6 Non-Binary Trees

Contents

Contents

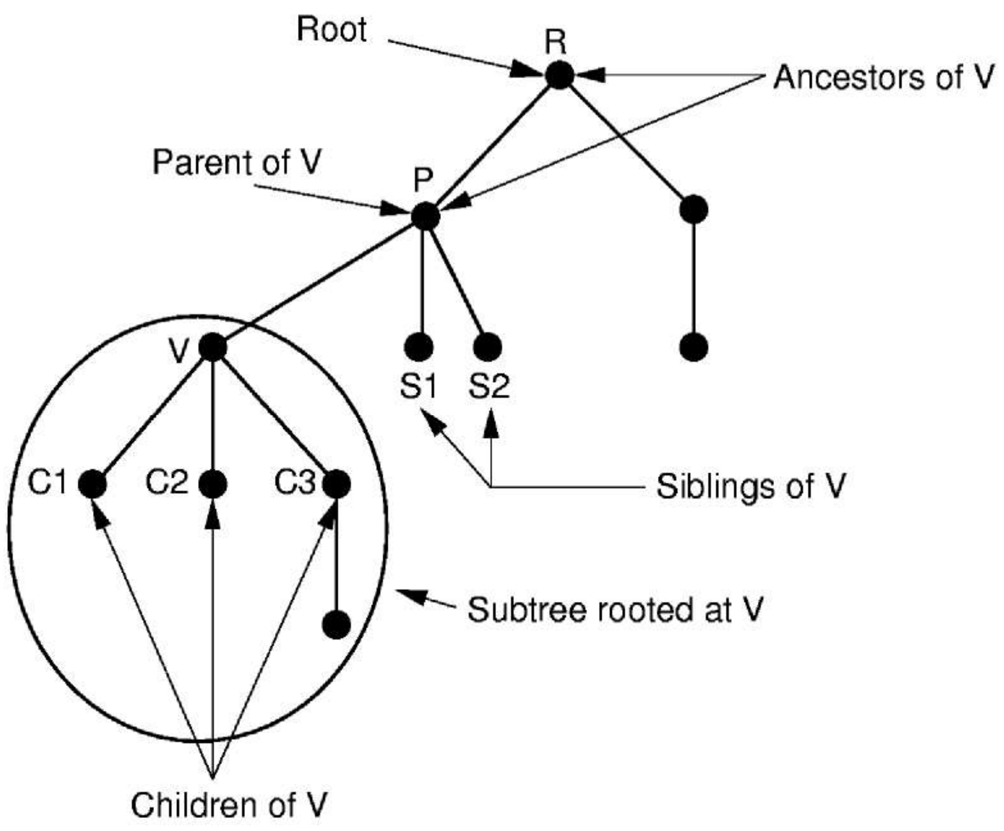
6.1 General Tree Definitions and Terminology

General Trees

A Tree is a finite set of n ($n>0$) nodes such that

- **One and only one node R, is called the root of T.**
- **The remaining nodes are partitioned into m($m \geq 0$) disjoint subsets T_0, T_1…..T_{m-1}, each of which is a tree, and whose roots R_0, R_1…..R_{m-1}, respectively, are children of R.**
- **The subsets T_i ($0 \leq i < m$) are said to be subtrees of T.**

General Trees

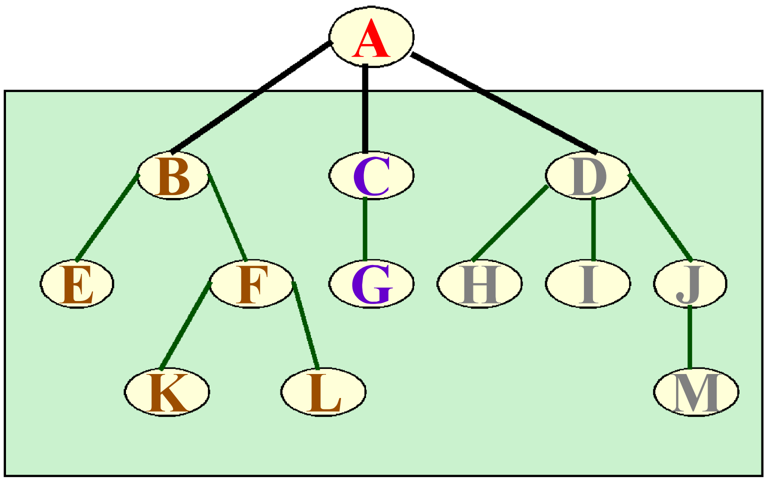


General Trees

A node’s **out degree** is the number of children for that node.

– Out degree of node D is 3.

A **forest** is a collection of one or more trees.



General Tree Node

```
// General tree node ADT
template <typename E> class GTNode {
public:
  E value();          // Return value
  bool isLeaf();         // TRUE if is a leaf
  GTNode* parent();      // Return parent
  GTNode* leftmostChild(); // First child
  GTNode* rightSibling();  // Right sibling
  void setValue(E&);      // Set value
  void insertFirst(GTNode<E>*);
  void insertNext(GTNode<E>*);
  void removeFirst(); // Remove first child
  void removeNext();  // Remove sibling
};
```

General Tree ADT

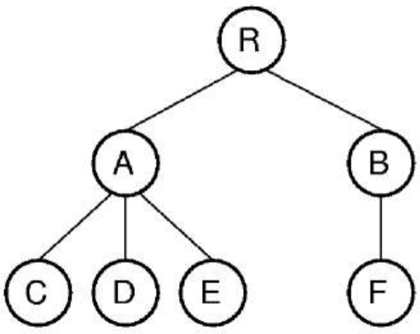
```
// General tree ADT
template <typename E> class GenTree {
public:
  void clear();    // Send nodes to free store
  GTNode<E>* root();       // Return the root
  //Combine two subtrees
  void newroot(E&, GTNode<E>*, GTNode<E>*);
  void print();  // Print a tree
};
```

General Tree Traversal

Preorder: First visit the root of the tree, then performs a preorder traversal of each subtree from left to right;

Postorder: First performs a postorder traversal of the root's subtrees from left to right, then visit the root;

Inorder traversal does not have a natural defination.



