Data Structures

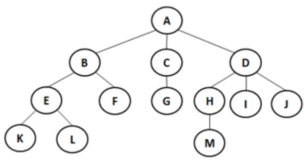
Binary Trees

Teacher : Wang Wei

1. Ellis Horowitz,etc., Fundamentals of Data Structures in C++
2. 殷人昆, 数据结构
3. 金远平, 数据结构
4. http://inside.mines.edu/~dmehta/

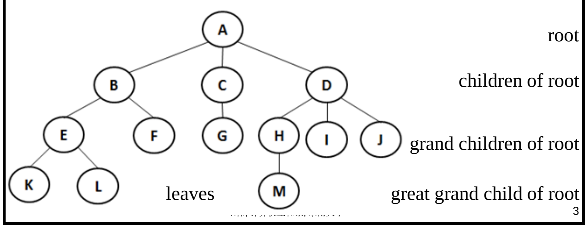
Linear Lists and Trees

- **Linear lists are useful for serially ordered data**
  - **(e_0, e_1, e_2, …, e_{n-1})**
    - Sample : days of week, months in a year, students in this class
- **Tree structure**
  - **the data are organized in a hierarchical manner**
- **Trees are useful for hierarchically ordered data**
  - **Sample :**



Hierarchical Data and Trees

- **In the hierarchy**
  - **root : the element at the top of the hierarchy**
  - **children : elements next in the hierarchy of the root**
  - **grandchildren : elements next in the hierarchy of the root, and so on**
  - **leaves : elements that have no children**



Definition of Tree

- A tree *t* is a finite nonempty set of elements
  - One or more nodes
- One of these elements is called the root
  - A specially designated node
- The remaining elements, if any, are partitioned into trees, which are called the subtrees of *t*
  - disjoint sets $T_1,\ldots,T_n$ $n>=0$

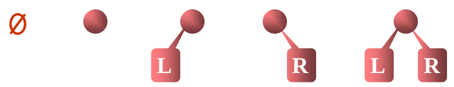
$$T = \{r, T_1, T_2, \ldots, T_n\},\ n > 0$$

- Notice : this is a recursive definition

Binary Tree

- Finite (possibly empty) collection of elements.
- A nonempty binary tree has a root element.
- The remaining elements (if any) are partitioned into two binary trees.
- These are called the left and right subtrees of the binary tree.

$$T = \begin{cases} \Phi, & n = 0 \\ \{r, T_1, T_2, \ldots, T_n\}, & n > 0 \end{cases}$$

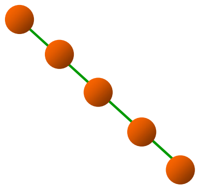


Abstract Data Type of Binary Tree

```
template <class T>
class BinaryTree
{
  public:
    BinaryTree();
    ~BinaryTree(BinaryTree<T> &bt1, T & root, BinaryTree<T> &bt2);
    bool IsEmpty();
    BinaryTree<T> leftSubtree();
    BinaryTree<T> rightSubtree();
    T RootData();
};
```

Minimum Number Of Nodes

- Minimum number of nodes in a binary tree whose height is k
- At least one node at each of first k levels



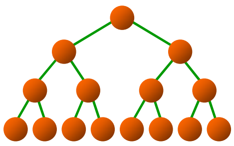
a skewed tree

Maximum Number Of Nodes

- All possible nodes at first k levels are present.

Maximum number of nodes = $1 + 2 + 4 + 8 + \ldots + 2^{k-1}$

$= 2^k - 1$



a complete tree

Properties 1 : maximum number of nodes

- The maximum number of nodes on level i ($i >= 1$) of a binary tree is 2^{i-1}

Properties 2 : maximum number of nodes

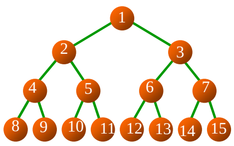
- The maximum number of nodes in a binary tree of depth k ($k >= 1$) is 2^k-1

Properties 3 : leaf nodes and degree-2 nodes

- Relation between number of leaf nodes and degree-2 nodes
  - For any nonempty binary tree, *T*, if
  - n_0 : the number of leaf nodes
  - n_2 : the number of nodes of degree 2
  - Then $n_0 = n_2 + 1$

