Analysis of Tree in Computer-based Application

Md. Kamal Ibne Sharif

011132104

Abhijeet Das

011132133

Nazmul Hyder

011131085

Al Amin khan

011132152

A thesis in the Department of Computer Science and Engineering presented

By, partial fulfillment of the requirements for the Degree of

Bachelor of Science in Computer Science and Engineering

United International University

Dhaka, Bangladesh

October 2018

Declaration

We Md. Kamal Ibne Sharif, Abhijeet Das, Nazmul Hyder and Al Amin Khan declare that this thesis "**Analysis of Tree in Computer-based Application**", Thesis Title and the work presented in it are our own. We confirm that:

- This work was done wholly or mainly while in candidature for a BSc degree at United International University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at United International University or any other institution, this has been clearly stated.
- Where we have consulted the published work of others, this is always clearly attributed.
- Where we have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely our own work.
- We have acknowledged all main sources of help.
- Where the thesis is based on work done by ourselves jointly with others, we have made clear exactly what was done by others and what we have contributed ourselves.

_____

Md. Kamal Ibne Sharif

ID: 011132104

Department of CSE

_____

Abhijeet Das

011132133

Department of CSE

_____

Nazmul Hyder

011131085

Department of CSE

_____

Al Amin Khan

011132085

Department of CSE

Certificate

I do hereby declare that the research works embodied in this thesis entitled "**Analysis of Tree in Computer-based Application**" is the outcome of an original work carried out by Md. Kamal Ibne Sharif, Abhijeet Das, Nazmul Hyder and Al Amin under our supervision.

I further certify that the dissertation meets the requirements and the standard for the degree of BSc in Computer Science and Engineering.

_____

Dr. Mohammad Nurul Huda

Professor, Dept. of CSE
United International University

Abstract

The main target of our thesis "**Analysis of Tree in Computer-based Application"** was implemented some tree and some algorithm so that we can able to make some computer based applications. The applications will help you to show which algorithm is working first in same dictionary to search any given words. We also show here how to sort many characters by applying heap sort. Here we provide the directory search by using in-order, pre-order, post-order and post-order trees we also show here a Huffman coding techniques.

Acknowledgement

At first we want to give thanks a lot to our honorable supervisor Dr. Mohammad Nurul Huda, Professor at United International University Dhaka, Bangladesh. He helped us so much without his lot of contribution it was impossible to solve our thesis.

We also thankful to Ayesha Akter, Junior Officer of MSCSE Program at United International University because she was her another helpful person who read our thesis and we are obliged for her valuable comments.

Table of Contents

LIST OF FIGURES

Chapter 1

Introduction

1.1 Data Structure

Data structure need for data organization, store, and management system. For sorting and manage a lot of data we use linked List, array, trees etc. We try to implement here some binary tree algorithms then set our goal to build some applications which can be used in real life-based. There have many scopes to use these algorithms to build some applications so we search here a word in a dictionary and try to show how it will be in two different tree algorithms.

1.2 Tree

A tree is very popular to make a search in many disorder data set in a single word it is famous for sorting something, In computer science, a tree has some parts, it starts by a root value and its sub trees has some children nodes overall a tree is a set of some linked node. We can search and store any data by using tree.

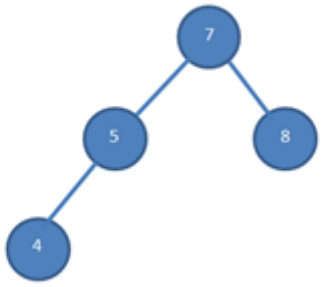


Figure 1 Tree

1.3 Types of Tree

We have been seen many kinds of binary tree, AVL tree, B-tree, B+ tree. So all trees have different technique to sort data and there time-consuming is also different. BSP trees, quadtrees, octrees, R-trees, and other tree are highly used for recursive space partitioning in data structures. The directory structure of a file system is also built by binary search tree. Though we have many options we will show here only a few trees.

1.4 Uses of Tree

A tree is using for manipulating lots of data, to make an easy way for searching information, in a network it highly useful for router algorithm. In real life, a tree algorithm can be used to build a searching application like the dictionary, directory search and much information what we want. On the other hand, a binary tree can be used for sorting data in a real life application too such as heapsort can be used for sorting data now in computer science tree is highly use day by day for searching and sorting data.

Chapter 2

Background

2.1 Searching and Sorting Data Structure

We have seen various kinds of sorting and searching tree in a data structure. First, we store data in an array or linked list then apply tree algorithm for searching and sorting them. In real life, there are many uses of tree algorithms. A binary tree is used for sorting big data and also use in searching big data too. There are many binary trees in data structure B-tree and B+tree are also one kind of binary tree. In real life, it uses for map and set objects in different languages libraries. In 3D video game to determine objects space partition is also highly use day by day.

2.2 Binary Tree

A binary tree has a root node and two parts more they are left side and right side node. These subtrees can have a father node. Our goal is to build some applications by using some binary tree, we try to implement some algorithm first so that we can make some applications. Binary search tree contains –O (log2n) time complexity.

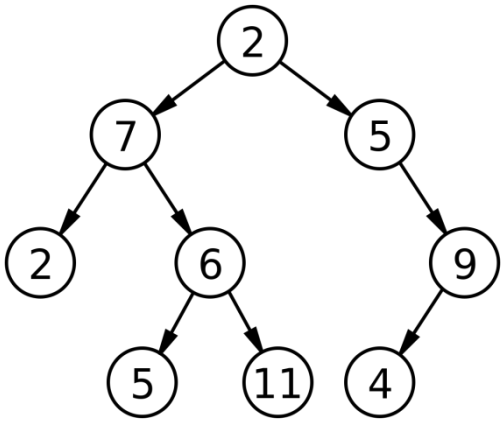


Figure 2 Binary Tree

Chapter 3

Tree Traversal Algorithms

3.1 Tree Traversal

Tree traversal is known as tree searching. It highly uses in graph traversal by order wise it visits each node in a tree. Here we traverse many kinds of binary tree order such as pre-order binary tree traversal, in-order binary tree traversal, post-order binary search tree, level-order binary tree traversal, Breadth-first search traversal, breadth-first tree traversal.

3.1.1 Pre-order Binary Tree

It structured in-order of (Root--Left--Right). We use this pre-order of binary tree in our directory search application we will show our login in the directory application.

An example for: 2,3,5,6,4 (from fig:1)

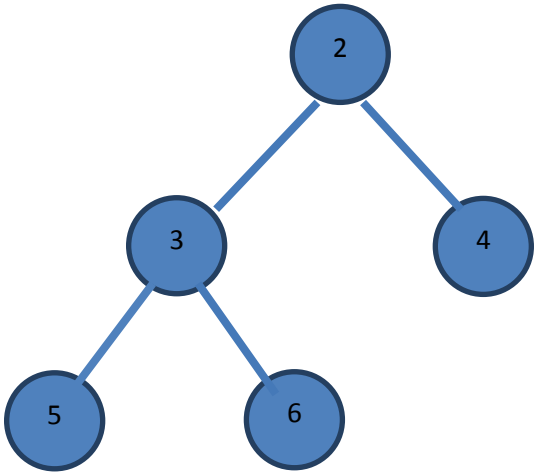


Figure 3 Pre-order Binary Tree

3.1.1.1 Pre-order Pseudo code

```
void preorder(B tree Node) {
      if(node ! = NULL) {
          preorder(node->left)
          preorder(node->right)
          visit(node)
    }
}
```

3.1.1.2 Where Uses

It has many uses in real life programming such as,

- We can use this technique in prefix expression.
- Also, useful for creating a copy of any tree.

3.1.2 In-order Binary Tree

It structured in-order of (Left--Root-- Right).