Preorder traversals

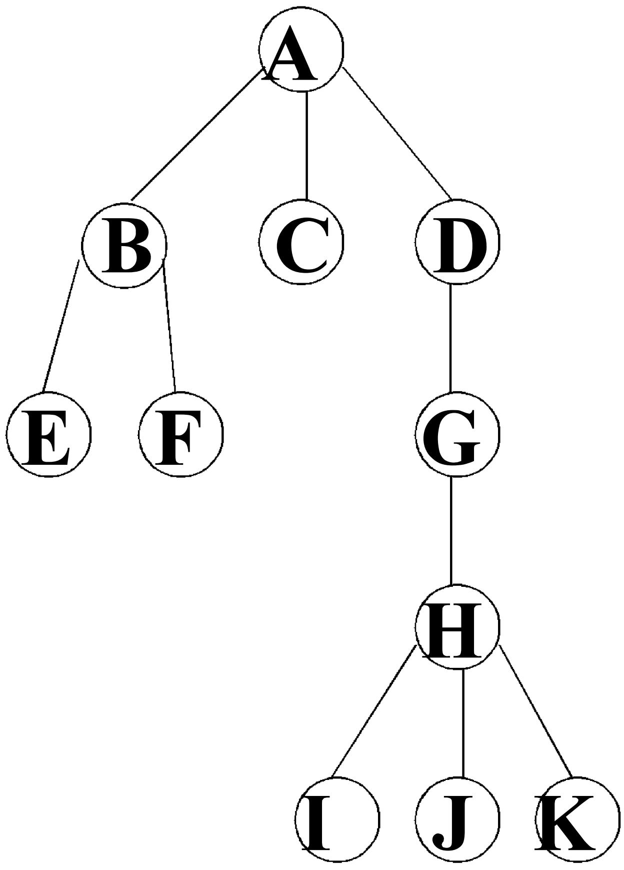
A B E F C D G H I J K

Postorder traversals

E F B C I J K H G D A

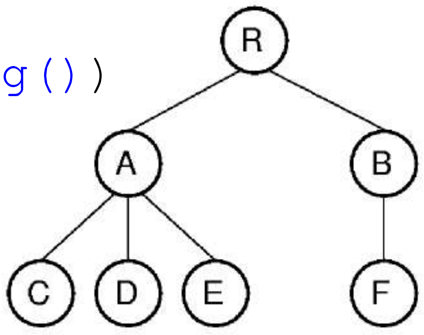
Level traversals

A B C D E F G H I J K



General Tree Traversal

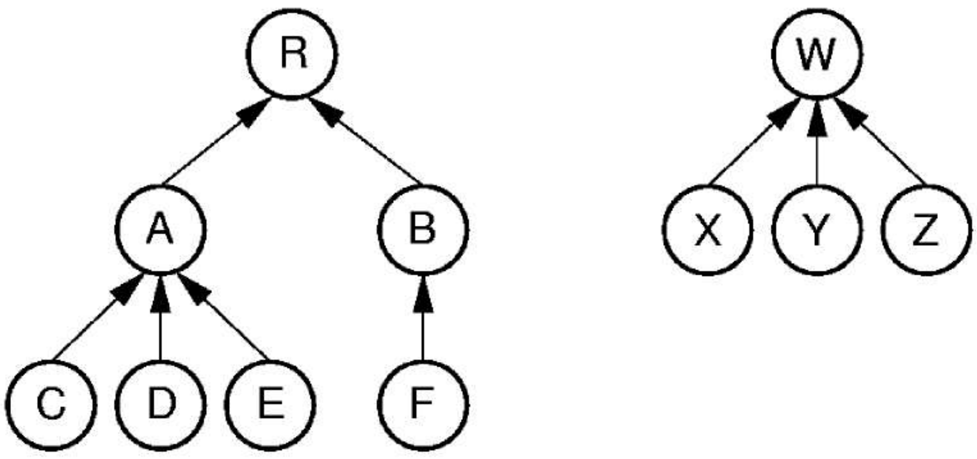
```
//Print using a preorder traversal
void printhelp(GTNode<E>* root) {
  //访问根
  if (root->isLeaf()) cout << "Leaf: ";
  else cout << "Internal: ";
  cout << root->value() << "\n";
  //从左到右访问各子树
  for (GTNode<E>* temp =root->leftmostChild();
         temp != NULL;
         temp = temp->rightSibling())
    printhelp(temp);
}
```



Contents

6.2 The Parent Pointer Implementation

Parent Pointer Implementation



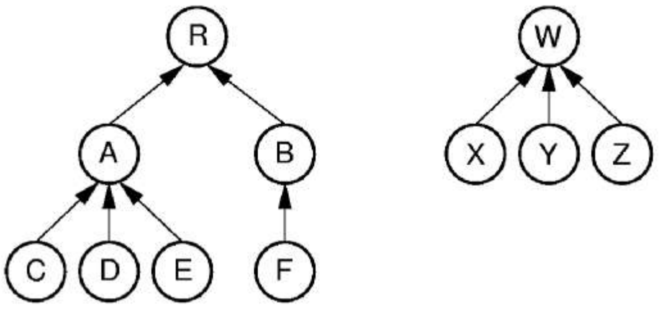
Parent's Index		0	0	1	1	1	2		7	7	7
Label	R	A	B	C	D	E	F	W	X	Y	Z
Node Index	0	1	2	3	4	5	6	7	8	9	10

Parent Pointer Implementation

Advantages: Can answer the following question easily: Given two or more nodes, are they in the same tree?

Disadvantages: It is inadequate for such important operations as finding the leftmost child or the right sibling

for a node.



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

ADT for parent pointer implementation

```
Class ParPtrTree{
Private:
  int* array;
  int   size:
  int FIND(int) const;//Find root
Public:
  ParPtrTree(int);
  ~ParPtrTree(){delete[] array;}
 void UNION(int,int)//Merge equivalences
 bool differ(int,int)//TRUE if not in same tree
};
```

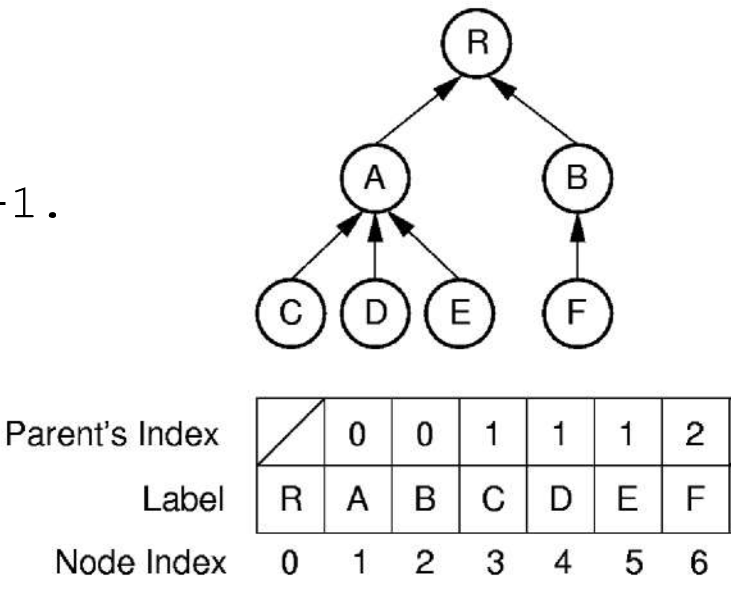
Implementation of Find()

```
int ParPtrTree::FIND(int curr) const{
  while (array[curr]!=ROOT)
      curr = array[curr];
  return curr;
}
```

假设根的双亲值为-1，ROOT=-1.

