PROJECT -REVIEW 1

FIND YOUR WORD

**B PRAJEETH - 205002062**

**HARISH A - 205002038**

**PROBLEM STATEMENT:**

Efficient way to search for words in a document using trees as the data structures for the search algorithms.

It is customary to take loads of manual time and labor for searching of words/keywords in a document. An efficient and comfortable approach to this inconvenience is to have a word searcher to find the required keywords/words in the given document or any other case with this issue.

**APPROACH:**

We propose on implementing this through the data structure “TREE”. The Tree of choice for implementing our problem is by using AVL trees. The reason of choosing AVL trees is that this tree has the best search time complexity comparing the other tree structures available to us.

This application first segregates the words using the size of those words as small letters are accessed more frequently than others. This segregation is done based on the AVL trees. By creating nodes of these tree with size of the word as key, those nodes will have an inner tree in which the original words are sorted based on existing AVL algorithm.

As they are first classified based on the number of size and then with the words, the search timing for the problem statement will be reduced drastically.

For example,

Let’s have a word “Hello” to be searched in a document. The program first checks the size of the input then goes to the appropriate node which have the key as 5. After reaching the node 5, the node will have a AVL tree in which the words are stored. It searches for the node where “Hello” is present. If that node 5 have the node “Hello” it will be found and shows where this word presents in the document.

**FLOW DIAGRAM:**

Diagram

Description automatically generated

**WORK ALLOCATION:**

As our application consist of two trees (having inner tree – segregating the original word and outer tree – for segregating the words based on length).

**HARISH A – SPLAY TREES, GUI, CONNECTING FRONT AND AND BACK END**

**B PRAJEETH – AVL TREE, BST TREES,ANALYSIS**

**SOURCE CODE:**

#AVL trees final

class NODE2:

def \_\_init\_\_(self,length):

self.val = length

self.tree = None

self.right= None

self.left = None

self.height=1

class NODE:

def \_\_init\_\_(self,data,pos):

self.val = data

self.data=[data]

self.right=None

self.left=None

self.pos = [pos]

self.height=1

def get\_height(node):

if(node == None):

return 0

else:

return node.height

def insert(root2, key ,pos):

if not root2:

return NODE(key,pos)

elif(key == root2.val):

root2.data.append(key)

root2.pos.append(pos)

elif(key == root2.val):

root2.data.append(key)

root2.pos.append(pos)

elif key < root2.val:

root2.left = insert(root2.left, key, pos)

else:

root2.right = insert(root2.right, key, pos)

root2.height=1+max(get\_height(root2.right), get\_height(root2.left))

balance=check\_balance(root2)

if balance > 1 and key < root2.left.val:

return rightRotate(root2)

if balance < -1 and key > root2.right.val:

return leftRotate(root2)

if balance > 1 and key > root2.left.val:

root2.left = leftRotate(root2.left)

return rightRotate(root2)

if balance < -1 and key < root2.right.val:

root2.right = rightRotate(root2.right)

return leftRotate(root2)

return root2

def check\_balance(root):

if not root:

return 0

return (get\_height(root.left) - get\_height(root.right))

def rightRotate(root):

new=root.left

temp=new.right

new.right=root

root.left=temp

root.height=1+max(get\_height(root.right), get\_height(root.left))

new.height=1+max(get\_height(new.right), get\_height(new.left))

return new

def leftRotate(root):

new=root.right

temp=new.left

new.left=root

root.right=temp

root.height=1+max(get\_height(root.right), get\_height(root.left))

new.height=1+max(get\_height(new.right), get\_height(new.left))

return new

def insert2(root,key2,key,pos):

if not root:

root = NODE2((key))

root.tree = insert(root.tree,key2,pos)

return root

elif((key) == root.val):

root.tree = insert(root.tree, key2, pos)

elif ((key) < root.val):

root.left = insert2(root.left, (key2), key, pos)

else:

root.right = insert2(root.right, key2, key, pos)

root.height=1+max(get\_height(root.right), get\_height(root.left))

balance=check\_balance(root)

if balance > 1 and (key) < root.left.val:

return rightRotate(root)

if balance < -1 and (key) > root.right.val:

return leftRotate(root)

if balance > 1 and (key) > root.left.val:

root.left = leftRotate(root.left)

return rightRotate(root)

if balance < -1 and (key) < root.right.val:

root.right = rightRotate(root.right)

return leftRotate(root)

return root

def search(root,key):

if root is None or root.val == key:

return root

if root.val < key:

return search(root.right,key)

return search(root.left,key)

def preOrder(root):

def preOrder2(root):

if(root == None):

return 0

print((root.val))

preOrder2(root.left)

preOrder2(root.right)

if(root == None):

return 0

print((root.val))

preOrder2(root.tree)

preOrder(root.left)

preOrder(root.right)

