1.TRIE

program:

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

typedef struct Node {
   int n;              // Number of keys in this node
   int keys[3];        // An array to store keys
   struct Node *children[4]; // An array of child pointers
   int isLeaf;         // Is true when node is a leaf. Otherwise false
} Node;

Node *createNode(int key, Node *child) {
   Node *newNode = (Node *)malloc(sizeof(Node));
   newNode->keys[0] = key;
   newNode->n = 1;
   newNode->children[0] = NULL;
   newNode->children[1] = child;
   newNode->isLeaf = 1;
   return newNode;
}

void insertNonFull(Node *node, int key);
void splitChild(Node *node, int i);

void insert(Node **root, int key) {
   if (*root == NULL) {
      *root = createNode(key, NULL);
   } else {
      Node *r = *root;
      if (r->n == 3) {
```

```
            Node *s = createNode(0, r);
            s->isLeaf = 0;
            splitChild(s, 0);
            insertNonFull(s, key);
            *root = s;
        } else {
            insertNonFull(r, key);
        }
    }
}

void insertNonFull(Node *node, int key) {
    int i = node->n - 1;
    if (node->isLeaf) {
        while (i >= 0 && key < node->keys[i]) {
            node->keys[i + 1] = node->keys[i];
            i--;
        }
        node->keys[i + 1] = key;
        node->n++;
    } else {
        while (i >= 0 && key < node->keys[i]) {
            i--;
        }
        i++;
        if (node->children[i]->n == 3) {
            splitChild(node, i);
            if (key > node->keys[i]) {
                i++;
            }
        }
```

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

void splitChild(Node *node, int i) {
    Node *y = node->children[i];
    Node *z = createNode(y->keys[2], NULL);
    node->children[i + 1] = z;
    node->keys[i] = y->keys[1];
    node->n++;
    y->n = 1;
    z->isLeaf = y->isLeaf;
    if (!y->isLeaf) {
        for (int j = 0; j < 2; j++) {
            z->children[j] = y->children[j + 2];
        }
    }
}

int search(Node *node, int key) {
    int i = 0;
    while (i < node->n && ke
```

output:

the --- Present in trie

these --- Not present in trie

their --- Present in trie

thaw --- Not present in trie

Aborted

2. 2-3 TREE

Program:

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

typedef struct Node {
   int n;              // Number of keys in this node
   int keys[3];        // An array to store keys
   struct Node *children[4]; // An array of child pointers
   int isLeaf;         // Is true when node is a leaf. Otherwise false
} Node;

Node *createNode(int key, Node *child) {
   Node *newNode = (Node *)malloc(sizeof(Node));
   newNode->keys[0] = key;
   newNode->n = 1;
   newNode->children[0] = NULL;
   newNode->children[1] = child;
   newNode->isLeaf = 1;
   return newNode;
}

void insertNonFull(Node *node, int key);
void splitChild(Node *node, int i);

void insert(Node **root, int key) {
   if (*root == NULL) {
      *root = createNode(key, NULL);
   } else {
      Node *r = *root;
      if (r->n == 3) {
         Node *s = createNode(0, r);
         s->isLeaf = 0;
```

```
            splitChild(s, 0);
            insertNonFull(s, key);
            *root = s;
        } else {
            insertNonFull(r, key);
        }
    }
}

void insertNonFull(Node *node, int key) {
    int i = node->n - 1;
    if (node->isLeaf) {
        while (i >= 0 && key < node->keys[i]) {
            node->keys[i + 1] = node->keys[i];
            i--;
        }
        node->keys[i + 1] = key;
        node->n++;
    } else {
        while (i >= 0 && key < node->keys[i]) {
            i--;
        }
        i++;
        if (node->children[i]->n == 3) {
            splitChild(node, i);
            if (key > node->keys[i]) {
                i++;
            }
        }
        insertNonFull(node->children[i], key);
    }
```

```
}

void splitChild(Node *node, int i) {
    Node *y = node->children[i];
    Node *z = createNode(y->keys[2], NULL);
    node->children[i + 1] = z;
    node->keys[i] = y->keys[1];
    node->n++;
    y->n = 1;
    z->isLeaf = y->isLeaf;
    if (!y->isLeaf) {
        for (int j = 0; j < 2; j++) {
            z->children[j] = y->children[j + 2];
        }
    }
}

int search(Node *node, int key) {
    int i = 0;
    while (i < node->n && key > node->keys[i]) {
        i++;
    }
    if (i < node->n && key == node->keys[i]) {
        return 1;
    }
    if (node->isLeaf) {
        return 0;
    }
    return search(node->children[i], key);
}
```

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

int main() {
    Node *root = NULL;
    int keys[] = {10, 20, 5, 6, 12, 30, 7, 17};
    int n = sizeof(keys) / sizeof(keys[0]);

    for (int i = 0; i < n; i++) {
        insert(&root, keys[i]);
    }

    printf("2-3 Tree structure:\n");
    printTree(root, 0);

    int searchKeys[] = {6, 15};
    for (int i = 0; i < sizeof(searchKeys) / sizeof(searchKeys[0]); i++) {
```

```
    printf("Key %d %s found in the tree.\n", searchKeys[i], search(root, searchKeys[i]) ? "is" : "is not");
  }

  return 0;
}
```

