**课程设计实验报告**

学院：国际教育学院

姓名：曹玮

班级：153609

学号：20151067

指导老师：孙暘

一：题目描述

二：题目分析

三：知识点分析

四：伪代码描述

五：程序设计

六：运行结果及结果分析

七：总结

一：题目描述

天然气经过管道网络从其生产基地输送到消耗地，在传输过程中，其性能的某一个或几个方面可能会有所衰减（例如气压）。为了保证信号衰减不超过容忍值，应在网络中的合适位置放置放大器以增加信号（例如电压）使其与源端相同。设计算法确定把信号放大器放在何处，能使所用的放大器数目最少并且保证信号衰减不超过给定的容忍值。

二：题目分析

此程序为解决此问题设计算法以确定把信号放大器放在何处。目标是所放置的信号放大器数目最少且信号衰弱不超过容忍值

三：知识点分析

为了简化问题，假设分布网络是二叉树结构，源端是树的根结点，信号从一个结点流向其孩子结点（处了根），表示一个可以用来放装置放大器的位置。图片中是一个网络示意图，边上标出的是从父结点到子节点的信号衰弱量

四：伪代码描述

1. D(i) = 0 ;

2. for (i 的每个孩子j )

2.1 如果D(j) +d(j)>容忍值，则在j处放置放大器;

2.2 否则D(i) = max{D(i)，D(j) +d(j)}

五：程序设计

#include <iostream>

using namespace std;

struct element{

int D; // 该结点的衰减量

int d; // 父结点的衰减量

bool boost; //当且仅当本处设置放大器，则boost为true

};

struct BiNode{

element data;

char name;

BiNode \*lchild,\*rchild;

};

class BiTree{

public:

BiTree();

~BiTree(){Release(root);}

void CalcD(){CalcD(root);}

void print(){print(root);}

void SetAMP(int stand){SetAMP(root,stand);}

private:

BiNode \*root;

void Release(BiNode \*root);

int CalcD(BiNode \*root);

void print(BiNode \*root);

void SetAMP(BiNode \*root,int stand);

};

void BiTree::print(BiNode \*root){

if(root!=NULL){

print(root->lchild);

print(root->rchild);

cout<<root->name<<" "<<root->data.d<<" "<<root->data.D<<endl;

}

else return ;

}

BiTree::BiTree(){

BiNode \*A=new BiNode;

BiNode \*B=new BiNode;

BiNode \*C=new BiNode;

BiNode \*D=new BiNode;

BiNode \*E=new BiNode;

BiNode \*F=new BiNode;

BiNode \*G=new BiNode;

BiNode \*H=new BiNode;

BiNode \*I=new BiNode;

BiNode \*J=new BiNode;

BiNode \*K=new BiNode;

root=A;

A->name='A';

A->lchild=B;

A->rchild=C;

A->data.d=0;

A->data.boost=0;

B->name='B';

B->lchild=D;

B->rchild=E;

B->data.d=1;

B->data.boost=0;

D->name='D';

D->lchild=H;

D->rchild=I;

D->data.d=2;

D->data.boost=0;

H->name='H';

H->lchild=NULL;

H->rchild=NULL;

H->data.d=2;

H->data.D=0;

H->data.boost=0;

I->name='I';

I->lchild=NULL;

I->rchild=NULL;

I->data.d=1;

I->data.D=0;

I->data.boost=0;

E->name='E';

E->lchild=NULL;

E->rchild=NULL;

E->data.D=0;

E->data.d=2;

E->data.boost=0;

C->name='C';

C->lchild=F;

C->rchild=G;

C->data.d=3;

C->data.boost=0;

F->name='F';

F->lchild=J;

F->rchild=NULL;

F->data.d=1;

F->data.boost=0;

G->name='G';

G->lchild=K;

G->rchild=NULL;

G->data.d=2;

G->data.boost=0;

J->name='J';

J->lchild=NULL;

J->rchild=NULL;

J->data.D=0;

J->data.d=2;

J->data.boost=0;

K->name='K';

K->lchild=NULL;

K->rchild=NULL;

K->data.D=0;

K->data.d=2;

K->data.boost=0;

}

void BiTree::Release(BiNode \*root){

if(root!=NULL){

Release(root->lchild);

Release(root->rchild);

delete root;

}

}

int BiTree::CalcD(BiNode \*root){

if(root->lchild==NULL && root->rchild==NULL) //如果root是叶子节点

return root->data.D; //返回D值

else{

int n=0,m=0,left=0,right=0;

if(root->lchild!=NULL){ //如果root有左孩子

n=CalcD(root->lchild); //计算root左子树的D值

if(root->lchild->data.boost==1) //如果被设置了放大器

n=0; //忽略左子树的D值

left=root->lchild->data.d+n; //root左侧给出的D值

}

if(root->rchild!=NULL){ //如果root有右孩子

m=CalcD(root->rchild); //计算root右孩子的D值

if(root->rchild->data.boost==1) //如果被设置了放大器

m=0; //忽略右子树的D值

right=root->rchild->data.d+m; //root右侧给出的D值

}

root->data.D=(left>right)?(left):(right); //root的D值是左侧和右侧的较大值

}

}

void BiTree::SetAMP(BiNode \*root,int stand){ //用入栈两次来省略flag的方法

BiNode \*s[200];

int top=-1;

