**TEXT EDITOR**

**DATA STRUCTURE**

**TEAM MEMBER:**

**SAAD ASHRAF (21F-9167)**

**AHMAD SHAYAN (21F-9135)**

**ABDUL MUTAAL (21F-9166)**

**SOURCE.CPP:**

#include<windows.h>

#include<iostream>

#include<conio.h>

#include<string>

#include "TextEditor.h"

#include<cstring>

using namespace std;

int main(){

system("color 7C");

cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\t TEXT EDITOR WITH AUTOMATED SUGGESTIONS" << endl;

Sleep(5000);

cout << "\t\t\t";

for (int s = 1; s < 5; s++) {

Sleep(500);

cout << ".................";

}

system("cls");

system("color 7C");

TextEditor TE("D.txt");

TE.Editing();

return 0;

}

**TextEditor.CPP:**

#include<windows.h>

#include<iostream>

#include<conio.h>

#include<string>

#include<cstring>

#include<fstream>

#include"Huffman.h"

#include "TextEditor.h"

//#pragma warning(disable:4996)

using namespace std;

node\* head2 = NULL;

parent a;

int count2 = 0;

int count3 = 0;

int count1 = 0;

void gotoRowCol(int rpos, int cpos) {

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

int xpos = cpos, ypos = rpos;

COORD scrn;

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

scrn.X = xpos;

scrn.Y = ypos;

SetConsoleCursorPosition(hOuput, scrn);

}

TextEditor::TextEditor(string dname) {

ADocument = new Document(dname);

ADocument->PrintDocument();

}

TextEditor::~TextEditor() {

}

void TextEditor::Editing() {

//cr is current row

//cc is current column 3

char ch;

int d;

cr = 0, cc = 0;

gotoRowCol(cr, cc);

while (true) {

if (\_kbhit()) { //\_kbhit tells whether there is key in the buffer

d = \_getch(); //if key has been pressed it is stored in d in integer

if (d == 224) { //special key

d = \_getch();

//if (d == 72) { //up arrow key so decrement in row

// if (cr > 0)

// cr--;

// if (ADocument->Ls[cr].Size < cc) {

// cc = ADocument->Ls[cr].Size;

// }

//}

if (d == 80) { //down arrow so increment in row

if (ADocument->NOLs - 1 != cr)

cr++;

if (ADocument->Ls[cr].Size < cc) {

cc = ADocument->Ls[cr].Size;

}

}

else if (d == 77) { //right arrow so increment in column

cc++;

}

else if (d == 75) { //left arrow so decrement in column

cc--;

}

}

else if (d == 13) { //enter key

cr++; //current row increment

cc = 0; //current column is 0

}

else {

ch = char(d);

//cout << ch;#

ADocument->InsertACharacter(cr, cc, ch);

ADocument->PrintDocument();

if ((ch != ';') && (ch != '/') && (ch != '\b') && (ch != 9)) {

cc++;

}

}

gotoRowCol(cr, cc);

}

}

}

void Document::PrintDocument() {

gotoRowCol(0, 0);

ofstream Wrt(DName);

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

Wrt << Ls[i].Cs << endl;

cout << Ls[i].Cs << endl;

}

}

void Document::InsertACharacter(int li, int& ci, int ch) {

if (ch == ';') { //to insert the suggestion you choose and complete the word

Ls[li].Insert(ci, count1);

return;

}

else if (ch == '/') { //to choose suggestion and traverse through it

Ls[li].check();

}

else if (ch == 9) { //tab ascii to encode decode data

Ls[li].HUFFMAN();

}

else if (ch == '\b') { //backspace to delete the word

Ls[li].Delete(ci);

}

else {

count1 = 0;

Ls[li].InsertCharAt(ci, ch); //if any other character then insert it and show on screen

}

}

void Document::Load(string fname) {

DName = fname;

ifstream Rdr("Ds.txt", ios::app);

Rdr >> NOLs;

Rdr.ignore();

Ls = new Line[NOLs];

int ri = 0, ci = 0;

char ch;

while (true) {

ch = Rdr.get();

if (!Rdr) {

break;

}

if (ch == '\n') {

ri++;

ci = 0;

