结点





多棵树可保存在同一个数组中

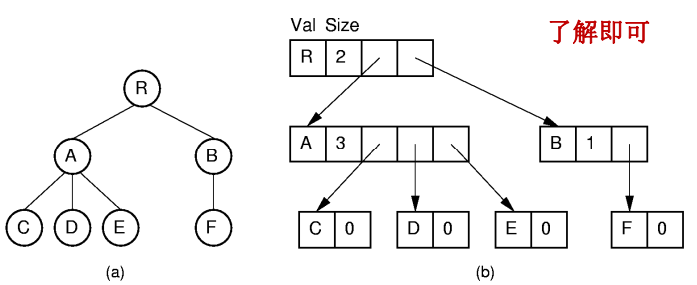
when two trees share the same array, union is simply

Easy for finding parant，first children and right_sibling

15

15

3） Dynamic Linked Implementations（链式, 基于指针）



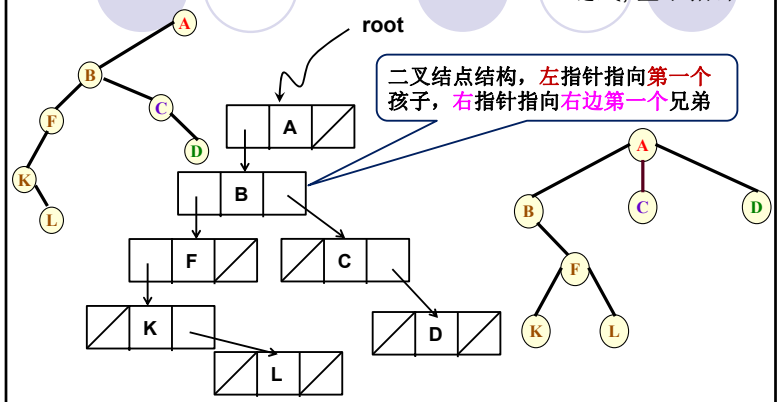
了解即可

16

16

4）Dynamic Leftmost Child/Right Sibling Implementations

----链式，基于指针

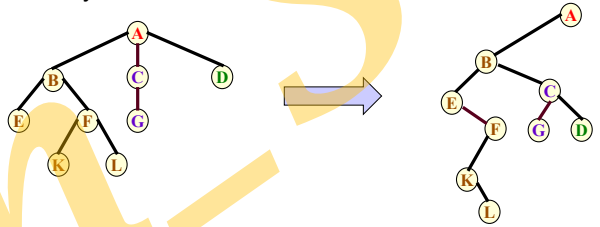


17

17

6.4 Converting General tree to a Binary Tree

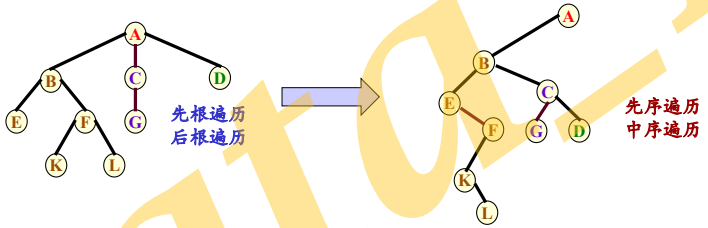
1. make mostLeft child as left child, and make right sibling as right child (Leftmost Child/Right Sibling)
2. Begin from the root, 从上到下从左到右，for each node, Use step 1）will convert any general tree to a binary tree.



18

18

树的各种操作均可由对应二叉树的操作来完成。



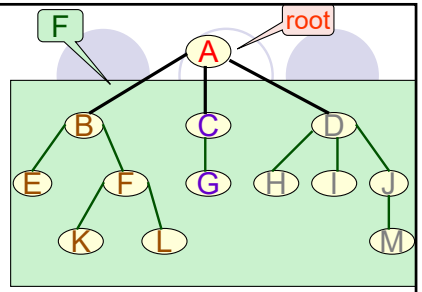
应当注意的是，和树对应的二叉树，其左、右子树的概念已改变为：左是第一个孩子，右是最近兄弟。

19

19

森林

是m(m≥0)棵互不相交的树的集合 $F=\{T_1,T_2,\ldots T_m\}$



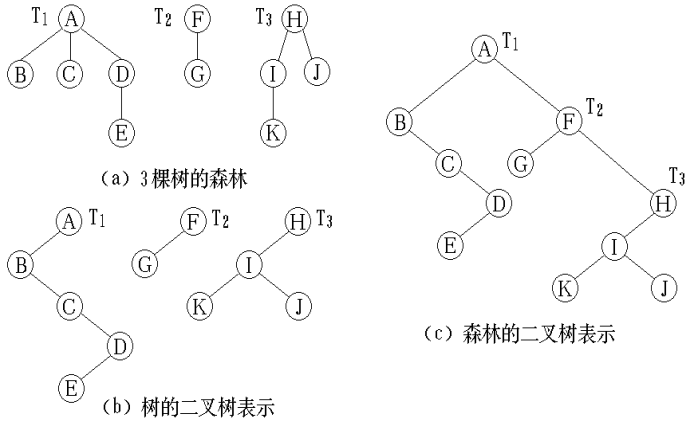
任何一棵非空树可描述为一个二元组

Tree =（root，F）

其中 root 称为根结点，F 称为子树森林

20

森林与二叉树的转换



（a）3棵树的森林

（b）树的二叉树表示

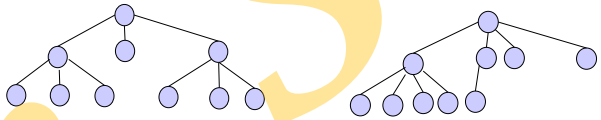
（c）森林的二叉树表示

Step1： 将每棵树转换为二叉树
Step2： 从k=1开始，将第（k+1）棵二叉树作为第k棵二叉树的根的右孩子

21

6.5 k-ary Trees(k叉树)

For k-ary tree(k叉树), any node cannot have more than k sub-trees



full 3-ary tree　　Complete 4-ary tree

22

22

6.6 Sequential tree Implementations/树的序列表示

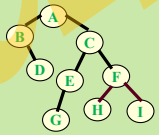
有兴趣的同学可课后自学

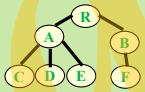
Goal: Store a series of nodes values with minimum information needed to reconstruct the tree structure

Method: List node values in the order they would be visited (preorder/inorder/postorder/ Breadth-first traversal), at the same time add some symbol to guard the reconstruct the of tree

Example1: AB/D//CEG///FH//I//

Or ABC/DEF////G/HI
Here ' / ' stands for NULL





Example2: RAC)D)E))BF)))

here, ')' mark the end of each subtree

23

23

Sequential Implementations (2)

Advantage: Saves space.

Disadvantage: allows only sequential access

Using: save the node value on disk, transmit between computer, do operations after reconstruction tree structure

24

24

25