抽象数据结构的物理实现

二叉树

二叉树是每个结点最多有两个子树的树结构。通常子树被称作“左子树”（left subtree）和“右子树”（right subtree）。二叉树常被用于实现二叉查找树和二叉堆。

二叉链表表示法

用二叉链表实现二叉树，我们需要的就是把左节点和右节点分别用链表的方法形成子树，又因为有左子树和右子树，所以称为二叉链表

对树的每个节点，（从根节点开始），设置三个值，比如，data，lchild，和rchild，分别代表该节点的值和左子树节点和右子树节点

基本定义如下

```
class node{
    private:
    int data;//值
    int height;//高度
    node* left;//左子树
    node* right; //右子树
    public :
    node(){left=right=NULL;}
    node(int tmp, node *l=NULL, node *r=NULL)//带参构造函数
    {
        data = tmp;
        left = l;
        right = r;
    }
    node* le()//返回左孩子
    {
        return left;
    }
    node* ri()//返回右孩子
    {
        return right;
    }
    void setLeft(node*l)//设置左孩子
    {
        left = l;
    }
    void setRight(node*r)//设置右孩子
    {
        right = r;
    }
    void setValue(int tmp)//设置当前结点的值
    {
        data = tmp;
    }
    int getValue()//获得当前结点的值
    {
```

```
        return data;
    }
    bool isLeaf()//判断当前结点是否为叶子结点
    {
        if (left == NULL&&right == NULL)
            return true;
        else
            return false;
    }
    node* create(int a[],int b[],int postL,int postR,int inL,int inR);
    void levelorder(node *root);
    void preOrder(node*tmp);
    void inOrder(node*tmp);
    void postOrder(node*tmp);
    int nodeDepth(node*tmp);
    int nodeNodes(node*tmp);
    int nodeHeight(node*tmp);
    int nodeLeafs(node*tmp);
    bool find(node*tmp, int e);
};
```

如何构建二叉链表树

把数据依次按照顺序（或者是一定的规则）加到相应的节点下面即可

```
node* node::create(int post[],int in[],int postL,int postR,int inL,int inR){
if(postL > postR){
return NULL;
}
node* root = new node;
root->data = post[postR];
int i;
for(i = inL; i <= inR;i++ ){
if(in[i] == post[postR]){
break;
}
}
int numLeft ;
numLeft = i - inL;
root->left = create(post,in,postL,postL+numLeft-1,inL,i-1);
root->right = create(post,in,postL+numLeft,postR-1,i+1,inR);
return root;
}
```

二叉链表的左子右兄节点表示法

树节点ADT

成员变量：节点存储的数据，左子节点的指向，父节点的指向，右兄弟节点的指向

成员函数：跟二叉链表很相似，都包括构造函数，获取，设置函数，但是少了判断是否为叶子节点的函数，因为仅仅根据以上已知的东西无法判断是否为叶子节点

> 简单声明如下，分别是节点的定义 和树的定义

```
#ifndef _BINNODE_HPP_
#define _BINNODE_HPP_
#include <iostream>
```

```
using namespace std;
class BinNode{
    private:
        char data;
        int parent;
        int lc;
        int rightbro;
    public:
        BinNode(){data='/';parent=-1;lc=-1;rightbro=-1;}
        void setdata(char d){data=d;}
        void setparent(int p){parent=p;}
        void setLc(int l){lc=l;}
        void setRightbro(int r){rightbro=r;}
        char getData(){return data;}
        int getParent(){return parent;}
        int getLc(){return lc;}
        int getRightbro(){return rightbro;}
};
#endif
```

```
#ifndef _BINTREE_HPP_
#define _BINTREE_HPP_
#include "BinNode.hpp"
#include <iostream>
using namespace std;
class BinTree{
    public:
        BinNode node[1024+10];
        int depth(int );
        int count(int );
        void preorder(int ,void(*visit)(BinNode no));
        void inorder(int ,void(*visit)(BinNode no));
        void postorder(int ,void(*visit)(BinNode no));
};
#endif
```

