Data Structures Lab Project

21F-9505, 21F-9403, 20F-0434

Moiz-uddin, Areesha Faisal, basim madni

**TEXT EDITOR CPP FILE:**

#include<iostream>

#include<string>

#include<vector>

#include<conio.h>

#include<windows.h>

#include"huffman.h"

using namespace std;

class TextEditor

{

private:

string text;

POINT currentPos; //will be pointing to current position of cursor.

void set\_cursor(int x = 0, int y = 0) //moving the cursor on screen using windows.h library

{

HANDLE handle; //to handle the cursor.

COORD coordinates; //for setting the x and y coordinates.

handle = GetStdHandle(STD\_OUTPUT\_HANDLE);

coordinates.X = x;

coordinates.Y = y;

SetConsoleCursorPosition(handle, coordinates); //will set the cursor.

}

void displaySuggestions(vector<string> suggestedWords)

{

//creating above line of suggestion box

set\_cursor(0, 29);

int j = 0;

while (j < 61) {

cout << "\*"; j++;

}

cout << "(SUGGESTIONS)";

j = 0;

while (j < 61) {

cout << "\*"; j++;

}

cout << endl << "\* ";

//printing suggestion

for (int i = 0; i < suggestedWords.size(); i++)

{

cout << suggestedWords[i] << " ";

//check for displaying next five suggestions in second line

if (i == 4)cout << endl << "\* ";

}

////////////////////////

cout << endl << "\*";

//printing bottom line of suggestion box

int k = 0;

while (k < 134) {

cout << "\*"; k++;

}

//////////////////////////////////////////

cout << endl;

//currently cursor is at bottom of page after printing bottom line of astericks of suggestion box i.e current\_cursor\_position(0,34)

//so setting cursor to first character of first suggested word by moving cursor up and right i.e(8,30)

currentPos.x = 8;

currentPos.y = 30;

set\_cursor(currentPos.x, currentPos.y);

}

string selectSuggestedWord(vector<string> suggestedWords) {

int i = 0;

char c = \_getch(); //will get the character and store it in c.

// while use is pressing '/' moving cursor and selected index forward

while (c != ';') {

if (c >= 27)

return "moreword";

else if (c != '/') //if first word is not selected by slash so dive in to second loop to select word

c = \_getch();

while (c == '/') {

if (i == 4) {

//when first five words are forwarded then moving cursor to next line i.e y: 30->31 and x to first character of 6th word i.e x=8

currentPos.x = 8;

currentPos.y = 31;

set\_cursor(currentPos.x, currentPos.y);

}

else if (i == 9) {

//when 10 words are forwarded then moving cursor to above line i.e y: 31->30 and x to first character of 1st word i.e x=8

currentPos.x = 8;

currentPos.y = 30;

set\_cursor(currentPos.x, currentPos.y);

}

else {

//moving x coordinate of cursor forward from one word to other word by adding size of previous word plus 4(skipping 4 spaces between words)

currentPos.x = currentPos.x + suggestedWords[i].size() + 4;

set\_cursor(currentPos.x, currentPos.y);

}

// moving selected index i.e(i) forward in array

i = (i + 1) % suggestedWords.size();

c = \_getch();

if (c == ';')

break;

}

}

return suggestedWords[i];

}

public:

TextEditor(string inputFile)//constructor

{

ShowWindow(GetConsoleWindow(), SW\_MAXIMIZE); //maximizing console

huffman h(inputFile);

this->text = h.decode(); //decoding previously saved data

}

void displayText()

{

cout << this->text; //displaying text string

}

void addSuggestedWord(char c, vector<string>& s)

{

string currentword;

currentword += c;

vector<string> suggestedWords;

string word = "moreword";

while (word.compare("moreword") == 0)

{

suggestedWords.clear();

int k = 0;

for (; k < s.size(); k++)

{

string temp;

for (int j = 0; j < currentword.size(); j++)

{

if (j < s[k].size()) {

temp += s[k][j];

}

}

if (currentword.compare(temp) == 0)

break;

}

int count = 0;

while (count < 10 && k < s.size())

{

suggestedWords.push\_back(s[k]);

k++;

count++;

