

1. Write a c program for TRIE.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #define ALPHABET_SIZE 26
5 struct TrieNode {
6     struct TrieNode* children[ALPHABET_SIZE];
7     int is_end_of_word;
8 };
9 struct TrieNode* createNode() {
10     struct TrieNode* node = (struct TrieNode*)malloc(sizeof(struct
        TrieNode));
11     for (int i = 0; i < ALPHABET_SIZE; i++) {
12         node->children[i] = NULL;
13     }
14     node->is_end_of_word = 0;
15     return node;
16 }
17 void insert(struct TrieNode* root, const char* word) {
18     struct TrieNode* current = root;
19     while (*word) {
20         int index = *word - 'a';
21         if (!current->children[index]) {
22             current->children[index] = createNode();
23         }
24         current = current->children[index];
25         word++;
26     }
27     current->is_end_of_word = 1;
28 }
29 int search(struct TrieNode* root, const char* word) {
```

```
/tmp/SRW8snYPTX.o
Searching for 'tea': Not Found
Searching for 'teabag': Found!
Searching for 'teacan': Found!
Searching for 'hi': Found!
Searching for 'hey': Not Found

=== Code Execution Successful ===
```

```
29 int search(struct TrieNode* root, const char* word) {
30     struct TrieNode* current = root;
31     while (*word) {
32         int index = *word - 'a';
33         if (!current->children[index]) {
34             return 0;
35         }
36         current = current->children[index];
37         word++;
38     }
39     return current->is_end_of_word;
40 }
41 int main() {
42     struct TrieNode* root = createNode();
43     insert(root, "hello");
44     insert(root, "hi");
45     insert(root, "teabag");
46     insert(root, "teacan");
47     printf("Searching for 'tea': %s\n", search(root, "tea") ? "Found!" :
        "Not Found");
48     printf("Searching for 'teabag': %s\n", search(root, "teabag") ?
        "Found!" : "Not Found");
49     printf("Searching for 'teacan': %s\n", search(root, "teacan") ?
        "Found!" : "Not Found");
50     printf("Searching for 'hi': %s\n", search(root, "hi") ? "Found!" :
        "Not Found");
51     printf("Searching for 'hey': %s\n", search(root, "hey") ? "Found!" :
        "Not Found");
52     return 0;
53 }
```

```
/tmp/SRW8snYPTX.o
Searching for 'tea': Not Found
Searching for 'teabag': Found!
Searching for 'teacan': Found!
Searching for 'hi': Found!
Searching for 'hey': Not Found

=== Code Execution Successful ===
```

2. Write a c program for B TREE (2-3).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 typedef struct BTreeNode {
4     int keys[2];
5     struct BTreeNode *children[3];
6     int numKeys;
7     int isLeaf;
8 } BTreeNode;
9 typedef struct BTree {
10     BTreeNode* root;
11 } BTree;
12 BTreeNode* createNode(int isLeaf) {
13     BTreeNode* newNode = (BTreeNode*)malloc(sizeof(BTreeNode));
14     newNode->numKeys = 0;
15     newNode->isLeaf = isLeaf;
16     for (int i = 0; i < 3; i++) {
17         newNode->children[i] = NULL;
18     }
19     return newNode;
20 }
21 BTree* createTree() {
22     BTree* newTree = (BTree*)malloc(sizeof(BTree));
23     newTree->root = createNode(1);
24     return newTree;
25 }
26 void splitChild(BTreeNode* parent, int i, BTreeNode* child) {
27     BTreeNode* newChild = createNode(child->isLeaf);
28     newChild->numKeys = 1;
29     newChild->keys[0] = child->keys[1];
30 }
31 if (!child->isLeaf) {
32     newChild->children[0] = child->children[1];
33     newChild->children[1] = child->children[2];
34 }
35
```

```
~/tmp/KM1nMN0aJ9.o
B-Tree:
5
6
6
7
10
6
7
10
10
12
12
17
20
10
20
20
30

=== Code Execution Successful ===
```

```
36 child->numKeys = 1;
37
38 for (int j = parent->numKeys; j >= i + 1; j--) {
39     parent->children[j + 1] = parent->children[j];
40 }
41
42 parent->children[i + 1] = newChild;
43
44 for (int j = parent->numKeys - 1; j >= i; j--) {
45     parent->keys[j + 1] = parent->keys[j];
46 }
47
48 parent->keys[i] = child->keys[1];
49 parent->numKeys++;
50 }
51 void insertNonFull(BTreeNode* node, int key) {
52     int i = node->numKeys - 1;
53
54     if (node->isLeaf) {
55         while (i >= 0 && key < node->keys[i]) {
56             node->keys[i + 1] = node->keys[i];
57             i--;
58         }
59         node->keys[i + 1] = key;
60         node->numKeys++;
61     } else {
62         while (i >= 0 && key < node->keys[i]) {
63             i--;
64         }
65         i++;
66         if (node->children[i]->numKeys == 2) {
67             splitChild(node, i, node->children[i]);
68             if (key > node->keys[i]) {
```

```
~/tmp/KM1nMN0aJ9.o
B-Tree:
5
6
6
7
10
6
7
10
10
12
12
17
20
10
20
20
30

=== Code Execution Successful ===
```

```

69         i++;
70     }
71 }
72 insertNonFull(node->children[i], key);
73 }
74 }
75 void insert(BTree* tree, int key) {
76     BTreeNode* root = tree->root;
77     if (root->numKeys == 2) {
78         BTreeNode* newRoot = createNode(0);
79         newRoot->children[0] = root;
80         splitChild(newRoot, 0, root);
81         tree->root = newRoot;
82         insertNonFull(newRoot, key);
83     } else {
84         insertNonFull(root, key);
85     }
86 }
87 void printTree(BTreeNode* node, int level) {
88     if (node != NULL) {
89         for (int i = 0; i < node->numKeys; i++) {
90             printTree(node->children[i], level + 1);
91             for (int j = 0; j < level; j++) {
92                 printf(" ");
93             }
94             printf("%d\n", node->keys[i]);
95         }
96         printTree(node->children[node->numKeys], level + 1);
97     }
98 }
99 int main() {
100     BTree* tree = createTree();
101     insert(tree, 10);
102     insert(tree, 20);
103     insert(tree, 5);

