#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

#define M (2)

#define ROOT(\_x\_) (\_x\_->c[1]) //树根

#define DATA(\_x\_) (\_x\_->c[1]) //叶子节点根

typedef struct BPlusNode

{

int k[2\*M];//关键字储存

struct BPlusNode \*c[2\*M+1]; //索引

int leaf;//节点类型,1代表叶子节点

int n;//关键字实际数目

struct BPlusNode \*Next;//叶子节点需要连接到相邻的下一个叶子节点

} BPlusNode;

BPlusNode \*allocate\_node() //建立新节点并初始化

{

BPlusNode \*x = (BPlusNode \*)malloc(sizeof(BPlusNode));

x->leaf = 1;

x->n = 0;

x->Next = NULL;

int i = 0;

for (i = 1; i < 2\*M; ++i)

{

x->k[i] = 0;

x->c[i] = NULL;

}

x->c[2\*M] = NULL;

return x;

}

void create(BPlusNode \*T, BPlusNode \*D)//建树，即建立一个新节点作为根节点

{

BPlusNode \*x = allocate\_node();

ROOT(T) = x;

DATA(D) = x;

}

void split\_node(BPlusNode \*x, int pos, BPlusNode \*y)//分割节点，当某一节点关键字多于m时，需要拆分节点，x代表当前节点的父亲节点，pos代表位置，y代表当前节点

{

BPlusNode \*z = allocate\_node();//新建一个新节点z

z->leaf = y->leaf;//z与y一样

z->n =M-1;

int i = 0;

for (i = 1; i < M; ++i)

{

z->k[i] = y->k[i+M];

}//复制z的关键字

if (!y->leaf)//y不是叶子节点，复制索引值

{

for (i = 1; i <= M; ++i)

{

z->c[i] = y->c[i+M];

}

}

y->n = M-1;//更新Y的关键字值

if (y->leaf)

{

y->n += 1;

}

for (i = x->n+1; i > pos; --i)

{

x->c[i+1] = x->c[i];

}

x->c[pos+1] = z;

for (i = x->n; i >= pos; --i)

{

x->k[i+1] = x->k[i];

}

x->k[pos] = y->k[M];

x->n += 1;

if (y->leaf)//更新Next，z代替Y

{

z->Next = y->Next;

y->Next = z;

}

}

void insert\_nonfull(BPlusNode \*x, int key)//插入新值时，关键字未满时直接插入

{

int i = x->n;

if (x->leaf)//x是叶子

{

while (i > 0 && key < x->k[i])

{

x->k[i+1] = x->k[i];

--i;

}

i += 1;

x->k[i] = key;

x->n += 1;

}

else//不是叶子，要一直找对应节点，往下递归

{

while (i > 0 && key < x->k[i])

{

--i;

}

i += 1;

if (2\*M-1 == x->c[i]->n)

{

split\_node(x, i, x->c[i]);

if (key > x->k[i])

{

i += 1;

}

}

insert\_nonfull(x->c[i], key);//递归向下操作

}

}

void insert\_node(BPlusNode \*T, int key)//插入节点

{

BPlusNode \*r = ROOT(T);

if (2\*M-1 == r->n)//如果插入之后要溢出，则需要拆分

{

BPlusNode \*x = allocate\_node();

ROOT(T) = x;

x->c[1] = r;

x->leaf = 0;

x->n = 0;

split\_node(x, 1, r);

insert\_nonfull(x, key);

}

else//不满直接插入

{

insert\_nonfull(r, key);

}

}

void merge\_node(BPlusNode \*x, int pos, BPlusNode \*y, BPlusNode \*z) //两个节点关键字数目都小于m,则需要合并，与拆分类似，X父节点，y,z为要合并的节点，与上面类似

{

int i = 0;

if (y->leaf)//y是叶子节点

{

for (i = 1; i <= z->n; ++i)