}

this->displaySuggestions(suggestedWords);

word = this->selectSuggestedWord(suggestedWords);

if (word.compare("moreword") != 0)//if word is selected from suggestions then adding it to text string and returning

{

this->text += word; //placing selected word in to word string

system("cls"); //clearing screen

set\_cursor(0, 0); //set cursor to 00

this->displayText(); //displaying previous word

}

// if user have not selected from current suggestions then taking next character of word to show next suggestions

else {

system("cls"); //clearing screen

set\_cursor(0, 0);//placing cursor to 00

this->displayText();//printing previous text

cout << currentword;

c = \_getch(); //get next character of currently typing word

cout << c;

currentword += c;

}

}

}

void compressAndSave(string path)

{

huffman h(path);

h.encode(text); //encoding and saving to file

}

void addSuggestedWord(char c)

{

if (c == '\r') // check for newline (new line is read as '\r' by getch() so manually printing it and inserting orignal '\n' character to text string)

{

cout << endl;

this->text += '\n';

}

}

vector<string> read\_word\_by\_word(string filename)

{

vector<string> data;

fstream file;

string word;

int i = 0;

file.open(filename.c\_str());

while (!file.eof()) {

file >> word;

data.push\_back(word);

//cout << data[i] << endl;

i++;

}

file.close();

return data;

}

};

**HUFFMAN HEADER FILE:**

#pragma once

#ifndef HUFFMAN\_H

#define HUFFMAN\_H

#include <string>

#include <queue>

#include <vector>

#include <fstream>

using namespace std;

struct node

{

char id;

int freq;

string code;

node\* left;

node\* right;

node()

{

left = right = NULL;

}

};

typedef node\* node\_ptr;

class huffman

{

protected:

//protected data members

char id;

node\_ptr node\_array[128];

node\_ptr child, parent, root;

fstream in\_file, out\_file;

string in\_file\_name, out\_file\_name;

class compare

{

public:

bool operator()(const node\_ptr& c1, const node\_ptr& c2) const

{

return c1->freq > c2->freq;

}

};

priority\_queue<node\_ptr, vector<node\_ptr>, compare> pq;

//protected functions

string decimal\_to\_binary(int);

int binary\_to\_decimal(string&);

void create\_node\_array();

inline void build\_tree(string&, char);

void traverse(node\_ptr, string);

void recreate\_huffman\_tree();

string decoding();

void create\_pq(string text);

void create\_huffman\_tree();

void calculate\_huffman\_codes();

void coding\_save(string in\_text);

public:

//public functions

huffman(string);

void encode(string text);

string decode();

};

#endif

**HUFFMAN CPP FILE:**

#include<iostream>

#include<string>

#include "huffman.h"

void huffman::create\_node\_array()

{

for (int i = 0; i < 128; i++)

{

node\_array[i] = new node;

node\_array[i]->id = i;

node\_array[i]->freq = 0;

}

}

void huffman::traverse(node\_ptr node, string code)

{

if (node->left == NULL && node->right == NULL)

{

node->code = code;

}

else

{

traverse(node->left, code + '0');

traverse(node->right, code + '1');

}

}

int huffman::binary\_to\_decimal(string& in)

{

int result = 0;

for (int i = 0; i < in.size(); i++)

result = result \* 2 + in[i] - '0';

return result;

}

string huffman::decimal\_to\_binary(int in)

{

string temp = "";

string result = "";

while (in)

{

temp += ('0' + in % 2);

in /= 2;

}

result.append(8 - temp.size(), '0');

for (int i = temp.size() - 1; i >= 0; i--)

{

result += temp[i];

}

return result;

}

inline void huffman::build\_tree(string& path, char a\_code)

{

node\_ptr current = root;

for (int i = 0; i < path.size(); i++)

{

if (path[i] == '0')

{

if (current->left == NULL)

current->left = new node;

current = current->left;

}

else if (path[i] == '1')

{

if (current->right == NULL)

current->right = new node;

current = current->right;

}

}

current->id = a\_code; //attach id to the leaf

}

huffman::huffman(string in) //constructor of huffman class.

{

in\_file\_name = in;

out\_file\_name = in;

create\_node\_array();

}

void huffman::create\_pq(string text) //creating priority queue as we have to suggest first 10 words from dictionary.