}

else {

InsertACharacter(ri, ci, ch);

ci++;

}

}

}

Document::Document(string fname) {

NOLs = 0;

Load(fname);

}

Line::Line() {

Cs = new char[1]{};

Size = 0;

}

Line::Line(const Line& AL) {

Cs = new char[AL.Size + 1];

Size = AL.Size;

for (int i = 0; i <= AL.Size; i++) {

Cs[i] = AL.Cs[i];

}

}

void Line::HUFFMAN() {

system("cls");

ifstream inn("D.txt");

while (!inn.eof()) {

char c[2000] = { 0 };

inn.getline(c, 2000);

if (c[0] == '\0') {

return;

}

int size = strlen(c);

string s = convertToString(c, size);

cout << endl;

H\_Tree(s);

}

system("pause");

return;

}

void Line::Delete(int& cc) {

string b = "\0";

for (int i = 0; i < Size - 1; i++) {

b = b + Cs[i];

}

delete[] Cs;

Size = Size;

Cs = new char[Size];

for (int i = 0; i < Size - 1; i++) {

Cs[i] = b[i];

}

Cs[Size - 1] = '\0';

Size = Size - 1;

cc = Size;

}

void Line::check() {

cout << endl;

node\* temp = a.R\_Head();

int su = 0;

while (temp != NULL) {

if (su == count1) {

system("cls");

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

SetConsoleTextAttribute(h, 9);

cout << temp->data << " ";

cout << endl;

Sleep(2000);

//system("pause");

count1 = count1 + 1;

system("color 7C");

return;

}

temp = temp->next;

su = su + 1;

}

}

void Line::Insert(int& ppp, int& count1) {

node\* temp = a.R\_Head();

int su = 0;

string l = "\0";

int k = 0;

string p;

while (temp != NULL) {

if (su == count1 - 1) {

p = temp->data;

for (int m = 0; m < this->Size; m++) {

if (k == count3) {

Cs = new char[l.length() + p.length() + 2];

for (int z = 0; z < l.length(); z++) {

Cs[z] = l[z];

}

if (k != 0) {/\*

Cs[l.length()] = 32;\*/

int c = 0;

//change in line

for (int z = l.length(); z < l.length() + p.length(); z++) {

Cs[z] = p[c];

c++;

}

Cs[l.length() + p.length()] = '\0';

Size = l.length() + p.length();

ppp = Size;

}

else {

int c = 0;

for (int z = l.length(); z < l.length() + p.length() + 1; z++) {

Cs[z] = p[c];

c++;

}

Cs[l.length() + p.length()] = '\0';

Size = l.length() + p.length();

ppp = Size;

}

return;

}

if (Cs[m] == 32) {

k = k + 1;

}

l = l + Cs[m];

}

}

temp = temp->next;

su = su + 1;

}

}

void Line::InsertCharAt(int i, char ch) {

if (ch == 32) {

/\*count2 = 0;

Cs = NULL;

Size = 0;

i = 0;\*/

count3 = count3 + 1;

node\* head = a.R\_Head();

a.deleteList(&head);

}

else if (count2 > 0) {

count2 = 1;

node\* head = a.R\_Head();

a.deleteList(&head);

}

count2 = count2 + 1;

char\* HA = new char[Size + 2];

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

HA[ci] = Cs[ci];

}

HA[i] = ch;

for (int ci = i; ci <= Size; ci++) {

HA[ci + 1] = Cs[ci];

}

delete[] Cs;

Cs = HA;

if (ch == '/') {

string l;

int o = 0;

for (int k = 0; k < i + 1; k++) {

if (Cs[k] != '/') {

l = l + Cs[o];

o = o + 1;

}

}

char\* hm = new char[o + 1];

for (int k = 0; k < o; k++) {

hm[k] = l[k];

}

hm[o] = '\0';

system("cls");

cout << HA;

delete[] Cs;

Cs = hm;

}

/\*int saze = sizeof(Cs)/ sizeof(Cs[0]);\*/

Trie\* t = new Trie();

ifstream fin;

string line;

char tc = ch;

fin.open("dic.txt");

while (!fin.eof()) {

getline(fin, line);

