Data Structures

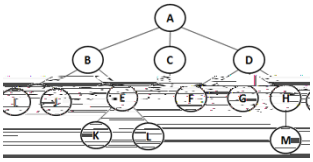
Binary Trees

Teacher : Wang Wei

1. Ellis Horowitz,etc., Fundamentals of Data Structures in C++
2. ,
3. ,
4. http://inside.mines.edu/~dmehta/

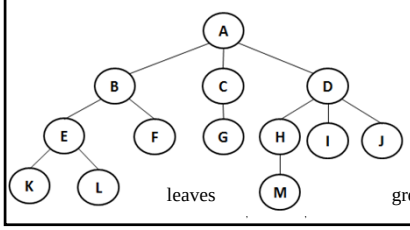
Linear Lists and Trees

- **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 $T_1,...,Tn$ n>=0

$$T \quad \{r, T_1, T_2, ..., T_n\}, \; n \; ! \; 0$$

  - Notice : this is a recursive definition

4

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.

$$T \quad \begin{cases} -Ó, & n \quad 0 \\ ®\{r, T_1, T_2, ..., T_n\}, & n \; ! \; 0 \end{cases}$$

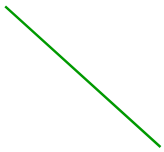
Ø

5

Abstract Data Type of Binary Tree

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

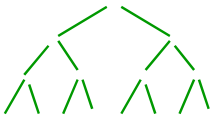
7

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

8

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

9

Properties 4 : Number Of Nodes & Height

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

Û ? Ú F Ú O " ' QF ŰŰ

F Ú O Ů: " E Ú ; Q '

' L⌈ Ů: " E Ú ;

13

Abstract Data Type of Binary Tree

```
template <class T>
class BinaryTree {
// )A' : 4 %é+´ 9K€Kö ß ¼ ù A _ 9 ¿ A
public:
    BinaryTree ();                                  // ´EP -
    BinaryTree ( BinTreeNode<T> *lch,
                 BinTreeNode<T> *rch,
                 T item );
      // ´EP -    ,    item j i   ,  lch j   # A    , rch j # € A
      // ´EP 0 % ¼ ù A
    int Height ();                                  //!r A#! Ö FP  Ö
    int Size ();                                    //!r A ]4 %é Z
```

14

```
BinTreeNode<T> *Parent (BinTreeNode<T> *t);
                                        //!r4 %é t +´ ü â
BinTreeNode<T> *LeftChild (BinTreeNode<T> *t);
                                        //!r4 %é t +´   € £
BinTreeNode<T> *RightChild (BinTreeNode<T> *t);
                                        //!r4 %é t +´ # € £

bool Insert (T item);                   // X A ]   • à s2P


bool Remove (T item);                   // X A ] PK" s2P
bool Find (T& item);                    // T Ý item _ V X A ]
bool getData (T& item);                 //   Ç4 %é   ž
bool IsEmpty ();                        // T ¼ ù A/ª V
```

15

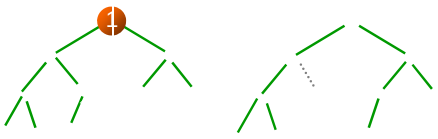
```
    BinTreeNode<T> *getRoot ();     //   i

    void preOrder (void (*visit) (BinTreeNode<T> *t));
        // } ¿E} ¶  , visit _@ïK  -
    void inOrder (void (*visit) (BinTreeNode<T> *t));
        // ] ¿E} ¶  , visit _@ïK  -
    void postOrder (void (*visit) (BinTreeNode<T> *t));
        // > ¿E} ¶ , (*visit) _@ïK  -
    void levelOrder (void (*visit)(BinTreeNode<T> *t));
        // r Q ¿E} ¶  , visit _@ïK  -
};
```

16

Binary Tree Representation

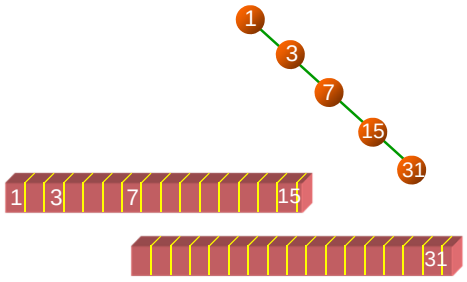
- **Array representation**
- **Linked representation**

