

TOPIC 8: TREE

Objective: At the end of this topic, students should be able to:

1. Understand the concept of list
2. Design and write programs for solving problems by using List data structure.

Section A: Text Concordance using BST

Section B: BST Data Structure

Section C: Worked-example 1-List of Sorted Unique Words

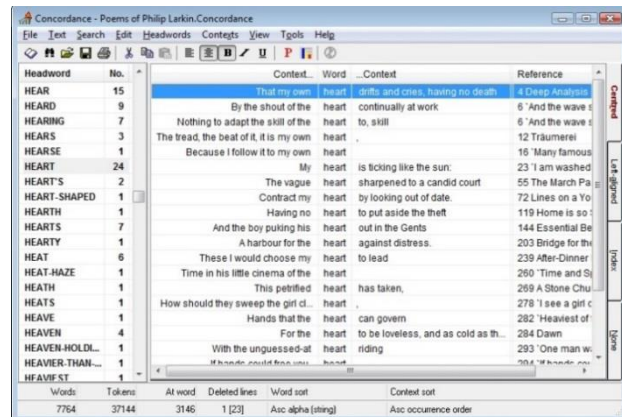
Section D: Lab Task: Word Frequencies

Problems: Text Concordance

Synopsis of the topic:

In this topic, we will learn how binary search tree is used in handling Text Concordance problem. In topic List, we discussed the Text Concordance problem, and how List structure manage the problem.

List structure is a linear structure, hence searching for a certain text will take a linear time. By using Binary Search Tree, we can reduce half of the searching time.



Tree is a nonlinear data structure. Tree consists of nodes. The main node is called a root. Nodes are connected to each other by using a link. A node can have children, where this children are also nodes. Node without child is called a leaf. All nodes must have parent, except for root. As a data structure, the node is where the information is stored.

Tree where its node can have a maximum of two children is called Binary Tree. Furthermore, Binary Tree that store information in some order form for easy searching is called Binary Search Tree. For example, information value of left child is less than its parent, and information value of right child is more than its parent.

Compared with linear structure, the nonlinear structure such as tree does not have a Standard Template Class (in short STL). Therefore, in this topic, we will learn how to create our own class and class template for Binary Search Tree. We will first create a template for Binary Search Tree that covers basic data type such as int, char, string, float and double. By having this template, in the case of Text Concordance problem, we can store our text in the tree. However, if we want to store advance data type such as class, we need to modify our template so that it caters only for that type.

Hence, this topic is structured as follows:

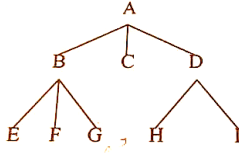
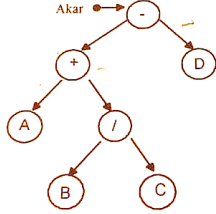
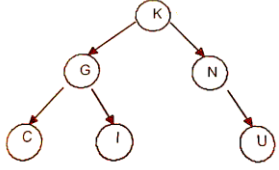
1. About Binary Search Tree
2. Class Template for Binary Search Tree
3. BST with Basic Data Type: Text Searching
4. Class for Binary Search Tree with Advance Data Type: Words Frequencies

About Tree

Unlike Arrays, Linked Lists, Stack and queues, which are linear data structures, trees are hierarchical data structures. Below is the diagram that represent the tree structure.

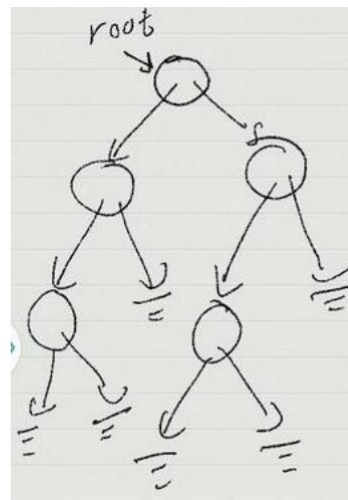
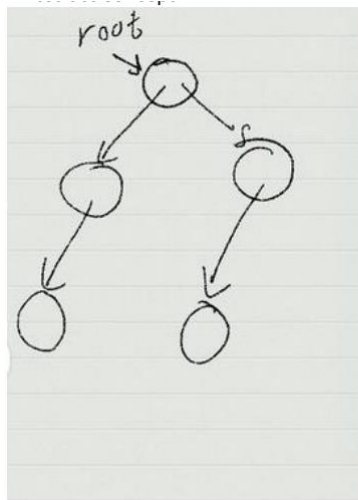
The top most node is called root node of the tree. The elements that are directly under an element are called its children. The element directly above is called its parent. Elements with no children are called leaf nodes.

