1.Solve for trie 2-3 tree 50,90,20,10,30,40,70,60,80,120,150,100,110,130,140,160

Code:

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

#define MAX_KEYS 2
#define MIN_KEYS 1

typedef struct Node {
    int keys[MAX_KEYS + 1]; // Two keys maximum
    struct Node* children[MAX_KEYS + 2]; // Three children maximum
    int numKeys; // Number of keys in this node
    int isLeaf; // Flag to indicate if the node is a leaf
} Node;

// Function prototypes
Node* createNode(int key, int isLeaf);
void splitChild(Node* parent, int index, Node* fullChild);
void insertNonFull(Node* node, int key);
void insert(Node** root, int key);
void printTree(Node* root, int level);
Node* findMinNode(Node* root);
void deleteNode(Node* root, int key);

// Create a new node
Node* createNode(int key, int isLeaf) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->keys[0] = key;
    newNode->numKeys = 1;
    newNode->isLeaf = isLeaf;
    newNode->children[0] = NULL;
    newNode->children[1] = NULL;
    newNode->children[2] = NULL;
    return newNode;
}

// Split a full child node
void splitChild(Node* parent, int index, Node* fullChild) {
    Node* newChild = createNode(0, fullChild->isLeaf);
    int midKey = fullChild->keys[1];
    parent->keys[index] = midKey;
    parent->numKeys = 1;

    // Move second key and children to new child
    newChild->keys[0] = fullChild->keys[2];
    newChild->children[0] = fullChild->children[2];
```

```
    newChild->children[1] = fullChild->children[3];
    if (!fullChild->isLeaf) {
        newChild->children[0] = fullChild->children[2];
        newChild->children[1] = fullChild->children[3];
    }
    fullChild->numKeys = 1;
    parent->children[index + 1] = newChild;
}
// Insert a key into a non-full node
void insertNonFull(Node* node, int key) {
    int i = node->numKeys - 1;

    if (node->isLeaf) {
        // Find location to insert
        while (i >= 0 && key < node->keys[i]) {
            node->keys[i + 1] = node->keys[i];
            i--;
        }
        node->keys[i + 1] = key;
        node->numKeys++;
    } else {
        // Find child to insert into
        while (i >= 0 && key < node->keys[i]) {
            i--;
        }
        i++;
        if (node->children[i]->numKeys == MAX_KEYS) {
            splitChild(node, i, node->children[i]);
            if (key > node->keys[i]) {
                i++;
            }
        }
        insertNonFull(node->children[i], key);
    }
}
// Insert a key into the 2-3 tree
void insert(Node** root, int key) {
    Node* r = *root;
    if (r->numKeys == MAX_KEYS) {
        Node* s = createNode(0, 0);
        *root = s;
        s->children[0] = r;
        splitChild(s, 0, r);
        insertNonFull(s, key);
    } else {
        insertNonFull(r, key);
    }
}
```

```
// Print the tree (For debugging)
void printTree(Node* root, int level) {
    if (root != NULL) {
        printf("Level %d: ", level);
        for (int i = 0; i < root->numKeys; i++) {
            printf("%d ", root->keys[i]);
        }
        printf("\n");
        if (root->isLeaf == 0) {
            for (int i = 0; i <= root->numKeys; i++) {
                printTree(root->children[i], level + 1);
            }
        }
    }
}
// Main function to demonstrate the 2-3 tree operations
int main() {
    Node* root = createNode(0, 1); // Create a root node (leaf initially)
    // Insert values into the 2-3 tree
    int values[] = {50, 90, 20, 10, 30, 40, 70, 60, 80, 120, 150, 100, 110, 130, 140, 160};
    int n = sizeof(values) / sizeof(values[0]);

