Project: Text Editor

**Group Members:**

Abdul Wahab(21f\_9457)

Amanullah(21f\_9517)

Muhammad Ahmad(21f\_9498)

**Source Files:**

**Huffman.cpp**

#include<iostream>

#include<string>

using namespace std;

//Node for Linked List to Store the data and its code

class LL\_node {

public:

char data;

string encode;

LL\_node\* next;

LL\_node() {

data = '\0';

encode = "\0";

next = NULL;

}

};

//Linked List ADT

class Linked\_List {

private:

LL\_node\* head, \* tail;

public:

Linked\_List() {

head = tail = NULL;

}

bool isEmpty() {

if (head == NULL) { return true; }

else { return false; }

}

bool Insert(char data, string encode) {

LL\_node\* newNode = new LL\_node;

newNode->data = data;

newNode->encode = encode;

if (isEmpty()) {

head = tail = newNode;

newNode->next = NULL;

return true;

}

tail->next = newNode;

tail = tail->next;

return true;

}

string Search(char Data) {

if (isEmpty()) {

return "\0";

}

else {

LL\_node\* Curr = head;

while (Curr != NULL) {

if (Curr->data == Data) {

return Curr->encode;

}

Curr = Curr->next;

}

}

}

LL\_node\* getHead() {

return head;

}

};

//Node for Priority Queue

class node {

public:

char data;//Alphabet

int freq;

node\* Next;

node\* Left;

node\* Right;

node()

{

data = 0;

freq = 0;

Next = NULL;

Left = NULL;

Right = NULL;

}

};

//Priority Queue Class

class Priority\_Queue {

private:

node\* front;//head

Linked\_List Encoding;

public:

Priority\_Queue() {

front = NULL;

}

node\* getFront() { return front; }

//isEmpty Function to Check the Queue is Empty or Not

bool isEmpty() {

if (front == NULL) { return true; }

else { return false; }

}

//Insertion of Character and its frequency in priority queue

void EnQueue(char Val, int frequency)

{

node\* newNode = new node;

newNode->data = Val;

newNode->freq = frequency;

// if the queue is empty

if (isEmpty()) {

front = newNode;

newNode->Next = NULL;

}

Add\_Node(newNode);

}

void Add\_Node(node\* newNode)

{

//if the frequency of charatcer is greater than the front(head) of priority\_queue

if (newNode->freq < front->freq) {

newNode->Next = front;

front = newNode;

}

else {

//Adding Node By function

node\* temp = NULL; //Temprarity Variable for the inserting Node

node\* curr = front;

//check the priority of counts of Sequence charatcer

while (curr->Next != NULL && curr->Next->freq < newNode->freq) {

curr = curr->Next;

}

temp = curr->Next;

curr->Next = newNode;

newNode->Next = temp;

}

}

bool DeQueue() {

if (isEmpty()) {

return false;

}

front = front->Next;

return true;

}

void Making\_Huffman\_Tree\_Step\_by\_Step(node\* temp1, node\* temp2)

{

node\* New\_Node = new node();

New\_Node->freq = (temp1->freq) + (temp2->freq);

New\_Node->Left = temp1;

New\_Node->Right = temp2;

DeQueue();

if (front->Next != NULL) {

DeQueue();

Add\_Node(New\_Node);

}

else {

front = New\_Node;

}

}

void Huffmain\_conding()

{

node\* curr = NULL;

while (front->Next != NULL)

{

curr = front;

node\* temp1 = curr;

node\* temp2 = curr->Next;

Making\_Huffman\_Tree\_Step\_by\_Step(temp1, temp2); //Adding data in Root Node and return the single Node

}

}

void Storing\_Encode(node\* node, string str) {

if (node) {

/\* first recur on left child \*/

Storing\_Encode(node->Left, str + "0");

/\* then print the data of node \*/

if (node->data != NULL) {

LL\_node\* curr = Encoding.getHead();

while (curr != NULL) {

if (curr->data == node->data) {

curr->encode = str;

}

curr = curr->next;

}

}

/\* now recur on right child \*/

Storing\_Encode(node->Right, str + "1");