Below is a table that explain the different between Tree, Binary Tree and Binary Search Tree.

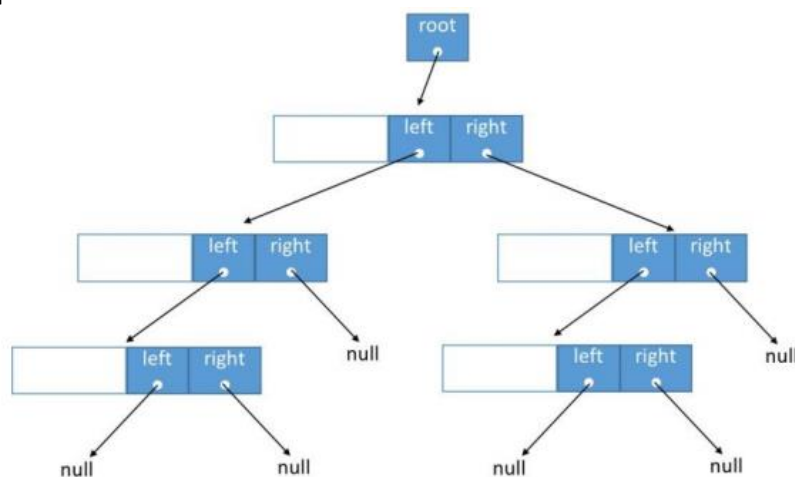
<i>Tree (T)</i>	<i>Binary Tree (BT)</i>	<i>Binary Search Tree (BST)</i>
Non Linear Data Structure. Consist of nodes. Main/first node == Root. Each node connected to other node by a link Node can have children, children == Nodes		
<p>Nodes/elements in tree can have 0 or many children.</p> 	<p>A tree whose elements have at most 2 children: left and right child.</p> 	<p>Binary Search Tree is a node-based binary tree data structure which has the following properties:</p> <ul style="list-style-type: none"> • The left subtree of a node contains only nodes with keys lesser than the node's key. • The right subtree of a node contains only nodes with keys greater than the node's key. • The left and right subtree each must also be a binary search tree. 

In this topic, we will be focus on Binary Search tree (BST).

Abstract concept of BST:



Concrete Concept of BST:



From the concrete concept of BST, we can see that each node from type of class Node that have of 3 items: (1) key, (2) pointer to left child and (3) pointer to right child. Therefore, to implement Binary Search Tree, we need declaration of Class Node as follows:



Node representation:

```

public class Node {
    <data type> key;
    Node left, right;

    public Node (<data type> item) { // constructor
        key = item;
        left = right = null;
    }
}

```

Class Node

Common methods for class BinarySearchTree:

1. Insert Record
2. Delete Record
3. Search Record
4. Inorder Traversal

Given the BinarySearchTree class as below. Full BinarySearchTree Class file can also be obtained from UKMFolio.