```

```

/tmp/KM1nMN0aJ9.o
B-Tree:
5
6
6
7
10
6
7
10
10
12
12
17
20
10
20
20
30

=== Code Execution Successful ===

```

```

main.c
80     splitChild(newRoot, 0, root);
81     tree->root = newRoot;
82     insertNonFull(newRoot, key);
83 } else {
84     insertNonFull(root, key);
85 }
86 }
87 void printTree(BTreeNode* node, int level) {
88     if (node != NULL) {
89         for (int i = 0; i < node->numKeys; i++) {
90             printTree(node->children[i], level + 1);
91             for (int j = 0; j < level; j++) {
92                 printf(" ");
93             }
94             printf("%d\n", node->keys[i]);
95         }
96         printTree(node->children[node->numKeys], level + 1);
97     }
98 }
99 int main() {
100     BTree* tree = createTree();
101     insert(tree, 10);
102     insert(tree, 20);
103     insert(tree, 5);
104     insert(tree, 6);
105     insert(tree, 12);
106     insert(tree, 30);
107     insert(tree, 7);
108     insert(tree, 17);
109     printf("B-Tree:\n");
110     printTree(tree->root, 0);
111     return 0;
112 }
113

```

```




/tmp/KM1nMN0aJ9.o
B-Tree:
5
6
6
7
10
6
7
10
10
12
12
17
20
10
20
20
30

=== Code Execution Successful ===

```

3. Write a c program for B TREE (2-3-4).

main.c



Run

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 typedef struct BTreeNode {
4     int keys[3];
5     struct BTreeNode *children[4];
6     int numKeys;
7     int isLeaf;
8 } BTreeNode;
9
10 typedef struct BTree {
11     BTreeNode* root;
12 } BTree;
13 BTreeNode* createNode(int isLeaf) {
14     BTreeNode* newNode = (BTreeNode*)malloc(sizeof(BTreeNode));
15     newNode->numKeys = 0;
16     newNode->isLeaf = isLeaf;
17     for (int i = 0; i < 4; i++) {
18         newNode->children[i] = NULL;
19     }
20     return newNode;
21 }
22 BTree* createTree() {
23     BTree* newTree = (BTree*)malloc(sizeof(BTree));
24     newTree->root = createNode(1);
25     return newTree;
26 }
27 void splitChild(BTreeNode* parent, int i, BTreeNode* child) {
28     BTreeNode* newChild = createNode(child->isLeaf);
29     newChild->numKeys = 1;
30     newChild->keys[0] = child->keys[2];
31
32     if (!child->isLeaf) {
33         newChild->children[0] = child->children[2];
34         newChild->children[1] = child->children[3];
35     }
36 }
```

Output

Clear

/tmp/wbec1dyZXX.o

B-Tree:

5

6

7

10

12

17

20

30

=== Code Execution Successful ===

main.c

Run

Output

Clear