}

}

void SetLinkedList(string a) {

for (int i = 0; i < a.length(); i++) {

Encoding.Insert(a[i], "0");

}

}

LL\_node\* getLinkedList() {

return Encoding.getHead();

}

};

class Huffman {

private:

Priority\_Queue q;

public:

Huffman(string Str) {

int\* Counts = new int[127];// int array to store the counts of each character

//initializztion the arrays of characters and their counts

for (int i = 0; i < 127; i++) {

Counts[i] = 0;

}

for (int i = 0; i < Str.length(); i++) {

Counts[Str[i]] = Counts[Str[i]] + 1;//storing counts of their character

}

for (int i = 0; i < 127; i++) {

if (Counts[i] != 0) {

q.EnQueue(char(i), Counts[i]);

}

}

q.SetLinkedList(Str);

q.Huffmain\_conding();

string String = "";

q.Storing\_Encode(q.getFront(), String);

}

LL\_node\* getList() {

return q.getLinkedList();

}

};

**Source.cpp**

#include<iostream>

#include<fstream>

#include<string>

#include<array>

#include<conio.h>

#include<Windows.h>

#include"huffman.cpp"

using namespace std;

void gotoxy(int x, int y)

{

COORD coord;

coord.X = x;

coord.Y = y;

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coord);

}

void color(int code) {

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole, code);

}

//COORD GetConsoleCursorPosition(HANDLE hConsoleOutput)

//{

// CONSOLE\_SCREEN\_BUFFER\_INFO cbsi;

// if (GetConsoleScreenBufferInfo(hConsoleOutput, &cbsi))

// {

// return cbsi.dwCursorPosition;

// }

// else

// {

// // The function failed. Call GetLastError() for details.

// COORD invalid = { 0, 0 };

// return invalid;

// }

//}

struct Node {

char ch;

Node\* children[26];//a-z

bool terminal;

};

class Trie {

private:

Node\* root;

array<string,11> arr;// for storing the suggested words

int i = 0;//for the indices of array

//Helper Functions

Node\* createNode(char character) {

Node\* newNode = new Node();

newNode->ch = character;

for (int i = 0; i < 26; i++)

newNode->children[i] = NULL;

newNode->terminal = false;

return newNode;

}

int charIndex(char ch) {//Hash index for char

return ((ch + 7) % 26);

}

void printWords(Node\* node, string str) {

str += node->ch;

if (node->terminal == true)

cout << str << endl;

for (int i = 0; i < 26; i++)

if (node->children[i] != NULL)

printWords(node->children[i], str);

}

void deleteTrie(Node\*& node) {//node is passed by reference

for (int i = 0; i < 26; i++) {

if (node->children[i] != NULL)

deleteTrie(node->children[i]);

}

delete node;

node = NULL;

}

void suggetWords(Node\* node, string prefix, string postfix) {

postfix += node->ch;

if (node->terminal == true) {

if(i<10)

arr[i++]=(prefix + postfix);

}

for (int i = 0; i < 26; i++) {

if (node->children[i] != NULL)

suggetWords(node->children[i], prefix, postfix);

}

}

public:

Trie() {//Constructor

root = NULL;

arr.fill("\0");

}

bool insertWord(string str) {

if (root == NULL) {

root = createNode('\0');

}

Node\* curr = root;

size\_t length = str.length();

for (int i = 0; i < length; i++) {

if (curr->children[charIndex(str[i])] == NULL) {

curr->children[charIndex(str[i])] = createNode(str[i]);

}

curr = curr->children[charIndex(str[i])];

}

if (curr->terminal == true) { return false; }// if a word already exits in the tree then it will return false

else { curr->terminal = true; return true; }// if the word doesn't exits then it will set terminal to true and return

}

void viewDictionary() {//to see all words inside dictionary

if (root != NULL) {

printWords(root, "\0");

}

}

void storeSuggestedWords(string for\_prefix) {

arr.fill("\0");

Node\* curr = root;

for (int i = 0; i < for\_prefix.length(); i++) {

if (curr->children[charIndex(for\_prefix[i])] != NULL) {

curr = curr->children[charIndex(for\_prefix[i])];

