Лабораторная работа №4

Дерево поиска. Частотный словарь.

Составить программу, реализующую частотный словарь текста.

Программа должна производить следующие действия;

1. Выделять в выбранном текстовом файле отдельные слова.

2. Помещать слова в дерево поиска.

3. Выводить из дерева частотный словарь слов в алфавитном порядке.

4. Выводить частотный словарь слов, отсортированный в порядке частоты их употребления.

5. Производить поиск заданного слова.

6. Производить фильтрацию, оставляющую в словаре слова определенной длины.

Алгоритм

Создаем экземпляр бинарного дерева поиска tree. Затем считываем текст из файла, если документ невозможно найти, то программа выдаст ошибку. Разбиваем текст на отдельные слова при помощи метода Split. Каждое слово отдельно очищается от символов пунктуаций внутри слова, и после проверки на наличие лишних пробелов добавляем слова в дерево.

У каждого узла есть поля: Word – значение узла, frequency – частота, и ссылки на левого и правого потомка. У дерева есть поле, указывающее на корень дерева. Проверяем пустое ли дерево, если да, то создаем узел и делаем его корнем, если нет, то рекурсивно вызываем метод AddWordRecursive(Node root, string word). В методе мы проверяем пустое ли дерево, если да, то возвращаем узел со значением word. Затем сравниваем значение текущего узла и word, если результат будет равен 0, то слова равны, если меньше 0, то word будет идти перед текущим словом, если больше 0, то word будет идти после текущего слова. Если результат сравнения меньше 0, то мы переходим к левому потомку узла и вызываем рекурсивно AddWordRecursive и передаем левый потомок и слово. Если результат сравнения больше 0, то переходим к правому потомку узла и вызываем AddWordRecursive и передаем правого потомка и слово. Если слова эквивалентны, то увеличиваем значение частоты. Метод возвращает корень дерева с обновленными данными после вставки слов.

После мы выводим слова в алфавитном порядке в отдельный файл "AlphSortedWords.txt". Для этого мы вызываем метод GetAlphabeticalFrequencyDictionary(), в котором создается пустой список и запускается процедура обхода дерева при помощи метода InOrderTraversal(). Метод обходит сначала левое поддерево, потом добавляем текущий узел в список nodeList и обходит правое поддерево. Метод возвращает список слов в алфавитном порядке. В основной программе выводим слова из списка в таком формате – Слово: частота.

Затем мы выводим слова, отсортированные по их частоте в порядке убывания, в файл "FreqSortedWords.txt". Для этого вызываем функцию GetFrequencySortedDictionary(). В методе мы создаем пустой список и запускаем обход дерева при помощи метода InOrderTraversal(). Затем полученный список мы сортируем в порядке убывания частоты и выводим полученный результат в файл.

Затем производим поиск слова, заданного пользователем, при помощи функции SearchWord(). Метод принимает ключевое слово в качестве аргумента и вызывает приватный метод SearchWordRecursive(), передавая ему узел дерева и ключевое слово в качестве аргументов. Внутри метода проверяется, является ли текущий узел пустым или содержит ключевое слово, если да, то текущий узел возвращается. Затем происходит сравнение ключевого слова с ключевым словом текущего узла. Если сравниваемое слово меньше, чем ключевое слово текущего узла, то происходит рекурсивный вызов метода для левого потомка текущего узла, если наоборот, то для правого. Функция SearchWord() возвращает узел с заданным ключевым словом, если оно присутствует в дереве, либо null, если такого узла нет. И Результат поиска выводится в консоль: если слово присутствует в дереве, то выведется само слово и его частота, если такого слова в дереве нет, то выведется сообщение об этом.

Затем производим фильтрацию слов, оставляющую в словаре только слова определенной длины, задаваемой пользователем. Вызываем метод GetWordLength(). Метод принимает значение искомой длины, задаваемой пользователем. Внутри метода мы создаем два списка nodeList, nodeList1 для искомого дерева, и для отфильтрованного дерева. Обходим дерево симметричным обходом, записываем значения узлов в nodeList, затем проходимся по списку и сравниваем длину значения слова в списке с искомой длиной. Если условие выполняется, то мы записываем слово в nodeList1, возвращаем список

Программа

#pragma once

#include <string>

#include <msclr/marshal\_cppstd.h>

namespace Searchtree {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

using namespace System::IO;

using namespace System::Text::RegularExpressions;

using namespace std;

struct node {

wstring word;

int k;

node\* left, \* right;

};

public ref class MyForm : public System::Windows::Forms::Form

{

public:

MyForm(void)

{

InitializeComponent();

}

protected:

~MyForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::DataGridView^ text;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ word;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ frequency;

private: System::Windows::Forms::Button^ filter;

private: System::Windows::Forms::Button^ search;

