**UNIVERSITE AKLI MOHAND OULHADJ – BOUIRA –**

**Faculté des sciences et des sciences appliquées**

**Département de Informatique**

# TP SDA ( B-TREE REPORT )

**<u>Realized by:</u>**

**YAHOUI SOHEIB**

**NCERBEY SARA**

**KAHLOUCHE NOURELHOUDA**

## 1) Introduction:

A **B-tree** is a self-balancing tree data structure that maintains sorted data and allows searches, sequential access, insertions, and deletions in logarithmic time. The B-tree generalizes the binary search tree, allowing for nodes with more than two children.

**IN our project** we will implement the b-tree of order 2 with a “ c “ code source . our work contains these main functions :

1-insertion into the tree

2- deleting

3-searching

4- print the value of all the tree (traversal the tree).

## 2) Main functions :

### 2-1) the insertion:

**Max = 5 , is the number of maximum pointers in one node which means 4 values .**

**Min = 3 , is the number of minimum pointers in one node which means 2 values**

**“order 2”**

**We can allow the user to choose the max and min ( a b-tree with customizable order )**

```
#define MAX 5
  #define MIN 3

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

  struct btreeNode *root;

  /* creating new 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;
  }
```

**this function it allows us to create a new node , then we have to know where to place this value into the node (her position in the node ) . for now , we suppose that we have no overflow in the node (we don’t need to split it ).**

```
void addValToNode(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++;
}
```

**In case of we need to split the node which means we have passed the maximum pointers allowed ,for this we have created this function**

```c
/* split the 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) {
                addValToNode(val, pos, node, child);
        } else {
                addValToNode(val, pos - median, *newNode, child);
        }
        *pval = node->val[node->count];
        (*newNode)->link[0] = node->link[node->count];
        node->count--;
}
```

**This function links the child node with parent node and initialize her variables ..**

**But To insert a value into a node we may split or we may not**
**In order to know we will use this function , which return 1 if we have to split and split the node , otherwise it will use precedent function and add the value . the function is this below :**

```c
int setValueInNode(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 not allowed\n");
                  return 0;
            }
      }
      if (setValueInNode(val, pval, node->link[pos], child)) {
            if (node->count < MAX) {
                  addValToNode(*pval, pos, node, *child);
            } else {
                  splitNode(*pval, pval, pos, node, *child, child);
                  return 1;
            }
      }
      return 0;
}
```

**The duplicates are not allowed .**

**Finally , the insert function we use all these function that seen before ,**

**It will run the function (set value in node) if it returns 1 than it call to the function (create new node)**

```c
/* insert val in B-Tree */
void insertion(int val) {
    int flag, i;
    struct btreeNode *child;

    flag = setValueInNode(val, &i, root, &child);
    if (flag)
        root = createnewNode(i, child);
}
```

## 2-2) deleting a value :

**the problem when we remove a value from a node that we will have a number of keys inferior than 2 which is not allowed in our b –tree (order 2)**

**first we have supposed a case when the number of keys is bigger than 2 and we created a function which remove a value from a node and rearrange the values**

```
/* removes the value from the given node and rearrange values */
void removeVal(struct btreeNode *myNode, int pos) {
        int i = pos + 1;
        while (i <= myNode->count) {
                myNode->val[i - 1] = myNode->val[i];
                myNode->link[i - 1] = myNode->link[i];
                i++;
        }
        myNode->count--;
}
```

**There is cases (underflow ) when we remove we need to down parent node with the left or right children therefore we have created these 2 function functions**

```
/* shifts value from parent to right child */
void doRightShift(struct btreeNode *myNode, int pos) {
        struct btreeNode *x = myNode->link[pos];
        int j = x->count;

        while (j > 0) {
                x->val[j + 1] = x->val[j];
                x->link[j + 1] = x->link[j];
        }
        x->val[1] = myNode->val[pos];
        x->link[1] = x->link[0];
        x->count++;

        x = myNode->link[pos - 1];
        myNode->val[pos] = x->val[x->count];
        myNode->link[pos] = x->link[x->count];
        x->count--;
        return;
```

```c
/* shifts value from parent to left child */
void doLeftShift(struct btreeNode *myNode, int pos) {
        int j = 1;
        struct btreeNode *x = myNode->link[pos - 1];

        x->count++;
        x->val[x->count] = myNode->val[pos];
        x->link[x->count] = myNode->link[pos]->link[0];

        x = myNode->link[pos];
        myNode->val[pos] = x->val[1];
        x->link[0] = x->link[1];
        x->count--;

        while (j <= x->count) {
                x->val[j] = x->val[j + 1];
                x->link[j] = x->link[j + 1];
                j++;
        }
        return;
}
```