Class BinarySearchTree

```

class BinarySearchTree
{
    class Node {
        int key;
        Node left, right;

        public Node(int item) {
            key = item;
            left = right = null;
        }
    }

    Node root; // Root of BST

    BinarySearchTree() { // Constructor
        root = null;
    }

    void deleteKey(int key) { // This method mainly calls deleteRec()
        root = deleteRec(root, key);
    }

    // A recursive function to delete a node in BST
    Node deleteRec(Node root, int key) {

        if (root == null) return root; // Base Case: If the tree is empty

        /* Otherwise, recur down the tree */
        if (key < root.key)
            root.left = deleteRec(root.left, key);
        else if (key > root.key)
            root.right = deleteRec(root.right, key);

        // if key is same as root's key, then This is the node to be deleted
        else {
            // node with only one child or no child

```

```

        if (root.left == null)
            return root.right;
        else if (root.right == null)
            return root.left;

        // node with two children: Get the inorder successor
        //(smallest in the right subtree)
        root.key = minValue(root.right);

        // Delete the inorder successor
        root.right = deleteRec(root.right, root.key);
    }
    return root;
}

int minValue(Node root)
{
    int minv = root.key;
    while (root.left != null)
    {
        minv = root.left.key;
        root = root.left;
    }
    return minv;
}

// This method mainly calls insertRec()
void insert(int key) {
    root = insertRec(root, key);
}

// A recursive function to insert a new key in BST
Node insertRec(Node root, int key) {

    if (root == null) // If the tree is empty, return a new node
    {
        root = new Node(key);
        return root;
    }

    /* Otherwise, recur down the tree */
    if (key == root.key)
        System.out.println("\nThe " + key+ " is in the BST");

    if (key < root.key)
        root.left = insertRec(root.left, key);
    else if (key > root.key)
        root.right = insertRec(root.right, key);

    return root; // return the (unchanged) node pointer
}

// This method mainly calls InorderRec()
void inorder() {
    printInorder(root);
}

// A utility function to do inorder traversal of BST
void printInorder(Node root) {
    if (root != null)
    {
        printInorder(root.left);
        System.out.print(root.key + " ");
        printInorder(root.right);
    }
}

```

```

    }
}

// This method mainly calls InorderRec()
boolean search(int key) {
    boolean found;
    searchRec(this.root, key);
    if (this.root.key==key)
        return true;
    else
        return false;
}

// This method is to Serach a Rec
public Node searchRec(Node root, int key)
{
    // Base Cases: root is null or key is present at root
    if (root==null || root.key==key)
        return root;

    // Key is greater than root's key
    if (root.key < key)
        return searchRec(root.right, key);

    // Key is smaller than root's key
    return searchRec(root.left, key);
}
}

```

B. Tutorial

1. State the disadvantages of using BST instead of List data structure?
2. When is the best condition to used BST compare to list data structure?
3. What is the best data structure to solve word frequencies problem? Why?
4. Draw and write the class node to represent binary tree and tree. Spot any differences with class node for binary search tree.

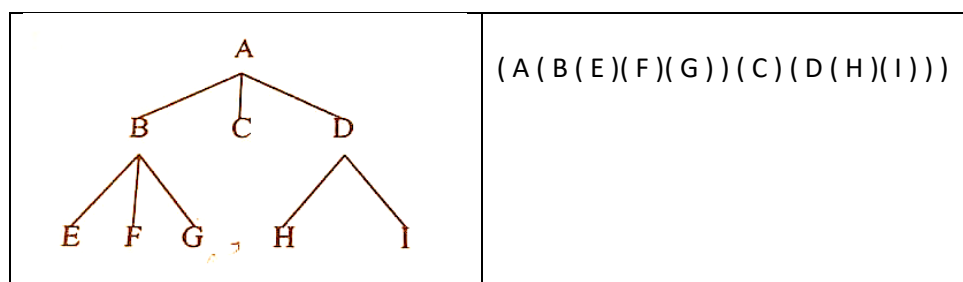
C. Lab Task

One way to represent a structure of a tree is to use parentheses. Below is the tree with parentheses format:

(root, (sub-left tree) (sub-right tree))

Where every sub-left and sub-right will follow the same format.