# Preorder Traversal

file = open("check.txt","r")

read=file.read()

ls=read.split()

root = None

for i in range(len(ls)):

ls[i] = ls[i].lower()

root= insert2(root, ls[i] ,len(ls[i]), i+1)

#print(ls)

print("PREORDER TRAVERSAL : ")

print("")

preOrder(root)

key= input("Enter search value = ")

key = key.lower()

root2 = search(root, len(key))

position = search(root2.tree, key)

print(position.pos)

**OUTPUT:**

Timeline

Description automatically generated with low confidence

A picture containing text

Description automatically generated

**Text

Description automatically generated**

**SOURCE CODE:**

#splay trees

class NODE2:

def \_\_init\_\_(self, length):

self.val = length

self.tree = None

self.right = None

self.left = None

class NODE:

def \_\_init\_\_(self, data, pos):

self.val = data

self.data=[data]

self.right = None

self.left = None

self.pos = [pos]

def splay\_right(node):

new=node.left

temp=new.right

new.right=node

node.left=temp

return new

def splay\_left(node):

new=node.right

temp=new.left

new.left=node

node.right=temp

return new

def insert(root2, key, pos):

if not root2:

return NODE(key,pos)

elif(key == root2.val):

root2.data.append(key)

root2.pos.append(pos)

elif key < root2.val:

root2.left = insert(root2.left, key, pos)

root2 = splay\_right(root2)

else:

root2.right = insert(root2.right, key, pos)

root2 = splay\_left(root2)

return root2

def insert2(root, key2, key, pos):

if not root:

root = NODE2((key))

root.tree = insert(root.tree,key2,pos)

return root

elif((key) == root.val):

root.tree = insert(root.tree, key2, pos)

elif ((key) < root.val):

root.left = insert2(root.left, (key2), key, pos)

root = splay\_right(root)

else:

root.right = insert2(root.right, key2, key, pos)

root = splay\_left(root)

return root

def search(root, key):

if(root == None or root.val == key):

return root

if(root.val < key):

root.right = search(root.right, key)

root = splay\_left(root)

if(root.val > key):

root.left = search(root.left, key)

root = splay\_right(root)

return root

def preOrder(root):

def preOrder2(root):

if(root == None):

return 0

print((root.val))

preOrder2(root.left)

preOrder2(root.right)

if(root == None):

return 0

print((root.val))

preOrder2(root.tree)

preOrder(root.left)

preOrder(root.right)

file = open("check.txt","r")

read=file.read()

ls=read.split()

root = None

for i in range(len(ls)):

ls[i] = ls[i].lower()

root= insert2(root, ls[i] ,len(ls[i]), i+1)

#print(ls)

print("PREORDER TRAVERSAL : ")

print("")

preOrder(root)

key= input("Enter search value = ")

key = key.lower()

root2 = search(root, len(key))

position = search(root2.tree, key)

print(position.pos)

**OUTPUT:**

Timeline

Description automatically generated with low confidence

**A picture containing text

Description automatically generated**

**Text

Description automatically generated**

**SOURCE CODE:**

#BST trees

class NODE2:

def \_\_init\_\_(self, length):

self.val = length

self.tree = None

self.right = None

self.left = None

class NODE:

def \_\_init\_\_(self, data, pos):

self.val = data

self.data=[data]

self.right = None

self.left = None

self.pos = [pos]

def insert(root2, key ,pos):

if not root2:

return NODE(key,pos)

elif(key == root2.val):

root2.data.append(key)

root2.pos.append(pos)

elif key < root2.val:

root2.left = insert(root2.left, key, pos)

else:

root2.right = insert(root2.right, key, pos)

return root2

def insert2(root,key2,key,pos):

if not root:

root = NODE2((key))

root.tree = insert(root.tree,key2,pos)

return root

elif((key) == root.val):

root.tree = insert(root.tree, key2, pos)

elif ((key) < root.val):

root.left = insert2(root.left, (key2), key, pos)

else:

root.right = insert2(root.right, key2, key, pos)

return root

def search(root,key):

if root is None or root.val == key:

return root

if root.val < key:

return search(root.right,key)

return search(root.left,key)

def preOrder(root):

def preOrder2(root):

if(root == None):

return 0

print((root.val))

preOrder2(root.left)

preOrder2(root.right)

if(root == None):

return 0

print((root.val))

preOrder2(root.tree)

preOrder(root.left)

preOrder(root.right)

# Preorder Traversal

file = open("check.txt","r")

read=file.read()

ls=read.split()

root = None

for i in range(len(ls)):

ls[i] = ls[i].lower()

root= insert2(root, ls[i] ,len(ls[i]), i+1)

#print(ls)

print("PREORDER TRAVERSAL : ")

print("")

preOrder(root)

key= input("Enter search value = ")

key = key.lower()

root2 = search(root, len(key))

position = search(root2.tree, key)

print(position.pos)

**OUTPUT:**

Timeline

Description automatically generated with low confidence

A picture containing text

Description automatically generated

**Text

Description automatically generated**