**Bài 8: Cấu trúc cây đa phân (B cây)**

#include <stdio.h>

#include <conio.h>

#define M 5

struct node{

int n; /\* So cay con cua nut \*/

int keys[M-1]; /\*mang chua khoa cua nut\*/

struct node \*p[M]; /\* con tro tro den cac cay con \*/

}\*root=NULL;

enum KeyStatus { Duplicate,SearchFailure,Success,InsertIt,LessKeys };

void insert(int key);

void display(struct node \*root,int);

void DelNode(int x);

void search(int x);

enum KeyStatus ins(struct node \*r, int x, int\* y, struct node\*\* u);

int searchPos(int x,int \*key\_arr, int n);

enum KeyStatus del(struct node \*r, int x);

int main()

{

int key;

int choice;

printf("TAO B CAY BAC %d:\n",M);

while(choice!=5)

{

printf("1.Chen nut\n");

printf("2.Xoa nut\n");

printf("3.Tim khoa\n");

printf("4.Duyet cay\n");

printf("5.Thoat\n");

printf("Chon chuc nang:");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Nhap vao khoa:");

scanf("%d",&key);

insert(key);

break;

case 2:

printf("Nhap vao khoa:");

scanf("%d",&key);

DelNode(key);

break;

case 3:

printf("Nhap vao khoa:");

scanf("%d",&key);

search(key);

break;

case 4:

printf("B cay da nhap:\n");

display(root,0);

break;

case 5:

break;

default:

printf("Nhap sai chuc nang!\n");

break;

}

}

}

void insert(int key)

{

struct node \*newnode;

int upKey;

enum KeyStatus value;

value = ins(root, key, &upKey, &newnode);

if (value == Duplicate)

printf("Khoa da co trong cay roi!\n");

if (value == InsertIt)

{

struct node \*uproot = root;

root=new(node);

root->n = 1;

root->keys[0] = upKey;

root->p[0] = uproot;

root->p[1] = newnode;

}/\*End of if \*/

}/\*End of insert()\*/

enum KeyStatus ins(struct node \*ptr, int key, int \*upKey,struct node

\*\*newnode)

{

struct node \*newPtr, \*lastPtr;

int pos, i, n,splitPos;

int newKey, lastKey;

enum KeyStatus value;

if (ptr == NULL)

{

\*newnode = NULL;

\*upKey = key;

return InsertIt;

}

n = ptr->n;

pos = searchPos(key, ptr->keys, n);

if (pos < n && key == ptr->keys[pos])

return Duplicate;

value = ins(ptr->p[pos], key, &newKey, &newPtr);

if (value != InsertIt)

return value;

/\*So khoa nho hon M-1 \*/

if (n < M - 1)

{

pos = searchPos(newKey, ptr->keys, n);

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

{

ptr->keys[i] = ptr->keys[i-1];

ptr->p[i+1] = ptr->p[i];

}

ptr->keys[pos] = newKey;

ptr->p[pos+1] = newPtr;

++ptr->n;

return Success;

}/\*End of if \*/

if (pos == M - 1)

{

lastKey = newKey;

lastPtr = newPtr;

}

else

{

lastKey = ptr->keys[M-2];

lastPtr = ptr->p[M-1];

for (i=M-2; i>pos; i--)

{

ptr->keys[i] = ptr->keys[i-1];

ptr->p[i+1] = ptr->p[i];

}

ptr->keys[pos] = newKey;

ptr->p[pos+1] = newPtr;

}

splitPos = (M - 1)/2;

(\*upKey) = ptr->keys[splitPos];

(\*newnode)=new(node);

ptr->n = splitPos;

(\*newnode)->n = M-1-splitPos;

for (i=0; i < (\*newnode)->n; i++)

{

(\*newnode)->p[i] = ptr->p[i + splitPos + 1];

if(i < (\*newnode)->n - 1)

(\*newnode)->keys[i] = ptr->keys[i + splitPos + 1];

else

(\*newnode)->keys[i] = lastKey;

}

(\*newnode)->p[(\*newnode)->n] = lastPtr;

return InsertIt;

}/\*End of ins()\*/

void display(struct node \*ptr, int blanks)

{

if (ptr)

{

int i;

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

printf(" ");

for (i=0; i < ptr->n; i++)

printf("%d ",ptr->keys[i]);

printf("\n");

for (i=0; i <= ptr->n; i++)

display(ptr->p[i], blanks+10);

}/\*End of if\*/

}/\*End of display()\*/

void search(int key)

{

int pos, i, n;

struct node \*ptr = root;

printf("Search path:\n");

while (ptr)

