ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE

المدرسة الوطنية العليا للإعلام الآلي
École nationale Supérieure d'Informatique

Practical Work Report - File Structures and Data Structures
Hierarchical Index with T1/T2 Trees

By:

- Azzouz Zaki Group 12 Section C
- Amara Khaled Walid Group 12 Section C

1 File Structure Declaration

1.1 Record Structure

```
Structure t_rec
        Key: Integer        //Record key
        blkAddr: Integer    //Block address
        recAddr: Integer    //Record address in block
End Structure
```

1.2 Block Structure

```
Structure t_block
        Tab[MAXTAB]: Array of t rec     //Records array
        Nrec: Integer                   //Number of records in block
End Structure
```

1.3 File Header Structure

```
Structure t_header
        Nblocks: Integer    //Number of blocks in file
End Structure
```

1.4 TOF File Structure

```
Structure t_TOF
        f: File Pointer     //Pointer to physical file
        h: t header         //Header in main memory
End Structure
```

1.5 T1 Tree Structure

```
Structure t_T1
        V1: Integer         //Minimum value
        V2: Integer         //Maximum value
        LC: Pointer to t T1 //Left child
        RC: Pointer to t T1 //Right child
        R: Pointer to t T2  //Root of T2 tree
End Structure
```

1.6 T2 Tree Structure

```
Structure t_T2
        Key: Integer        //Key
        BlkNum: Integer     //Block number
        RecNum: Integer     //Record number
        LC: Pointer to t T2 //Left child
        RC: Pointer to t T2 //Right child
End Structure
```

1.7 B-Tree Node Structure

```
Structure B_Tree
        Key[4]: Array of Integer                    //Maximum 4 keys
        child[5]: Array of Pointer to NodeBtree     //Maximum 5 children
        parent: Pointer to NodeBtree                //Pointer to parent node
        degree: Integer                             //Number of keys + 1
End Structure
```

2 Index Loading Module into RAM

```
procedure LoadTreeFromFile(f: Pointer to t_TOF, var Root: Pointer to t_T1)
        int i
        int j
        t_block Buf
        Create empty tree (Root ← NULL)
        nBlocks ← getHeader(f, ”Nblocks”)
        i ← 0
        while i < nBlocks do
                Read block i+1 into Buf
                for j = 0 to Buf.Nrec - 1 do
                        Insert into tree (Root, Buf.Tab[j].Key, Buf.Tab[j].blkAddr,
                        Buf.Tab[j].recAddr)
                end for
                i ← i + 1
        end while
end procedure
```

3 Index Saving Module

3.1 Main Procedure

```
procedure SaveTreeToFile(f: Pointer to t_TOF, Root: Pointer to t_T1)
        int Nblocks ← 0
        ProcessT1(f, Root, Nblocks)
        setHeader(f, ”Nblocks”, Nblocks)
end procedure
```

3.2 T1 Tree Processing

```
procedure ProcessT1(f: Pointer to t_TOF, Node: Pointer to t_T1,var Nblocks: Integer)
        if Node  != NULL then
                ProcessT1(f, Node.LC, Nblocks)  //Traverse left child
                ProcessT2(f, Node.R, Nblocks)   //Process T2 tree
                ProcessT1(f, Node.RC, Nblocks)  //Traverse right child
        end if
end procedure
```

3.3 T2 Tree Processing

```
procedure ProcessT2(f: Pointer to t TOF, Root: Pointer to t T2,var Nblocks: Integer)
        if Root != NULL then
                Bool finished ← False
                Pointer to t_T2 current ← Root
                int j ← 0
```

```
                t_block Buf
                Buf.Nrec ← 0
                Create empty stack S
                while NOT finished do
                        while current != NULL do
                                Push current onto S
                                current ← current.LC
                        end while
                        if S is empty then
                                finished ← True
                        else
                                Pop S to current
                                Buf.Tab[j].Key ← current.Key
                                Buf.Tab[j].blkAddr ← current.BlkNum
                                Buf.Tab[j].recAddr ← current.RecNum
                                Buf.Nrec ← j + 1
                                j ← j + 1
                                if j > MAXTAB then
                                        Write Buf to block Nblocks
                                        Nblocks ← Nblocks + 1
                                        j ← 0
                                end if
                                current ← current.RC
                        end if
                end while
                if j > 0 then
                        Buf.Nrec ← j
                        Write Buf to block Nblocks + 1
                        Nblocks ← Nblocks + 1
                end if
        end if
end procedure
```

4 Key Search Module

```
procedure SearchKey(Root: Pointer to t_T1, Key: Integer,var Found: Boolean,
var BlkNum: Integer,var RecNum: Integer)
        Found ← False
        Pointer to t_T1 current ← Root
        while current != NULL AND NOT Found do
                if Key < current.V1 then
                        current ← current.LC                    //Go left
                else
                        if Key > current.V2 then
                                current ← current.RC            //Go right
                        else
                                Pointer to t_T1 currentT2 ← current.R    //Search in T2
                                while currentT2 != NULL AND NOT Found do
                                        if Key < currentT2.Key then
                                                currentT2 ← currentT2.LC
```

```
                                else
                                        if Key > currentT2.Key then
                                                currentT2 ← currentT2.RC
                                        else
                                                Found ← True
                                                BlkNum ← currentT2.BlkNum
                                                RecNum ← currentT2.RecNum
                                        end if
                                end if
                        end while
                end if
        end if
    end while
end procedure
```