e.g. FIND(4)

  curr: 4->1->0



Parent's Index		0	0	1	1	1	2
Label	R	A	B	C	D	E	F
Node Index	0	1	2	3	4	5	6

Implementation of differ()

The parent pointer representation is good for answering:

- Are two elements in the same tree?

```
// Return TRUE if nodes in different trees
bool ParPtrTree::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
}
```

Implementation of Union()

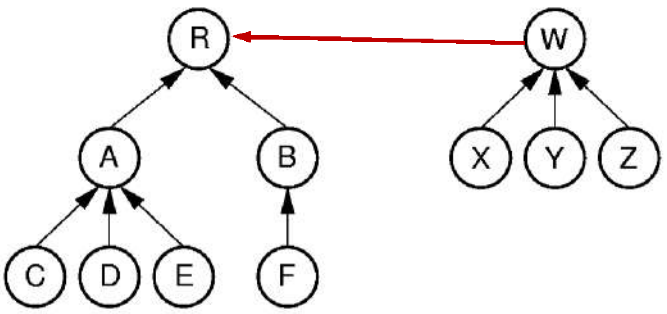
```
//Merge two subtrees
void ParPtrTree::UNION(int a, int b) {
  int root1 = FIND(a); // Find root for a
  int root2 = FIND(b); // Find root for b
  if (root1 != root2) array[root2] = root1;
         //root2 as a subtree of root1
}
```

e.g. UNION(4,9)

root1=0;

root2=7;

array[7]=0;

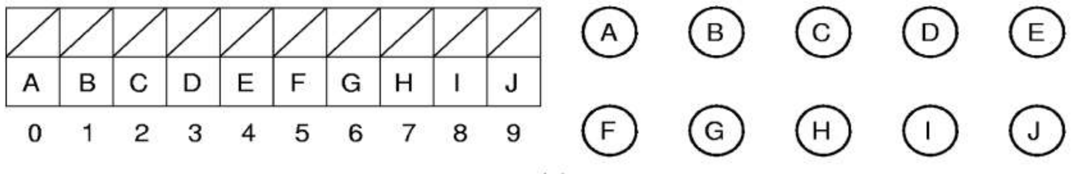


Parent's Index		0	0	1	1	1	2	0	7	7	7
Label	R	A	B	C	D	E	F	W	X	Y	Z
Node Index	0	1	2	3	4	5	6	7	8	9	10

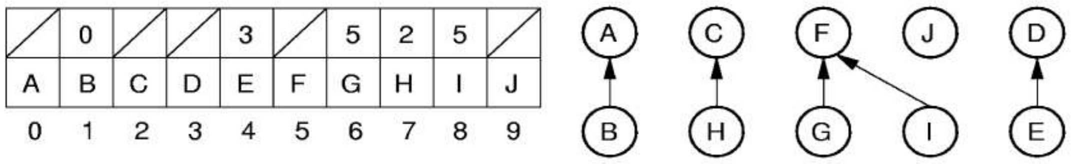
Equivalence Classes(并查集)

- Consider the problem of assigning the members of a set to disjoint subsets called **equivalence classes**.
- Examples of equivalence classes:
  - Connected components in graphs
    - When two nodes are connected, they are said to be equivalent.
  - Point clustering

Equiv Class Processing (1)

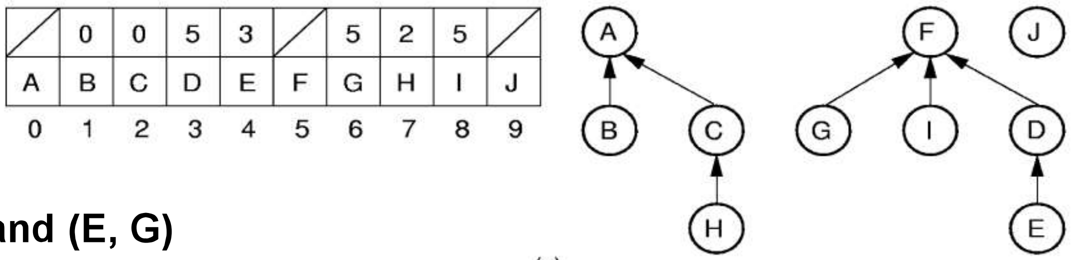


(a)



(A, B), (C, H), (G, F), (D, E), and (I, F).

(b)

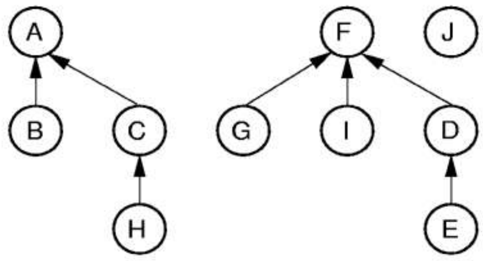


(H, A) and (E, G)

(c)

Equiv Class Processing (2)

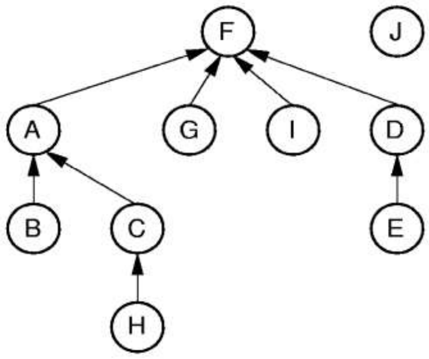
/	0	0	5	3	/	5	2	5	/
A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9



(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

(H, E)

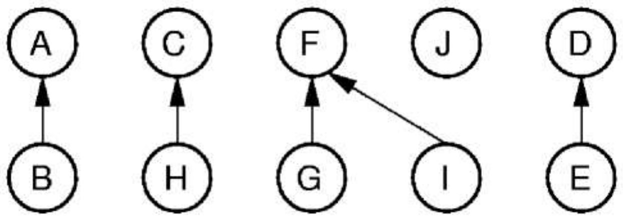


(d)

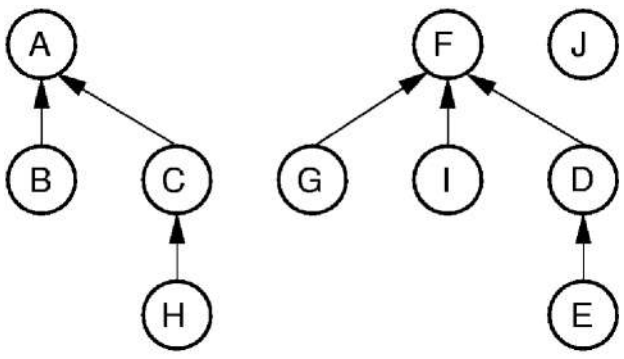
Weighted union

Keep the depth small.

Weighted union rule: Join the tree with fewer nodes to the tree with more nodes by making the smaller tree's root point to the root of the bigger tree.



(A, B), (C, H), (G, F), (D, E), and (I, F).



