Project Proposal Form

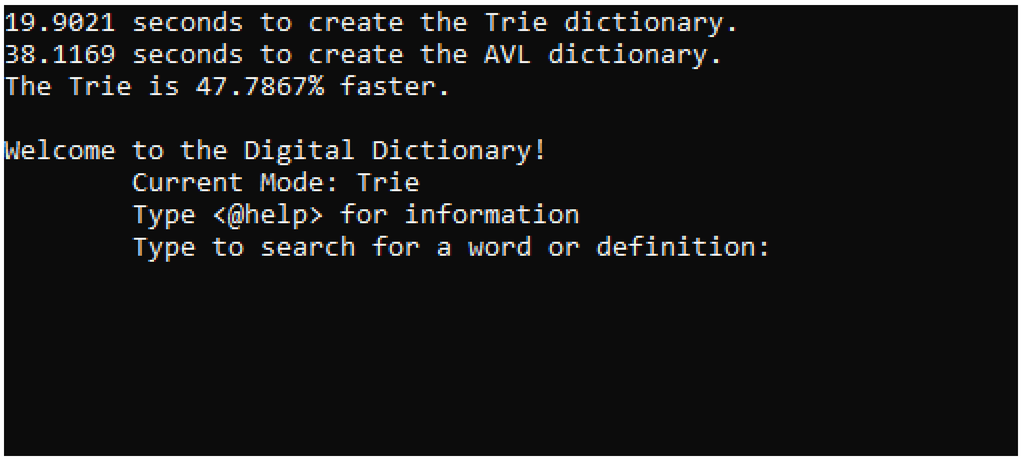
BSCS 10 AB – CS250 – DSA

Project Title	Digital Dictionary
Team Members	Faran Ahmad Khan - 333858 - A Muhammad Haseeb - 342608 - A
Description	Digital Dictionary is a console based application that will read the file having a list of words, their meanings/synonyms and parts of speech. Users can search a word for the meaning/synonym and searching will be done using ***AVL*** and ***Trie*** Trees. Other utilities will be provided in the application. Digital Dictionary will mainly cover the implementations of the concepts of ***Binary*** and ***m-ary*** Trees. Technologies used: ● C / C++ ● Database [for words & their meanings]

.cpp file:

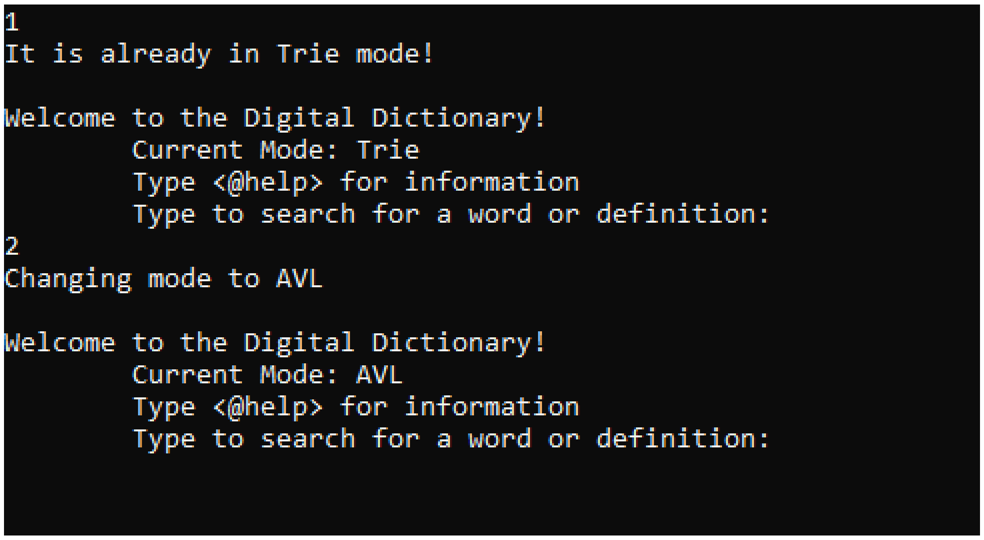
```
/* Digital Dictionary via AVL & Trie trees
 * Project (C++ console based)
 * Course:
 *      ~ CS-250 Data Structures & Algorithms
 * Group Member(s):
 *      ~ Faran - 333858
 *      ~ Haseeb - 342608
 * Course Instructor:
 *     ~ Dr. Yasir Faheem
 * Description:
 *      ~ This is a C++ console based project that allows
 *      ~ the user to search a word just like an Online Dictionary.
 *      ~ Application will return its meaning (synonym), part of speech,
 *      ~ and history of word (if any). The user can compare between
 *      ~ the Data Structures used (AVL Tree, Trie Tree) which is faster.
 */


 //libraries are included

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <chrono>
#include "Dictionary.h"

using namespace std;

//to measure execution time
using namespace chrono;



//Parsing algorithm that read dictionary and add to Trie
Trie* readFiletoTrie(Trie* root) {
        ifstream file;
        file.open("Dictionary.csv", ios::in);

        string line, word, partOfSpeech, definition;
        while (getline(file, line)) {

                //stream of entire line of csv
                stringstream s(line);

                getline(s, word, ',');
                //word without quotations
                word.erase(remove(word.begin(), word.end(), '\"'), word.end());

                getline(s, partOfSpeech, ',');
                //Part of Speech without quotations
                partOfSpeech.erase(remove(partOfSpeech.begin(), partOfSpeech.end(), '\"'), partOfSpeech.end());

                getline(s, definition);
                //definition without quotations
                definition.erase(remove(definition.begin(), definition.end(), '\"'), definition.end());

                //add to Trie Tree
                insertTrie(root, word, partOfSpeech, definition);
        }
        file.close();
        return root;
}

//Parsing algorithm that read dictionary and add to AVL
Node* readFiletoAVL(Node* root) {
        fstream file;
	file.open("Dictionary.csv", ios::in);
	string line, word, partOfSpeech, definition;
	while (getline(file, line)) {
		stringstream s(line);
		getline(s, word, ',');
		word.erase(remove(word.begin(), word.end(), '\"'), word.end());

		getline(s, partOfSpeech, ',');
		partOfSpeech.erase(remove(partOfSpeech.begin(), partOfSpeech.end(), '\"'), partOfSpeech.end());

		getline(s, definition);
		definition.erase(remove(definition.begin(), definition.end(), '\"'), definition.end());

		//add to AVL Tree
		root = insert(root, word, partOfSpeech, definition);
	}
	file.close();
	return root;
}

//compares execution time between using Trie and AVL
void timeComp(microseconds timeTrie, microseconds timeAVL) {
	float percentAVL = ((timeTrie.count() - timeAVL.count()) / (float)timeTrie.count()) * 100.0;
	float percentTrie = ((timeAVL.count() - timeTrie.count()) / (float)timeAVL.count()) * 100.0;
	if (timeTrie.count() > timeAVL.count()) {
		cout << "The AVL Tree is " << percentAVL << "% faster." << endl;
	}
	else {
		cout << "The Trie is " << percentTrie << "% faster." << endl;
	}
}


//search definition
void definitionFinder(Node* root, Trie* root2, string mode, string input) {
	string def = "DNE";
	microseconds findDefDurationTrie, findDefDurationAVL;

	if (mode == "Trie" || mode == "Comparison") {
		auto start = high_resolution_clock::now();

		//returns definition
		def = findDefinitionTrie(root2, input);

		auto stop = high_resolution_clock::now();
		findDefDurationTrie = duration_cast<microseconds>(stop - start);
	}
	if (mode == "AVL" || mode == "Comparison") {
		auto start2 = high_resolution_clock::now();

		//returns definition
