**PROJECT REPORT**

**TOPIC**: Search optimization using OBST

**Submitted By:**

**Name Roll No.**

Gaurav Baisware 47

Jereen Valsson 49

Mrugank Thatte 55

**Status :** Complete

**Description:**

**Optimal Binary Search Tree**

Also called as weight-balanced binary tree, it is a binary search tree which provides the smallest possible search cost for a given sequences. In layman terms, it is adjusting the binary search tree such that the cost of obtaining data reduces by arranging them with the help of their help of their probabilities

**Natural Language Processing Toolkit**

NLTK is a suite of libraries and programs for symbolic natural language processing for Python programming language. Functionalities used in this project are:

word\_tokenize(): It returns words from the passed strings

stopwords(): It consists of all the stop words that are used in the passed language

FreqDist() : It creates a frequency distribution for the passed list. Is mostly used to find the frequency of elements

**Tkinter**

It is a toolkit in python which is used for creation of graphical user interfaces.

**Task:**

Extract words from user defined files and create an Optimal Binary Search Tree.

**Process:**

With files given by user, we used word\_tokenizer() and FreqDist() to create a sorted frequency table consisting of word, frequency, probability and unsuccessful probability.

Probability(word) = Frequency(word)/Total words

Unsuccessful Probability (word) = No. of files in which the word is not occurring/ Total no. of files

This table is passed to dispmat() to generate which returns the root matrix after performing OBST dynamic programming algorithm , thus creating its evolution as well as weight matrix.

Root matrix is passed to recursive function construct\_tree() which creates the OBST and returning root node of the tree.

The root of the tree is passed to level\_o() , which with the help of height() and nodes\_level() returns a list of nodes of the tree in level order.

This list is passed to the OBST() class to create the binary search tree.

The constructor of the class initiates UI() which iterating through the list, creates a node with addEntry and createNode() and adds it to the tree.The inorder() function is used to display the tree on the canvas .

searchKey() is a function which is initiated when the “SEARCH” button is pressed and passes the string in the searchEntry() to a recursive function search() which displays whether the string was found or not and also the cost of the iteration.

setColorScheme() is associated with setting the color scheme of the canvas,buttons and the nodes itself.

**Code Snippets:**

def clickbutt1():  
   
 n = int(e1.get())  
 file\_list = e2.get().split(",")  
 # print(file\_list,n)  
 word\_list = getwordlist(file\_list, n)  
 word\_list = printlist(word\_list)  
 Label(groot, text=" ", bg=themecolor).grid(row=rowvar + n + 3, column=colvar)  
 root\_mat = dispmat(word\_list, rowvar + n + 4, 0)  
 tree\_root = None  
 tree\_root = construct\_tree(tree\_root, 1, n, root\_mat, word\_list, "")  
 button3 = Button(groot, text="Print tree", command=lambda: OBST(level\_o(tree\_root)), bg=buttoncolor, fg='white').grid(row=1, column=colvar + 3)  
 # OBST(level\_o(tree\_root))  
 return word\_list

This function runs on clicking the first button and is used to take input from the user and initiate calculations.

def dis(mat, rowvar, colvar):  
 # This Function is used to display a matrix on the Main Window  
 k = 0  
 for i in mat:  
 if len(i) != 0:  
 s = ""  
 for j in i:  
 s = s + str(j) + "\t"  
 Label(groot, text=s, bg=themecolor).grid(row=rowvar, column=colvar)  
 rowvar += 1

This function takes a matrix mat and displays it on the GUI in grid pattern using the rowvar and colvar as co-ordinates.

def inorder(self, subRoot, row, col, sep):  
 # This is a Recursive Function used to display the Tree on the Canvas properly  
 if subRoot:  
 if subRoot.right is not None:  
 self.canvas.create\_line(col, row, col + sep, row + 50, width=2, fill=self.outlineColor)  
 if subRoot.left is not None:  
 self.canvas.create\_line(col, row, col - sep, row + 50, width=2, fill=self.outlineColor)  
  
 self.inorder(subRoot.left, row + 50, col - sep, sep / 2)  
  
 # print(subRoot.data, end=" ")  
 self.canvas.create\_oval(col - self.nodeWidth / 2, row - self.nodeWidth / 2, col + self.nodeWidth / 2, row + self.nodeWidth / 2, fill=self.nodeColor, outline=self.outlineColor, width=2)  
 self.canvas.create\_text(col, row, text=str(subRoot.name), fill='black', font=('arial', 10, 'bold'))  
  
 self.inorder(subRoot.right, row + 50, col + sep, sep / 2)

This function is used to display the tree on the canvas .