17



Skewed tree:skewed to the right

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



19

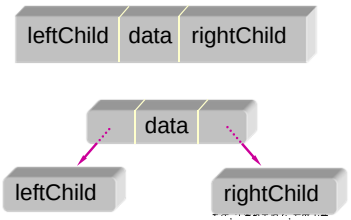
Properties : 5 Node Number

- If a complete binary tree with *n* nodes is represented sequentially, then for any node with index *i* (t E E s P) J have
  - if E M *parent (i)* at [s t
  - if E L, E is at the root and not parent
  - if t E Q , J *leftchild(i)* at t E
  - if t E P , J E has no left child
  - if t E E s Q , *rightChild(i)* at t E E s
  - if t E E s P, J E has no right child

20

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

21

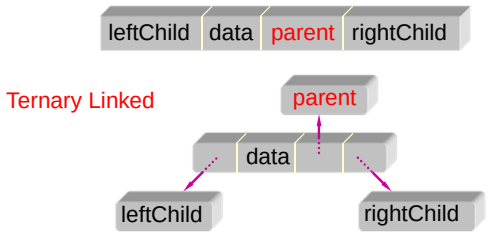
Binary Tree Node

```
<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; }
};
```

22

Linked Representation (con.)

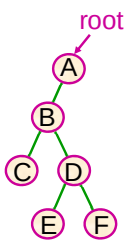
- **If it is necessary to be able to determine the parent of random nodes, then a fourth filed parent, may be include in TreeNode**



23

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

24

```
template <class T>
struct BinTreeNode {                  // ¼ ù A4 %é1« Ê y
    T data;                           //   ž
    BinTreeNode<T> *leftChild, *rightChild;
                                      //   € £ ! # € £J.
    BinTreeNode ()                    // ´EP -
    { leftChild = NULL;  rightChild = NULL; }

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

25

```
template <class T>
class BinaryTree {                            // ¼ ù A1« Ê y
public:
    BinaryTree () : root (NULL) { }           // ´EP -
    BinaryTree (T value) : RefValue(value), root(NULL)
        { }                                   // ´EP -
    BinaryTree (BinaryTree<T>& s);            // = f ´EP -
    ~BinaryTree () { destroy(root); }         // À ´ -

    bool IsEmpty () { return root == NULL;}
                                              // T ¼ ù A/ª V
    int Height () { return Height(root); }    //!r AP  Ö
    int Size () { return Size(root); }        //!r4 %é
```

26

```
    BinTreeNode<T> *Parent (BinTreeNode <T> *t)
     { return (root == NULL || root ==  t) ?
          NULL : Parent (root, t); }    //E    ü â4 %é

    BinTreeNode<T> *LeftChild (BinTreeNode<T> *t)
     { return (t != NULL)U t->leftChild : NULL; }
                                        //E      € £

    BinTreeNode<T> *RightChild (BinTreeNode<T> *t)
     { return (t != NULL)U t->rightChild : NULL; }
                                        //E    # € £

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

27

```
void preOrder (void (*visit) (BinTreeNode<T> *t))
  { preOrder (root, visit); }        // } ¿E} ¶
void inOrder (void (*visit) (BinTreeNode<T> *t))
  { inOrder (root, visit); }         // ] ¿E} ¶
void postOrder (void (*visit) (BinTreeNode<T> *t))
  { postOrder (root, visit); }       // > ¿E} ¶
void levelOrder (void (*visit)(BinTreeNode<T> *t));
                                     // r Q ¿E} ¶

int Insert (const T item);                    //   • à s2P
BinTreeNode<T> *Find (T item) const;          // L2R
```

28

```
protected:
    BinTreeNode<T> *root;              // ¼ ù A+´ i 7l¸
    T RefValue;                        //   žDÃ • Œ ’ 7
    void CreateBinTree (istream& in,
                BinTreeNode<T> *& subTree);
                                       // þ · &A+ • * A
    bool Insert (BinTreeNode<T> *& subTree,  T& x);
                                       //   •
    void destroy (BinTreeNode<T> *& subTree);
                                            // PK”
    bool Find (BinTreeNode<T> *subTree, T& x);
                                       //   ®
```