		def = findDefinitionAVL(root, input);

		auto stop2 = high_resolution_clock::now();
		findDefDurationAVL = duration_cast<microseconds>(stop2 - start2);
	}

	//if word does not exist
	if (def == "DNE") {
		cout << "Your input, '" << input << "', was not found." << endl;

		if (mode == "Trie" || mode == "Comparison") {
			cout << "It took " << findDefDurationTrie.count() << " microseconds using Trie mode." << endl;
		}
		if (mode == "AVL" || mode == "Comparison") {
			cout << "It took " << findDefDurationAVL.count() << " microseconds using AVL mode." << endl;
		}
		if (mode == "Comparison") {
			timeComp(findDefDurationTrie, findDefDurationAVL);
		}
	}
	//if word exists
	else {
		cout << def << endl;
		if (mode == "Trie" || mode == "Comparison") {
			cout << "It took " << findDefDurationTrie.count() << " microseconds to find the defintion using Trie mode." << endl;
		}
		if (mode == "AVL" || mode == "Comparison") {
			cout << "It took " << findDefDurationAVL.count() << " microseconds to find the defintion using AVL mode." << endl;
		}
		if (mode == "Comparison") {
			timeComp(findDefDurationTrie, findDefDurationAVL);
		}
	}
}

//start main()

int main() {
	//pointer to AVL node
	Node* rootAVL = NULL;
	//pointer to Trie node
	Trie* rootTrie = NULL;

	//read file and measure time to load dictionary in Trie Tree
	auto start = high_resolution_clock::now();
	rootTrie = readFiletoTrie(rootTrie);
	auto stop = high_resolution_clock::now();
	auto TrieDuration = duration_cast<microseconds>(stop - start);
	cout << TrieDuration.count() / 1000000.0 << " seconds to create the Trie dictionary." << endl;

	//read file and measure time to load dictionary in AVL Tree
	auto start2 = high_resolution_clock::now();
	rootAVL = readFiletoAVL(rootAVL);
	auto stop2 = high_resolution_clock::now();
	auto AVLDuration = duration_cast<microseconds>(stop2 - start2);
	cout << AVLDuration.count() / 1000000.0 << " seconds to create the AVL dictionary." << endl;

	timeComp(TrieDuration, AVLDuration);

	string mode = "Trie";
	string input = "";
	while (input != "@Exit") {
		cout << "\nWelcome to the Digital Dictionary!";
		cout << "\n\tCurrent Mode: " << mode;
		cout << "\n\tType <@help> for information";
		cout << "\n\tType to search for a word or definition:" << endl;
		cin >> input;

		if (input == "@Help" || input == "@help") {
			cout << "\nInstructions:";
			cout << "\n\tType <a word> to check if it is a word and find the definition";
			cout << "\n\t\tNote: You must capitalize the first letter of the searched word.";
			cout << "\n\tType <1> for Trie dictionary mode";
			cout << "\n\tType <2> for AVL dictionary mode";
			cout << "\n\tType <3> for a comparison mode of Trie and AVL Structures";
			cout << "\n\tType <@Exit> to exit the program anytime" << endl;
		}
		//change to Trie mode
		else if (input == "1") {
			if (mode == "Trie") {
				cout << "It is already in Trie mode!" << endl;
			}
			else {
				mode = "Trie";
				cout << "Changing mode to Trie\n";
			}
		}
		//change to AVL mode
		else if (input == "2") {
				if (mode == "AVL") {
					cout << "It is already in AVL mode!" << endl;
				}
				else {
					mode = "AVL";
					cout << "Changing mode to AVL\n";
				}
			}
			//change to comparison mode
			else if (input == "3") {
				if (mode == "Comparison") {
					cout << "It is already in Comparison mode!" << endl;
				}
				else {
					mode = "Comparison";
					cout << "Changing mode to Comparison\n";
				}
			}
			//exit program
			else if (input == "@Exit" || input == "@exit") {
				return 0;
			}
			//finds definition
			else {
				definitionFinder(rootAVL, rootTrie, mode, input);
			}
		}
		return 0;

}//end main()
```