}

else {

return;//word not exits

}

}

int count=0;

for (int i = 0; i < 26; i++) {

if (curr->children[i] != NULL)

suggetWords(curr->children[i], for\_prefix, "\0");

else

count++;

}

if (count == 26) { arr[0]=for\_prefix; }

i = 0;

}

void selectSuggestedWords(string &str) {

gotoxy(0, 2);

color(112); cout << " A V A I L A B L E W O R D S "; color(7);

int s = 0;

if (arr[0]=="\0") {

gotoxy(0,3);

color(4); cout << "\nNo other word for that prefix exist\n"; color(7);

system("pause>0");

}

else {

int elements = 0;

char ch = '\0';

int i = 0;

int x = 0;

int y = 2;

color(4);

for (int i = 0; (i<10 && arr[i] != "\0"); i++) {

gotoxy(x, ++y);

cout << arr[i];

elements++;

color(7);

}

y = 2; arr[elements] = "Skip"; cout << "\n"<<arr[elements];

do

{

ch = \_getch();

if (ch == 47) // backslash

{

i = (i + 1) % (elements+1);

s = i;

for (int j = 0; j <=elements; j++)

{

gotoxy(x,++y);

if (i != j)

{

cout << arr[j];

}

else

{

color(4);

cout << arr[i];

color(7);

}

}

y = 2;

}

} while (ch!=9);

}

if (arr[s] == "Skip") { str = "\0"; return; }

str = arr[s];

//system("pause");

}

void printArray() {

for (int i = 0; i < 10; i++) {

cout << arr[i] << " ";

}

}

~Trie()

{

deleteTrie(root);

}

};

void mainMenu(char &choice) {

system("cls");

color(112);

cout << " M A I N M E N U \n\n";

color(7);

cout << "1. Create a new file\n";

cout << "2. Open a file\n";

cout << "0. Exit\n\n";

cout << "Enter your choice : "; cin >> choice;

}

void subMenu(char& choice) {

color(2); cout << "(=)";

color(7); cout << " to save file\n";

color(2); cout << "(/)";

color(7); cout<<" to show suggested words\n";

color(2); cout << "(-)";

color(7); cout << " to continue\n";

color(2); cout << "(0)";

color(7); cout << " to exit\n";

cout << "\nEnter your choice: "; cin >> choice;

}

string getSubStr(string txt) {

string result="";

int l=txt.length() - 1;

while(l>-1 && txt[l]!=' '){

l--;

}

while(txt[++l]!='\0'){

result+=txt[l];

}

return result;

}

int main() {

Trie trie;

fstream fin;

string wordStr;

fin.open("dictionary.txt", ios::in);

while (!fin.eof()) {

getline(fin, wordStr);

trie.insertWord(wordStr);

}

fin.close();

//trie.viewDictionary();

//writing text in editor

ofstream fout;

char mainchoice='\0';

char subchoice='\0';

string txt="\0";

string subStr="\0";

string temp="\0";

do

{

mainMenu(mainchoice);

switch (mainchoice)

{

case '1':

txt = "\0"; temp = "\0";

system("cls");

cout << "Write the text here\_ :";

label:

cin.clear();

cin.ignore();

temp = txt;

getline(cin, txt);

temp += txt;

txt = temp;

//

do {

subMenu(subchoice);

switch (subchoice) {

case '=':

{

fout.open("Encoding.txt");

Huffman hf(txt);

LL\_node\* head = hf.getList();

while (head != NULL) {

fout << head->encode<<" ";

head = head->next;

}

fout.close();

break;

}

case '/':

system("cls");

cout << "Write the text here\_ :" << txt;

subStr = getSubStr(txt);

trie.storeSuggestedWords(subStr);

trie.selectSuggestedWords(subStr);

break;

case '-':

system("cls");

cout << "Write the text here\_ :" << txt;

goto label;

break;

default:

break;

}

system("cls");

if (subStr != "\0" && subchoice!='=') {

temp = getSubStr(txt);

size\_t index = (txt.length()-temp.length());

txt.erase(index, temp.length());

txt += subStr;

}

cout << "Write the text here\_ :"<<txt<<endl;

} while (subchoice!='0');

//

break;

default:

break;

}

} while (mainchoice!='0');

return 0;

}