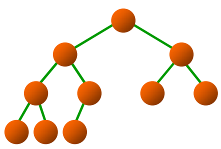
Full Binary Tree

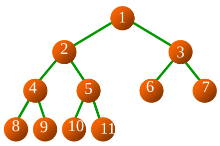
- Definition
  - A full binary tree of depth *k* is a binary tree of depth *k* having $2^k - 1$ nodes
- Numbering Nodes
  - Number the nodes 1 through $2^k - 1$
  - Number by levels from top to bottom
  - Within a level number from left to right



Complete Binary Tree

- Definition
  - A complete binary tree with *n* nodes and depth *k*
  - iff its nodes correspond to the nodes numbered from 1 to *n* in the full binary tree of depth *k* having $2^k - 1$ nodes
- Start with a full binary tree that has at least n nodes
- Number the nodes as described earlier
- The binary tree defined by the nodes numbered 1 through n is the unique n node complete binary tree





Properties 4 : Number Of Nodes & Height

- Let n be the number of nodes in a binary tree whose height is k

$$2^{k-1} - 1 < n \le 2^k - 1$$

⬇

$$k - 1 < \log_2(n+1) \le k$$

⬇

$$k = \lceil \log_2(n+1) \rceil$$

Abstract Data Type of Binary Tree

```
template <class T>
class BinaryTree {
//对象: 结点的有限集合, 二叉树是有序树
public:
    BinaryTree ();                          //构造函数
    BinaryTree ( BinTreeNode<T> *lch,
                 BinTreeNode<T> *rch,
                 T item );
    //构造函数, 以item为根, lch为左右树, rch为右子树
    //构造一棵二叉树
    int Height ();                  //求树深度或高度
    int Size ();                    //求树中结点个数
```



```
BinTreeNode<T> *Parent (BinTreeNode<T> *t);
                        //求结点 t 的双亲
BinTreeNode<T> *LeftChild (BinTreeNode<T> *t);
                        //求结点 t 的左子女
BinTreeNode<T> *RightChild (BinTreeNode<T> *t);
                        //求结点 t 的右子女

bool Insert (T item);           //在树中插入新元素

bool Remove (T item);           //在树中删除元素
bool Find (T& item);            //判断item是否在树中
bool getData (T& item);         //取得结点数据
bool IsEmpty ();                //判二叉树空否
```

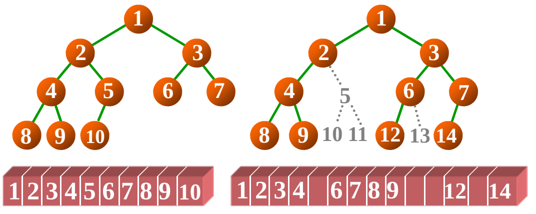
```
BinTreeNode<T> *getRoot ();      //取根

void preOrder (void (*visit) (BinTreeNode<T> *t));
    //前序遍历, visit是访问函数
void inOrder (void (*visit) (BinTreeNode<T> *t));
    //中序遍历, visit是访问函数
void postOrder (void (*visit) (BinTreeNode<T> *t));
    //后序遍历, (*visit)是访问函数
void levelOrder (void (*visit)(BinTreeNode<T> *t));
    //层次序遍历, visit是访问函数
};
```


Binary Tree Representation

- **Array representation**
- **Linked representation**

Sequential Representation

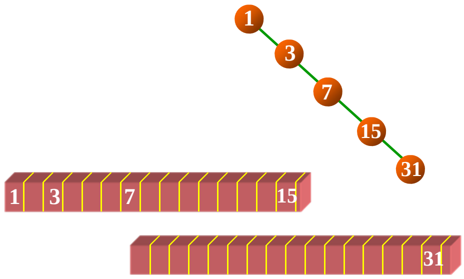
- **Number the nodes using the numbering scheme for a full binary tree**
- **The node that is numbered i is stored in array tree[i]**