The dictionary is loaded.



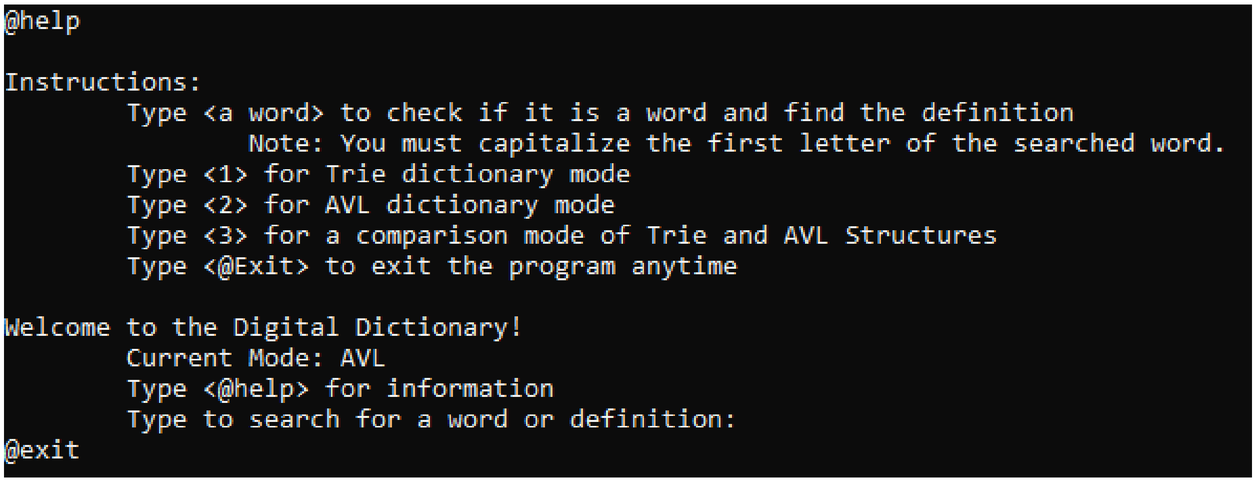
```
19.9021 seconds to create the Trie dictionary.
38.1169 seconds to create the AVL dictionary.
The Trie is 47.7867% faster.

Welcome to the Digital Dictionary!
        Current Mode: Trie
        Type <@help> for information
        Type to search for a word or definition:
```

Checking modes.



```
1
It is already in Trie mode!

Welcome to the Digital Dictionary!
        Current Mode: Trie
        Type <@help> for information
        Type to search for a word or definition:
2
Changing mode to AVL

Welcome to the Digital Dictionary!
        Current Mode: AVL
        Type <@help> for information
        Type to search for a word or definition:
```

Checking help and exit commands.



```
@help

Instructions:
        Type <a word> to check if it is a word and find the definition
                Note: You must capitalize the first letter of the searched word.
        Type <1> for Trie dictionary mode
        Type <2> for AVL dictionary mode
        Type <3> for a comparison mode of Trie and AVL Structures
        Type <@Exit> to exit the program anytime

Welcome to the Digital Dictionary!
        Current Mode: AVL
        Type <@help> for information
        Type to search for a word or definition:
@exit
```