**And also in other cases we need to merge the nodes with each other**

```c
/* merge nodes */
void mergeNodes(struct btreeNode *myNode, int pos) {
        int j = 1;
        struct btreeNode *x1 = myNode->link[pos], *x2 = myNode->link[pos - 1];

        x2->count++;
        x2->val[x2->count] = myNode->val[pos];
        x2->link[x2->count] = myNode->link[0];

        while (j <= x1->count) {
                x2->count++;
                x2->val[x2->count] = x1->val[j];
                x2->link[x2->count] = x1->link[j];
                j++;
        }

        j = pos;
        while (j < myNode->count) {
                myNode->val[j] = myNode->val[j + 1];
                myNode->link[j] = myNode->link[j + 1];
                j++;
        }
        myNode->count--;
        free(x1);
}
```

**This function uses all the precedent functions , in order to delete a value from a node it test which case we are and adjust the result , we test the number of keys in node (count) with min and max**

```
/* delete val from the node */
int delValFromNode(int val, struct btreeNode *myNode) {
        int pos, flag = 0;
        if (myNode) {
                if (val < myNode->val[1]) {
                        pos = 0;
                        flag = 0;
                } else {
                        for (pos = myNode->count;
                                (val < myNode->val[pos] && pos > 1); pos--);
                         if (val == myNode->val[pos]) {
                                flag = 1;
                        } else {
                                flag = 0;
                        }
                }
                if (flag) {
                        if (myNode->link[pos - 1]) {
                                copySuccessor(myNode, pos);
                                flag = delValFromNode(myNode->val[pos], myNode->link[p
                                if (flag == 0) {
                                        printf("Given data is not present in B-Tree\n"
                                }
                        } else {
                                removeVal(myNode, pos);
                        }
                } else {
                        flag = delValFromNode(val, myNode->link[pos]);
                }
                if (myNode->link[pos]) {
                        if (myNode->link[pos]->count < MIN)
                                adjustNode(myNode, pos);
```

## 2-3) search a value in a b-tree :

**This function print the value if it exists . else , it will return nothing**

```
/* search val in B-Tree */
void searching(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("Given data %d is present in B-Tree", val);
                        return;
                }
        }
        searching(val, pos, myNode->link[*pos]);
        return;
}
```

## 2-4) traversal the tree :

**In order to print all the values of the tree We generate a function which traverse all the tree .**

```
/* B-Tree Traversal */
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]);
        }
}
```

## 3) test :

**In order to test our functions we run them into a program in the main function and give the user the choice to choose what to do . like this :**

```c
int main() {
    int val, ch;
    while (1) {
        printf("1. Insertion\t2. Deletion\n");
        printf("3. Searching\t4. Traversal\n");
        printf("5. Exit\nEnter your choice:");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                printf("Enter your input:");
                scanf("%d", &val);
                insertion(val);
                break;
            case 2:
                printf("Enter the element to delete:");
                scanf("%d", &val);
                deletion(val, root);
                break;
            case 3:
                printf("Enter the element to search:");
                scanf("%d", &val);
                searching(val, &ch, root);
                break;
            case 4:
                traversal(root);
                break;
            case 5:
                exit(0);
            default:
                printf("U have entered wrong option!!\n");
                break;
        }
```

**A simple demonstration :**

```
1. Insertion 2. Deletion
3. Searching 4. Traversal
5. Exit
Enter your choice:1
Enter your input:70

1. Insertion 2. Deletion
3. Searching 4. Traversal
5. Exit
Enter your choice:1
Enter your input:17

1. Insertion 2. Deletion
3. Searching 4. Traversal
5. Exit
Enter your choice:1
Enter your input:67

1. Insertion 2. Deletion
3. Searching 4. Traversal
5. Exit
Enter your choice:1
Enter your input:89

1. Insertion 2. Deletion
3. Searching 4. Traversal
5. Exit
Enter your choice:4
17 67 70 89
```

```
1. Insertion 2. Deletion
3. Searching 4. Traversal
5. Exit
Enter your choice:3
Enter the element to search:70
Given data 70 is present in B-Tree

1. Insertion 2. Deletion
3. Searching 4. Traversal
5. Exit
Enter your choice:2
Enter the element to delete:17

1. Insertion 2. Deletion
3. Searching 4. Traversal
5. Exit
Enter your choice:4
67 70 89

1. Insertion 2. Deletion
3. Searching 4. Traversal
5. Exit
Enter your choice:5
```

**4) conclusion : we have implemented the b tree in c language , b tree of order 2 , we could simply make it customizable by letting the users choosing the values of max and min .**