An example for: 5, 3, 6, 2, 4 (from fig: 1)

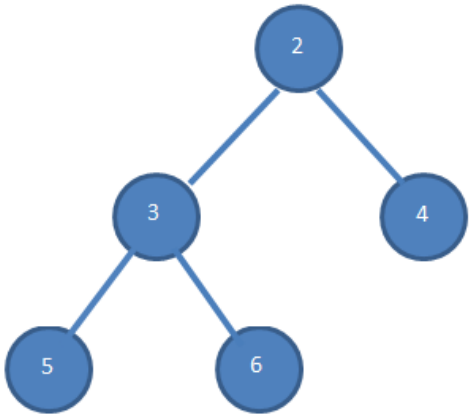


Figure 4 In-order Tree

3.1.2.1 In-order Pseudo code

```
inorder(TreeNode<T> n) {
        if (n != null) {
            inorder(n.getLeft());
            visit(n)
            inorder(n.getRight());
        }
}
```

3.1.2.2 Where Use

Real life programming has many uses of this technique such as in the BST it uses for sort in non-decreasing order, in non-increasing order etc.

3.1.3 Post-order Binary Tree

It structured in-order of (Left-- Right-- Root). We use this post-order of binary tree in our directory search application we will show our login in directory application.

An example for: 5, 6, 3, 4, 2, (from fig: 1)

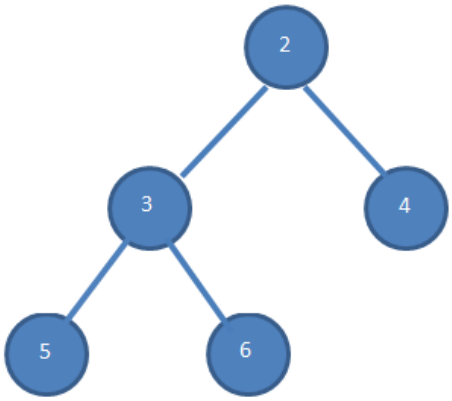


Figure 5 Post-order Binary Tree

3.1.3.1 Post-order Pseudo code

```
postorder(node){
        if (node = null){
            return
            postorder(node.left)
            postorder(node.right)
            visit(node)
        }
}
```

3.1.3.2 Where Use

- For deleting any tree
- get for postfix expression

3.1.4 level-order Binary Tree

Now we are trying to give an example of level-order binary tree.
Nodes are: 6, 4, 9, 3, and 5.

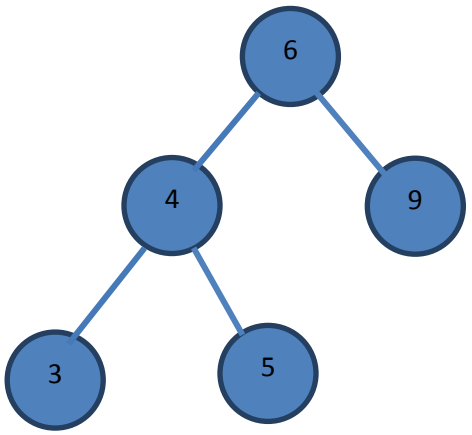


Figure 6 Level-order Binary Tree

3.1.4.1 Level-order Pseudo code

```
while(!q.isEmpty()){
        levelNodes = q.size();
        while(levelNodes>0){
                Node n = (Node)q.remove();
                System.out.print(" " + n.data);
                if(n.left!=null) q.add(n.left);
                if(n.right!=null) q.add(n.right);
                levelNodes--;
        }
        System.out.println("");
}
```

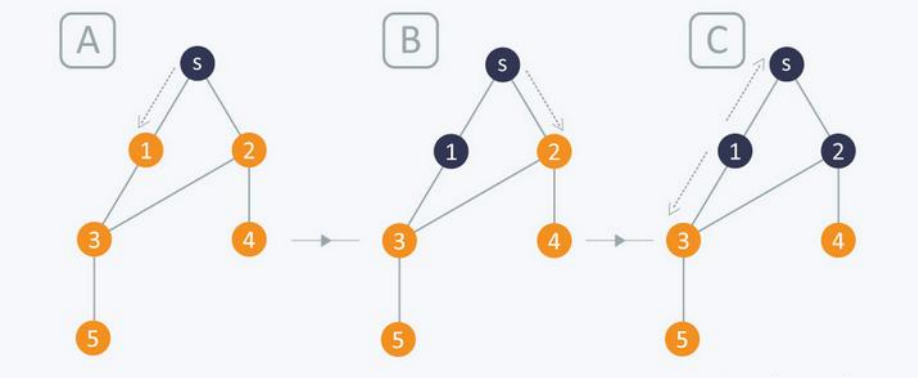
3.1.5 BFS (Breadth-first search)

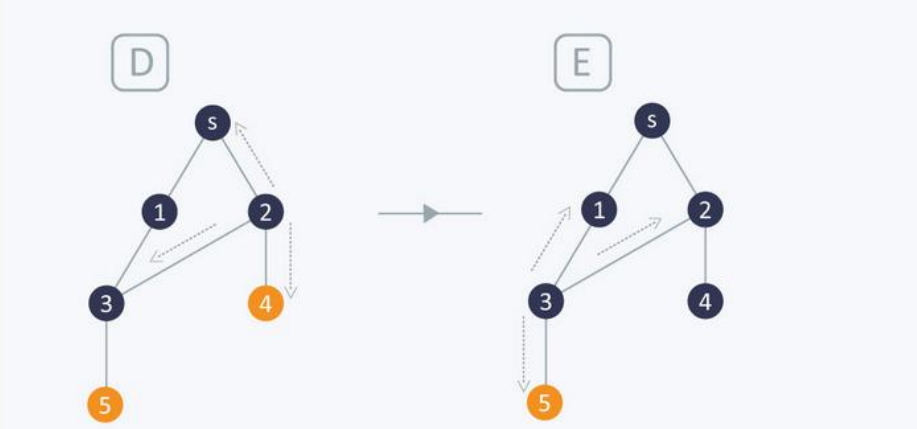
This is an another famous graph searching algorithm, it uses in many ways such as use for searching the shortest path of any graph and in gamming it uses much like Rubiks cubes.