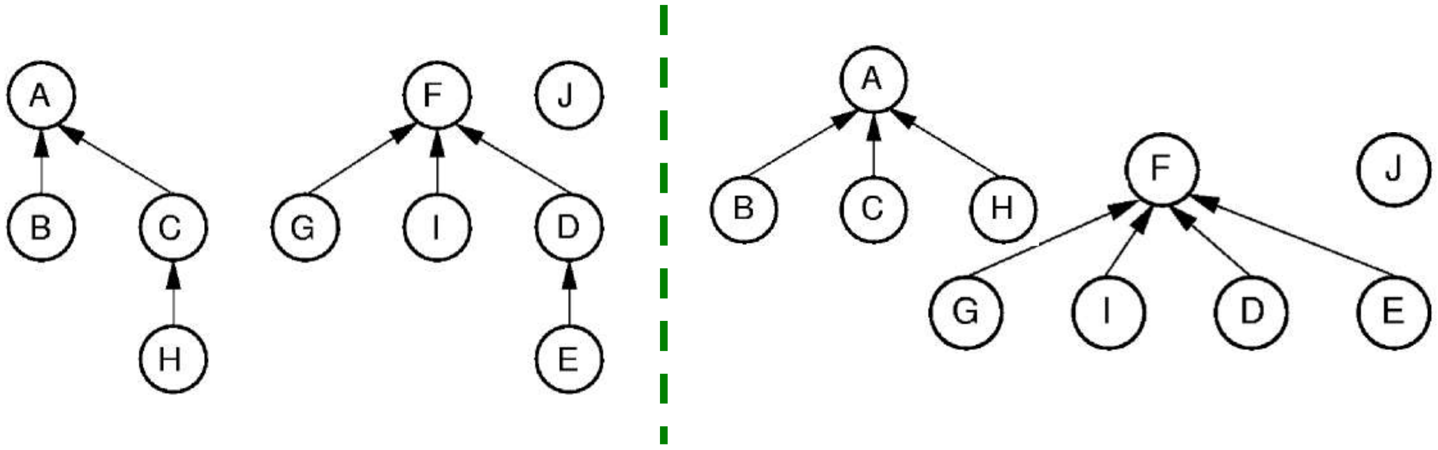
(H, A) and (E, G)

Path Compression

Path compression is a method that tends to create extremely shallow(浅的) trees.

- First, find the root of node X. Call this root R.
- Reset the parent of every node on the path from X to R to point directly to R.

Note: Path compression takes place during FIND.



Path Compression

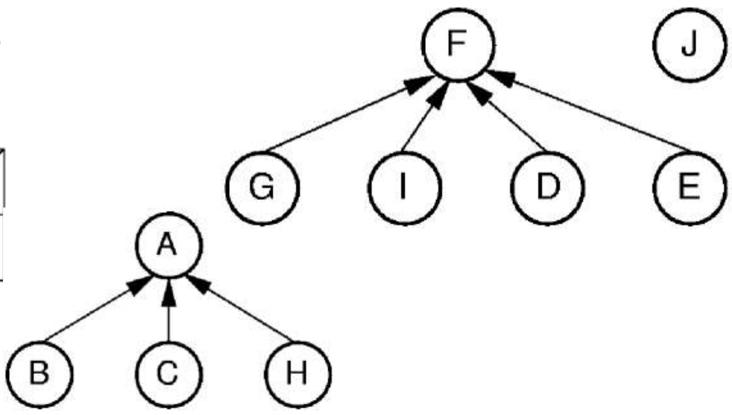
```
int ParPtrTree::FIND(int curr) const {
  if (array[curr] == ROOT) return curr;
  array[curr] = FIND(array[curr]);
  return array[curr];
} //With path compression
```

This `FIND` can: (1)returns the root of curr node, (2)makes all ancestors of curr node point to the root.

e.g. `UNION(7,4)` 即合并`(H,E)`

需要首先 `FIND(7)`, `FIND(4)`

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

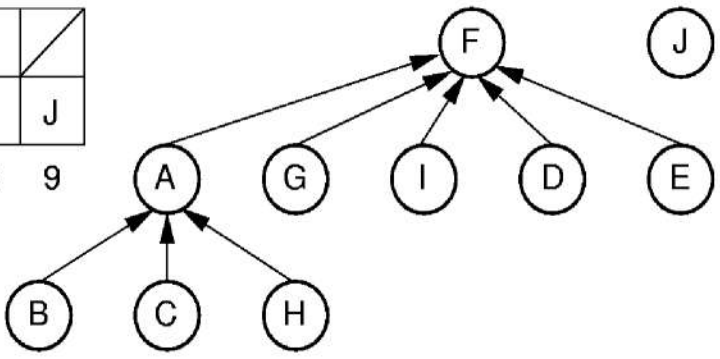


Path Compression

e.g. UNION(7,4) （续）

最后，结点0(A)做为结点5(F)的孩子加入到F的子树中。

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



Application of Eqiv. Class(1)

- **Problem 1: How many tables**

Today is Smith's birthday. He invites a lot of friends. Now it's dinner time. Smith wants to know how many tables he needs at least. You have to notice that not all the friends know each other, and all the friends do not want to stay with strangers.

One important rule for this problem is that if I tell you A knows B, and B knows C, that means A, B, C know each other, so they can stay in one table.

For example: If I tell you A knows B, B knows C, and D knows E, so A, B, C can stay in one table, and D, E have to stay in the other one. So Smith needs 2 tables at least.

Application of Eqiv. Class(2)

- **Problem 2:** 细胞统计

一矩形阵列由数字0到9组成，数字1到9代表细胞，细胞的定义为沿细胞数字上下左右还是细胞数字则为同一细胞，求给定矩形阵列的细胞个数。如阵列:

0	2	3	4	5	0	0	0	6	7
1	0	3	4	5	6	0	5	0	0
2	0	4	5	6	0	0	6	7	1
0	0	0	0	0	0	0	0	8	9

有**4**个细胞。

	C(i-1,j)
B(i,j-1)	A(i,j)