Complete binary tree with 10 nodes **Binary tree with 10 nodes**

Skewed tree: skewed to the right

- An n node binary tree needs an array whose length is between n+1 and 2^n

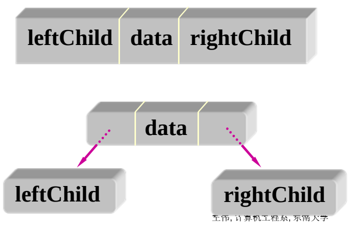


Properties : 5 Node Number

- If a complete binary tree with *n* nodes is represented sequentially, then for any node with index *i* ($2i+1>n$) have
  - if $i \neq 1$, *parent (i)* at $\lfloor i/2 \rfloor$
  - if $i = 1$, i is at the root and not parent
  - if $2i \leq n$, *leftchild(i)* at $2i$
  - if $2i > n$, i has no left child
  - if $2i+1 \leq n$, *rightChild(i)* at $2i+1$
  - if $2i+1 > n$, i has no right child

Linked Representation

- Each binary tree node is represented as an object whose data type is TreeNode
- The space required = n * (space required by one node)



Binary Linked

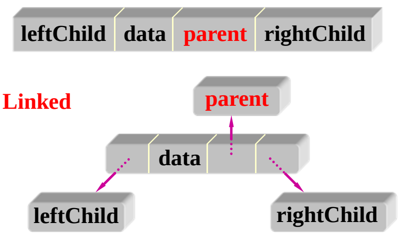
Binary Tree Node

```
<class T> class BinaryTree;          // declaration
 <class T>
 BinTreeNode
{   friend class BinaryTree<T>;      // friend class
    T data;
    BinTreeNode<T> *leftChild;
    BinTreeNode<T> *rightChild;
    BinTreeNode(){ leftChild = rightChild = NULL; }
    BinTreeNode(T d)
     { data = d; leftChild = rightChild =NULL; }
};
```


Linked Representation (con.)

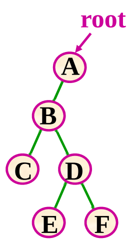
If it is necessary to be able to determine the parent of random nodes, then a fourth filed parent, may be include in TreeNode

Ternary Linked



Linked Representation using Array

- A binary tree node is represented as an object whose data type is TreeNode using a one-dimensional array representation in memory