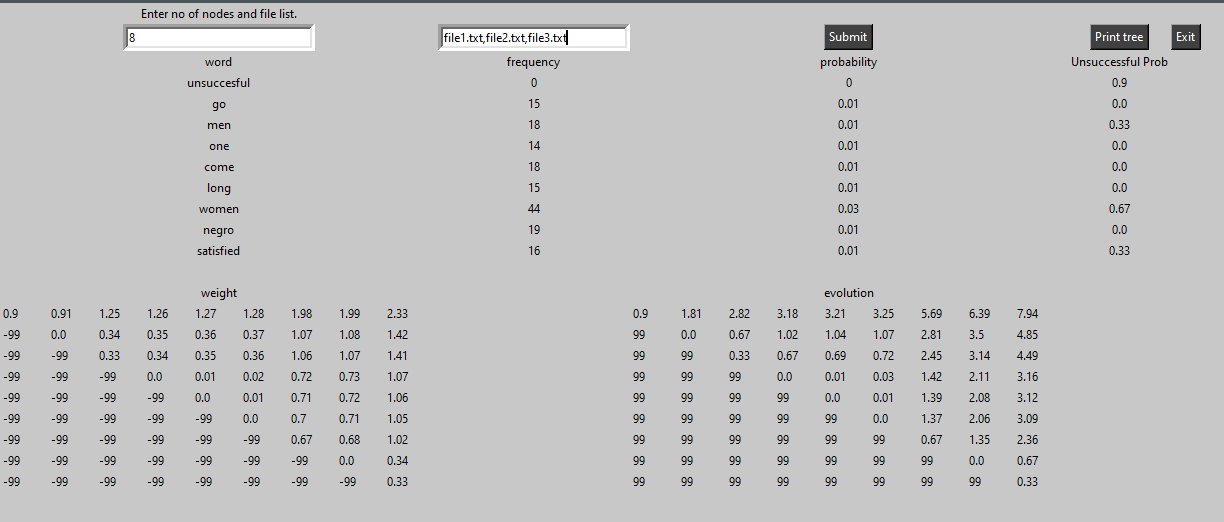
def UI(self):  
 # This Function is used to update the window continuously for any changes if made  
 # And displaying the Tree on the Canvas  
 i = 0  
 while self.loopActive:  
 self.canvas.update()  
 if i < len(self.l):  
 self.addEntry(self.l[i])  
 i += 1  
 if self.n == 0:  
 self.inorder(self.root, 30, 600, 300)  
 if self.adding:  
 self.root = self.insert(self.root, self.name, self.cost, self.cost2)  
 self.adding = False  
 self.n = 0  
 self.inorder(self.root, 30, 600, 300)  
 self.window.destroy()