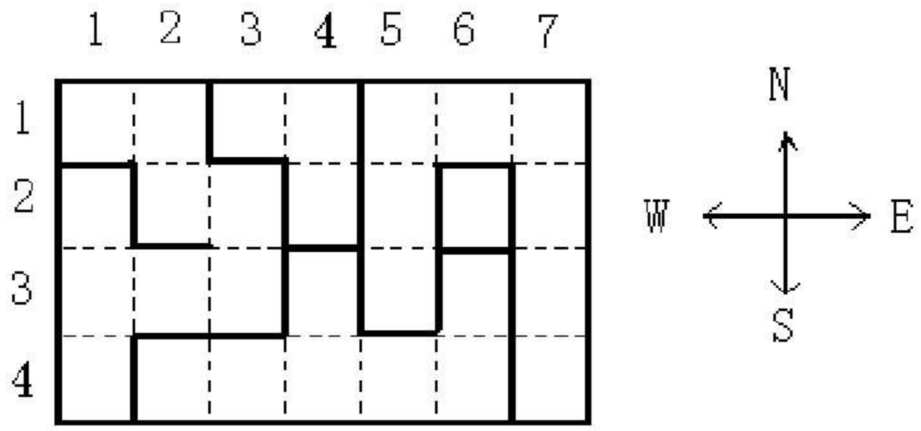
- 并查集实现
  - ◆ 逐行扫描，依次处理每一个点
  - ◆ 初始化：每个点的父亲指向本身
    //每个数是独立的一个细胞
  - ◆ 若点A(i, j)是细胞，分3种情况处理：
    (1)B和C都是细胞，则合并
    UNION(A, B), UNION(A, C)
    (2)B是而C不是，则 UNION(A, B)
    (3)B不是而C是，则 UNION(A, C)
  - ◆ 统计父亲是自身(find(i)=i)的结点数即细胞的个数

Application of Eqiv. Class(3)

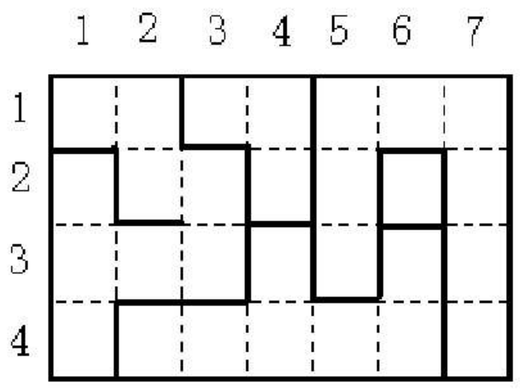
- **Problem 3: 房间问题**

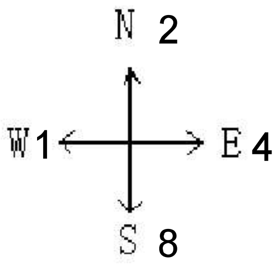
给出建筑平面图，计算：(1)房间数量；(2)最大的房间有多大；(3)拆除其中某一堵墙，以形成一个尽可能大的房间。指出该墙。

该建筑分成m*n个方块（m≤50，n≤50），每个方块可有0～4堵墙（0表示无墙）。



样例输入：
4
7
11,6,11,6,3,10,6
7,9,6,13,5,15,5
1,10,12,7,13,7,5
13,11,10,8,10,12,13





输入数据解释：

●1、2行分别是南北、东西方向的方块数。

●后面的行中，一个数字表示一个方块，数字P描述（0≤P≤15）墙的情况。P是下面的可能取的数字之和：

1(西墙W)、2(北墙N)、4(东墙E)、8(南墙S)

