Name Tejal A. Chavan

Roll no. 70

Batch E

Class Se Comp

#include <stdio.h>

#include <stdlib.h>

#define T 3

typedef struct node {

int n; // keys

int keys[T-1]; // key array

struct node \*p[T]; //children

}vertex, \*vertexp;

vertexp root = NULL; //null at first

//Methods

int position(int key, int \*k\_array, int n);

int insertHelp(vertexp ptr, int key, int \*pKey,vertexp \*newnode);

void insert(int key);

void inorder(vertexp ptr);

void search(int key);

void insert(int key)

{

vertexp newnode; //whenever a key is added a null child is added to its next

/\*when a node is overflowed it is split and median is propagated up i.e "pKey"

in this case it is child[1]\*/

int pKey;

int value; /\*int value;\*/ /\*\*\*\*/

value = insertHelp(root, key, &pKey, &newnode); //call by reference {multiple return values so}

/\*level increases by 1 if enter below\*/

if (value == 1) /\*if(value == 1)\*/ /\*\*\*\*/

{

vertexp uproot = root;

root=malloc(sizeof(vertex));

root->n = 1;

root->keys[0] = pKey;

root->p[0] = uproot;

root->p[1] = newnode;

}/\*if\*/

}/\*insert()\*/

int insertHelp(vertexp ptr, int key, int \*pKey, vertexp \*newnode)

{

vertexp newPtr;

vertexp lastPtr;

int pos, median;

int i, n;

int newKey, lastKey;

int value;

if (ptr == NULL)

{

\*newnode = NULL;

\*pKey = key;

return 1;

}

n = ptr->n;

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

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

if (value != 1)

return value;

/\*If node is not full\*/

if (n < T - 1)

{

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

/\*Shifting keys and children right to create space for new key\*/

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

{

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

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

}

/\*Key is inserted at space created\*/

ptr->keys[pos] = newKey;

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

++ptr->n; /\*Number of keys in increased by 1\*/

return 2;

}/\*if\*/

/\*If full \*/

//if key is to be inserted at last position

if (pos == T - 1)

{

lastKey = newKey;

lastPtr = newPtr;

}

else // full and not last

{

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

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

for (i=T-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;

}

median = (T - 1)/2;

(\*pKey) = ptr->keys[median];

(\*newnode)=malloc(sizeof(vertex));/\*Right node after split\*/ /\* ptr is now left split node\*/

ptr->n = median; /\*no of keys for Left split node\*/

(\*newnode)->n = T-1-median;/\*no of keys for Right split node\*/

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

{

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

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

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

else

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

}

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

return 1;

}/\*insertHelp()\*/

int position(int key, int \*k\_array, int n)

{

int pos=0;

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

pos++;

return pos;

}/\*position()\*/

void search(int key)

{

int pos, n;

vertexp ptr = root;

while (ptr)

{

n = ptr->n;

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

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

{

printf("Key %d found\n",key);

return;

}

ptr = ptr->p[pos];

}

printf("Key %d not found\n",key);

}/\*search()\*/

void inorder(vertexp ptr)

{

if(ptr)

{

if (ptr->n >= 1)

{

inorder(ptr->p[0]);

printf("%d ", ptr->keys[0]);

inorder(ptr->p[1]);

if (ptr->n == 2)

{

printf("%d ", ptr->keys[1]);

inorder(ptr->p[2]);

}

}

}

}/\*inorder()\*/

int main()

{

int key;

int choice;

printf("BTREE OPERATIONS=%d\n",T);

while(1)

{

printf("1.Insert\n");

printf("2.Search\n");

printf("3.Traverse\n");

printf("4.Quit\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Enter the key : ");

scanf("%d",&key);

insert(key);

printf("Traversal: ");

inorder(root);

printf("\n");

break;

case 2:

printf("Enter the key : ");

scanf("%d",&key);

search(key);

break;

case 3:

printf("Btree traversal:\n");

inorder(root); printf("\n");

break;

case 4:

exit(1);

default:

printf("Wrong choice\n");

break;

}/\*switch\*/

}/\*while\*/

return 0;

}/\*main()\*/

Output

BTREE OPERATIONS=3

1.Insert

2.Search

3.Traverse

4.Quit

Enter your choice : 1

Enter the key : 3

Traversal: 3

1.Insert

2.Search

3.Traverse

4.Quit

Enter your choice : 1

Enter the key : 2

Traversal: 2 3

1.Insert

2.Search

3.Traverse

4.Quit

Enter your choice : 1

Enter the key : 10

Traversal: 2 3 10

1.Insert

2.Search

3.Traverse

4.Quit

Enter your choice : 1

Enter the key : 6

Traversal: 2 3 6 10

1.Insert

2.Search

3.Traverse

4.Quit

Enter your choice : 1

Enter the key : 14

Traversal: 2 3 6 10 14

1.Insert

2.Search

3.Traverse

4.Quit

Enter your choice : 1

Enter the key : 4

Traversal: 2 3 4 6 10 14

1.Insert

2.Search

3.Traverse

4.Quit

Enter your choice : 1

Enter the key : 9

Traversal: 2 3 4 6 9 10 14

1.Insert

2.Search

3.Traverse

4.Quit

Enter your choice : 1

Enter the key : 5

Traversal: 2 3 4 5 6 9 10 14

1.Insert

2.Search

3.Traverse

4.Quit

Enter your choice : 3

Btree traversal:

2 3 4 5 6 9 10 14

1.Insert

2.Search

3.Traverse

4.Quit

Enter your choice : 2

Enter the key : 9

Key 9 found

1.Insert

2.Search

3.Traverse

4.Quit

Enter your choice : 2

Enter the key : 1

Key 1 not found

1.Insert

2.Search

3.Traverse

4.Quit

Enter your choice : 4