	data	parent	leftChild	rightChild
0	A	−1	1	−1
1	B	0	2	3
2	C	1	−1	−1
3	D	1	4	5
4	E	3	−1	−1
5	F	3	−1	−1

```
template <class T>
struct BinTreeNode {              //二叉树结点类定义
    T data;                       //数据域
    BinTreeNode<T> *leftChild, *rightChild;
                                  //左子女、右子女链域
    BinTreeNode ()                //构造函数
    { leftChild = NULL;  rightChild = NULL; }

    BinTreeNode (T x, BinTreeNode<T> *l = NULL,
                 BinTreeNode<T> *r = NULL)
    { data = x;  leftChild = l;  rightChild = r; }
};
```



```
template <class T>
class BinaryTree {                          //二叉树类定义
public:
    BinaryTree () : root (NULL) { }         //构造函数
    BinaryTree (T value) : RefValue(value), root(NULL)
        { }                                 //构造函数
    BinaryTree (BinaryTree<T>& s);          //复制构造函数
    ～BinaryTree () { destroy(root); }      //析构函数

    bool IsEmpty () { return root == NULL;}
                                            //判二叉树空否
    int Height () { return Height(root); }  //求树高度
    int Size () { return Size(root); }      //求结点数
```



```
    BinTreeNode<T> *Parent (BinTreeNode <T> *t)
      { return (root == NULL || root ==  t) ?
          NULL : Parent (root, t); }   //返回双亲结点

    BinTreeNode<T> *LeftChild (BinTreeNode<T> *t)
      { return (t != NULL) ? t−>leftChild : NULL; }
                                       //返回左子女

    BinTreeNode<T> *RightChild (BinTreeNode<T> *t)
      { return (t != NULL) ? t−>rightChild : NULL; }
                                       //返回右子女

    BinTreeNode<T> *getRoot () const { return root; }
                                       //取根
```

```
void preOrder (void (*visit) (BinTreeNode<T> *t))
  { preOrder (root, visit); }        //前序遍历
void inOrder (void (*visit) (BinTreeNode<T> *t))
  { inOrder (root, visit); }         //中序遍历
void postOrder (void (*visit) (BinTreeNode<T> *t))
  { postOrder (root, visit); }       //后序遍历
void levelOrder (void (*visit)(BinTreeNode<T> *t));
                                     //层次序遍历

int Insert (const T item);                   //插入新元素
BinTreeNode<T> *Find (T item) const;         //搜索
```



```
protected:
   BinTreeNode<T> *root;          //二叉树的根指针
   T RefValue;                    //数据输入停止标志
   void CreateBinTree (istream& in,
           BinTreeNode<T> *& subTree);
                                  //从文件读入建树
   bool Insert (BinTreeNode<T> *& subTree,  T& x);
                                  //插入
   void destroy (BinTreeNode<T> *& subTree);
                                  //删除
   bool Find (BinTreeNode<T> *subTree, T& x);
                                  //查找
```



```
BinTreeNode<T> *Copy (BinTreeNode<T> *r);   //复制
int Height (BinTreeNode<T> *subTree);       //返回树高度
int Size (BinTreeNode<T> *subTree);         //返回结点数
BinTreeNode<T> *Parent (BinTreeNode<T> *
    subTree, BinTreeNode<T> *t);
                                            //返回父结点
BinTreeNode<T> *Find (BinTreeNode<T> *
    subTree, T& x) const;                   //搜寻x
```

```
void Traverse (BinTreeNode<T> *subTree, ostream& out);
                                        //前序遍历输出
void preOrder (BinTreeNode<T>& subTree,
               void (*visit) (BinTreeNode<T> *t));
                                        //前序遍历
void inOrder (BinTreeNode<T>& subTree,
     void (*visit) (BinTreeNode<T> *t));
                                        //中序遍历
void postOrder (BinTreeNode<T>& Tree,
     void (*visit) (BinTreeNode<T> *t));
                                        //后序遍历
```



```
    friend istream& operator >> (istream& in,
         BinaryTree<T>& Tree);   //重载操作：输入
    friend ostream& operator << (ostream& out,
         BinaryTree<T>& Tree);   //重载操作：输出
};
```



```
template <class T>
BinTreeNode<T> *BinaryTree<T>::Parent (BinTreeNode <T> *subTree,
        BinTreeNode <T> *t)
{
   //私有函数: 从结点 subTree 开始, 搜索结点 t 的双亲,
   //若找到, 则返回双亲结点地址; 否则, 返回NULL
    if (subTree == NULL) return NULL;
    if (subTree->leftChild == t || subTree->rightChild == t )
        return subTree;                 //找到, 返回父结点地址

    BinTreeNode <T> *p;

    if ((p = Parent (subTree->leftChild, t)) != NULL)
        return p;                       //递归在左子树中搜索
    else return Parent (subTree->rightChild, t);
                                        //递归在左子树中搜索
}
```

```
template<class T>
void BinaryTree<T>::
                    destroy (BinTreeNode<T> * subTree)
{
  //私有函数: 删除根为subTree的子树
    if (subTree != NULL) {
        destroy (subTree−>leftChild);    //删除左子树
        destroy (subTree−>rightChild);   //删除右子树
        delete subTree;                  //删除根结点
    }
}
```



```
template<class T>
istream& operator >> (istream& in, BinaryTree<T>& Tree)
{
   //重载操作: 输入并建立一棵二叉树Tree
   // in是输入流对象
   CreateBinTree (in, Tree.root);     //建立二叉树
   return in;
}
```


Data Structures

Binary Trees Traversal

Teacher : Wang Wei

1. Ellis Horowitz,etc., Fundamentals of Data Structures in C++
2. 殷人昆, 数据结构
3. 金远平, 数据结构
4. http://inside.mines.edu/~dmehta/

Binary Tree Traversal

- Many binary tree operations are done by performing a traversal of the binary tree
- **In a traversal, each element of the binary tree is visited exactly once**
- **During the visit of an element**
  - **all action with respect to this element is taken**
    - **display, make a clone, evaluate the operator, etc.**