室内的墙被定义两次： 例如方块 (1, 1)中的南墙也被位于其南面的方块(2, 1)定义了一次。

●建筑中至少有两个房间。

算法分析：类似细胞统计

第一问：房间数量；

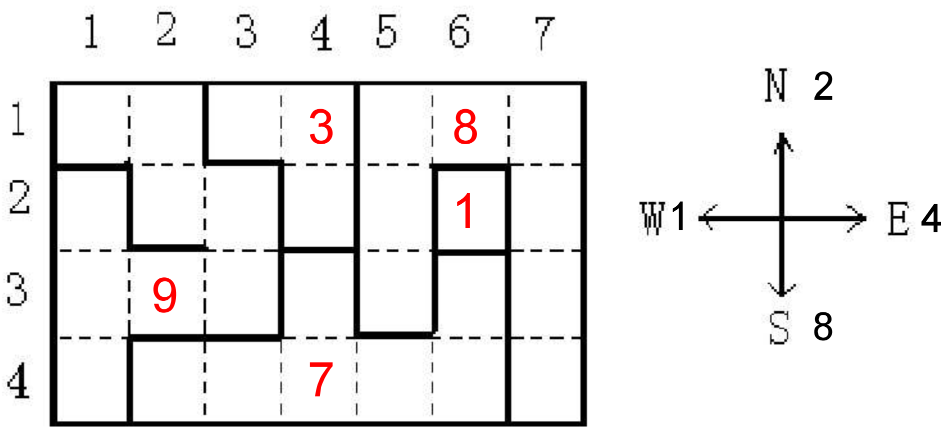
逐行扫描，处理当前格子与上方和左方的格子。

第二问：最大的房间有多大；

父亲格子设置一个统计格子数量的变量，每扫描到一个以当前格子为父亲的格子，数量加1。

第三问：拆除建筑中的某一堵墙，以形成一个尽可能大的房间。指出该墙。

枚举每一堵墙。如果是竖直方向的墙：看左右是否连通，如果连通（同父亲），拆除后不改变面积；不连通则求左右两部分面积和，比较。如果水平的墙：看上下两部分。

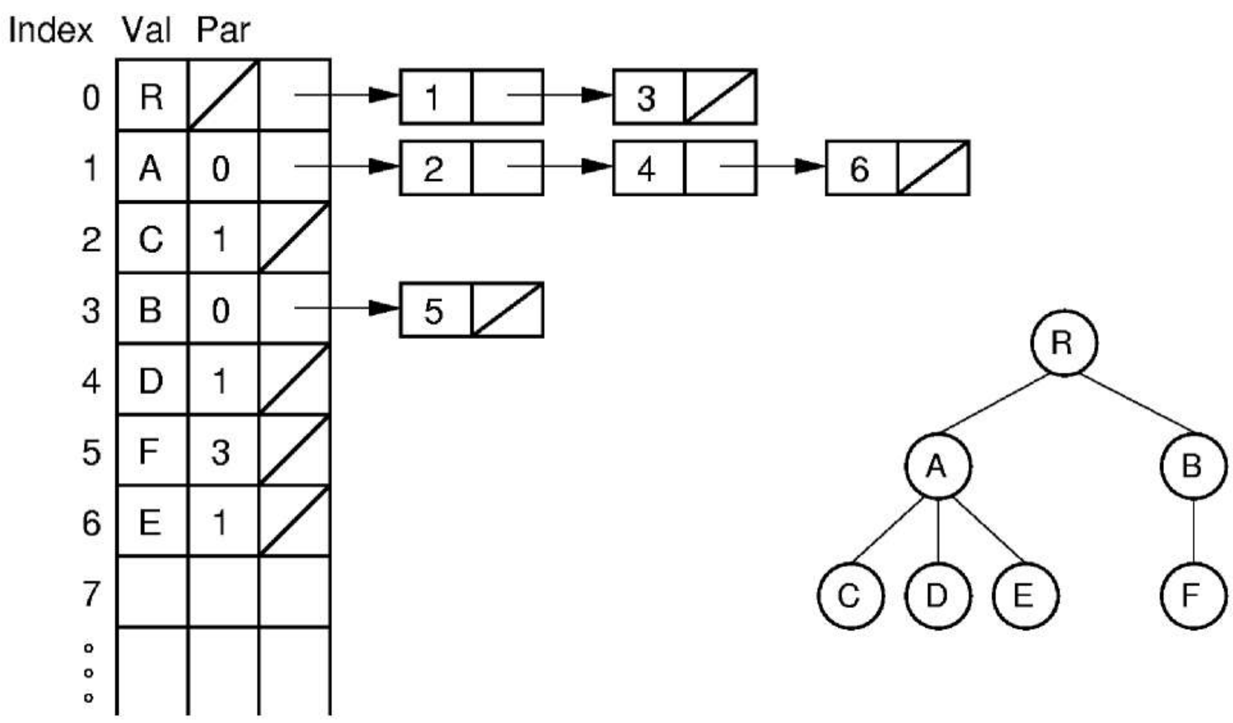


输出：	
5	有5个房间；
9	最大房间的大小：9；
(3,3)(3,4)	拆除的墙： (3,3)(3,4) 之间的墙；
16	拆除后形成的房间面积为16

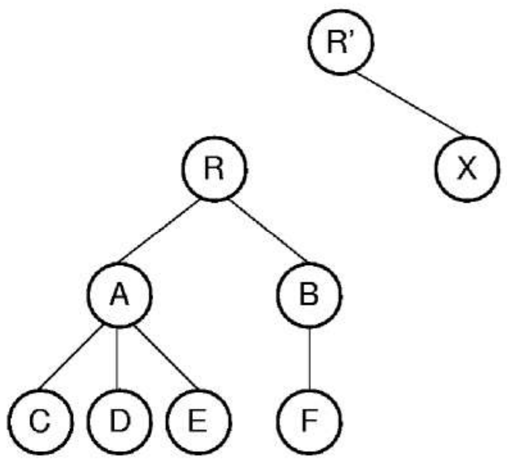
Contents

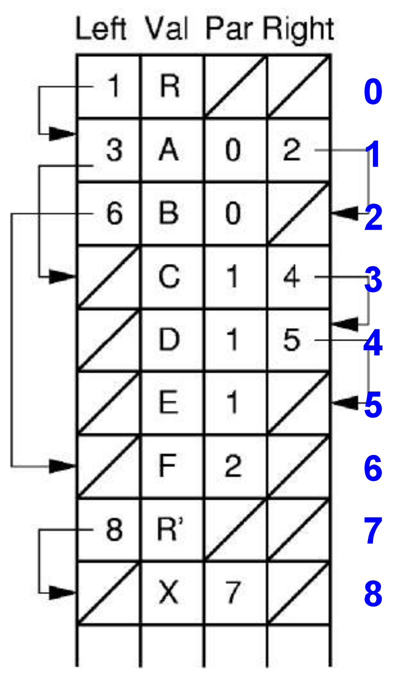
6.3 General Tree Implementation

Lists of Children



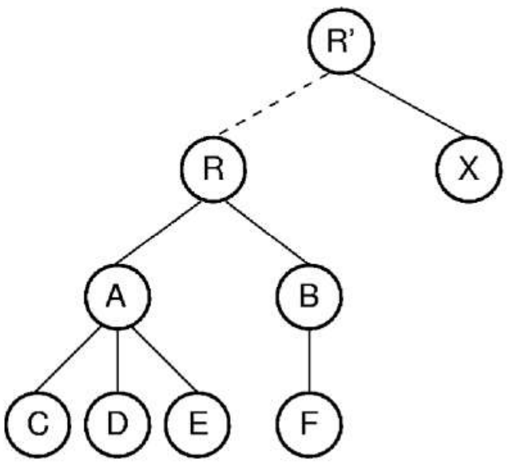
Leftmost Child/Right Sibling (1)

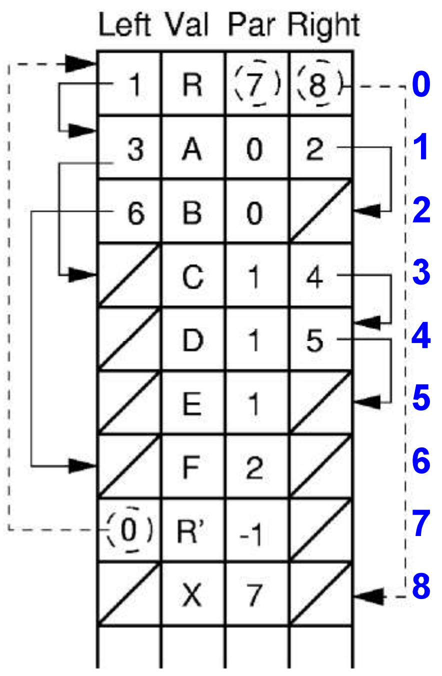




Left	Val	Par	Right	
1	R	/	/	0
3	A	0	2	1
6	B	0	/	2
/	C	1	4	3
/	D	1	5	4
/	E	1	/	5
/	F	2	/	6
8	R'	/	/	7
/	X	7	/	8

Leftmost Child/Right Sibling (2)



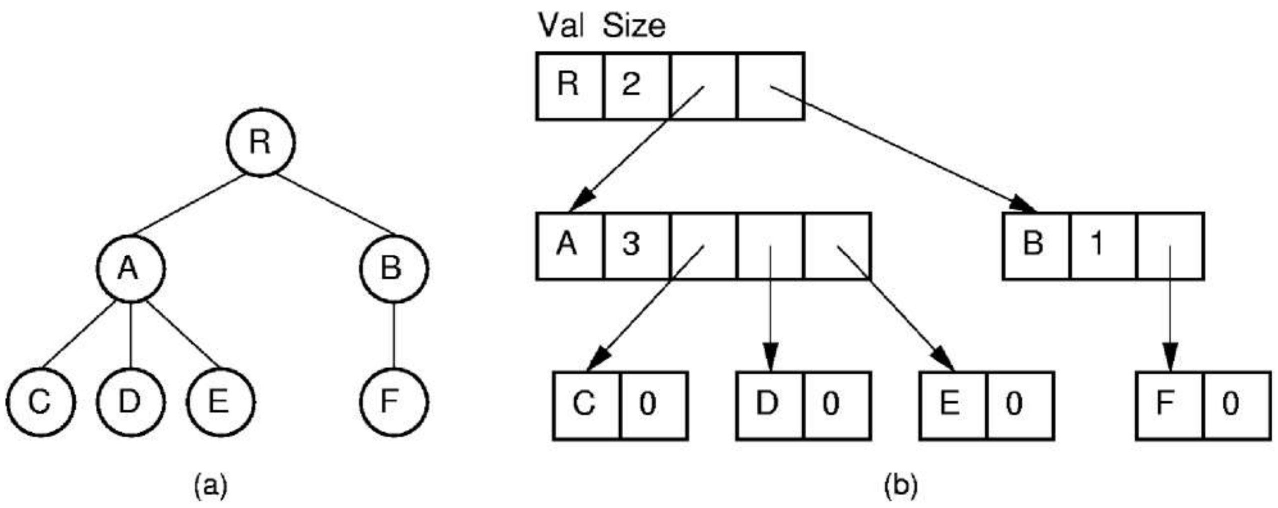


	Left	Val	Par	Right
0	1	R	7	8
1	3	A	0	2
2	6	B	0	/
3	/	C	1	4
4	/	D	1	5
5	/	E	1	/
6	/	F	2	/
7	0	R'	-1	/
8	/	X	7	/