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::TextBox^ wordSearch;

private: System::Windows::Forms::NumericUpDown^ filterUpDown;

private: System::Windows::Forms::Button^ alphabet;

private: System::Windows::Forms::Button^ ascending;

private: System::Windows::Forms::Button^ descending;

private: System::Windows::Forms::Button^ download;

private: System::Windows::Forms::Label^ label2;

private:

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

void InitializeComponent(void)

{

this->text = (gcnew System::Windows::Forms::DataGridView());

this->word = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->frequency = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->filter = (gcnew System::Windows::Forms::Button());

this->search = (gcnew System::Windows::Forms::Button());

this->label1 = (gcnew System::Windows::Forms::Label());

this->wordSearch = (gcnew System::Windows::Forms::TextBox());

this->filterUpDown = (gcnew System::Windows::Forms::NumericUpDown());

this->alphabet = (gcnew System::Windows::Forms::Button());

this->ascending = (gcnew System::Windows::Forms::Button());

this->descending = (gcnew System::Windows::Forms::Button());

this->download = (gcnew System::Windows::Forms::Button());

this->label2 = (gcnew System::Windows::Forms::Label());

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->text))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->filterUpDown))->BeginInit();

this->SuspendLayout();

//

// text

//

this->text->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->text->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(2) { this->word, this->frequency });

this->text->Location = System::Drawing::Point(12, 12);

this->text->Name = L"text";

this->text->ReadOnly = true;

this->text->Size = System::Drawing::Size(240, 232);

this->text->TabIndex = 0;

//

// word

//

this->word->HeaderText = L"Слово";

this->word->Name = L"word";

this->word->ReadOnly = true;

//

// frequency

//

this->frequency->HeaderText = L"Частота";

this->frequency->Name = L"frequency";

this->frequency->ReadOnly = true;

//

// filter

//

this->filter->Enabled = false;

this->filter->Location = System::Drawing::Point(330, 278);

this->filter->Name = L"filter";

this->filter->Size = System::Drawing::Size(75, 23);

this->filter->TabIndex = 1;

this->filter->Text = L"Фильтр";

this->filter->UseVisualStyleBackColor = true;

this->filter->Click += gcnew System::EventHandler(this, &MyForm::filter\_Click);

//

// search

//

this->search->Enabled = false;

this->search->Location = System::Drawing::Point(330, 250);

this->search->Name = L"search";

this->search->Size = System::Drawing::Size(75, 23);

this->search->TabIndex = 2;

this->search->Text = L"Поиск";

this->search->UseVisualStyleBackColor = true;

this->search->Click += gcnew System::EventHandler(this, &MyForm::search\_Click);

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(12, 253);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(82, 13);

this->label1->TabIndex = 3;

this->label1->Text = L"Введите слово";

//

// wordSearch

//

this->wordSearch->Location = System::Drawing::Point(140, 250);

this->wordSearch->Name = L"wordSearch";

this->wordSearch->Size = System::Drawing::Size(184, 20);

this->wordSearch->TabIndex = 4;

//

// filterUpDown

//

this->filterUpDown->Location = System::Drawing::Point(140, 281);

this->filterUpDown->Name = L"filterUpDown";

this->filterUpDown->Size = System::Drawing::Size(184, 20);

this->filterUpDown->TabIndex = 5;

//

// alphabet

//

this->alphabet->Enabled = false;

this->alphabet->Location = System::Drawing::Point(258, 41);

this->alphabet->Name = L"alphabet";

this->alphabet->Size = System::Drawing::Size(100, 42);

this->alphabet->TabIndex = 6;

this->alphabet->Text = L"Алфавитный порядок";

this->alphabet->UseVisualStyleBackColor = true;

this->alphabet->Click += gcnew System::EventHandler(this, &MyForm::alphabet\_Click);

//

// ascending

//

this->ascending->Enabled = false;

this->ascending->Location = System::Drawing::Point(258, 122);

this->ascending->Name = L"ascending";

this->ascending->Size = System::Drawing::Size(100, 27);

this->ascending->TabIndex = 7;

this->ascending->Text = L"По возрастанию";

this->ascending->UseVisualStyleBackColor = true;

this->ascending->Click += gcnew System::EventHandler(this, &MyForm::ascending\_Click);

//

// descending

//

this->descending->Enabled = false;

this->descending->Location = System::Drawing::Point(258, 89);

this->descending->Name = L"descending";

this->descending->Size = System::Drawing::Size(100, 27);

this->descending->TabIndex = 8;

this->descending->Text = L"По убыванию";

this->descending->UseVisualStyleBackColor = true;

this->descending->Click += gcnew System::EventHandler(this, &MyForm::descending\_Click);