{

y->k[i+M-1] = z->k[i];//y合并z节点

}

y->n = 2\*M-2;//更新y的实际节点值

}

else//

{

y->k[M] = x->k[pos];

for (i = 1; i <= z->n; ++i)

{

y->k[i+M] = z->k[i];

}

for (i = 1; i <= z->n+1; ++i)

{

y->c[i+M] = z->c[i];

}

y->n = 2\*M-1;

}

for (i = pos; i < x->n; ++i)

{

x->k[i] = x->k[i+1];//更新父节点的关键字值，少了一个节点

}

for (i = pos+1; i <= x->n; ++i)//更新父节点的索引值，少了一个节点

{

x->c[i] = x->c[i+1];

}

x->n -= 1;//更新父节点的实际关键字值

if (y->leaf)//更新Next值

{

y->Next = z->Next;

}

free(z);

}

int binsearch(BPlusNode \*x, int key)//二分查找，加快线性查找速度

{

int left = 0;

int right = x->n+1;

while (left+1 != right)

{

int mid = left + (right-left)/2;

if (x->k[mid] < key)

{

left = mid;

}

else

{

right = mid;

}

}

if (right <= x->n && x->k[right] == key)

{

return right;

}

return 0;

}

void shift\_to\_left(BPlusNode \*x, int pos, BPlusNode \*y, BPlusNode \*z) //左边节点分到右边

{

if (!z->leaf)//不是叶子节点,更新关键字

{

y->k[y->n+1] = x->k[pos];//复制节点

}

else

{

y->k[y->n+1] = z->k[1];

}

x->k[pos] = z->k[1];

int i = 0;

for (i = 1; i < z->n; ++i)

{

z->k[i] = z->k[i+1];

}

if (!z->leaf)//更新索引值

{

y->c[y->n+2] = z->c[1];

for (i = 1; i <= z->n; ++i)

{

z->c[i] = z->c[i+1];

}

}

y->n += 1;

z->n -= 1;

}

void shift\_to\_right(BPlusNode \*x, int pos, BPlusNode \*y, BPlusNode \*z)//右边节点分到左边

{

int i = 0;

for (i = x->n; i > 0; --i)

{

y->k[i+1] = y->k[i];

}

if (!y->leaf)

{

y->k[1] = x->k[pos];

x->k[pos] = z->k[z->n];

}

else

{

y->k[1] = z->k[z->n];

x->k[pos] = z->k[z->n-1];

}

if (!y->leaf)

{

for (i = y->n+1; i >= 1; --i)

{

y->c[i+1] = y->c[i];

}

y->c[1] = z->c[z->n+1];

}

y->n += 1;

z->n -= 1;

}

BPlusNode \*del(BPlusNode \*x, int key)//分情况考虑删除节点

{

while (1)

{

if (x->leaf)//叶子节点。找到直接删除，找不到报错

{

int pos = binsearch(x, key);

if (pos)

{

int i = pos;

while (i < x->n)

{

x->k[i] = x->k[i+1];

++i;

}

x->n-= 1;

return x;

}

else

{

printf("error !\n");

return NULL;

}

}

else//非叶子节点,按索引找到对应节点

{

int i = x->n;

while (i > 0 && x->k[i] >= key)

{

--i;

}

i += 1;

BPlusNode \*pre = NULL;

BPlusNode \*Next = NULL;

if (x->c[i]->n >= M)

{

x = x->c[i];

}

//按情况转移节点

else if (i > 1 && x->c[i-1]->n >= M)

{

pre = x->c[i-1];

shift\_to\_right(x, i-1, x->c[i], pre);

x = x->c[i];

}

else if (i <= x->n && x->c[i+1]->n >= M)

{

Next = x->c[i+1];

shift\_to\_left(x, i, x->c[i], Next);

x = x->c[i];

}

else if (i > 1)

{

pre = x->c[i-1];

merge\_node(x, i-1, pre, x->c[i]);

x = pre;