//a.create\_node(line);

t->insertWord(line);

}

fin.close();

bool j = 1;

while (j) {

system("cls");

cout << endl;

cout << endl << "\n\n\n\nEnter a word to auto-complete it :\n";

string key = Cs;

string chag = "\0";

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

if (key[i] == ' ') {

chag = "\0";

}

else {

chag = chag + key[i];

}

}

key = chag;

if (key == "\0") {

Size++;

break;

}

bool possible = 0;

t->autoComplete(key, possible);

int x = 1;

Size++;

return;

}

}

Trie::Trie() {

this->count = 0;

root = new TrieNode('\0');

}

bool Trie::insertWord(TrieNode\* root, string word) {

if (word.size() == 0) {

if (!root->isTerminal) {

root->isTerminal = true;

return true;

}

else {

return false;

}

}

int index = word[0] - 'a';

TrieNode\* child;

if (root->children[index] != NULL) {

child = root->children[index];

}

else {

child = new TrieNode(word[0]);

root->children[index] = child;

}

return insertWord(child, word.substr(1));

}

void Trie::insertWord(string word) {

if (insertWord(root, word)) {

this->count++;

}

}

void Trie::complete(TrieNode\* root, string word, string output, bool& possible, string& key) {

if (word.length() == 0) {

if (root->isTerminal) {

if (!possible)

///this is first word we found that has key as a prefix

cout << "\nWords containing " << key << " as prefix are:\n";

possible = 1;

cout << "\t\t\t\t\t\t\t\t\t\t\t\t\t" << output << endl;

a.create\_node(output);

}

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

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

string output1 = output;

output1 += root->children[i]->data;

complete(root->children[i], word.substr(), output1, possible, key);

}

}

return;

}

int index = word[0] - 'a';

if (root->children[index] == NULL)

return;

else

complete(root->children[index], word.substr(1), output + word[0], possible, key);

}

void Trie::autoComplete(string& key, bool& possible) {

string output = "";

complete(root, key, output, possible, key);

}

TrieNode::TrieNode(char data) {

this->data = data;

children = new TrieNode \* [26];

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

children[i] = NULL;

}

isTerminal = false;

}

parent::parent() {

head = nullptr;

curr = nullptr;

}

bool parent::isEmpty() {

return head == NULL;

}

void parent::create\_node(string value) {

node\* temp = new node;

temp->data = value;

temp->next = NULL;

if (this->head == NULL) {

head2 = this->head;

this->head = curr = temp;

}

else {

curr->next = temp;

curr = curr->next;

}

}

void parent::print() {

cout << "Linked list is: " << endl;

node\* temp = this->head;

while (temp != NULL) {

cout << temp->data << endl;

temp = temp->next;

}

}

void parent::deleteList(node\*\* head\_ref) {

if (\*head\_ref == 0) {

return;

}

node\* current = \*head\_ref;

node\* next = NULL;

while (current != NULL) {

next = current->next;

free(current);

current = next;

}

this->head = NULL;

\*head\_ref = NULL;

cout << "\n";

}

**TextEditor.h:**

#include<iostream>

#include<string>

#include<fstream>

using namespace std;

#pragma once

void gotoRowCol(int rpos, int cpos);

struct node {

string data;

node\* next;

};

class parent {

node\* head = nullptr;

node\* curr = nullptr;

public:

parent();

bool isEmpty();

void create\_node(string value);

void print();

node\* R\_Head() {

return this->head;

}

void deleteList(node\*\* head\_ref);

};

class TrieNode {

public:

char data;

TrieNode\*\* children;

bool isTerminal;

TrieNode(char data);

};

class Trie {

TrieNode\* root;

public:

int count;

Trie();

bool insertWord(TrieNode\* root, string word);

void insertWord(string word);

void complete(TrieNode\* root, string word, string output, bool& possible, string& key);

void autoComplete(string& key, bool& possible);

};

class Line {

private:

friend class TextEditor;

friend class Document;

int Size;

char\* Cs;

public:

Line();

void Delete(int& cc);

void check();

void HUFFMAN();

