ASSIGNMENT 2

2-3 Tree Insertion

CSA0303 – DATA STRUCTURES FOR Problem Solving

NAME: KAVYA SHRI G

REG.NO: 192421052

AIM: To implement insertion into a 2-3 Tree and demonstrate splitting of nodes during the insertion of integers.

ALGORITHM:

1. Define a node structure that holds up to 2 keys and 3 children, and a flag to indicate if it's a leaf.
2. Traverse down the tree to find the correct position for the key.
3. Insert the key in sorted order in the leaf node.
4. If a node overflows (3 keys), split it by promoting the middle key to the parent and creating two new children.
5. If the root overflows, create a new root and repeat the splitting process.
6. Display the tree after each insertion.

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

typedef struct Node {

   int keys[2];

   struct Node* children[3];

   int keyCount;

   bool isLeaf;

} Node;
```

```
Node* createNode(int key, bool isLeaf) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->keys[0] = key;
    newNode->keyCount = 1;
    newNode->isLeaf = isLeaf;
    for (int i = 0; i < 3; i++) newNode->children[i] = NULL;
    return newNode;
}
Node* insertNonFull(Node* root, int key, Node** newChild) {
    *newChild = NULL;
    if (root->isLeaf) {
        if (root->keyCount == 1) {
            if (key < root->keys[0]) {
                root->keys[1] = root->keys[0];
                root->keys[0] = key;
            } else {
                root->keys[1] = key;
            }
            root->keyCount = 2;
            return root;
        } else {
            int keys[3] = {root->keys[0], root->keys[1], key};
            for (int i = 0; i < 2; i++)
                for (int j = i + 1; j < 3; j++)
                    if (keys[i] > keys[j]) {
                        int temp = keys[i];
```

```
                keys[i] = keys[j];
                keys[j] = temp;
            }
        Node* left = createNode(keys[0], true);
        Node* right = createNode(keys[2], true);
        Node* midNode = createNode(keys[1], false);
        midNode->children[0] = left;
        midNode->children[1] = right;
        *newChild = midNode;
        return NULL;
    }
} else {
    int pos;
    if (key < root->keys[0]) pos = 0;
    else if (root->keyCount == 1 || key < root->keys[1]) pos = 1;
    else pos = 2;
    Node* child = insertNonFull(root->children[pos], key, newChild);
    if (*newChild == NULL) return root;
    if (root->keyCount == 1) {
        if (pos == 0) {
            root->keys[1] = root->keys[0];
            root->keys[0] = (*newChild)->keys[0];
            root->children[2] = root->children[1];
            root->children[0] = (*newChild)->children[0];
            root->children[1] = (*newChild)->children[1];
        } else {
```

```
        root->keys[1] = (*newChild)->keys[0];
        root->children[1] = (*newChild)->children[0];
        root->children[2] = (*newChild)->children[1];
    }
    root->keyCount = 2;
    return root;
} else {
    int keys[3] = {root->keys[0], root->keys[1], (*newChild)->keys[0]};
    Node* tempChildren[4];
    if (pos == 0) {
        tempChildren[0] = (*newChild)->children[0];
        tempChildren[1] = (*newChild)->children[1];
        tempChildren[2] = root->children[1];
        tempChildren[3] = root->children[2];
    } else if (pos == 1) {
        tempChildren[0] = root->children[0];
        tempChildren[1] = (*newChild)->children[0];
        tempChildren[2] = (*newChild)->children[1];
        tempChildren[3] = root->children[2];
    } else {
        tempChildren[0] = root->children[0];
        tempChildren[1] = root->children[1];
        tempChildren[2] = (*newChild)->children[0];
        tempChildren[3] = (*newChild)->children[1];
    }
    for (int i = 0; i < 2; i++)
```

```
            for (int j = i + 1; j < 3; j++)
                if (keys[i] > keys[j]) {
                    int temp = keys[i];
                    keys[i] = keys[j];
                    keys[j] = temp;
                }
            Node* left = createNode(keys[0], false);
            left->children[0] = tempChildren[0];
            left->children[1] = tempChildren[1];
            Node* right = createNode(keys[2], false);
            right->children[0] = tempChildren[2];
            right->children[1] = tempChildren[3];
            Node* mid = createNode(keys[1], false);
            mid->children[0] = left;
            mid->children[1] = right;

            *newChild = mid;
            return NULL;
        }
    }
}
Node* insert(Node* root, int key) {
    Node* newChild = NULL;
    Node* updated = insertNonFull(root, key, &newChild);
    if (newChild != NULL) return newChild;
    return updated;
```

```
}
void inorder(Node* root) {
    if (root == NULL) return;
    if (root->isLeaf) {
        for (int i = 0; i < root->keyCount; i++)
            printf("%d ", root->keys[i]);
    } else {
        if (root->keyCount == 1) {
            inorder(root->children[0]);
            printf("%d ", root->keys[0]);
            inorder(root->children[1]);
        } else {
            inorder(root->children[0]);
            printf("%d ", root->keys[0]);
            inorder(root->children[1]);
            printf("%d ", root->keys[1]);
            inorder(root->children[2]);
        }
    }
}
int main() {
    int arr[] = {10, 20, 5, 6, 12, 30, 7, 17};
    int n = sizeof(arr) / sizeof(arr[0]);
    Node* root = createNode(arr[0], true);
    for (int i = 1; i < n; i++) {
        root = insert(root, arr[i]);
```

```
    }
    printf("In-order traversal of 2-3 tree:\n");
    inorder(root);
    printf("\n");
    return 0;
}
```

OUTPT