VC++课程设计

电子工程与光电技术学院

指导老师 : 徐新华

学号 : 921040G0637

学生 : 张悦熠

题目 : 二叉树操作

- 程序功能简介:

  一个基本的二叉树程序。实现二叉树节点的数据的插入、删除、查找和打印输出等功能。

- 课程设计要求:

  （1）设计对二叉树节点中的数据进行由大到小排序的函数

  （2）设计对二叉树上节点的数据进行有小到大排序的函数

  （3）选做：设计计算二叉树节点上数据的平均值的函数

  （4）用类来实现封装

- 课程设计说明:
  - 将要操作的数据按以下格式输入到文档中：1表示插入数据，2表示删除数据，3表示查找数据，4表示找前驱数据，5表示找后继数据，6表示最小值，7表示最大值，8表示输出二叉树，9表示输出平均值，10表示顺序输出二叉树。
  - 评定难易级别：A级

完成情况

- 全部完成
- 额外添加多项功能
  - 在输出平均值的基础上支持输出指定节点及其子树的平均值
  - 支持输出指定节点及其子树的总和
  - 支持保存程序数据到文档
  - 支持从文档中读取数据
  - 考虑程序边界，防止数据越界
  - 较为优美的界面
  - 演示模式

功能列表

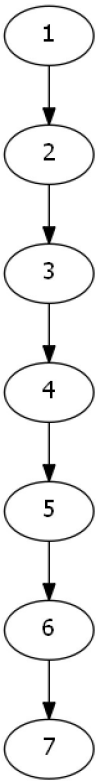
1. 插入数据	**9. 输出平均值**
2. 删除数据	**10. 按遍历顺序输出二叉树**
3. 查找数据	**11. 从小到大排序输出**
4. 找出前驱数据	**12. 从大到小排序输出**
5. 找出后继数据	**13. 某一个节点及其子树的平均值**
6. 找最小值	**14. 某一个节点及其子树的总和**
7. 找最大值	**15. 保存数据并退出**
8. 输出二叉树	**16.载入数据**

平衡树简介

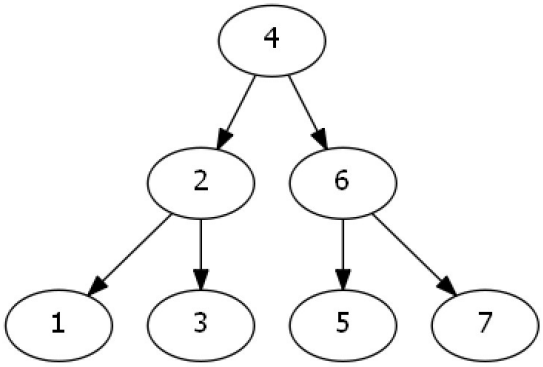
二叉树是一种重要的树形结构，由于它只有两个叉的重要性质，可用于二叉搜索树。在二叉搜索树中，对于一个节点来说，一个节点左边值的都是大于它的，右边的值都是小于它的，所以进行搜索的时候只需要与当前节点的值进行判断，算法的期望时间复杂度为$O(logN)$。

平衡树是基于二叉树的数据结构，常见的平衡树有FHQ，伸展树Splay，红黑树RBtree，Treap，替罪羊树，本课程设计使用的是FHQ平衡树。

如果只将二叉树按照小的在左边，大的在右边这样的排序方式，生成的树深度可能会非常的大，在最坏的情况下甚至能得到$O(n)$的时间复杂度，如下图。



在这种情况下，这棵树确实符合二叉搜索树的性质，但是查找速度等于$O(n)$的遍历速度，例如我们要查找7，那么我们就需要遍历7次才能找到我们想要找到的节点，我们想要让这个树看起来更加的平衡，这样才能使我们的查找速度尽可能的快，平衡后的树如下图。