5 Insertion Module

5.1 Insertion in T1 Tree

```
procedure InsertInTree(var Root: Pointer to t_T1, Key: Integer, BlkNum: Integer,
RecNum: Integer)
    if Root is NULL then
        Allocate new node for Root
        Root.V1 ← Key
        Root.V2 ← Key
        InsertInT2(Root.R, Key, BlkNum, RecNum)
        return
    end if
    Pointer to t_T1 current ← Root
    while current != NULL do
        if current.V1 = current.V2 AND Key < current.V1 then
            current.V1 ← Key
            InsertInT2(current.R, Key, BlkNum, RecNum)
            return
        end if
        if current.V1 = current.V2 AND Key > current.V2 then
            current.V2 ← Key
            InsertInT2(current.R, Key, BlkNum, RecNum)
            return
        end if
        if Key current.V1 then
            if current.LC != NULL then
                current ← current.LC
            else
                Create new node newNode
                newNode.V1 ← Key
                newNode.V2 ← Key
                current.LC ← newNode
                InsertInT2(newNode.R, Key, BlkNum, RecNum)
                return
            end if
```

```
            else
                if Key > current.V2 then
                    if current.RC != NULL then
                        current ← current.RC
                    else
                        Create new node newNode
                        newNode.V1 ← Key
                        newNode.V2 ← Key
                        current.RC ← newNode
                        InsertInT2(newNode.R, Key, BlkNum, RecNum)
                        return
                    end if
                else
                    InsertInT2(current.R, Key, BlkNum, RecNum)
                    return
                end if
            end if
        end while
end procedure
```

5.2 Insertion in T2 Tree

```
procedure InsertInT2(var Root: Pointer to t_T2, Key: Integer, BlkNum: Integer,
RecNum: Integer)
        if Root is NULL then
            Allocate new node for Root
            Root.Key ← Key
            Root.BlkNum ← BlkNum
            Root.RecNum ← RecNum
            return
        end if
        Pointer to t T2 current ← Root
        while current != NULL do
            if Key < current.Key then
                if current.LC != NULL then
                    current ← current.LC
                else
                    Create new node newNode
                    newNode.Key ← Key
                    newNode.BlkNum ← BlkNum
                    newNode.RecNum ← RecNum
                    current.LC ← newNode
                    return
                end if
            else
                if current.RC != NULL then
                    current ← current.RC
                else
                    Create new node newNode
                    newNode.Key ← Key
                    newNode.BlkNum ← BlkNum
```

```
                        newNode.RecNum ← RecNum
                        current.RC ← newNode
                        return
                    end if
                end if
            end while
end procedure
```

6 Leaf Node Splitting Module (B-Tree)

6.1 Creating Temporary Array

```
procedure CreateLeafArray(LeafNode: Pointer to B_Tree, newKey: Integer, tmpArr: Ar-
ray[5])
    if LeafNode is NULL then
        return
    end if
    int i ← 0
    Bool processed ← False
    while i < 5 do //Copy existing keys and insert new key in order
        if newKey < LeafNode.Key[i] AND NOT processed then
            tmpArr[i] ← newKey
            processed ← Vrai
        else
            if processed then
                tmpArr[i] ← LeafNode.Key[i - 1]
            Else
                tmpArr[i] ← LeafNode.Key[i]
            end if
        end if
        i ← i + 1
    end while
    if NOT processed then       //New key is the largest
        tmpArr[4] ← newKey
    end if
end procedure
```

6.2 Splitting a Leaf Node

```
procedure SplitLeafNode(var Root: Pointer to B Tree, fullLeafNode: Pointer to B
Tree, newKey: Integer, var middleValue:Integer,var newLeftNode: Pointer to B Tree,
var newRightNode: Pointer to B Tree)
    if fullLeafNode is NULL OR Root is NULL then
        return
    end if
    if NOT checkIfLeafNode(fullLeafNode) then
        return //Node is not a leaf
    end if
    Declare tmpArr: Array[5] of integer
    CreateLeafArray(fullLeafNode, newKey, tmpArr) ▷ Create sorted array
    middleValue ← tmpArr[2]   //Median value to promote
    Allocate newLeftNode
```

```
        Allocate newRightNode      //Fill left node with first 2 keys
        for i = 0 to 1 do
                newLeftNode.Key[i] ← tmpArr[i]
        end for
        newLeftNode.degree ← 3  //2 keys + 1 = 3
        for i = 3 to 4 do                  //Fill right node with last 2 keys
                newRightNode.Key[i - 3] ← tmpArr[i]
        end for
        newRightNode.degree ← 3          //2 keys + 1 = 3
end procedure
```

6.3 Leaf Node Verification

```
function CheckIfLeafNode(Node: B Tree): Boolean
        if Node is NULL then
                return False
        end if
        if Node.child[0] = NULL AND Node.child[1] = NULL AND Node.child[2] =
        NULL AND Node.child[3] = NULL AND Node.child[4] = NULL then
                return True    //All children are NULL
        end if
        return False
end function
```