Example: Tree in parentheses format



Tree With Parentheses	
Input	Standard Input
Output	Standard Output
Data Structure	BST

Problem Description

Write a Java program that reads a passage and assign all words into BST. Your code then will print all unique words in parentheses format as below:

(root, (sub-left tree) (sub-right tree))

Sample Input and Ouput

Input	Output
E C A D G I.	(E(C(A())(D()))(G()(I())))
i go to school by bus. the bus is big.	(i(go(by(bus(big()))())())(to(school(is())(the())))))

LIST OF UNIQUE WORDS USING BST

Input	Standard Input
Output	Standard Output
Data Structure	BST

Problem Description

List of Unique Words problem aims to display all words from a given passage in a non-recurring form. Your task is to write a Java program for the **List of Unique Words**.

Input

Input of this program is a passage. A passage consists of N words and symbols. Symbols that will be considered in the passage are full stop (.), comma (,), question mark (?) and exclamation mark (!). The passage will have M unique words, where the M is less than or equal to N.

Output

Output of the program is M lines, where each line contains a unique word.

Sample Input-Output

Input	Output
I go to school by bus. The bus is big. The school is also big. I like big school and big bus.	I The also and big bus by go is like school to

Solution

The algorithm for List of Unique Words:

Read a passage

Remove unnecessary characters

For all words in the passage:

Search the word in BST

If not found

Add the word in BST

Traverse to display all words using in order traversal.

Full structure of worked-example program: List of Unique Words using BST.

```
1 import java.util.Scanner;
2 public class UniqueWordTree {
3     public static void main(String[] args) {
4         BinarySearchTreeWord1 tree = new BinarySearchTreeWord1();
5         Scanner in = new Scanner(System.in);
6
7         String passage = in.nextLine(); // read input passage
8         String delims = "\\W+"; // split any non word
9         String [] words = passage.split(delims);
10        for (String str : words){
11            // add code to add word into tree
12        }
13
14        //add code to print all unique words
15    }
16 }
17
```

C. Tutorial:

1. Write program segment to print the number of repeated words from BST.
2. Discuss the different between the **list of words** and **list of unique words** using BST.

WORD FREQUENCIES

Input	Standard Input
Output	Standard Output
Data Structure	Binary Search Tree

Problem Description

Write a JAVA program that will read all words from a passage and store them in BST. The program then will display a menu and perform the following task:

- **Delete a word.** If word exists, update the word frequency otherwise print word not exist.
- **Search a word.** If word exists, print the word and its frequency otherwise print not exist.
- **Add a new word.** If the word is in the passage, update its frequency.
- **Print all words** and its frequency in ascending order.

Input

Input of this program is a passage. A passage consists of N words and symbols. Symbols that will be considered in the passage are full stop (.), comma (,), question mark (?) and exclamation mark (!). The passage will have M unique words, where the M is less than or equal to N.

Followed by the input code and the required data as specified in the sample input-output.

Sample Input-Output

Input	Output
Stay at home, stay safe and stay healthy. Practice social distancing at work and at home. Help reduce the risk of infection and preventing covid19. Together we break the chain of covid19. #kita jaga kita. Hapuskan covid19!.	Menu: 1. Add new word. 2. Delete word. 3. Search word. 4. Print all words and its frequency. 5. Exit. Input code: 2 Enter word to be deleted: pendamic the word pendamic is not in the passage. Menu: 1. Add new word. 2. Delete word. 3. Search word. 4. Print all words and its frequency. 5. Exit. Input code: 2

	<p>Enter word to be deleted: covid19 one of the words covid19 was successfully deleted.</p> <p>Menu:</p> <ol style="list-style-type: none"> 1. Add new word. 2. Delete word. 3. Search word. 4. Print all words and its frequency. 5. Exit. <p>Input code: 2 Enter word to be deleted: we the word we was successfully deleted.</p> <p>Menu:</p> <ol style="list-style-type: none"> 1. Add new word. 2. Delete word. 3. Search word. 4. Print all words and its frequency. 5. Exit. <p>Input code: 1 Enter new word: pendamic the word pendamic has been added successfully.</p> <p>Menu:</p> <ol style="list-style-type: none"> 1. Add new word. 2. Delete word. 3. Search word. 4. Print all words and its frequency. 5. Exit. <p>Input code: 4 Hapuskan(1) Help(1) Practice(1) Stay(1) Together(1) and(3) at(3) break(1) chain(1) covid19(2) distancing(1) healthy(1) home(2) infection(1) jaga(1) kita(2) of(2) pendamic(1) preventing(1)</p>
--	--