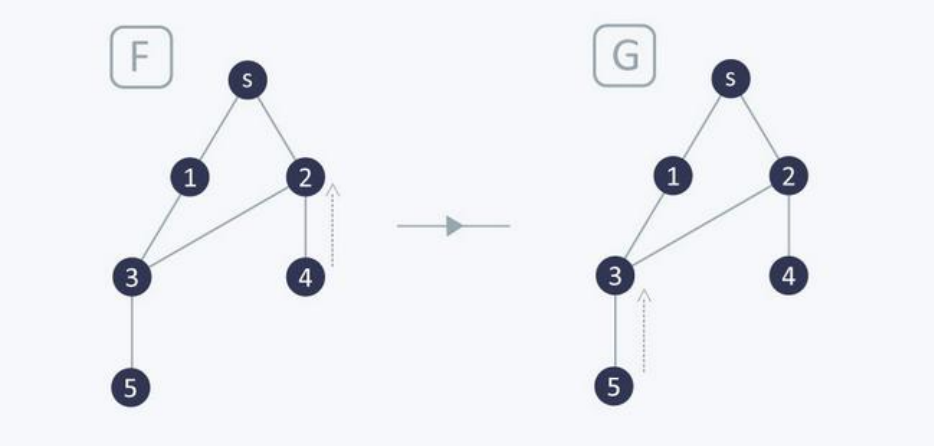
3.1.5.1 Pseudocode of BFS

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2       u.color = WHITE
 3       u.d = ∞
 4       u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11       u = DEQUEUE(Q)
12       for each v ∈ G.Adj[u]
13            if v.color == WHITE
14                 v.color = GRAY
15                 v.d = u.d + 1
16                 v.π = u
17                 ENQUEUE(Q, v)
18       u.color = BLACK
```

3.1.5.2 Mechanism of BFS



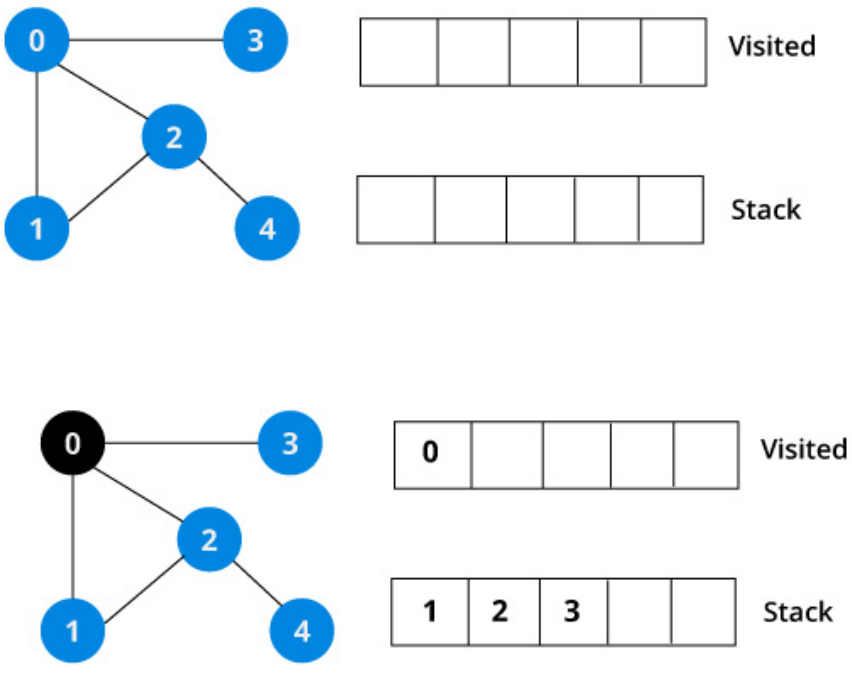


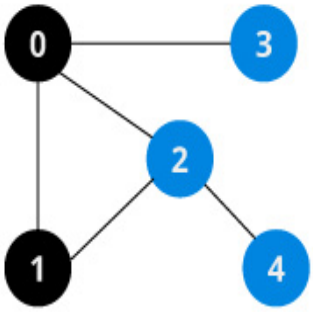


3.1.5.3 DFS Pseudo code

```
DFS(G,v)    ( v is the vertex where the search starts )
   Stack S := {};    ( start with an empty stack )
   for each vertex u, set visited[u] := false;
   push S, v;
   while (S is not empty) do
      u := pop S;
      if (not visited[u]) then
         visited[u] := true;
         for each unvisited neighbour w of u
            push S, w;
      end if
   end while
END DFS()
```

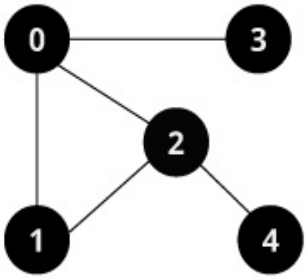
3.1.5.4 Mechanism of DFS





0	1				Visited

2	3				Stack



0	1	2	4	3	Visited

					Stack

Chapter 4

Applications

4.1 Tree Algorithms

A tree algorithm is a technique means one kind of rules for solving any mathematical calculation that has big use in computer science.There are many algorithms for sorting big values, searching algorithms are using to search any value. In our application, we did B-Tree and B+Tree searching algorithm for searching words in a dictionary also did Heapsort for short some characters.

4.2 Tree Sorting

Tree sorting is a sorting algorithm it works like binary tree first it makes a binary tree by given input data list and then traverses the all nodes by using tree algorithms. For sorting some characters we implement here heapsort.

4.2.1 Heapshort

Heapsort is one of the famous sorting and effective algorithm where steps are executed one by one. The order is (nlogn). It has no multiplied complexity. We did ascend order of some characters by using heapsort.

4.2.1.1 Our Application

First, we make the max_heap from the given character tree, this tree will use for our next step where we apply the heap sort to make an ascending order of given characters.
The code of build a max heap from given characters.

```
buildMaxHeap(int size)
{
        for( initial i will be ((size - 1) / 2); i >= 0; i-- )
        {
                maxHeapify(size, i);
        }
}
```

Then we exchange the root node by the last node and placed the root node in a new array and then did heapify between the two nodes then which is grater it replaces by the root node then exchange by the last node and again continue the process.

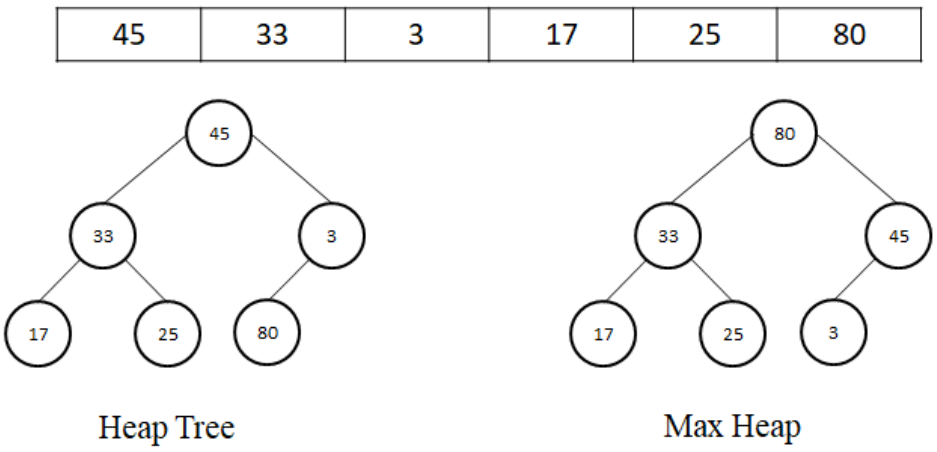
```
heapSortAscendingOrder(int size)

        {
                buildMaxHeap(size);

                int temp for sowap;

                for( int i = (size - 1); i > 0; i-- )
                {
                        temp = myArray[0];
                        myArray[0] = myArray[i];
                        myArray[i] = temp;

                        heapSize--;

