Generating a 2-3 Tree in c

```c
#include <stdio.h>

#include <stdlib.h>

#define MAX_KEYS 2

struct Node {

    int num_keys;

    int keys[MAX_KEYS];

    struct Node* children[MAX_KEYS + 1];

};

struct Node* createNode(int key);

void insert(struct Node** root, int key);

void splitChild(struct Node* parent, int index, struct Node* child);

void insertNonFull(struct Node* node, int key);

void printTree(struct Node* root);

int main() {

    struct Node* root = NULL;

    int keys[] = {10, 5, 15, 3, 7, 20, 12};

    for (int i = 0; i < sizeof(keys) / sizeof(keys[0]); i++) {

        insert(&root, keys[i]);

        printf("Inserted key: %d\n", keys[i]);

    }

    printf("2-3 Tree:\n");

    printTree(root);

    return 0;

}
```

```c
struct Node* createNode(int key) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    newNode->num_keys = 1;

    newNode->keys[0] = key;

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

        newNode->children[i] = NULL;

    }

    return newNode;

}

void insert(struct Node** root, int key) {

    if (*root == NULL) {

        *root = createNode(key);

    } else {

        if ((*root)->num_keys == MAX_KEYS) {

            struct Node* newRoot = createNode((*root)->keys[1]);

            newRoot->children[0] = *root;

            splitChild(newRoot, 1, *root);

            *root = newRoot;

            insertNonFull(newRoot, key);

        } else {

            insertNonFull(*root, key);

        }

    }

}

void splitChild(struct Node* parent, int index, struct Node* child) {
```

```c
        struct Node* newChild = createNode(child->keys[1]);

        newChild->children[0] = child->children[1];

        newChild->children[1] = child->children[2];

        child->num_keys = 1;

        parent->children[index] = child;

        parent->children[index + 1] = newChild;
}
void insertNonFull(struct Node* node, int key) {

        int i = node->num_keys - 1;

        if (node->children[0] == NULL) {

                while (i >= 0 && key < node->keys[i]) {

                        node->keys[i + 1] = node->keys[i];

                        i--;

                }

                node->keys[i + 1] = key;

                node->num_keys++;

        } else {

                while (i >= 0 && key < node->keys[i]) {

                        i--;

                }

                i++;

                if (node->children[i]->num_keys == MAX_KEYS) {

                        splitChild(node, i, node->children[i]);

                        if (key > node->keys[i]) {

                                i++;
```

```c
            }
        }

        insertNonFull(node->children[i], key);
    }
}


void printTree(struct Node* root) {
    if (root != NULL) {
        for (int i = 0; i < root->num_keys; i++) {
            printf("%d ", root->keys[i]);
        }
        printf("\n");
        for (int i = 0; i < root->num_keys + 1; i++) {
            printTree(root->children[i]);
        }
    }
}
```

output:

Inserted key: 10

Inserted key: 5

Inserted key: 15

Inserted key: 3

Inserted key: 20

Inserted key: 12

2-3 tree:

10

3 7

5

2.Generating a 2-3-4 Tree in c

```c
#include <stdio.h>

#include <stdlib.h>


#define MAX_KEYS 3


struct Node {

    int num_keys;

    int keys[MAX_KEYS];

    struct Node* children[MAX_KEYS + 1];

};


struct Node* createNode(int key);

void insert(struct Node** root, int key);

void splitChild(struct Node* parent, int index, struct Node* child);

void insertNonFull(struct Node* node, int key);

void printTree(struct Node* root);


int main() {

    struct Node* root = NULL;

    int keys[] = {10, 20, 5, 6, 12, 30, 7};
```

```c
    for (int i = 0; i < sizeof(keys) / sizeof(keys[0]); i++) {

        insert(&root, keys[i]);

        printf("Inserted key: %d\n", keys[i]);

        printTree(root);

        printf("\n");

    }


    return 0;

}


struct Node* createNode(int key) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    newNode->num_keys = 1;

    newNode->keys[0] = key;

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

        newNode->children[i] = NULL;

    }

    return newNode;

}


void insert(struct Node** root, int key) {

    if (*root == NULL) {

        *root = createNode(key);

    } else {

        if ((*root)->num_keys == MAX_KEYS) {
```

```c
                struct Node* newRoot = createNode((*root)->keys[1]);

                newRoot->children[0] = *root;

                splitChild(newRoot, 0, *root);

                insertNonFull(newRoot, key);

                *root = newRoot;

        } else {

                insertNonFull(*root, key);

        }

    }

}


void splitChild(struct Node* parent, int index, struct Node* child) {

        struct Node* newChild = createNode(child->keys[2]);

        newChild->children[0] = child->children[2];

        child->num_keys = 1;

        parent->children[index + 1] = newChild;

}


void insertNonFull(struct Node* node, int key) {

        int i = node->num_keys - 1;

        if (node->children[0] == NULL) {

                while (i >= 0 && key < node->keys[i]) {

                        node->keys[i + 1] = node->keys[i];

                        i--;

                }
```

```c
            node->keys[i + 1] = key;

            node->num_keys++;

        } else {

            while (i >= 0 && key < node->keys[i]) {

                i--;

            }

            i++;

            if (node->children[i]->num_keys == MAX_KEYS) {

                splitChild(node, i, node->children[i]);

                if (key > node->keys[i]) {

                    i++;

                }

            }

            insertNonFull(node->children[i], key);

        }

}


void printTree(struct Node* root) {

    if (root != NULL) {

        for (int i = 0; i < root->num_keys; i++) {

            printf("%d ", root->keys[i]);

        }

        printf("\n");

        for (int i = 0; i < root->num_keys + 1; i++) {

            printTree(root->children[i]);
```

```
        }
    }
}
```

output:

Inserted key: 10

10

Inserted key: 20

10 20

Inserted key: 5

5 10 20

Inserted key: 6

10

5 6

Inserted key: 12

10

5 6

12 20

Inserted key:30

10

5 6