Output:

```
2-3 Tree structure:
10
 5 6
 12 17 20
  7
  30
Key 6 is found in the tree.
Key 15 is not found in the tree.
```

3. 2-3-4 TREE

Program:

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

#define MAX_KEYS 3
#define MIN_KEYS 1

typedef struct Node {
  int n;              // Number of keys in this node
  int keys[MAX_KEYS];   // Array to store keys
  struct Node *children[MAX_KEYS + 1]; // Array of child pointers
  int isLeaf;         // Is true when node is a leaf. Otherwise false
} Node;

Node *createNode(int key, Node *child) {
  Node *newNode = (Node *)malloc(sizeof(Node));
```

```
    newNode->keys[0] = key;
    newNode->n = 1;
    newNode->children[0] = NULL;
    newNode->children[1] = child;
    newNode->isLeaf = 1;
    return newNode;
}

void insertNonFull(Node *node, int key);
void splitChild(Node *node, int i);

void insert(Node **root, int key) {
    if (*root == NULL) {
        *root = createNode(key, NULL);
    } else {
        Node *r = *root;
        if (r->n == MAX_KEYS) {
            Node *s = createNode(0, r);
            s->isLeaf = 0;
            splitChild(s, 0);
            insertNonFull(s, key);
            *root = s;
        } else {
            insertNonFull(r, key);
        }
    }
}

void insertNonFull(Node *node, int key) {
    int i = node->n - 1;
    if (node->isLeaf) {
```

```
        while (i >= 0 && key < node->keys[i]) {
            node->keys[i + 1] = node->keys[i];
            i--;
        }
        node->keys[i + 1] = key;
        node->n++;
    } else {
        while (i >= 0 && key < node->keys[i]) {
            i--;
        }
        i++;
        if (node->children[i]->n == MAX_KEYS) {
            splitChild(node, i);
            if (key > node->keys[i]) {
                i++;
            }
        }
        insertNonFull(node->children[i], key);
    }
}

void splitChild(Node *node, int i) {
    Node *y = node->children[i];
    Node *z = createNode(y->keys[2], NULL);
    z->isLeaf = y->isLeaf;
    z->n = 1;
    node->children[i + 1] = z;
    for (int j = node->n; j >= i + 1; j--) {
        node->children[j + 1] = node->children[j];
    }
    for (int j = node->n - 1; j >= i; j--) {
```

```
        node->keys[j + 1] = node->keys[j];
    }
    node->keys[i] = y->keys[1];
    node->n++;
    y->n = 1;
    if (!y->isLeaf) {
        for (int j = 0; j < 2; j++) {
            z->children[j] = y->children[j + 2];
        }
    }
}

int search(Node *node, int key) {
    int i = 0;
    while (i < node->n && key > node->keys[i]) {
        i++;
    }
    if (i < node->n && key == node->keys[i]) {
        return 1;
    }
    if (node->isLeaf) {
        return 0;
    }
    return search(node->children[i], key);
}

void printTree(Node *node, int level) {
    if (node != NULL) {
        for (int i = 0; i < level; i++) {
            printf("  ");
        }
```

```
        for (int i = 0; i < node->n; i++) {
            printf("%d ", node->keys[i]);
        }
        printf("\n");
        if (!node->isLeaf) {
            for (int i = 0; i <= node->n; i++) {
                printTree(node->children[i], level + 1);
            }
        }
    }
}

int main() {
    Node *root = NULL;
    int keys[] = {10, 20, 5, 6, 12, 30, 7, 17};
    int n = sizeof(keys) / sizeof(keys[0]);

    for (int i = 0; i < n; i++) {
        insert(&root, keys[i]);
    }

    printf("2-3-4 Tree structure:\n");
    printTree(root, 0);

    int searchKeys[] = {6, 15};
    for (int i = 0; i < sizeof(searchKeys) / sizeof(searchKeys[0]); i++) {
        printf("Key %d %s found in the tree.\n", searchKeys[i], search(root, searchKeys[i]) ? "is" : "is
not");
    }

    return 0;
```

}

Output:

Segmentation fault