树的实现如下

```
#include "BinTree.hpp"
void BinTree::preorder(int n,void(*visit)(BinNode no)){
    if(node[n].getData()=='/'||n==-1)return;
    visit(node[n]);
    preorder(node[n].getLc(),visit);
    preorder(node[node[n].getLc()].getRightbro(),visit);
}
void BinTree::inorder(int n,void(*visit)(BinNode no)){
    if(node[n].getData()=='/'||n==-1)return;
    inorder(node[n].getLc(),visit);
    visit(node[n]);
    inorder(node[node[n].getLc()].getRightbro(),visit);
}
void BinTree::postorder(int n,void(*visit)(BinNode no)){
    if(node[n].getData()=='/'||n==-1)return;
    postorder(node[n].getLc(),visit);
    postorder(node[node[n].getLc()].getRightbro(),visit);
    visit(node[n]);
```

```
}
int BinTree::count(int n){
    if(node[n].getData()=='/'||n==-1)return 0;
    return count(node[n].getLc())+count(node[node[n].getLc()].getRightbro())+1;
}
int BinTree::depth(int n){
    if(node[n].getData()=='/'||n==-1)return 0;
    int lh=depth(node[n].getLc());
    int rh=depth(node[node[n].getLc()].getRightbro());
    return (lh>rh?lh:rh)+1;
}
```

构建左子右兄二叉树时，我们需要注意的是如下

（1）在构造一颗二叉树时使用的是按层次遍历的顺序，使用其他的顺序都不好构造

（2）在构造函数的时候，注意可以利用二叉树的性质，比如左节点的下标一定是奇数，左子节点的下标与父节点的下标之间的关系，兄弟节点之间下标之间的关系

弄清楚这些后就很容易的按照层次遍历的顺序构造好一颗二叉树

无向带权标号图

针对一个图而言，分为有向和无向，有权和无权，针对此无向带权编号图，我们要明确的是，两个点只要相连，（比如，a-b =5 [a与b相连，且权为5]，则 b-a=5），因此我们容易实现邻接矩阵和邻接表的方法如下

邻接矩阵表示法

在上次的有向无权邻接矩阵的基础上修改，使其生成时考虑到双向的问题，和权值的问题，邻接矩阵法，ADT声明和定义如下

```
class Graphm : public Graph{
    public:
        int numVertex,numEdge;
        int **matrix;
        int *mark;
    public:
        Graphm (int numVert) {Init(numVert);};
        ~Graphm(){
            delete [] mark;
            for(int i=0;i<numVertex;i++)
                delete []matrix[i];
            delete []matrix;
        }
        void Init(int n){
            int i;
            numVertex=n;
            numEdge=0;
            mark=new int[n];
            for(i=0 ;i<numVertex;i++)
                mark[i]=0;
            matrix=(int **) new int*[numVertex];
            for(i=0;i<numVertex;i++)
                matrix[i]=new int[numVertex];
            for(i=0;i<numVertex;i++)
```

```
                for(int j=0;j<numVertex;j++)
                    matrix[i][j]=0;
        }
        int n(){return numVertex;}
        int e(){return numEdge;}
        int first(int v){
            for(int i=0;i<numVertex;i++)
                if(matrix[v][i]!=0) return i;
            return numVertex;
        }
        int next(int v,int w){
            for(int i=w+1;i<numVertex;i++)
                if(matrix[v][i]!=0) return i;
            return numVertex;
        }
        void setEdge(int v1,int v2,int wt){
            assert(wt>0);
            if(matrix[v1][v2]==0) numEdge++;
            matrix[v1][v2]=wt;
        }
        void delEdge(int v1,int v2){
            if(matrix[v1][v2]!=0) numEdge--;
            matrix[v1][v2]=0;
        }
        bool isEdge(int i,int j){
            return matrix[i][j]!=0;
        }
        int weight(int v1,int v2){
            return matrix[v1][v2];
        }
        int getMark(int v){
            return mark[v];
        }
        void setMark(int v,int val){
            mark[v]=val;
        }
};
```

我们在使用时，可以这样使用声明

```
for(int i=0;i<m;i++)
    {
        char aa,bb;
        int a1=0,b1=0,c1=0,j=0;
        cin >> aa>>bb>>c1;
        for(j=0;j<n;j++)
        {
            if(aa==name[j]) a1=j;
            if(bb==name[j]) b1=j;
        }
        a.setEdge(a1,b1,c1);
        a.setEdge(b1,a1,c1);
    }
```

(转化数组字母为数组下标)

邻接表表示法