```
36 child->numKeys = 1;
37
38
39 for (int j = parent->numKeys; j >= i + 1; j--) {
40     parent->children[j + 1] = parent->children[j];
41 }
42
43 parent->children[i + 1] = newChild;
44
45 for (int j = parent->numKeys - 1; j >= i; j--) {
46     parent->keys[j + 1] = parent->keys[j];
47 }
48
49 parent->keys[i] = child->keys[1];
50 parent->numKeys++;
51 }
52 void insertNonFull(BTreeNode* node, int key) {
53     int i = node->numKeys - 1;
54
55     if (node->isLeaf) {
56         while (i >= 0 && key < node->keys[i]) {
57             node->keys[i + 1] = node->keys[i];
58             i--;
59         }
60         node->keys[i + 1] = key;
61         node->numKeys++;
62     } else {
63         while (i >= 0 && key < node->keys[i]) {
64             i--;
65         }
66         i++;
67         if (node->children[i]->numKeys == 3) {
68             splitChild(node, i, node->children[i]);
69             if (key > node->keys[i]) {
70                 i++;
```

```
main.c
68     splitChild(node, i, node->children[i]);
69     if (key > node->keys[i]) {
70         i++;
71     }
72 }
73 insertNonFull(node->children[i], key);
74 }
75 }
76 void insert(BTree* tree, int key) {
77     BTreeNode* root = tree->root;
78     if (root->numKeys == 3) {
79         BTreeNode* newRoot = createNode(0);
80         newRoot->children[0] = root;
81         splitChild(newRoot, 0, root);
82         tree->root = newRoot;
83         insertNonFull(newRoot, key);
84     } else {
85         insertNonFull(root, key);
86     }
87 }
88 void printTree(BTreeNode* node, int level) {
89     if (node != NULL) {
90         for (int i = 0; i < node->numKeys; i++) {
91             printTree(node->children[i], level + 1);
92             for (int j = 0; j < level; j++) {
93                 printf(" ");
94             }
95             printf("%d\n", node->keys[i]);
96         }
97         printTree(node->children[node->numKeys], level + 1);
98     }
99 }
100 int main() {
101     BTree* tree = createTree();
102     insert(tree, 10);
```

```
Output
/tmp/wbec1dyZXX.o
B-Tree:
5
6
7
10
12
17
20
30

=== Code Execution Successful ===
```

```
main.c
81     splitChild(newRoot, 0, root);
82     tree->root = newRoot;
83     insertNonFull(newRoot, key);
84 } else {
85     insertNonFull(root, key);
86 }
87 }
88 void printTree(BTreeNode* node, int level) {
89     if (node != NULL) {
90         for (int i = 0; i < node->numKeys; i++) {
91             printTree(node->children[i], level + 1);
92             for (int j = 0; j < level; j++) {
93                 printf(" ");
94             }
95             printf("%d\n", node->keys[i]);
96         }
97         printTree(node->children[node->numKeys], level + 1);
98     }
99 }
100 int main() {
101     BTree* tree = createTree();
102     insert(tree, 10);
103     insert(tree, 20);
104     insert(tree, 5);
105     insert(tree, 6);
106     insert(tree, 12);
107     insert(tree, 30);
108     insert(tree, 7);
109     insert(tree, 17);
110     printf("B-Tree:\n");
111     printTree(tree->root, 0);
112     return 0;
113 }
114 }
```

```
Output
/tmp/wbec1dyZXX.o
B-Tree:
5
6
7
10
12
17
20
30

=== Code Execution Successful ===
```

4. Write a c program for B TREE (2-3-4-5).

```
main.c  Run  Output  Clear

1 #include <stdio.h>
2 #include <stdlib.h>
3 #define MAX_KEYS 4
4 #define MAX_CHILDREN 5
5 typedef struct BTreeNode {
6     int keys[MAX_KEYS];
7     struct BTreeNode *children[MAX_CHILDREN];
8     int numKeys;
9     int isLeaf;
10 } BTreeNode;
11 typedef struct BTree {
12     BTreeNode* root;
13 } BTree;
14 BTreeNode* createNode(int isLeaf) {
15     BTreeNode* newNode = (BTreeNode*)malloc(sizeof(BTreeNode));
16     newNode->numKeys = 0;
17     newNode->isLeaf = isLeaf;
18     for (int i = 0; i < MAX_CHILDREN; i++) {
19         newNode->children[i] = NULL;
20     }
21     return newNode;
22 }
23 BTree* createTree() {
24     BTree* newTree = (BTree*)malloc(sizeof(BTree));
25     newTree->root = createNode(1);
26     return newTree;
27 }
28 void splitChild(BTreeNode* parent, int i, BTreeNode* child) {
29     BTreeNode* newChild = createNode(child->isLeaf);
30     newChild->numKeys = 2;
31     newChild->keys[0] = child->keys[2];
32     newChild->keys[1] = child->keys[3];
33 }
34 if (!child->isLeaf) {
35     newChild->children[0] = child->children[3];
    ...
}
```