	<pre> reduce(1) risk(1) safe(1) social(1) stay(2) the(2) work(1) Menu: 1. Add new word. 2. Delete word. 3. Search word. 4. Print all words and its frequency. 5. Exit. Input code: 1 Enter new word: social the word social has been added successfully. Menu: 1. Add new word. 2. Delete word. 3. Search word. 4. Print all words and its frequency. 5. Exit. Input code: 4 Hapuskan(1) Help(1) Practice(1) Stay(1) Together(1) and(3) at(3) break(1) chain(1) covid19(2) distancing(1) healthy(1) home(2) infection(1) jaga(1) kita(2) of(2) pendamic(1) preventing(1) reduce(1) risk(1) safe(1) social(2) stay(2) the(2) work(1) Menu: 1. Add new word. </pre>
--	---

	<pre> 2. Delete word. 3. Search word. 4. Print all words and its frequency. 5. Exit. Input code: 3 Input Search text: home the word home is used 2 times. Menu: 1. Add new word. 2. Delete word. 3. Search word. 4. Print all words and its frequency. 5. Exit. Input code: 3 Input Search text: work the word work is used once. Menu: 1. Add new word. 2. Delete word. 3. Search word. 4. Print all words and its frequency. 5. Exit. Input code: 3 Input Search text: good the word good is not in the passage. Menu: 1. Add new word. 2. Delete word. 3. Search word. 4. Print all words and its frequency. 5. Exit. Input code: 2 Enter word to be deleted: work the word work was successfully deleted. Menu: 1. Add new word. 2. Delete word. 3. Search word. 4. Print all words and its frequency. 5. Exit. Input code: 2 Enter word to be deleted: </pre>
--	---

	<p>social</p> <p>one of the words social was successfully deleted.</p> <p>Menu:</p> <ol style="list-style-type: none"> 1. Add new word. 2. Delete word. 3. Search word. 4. Print all words and its frequency. 5. Exit. <p>Input code:</p> <p>4</p> <p>Hapuskan(1)</p> <p>Help(1)</p> <p>Practice(1)</p> <p>Stay(1)</p> <p>Together(1)</p> <p>and(3)</p> <p>at(3)</p> <p>break(1)</p> <p>chain(1)</p> <p>covid19(2)</p> <p>distancing(1)</p> <p>healthy(1)</p> <p>home(2)</p> <p>infection(1)</p> <p>jaga(1)</p> <p>kita(2)</p> <p>of(2)</p> <p>pendamic(1)</p> <p>preventing(1)</p> <p>reduce(1)</p> <p>risk(1)</p> <p>safe(1)</p> <p>social(1)</p> <p>stay(2)</p> <p>the(2)</p> <p>Menu:</p> <ol style="list-style-type: none"> 1. Add new word. 2. Delete word. 3. Search word. 4. Print all words and its frequency. 5. Exit. <p>Input code:</p> <p>10</p> <p>Invalid code, Enter code 1-5:</p> <p>Menu:</p> <ol style="list-style-type: none"> 1. Add new word. 2. Delete word. 3. Search word. 4. Print all words and its frequency. 5. Exit. <p>Input code:</p> <p>99</p> <p>Invalid code, Enter code 1-5:</p>
--	---

	<p>Menu:</p> <ol style="list-style-type: none">1. Add new word.2. Delete word.3. Search word.4. Print all words and its frequency.5. Exit. <p>Input code:</p> <p>5</p> <p>Bye..</p>
--	---

Solution

The algorithm for Word Frequencies:

Read a passage

Remove unnecessary characters

For all words in the passage

Travers to find the word from the BST

If not found

add the word into the BST and with frequency = 1

else

add 1 to the frequency of the word

Traverse to display all words with its frequency using in order traversal.