BiNode \*visit;

s[++top]=root;

s[++top]=root;

while(s&&top>=0){

root=s[top];

top--;

visit=s[top];

if(root->rchild!=NULL && root==visit){

s[++top]=root->rchild;

s[++top]=root->rchild;

}

if(root->lchild!=NULL && root==visit){

s[++top]=root->lchild;

s[++top]=root->lchild;

}

if(root->lchild==NULL && root->rchild==NULL){

if(root->data.D+root->data.d>stand){

root->data.boost=1;

CalcD();

}

top--;

}

if(root!=visit){

if(root->data.D+root->data.d>stand){

root->data.boost=1;

CalcD();

}

}

}

CalcD(); //改进一下节点的D值

}

int main(){

BiTree test;

test.CalcD();

cout<<endl;

test.print();

cout<<endl;

test.SetAMP(3);

test.print();

return 0;

}

六：运行结果及结果分析



前一段是未设置放大器的时候

从左到右依次是节点名字，d值，D值

后一段是设置完放大器后，结果无误

七：总结

本题的难点在于计算D值并设置放大器，考虑到它的难度，可以将它分为两步进行。第一步是计算未设置放大器时D值，第二步是设置放大器并且每次设置放大器的时候重新计算D值。

设置放大器的时候需要交互的数值比较多，用非递归算法为宜。

另附思考题代码：

#include <iostream>

using namespace std;

struct element{

int D;

int d;

bool boost;

};

struct TreeNode{

element data;

char name;

TreeNode \*first,\*right;

};

class Tree{

public:

Tree();

~Tree(){Release(root);}

void print(){print(root);}

void CalcD(){CalcD(root);}

void SetAMP(int stand){SetAMP(root,stand);}

private:

TreeNode \*root;

void Release(TreeNode \*root);

int CalcD(TreeNode \*root);

void print(TreeNode \*root);

void SetAMP(TreeNode \*root,int stand);

};

void Tree::print(TreeNode \*root){

if(root!=NULL){

print(root->first);

cout<<root->name<<" "<<root->data.d<<" "<<root->data.D<<endl;

print(root->right);

}

else

return ;

}

Tree::Tree(){

TreeNode \*A=new TreeNode;

TreeNode \*B=new TreeNode;

TreeNode \*C=new TreeNode;

TreeNode \*D=new TreeNode;

TreeNode \*E=new TreeNode;

TreeNode \*F=new TreeNode;

TreeNode \*G=new TreeNode;

TreeNode \*H=new TreeNode;

TreeNode \*I=new TreeNode;

TreeNode \*J=new TreeNode;

TreeNode \*K=new TreeNode;

root=A;

A->name='A';

A->first=B;

A->right=NULL;

A->data.d=0;

A->data.boost=0;

B->name='B';

B->first=D;

B->right=C;

B->data.d=1;

B->data.boost=0;

D->name='D';

D->first=H;

D->right=E;

D->data.d=2;

D->data.boost=0;

H->name='H';

H->first=NULL;

H->right=I;

H->data.d=2;

H->data.D=0;

H->data.boost=0;

I->name='I';

I->first=NULL;

I->right=NULL;

I->data.d=1;

I->data.D=0;

I->data.boost=0;

E->name='E';

E->first=NULL;

E->right=NULL;

E->data.D=0;

E->data.d=2;

E->data.boost=0;

C->name='C';

C->first=F;

C->right=NULL;

C->data.d=3;

C->data.boost=0;

F->name='F';

F->first=J;

F->right=G;

F->data.d=1;

F->data.boost=0;

G->name='G';

G->first=K;

G->right=NULL;

G->data.d=2;

G->data.boost=0;

J->name='J';

J->first=NULL;

J->right=NULL;

J->data.D=0;

J->data.d=2;

J->data.boost=0;

K->name='K';

K->first=NULL;

K->right=NULL;

K->data.D=0;

K->data.d=2;

K->data.boost=0;

}

void Tree::Release(TreeNode \*root){

if(root!=NULL){

Release(root->first);

Release(root->right);

delete root;

}

}

int Tree::CalcD(TreeNode \*root){

if(root->first==NULL){ //如果root是叶子节点

return root->data.D;

} //返回D值

else{ //需要根据支路的数量创建变量？

int count=1; //计数孩子个数

TreeNode \*temp=root->first;

while(temp->right!=NULL){

count++;

temp=temp->right;

}

temp=root->first;

int a[count]={0}; //支路是count条，每条支路都应该返回一个D值

for(int n=0;n<count;n++){

for(int i=0;i<n;i++)

temp=temp->right;

a[n]=CalcD(temp);

if(temp->data.boost==1)

a[n]=0;

a[n]=temp->data.d+a[n];

}

int max=0;;

for(int i=0;i<count;i++){

if(a[i]>max)

max=a[i];

}

root->data.D=max;

return root->data.D;

}

}

void Tree::SetAMP(TreeNode \*root,int stand){

//中序遍历按照每个节点的顺序考虑其是否应该设置放大器。

TreeNode \*s[80];

int top=-1;

while(root!=NULL||top!=-1){

while(root!=NULL){

s[++top]=root;

root=root->first;

}

if(top!=-1){

root=s[top--];

if((root->data.d+root->data.D)>stand){

root->data.boost=1;

CalcD();

}

root=root->right;

}

}

}

int main(){

Tree A;

A.print();

cout<<endl;

A.CalcD();

A.print();

cout<<endl;

A.SetAMP(3);

A.print();

cout<<endl;

}