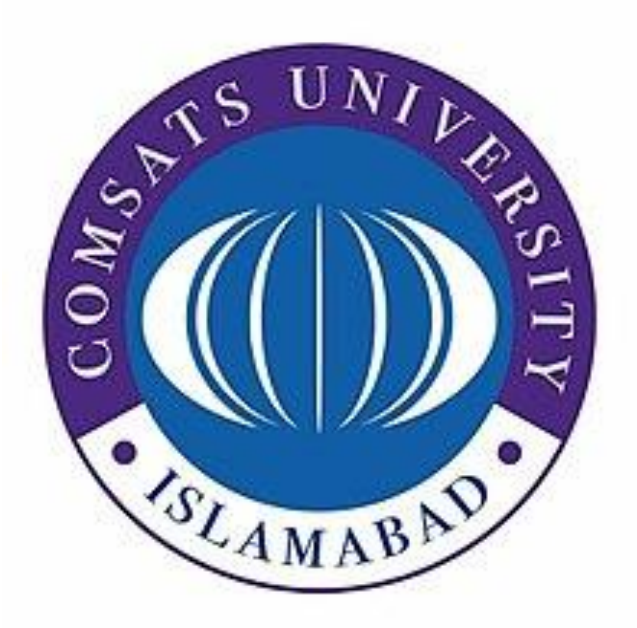
COMSATS UNIVERSITY ISLAMABAD

Park Road Tarlai Kalan, Islamabad



Data Structures and Algorithm

Project Report

Submitted To:

Sir Inayat Ur Rehman

Prepared By:

Name:	Registration No:
M Shaharyar	SP20-BCS-065
Uzair Ahmad	SP20-BCS-100
Tayyab Habib	SP20-BCS-069
Shahzaib Khurshid	SP20-BCS-066

Introduction

Background

This project incorporates the concept of AVL Tree for developing a simple console-based language translator that translates English words to French and also vice versa. AVL tree was used for this project because of its minimal time complexity while performing functions such as searching, insertion and deletion. The data is fed into the tree from files by file handling and then all other operations are performed directly on the tree. All the primary operations of the tree as well as the helper functions has been coded in a very modular way to aid further usage. Prebuild function from libraries were also used and new helper function were also coded wherever felt necessary. Proper messages are displayed everywhere to help the user use the program as well as the data output has been formatted in orderly fashion to aid the user in reading.

Concepts Learned

This project helped us learn some very vital concepts regarding data structures and programming in general. Some of the key concepts that we learned are mention below:

Concept 1: AVL Tree

All the basic operation of the AVL Tree. Each node of the tree stores a word, its meaning, frequency etc.

Concept 2: Linked List

Basic operations of linked list such as insertion at start, end and specific position. And deletion operations

Concept 3: Recursion

Better understanding of how recursive calls work and how we can use them to traverse trees.

General Concepts:

- Core concepts of different data structures and their basic operations
- Identifying and using the data structure that best suit the problem at hand
- Basic differences between different tree structures and differences in their implementation and time complexities for different operations
- Critically thinking of ways to come up with new possible algorithms for solution of a problem and incorporate the one that uses the least resources
- A better understanding of how recursive calls to functions work and how to manipulate them in our favor.
- It's always good to build small segments of code and keep testing them while we code further, it helps avoid problems with debugging in the later stages of development
- Diving the code in module is always the best approach as it makes debugging the code easy, also it can be reused and makes understanding it easier for everyone else

Problem statement

Develop a language translator that translates English words to French and vice versa incorporating AVL trees.

Project Description

This project incorporates the concept of AVL Tree for developing a simple console-based language translator that translates English words to French and also vice versa. AVL tree was used for this project because of it's minimal time complexity while performing functions such as searching, insertion and deletion. The data is fed into the tree from files by file handling and then all other operations are performed directly on the tree. All the primary operations of the tree as well as the helper functions has been coded in a very modular way to aid further usage. Prebuild function from libraries were also used and new helper function were also coded wherever felt necessary. Checks for correct input has also been incorporated to aid novice users in feeding correct data. Proper messages are displayed everywhere to help the user use the program as well as the data output has been formatted in orderly fashion to aid the user in reading.

For input, the user has to be careful in some places, for example , for deleting a word from the list the valid syntax is "d wordToBeDeleted", if the user inputs just the word or something other than this, the program would tell him the input was correct him and will show his the correct syntax. Same goes for insertion. The program can also build tree from a given traversal which can fed into the program from a file. It validates the traversal and then build the tree according to it. The project has been tested on a data of approximately 600 words and seems to run error free and very smoothly. And should withstand even larger data as long as the code remains intact.