此树还有其他的平衡方式，但是现在已经非常的平衡了，查找7仅仅需要3次。几乎所有的平衡树算法都是在寻找如何才能使一棵树变得更加的平衡。

FHQ 平衡树

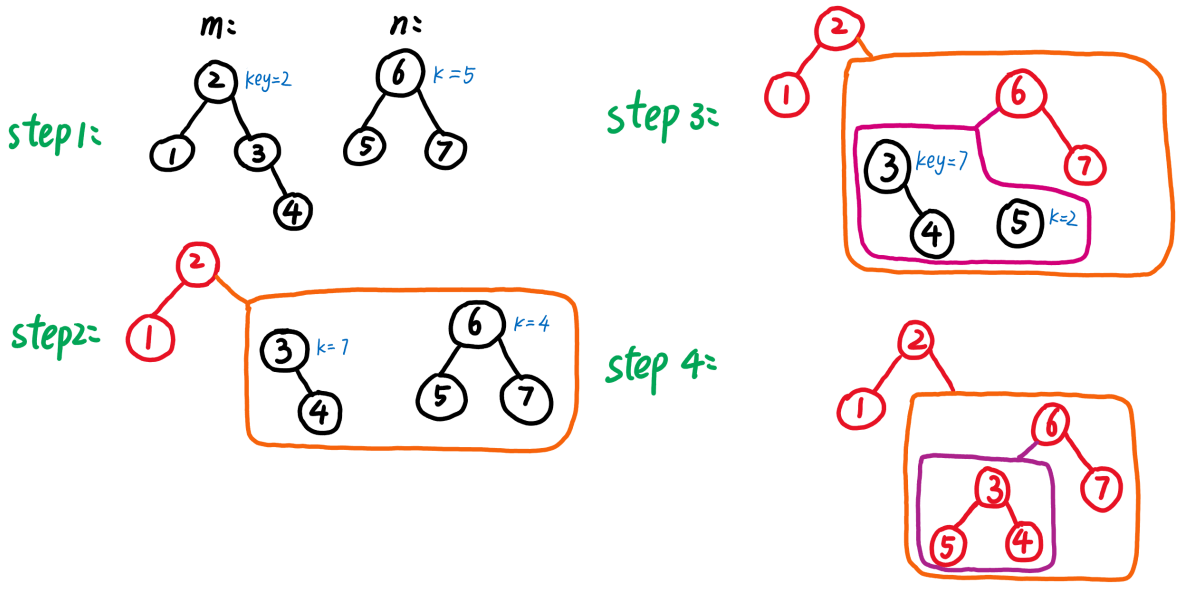
符号说明：

符号	val	size	l	r	key	root
名称	节点的值	节点的个数	左儿子	右儿子	平衡因子	根节点

为了使一棵树平衡，我们不仅仅依靠一个节点的值val来确定它在左边还是右边，我们对于每一个节点还随机生成一个平衡因子key，通过key和val共同确定节点的位置， 由于key的值是随机的，所以FHQ平衡树一般来说都是很平衡的，以下是FHQ最核心的两个操作的介绍——**merge** & **split**

merge：实现这一功能的具体做法是在合并两颗树的同时对比两棵树的根节点$root$的key的大小，假设有两个子树m,n，如果左儿子m的key更小，那么左儿子做根节点，如果右儿子n的key更小，则右儿子做根节点，当然，以上只是选择了根节点是哪一个，还需要进一步的合并才能使这棵树完整，如果是左儿子做根节点，那么继续合并左儿子m的右儿子，和右儿子n，将其合并所得到的根节点接到左儿子m的右边，同理，当右儿子被选为根节点的时候，继续合并右儿子n的左儿子和左儿子m，并将其合并所得到的结果作为右儿子n的左儿子。

这样的合并为什么能够保证这颗平衡树平衡呢，这是因为只要我们保证右儿子n的所有值大于n，且m,n都是一个满足二叉搜索性质的树，那么在上述合并的时候，左儿子m的右儿子和右儿子n都大于左儿子m的根节点，所以合并之后符合二叉搜索树的性质，同理，右儿子n的左儿子和左儿子m都小于右儿子n的根节点，所以合并之后也符合二叉搜索树的性质，以下为模拟合并操作。



split：split函数相对来说简单，但是需要传入四个参数，也就是当前节点u，左儿子l，右儿子r，分裂依据的值x，分裂完了之后，左边的子树m都小于x，右边的子树都大于等于x，此函数与$merge$配合能够完成FHQ绝大多数的功能

- 插入一个新的节点

```
root=fhq.merge(fhq.merge(l,fhq.add(x)),r);
```

- 删除一个节点

```
fhq.split(root,x,l,r);
fhq.split(l,x-1,l,p);
p=fhq.merge(tree[p].l,tree[p].r);
root=fhq.merge(fhq.merge(l,p),r);
```