29

```
BinTreeNode<T> *Copy (BinTreeNode<T> *r);     // = f
int Height (BinTreeNode<T> *subTree);         //E    AP  Ö
int Size (BinTreeNode<T> *subTree);           //E   4 %é
BinTreeNode<T> *Parent (BinTreeNode<T> *
    subTree, BinTreeNode<T> *t);
                                              //E   'f4 %é
BinTreeNode<T> *Find (BinTreeNode<T> *
    subTree, T& x) const;                     // L + x
```

30

```
    void Traverse (BinTreeNode<T> *subTree, ostream& out);
                                        // } ¿E} ¶DÃ *
    void preOrder (BinTreeNode<T>& subTree,
                    void (*visit) (BinTreeNode<T> *t));
                                        // } ¿E} ¶
    void inOrder  (BinTreeNode<T>& subTree,
        void (*visit) (BinTreeNode<T> *t));
                                        // ] ¿E} ¶
    void postOrder (BinTreeNode<T>& Tree,
        void (*visit) (BinTreeNode<T> *t));
                                        // > ¿E} ¶
```

31

```
    friend istream& operator >> (istream& in,
          BinaryTree<T>& Tree);   //FýD- ý ŒU›DÃ •
    friend ostream& operator << (ostream& out,
          BinaryTree<T>& Tree);   //FýD- ý ŒU›DÃ *
};
```

32

```
template <class T>
BinTreeNode<T> *BinaryTree<T>::Parent (BinTreeNode <T> *subTree,
        BinTreeNode <T> *t)
{
  //.ñ 9 -     : þ4 %é subTree  0 û ,  L2R4 %é +´ ü â ,
  //8  ®` ,  IE    ü â4 %é ` p    ;  V I , E    NULL
   if (subTree == NULL) return NULL;
   if (subTree->leftChild == t || subTree->rightChild == t )
       return subTree;                  // ® ` , E   'f4 %é ` p

   BinTreeNode <T> *p;

   if ((p = Parent (subTree->leftChild, t)) != NULL)
       return p;                        //EB , X   € A ] L2R
   else return Parent (subTree->rightChild, t);
                                        //EB , X   € A ] L2R
}
```

33

```
template<class T>
void BinaryTree<T>::
                destroy (BinTreeNode<T> * subTree)
{
  //.ñ 9 -     : PK" i j   subTree+´ € A
   if (subTree != NULL) {
      destroy (subTree->leftChild);    // PK"   € A
      destroy (subTree->rightChild);   // PK" # € A
      delete subTree;                          // PK" i4 %é
   }
}
```

34

```
template<class T>
istream& operator >> (istream& in, BinaryTree<T>& Tree)
{
   //FýD- ý Œ DÃ • ¦ */û 0 % ¼ ù A   Tree
   // in _DÃ •"q )A‘
    CreateBinTree (in, Tree.root);      // */û ¼ ù A
    return in;
}
```

35

Data Structures

Binary Trees Traversal

Teacher : Wang Wei

1. Ellis Horowitz,etc., Fundamentals of Data Structures in C++
2. ,
3. ,
4. http://inside.mines.edu/~dmehta/

36

Binary Tree Traversal

- Many binary tree operations are done by performing a traversal of the binary tree
-

Program : preorder recursive

```
template <class T>
void BinaryTree<T>::PreOrder (BinTreeNode<T> * subTree,
                              void (*visit) (BinTreeNode<T> *t))
{
   if (subTree != NULL)
   {
        visit (subTree);                          //@ïK  i4 %é
        PreOrder (subTree->leftChild, visit);     //E} ¶   € A
        PreOrder (subTree->rightChild, visit);    //E} ¶ # € A
    }
}
```

40

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);     //E} ¶   € A
        visit (subTree);                         //@ïK  i4 %é
        InOrder (subTree->rightChild, visit);    //E} ¶ # € A
    }
}
```

41

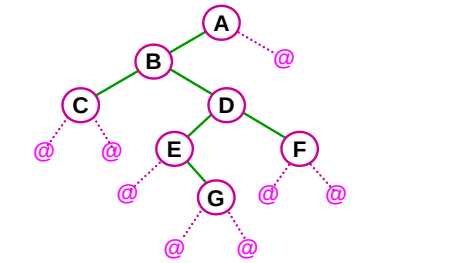
Program : Postorder