    for (int i = 0; i < n; i++) {
        insert(&root, values[i]);
    }
    // Print the tree
    printTree(root, 0);
    return 0;
}
```

2. Solve for trie 2-3-4 tree30,60,22,10,17,24,2629,48,40,41,52,70,62,65,80,72,90,81,85,95.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_KEYS 3
#define MIN_KEYS 1
typedef struct Node {
    int keys[MAX_KEYS + 1];  // Three keys maximum
    struct Node* children[MAX_KEYS + 2];  // Four children maximum
    int numKeys;  // Number of keys in this node
    int isLeaf;   // Flag to indicate if the node is a leaf
} Node;
// Function prototypes
Node* createNode(int key, int isLeaf);
void splitChild(Node* parent, int index, Node* fullChild);
void insertNonFull(Node* node, int key);
```

```
void insert(Node** root, int key);
void printTree(Node* root, int level);
// Create a new node
Node* createNode(int key, int isLeaf) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->keys[0] = key;
    newNode->numKeys = 1;
    newNode->isLeaf = isLeaf;
    for (int i = 1; i <= MAX_KEYS; i++) {
        newNode->keys[i] = 0;
    }
    for (int i = 0; i <= MAX_KEYS + 1; i++) {
        newNode->children[i] = NULL;
    }
    return newNode;
}
// Split a full child node
void splitChild(Node* parent, int index, Node* fullChild) {
    Node* newChild = createNode(0, fullChild->isLeaf);
    parent->children[index + 1] = newChild;
    newChild->numKeys = MIN_KEYS;
    // Move the second half of the fullChild keys and children to newChild
    for (int j = 0; j < MIN_KEYS; j++) {
        newChild->keys[j] = fullChild->keys[j + MIN_KEYS + 1];
    }
    if (!fullChild->isLeaf) {
        for (int j = 0; j <= MIN_KEYS; j++) {
            newChild->children[j] = fullChild->children[j + MIN_KEYS + 1];
        }
    }
    parent->keys[index] = fullChild->keys[MIN_KEYS];
    parent->numKeys = MIN_KEYS + 1;
    fullChild->numKeys = MIN_KEYS;
}
// Insert a key into a non-full node
void insertNonFull(Node* node, int key) {
    int i = node->numKeys - 1;
    if (node->isLeaf) {
        // Insert key into leaf node
        while (i >= 0 && key < node->keys[i]) {
            node->keys[i + 1] = node->keys[i];
            i--;
        }
        node->keys[i + 1] = key;
        node->numKeys++;
    } else {
        // Find child to insert into
        while (i >= 0 && key < node->keys[i]) {
```

```
            i--;
        }
        i++;

        if (node->children[i]->numKeys == MAX_KEYS) {
            splitChild(node, i, node->children[i]);
            if (key > node->keys[i]) {
                i++;
            }
        }
        insertNonFull(node->children[i], key);
    }
}
// Insert a key into the 2-3-4 tree
void insert(Node** root, int key) {
    Node* r = *root;
    if (r->numKeys == MAX_KEYS) {
        Node* s = createNode(0, 0);
        *root = s;
        s->children[0] = r;
        splitChild(s, 0, r);
        insertNonFull(s, key);
    } else {
        insertNonFull(r, key);
    }
}
// Print the tree (For debugging)
void printTree(Node* root, int level) {
    if (root != NULL) {
        printf("Level %d: ", level);
        for (int i = 0; i < root->numKeys; i++) {
            printf("%d ", root->keys[i]);
        }
        printf("\n");
        if (root->isLeaf == 0) {
            for (int i = 0; i <= root->numKeys; i++) {
                printTree(root->children[i], level + 1);
            }
        }
    }
}

// Main function to demonstrate the 2-3-4 tree operations
int main() {
    Node* root = createNode(0, 1); // Create a root node (leaf initially)
    // Insert values into the 2-3-4 tree
    int values[] = {30, 60, 22, 10, 17, 24, 26, 29, 48, 40, 41, 52, 70, 62, 65, 80, 72, 90, 81, 85,
95};
```

```
    int n = sizeof(values) / sizeof(values[0]);
    for (int i = 0; i < n; i++) {
        insert(&root, values[i]);
    }
    // Print the tree
    printTree(root, 0);
    return 0;
}
```