这里将`tree[p].l`和`tree[p].r`合并，目的是如果有重复的数字出现的话，就会只删除一个数字

- 查询前驱

```
fhq.split(root,x-1,l,r);
tree[fhq.kth(l,tree[l].size)].val);
```

- 查询后继

```
fhq.split(root,x,l,r);
tree[fhq.kth(r,1)].val);
```

注：kth函数是查询平衡树第k大小的函数

代码调试及功能完善

在代码的运行过程中，发现了代码中存在的可优化部分，比如合理考虑代码的边界问题，如：如果没有前驱的话，应该输出不存在当前这个数前驱，并且输出当前这个平衡树的最小值，如果没有后继的话，应该输出当前这个平衡树的最大值。

尝试了添加了演示模式，在操作的过程中输出整个二叉树的形状，便于观看和演示。

添加了数据保存和读取的功能，便于第二次运行的时候调用数据，使程序能够在关掉之后再次运行。

添加了一个新的类UI，便于输出数据，整理需要输出的内容，帮助等等，合理利用windows.h库文件进行清除屏幕内容，再次进行输出，实现美观的目的。

基于FHQ强大的算法功能，还添加了功能如输出某一个节点的子树平均值和某一个节点的子树的总和等等，添加了遍历整个树的功能……

分析

时间复杂度：

- 插入：$O(n)$
- 删除：$O(n)$
- 查询：$O(n)$

空间复杂度：

- $O(n)$

总结

通过对平衡树代码的编写，使我对程序的设计有了更加深刻的知识，对程序的模块化和代码的封装有了更加强的操作能力，对数据结构有了更加强的编写能力，了解并掌握了平衡树的核心和操作，并在代码的编写过程中提高了自己的能力，让自己得到了充分的锻炼。

使用平衡树的思想几乎能够完成几乎所有书上的数据结构的题目，可以给每一个节点赋一个递增的值，使其成为链表，可以通过再在结构体里添加一个标签完成排行榜等题目，再加上所有的排序题目都可以使用此数据结构，所以平衡树是非常强大的。

核心代码

- 树的结构体

```
struct node{
    int key,l,r,val,size,sum;
}tree[maxn];
```

- 增加一个节点

```
int add(int x)
{
    tree[++k].key=rand();
    tree[k].size=1;
    tree[k].val=x;
    tree[k].sum=tree[k].val;
    return k;
}
```

- 回溯更新树节点的值

```
void update(int id)
{
    tree[id].size=tree[tree[id].l].size+tree[tree[id].r].size+1;
    tree[id].sum=tree[tree[id].l].sum+tree[tree[id].r].sum+tree[id].val;
}
```

- 将树分裂

```
void split(int u,int x,int &l,int &r)
{
    if(!u)
    {
        l=r=0;
        return ;
    }
    if(x>=tree[u].val)
    {
        l=u;
        split(tree[u].r,x,tree[u].r,r);
    }
```

```
    else
    {
        r=u;
        split(tree[u].l,x,l,tree[u].l);
    }
    update(u);
}
```

- 将两个树合并

```
int merge(int l,int r)
{
    if(!l||!r) return l+r;
    if(tree[l].key<=tree[r].key)
    {
        tree[l].r=merge(tree[l].r,r);
        update(l);
        return l;
    }
    else
    {
        tree[r].l=merge(l,tree[r].l);
        update(r);
        return r;
    }
}
```

- 寻找树中第k小的节点

```
int kth(int u,int x)
{
    if(tree[tree[u].l].size+1==x) return u;
    if(tree[tree[u].l].size>=x) return kth(tree[u].l,x);
    return kth(tree[u].r,x-tree[tree[u].l].size-1);
}
```

- 寻找树中值为x的节点

```
int find(int u,int x)
{
    if(tree[u].val==x) return u;
    if(!tree[u].l && tree[u].val>x) return 0;
    if(!tree[u].r && tree[u].val<x) return 0;
    if(tree[u].val>x) return find(tree[u].l,x);
    if(tree[u].val<x) return find(tree[u].r,x);
}
```

- 将树中的值按遍历顺序输出

```
void dfs(int u)
{
    printf("%d ",tree[u].val);
    if(tree[u].l) dfs(tree[u].l);
    if(tree[u].r) dfs(tree[u].r);
}
```

- 将树中的值按照形状输出出来

```
    void print(int u,int deep)
    {
        if(tree[u].r) print(tree[u].r,deep+1);
        for(int i=1;i<deep;i++) printf("      ");
        printf("| %d |\n",tree[u].val);
        if(tree[u].l) print(tree[u].l,deep+1);
    }
```

