**B-Trees**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 4

#define MIN 2

struct btreeNode {

int val[MAX + 1], count;

struct btreeNode \*link[MAX + 1];

};

struct btreeNode \*root;

/\* creating new node \*/

struct btreeNode \* createNode(int val, struct btreeNode \*child) {

struct btreeNode \*newNode;

newNode = (struct btreeNode \*)malloc(sizeof(struct btreeNode));

newNode->val[1] = val;

newNode->count = 1;

newNode->link[0] = root;

newNode->link[1] = child;

return newNode;

}

/\* Places the value in appropriate position \*/

void addValToNode(int val, int pos, struct btreeNode \*node,

struct btreeNode \*child) {

int j = node->count;

while (j > pos) {

node->val[j + 1] = node->val[j];

node->link[j + 1] = node->link[j];

j--;

}

node->val[j + 1] = val;

node->link[j + 1] = child;

node->count++;

}

/\* split the node \*/

void splitNode (int val, int \*pval, int pos, struct btreeNode \*node,

struct btreeNode \*child, struct btreeNode \*\*newNode) {

int median, j;

if (pos > MIN)

median = MIN + 1;

else

median = MIN;

\*newNode = (struct btreeNode \*)malloc(sizeof(struct btreeNode));

j = median + 1;

while (j <= MAX) {

(\*newNode)->val[j - median] = node->val[j];

(\*newNode)->link[j - median] = node->link[j];

j++;

}

node->count = median;

(\*newNode)->count = MAX - median;

if (pos <= MIN) {

addValToNode(val, pos, node, child);

} else {

addValToNode(val, pos - median, \*newNode, child);

}

\*pval = node->val[node->count];

(\*newNode)->link[0] = node->link[node->count];

node->count--;

}

/\* sets the value val in the node \*/

int setValueInNode(int val, int \*pval,

struct btreeNode \*node, struct btreeNode \*\*child) {

int pos;

if (!node) {

\*pval = val;

\*child = NULL;

return 1;

}

if (val < node->val[1]) {

pos = 0;

} else {

for (pos = node->count;

(val < node->val[pos] && pos > 1); pos--);

if (val == node->val[pos]) {

printf("Duplicates not allowed\n");

return 0;

}

}

if (setValueInNode(val, pval, node->link[pos], child)) {

if (node->count < MAX) {

addValToNode(\*pval, pos, node, \*child);

} else {

splitNode(\*pval, pval, pos, node, \*child, child);

return 1;

}

}

return 0;

}

/\* insert val in B-Tree \*/

void insertion(int val) {

int flag, i;

struct btreeNode \*child;

flag = setValueInNode(val, &i, root, &child);

if (flag)

root = createNode(i, child);

}

/\* copy successor for the value to be deleted \*/

void copySuccessor(struct btreeNode \*myNode, int pos) {

struct btreeNode \*dummy;

dummy = myNode->link[pos];

for (;dummy->link[0] != NULL;)

dummy = dummy->link[0];

myNode->val[pos] = dummy->val[1];

}

/\* removes the value from the given node and rearrange values \*/

void removeVal(struct btreeNode \*myNode, int pos) {

int i = pos + 1;

while (i <= myNode->count) {

myNode->val[i - 1] = myNode->val[i];

myNode->link[i - 1] = myNode->link[i];

i++;

}

myNode->count--;

}

/\* shifts value from parent to right child \*/

void doRightShift(struct btreeNode \*myNode, int pos) {

struct btreeNode \*x = myNode->link[pos];

int j = x->count;

while (j > 0) {

x->val[j + 1] = x->val[j];

x->link[j + 1] = x->link[j];

}

x->val[1] = myNode->val[pos];

x->link[1] = x->link[0];

x->count++;

x = myNode->link[pos - 1];

myNode->val[pos] = x->val[x->count];

myNode->link[pos] = x->link[x->count];

x->count--;

return;

}

/\* shifts value from parent to left child \*/

void doLeftShift(struct btreeNode \*myNode, int pos) {

int j = 1;

struct btreeNode \*x = myNode->link[pos - 1];

x->count++;

x->val[x->count] = myNode->val[pos];

x->link[x->count] = myNode->link[pos]->link[0];

x = myNode->link[pos];

myNode->val[pos] = x->val[1];

x->link[0] = x->link[1];

x->count--;

while (j <= x->count) {

x->val[j] = x->val[j + 1];

x->link[j] = x->link[j + 1];

j++;

}

return;

}

/\* merge nodes \*/

void mergeNodes(struct btreeNode \*myNode, int pos) {

int j = 1;

struct btreeNode \*x1 = myNode->link[pos], \*x2 = myNode->link[pos - 1];

x2->count++;

x2->val[x2->count] = myNode->val[pos];

x2->link[x2->count] = myNode->link[0];

while (j <= x1->count) {

x2->count++;

x2->val[x2->count] = x1->val[j];

x2->link[x2->count] = x1->link[j];

j++;

}

j = pos;

while (j < myNode->count) {

myNode->val[j] = myNode->val[j + 1];

myNode->link[j] = myNode->link[j + 1];

j++;

}

myNode->count--;

free(x1);

}

/\* adjusts the given node \*/

void adjustNode(struct btreeNode \*myNode, int pos) {

if (!pos) {

if (myNode->link[1]->count > MIN) {

doLeftShift(myNode, 1);

} else {

mergeNodes(myNode, 1);

}

} else {

if (myNode->count != pos) {

if(myNode->link[pos - 1]->count > MIN) {

doRightShift(myNode, pos);

} else {

if (myNode->link[pos + 1]->count > MIN) {

doLeftShift(myNode, pos + 1);

} else {

mergeNodes(myNode, pos);

}

}

} else {

if (myNode->link[pos - 1]->count > MIN)

doRightShift(myNode, pos);

else

mergeNodes(myNode, pos);

}

}

}

/\* delete val from the node \*/

int delValFromNode(int val, struct btreeNode \*myNode) {

int pos, flag = 0;

if (myNode) {

if (val < myNode->val[1]) {

pos = 0;

flag = 0;

} else {

for (pos = myNode->count;

(val < myNode->val[pos] && pos > 1); pos--);

if (val == myNode->val[pos]) {

flag = 1;

} else {

flag = 0;

}

}

if (flag) {

if (myNode->link[pos - 1]) {

copySuccessor(myNode, pos);

flag = delValFromNode(myNode->val[pos], myNode->link[pos]);

if (flag == 0) {

printf("Given data is not present in B-Tree\n");

}

} else {

removeVal(myNode, pos);

}

} else {

flag = delValFromNode(val, myNode->link[pos]);

}

if (myNode->link[pos]) {

if (myNode->link[pos]->count < MIN)

adjustNode(myNode, pos);

}

}

return flag;

}

/\* delete val from B-tree \*/

void deletion(int val, struct btreeNode \*myNode) {

struct btreeNode \*tmp;

if (!delValFromNode(val, myNode)) {

printf("Given value is not present in B-Tree\n");

return;

} else {

if (myNode->count == 0) {

tmp = myNode;

myNode = myNode->link[0];

free(tmp);

}

}

root = myNode;

return;

}

/\* search val in B-Tree \*/

void searching(int val, int \*pos, struct btreeNode \*myNode) {

if (!myNode) {

return;

}

if (val < myNode->val[1]) {

\*pos = 0;

} else {

for (\*pos = myNode->count;

(val < myNode->val[\*pos] && \*pos > 1); (\*pos)--);

if (val == myNode->val[\*pos]) {

printf("Given data %d is present in B-Tree", val);

return;

}

}

searching(val, pos, myNode->link[\*pos]);

return;

}

/\* B-Tree Traversal \*/

void traversal(struct btreeNode \*myNode) {

int i;

if (myNode) {

for (i = 0; i < myNode->count; i++) {

traversal(myNode->link[i]);

printf("%d ", myNode->val[i + 1]);

}

traversal(myNode->link[i]);

}

}

int main() {

int val, ch;

while (1) {

printf("1. Insertion\t2. Deletion\n");

printf("3. Searching\t4. Traversal\n");

printf("5. Exit\nEnter your choice:");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("Enter your input:");

scanf("%d", &val);

insertion(val);

break;

case 2:

printf("Enter the element to delete:");

scanf("%d", &val);

deletion(val, root);

break;

case 3:

printf("Enter the element to search:");

scanf("%d", &val);

searching(val, &ch, root);

break;

case 4:

traversal(root);

break;

case 5:

exit(0);

default:

printf("U have entered wrong option!!\n");

break;

}

printf("\n");

}

}

**Output:**

1. Insertion 2. Deletion 3. Searching 4. Traversal 5. Exit

Enter your choice:1

Enter your input:70

1. Insertion 2. Deletion 3. Searching 4. Traversal 5. Exit

Enter your choice:1

Enter your input:17

1. Insertion 2. Deletion 3. Searching 4. Traversal 5. Exit

Enter your choice:1

Enter your input:67

1. Insertion 2. Deletion 3. Searching 4. Traversal 5. Exit

Enter your choice:1

Enter your input:89

1. Insertion 2. Deletion 3. Searching 4. Traversal 5. Exit

Enter your choice:4

17 67 70 89

1. Insertion 2. Deletion 3. Searching 4. Traversal 5. Exit

Enter your choice:3

Enter the element to search:70

Given data 70 is present in B-Tree

1. Insertion 2. Deletion 3. Searching 4. Traversal 5. Exit

Enter your choice:2

Enter the element to delete:17

1. Insertion 2. Deletion 3. Searching 4. Traversal 5. Exit

Enter your choice:4

67 70 89

1. Insertion 2. Deletion 3. Searching 4. Traversal 5. Exit

Enter your choice:5