```
/tmp/2k045Cdw8U.o
B-Tree:
5
6
7
10
12
17
20
20
30

=== Code Execution Successful ===
```

```
main.c  Run  Output  Clear

35     newChild->children[0] = child->children[3];
36     newChild->children[1] = child->children[4];
37 }
38 child->numKeys = 2;
39 for (int j = parent->numKeys; j >= i + 1; j--) {
40     parent->children[j + 1] = parent->children[j];
41 }
42 parent->children[i + 1] = newChild;
43 }
44 for (int j = parent->numKeys - 1; j >= i; j--) {
45     parent->keys[j + 1] = parent->keys[j];
46 }
47 parent->keys[i] = child->keys[2];
48 parent->numKeys++;
49 }
50 }
51 void insertNonFull(BTreeNode* node, int key) {
52     int i = node->numKeys - 1;
53     if (node->isLeaf) {
54         while (i >= 0 && key < node->keys[i]) {
55             node->keys[i + 1] = node->keys[i];
56             i--;
57         }
58         node->keys[i + 1] = key;
59         node->numKeys++;
60     } else {
61         while (i >= 0 && key < node->keys[i]) {
62             i--;
63         }
64         i++;
65         if (node->children[i]->numKeys == MAX_KEYS) {
66             splitChild(node, i, node->children[i]);
67             if (key > node->keys[i]) {
68                 i++;
69             }
    ...
}
```

```
/tmp/2k045Cdw8U.o
B-Tree:
5
6
7
10
12
17
20
20
30

=== Code Execution Successful ===
```

```
main.c
70     }
71     insertNonFull(node->children[i], key);
72 }
73 }
74 void insert(BTree* tree, int key) {
75     BTreeNode* root = tree->root;
76     if (root->numKeys == MAX_KEYS) {
77         BTreeNode* newRoot = createNode(0);
78         newRoot->children[0] = root;
79         splitChild(newRoot, 0, root);
80         tree->root = newRoot;
81         insertNonFull(newRoot, key);
82     } else {
83         insertNonFull(root, key);
84     }
85 }
86 void printTree(BTreeNode* node, int level) {
87     if (node != NULL) {
88         for (int i = 0; i < node->numKeys; i++) {
89             printTree(node->children[i], level + 1);
90             for (int j = 0; j < level; j++) {
91                 printf(" ");
92             }
93             printf("%d\n", node->keys[i]);
94         }
95         printTree(node->children[node->numKeys], level + 1);
96     }
97 }
98 int main() {
99     BTree* tree = createTree();
100     insert(tree, 10);
101     insert(tree, 20);
102     insert(tree, 5);
103     insert(tree, 6);
104     //insert(tree, 12);
105     //insert(tree, 30);
106     //insert(tree, 7);
107     //insert(tree, 17);
108     printf("B-Tree:\n");
109     printTree(tree->root, 0);
110     return 0;
111 }
112 }
```

```
Output
/tmp/2k04SCdw8U.o
B-Tree:
5
6
7
10
12
17
20
30

=== Code Execution Successful ===
```

```
main.c
79     splitChild(newRoot, 0, root);
80     tree->root = newRoot;
81     insertNonFull(newRoot, key);
82 } else {
83     insertNonFull(root, key);
84 }
85 }
86 void printTree(BTreeNode* node, int level) {
87     if (node != NULL) {
88         for (int i = 0; i < node->numKeys; i++) {
89             printTree(node->children[i], level + 1);
90             for (int j = 0; j < level; j++) {
91                 printf(" ");
92             }
93             printf("%d\n", node->keys[i]);
94         }
95         printTree(node->children[node->numKeys], level + 1);
96     }
97 }
98 int main() {
99     BTree* tree = createTree();
100     insert(tree, 10);
101     insert(tree, 20);
102     insert(tree, 5);
103     insert(tree, 6);
104     insert(tree, 12);
105     insert(tree, 30);
106     insert(tree, 7);
107     insert(tree, 17);
108     printf("B-Tree:\n");
109     printTree(tree->root, 0);
110     return 0;
111 }
112 }
```

```
Output
/tmp/2k04SCdw8U.o
B-Tree:
5
6
7
10
12
17
20
30

=== Code Execution Successful ===
```