- 将树中的值从大到小输出

```
void sort_max_to_min(int u){
    if(tree[u].r) sort_max_to_min(tree[u].r);
    printf("%d ",tree[u].val);
    if(tree[u].l) sort_max_to_min(tree[u].l);
}
```

- 将树中的值从小到大输出

```
void sort_min_to_max(int u){
    if(tree[u].l) sort_min_to_max(tree[u].l);
    printf("%d ",tree[u].val);
    if(tree[u].r) sort_min_to_max(tree[u].r);
}
```

完整代码

```
// 项目名称 :  二叉树操作
// 指导老师 :  徐新华
// 作者     :  张悦熠
// 学号     :  9211040G0637
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<ctime>
#include<windows.h>
#define maxn 10000
using namespace std;
int opt,l,r,p,k,root;
char type;
struct node{
    int key,l,r,val,size,sum;
}tree[maxn];
class fhqtree
{
public:
    int add(int x)
    {
        tree[++k].key=rand();
        tree[k].size=1;
        tree[k].val=x;
        tree[k].sum=tree[k].val;
        return k;
```

```
}
void update(int id)
{
    tree[id].size=tree[tree[id].l].size+tree[tree[id].r].size+1;
    tree[id].sum=tree[tree[id].l].sum+tree[tree[id].r].sum+tree[id].val;
}
void split(int u,int x,int &l,int &r)
{
    if(!u)
    {
        l=r=0;
        return ;
    }
    if(x>=tree[u].val)
    {
        l=u;
        split(tree[u].r,x,tree[u].r,r);
    }
    else
    {
        r=u;
        split(tree[u].l,x,l,tree[u].l);
    }
    update(u);
}
int merge(int l,int r)
{
    if(!l||!r) return l+r;
    if(tree[l].key<=tree[r].key)
    {
        tree[l].r=merge(tree[l].r,r);
        update(l);
        return l;
    }
    else
    {
        tree[r].l=merge(l,tree[r].l);
        update(r);
        return r;
    }
}
int kth(int u,int x)
{
    if(tree[tree[u].l].size+1==x) return u;
    if(tree[tree[u].l].size>=x) return kth(tree[u].l,x);
    return kth(tree[u].r,x-tree[tree[u].l].size-1);
}
int find(int u,int x)
{
    if(tree[u].val==x) return u;
    if(!tree[u].l && tree[u].val>x) return 0;
    if(!tree[u].r && tree[u].val<x) return 0;
    if(tree[u].val>x) return find(tree[u].l,x);
    if(tree[u].val<x) return find(tree[u].r,x);
}
void dfs(int u)
{
    printf("%d ",tree[u].val);
```

```
        if(tree[u].l) dfs(tree[u].l);
        if(tree[u].r) dfs(tree[u].r);
    }
    void print(int u,int deep)
    {
        if(tree[u].r) print(tree[u].r,deep+1);
        for(int i=1;i<deep;i++) printf("      ");
        printf("| %d |\n",tree[u].val);
        if(tree[u].l) print(tree[u].l,deep+1);
    }
    void sort_min_to_max(int u){
        if(tree[u].l) sort_min_to_max(tree[u].l);
        printf("%d ",tree[u].val);
        if(tree[u].r) sort_min_to_max(tree[u].r);
    }
    void sort_max_to_min(int u){
        if(tree[u].r) sort_max_to_min(tree[u].r);
        printf("%d ",tree[u].val);
        if(tree[u].l) sort_max_to_min(tree[u].l);
    }
}fhq;
class UI
{
public:
    void menu()
    {
        printf("-----------------------------------------\n");
        printf("|项目 : 二叉树操作                       |\n");
        printf("|                                         |\n");
        printf("|指导老师 : 徐新华                       |\n");
        printf("|所使用数据结构 : fqh平衡树              |\n");
        printf("|作者 : 张悦熠    学号 : 9211040G0637   |\n");
        printf("-----------------------------------------\n");
    }
    void help()
    {
        printf("\n                    帮助:\n");
        printf("1  : 插入数据                9  : 输出平均值\n");
        printf("2  : 删除数据                10 : 按遍历顺序输出二叉树\n");
        printf("3  : 查找数据                11 : 从小到大排序输出\n");
        printf("4  : 找出前驱数据            12 : 从大到小排序输出\n");
        printf("5  : 找出后继数据            13 : 某一个节点及其子树的平均值\n");
        printf("6  : 找最小值                14 : 某一个节点及其子树的总和\n");
        printf("7  : 找最大值                15 : 保存数据并退出\n");
        printf("8  : 输出二叉树              16 : 载入数据\n");
        printf("-------------------------------------------------------\n");
    }
    void cls(){
        system("cls");
        menu();
        help();
    }
    void set(){
        printf("----请选择模式-----\n");
        printf("a : 正常模式\n");
        printf("b : 演示模式\n");
        scanf("%c",&type);
        cls();
```

```
        while(type!='a' && type!='b'){
            printf("\n请重新选择\n");
            scanf("%c",&type);
        }
    }
}UI;
signed main()
{
    UI.cls();
    srand(unsigned(time(0)));
    UI.set();
    if(type=='a'){