Combining two trees

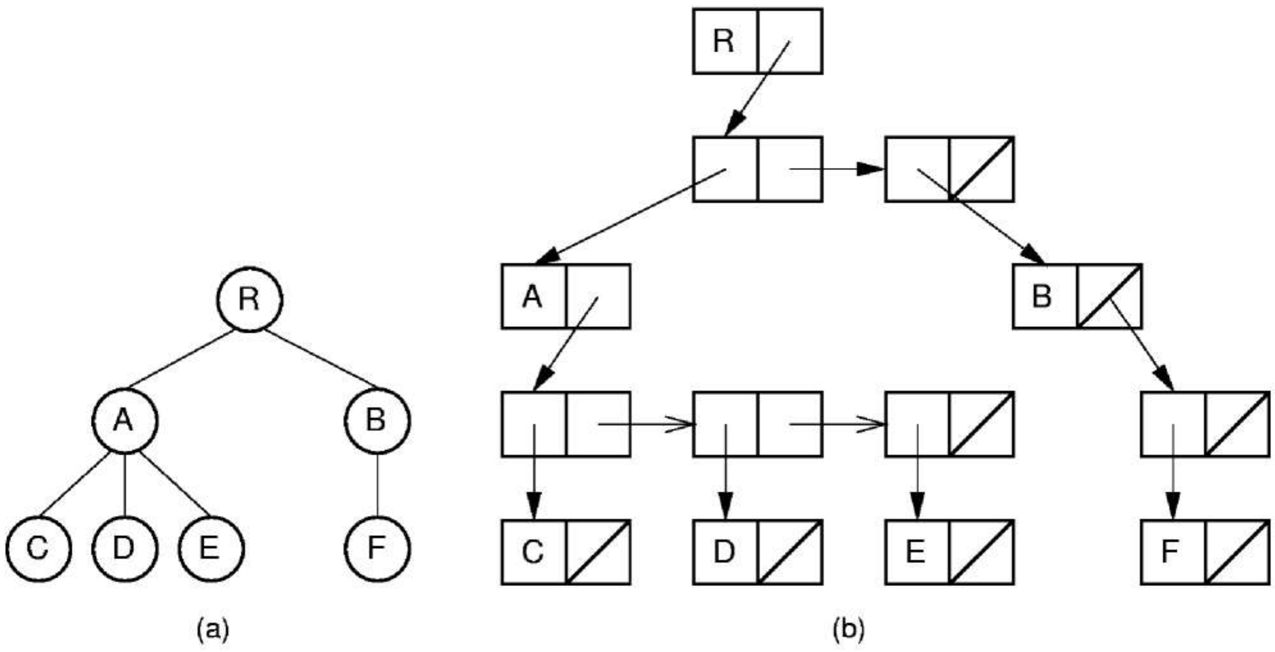
Dynamic Node Implementations (1)

- Allocate variable space for each node
- Each node stores an array-based list of child pointers

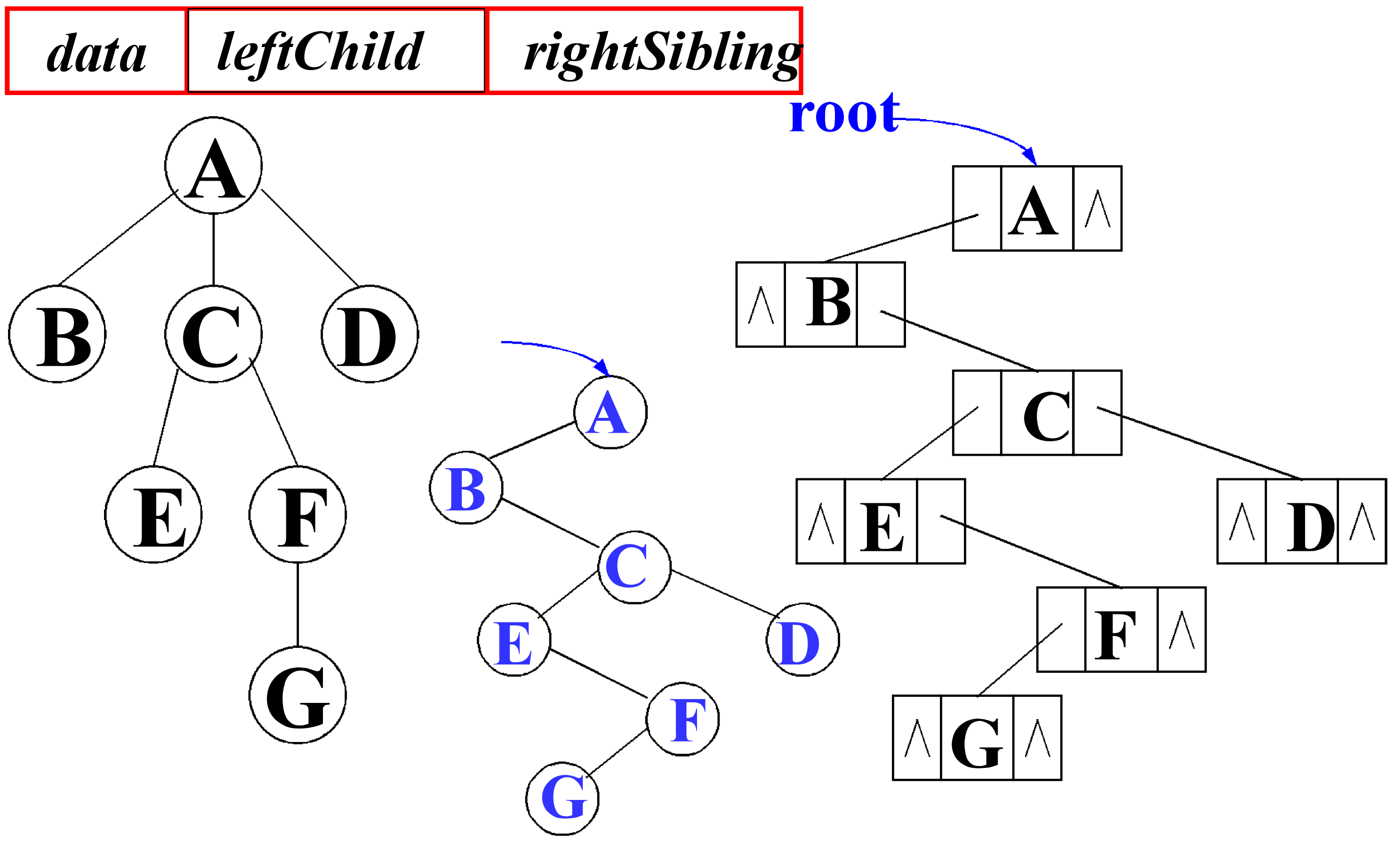


Dynamic node Implementations (2)

