

#### **practical - 4**

**aim : Create a B-tree of order 4**

**code :**

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

#define MAX 3
#define MIN 2

struct BTreeNode {
    int val[MAX + 1], count;
    struct BTreeNode *link[MAX + 1];
};

struct BTreeNode *root;

// Create a node
struct BTreeNode *createNode(int val, struct BTreeNode *child) {
    struct BTreeNode *newNode;
    newNode = (struct BTreeNode *)malloc(sizeof(struct BTreeNode));
    newNode->val[1] = val;
    newNode->count = 1;
    newNode->link[0] = root;
    newNode->link[1] = child;
    return newNode;
}

// Insert node
void insertNode(int val, int pos, struct BTreeNode *node,
    struct BTreeNode *child) {
    int j = node->count;
    while (j > pos) {
        node->val[j + 1] = node->val[j];
        node->link[j + 1] = node->link[j];
        j--;
    }
    node->val[j + 1] = val;
    node->link[j + 1] = child;
    node->count++;
}

// Split node
void splitNode(int val, int *pval, int pos, struct BTreeNode *node,
    struct BTreeNode *child, struct BTreeNode **newNode) {
    int median, j;
```

```

if (pos > MIN)
    median = MIN + 1;
else
    median = MIN;

*newNode = (struct BTreeNode *)malloc(sizeof(struct BTreeNode));
j = median + 1;
while (j <= MAX) {
    (*newNode)->val[j - median] = node->val[j];
    (*newNode)->link[j - median] = node->link[j];
    j++;
}
node->count = median;
(*newNode)->count = MAX - median;

if (pos <= MIN) {
    insertNode(val, pos, node, child);
} else {
    insertNode(val, pos - median, *newNode, child);
}
*pval = node->val[node->count];
(*newNode)->link[0] = node->link[node->count];
node->count--;
}

// Set the value
int setValue(int val, int *pval,
             struct BTreeNode *node, struct BTreeNode **child) {
    int pos;
    if (!node) {
        *pval = val;
        *child = NULL;
        return 1;
    }

    if (val < node->val[1]) {
        pos = 0;
    } else {
        for (pos = node->count;
             (val < node->val[pos] && pos > 1); pos--)
            ;
        if (val == node->val[pos]) {
            printf("Duplicates are not permitted\n");
            return 0;
        }
    }
}

```

```

}
if (setValue(val, pval, node->link[pos], child)) {
    if (node->count < MAX) {
        insertNode(*pval, pos, node, *child);
    } else {
        splitNode(*pval, pval, pos, node, *child, child);
        return 1;
    }
}
return 0;
}

```

// Insert the value

```

void insert(int val) {
    int flag, i;
    struct BTreeNode *child;

    flag = setValue(val, &i, root, &child);
    if (flag)
        root = createNode(i, child);
}

```

// Search node

```

void search(int val, int *pos, struct BTreeNode *myNode) {
    if (!myNode) {
        return;
    }

    if (val < myNode->val[1]) {
        *pos = 0;
    } else {
        for (*pos = myNode->count;
            (val < myNode->val[*pos] && *pos > 1); (*pos)--);
        ;
        if (val == myNode->val[*pos]) {
            printf("%d is found", val);
            return;
        }
    }
    search(val, pos, myNode->link[*pos]);

    return;
}

```

// Traverse then nodes

```

void traversal(struct BTreeNode *myNode) {

```

```
int i;
if (myNode) {
    for (i = 0; i < myNode->count; i++) {
        traversal(myNode->link[i]);
        printf("%d ", myNode->val[i + 1]);
    }
    traversal(myNode->link[i]);
}
}
```

```
int main() {
    int val, ch;

    insert(8);
    insert(9);
    insert(10);
    insert(11);
    insert(15);
    insert(16);
    insert(17);
    insert(18);
    insert(20);
    insert(23);

    traversal(root);

    printf("\n");
    search(11, &ch, root);
}
```

**output :**

8 9 10 11 15 16 17 18 20 23

11 is found