Functionalities

Function prototype	void insertFile(string filename)
Function Description	Insert the words, their meanings and the part of speech into the AVL tree
Pre-condition	Filename be initialized
Post-condition	A complete balanced AVL Tree with words as nodes

Function prototype	AVLNode* insert(AVLNode *node, string key, string mean, string pof_speech)
Function Description	Insert a new node into the AVL tree
Pre-condition	Root of the tree be initialize and attributes of the node
Post-condition	A node added to the existing tree

Function prototype	AVLNode* createNode(string key,string mean,string

	pof_speech)
Function Description	Create a new node
Pre-condition	
Post-condition	Pointer to the newly created node

Function prototype	int getHeight(AVLNode *node)
Function Description	Calculate the height of a node
Pre-condition	Node pointer
Post-condition	Interger height

Function prototype	int calBalance(AVLNode *node)
Function Description	Calculate the balance of a node
Pre-condition	Node pointer
Post-condition	Int balance

Function prototype	AVLNode* minNode(AVLNode *node)
Function Description	Calculate the min value node for a given sub-tree
Pre-condition	Node pointer
Post-condition	

Function prototype	int max(int a, int b){
Function Description	Calculate the larger value from given two
Pre-condition	
Post-condition	

Function prototype	AVLNode* rotateLeft(AVLNode *node)
Function Description	Rotate the subtree to the left
Pre-condition	Node pointer
Post-condition	Balance of partially balanced tree

Function prototype	AVLNode* rotateRight(AVLNode *node)
Function Description	Rotate the subtree to the right
Pre-condition	Node pointer
Post-condition	Balanced to partially balanced tree

Function prototype	AVLNode* frenchCreate(AVLNode *node)
Function Description	Create an AVL Tree for French to English from existing AVL Tree
Pre-condition	Existing AVL Tree
Post-condition	AVL tree with French words as keys

Function prototype	void createFNodeStart(int f, AVLNode *node, FNode *&head)
Function Description	Add a node at the start of the list
Pre-condition	Exisiting list
Post-condition	Change head pointer to the list

Function prototype	void createFNode(int freq, AVLNode *node, FNode *&head, FNode *&tail)
Function Description	Add a node to the end of the list if head not null other wise creates head
Pre-condition	Existing list
Post-condition	Change tail pointer to the list

Function prototype	void createFNodeSpecific(int f , AVLNode *node,FNode *&head){
Function Description	Add a node to specific location in the list
Pre-condition	Existing list
Post-condition	

Function prototype	void sortFrequency(AVLNode *node,AVLNode *root,FNode *&head,FNode *&tail)
Function Description	Creates a linked list storing the pointers of all the nodes of the tree in order of decreasing frequency
Pre-condition	Existing AVL Tree
Post-condition	Sorted Double Linked List

Function prototype	void inOrderDisplay(AVLNode *node)
Function Description	Display the inorder traversal of the tree
Pre-condition	Root of the AVL Tree
Post-condition	

Function prototype	void displaySortedList(FNode *node)
Function Description	Display the words in decreasing order of frequency
Pre-condition	A sorted list
Post-condition	

Function prototype	void displayPOF(AVLNode *node,string pof,bool &check)
Function Description	Display the words by specific part of speech
Pre-condition	Existing AVL Tree, string part of speech
Post-condition	

Function prototype	int compare(string key, string to_check)
Function Description	Compare two strings
Pre-condition	
Post-condition	

Function prototype	void searchWord(AVLNode *node,string key)
Function Description	Search and display a specific word from the list
Pre-condition	Existing AVL Tree
Post-condition	

Function prototype	AVLNode* deleteNode(AVLNode *node, string key, bool &check)
Function Description	Delete node from AVL Tree and balance it again
Pre-condition	Existing AVL Tree
Post-condition	AVL Tree with the specific node deleted and rebalanced

Function prototype	void readFile(string filename)
Function Description	Read traversals from file
Pre-condition	File name string initialized
Post-condition	

Function prototype	void createTree(vector<string> v_arr1, vector<string> v_arr2)
Function Description	Create tree from a given traversal
Pre-condition	Inorder and preorder traversals
Post-condition	A complete tree from the traversals

Function prototype	bool checkTree(vector<string> prearray)
Function Description	Check the validity of the given traversals
Pre-condition	Preorder of the tree
Post-condition	Bool value

Test Plan

Most of the function takes pointers and returns pointers in addition with other data types and returns mostly pointers which can't be written in test plan. They have to be tested directly on machine by printing the data valued stored at these pointers.