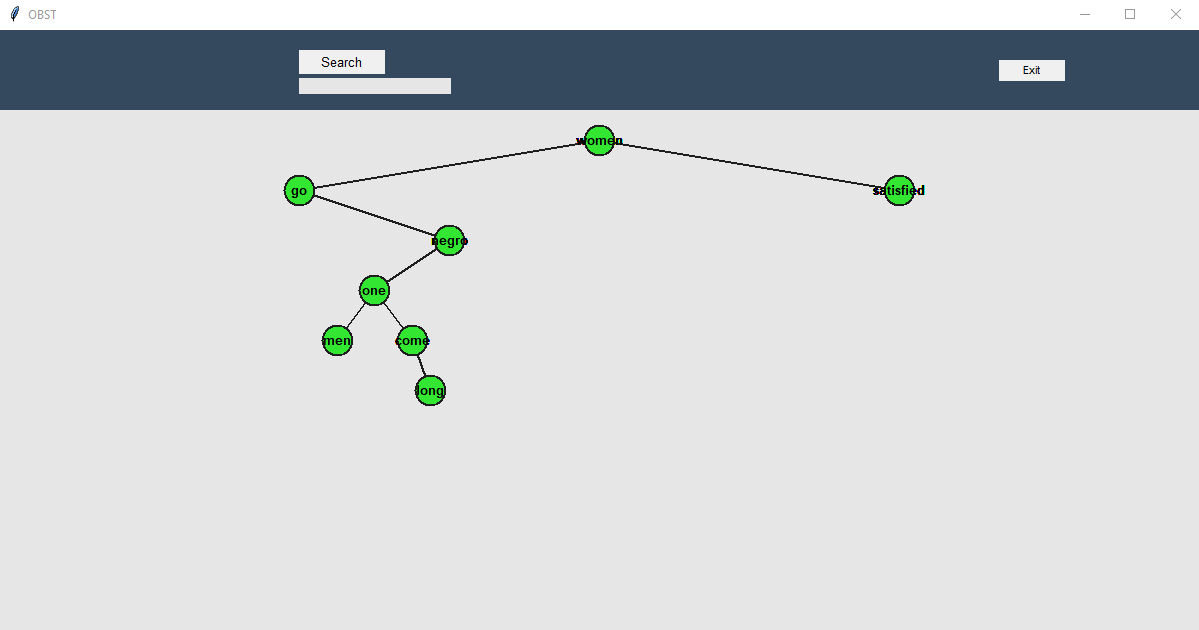
This function is used to update the canvas continuously.

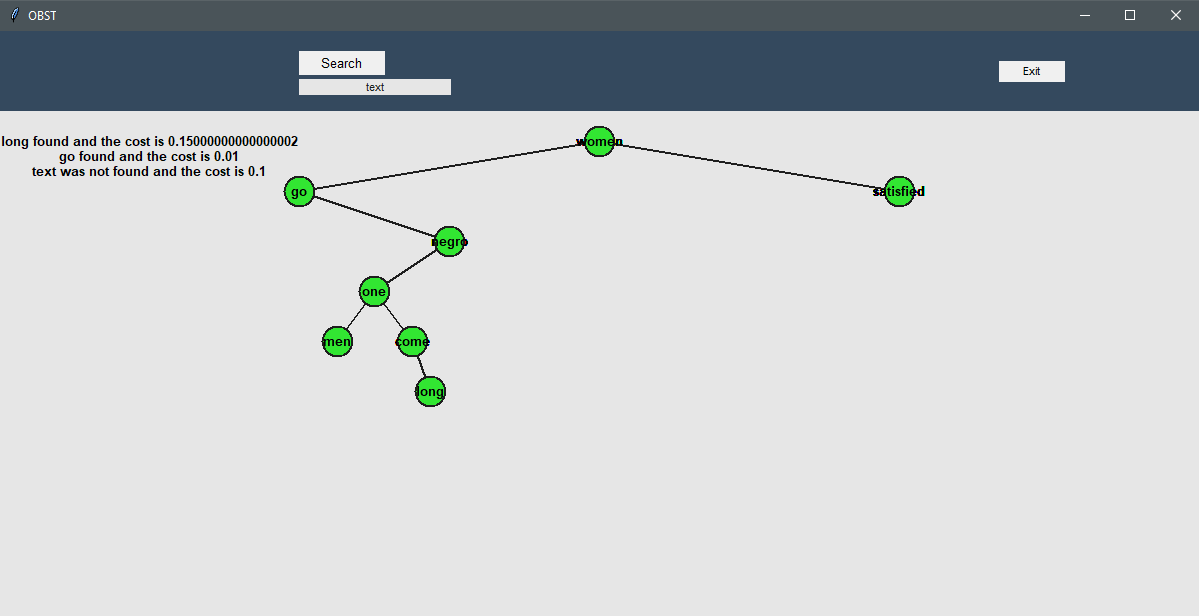
def searchKey(self):  
 # This Function takes the Search Key value from the Textbox if its length is not 0  
 if len(self.searchEntry.get()) == 0:  
 self.n = 0  
 else:  
 self.search(self.root, self.searchEntry.get(), 0, 0)

This function is used to search the user input values in the generated.

**OUTPUT**:





****

**References:**

<https://docs.python.org/3/library/tkinter.html>

http://www.nltk.org/index.html