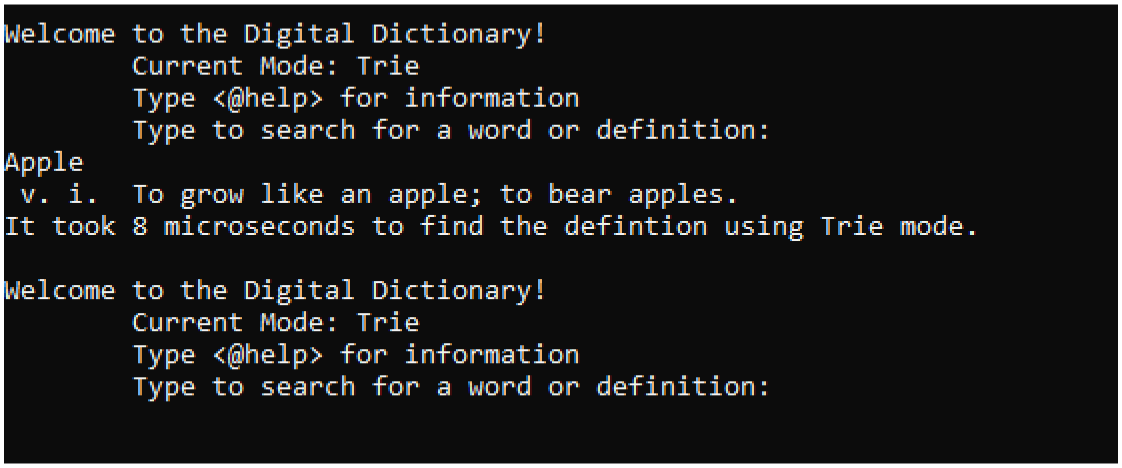
.h file:

```
#ifndef Dictionary_H
#define Dictionary_H

//libraries are included

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <chrono>

using namespace std;


//AVL Tree Data Structure


//node AVL
class Node {
public:
	Node* left;
	Node* right;
	string word;
	string partOfSpeech;
	string definition;
	int height;
};

//max function
int max(int first, int second) {
	if (first > second)
		return first;
	else
		return second;
}
//returns the height
int getHeight(Node* temp) {
	if (temp == NULL)
		return 0;
	return temp->height;
}

//This function will create new AVL node and initialize it
Node* createAvlNode(string input_word, string input_partOfSpeech, string input_definition) {
	Node* node = new Node();
	node->left = NULL;
	node->right = NULL;

	node->word = input_word;
	node->partOfSpeech = input_partOfSpeech;
	node->definition = input_definition;
	node->height = 1;
	return node;
}

//left rotation
Node* rotateLeft(Node* node) {

	//Set and do left rotation

	Node* grandchild = node->right->left;
	Node* newParent = node->right;
	newParent->left = node;
	node->right = grandchild;

	//Update height
	node->height = max(getHeight(node->left), getHeight(node->right)) + 1;
```

```
        newParent->height = max(getHeight(newParent->left), getHeight(newParent->right)) + 1;

        //Return the new parent
        return newParent;
}

//right rotation
Node* rotateRight(Node* node) {
        //Set and do right rotation
        Node* grandchild = node->left->right;
        Node* newParent = node->left;
        newParent->right = node;
        node->left = grandchild;

        //Update height
        node->height = max(getHeight(node->left), getHeight(node->right)) + 1;
        newParent->height = max(getHeight(newParent->left), getHeight(newParent->right)) + 1;

        //Return the new parent
        return newParent;
}

//calculate balance factor
int balanceFactor(Node* temp) {
        if (temp == NULL)
                return 0;
        return getHeight(temp->left) - getHeight(temp->right);
}

//insert function for AVL
Node* insert(Node* node, string input_word, string input_partOfSpeech, string input_definition) {

        //Perform recursive insertion
        if (node == NULL) {
                Node* temp_node = createAvlNode(input_word, input_partOfSpeech, input_definition);
                return temp_node;
        }
        //goes to left
        if (input_word < node->word)
                node->left = insert(node->left, input_word, input_partOfSpeech, input_definition);

        //goes to right
        else if (input_word > node->word)
                node->right = insert(node->right, input_word, input_partOfSpeech, input_definition);
        else {
                return node;
        }
        //Again calculate the height of the node
        node->height = max(getHeight(node->left), getHeight(node->right)) + 1;

        //Determine the balance factor to see if rotations are necessary
        int balance_factor = balanceFactor(node);


        /*********Rotation Cases**********/

        //Left Left
        if (balance_factor > 1 && input_word < node->left->word)
                return rotateRight(node);
        //Right Right
        if (balance_factor < -1 && input_word > node->right->word)
                return rotateLeft(node);
        //Left Right
        if (balance_factor > 1 && input_word > node->left->word) {
                node->left = rotateLeft(node->left);
                return rotateRight(node);
        }
        //Right Left
        if (balance_factor < -1 && input_word < node->right->word) {
                node->right = rotateRight(node->right);
```