                        maxHeapify(size, 0);
                }
        }
```

4.2.1.2 Input / Output

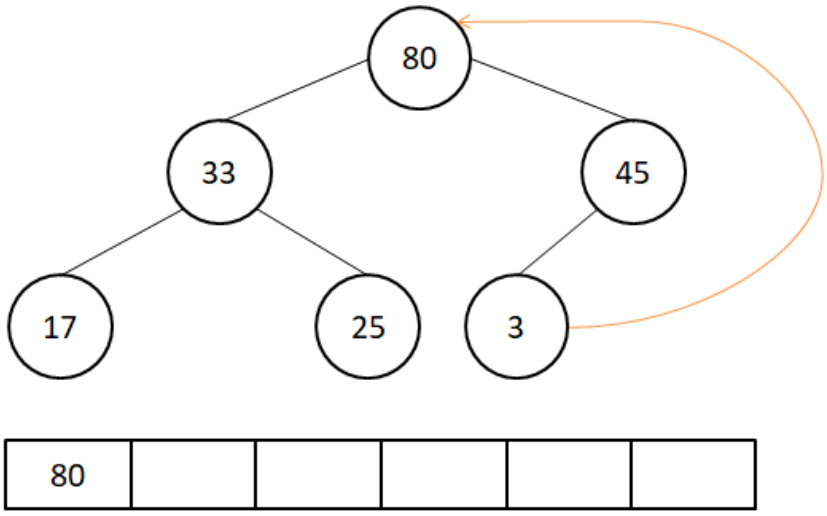
Input: e a c d b

Output: a b c d e

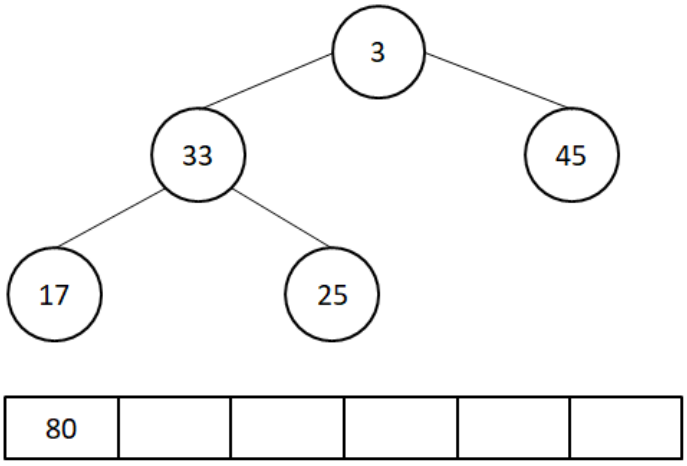
4.2.1.3 Mechanism



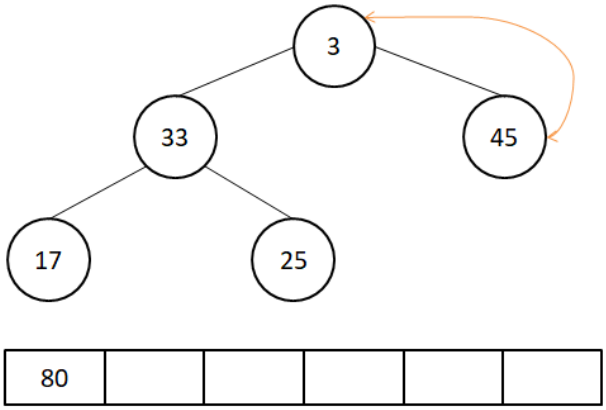
Step:1 (Build a max Heap)



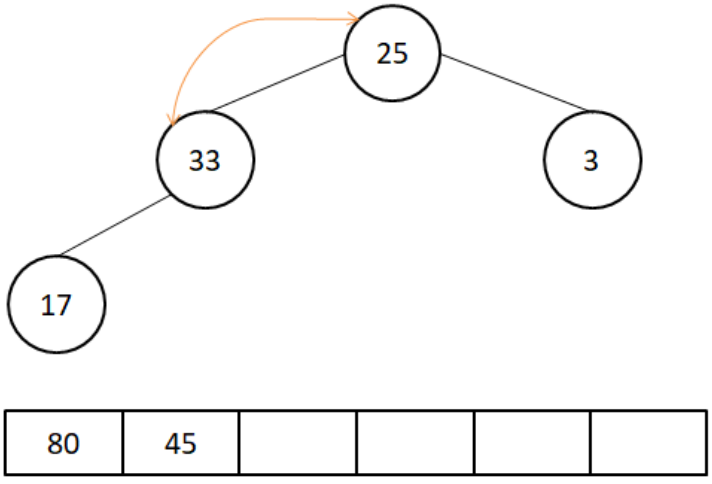
Step: 2 (placed root and exchange with last value)



Step: 3 (Heapifying between 33 and 45)

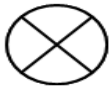


Step: 4 (Swap root with greater node which found by heapify)



Step: 5 (45 was in root so placed in array and now last value 25 is in root)

So this process will continue when we find the last one single node then it becomes stop after doing all the process we got.



80	45	33	25	17	3

Last Step

4.3 Searching Tree

Another data structure is searching tree, it uses for a set position of keys from within a set. There are many searching trees. Many types of binary tree are using for searching data. Example: B-tree, B+tree,(a,b)-tree, ternary search tree. In our dictionary application, we use B-tree and B+tree.

4.3.1 B-Tree

B-tree is a self-balancing tree in computer science. In that tree, a node can have more than two children though it is a generalization of a binary search tree. By using B-tree we made a dictionary. It uses in database and file system. For sequential traversal, it keeps a key in sorted order. For minimizing the number of disks read it works well. It uses partially full blocks to speed insertion and deletion. By using the recursive algorithm it can balance the index. In the figure a B-tree is:

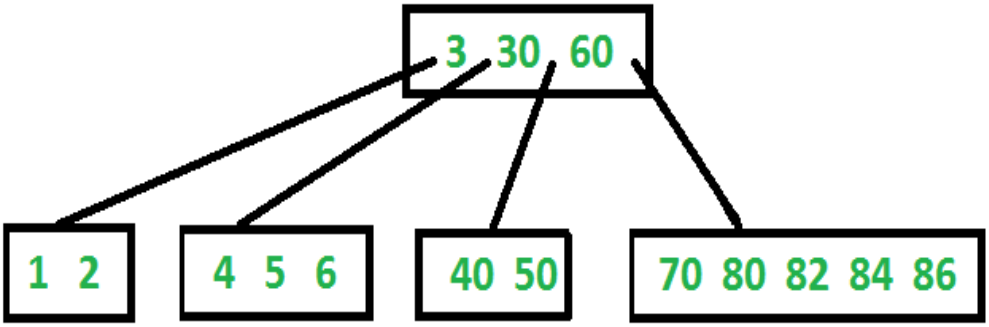


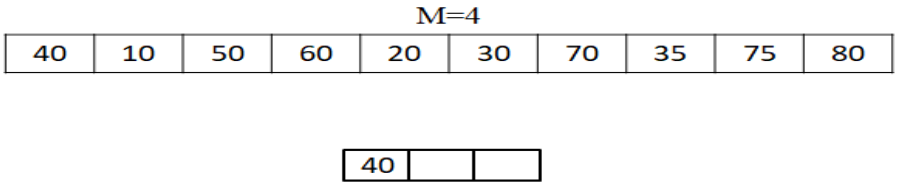
Figure 7 B-tree

4.3.1.1 Pseudo code of B-Tree

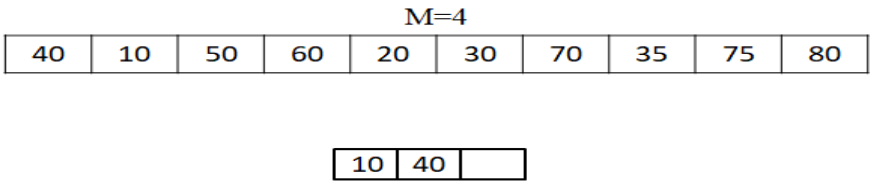
```
B-TREE-INSERT(T, k)
1  r = T.root
2  if r.n == 2t − 1
3      s = ALLOCATE-NODE()
4      T.root = s
5      s.leaf = FALSE
6      s.n = 0
7      s.c_1 = r
8      B-TREE-SPLIT-CHILD(s, 1)
9      B-TREE-INSERT-NONFULL(s, k)
10 else B-TREE-INSERT-NONFULL(r, k)
```