        while(scanf("%d",&opt)!=EOF)
        {
            if(opt==0){
                UI.cls();
                UI.help();
            }
            else if(opt==1)
            { //插入数据
                int x;
                scanf("%d",&x);
                fhq.split(root,x,l,r);
                root=fhq.merge(fhq.merge(l,fhq.add(x)),r);
                UI.cls();
                printf("上条指令结果 : 插入 %d\n",x);
            }
            else if(opt==2)
            { //删除数据
                int x;
                scanf("%d",&x);
                if(!fhq.find(root,x))
                {
                    UI.cls();
                    printf("上条指令结果 : 删除失败 , 二叉树中不存在 %d\n",x);
                    continue;
                }
                fhq.split(root,x,l,r);
                fhq.split(l,x-1,l,p);
                p=fhq.merge(tree[p].l,tree[p].r);
                root=fhq.merge(fhq.merge(l,p),r);
                UI.cls();
                printf("上条指令结果 : 删除 %d 成功\n",x);
            }
            else if(opt==3)
            {
                int x;
                scanf("%d",&x);
                UI.cls();
                if(fhq.find(root,x)) printf("上条指令结果 : %d 存在于二叉树中\n",x);
                else printf("上条指令结果 : %d 不存在于二叉树中\n",x);
            }
            else if(opt==4)
            { //前驱
                int x;
                scanf("%d",&x);
                if(x<=tree[fhq.kth(root,1)].val)
                {
```

```
                    UI.cls();
                    printf("上条指令结果 : %d 没有前驱数据",x);
                    printf(" , 当前的最小值是: %d\n",tree[fhq.kth(root,1)].val);
                    continue;
                }
                fhq.split(root,x-1,l,r);
                UI.cls();
                printf("上条指令结果 : %d 的前驱数据是
%d\n",x,tree[fhq.kth(l,tree[l].size)].val);
                root=fhq.merge(l,r);
            }
            else if(opt==5)
            { //后继
                int x;
                scanf("%d",&x);
                if(x>=tree[fhq.kth(root,tree[root].size)].val)
                {
                    UI.cls();
                    printf("上条指令结果 : %d 没有后继数据",x);
                    printf(" , 当前最大的值是:
%d\n",tree[fhq.kth(root,tree[root].size)].val);
                    continue;
                }
                fhq.split(root,x,l,r);
                UI.cls();
                printf("上条指令结果 : %d 的后继数据是
%d\n",x,tree[fhq.kth(r,1)].val);
                root=fhq.merge(l,r);
            }
            else if(opt==6)
            { //最小值
                UI.cls();
                printf("上条指令结果 : 二叉树最小值为
%d\n",tree[fhq.kth(root,1)].val);
            }
            else if(opt==7)
            {//最大值
                UI.cls();
                printf("上条指令结果 : 二叉树最大值为
%d\n",tree[fhq.kth(root,tree[root].size)].val);
            }
            else if(opt==8)
            {//输出二叉树
                UI.cls();
                printf("上条指令结果 : 二叉树的形状为\n");
                fhq.print(root,1);
                printf("\n");
            }
            else if(opt==9)
            {//输出二叉树的平均值
                UI.cls();
                printf("上条指令结果 : 二叉树平均值为
%.3lf\n",1.0*tree[root].sum/tree[root].size);
            }
            else if(opt==10)
            {//按照遍历顺序输出二叉树
                UI.cls();
                printf("上条指令结果 : 遍历顺序是\n");
```

```
                fhq.dfs(root);
                printf("\n");
            }
            else if(opt==11){//从小到大输出数据
                UI.cls();
                printf("上条指令结果 : 从小到大输出数据为\n");
                fhq.sort_min_to_max(root);
                printf("\n");
            }
            else if(opt==12){//从大到小输出数据
                UI.cls();
                printf("上条指令结果 : 从大到小输出数据为\n");
                fhq.sort_max_to_min(root);
                printf("\n");
            }
            else if(opt==13){
                int x;
                scanf("%d",&x);
                int treeid=fhq.find(root,x);
                UI.cls();
                if(treeid==0) printf("上条指令结果 : 节点 %d 不存在\n",x);
                else printf("上条指令结果 : 节点 %d 平均值为
%.2lf\n",x,1.0*tree[treeid].sum/tree[treeid].size);
            }
            else if(opt==14){
                UI.cls();
                int x;
                scanf("%d",&x);
                int treeid=fhq.find(root,x);