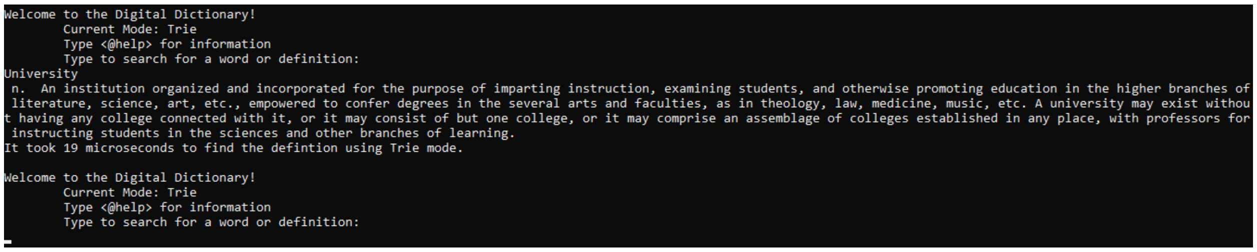
```
			return rotateLeft(node);
		}

		//Insertion successful, return node
		return node;
}

//search function for AVL
string findDefinitionAVL(Node* current, string input) {
		if (current == NULL) {
			return "DNE";
		}
		else {
			//base case
			if (current->word == input) {
				return (current->partOfSpeech + " " + current->definition);
			}
			//recursive cases

			else if (current->word > input) {
				return findDefinitionAVL(current->left, input);
			}
			else if (current->word < input) {
				return findDefinitionAVL(current->right, input);
			}
		}
		return "DNE";
}

/*----------------------------------------------------------------------------------*/

//Trie Tree Data Structure



//node Trie
struct Trie {
		//since there are 26 alphabets
		Trie* child[26];
		string meaning;
		bool isWord;
};


//new node
Trie* getNewTrieNode()
{
		Trie* node = new Trie;

		for (int i = 0; i < 26; i++) {
			node->child[i] = NULL;
		}
		node->isWord = false;
		return node;
}

//This insert function will be used by the parser we created to insert string
void insertTrie(Trie*& root, string str, const string& type, const string& def)
{
		if (root == NULL)
			root = getNewTrieNode();

		Trie* temp = root;

		for (int i = 0; str[i] != '\0'; i++) {

			//provide proper indexing from 0-25
			int index = tolower(str[i]) - 'a';

			//handle the error
			if (index > 25 || index < 0)
				continue;

			if (temp->child[index] == NULL) {
				temp->child[index] = getNewTrieNode();
			}
			temp = temp->child[index];
		}


		// store the meaning
		temp->isWord = true;
		temp->meaning = type + " " + def;

}

//search function for Trie
string findDefinitionTrie(Trie* root, const string& word)
{
		// If dictionary is empty
		if (root == nullptr) {
			cout << "root is empty";
			return "DNE";
		}

		Trie* temp = root;


		// Search a word in the Trie
		for (int i = 0; word[i] != '\0'; i++) {

			//provide proper indexing from 0-25
			int index = tolower(word[i]) - 'a';

			//handle the error
			if (index > 25 || index < 0)
				continue;

			if (temp->child[index] == NULL)
				return "DNE";
			temp = temp->child[index];
		}

		//return its meaning
		if (temp->isWord)
			return temp->meaning;
		return "DNE";
}


#endif /* Dictionary_H */
```

Word “Apple” is entered using the Trie Tree.