//

// download

//

this->download->Location = System::Drawing::Point(258, 12);

this->download->Name = L"download";

this->download->Size = System::Drawing::Size(100, 23);

this->download->TabIndex = 9;

this->download->Text = L"Загрузить";

this->download->UseVisualStyleBackColor = true;

this->download->Click += gcnew System::EventHandler(this, &MyForm::download\_Click);

//

// label2

//

this->label2->AutoSize = true;

this->label2->Location = System::Drawing::Point(12, 283);

this->label2->Name = L"label2";

this->label2->Size = System::Drawing::Size(122, 13);

this->label2->TabIndex = 10;

this->label2->Text = L"Выберите длину слова";

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(433, 325);

this->Controls->Add(this->label2);

this->Controls->Add(this->download);

this->Controls->Add(this->descending);

this->Controls->Add(this->ascending);

this->Controls->Add(this->alphabet);

this->Controls->Add(this->filterUpDown);

this->Controls->Add(this->wordSearch);

this->Controls->Add(this->label1);

this->Controls->Add(this->search);

this->Controls->Add(this->filter);

this->Controls->Add(this->text);

this->Name = L"MyForm";

this->Text = L"MyForm";

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->text))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->filterUpDown))->EndInit();

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

node\* Node = nullptr;

int count = 0;

void insert(node\*& root, std::wstring word)

{

if (root == nullptr) {

count++;

root = new node;

root->word = word;

root->k=1;

root->left = nullptr;

root->right = nullptr;

}

else {

if (word < root->word) {

insert(root->left, word);

}

else if (word > root->word) {

insert(root->right, word);

}

else {

root->k++;

}

}

}

void PrintTree(node\* root)

{

if (root != nullptr)

{

PrintTree(root->left);

text->Rows->Add(msclr::interop::marshal\_as<String^>(root->word), root->k);

PrintTree(root->right);

}

}

void Filter(node\* node, int count)

{

if (node == nullptr) {

return;

}

else if (node != nullptr) {

Filter(node->left, count);

Filter(node->right, count);

if (node->word.length() == count) {

text->Rows->Add(msclr::interop::marshal\_as<String^>(node->word), node->k);

}

}

}

void Search(node\* root, std::wstring word)

{

if (root != nullptr)

{

if (word < root->word)

{

Search(root->left, word);

}

else if (word > root->word)

{

Search(root->right, word);

}

else if (word == root->word)

{

wordSearch->Text = msclr::interop::marshal\_as<String^>(root->word) + " " + root->k;

}

else

{

wordSearch->Text = "Слово не найдено";

}

}

else

{

wordSearch->Text = "Слово не найдено";

}

}

private: System::Void alphabet\_Click(System::Object^ sender, System::EventArgs^ e) {

text->Rows->Clear();

PrintTree(Node);

descending->Enabled = true;

ascending->Enabled = true;

search->Enabled = true;

filter->Enabled = true;

}

private: System::Void download\_Click(System::Object^ sender, System::EventArgs^ e) { // разделение изначального текст на слова и добавление слов в дерево

OpenFileDialog^ openFileDialog1 = gcnew OpenFileDialog;

if (openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)

{

try {

Stream^ myStream = openFileDialog1->OpenFile();

if (myStream != nullptr) {

StreamReader^ reader = gcnew StreamReader(myStream);

String^ line;

String^ word;

node\* root = Node;

while (reader->Peek() != -1) {

line = reader->ReadLine();

String^ regexpr = "([\\p{L}]+)";

MatchCollection^ words;

Regex^ expression = gcnew Regex(regexpr);

words = expression->Matches(line);

for (int i = 0; i < words->Count; i++) {

word = words[i]->Value;

insert(root, msclr::interop::marshal\_as<wstring>(word));

}

}

Node = root;

}

alphabet->Enabled = true;

}

catch (Exception^ ex) {

MessageBox::Show("Ошибка при открытии файла: " + ex->Message, "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

}

private: System::Void ascending\_Click(System::Object^ sender, System::EventArgs^ e) {

text->Sort(text->Columns["frequency"], ListSortDirection::Ascending);

}

private: System::Void descending\_Click(System::Object^ sender, System::EventArgs^ e) {

text->Sort(text->Columns["frequency"], ListSortDirection::Descending);

}

private: System::Void search\_Click(System::Object^ sender, System::EventArgs^ e) {

Search(Node, msclr::interop::marshal\_as<std::wstring>(wordSearch->Text));

}

private: System::Void filter\_Click(System::Object^ sender, System::EventArgs^ e) {

text->Rows->Clear();

Filter(Node, Convert::ToInt32(filterUpDown->Value));

}

};

}