```
B-TREE-SEARCH(x, k)
1  i = 1
2  while i ≤ x.n and k > x.key_i
3      i = i + 1
4  if i ≤ x.n and k == x.key_i
5      return (x, i)
6  elseif x.leaf
7      return NIL
8  else DISK-READ(x.c_i)
9      return B-TREE-SEARCH(x.c_i, k)
```

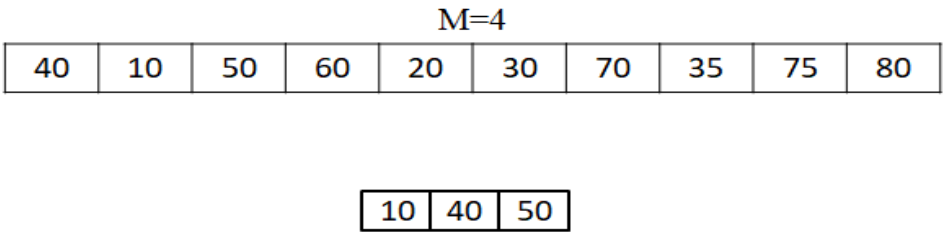
4.3.1.2 Mechanism



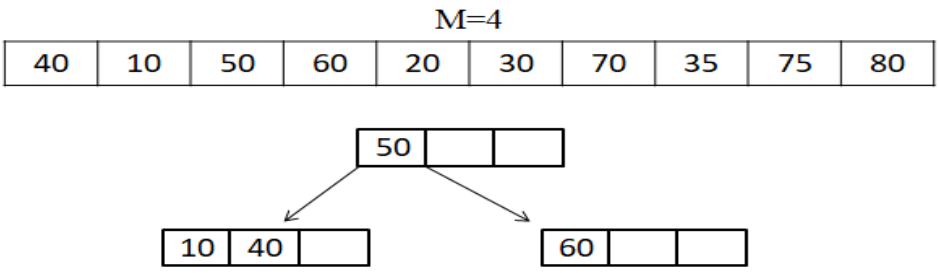
Step: 1



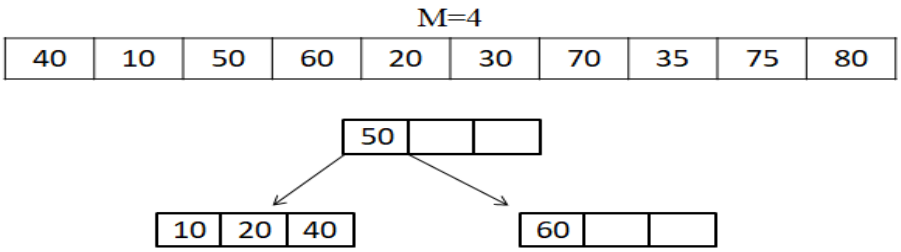
Step: 2



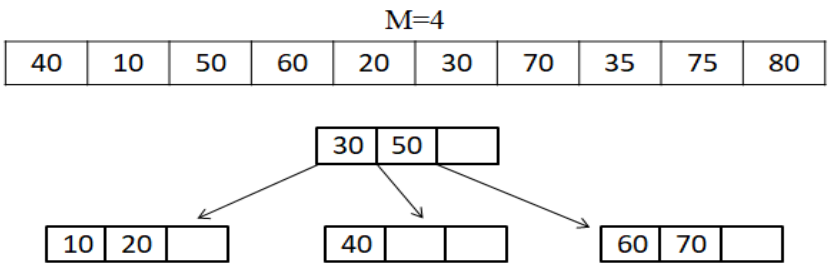
Step: 3



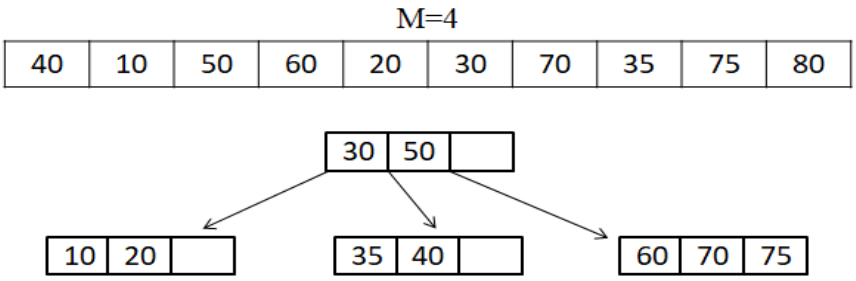
Step: 4



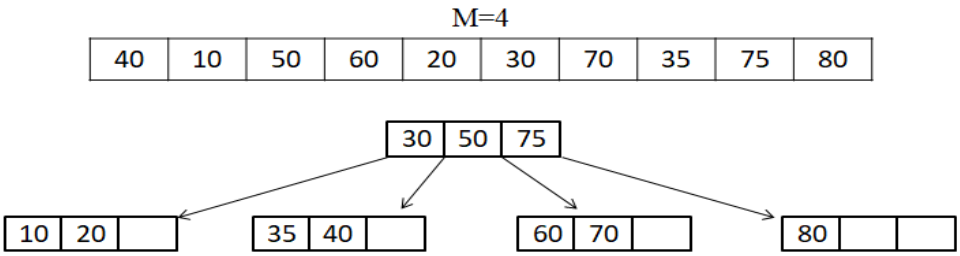
Step: 5



Step: 6



Step: 7



Step: 8

4.3.2 B+Tree

A B+ tree is another type of B tree which also has search operations for data, insertion data and delete any data. Here keys and records can initial place as the left node. We search here some words by using a B+ tree. A B+tree use for store a big amount of data which not stored in main memory. So main memory size keeps limited. In main memory the internal nodes of B+tree are stored, in the secondary memory, the leaf nodes are stored.

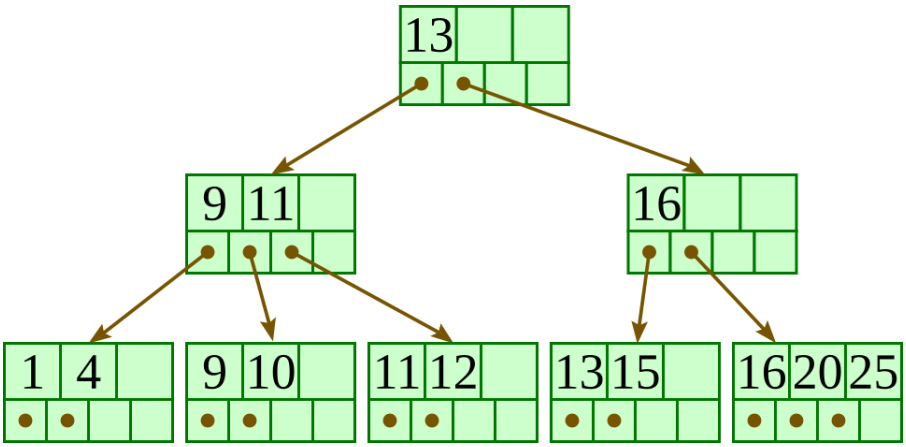
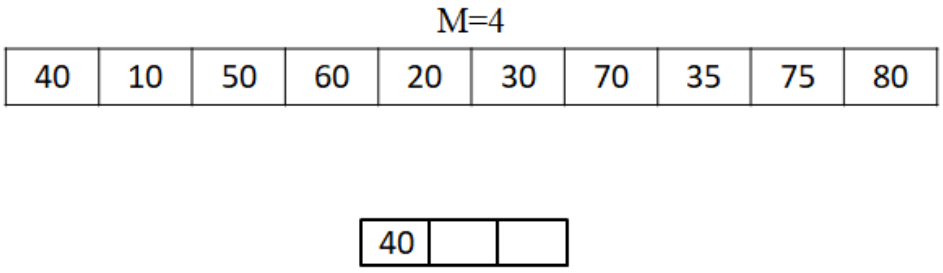


Figure 8 B+tree

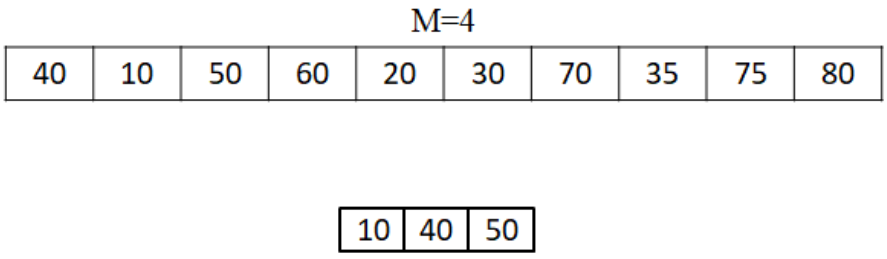
4.3.2.1 Pseudo code of B+Tree