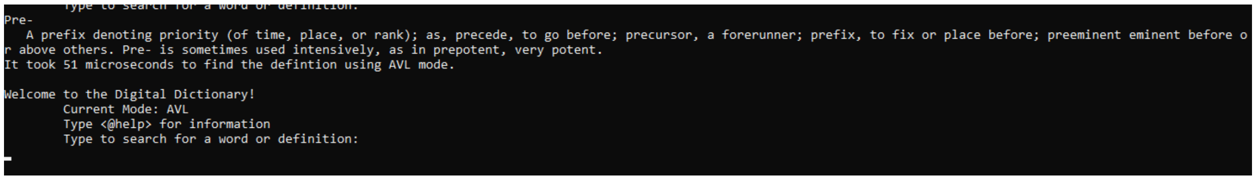
```
Welcome to the Digital Dictionary!
        Current Mode: Trie
        Type <@help> for information
        Type to search for a word or definition:
Apple
 v. i.  To grow like an apple; to bear apples.
It took 8 microseconds to find the defintion using Trie mode.

Welcome to the Digital Dictionary!
        Current Mode: Trie
        Type <@help> for information
        Type to search for a word or definition:
```

Word “University” is entered using the Trie Tree.



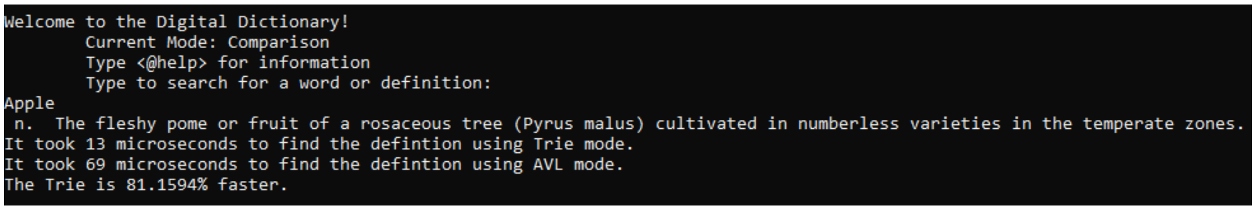
```
Welcome to the Digital Dictionary!
        Current Mode: Trie
        Type <@help> for information
        Type to search for a word or definition:
University
 n.  An institution organized and incorporated for the purpose of imparting instruction, examining students, and otherwise promoting education in the higher branches of
 literature, science, art, etc., empowered to confer degrees in the several arts and faculties, as in theology, law, medicine, music, etc. A university may exist withou
t having any college connected with it, or it may consist of but one college, or it may comprise an assemblage of colleges established in any place, with professors for
 instructing students in the sciences and other branches of learning.
It took 19 microseconds to find the defintion using Trie mode.

Welcome to the Digital Dictionary!
        Current Mode: Trie
        Type <@help> for information
        Type to search for a word or definition:
```

Prefix “Pre-” is searched using the AVL Tree.



```
        Type to search for a word or definition:
Pre-
   A prefix denoting priority (of time, place, or rank); as, precede, to go before; precursor, a forerunner; prefix, to fix or place before; preeminent eminent before o
r above others. Pre- is sometimes used intensively, as in prepotent, very potent.
It took 51 microseconds to find the defintion using AVL mode.

Welcome to the Digital Dictionary!
        Current Mode: AVL
        Type <@help> for information
        Type to search for a word or definition:
```

Word “Apple” is searched in Comparison mode.



```
Welcome to the Digital Dictionary!
        Current Mode: Comparison
        Type <@help> for information
        Type to search for a word or definition:
Apple
 n.  The fleshy pome or fruit of a rosaceous tree (Pyrus malus) cultivated in numberless varieties in the temperate zones.
It took 13 microseconds to find the defintion using Trie mode.
It took 69 microseconds to find the defintion using AVL mode.
The Trie is 81.1594% faster.
```