{

int i = 0;

while (i < text.size())

{

node\_array[text[i]]->freq++;

i++;

}

for (int i = 0; i < 128; i++)

{

if (node\_array[i]->freq)

{

pq.push(node\_array[i]);

}

}

}

void huffman::create\_huffman\_tree()

{

priority\_queue<node\_ptr, vector<node\_ptr>, compare> temp(pq);

while (temp.size() > 1)

{

root = new node;

root->freq = 0;

root->left = temp.top();

root->freq += temp.top()->freq;

temp.pop();

root->right = temp.top();

root->freq += temp.top()->freq;

temp.pop();

temp.push(root);

}

}

void huffman::calculate\_huffman\_codes()

{

traverse(root, "");

}

void huffman::coding\_save(string in\_text)

{

out\_file.open(out\_file\_name, ios::out | ios::binary);

string in = "", s = "";

in += (char)pq.size();

priority\_queue<node\_ptr, vector<node\_ptr>, compare> temp(pq);

while (!temp.empty())

{

node\_ptr current = temp.top();

in += current->id;

s.assign(127 - current->code.size(), '0');

s += '1';

s.append(current->code);

string c = s.substr(0, 8);

in += (char)binary\_to\_decimal(c);

for (int i = 0; i < 15; i++)

{

s = s.substr(8);

string tempStr = s.substr(0, 8);

in += (char)binary\_to\_decimal(tempStr);

}

temp.pop();

}

s.clear();

int i = 0;

while (i < in\_text.size())

{

s += node\_array[in\_text[i]]->code;

while (s.size() > 8)

{

string tempstr = s.substr(0, 8);

in += (char)binary\_to\_decimal(tempstr);

s = s.substr(8);

}

i++;

}

int count = 8 - s.size();

if (s.size() < 8)

{

s.append(count, '0');

}

in += (char)binary\_to\_decimal(s);

in += (char)count;

out\_file.write(in.c\_str(), in.size());

out\_file.close();

}

void huffman::recreate\_huffman\_tree()

{

in\_file.open(in\_file\_name, ios::in | ios::binary);

in\_file.seekg(0, ios::end);

if (in\_file.tellg() != 0) {

in\_file.seekg(0, ios::beg);

unsigned char size;

in\_file.read(reinterpret\_cast<char\*>(&size), 1);

root = new node;

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

{

char a\_code;

unsigned char h\_code\_c[16];

in\_file.read(&a\_code, 1);

in\_file.read(reinterpret\_cast<char\*>(h\_code\_c), 16);

string h\_code\_s = "";

for (int i = 0; i < 16; i++)

{

h\_code\_s += decimal\_to\_binary(h\_code\_c[i]);

}

int j = 0;

while (h\_code\_s[j] == '0')

{

j++;

}

h\_code\_s = h\_code\_s.substr(j + 1);

build\_tree(h\_code\_s, a\_code);

}

}

in\_file.close();

}

string huffman::decoding()

{

string textt;

int i = 0;

in\_file.open(in\_file\_name, ios::in | ios::binary);

in\_file.seekg(0, ios::end);

if (in\_file.tellg() != 0) {

in\_file.seekg(0, ios::beg);

unsigned char size;

in\_file.read(reinterpret\_cast<char\*>(&size), 1);

in\_file.seekg(-1, ios::end);

char count0;

in\_file.read(&count0, 1);

in\_file.seekg(1 + 17 \* size, ios::beg);

vector<unsigned char> text;

unsigned char textseg;

in\_file.read(reinterpret\_cast<char\*>(&textseg), 1);

while (!in\_file.eof())

{//get the text byte by byte using unsigned char

text.push\_back(textseg);

in\_file.read(reinterpret\_cast<char\*>(&textseg), 1);

}

node\_ptr current = root;

string path;

for (int i = 0; i < text.size() - 1; i++)

{//translate the huffman code

path = decimal\_to\_binary(text[i]);

if (i == text.size() - 2)

path = path.substr(0, 8 - count0);

for (int j = 0; j < path.size(); j++)

{

if (path[j] == '0')

current = current->left;

else

current = current->right;

if (current->left == NULL && current->right == NULL)