```
BTreeSearch(T,k)
1   x = T.root
2   while not x.leaf
3       i = 1
4       while i ≤ x.n and k > x.router_i
5           i = i+1
6       x = x.c_i
7       DiskRead(x)
8   i = 1
9   while i ≤ x.n and k > x.key_i
10      i = i+1
11  if i ≤ x.n and k = x.key_i
12      return ( x, i )
13  else return NIL
```

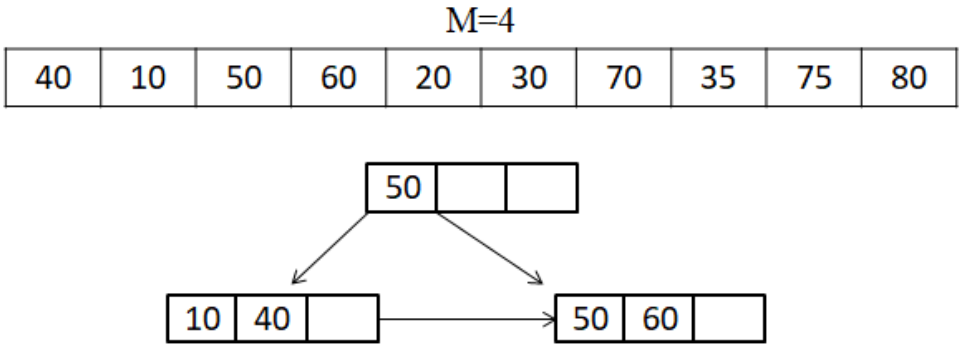
4.3.2.2 B+Tree Mechanism



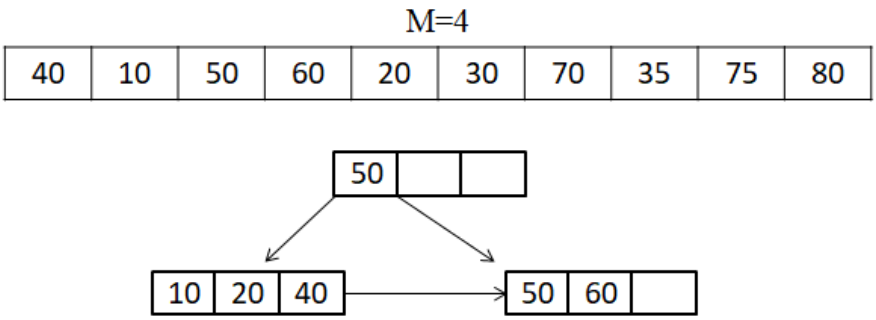
Step: 1



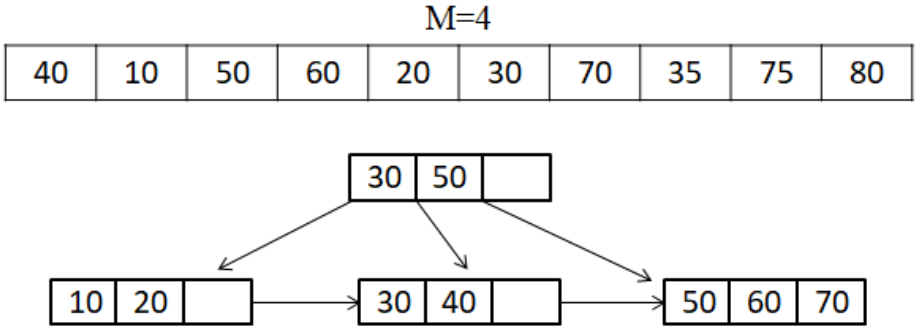
Step: 2



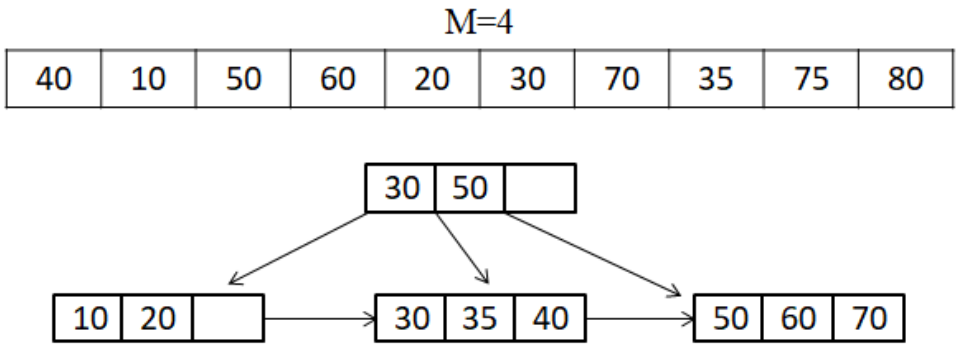
Step: 3



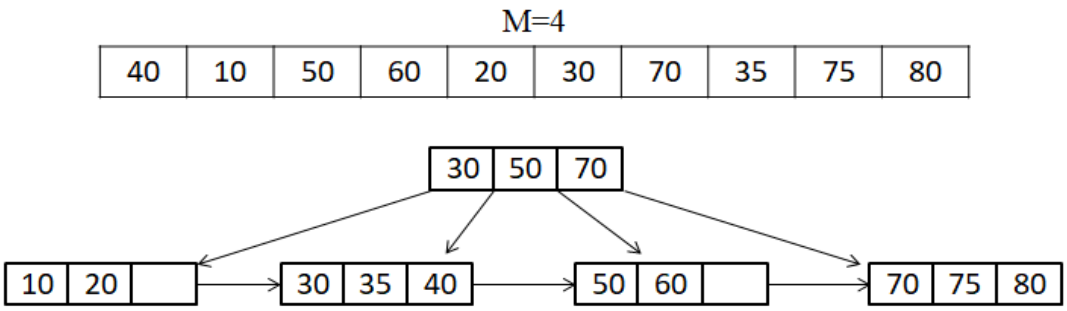
Step: 4



Step: 5



Step: 6



Step: 7

4.4 Implemention of Huffman Algorithm

For compressing a word we implement here Huffman coding. When we input a word it finds out the compression length, then how much it compressed, the word after compressed. It has two main parts:
a) using input tree first build a Huffman tree.
b) assign codes to characters by traversing the Huffman tree.
In a figure Huffman coding is:

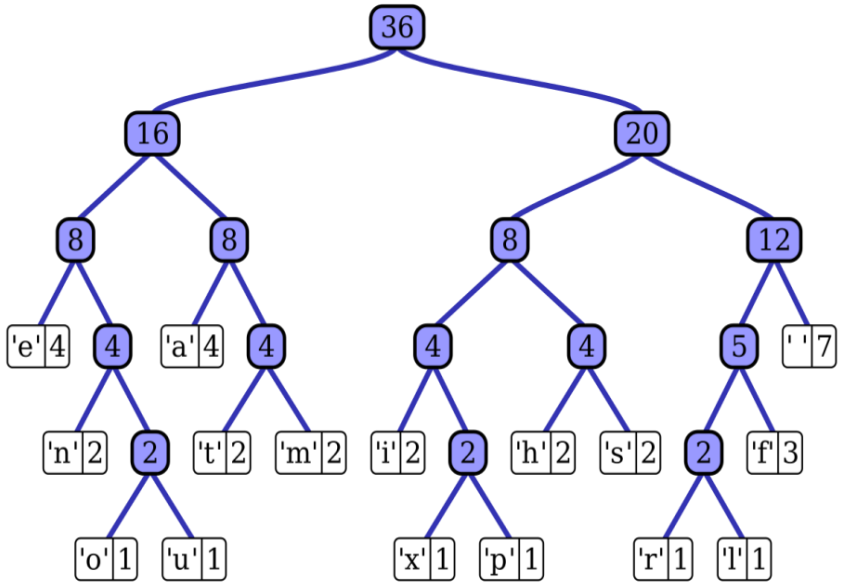


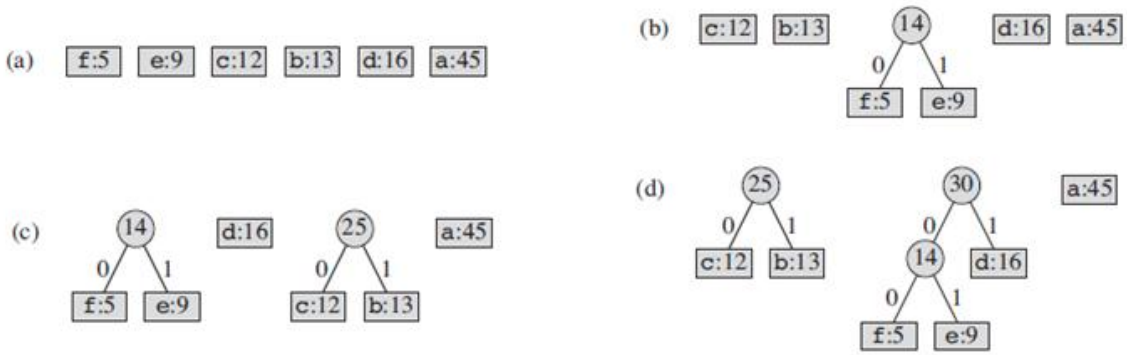
Figure 9 Huffman Coding

4.4.1 Pseudo code of Huffman Coding

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4       allocate a new node z
5       z.left = x = EXTRACT-MIN(Q)
6       z.right = y = EXTRACT-MIN(Q)
7       z.freq = x.freq + y.freq
8       INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return the root of the tree
```

4.4.2 Huffman Coding Mechanism

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101



Step: 1

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Step: 2

Chapter 5

Conclusion & Future Work

5.1 Conclusion

Trees Application is very important in real-world application. Because, there are so many computer-based application uses tree data structure such as Folder in Operating System, Linux File System, Network Routing, Syntax Tree in Compiler, Auto Corrector and Spell Checker.

5.2 Future Work

In future, we will try to build a new technique by using these tree-based algorithm that's why we can easily build Linux File System and Network Routing and also try to develop our dictionary word search like connect with a database so that we can able to search more words.

Reference

[1] Thomas H. Corman,Charles E. Leiserson,Ronald L Rivest,Clifford Stein,
"Introduction to Algorithms". Thired Edition . 2009 Massachusetts Institute of Technology,pp.65-130.

[2] Geeksforgeeks, Tree traversal,2018,
https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder,
[Online; accessed 9-October-2018].

[3] Stantmob , Data compression with huffman coding,2018,
https://medium.com/stantmob/data-compression-with-huffman-coding-ad7bcb07c5d5,
[Online; accessed 10-October-2018].

[4] Wikipedia,Tree traversal,2018,
https://en.wikipedia.org/wiki/Tree_traversal/, [Online; accessed 13-October-2018].

Appendix

Source code of application of B-Tree

B-tree.java

```
import java.util.LinkedList;
import java.util.Queue;

public class BTree {

   private Node root;
   private int t;

   public BTree(int t) {
      this.t = t;
   }

   public Node getRoot() {
      return root;
   }

   public void bTreeCreate() {
      Node x = new Node(t);
      x.leaf = true;
      x.n = 0;
      root = x;
   }

   private void bTreeSplitChild(Node x, int i) {
      Node z = new Node(t);
      Node y = x.c[i];
      z.leaf = y.leaf;
      z.n = t - 1;

      for (int j = 0; j < t - 1; j++)
         z.key[j] = y.key[j + t];

      if (!y.leaf) {
         for (int j = 0; j < t; j++)
            z.c[j] = y.c[j + t];
      }

      y.n = t - 1;

      for (int j = x.n; j >= i + 1; j--)
         x.c[j + 1] = x.c[j];
```

```
    x.c[i + 1] = z;

    for (int j = x.n - 1; j >= i; j--)
      x.key[j + 1] = x.key[j];

    x.key[i] = y.key[t - 1];

    x.n++;
  }

  private void bTreeInsertNonfull(Node x, String k) {
    int i = x.n - 1;

    if (x.leaf) {
      while (i >= 0 && compare(k, x.key[i]) < 0) {
        x.key[i + 1] = x.key[i];
        i--;
      }
      x.key[i + 1] = k;
      x.n++;
    } else {
      while (i >= 0 && compare(k, x.key[i]) < 0)
        i--;
      i++;
      if (x.c[i].n == 2 * t - 1) {
        bTreeSplitChild(x, i);
        if (compare(k, x.key[i]) > 0)
          i++;
      }
      bTreeInsertNonfull(x.c[i], k);
    }
  }

  public void bTreeInsert(String k) {
    Node r = root;

    if (r.n == 2 * t - 1) {
      Node s = new Node(t);
      root = s;
      s.leaf = false;
      s.n = 0;
      s.c[0] = r;
      bTreeSplitChild(s, 0);
      bTreeInsertNonfull(s, k);
    } else bTreeInsertNonfull(r, k);
  }

  public boolean bTreeSearch(Node x, String k) {
    System.out.print("( ");
    for (int j = 0; j < x.n; j++)
```

```
        System.out.print(x.key[j] + " ");
    System.out.print(") ");
    System.out.println();
    int i = 0;
    while (i < x.n && compare(k, x.key[i]) > 0)
        i++;
    if (i < x.n && k.equals(x.key[i]))
        return true;
    else if (x.leaf)
        return false;
    else return bTreeSearch(x.c[i], k);
}

private int compare(String firstWord, String secondWord) {
    char[] firstWordChars = firstWord.toCharArray();
    char[] secondWordChars = secondWord.toCharArray();
    int i = 0;
    char a, b;
    while (i < Math.max(firstWordChars.length, secondWordChars.length)) {
        if (i >= firstWordChars.length)
            a = ' ';
        else a = firstWordChars[i];
        if (i >= secondWordChars.length)
            b = ' ';
        else b = secondWordChars[i];

        if (a < b)
            return -1;
        else if (a == b)
            i++;
        else return 1;
    }
    return 0;
}

private boolean checkVisit(Node node, LinkedList<Node> visitedList) {
    return visitedList.contains(node);
}

private void visit(Node node, LinkedList<Node> visitedList) {
    visitedList.add(node);
}

public void levelOrder(Node root, LinkedList<Node> visitedList) {
    Queue<Node> queue = new LinkedList<>();
    queue.add(root);

    while (queue.peek() != null) {
        Node node = queue.remove();
        if (!checkVisit(node, visitedList)) {
            if (!node.leaf) {
```

```
            System.out.print("( ");
            for (int i = 0; i < node.n; i++)
               System.out.print(node.key[i] + " ");
            System.out.print(") ");

            System.out.print("--> ");
            for (int j = 0; j <= node.n; j++) {
               System.out.print("( ");
               for (int k = 0; k < node.c[j].n; k++)
                  System.out.print(node.c[j].key[k] + " ");
               System.out.print(") ");
            }
            System.out.println();
         }
         visit(node, visitedList);

         if (!node.leaf) {
            for (int i = 0; i <= node.n; i++)
               queue.add(node.c[i]);

            }
         }
      }

   }
}
```

Main.java

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.LinkedList;
import java.util.Scanner;

public class Main {

   public static void main(String [] args)
   {
      File file = new File("E:\\Thesis\\DictionarySearch-master\\src\\WordList.txt");
      Scanner scanner = null;
      try {
         scanner = new Scanner(file);
      } catch (FileNotFoundException e) {
         e.printStackTrace();
      }
      BTree bTree = new BTree(2);
      bTree.bTreeCreate();
      while (scanner != null && scanner.hasNext())
         bTree.bTreeInsert(scanner.next());

      LinkedList<Node> visitedList = new LinkedList<>();
      bTree.levelOrder(bTree.getRoot(), visitedList);
```

```
        String search = "use";
        System.out.println("\nSearch: " + search);
        System.out.println(bTree.bTreeSearch(bTree.getRoot(), search) ? "Yes" : "No");

    }
}
```

Copy.java

```
public class Node {

    public int n;
    public String [] key;
    public Node[] c;
    public boolean leaf;

    public Node(int t) {
        key = new String[2 * t - 1];
        c = new Node[2 * t];
    }
}
```

Source code of application of B+Tree

Bplustree.java

```
import java.util.LinkedList;
import java.util.Queue;

public class BPlusTree {

   private Node root;
   private int t;

   public BPlusTree(int t) {
      this.t = t;
   }

   public Node getRoot() {
      return root;
   }

   public void bPlusTreeCreate() {
      Node x = new Node(t);
      x.leaf = true;
      x.n = 0;
      root = x;
   }

   private void bPlusTreeSplitChild(Node x, int i) {
      Node z = new Node(t);
      Node y = x.c[i];
      z.leaf = y.leaf;
      z.n = y.n - t;

      if (y.leaf) {
         for (int j = 0; j < t; j++)
            z.key[j] = y.key[j + t];
      }
      else {
         for (int j = 0; j < t - 1; j++)
            z.key[j] = y.key[j + t];
      }

      if (!y.leaf) {
         for (int j = 0; j < t; j++)
            z.c[j] = y.c[j + t];
      }

      if (y.leaf)
         y.n = t;
      else y.n = t - 1;
```

```
    for (int j = x.n; j >= i + 1; j--)
        x.c[j + 1] = x.c[j];

    x.c[i + 1] = z;

    for (int j = x.n - 1; j >= i; j--)
        x.key[j + 1] = x.key[j];

    if (y.leaf) {
        x.key[i] = y.key[t];
        x.n++;
    } else {
        x.key[i] = y.key[t - 1];
        x.n++;
    }
}

private void bPlusTreeInsertNonfull(Node x, String k) {
    int i = x.n - 1;

    if (x.leaf) {
        while (i >= 0 && compare(k,x.key[i]) < 0) {
            x.key[i + 1] = x.key[i];
            i--;
        }
        x.key[i + 1] = k;
        x.n++;
    }
    else {
        while (i >= 0 && compare(k, x.key[i]) < 0)
            i--;
        i++;
        if ((x.c[i].n == 2 * t && x.c[i].leaf) || (x.c[i].n == 2 * t - 1 && !x.c[i].leaf)) {
            bPlusTreeSplitChild(x, i);
            if (compare(k, x.key[i]) > 0)
                i++;
        }
        bPlusTreeInsertNonfull(x.c[i], k);
    }
}

public void bPlusTreeInsert(String k) {
    Node r = root;

    if ((r.n == 2 * t && r.leaf) || (!r.leaf && r.n == 2 * t - 1)) {
        Node s = new Node(t);
        root = s;
        s.leaf = false;
        s.n = 0;
        s.c[0] = r;
```

```
        bPlusTreeSplitChild(s, 0);
        bPlusTreeInsertNonfull(s, k);
    }
    else bPlusTreeInsertNonfull(r, k);
}

public boolean bPlusTreeSearch(Node x, String k) {
    while (!x.leaf) {

        System.out.print("( ");
        for (int j = 0; j < x.n; j++)
            System.out.print(x.key[j] + " ");
        System.out.println(") ");

        int i = 0;
        while (i < x.n && compare(k, x.key[i]) >= 0)
            i++;
        x = x.c[i];
    }
    int i = 0;
    while (i < x.n && compare(k, x.key[i]) > 0)
        i++;

    return (i < x.n && k.equals(x.key[i]));
}

private int compare(String firstWord, String secondWord) {
    char[] firstWordChars = firstWord.toCharArray();
    char[] secondWordChars = secondWord.toCharArray();
    int i = 0;
    char a, b;
    while (i < Math.max(firstWordChars.length, secondWordChars.length)) {
        if (i >= firstWordChars.length)
            a = ' ';
        else a = firstWordChars[i];
        if (i >= secondWordChars.length)
            b = ' ';
        else b = secondWordChars[i];

        if (a < b)
            return -1;
        else if (a == b)
            i++;
        else return 1;
    }
    return 0;
}

private boolean checkVisit(Node node, LinkedList<Node> visitedList) {
    return visitedList.contains(node);
}
```

```
  private void visit(Node node, LinkedList<Node> visitedList) {
    visitedList.add(node);
  }

  public void levelOrder(Node root, LinkedList<Node> visitedList) {
    Queue<Node> queue = new LinkedList<>();
    queue.add(root);

    while (queue.peek() != null) {
      Node node = queue.remove();
      if (!checkVisit(node, visitedList)) {
        if (!node.leaf) {
          System.out.print("( ");
          for (int i = 0; i < node.n; i++)
            System.out.print(node.key[i] + " ");
          System.out.print(") ");

          System.out.print("--> ");
          for (int j = 0; j <= node.n; j++) {
            System.out.print("( ");
            for (int k = 0; k < node.c[j].n; k++)
              System.out.print(node.c[j].key[k] + " ");
            System.out.print(") ");
          }
          System.out.println();
        }
        visit(node, visitedList);

        if (!node.leaf) {
          for (int i = 0; i <= node.n; i++)
            queue.add(node.c[i]);

        }
      }
    }

  }
}
```

Main.java

```
import java.io.File;

import java.io.FileNotFoundException;

import java.util.LinkedList;

import java.util.Scanner;
```

```
public class Main {

    public static void main(String [] args)
    {
        File file = new File("/home/anik/IdeaProjects/DictionarySearch/src/WordList.txt");
        Scanner scanner = null;
        try {
            scanner = new Scanner(file);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        BPlusTree bPlusTree = new BPlusTree(2);
        bPlusTree.bPlusTreeCreate();
        while (scanner != null && scanner.hasNext())
            bPlusTree.bPlusTreeInsert(scanner.next());

        LinkedList<Node> visitedList = new LinkedList<>();
        bPlusTree.levelOrder(bPlusTree.getRoot(), visitedList);

        String search = "use";
        System.out.println("\nSearch: " + search);
        System.out.println(bPlusTree.bPlusTreeSearch(bPlusTree.getRoot(), search) ? "Yes" : "No");

    }
}
```

Node.java

```
public class Node {

    public int n;
    public String [] key;
    public Node[] c;
    public boolean leaf;

    public Node(int t) {
        key = new String[2 * t];
        c = new Node[2 * t];
    }
}
```

Output

(experience provoke) --> (compliance) (injury midnight) (sip)

(compliance) --> (bloody) (define)

(injury midnight) --> (harm) (manual) (paralyzed)

(sip) --> (reverse secretion) (suit troop)

(bloody) --> (abolish afford appeal bay) (bloody bury charge cheat)

(define) --> (compliance decrease) (define discipline distance economy)

(harm) --> (experience explain feature feign) (harm hot)

(manual) --> (injury kinship lost main) (manual marriage menu)

(paralyzed) --> (midnight nature paradox) (paralyzed partner prize)

(reverse secretion) --> (provoke rank reconcile) (reverse ritual) (secretion shareholder)

(suit troop) --> (sip solution store student) (suit tear trip) (troop use willpower)

Search: use

(experience provoke)

(sip)

(suit troop)

Yes

Directory Search:

Initial Output:

FOLDER:E:\$RECYCLE.BIN:=0
FILE:E:\Bolo Dugga Mai Ki (2017) 1080p HDRip x264 AC-3 Team Jio Exclusive.mkv:=3222367336
FOLDER:E:\Books:=0
FOLDER:E:\CV Making:=0
FILE:E:\CV-Making-20171007T112959Z-001.zip:=2680854
FOLDER:E:\Important:=4096
FOLDER:E:\NID_provisional Form:=0

FILE:E:\peak-web.zip:=1123356
FOLDER:E:\System Volume Information:=0
FOLDER:E:\voca:=0
FOLDER:E:\Wallpaper:=0
FOLDER:E:\web:=0
FOLDER:E:\YouTube:=0
FOLDER:E:_nishat:=0

Pre Order Traversal:

E:\$RECYCLE.BIN E:\Bolo Dugga Mai Ki (2017) 1080p HDRip x264 AC-3 Team Jio Exclusive.mkv E:\Books E:\CV Making E:\CV-Making-20171007T112959Z-001.zip E:\Important E:\NID_provisional Form E:\System Volume Information E:\voca E:\Wallpaper E:\web E:\YouTube E:_nishat E:\peak-web.zip

In Order Traversal:

E:\$RECYCLE.BIN E:\Books E:\CV Making E:\NID_provisional Form E:\System Volume Information E:\voca E:\Wallpaper E:\web E:\YouTube E:_nishat E:\Important E:\peak-web.zip E:\CV-Making-20171007T112959Z-001.zip E:\Bolo Dugga Mai Ki (2017) 1080p HDRip x264 AC-3 Team Jio Exclusive.mkv

Post Order Traversal:

E:_nishat E:\YouTube E:\web E:\Wallpaper E:\voca E:\System Volume Information E:\NID_provisional Form E:\peak-web.zip E:\Important E:\CV-Making-20171007T112959Z-001.zip E:\CV Making E:\Books E:\Bolo Dugga Mai Ki (2017) 1080p HDRip x264 AC-3 Team Jio Exclusive.mkv E:\$RECYCLE.BIN

Level Order Traversal:

E:\$RECYCLE.BIN E:\Bolo Dugga Mai Ki (2017) 1080p HDRip x264 AC-3 Team Jio Exclusive.mkv E:\Books E:\CV Making E:\CV-Making-20171007T112959Z-001.zip E:\Important E:\NID_provisional Form E:\peak-web.zip E:\System Volume Information E:\voca E:\Wallpaper E:\web E:\YouTube E:_nishat

Output2: (If we want to search any directory in the folder)

Enter folder/file name: (enter 'q' to exit)

E:\YouTube

Path :

E:\$RECYCLE.BIN-> E:\Bolo Dugga Mai Ki (2017) 1080p HDRip x264 AC-3 Team Jio Exclusive.mkv-> E:\Books-> E:\CV Making-> E:\CV-Making-20171007T112959Z-001.zip-> E:\Important-> E:\NID_provisional Form-> E:\System Volume Information-> E:\voca-> E:\Wallpaper-> E:\web-> E:\YouTube