{

n = ptr->n;

for (i=0; i < ptr->n; i++)

printf(" %d",ptr->keys[i]);

printf("\n");

pos = searchPos(key, ptr->keys, n);

if (pos < n && key == ptr->keys[pos])

{

printf("Khoa %d tim thay o vi tri %d cua nut\n",key,i);

return;

}

ptr = ptr->p[pos];

}

printf("Khoa %d khong ton tai trong cay\n",key);

}/\*End of search()\*/

int searchPos(int key, int \*key\_arr, int n)

{

int pos=0;

while (pos < n && key > key\_arr[pos])

pos++;

return pos;

}/\*End of searchPos()\*/

void DelNode(int key)

{

struct node \*uproot;

enum KeyStatus value;

value = del(root,key);

switch (value)

{

case SearchFailure:

printf("Khoa %d khong ton tai trong cay\n",key);

break;

case LessKeys:

uproot = root;

root = root->p[0];

delete(uproot);

break;

}/\*End of switch\*/

}/\*End of delnode()\*/

enum KeyStatus del(struct node \*ptr, int key)

{

int pos, i, pivot, n ,min;

int \*key\_arr;

enum KeyStatus value;

struct node \*\*p,\*lptr,\*rptr;

if (ptr == NULL)

return SearchFailure;

/\*Gan gia tri cua nut\*/

n=ptr->n;

key\_arr = ptr->keys;

p = ptr->p;

min = (M - 1)/2;/\*Kiem tra so phan tu toi thieu cua nut\*/

pos = searchPos(key, key\_arr, n);

if (p[0] == NULL)

{

if (pos == n || key < key\_arr[pos])

return SearchFailure;

for (i=pos+1; i < n; i++)

{

key\_arr[i-1] = key\_arr[i];

p[i] = p[i+1];

}

return --ptr->n >= (ptr==root ? 1 : min) ? Success : LessKeys;

}/\*End of if \*/

if (pos < n && key == key\_arr[pos])

{

struct node \*qp = p[pos], \*qp1;

int nkey;

while(1)

{

nkey = qp->n;

qp1 = qp->p[nkey];

if (qp1 == NULL)

break;

qp = qp1;

}/\*End of while\*/

key\_arr[pos] = qp->keys[nkey-1];

qp->keys[nkey - 1] = key;

}/\*End of if \*/

value = del(p[pos], key);

if (value != LessKeys)

return value;

if (pos > 0 && p[pos-1]->n > min)

{

pivot = pos - 1;

lptr = p[pivot];

rptr = p[pos];

/\*Gan gia tri cho nut ben phai\*/

rptr->p[rptr->n + 1] = rptr->p[rptr->n];

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

{

rptr->keys[i] = rptr->keys[i-1];

rptr->p[i] = rptr->p[i-1];

}

rptr->n++;

rptr->keys[0] = key\_arr[pivot];

rptr->p[0] = lptr->p[lptr->n];

key\_arr[pivot] = lptr->keys[--lptr->n];

return Success;

}/\*End of if \*/

if (pos > min)

{

pivot = pos;

lptr = p[pivot];

rptr = p[pivot+1];

/\*Gan gia tri cho nut ben trai\*/

lptr->keys[lptr->n] = key\_arr[pivot];

lptr->p[lptr->n + 1] = rptr->p[0];

key\_arr[pivot] = rptr->keys[0];

lptr->n++;

rptr->n--;

for (i=0; i < rptr->n; i++)

{

rptr->keys[i] = rptr->keys[i+1];

rptr->p[i] = rptr->p[i+1];

}/\*End of for\*/

rptr->p[rptr->n] = rptr->p[rptr->n + 1];

return Success;

}/\*End of if \*/

if(pos == n)

pivot = pos-1;

else

pivot = pos;

lptr = p[pivot];

rptr = p[pivot+1];

/\*Tron nut ben phai voi nut ben trai\*/

lptr->keys[lptr->n] = key\_arr[pivot];

lptr->p[lptr->n + 1] = rptr->p[0];

for (i=0; i < rptr->n; i++)

{

lptr->keys[lptr->n + 1 + i] = rptr->keys[i];

lptr->p[lptr->n + 2 + i] = rptr->p[i+1];

}

lptr->n = lptr->n + rptr->n +1;

delete(rptr); /\*Xoa nut ben phai\*/

for (i=pos+1; i < n; i++)

{

key\_arr[i-1] = key\_arr[i];

p[i] = p[i+1];

}

return --ptr->n >= (ptr == root ? 1 : min) ? Success : LessKeys;

}/\*End of del()\*/