{

textt += current->id;

current = root;

}

}

}

in\_file.close();

}

return textt;

}

void huffman::encode(string text)

{

create\_pq(text);

create\_huffman\_tree();

calculate\_huffman\_codes();

coding\_save(text);

}

string huffman::decode()

{

recreate\_huffman\_tree();

return decoding();

}

**TRIE TREE HEADER FILE:**

#include <iostream>

#include <string>

#include<vector>

using namespace std;

#define NO\_OF\_ALPHABETS 26

#define MAX\_WORD\_SIZE 100

struct TrieNode

{

TrieNode\* child[NO\_OF\_ALPHABETS];

bool isEndOfWord;

TrieNode() :isEndOfWord(false) //constructor

{

for (int i = 0; i < NO\_OF\_ALPHABETS; i++)

child[i] = NULL;

}

};

void insert(TrieNode\* root, string word) //inserting words to tree

{

for (int i = 0; word[i] != '\0'; i++) //loop will work till NULL character.

{

if (root->child[word[i] - 'a'] == NULL) //checking from letter a to entered letter and if it is equal to NULL.

{

root->child[word[i] - 'a'] = new TrieNode; //creating new node of tree.

}

root = root->child[word[i] - 'a'];

}

root->isEndOfWord = true; //if word is end and found then return true and loop will terminate.

}

void getWord(string str, int n, char c, vector<string>& s)

{

string temp;

if (str[0] == c) //comparing character with the first index of the string.

{

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

{

temp += str[i];

}

s.push\_back(temp); //if all the characters are entered it will push the whole string.

}

}

void getAllWords(TrieNode\* root, char\* wordArray, int pos, char c, vector<string>& s) //traversing tree to get words

{

if (root == NULL)

return;

if (root->isEndOfWord)

{

getWord(wordArray, pos, c, s);

}

for (int i = 0; i < NO\_OF\_ALPHABETS; i++) //loop is working till all the alphabets which is 26.

{

if (root->child[i] != NULL) //loop will traverse until it is not equals to NULL from the root

{

wordArray[pos] = i + 'a';

getAllWords(root->child[i], wordArray, pos + 1, c, s);

}

}

}

**MAIN CPP FILE:**

#include <iostream>

#include "huffman.h"

#include "texteditor.cpp"

#include "tree.h"

#include<conio.h>

#include<string.h>

#include<windows.h>

#include<vector>

using namespace std;

int main()

{

//\*screen maximize function is in constructor of text editor\*

string path = "C:/Users/Hp/Documents/Editor/Editor/output.txt"; //path of output file where text will be saved

TextEditor t(path);//creating textEdiitor//contructor

t.displayText(); // displaying previously saved text after decoding it in TextEditor's cunstructor

vector<string> for\_insert;

//reading dictionary, file handling and storing dictionary in "for\_insert" vector

for\_insert = t.read\_word\_by\_word("C:/Users/Hp/Documents/Editor/Editor/dictionary.txt");

TrieNode\* root = new TrieNode;//creating tree //cunstructor call

//Moving dictionary word form "for\_insert" vector to tree

for (int i = 0; i < for\_insert.size(); i++)

insert(root, for\_insert[i]);

while (1)

{

char c = \_getch(); //taking character input without pressing enter

if (c == '\b') {//do nothing on pressing backspace to remove bactward editing functionality

}

else if (c == '=')

{

t.compressAndSave(path); //encoding and saving text

break;//exiting

}

else if (c == 27)//esc button to retun without saving

return 0;

{

//if input character is in range 'a' to 'z' then show suggestions else simply add to text

if (c >= 'a' && c <= 'z')

{

vector<string> suggestion;

char wordArray[1000];

getAllWords(root, wordArray, 0, c, suggestion); //getting all Suggestions of input charcter from tree in to "suggestion" vector

cout << c;

t.addSuggestedWord(c, suggestion); //1. displaying suggestions, 2. selecting one of the suggested word and 3. adding to text