- Each node stores an linked list of child pointers



Dynamic Left-Child/Right-Sibling

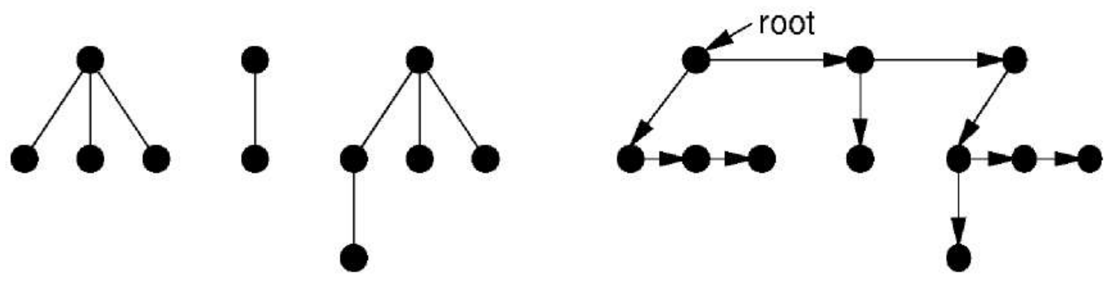


Converting to a Binary Tree

Left child/right sibling representation essentially stores a binary tree.

Use this process to convert any general tree or forest to a binary tree.

A forest is a collection of one or more general trees.



(a) (b)

Contents

6.4 *K*-ary Trees

*K*叉树

K-ary Trees

- *K*-ary trees are trees whose nodes have *K* children.
  - A binary tree is a 2-ary tree.
- *K*-ary tree nodes have a fixed number of children.
- In general, *K*-ary trees bear many similarities to binary trees.

K-ary Trees

Full and **complete** *K*-ary trees

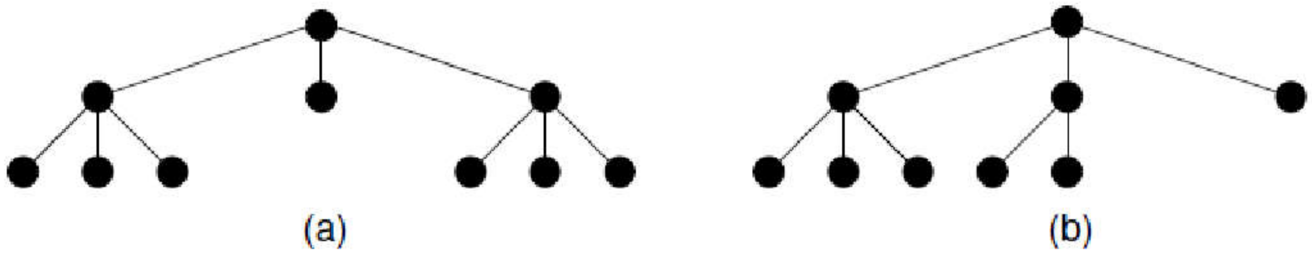


Figure 6.16 Full and complete 3-ary trees. (a) This tree is full (but not complete). (b) This tree is complete (but not full).

Contents

6.5 Sequential Tree Implementations
树的顺序(数组)存储

Sequential Implementations (1)

- List node values in the order they would be visited by a preorder traversal.
- Saves space, but allows only sequential access.
- Need to retain tree structure for reconstruction.

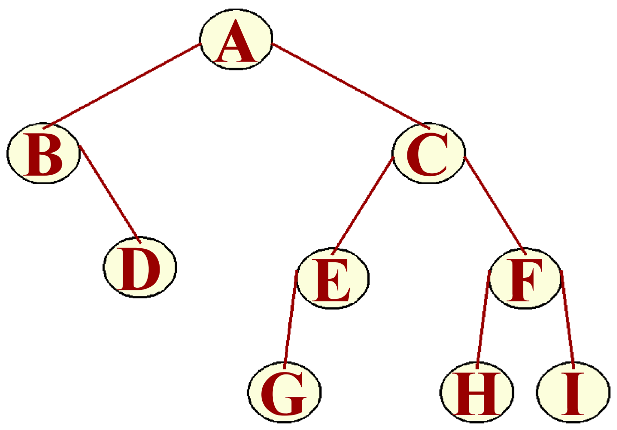
Sequential Implementations (2)

Example: For binary trees, use a symbol to mark `null` links.

preorder traversal：ABDCEGFHI

Sequential Implementation:

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

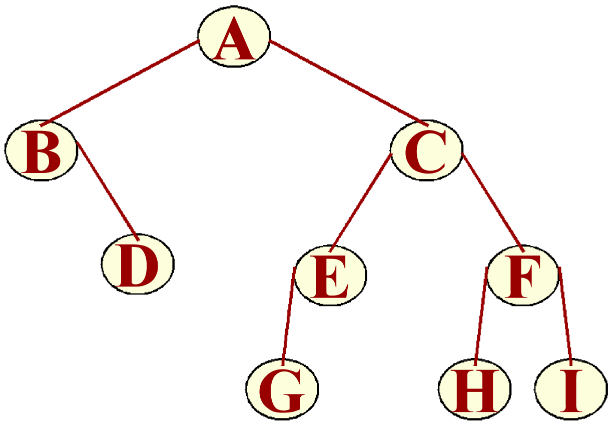


Sequential Implementations (3)

- Explicitly list with each node whether it is an internal node or a leaf.
- Each internal node has two children (full)

Example: Take the tree as a full binary trees, mark nodes as leaf or internal.

A'B'/DC'E'G/F'HI

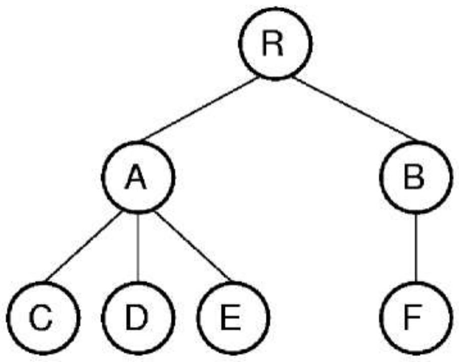


Sequential Implementations (4)

For general trees, use a special mark to indicate the end of each subtree.

Example:

RAC)D)E))BF)))



Summary

6.1 General Tree Definitions and Terminology

6.2 The Parent Pointer Implementation

6.3 General Tree Implementations

6.4 *K*-ary Trees

6.5 Sequential Tree Implementations

Homework

P223~P225

6.7

6.13

6.17