}

else

{

Next = x->c[i+1];

merge\_node(x, i, x->c[i], Next);

x = x->c[i];

}

}

}

}

BPlusNode \*delete\_node(BPlusNode \*T, int key)//删除节点

{

BPlusNode \*x = ROOT(T);

if (x->n==1) //如果删除的是根节点

{

BPlusNode \*y = x->c[1];

BPlusNode \*z = x->c[2];

if (y && z && M-1 == y->n && M-1 == z->n)//合并节点

{

merge\_node(x, 1, y, z);

free(x);

x = y;

ROOT(T) = x;

}

}

return del(x, key);

}

void print\_node(BPlusNode \*x)//打印节点值

{

int i = 0;

for (i = 1; i < x->n; ++i)

{

printf("%d--", x->k[i]);

}

printf("%d ", x->k[x->n]);

}

int search\_node(BPlusNode \*x, int key)//查找值，先找到对应的节点，再二分查找

{

BPlusNode \*y = NULL;

while (x)

{

y = x;

int i = x->n;

while (i > 0 && key <= x->k[i])

{

--i;

}

i += 1;

x = x->c[i];

}

return binsearch(y, key);

}

//void print\_tree(BPlusNode \*x)//打印树结构，输出所有节点

//{

// if (x!=NULL)

// {

// print\_node(x);

// int i = 0;

// for (i = 1; i <= x->n+1; ++i)

// {

// print\_tree(x->c[i]);

// }

// }

//}

struct out

{

BPlusNode \*x;

int dep;

}qe[10000];

void print\_tree(BPlusNode \*\_x)

{

int cnt=0,top=0;

struct out o;

o.x=\_x;

o.dep=1;

qe[cnt++]=o;

int i;

while(top!=cnt)

{

//printf("sdkfhlfksd\n");

struct out s=qe[top++];

for(i=1;i<=(s.x)->n+1;i++)

{

o.x=(s.x)->c[i];

o.dep=s.dep+1;

if(o.x!=NULL)

qe[cnt++]=o;

}

}

//int tmp=0;

for(i=0;i<cnt;i++)

{

struct out s=qe[i];

print\_node(s.x);

if(i==cnt-1||s.dep!=qe[i+1].dep)

{

printf("\n");

}

}

return 0;

}

int main()

{

BPlusNode \*T = allocate\_node();

ROOT(T) = NULL;

BPlusNode \*D = allocate\_node();

DATA(D) = NULL;

while(1)

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* 欢迎进入B+树演示程序，请选择相应功能: \*\n");

printf("\* 1.随机建立一颗B+树: \*\n");

printf("\* 2.在B+树中查找一个数: \*\n");

printf("\* 3.在B+树中插入一个数: \*\n");

printf("\* 4.在B+树中删除一个数: \*\n");

printf("\* 5.显示整个B+树: \*\n");

printf("\* 0.退出程序: \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

printf("您的选择是：");

char button;

button=getch();

putchar(button);

printf("\n");

if(button=='1')

{

create(T,D);

}

if(button=='2')

{

int key;

printf("请输入您要查找的数key:");

scanf("%d",&key);

int pos = search\_node(ROOT(T), key);

if (pos)

{

printf("found!\n");

}

else

printf("Not found!\n");

}

if(button=='3')

{

int key;

printf("请输入您要插入的数key:");

scanf("%d",&key);

insert\_node(T, key);

}

if(button=='4')

{

int key;

printf("请输入您要查删除的数key:");

scanf("%d",&key);

BPlusNode \*x = delete\_node(T, key);

if (x)

{

if (0 == x->n)

{

free(x);

ROOT(T) = NULL;

DATA(D) = NULL;

}

else

{

print\_node(x);

}

}

}

if(button=='5')

{

printf("以下为B+树结构：\n");

print\_tree(ROOT(T));

}

if(button=='0')

{

printf("程序结束");

break;

}

}

return 0;

}