Binary Tree Traversal Methods

- **When at a node, let:**
  - **L : moving left child**
  - **V : visiting the node**
  - **R : moving right child**
- **There are six possible combinations of traversal**

	traverse left before right		traverse right before left
preorder :	VLR	or	VRL
inorder :	LVR	or	RVL
postorder :	LRV	or	RLV

Binary Tree Traversal Methods

- **preorder**
- **inorder**
- **postorder**
- **level-order**

Program : preorder （recursive）

```
template <class T>
void BinaryTree<T>::PreOrder (BinTreeNode<T> * subTree,
                              void (*visit) (BinTreeNode<T> *t))
{
   if (subTree != NULL)
   {
        visit (subTree);                             //访问根结点
        PreOrder (subTree->leftChild, visit);        //遍历左子树
        PreOrder (subTree->rightChild, visit);       //遍历右子树
   }
}
```


Program : inorder （recursive）

```
template <class T>
void BinaryTree<T>::InOrder (
                     BinTreeNode<T> * subTree,
                     void (*visit) (BinTreeNode<T> *t) )
{
   if (subTree != NULL)
   {
        InOrder (subTree->leftChild, visit);     //遍历左子树
        visit (subTree);                         //访问根结点
        InOrder (subTree->rightChild, visit);    //遍历右子树
   }
}
```


Program : Postorder

```
template <class T>
void BinaryTree<T>::PostOrder (BinTreeNode<T> * subTree,
                               void (*visit) (BinTreeNode<T> *t )
{
   if (subTree != NULL )
   {
        PostOrder (subTree->leftChild, visit);     //遍历左子树
        PostOrder (subTree->rightChild, visit);    //遍历右子树
        visit (subTree);                           //访问根结点
   }
}
```

Level-Order Traversal

- Visits the nodes using the ordering
  - Visit the root first
  - Then visiting the nodes at each level from the leftmost node to the rightmost node
- Requires a queue

Level Order

```
Let t be the tree root
while (t != NULL)
{
    visit t and put its children on a FIFO queue;
        FIFO queue is empty, set t = NULL;
    otherwise, pop a node from the FIFO queue and call it t;
}
```


Operations : Height

```
//私有函数：利用二叉树后序遍历算法计算二叉树的高度或深度
template <class T>
int  BinaryTree<T>::Height(BinTreeNode<T> *  subTree) const
{
    if (subTree == NULL) return 0;   //空树高度为0
    else {
        int i = Height(subTree->leftChild);
        int j = Height(subTree->rightChild);
        return (i < j) ? j+1 : i+1;
}
```

Operations： Size

```
//私有函数：利用二叉树后序遍历算法计算二叉树的结点个数
template <class T>
int BinaryTree<T>::Size(BinTreeNode<T> * subTree) const
{
    if (subTree == NULL) return 0;        //空树
    else    return 1
            + Size(subTree->leftChild)
            + Size(subTree->rightChild);
}
```


Data Structures

Binary Trees Construction

Teacher : Wang Wei

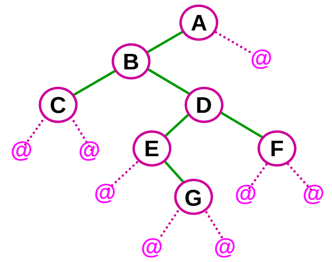
1. Ellis Horowitz,etc., Fundamentals of Data Structures in C++
2. 殷人昆, 数据结构
3. 金远平, 数据结构
4. http://inside.mines.edu/~dmehta/

Binary Tree Construction

- **Suppose**
  - **the elements in a binary tree are distinct**
- **Can you construct the binary tree from which a given traversal sequence came?**
  - **Such as**
    - **Method 1： preorder traversal sequence**
    - **Method 2: preorder and inorder sequence**