```
template <class T>
void BinaryTree<T>::PostOrder (BinTreeNode<T> * subTree,
                               void (*visit) (BinTreeNode<T> *t )
{
    if (subTree != NULL )
    {
        PostOrder (subTree->leftChild, visit);    //E} ¶   € A
        PostOrder (subTree->rightChild, visit);   //E} ¶ # € A
        visit (subTree);                          //@ïK  i4 %é
    }
}
```

42

Method 1 : preorder traversal sequence

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



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

49

```
template<class T>
void BinaryTree<T>::CreateBinTree (ifstream& in,
                                   BinTreeNode<T> *& subTree)
{
   //.ñ 9 -     :  EB , é ? */û ¼ ù A
   T item;
   if ( !in.eof () ) {              // ZA+ ¼ A+ • ¦ * A
      in >> item;                   //A+ • i4 %é+´ I
      if (item != RefValue) {
         subTree = new BinTreeNode<T>(item);
                                    // */û i4 %é
         if (subTree == NULL)
            {cerr << " ˆ Ø 6F}JI   !" << endl;  exit (1);}
```

50

```
            CreateBinTree (in, subTree->leftChild);
                  //EB , */û   € A
               CreateBinTree (in, subTree->rightChild);
                  //EB , */û # € A
         }
          else subTree = NULL;
                  // 1K  7 A/ª € A+´ 7I¸
      }
   }
```

51

Binary Tree Construction

Method 2

52

Binary Tree Construction

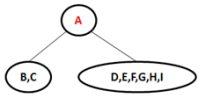
- When a traversal sequence has more than one element, the binary tree is not uniquely defined
- Therefore, the tree from which the sequence was obtained cannot be reconstructed uniquely
- Can you construct the binary tree, given two traversal sequences?
- Depends on which two sequences are given, such 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

53

Constructing a binary tree from its inorder and preorder

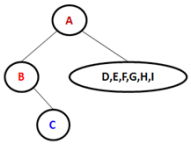
Inorder : BCAEDGHFI

Perorder : ABCDEFGHI



Inorder : BCAEDGHFI

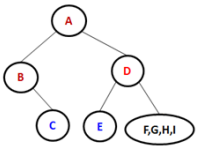
Perorder : ABCDEFGHI



55

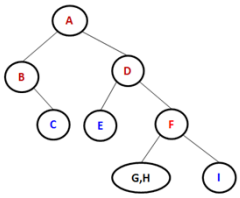
Inorder : BCAEDGHFI

Perorder : ABCDEFGHI



Inorder : BCAEDGHFI

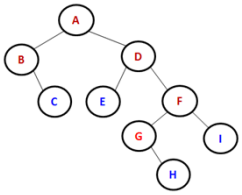
Perorder : ABCDEFGHI



56

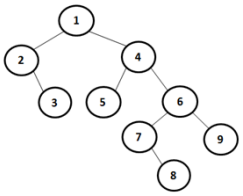
Inorder : BCAEDGHFI

Perorder : ABCDEFGHI



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

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



57

Data Structures

Counting Binary Trees

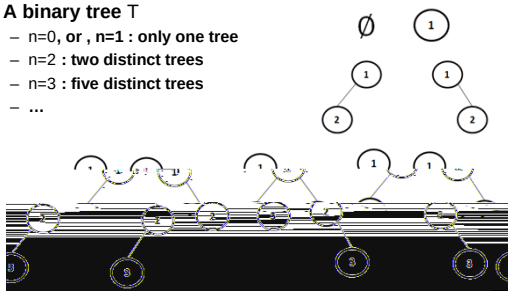
Teacher : Wang Wei

1. Ellis Horowitz,etc., Fundamentals of Data Structures in C++
2. ,
3. ,
4. http://inside.mines.edu/~dmehta/

58

Distinct binary Trees

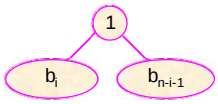
- **A binary tree** T
  - n=0**, or , n=1 : only one tree**
  - n=2 **: two distinct trees**
  - n=3 **: five distinct trees**
  - **…**



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

59

Computer the number of distinct binary trees with n nodes



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

Catalan Function

$$b_n = \frac{1}{n+1} C_n^{2n} = \frac{1}{n+1} \frac{(2n)!}{n!\,n!}$$

60