}

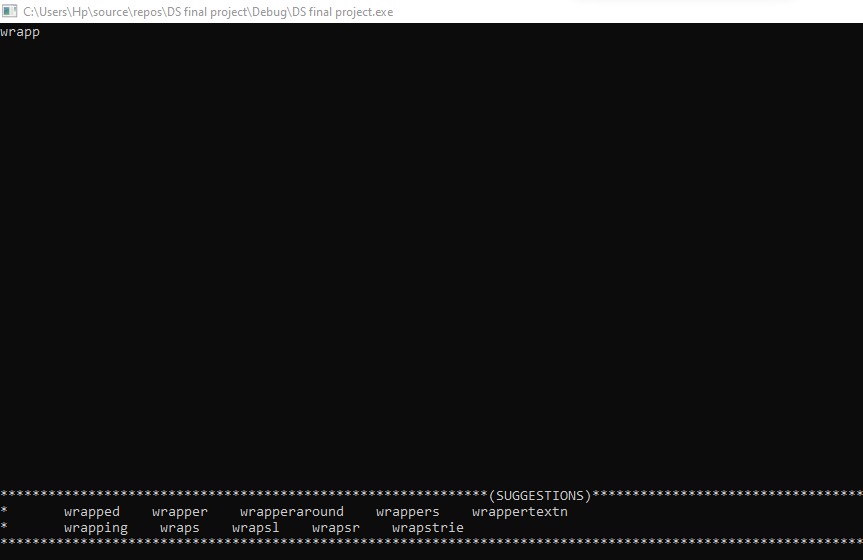
}

}

return 0;

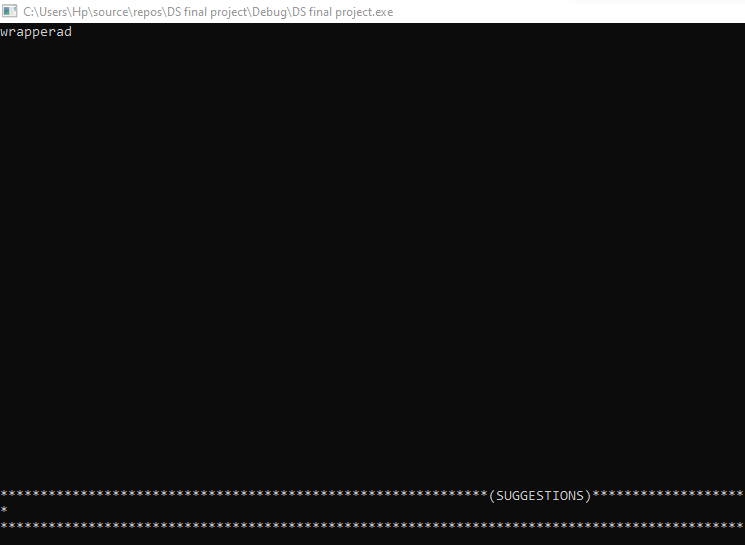
}

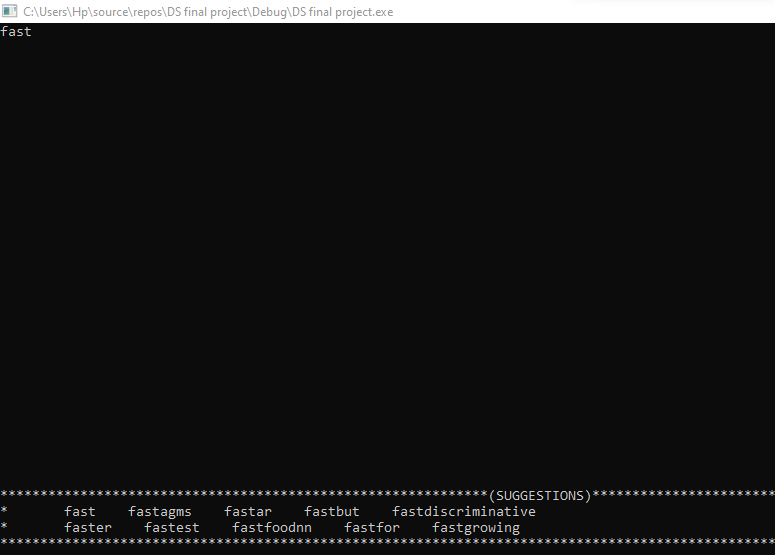
**Output:**

****

**Text

Description automatically generated**

****

****