Method 1 : preorder traversal sequence

A B C @ @ D E @ G @ @ F @ @ @



Empty node : such as '@' or '-1'


```
template<class T>
void BinaryTree<T>::CreateBinTree (ifstream& in,
                                   BinTreeNode<T> *& subTree)
{
    //私有函数: 以递归方式建立二叉树
    T item;
    if ( !in.eof () ) {            //未读完, 读入并建树
        in >> item;                //读入根结点的值
        if (item != RefValue) {
            subTree = new BinTreeNode<T>(item);
                                   //建立根结点
            if (subTree == NULL)
                {cerr << "存储分配错!" << endl;  exit (1);}
```



```
            CreateBinTree (in, subTree->leftChild);
                //递归建立左子树
            CreateBinTree (in, subTree->rightChild);
                //递归建立右子树
        }
        else subTree = NULL;
                //封闭指向空子树的指针
    }
}
```

Binary Tree Construction

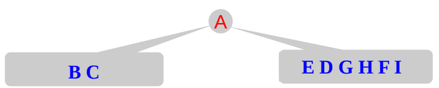
Method 2

Binary Tree Construction

- When a traversal sequence has more than one element, the binary tree is not uniquely defined
- Therefore, the tree from which the sequence was obtained cannot be reconstructed uniquely
- Can you construct the binary tree, given two traversal sequences?
- Depends on which two sequences are given, such as *preorder* and *inorder* sequences, can construct a uniquely binary tree
- Suppose : for a same binary tree
  - *preorder* sequence A B C D E F G H I
  - *inorder* sequence B C A E D G H F I

Inorder and Preorder

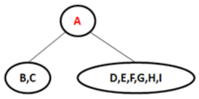
- Scan the preorder left to right using the inorder to separate left and right subtrees
- inorder : B C A E D G H F I
- preorder : A B C D E F G H I



Constructing a binary tree from its *inorder* and *preorder*

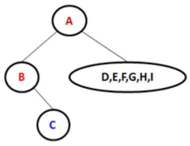
Inorder : BCAEDGHFI

Perorder : ABCDEFGHI



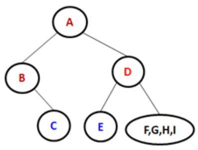
Inorder : BCAEDGHFI

Perorder : ABCDEFGHI



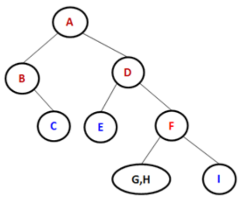
Inorder : BCAEDGHFI

Perorder : ABCDEFGHI



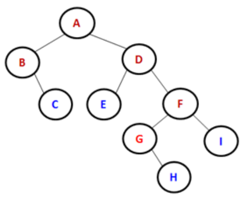
Inorder : BCAEDGHFI

Perorder : ABCDEFGHI



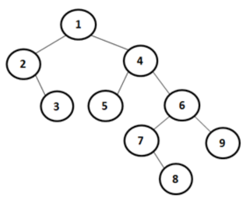
Inorder : BCAEDGHFI

Perorder : ABCDEFGHI



Inorder : 2,3,1,5,4,7,8,6,9

Perorder : 1,2,3,4,5,6,7,8,9



Data Structures

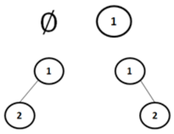
Counting Binary Trees

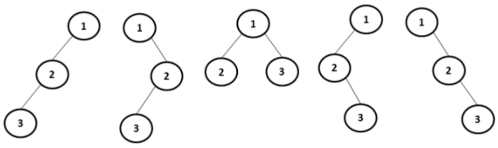
Teacher : Wang Wei

1. Ellis Horowitz,etc., Fundamentals of Data Structures in C++
2. 殷人昆, 数据结构
3. 金远平, 数据结构
4. http://inside.mines.edu/~dmehta/

Distinct binary Trees

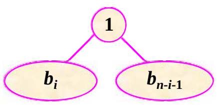
- **A binary tree *T***
  - ***n=0*, or , n=1 : only one tree**
  - ***n=2* : two distinct trees**
  - ***n=3* : five distinct trees**
  - **…**





  - ***n ? : how many* distinct trees are there with *n* nodes?**

Computer the number of distinct binary trees with n nodes



$$b_n = \sum_{i=0}^{n-1} b_i \cdot b_{n-i-1}$$

Catalan Function

$$b_n = \frac{1}{n+1} C_{2n}^{n} = \frac{1}{n+1} \frac{(2n)!}{n! \cdot n!}$$