**Assignment 6(a)**

**Implement B-Tree of order three and perform following operations:**

**1. Insert**

**2. Level order display**

**3. Delete**

#include <iostream>

using namespace std;

struct node {

int key[3]; // B-Tree of order 3 can hold up to 2 keys per node

node\* child[4]; // One extra child pointer than keys

int count; // To store the number of keys in the node

};

node\* root = NULL;

// Create a new node with a given key and children

node\* createNode(int val, node\* left, node\* right) {

node\* temp = new node;

temp->key[0] = val;

temp->count = 1;

temp->child[0] = left;

temp->child[1] = right;

for (int i = 2; i < 4; ++i) {

temp->child[i] = nullptr;

}

return temp;

}

// Add a value to a node at a specific position, shifting keys as needed

void addToNode(int val, int pos, node\* ptr, node\* child) {

for (int i = ptr->count; i > pos; i--) {

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

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

}

ptr->key[pos] = val;

ptr->child[pos + 1] = child;

ptr->count++;

}

// Split a node when it exceeds the maximum key count

void splitNode(int val, int\* pval, int pos, node\* ptr, node\* child, node\*\* newChild) {

int tempKey[4]; // Temporary array for keys

node\* tempChild[5]; // Temporary array for child pointers

for (int i = 0; i < 3; ++i) {

tempKey[i] = ptr->key[i];

tempChild[i] = ptr->child[i];

}

tempChild[3] = ptr->child[3];

for (int i = 2; i > pos; i--) {

tempKey[i + 1] = tempKey[i];

tempChild[i + 1] = tempChild[i];

}

tempKey[pos] = val;

tempChild[pos] = child;

\*pval = tempKey[1]; // Middle key becomes the parent

\*newChild = new node;

(\*newChild)->count = 0;

ptr->count = 0;

for (int i = 0; i < 1; i++) {

ptr->key[i] = tempKey[i];

ptr->child[i] = tempChild[i];

ptr->count++;

}

ptr->child[ptr->count] = tempChild[ptr->count];

for (int i = 2, j = 0; i < 3; i++, j++) {

(\*newChild)->key[j] = tempKey[i];

(\*newChild)->child[j] = tempChild[i];

(\*newChild)->count++;

}

(\*newChild)->child[0] = tempChild[2];

}

// Insert a value into the B-Tree recursively

bool insertHelper(int val, int\* pval, node\* ptr, node\*\* newChild) {

if (ptr == NULL) {

\*pval = val;

\*newChild = NULL;

return true;

}

int pos;

if (val < ptr->key[0]) pos = 0;

else {

for (pos = ptr->count; pos > 0 && val < ptr->key[pos - 1]; pos--);

}

if (insertHelper(val, pval, ptr->child[pos], newChild)) {

if (ptr->count < 2) {

addToNode(\*pval, pos, ptr, \*newChild);

return false;

} else {

splitNode(\*pval, pval, pos, ptr, \*newChild, newChild);

return true;

}

}

return false;

}

// Insert a new key into the B-Tree

void insert(int val) {

int i;

node\* child;

if (insertHelper(val, &i, root, &child)) {

node\* temp = new node;

temp->count = 1;

temp->key[0] = i;

temp->child[0] = root;

temp->child[1] = child;

root = temp;

}

}

// Print the tree in level order

void printLevelOrder(node\* ptr) {

if (ptr == NULL) {

cout << "Tree is empty\n";

return;

}

node\* level[100];

int front = 0, rear = 0;

level[rear++] = ptr;

while (front < rear) {

int currentLevelSize = rear - front;

for (int i = 0; i < currentLevelSize; i++) {

node\* temp = level[front++];

for (int j = 0; j < temp->count; j++) {

cout << temp->key[j] << " ";

}

for (int j = 0; j <= temp->count; j++) {

if (temp->child[j] != NULL) {

level[rear++] = temp->child[j];

}

}

}

cout << "\n";

}

}

// Delete a key (only from leaf nodes for simplicity)

void deleteVal(node\* ptr, int val) {

if (ptr == NULL) {

cout << "Tree is empty\n";

return;

}

int i;

for (i = 0; i < ptr->count && ptr->key[i] < val; i++);

if (i < ptr->count && ptr->key[i] == val) {

if (ptr->child[0] == NULL) { // Leaf node

for (int j = i; j < ptr->count - 1; j++) {

ptr->key[j] = ptr->key[j + 1];

}

ptr->count--;

cout << "Deleted " << val << " from leaf node.\n";

} else {

cout << "Only leaf node deletion is supported.\n";

}

} else {

deleteVal(ptr->child[i], val);

}

}

// Main function for menu-driven interface

int main() {

int choice, num;

while (1) {

cout << "\n1. Insert\n2. Level Order Display\n3. Delete\n4. Exit\nEnter choice: ";

cin >> choice;

if (choice == 1) {

cout << "Enter number to insert: ";

cin >> num;

insert(num);

} else if (choice == 2) {

cout << "\nB-Tree Level Order:\n";

printLevelOrder(root);

} else if (choice == 3) {

cout << "Enter number to delete (from leaf): ";

cin >> num;

deleteVal(root, num);

} else if (choice == 4) {

break;

} else {

cout << "Invalid choice.\n";

}

}

return 0;

}