                if(treeid==0) printf("上条指令结果 : 节点 %d 不存在\n",x);
                else printf("上条指令结果 : 节点 %d 总和为
%d\n",x,tree[treeid].sum);
            }
            else if(opt==15){
                UI.cls();
                printf("上条指令结果 : 数据保存成功 数据文件名为\"data.out\"\n");
                freopen("data.out","w",stdout);
                printf("%d\n",k);
                for(int i=1;i<=k;i++){
                    printf("%d %d %d %d %d
%d\n",tree[i].key,tree[i].l,tree[i].r,tree[i].size,tree[i].sum,tree[i].val);
                }
                printf("%d\n",root);
                freopen("CON","w",stdout);
                exit(0);
            }
            else if(opt==16){
                UI.cls();
                printf("上条指令结果 : 数据载入成功 \n");
                freopen("data.out","r",stdin);
                scanf("%d",&k);
                memset(tree,0,sizeof tree);
                for(int i=1;i<=k;i++){
                    scanf("%d %d %d %d %d
%d",&tree[i].key,&tree[i].l,&tree[i].r,&tree[i].size,&tree[i].sum,&tree[i].val);
                }
                scanf("%d",&root);
                freopen("CON","r",stdin);
```

```
            }
            else {
                UI.cls();
                printf("上条指令结果 : input error\n");
            }
        }
    }

//-------------------------------------------------------------------------------------
-----------------------------------------------

    else if(type=='b'){
        while(scanf("%d",&opt)!=EOF)
        {
            if(opt==0){
                UI.cls();
                UI.help();
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
            }
            else if(opt==1)
            { //插入数据
                int x;
                scanf("%d",&x);
                fhq.split(root,x,l,r);
                root=fhq.merge(fhq.merge(l,fhq.add(x)),r);
                UI.cls();
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
                printf("上条指令结果 : 插入 %d\n",x);
            }
            else if(opt==2)
            { //删除数据
                int x;
                scanf("%d",&x);
                if(!fhq.find(root,x))
                {
                    UI.cls();
                    printf("-------------------------\n");
                    fhq.print(root,1);
                    printf("-------------------------\n");
                    printf("上条指令结果 : 删除失败 , 二叉树中不存在 %d\n",x);
                    continue;
                }
                fhq.split(root,x,l,r);
                fhq.split(l,x-1,l,p);
                p=fhq.merge(tree[p].l,tree[p].r);
                root=fhq.merge(fhq.merge(l,p),r);
                UI.cls();
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
                printf("上条指令结果 : 删除 %d 成功\n",x);
            }
            else if(opt==3)
            {
```

```
                int x;
                scanf("%d",&x);
                UI.cls();
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
                if(fhq.find(root,x)) printf("上条指令结果 : %d 存在于二叉树中\n",x);
                else printf("上条指令结果 : %d 不存在于二叉树中\n",x);
            }
            else if(opt==4)
            { //前驱
                int x;
                scanf("%d",&x);
                if(x<=tree[fhq.kth(root,1)].val)
                {
                    UI.cls();
                    printf("-------------------------\n");
                    fhq.print(root,1);
                    printf("-------------------------\n");
                    printf("上条指令结果 : %d 没有前驱数据",x);
                    printf(" , 当前的最小值是: %d\n",tree[fhq.kth(root,1)].val);
                    continue;
                }
                fhq.split(root,x-1,l,r);
                UI.cls();
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
                printf("上条指令结果 : %d 的前驱数据是
%d\n",x,tree[fhq.kth(l,tree[l].size)].val);
                root=fhq.merge(l,r);
            }
            else if(opt==5)
            { //后继
                int x;
                scanf("%d",&x);
                if(x>=tree[fhq.kth(root,tree[root].size)].val)
                {
                    UI.cls();
                    printf("-------------------------\n");
                    fhq.print(root,1);
                    printf("-------------------------\n");