Line(const Line& AL);

void Insert(int& a, int& count);

void InsertCharAt(int i, char ch);

};

class Document {

private:

friend class TextEditor;

int NOLs; //number of lines

Line\* Ls; //lines

string DName; //document name

public:

Document(string fname);

void Load(string fname);

void Insert();

void InsertACharacter(int li, int& ci, int ch);

void PrintDocument();

};

class TextEditor {

private:

int cr = 0, cc = 0;

Document\* ADocument;

public:

// addition

void check(); //addition

TextEditor(string dname);

void Editing();

void InsertACharacter(int ch);

~TextEditor();

};

**Huffman.h:**

#include <iostream>

#include <fstream>

#include <string>

#include <queue>

#include <bitset>

#include <unordered\_map>

using namespace std;

class NODE;

void To\_bin(string s);

void binarY\_convertor(string s);

static string str;

static int extra = 0;

struct compare;

int hcount = 0;

void encode(NODE\* root, string str, unordered\_map<char, string>& compressed\_code);

string convertToString(char\* a, int size);

void H\_Tree(string text);

//NODE Structure

class NODE {

public:

char ch;

int FrequencY;

NODE\* left, \* right;

NODE(char ch, int FrequencY, NODE\* left, NODE\* right) {

this->ch = ch;

this->FrequencY = FrequencY;

this->left = left;

this->right = right;

}

};

struct compare {

bool operator()(NODE\* l, NODE\* r) {

return (l->FrequencY > r->FrequencY);

}

};

void encode(NODE\* root, string str, unordered\_map<char, string>& compressed\_code) {

if (root == nullptr) {

return;

}

if (!root->left && !root->right) {

compressed\_code[root->ch] = str;

}

encode(root->left, str + "0", compressed\_code);

encode(root->right, str + "1", compressed\_code);

}

string convertToString(char\* a, int size) {

int i;

string s = "";

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

s = s + a[i];

}

return s;

}

void decode(NODE\* root, int& index, string str) {

if (root == nullptr) {

return;

}

if (!root->left && !root->right) {

cout << root->ch;

return;

}

index++;

if (str[index] == '0')

decode(root->left, index, str);

else

decode(root->right, index, str);

}

void H\_Tree(string text) {

hcount++;

unordered\_map <char, int> FrequencY;

for (char ch : text) {

FrequencY[ch]++;

}

priority\_queue <NODE\*, vector <NODE\*>, compare> minheap;

for (auto pair : FrequencY) {

minheap.push(new NODE(pair.first, pair.second, nullptr, nullptr));

}

while (minheap.size() != 1) {

NODE\* left = minheap.top();

minheap.pop();

NODE\* right = minheap.top();

minheap.pop();

int sum = left->FrequencY + right->FrequencY;

minheap.push(new NODE('\0', sum, left, right));

}

NODE\* root = minheap.top();

unordered\_map<char, string> compressed\_code;

encode(root, "", compressed\_code);

string str = "";

for (char ch : text) {

str += compressed\_code[ch];

}

cout << "\nENCODED DATA IS : " << str << '\n';

ofstream fout;

fout.open("encoding.txt",ios::app);

fout << str<<endl;

fout.close();

int index = -1;

cout << "\nDECODED DATA IS : ";

while (index < (int)str.size() - 2) {

decode(root, index, str);

}

To\_bin(str);

}

void binarY\_convertor(string s) {

int x = 0;

int sum = 0;

for (int i = 0, j = 8; i < 8; i++, j--) {

x = x + (int)s[i];

x = x - 48;

if (i != 7) {

x = x << 1;

}

}

str.push\_back((char)x);

}

void To\_bin(string s) {

int count = 0;

string temp;

int size = s.size();

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

count++;

if (count == 8) {

temp = s.substr(0, 8);

binarY\_convertor(temp);

s.erase(0, 8);

count = 0;

}

}

cout << endl;

if (s.size() < 8 && s.size() > 0) {

for (int i = s.size(); i < 8; i++) {

s.push\_back('0');

extra++;

}

binarY\_convertor(s);

}

}

**OUTPUTS:**

Graphical user interface, text

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, text

Description automatically generated with medium confidence

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated