```
//Created By Ritwik Chandra Pandey on 3 Nov' 2021
//B-tree: Deletion, Search



#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#define MAX 4
#define MIN 2


struct BTreeNode {
    int val[MAX + 1], count;
    struct BTreeNode *link[MAX + 1];
};
typedef struct BTreeNode* BTNODE;
BTNODE root;
/* creating new node */
BTNODE createNode(int val,BTNODE child) {
    BTNODE newNode = (BTNODE)malloc(sizeof(struct BTreeNode));
    newNode->val[1] = val;
    newNode->count = 1;
    newNode->link[0] = root;
    newNode->link[1] = child;
    return newNode;
}
void addValToNode(int val, int pos, BTNODE node, BTNODE 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++;
}
void splitNode(int val, int *pval, int pos, BTNODE node,BTNODE child, BTNODE *newNode) {
```

```
    int median, j;
    if (pos > MIN)
        median = MIN + 1;
    else
        median = MIN;
    *newNode = (BTNODE)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--;
}
int setValueInNode(int val, int *pval,BTNODE node,BTNODE* 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");
        }
```

```
    }
    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;
}
void insertNodeInBTree(int val) {
    int flag, i;
    BTNODE child;
    flag = setValueInNode(val, &i, root, &child);
    if (flag) {
        root = createNode(i, child);
    }
}
void copySuccessor(BTNODE myNode, int pos) {
        BTNODE dummy = myNode->link[pos];
        while(dummy->link[0]!=NULL){
                dummy = dummy->link[0];
        }
        myNode->val[pos] = dummy->val[1];
}
void removeVal(BTNODE 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--;
}
void doRightShift(BTNODE myNode, int pos) {
        BTNODE 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;
}
void doLeftShift(BTNODE myNode, int pos) {
        int j = 1;
        BTNODE 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;
}
void mergeNodes(BTNODE myNode, int pos) {
        int j =1;
        BTNODE x1, x2;
        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);
}
void adjustNode(BTNODE myNode, int pos) {
        if(pos!=0){
                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);
                        }
                }
        }
}
int delValFromNode(int val,BTNODE myNode) {
```

```
	int pos=0, flag=0;
	if(myNode!=NULL){
		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==1){
			if(myNode->link[pos-1]!=0){
				copySuccessor(myNode,pos);
				flag = delValFromNode(myNode->val[pos], myNode->link[pos]);
				if(flag==0){
					printf("Element not found in B-Tree.\n");
				}
			}else{
			    removeVal(myNode,pos);
			}
		}else{
			flag = delValFromNode(val,myNode->link[pos]);
		}
		if(myNode->link[pos]!=0){
			if(myNode->link[pos]->count < MIN)
			   adjustNode(myNode, pos);
		}
	}
	return flag;
}
void deleteNodeInBTree(int val,BTNODE myNode) {
	BTNODE tmp;
	if(delValFromNode(val,myNode)!=1){
		printf("Element not found in B-Tree.\n");
	}else{
		if(myNode->count==0){
```

```
                        tmp = myNode;
                        myNode = myNode->link[0];
                        free(tmp);
                }
        }
        root = myNode;
        return;
}
void searchNodeInBTree(int val, int *pos,BTNODE myNode) {
        if(myNode==NULL){
                *pos = 0;
                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]){
                        return;
                }
        }
        searchNodeInBTree(val, pos, myNode->link[*pos]);
        return;
}
void traverseBTree(BTNODE myNode) {
    int i;
    if (myNode) {
        for (i = 0; i < myNode->count; i++) {
            traverseBTree(myNode->link[i]);
            printf("%d ",myNode->val[i + 1]);
        }
        traverseBTree(myNode->link[i]);
    }
}

int main() {
        int ele, op, pos;
        while(1)
        {
                printf("1.Insert 2.Delete 3.Search 4.Traversal 5.Exit\n");
```

```
		printf("Enter your option : ");
		scanf("%d", &op);
		switch(op) {
			case 1:printf("Enter an element to be inserted : ");
				scanf("%d", &ele);
				insertNodeInBTree(ele);
				break;
			case 2: printf("Enter the element to be deleted : ");
				scanf("%d", &ele);
				deleteNodeInBTree(ele,root);
				break;
			case 3: printf("Enter the element to be searched : ");
				scanf("%d", &ele);
				searchNodeInBTree(ele,&pos,root);
				if(pos)
					printf("Element found in B-Tree.\n");
				else
					printf("Element not found in B-Tree.\n");
				pos=0;
				break;
			case 4:
				if(root == NULL) {
					printf("B-Tree is empty.\n");
				}
				else {
					printf("Elements of the B-Tree : ");
					traverseBTree(root);
					printf("\n");
				}
				break;
			case 5:exit(0);
		}
	}
}
```