                    printf("上条指令结果 : %d 没有后继数据",x);
                    printf(" , 当前最大的值是:
%d\n",tree[fhq.kth(root,tree[root].size)].val);
                    continue;
                }
                fhq.split(root,x,l,r);
                UI.cls();
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
                printf("上条指令结果 : %d 的后继数据是
%d\n",x,tree[fhq.kth(r,1)].val);
                root=fhq.merge(l,r);
            }
            else if(opt==6)
            { //最小值
```

```
                UI.cls();
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
                printf("上条指令结果 : 二叉树最小值为
%d\n",tree[fhq.kth(root,1)].val);
            }
            else if(opt==7)
            {//最大值
                UI.cls();
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
                printf("上条指令结果 : 二叉树最大值为
%d\n",tree[fhq.kth(root,tree[root].size)].val);
            }
            else if(opt==8)
            {//输出二叉树
                UI.cls();
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
                printf("上条指令结果 : 输出二叉树的形状\n");
                printf("\n");
            }
            else if(opt==9)
            {//输出二叉树的平均值
                UI.cls();
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
                printf("上条指令结果 : 二叉树平均值为
%.3lf\n",1.0*tree[root].sum/tree[root].size);
            }
            else if(opt==10)
            {//按照遍历顺序输出二叉树
                UI.cls();
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
                printf("上条指令结果 : 遍历顺序是\n");
                fhq.dfs(root);
                printf("\n");
            }
            else if(opt==11){//从小到大输出数据
                UI.cls();
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
                printf("上条指令结果 : 从小到大输出数据为\n");
                fhq.sort_min_to_max(root);
                printf("\n");
            }
            else if(opt==12){//从大到小输出数据
                UI.cls();
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
```

```
                printf("上条指令结果 : 从大到小输出数据为\n");
                fhq.sort_max_to_min(root);
                printf("\n");
            }
            else if(opt==13){
                int x;
                scanf("%d",&x);
                int treeid=fhq.find(root,x);
                UI.cls();
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
                if(treeid==0) printf("上条指令结果 : 节点 %d 不存在\n",x);
                else printf("上条指令结果 : 节点 %d 平均值为
%.2lf\n",x,1.0*tree[treeid].sum/tree[treeid].size);
            }
            else if(opt==14){
                UI.cls();
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
                int x;
                scanf("%d",&x);
                int treeid=fhq.find(root,x);
                if(treeid==0) printf("上条指令结果 : 节点 %d 不存在\n",x);
                else printf("上条指令结果 : 节点 %d 总和为
%d\n",x,tree[treeid].sum);
            }
            else if(opt==15){
                UI.cls();
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
                printf("上条指令结果 : 数据保存成功 数据文件名为\"data.out\"\n");
                freopen("data.out","w",stdout);
                printf("%d\n",k);
                for(int i=1;i<=k;i++){
                    printf("%d %d %d %d %d
%d",tree[i].key,tree[i].l,tree[i].r,tree[i].size,tree[i].sum,tree[i].val);
                }
                printf("%d\n",root);
                freopen("CON","w",stdout);
                exit(0);
            }
            else if(opt==16){
                UI.cls();
                freopen("data.out","r",stdin);
                scanf("%d",&k);
                memset(tree,0,sizeof tree);
                for(int i=1;i<=k;i++){
                    scanf("%d %d %d %d %d
%d",&tree[i].key,&tree[i].l,&tree[i].r,&tree[i].size,&tree[i].sum,&tree[i].val);
                }
                scanf("%d",&root);
                freopen("CON","r",stdin);
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
```

```
                printf("上条指令结果 : 数据载入成功 \n");
            }
            else{
                UI.cls();
                printf("-------------------------\n");
                fhq.print(root,1);
                printf("-------------------------\n");
                printf("上条指令结果 : input error\n");
            }
        }
    }
    return 0;
}
```