**Отчёт по лабораторной работе №4**

**Дерево поиска. Частотный словарь.**

Задание:

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.

Код программы:

|  |
| --- |
| search\_tree.cpp |
| #include "search\_tree.h"  #include <iostream>  #include <vector>  // Инстанцирование для std::wstring  template class SearchTree<std::wstring>;  template SearchTree<std::wstring>::Node::Node(std::wstring data);  template void SearchTree<std::wstring>::create(Node\*&, std::wstring&);  template void SearchTree<std::wstring>::search(Node\*&, Node\*&, std::wstring&);  template void SearchTree<std::wstring>::to\_datagrid(Node\*&, System::Data::DataTable^&, int&);  template void SearchTree<std::wstring>::to\_vector(Node\*&, std::vector<Node\*>&, int&);  template void SearchTree<std::wstring>::add(std::wstring);  template const SearchTree<std::wstring>::Node\* SearchTree<std::wstring>::search(std::wstring);  template void SearchTree<std::wstring>::to\_datagrid(System::Data::DataTable^, int);  template void SearchTree<std::wstring>::to\_vector(std::vector<Node\*>&, int);  // Инстанцирование для std::string  template class SearchTree<std::string>;  template SearchTree<std::string>::Node::Node(std::string data);  template void SearchTree<std::string>::create(Node\*&, std::string&);  template void SearchTree<std::string>::search(Node\*&, Node\*&, std::string&);  template void SearchTree<std::string>::to\_datagrid(Node\*&, System::Data::DataTable^&, int&);  template void SearchTree<std::string>::add(std::string);  template const SearchTree<std::string>::Node\* SearchTree<std::string>::search(std::string);  template void SearchTree<std::string>::to\_datagrid(System::Data::DataTable^, int);  /\*\*  \* Коструктор листа дерева  \*  \* @param сохраняемые данные  \*/  template <class T>  SearchTree<T>::Node::Node(T data) {  this->data = data;  this->count = 1;  this->left = nullptr;  this->right = nullptr;  }  /\*\*  \* Декоструктор листа дерева  \*/  template <class T>  SearchTree<T>::Node::~Node() {  if (right) delete right;  if (left) delete left;  }  /\*\*  \* Декоструктор дерева  \*/  template <class T>  SearchTree<T>::~SearchTree() {  delete root;  }  /\*\*  \* Приватный метод, создающий новый лист дерева  \*  \* @param указатель на корень дерева  \* @param сохраняемые в листе данные  \*/  template <class T>  void SearchTree<T>::create(Node\*& root, T& key) {  if (root == nullptr)  root = new Node(key);  else if (key < root->data)  create(root->left, key);  else if (key > root->data)  create(root->right, key);  };  /\*\*  \* Приватный метод, производящий рекурсивный поиск по дереву  \*  \* @param указатель на корень дерева  \* @param указатель куда вернуть найденное значение  \* @param искомые данные  \*/  template <class T>  void SearchTree<T>::search(Node\*& root, Node\*& result, T& key) {  if (root != nullptr) {  if (key == root->data)  result = root;  else if (key < root->data)  search(root->left, result, key);  else if (key > root->data)  search(root->right, result, key);  }  }  /\*\*  \* Метод, рекурсивно заполняющий заданную таблицу значениями из дерева  \*  \* @param указатель на корень дерева  \* @param CLR таблица (System::Data::DataTable)  \* @param (необязательный) фильтр длины  \*/  template <class T>  void SearchTree<T>::to\_datagrid(Node\*& root, System::Data::DataTable^& table, int& len) {  if (root != nullptr) {  to\_datagrid(root->left, table, len);  if (len) {  // Добавление значений только с заданной длинной  if (root->data.size() == len) {  System::Data::DataRow^ row = table->NewRow();  row[0] = gcnew System::String(root->data.c\_str());  row[1] = root->count;  table->Rows->Add(row);  }  }  else {  System::Data::DataRow^ row = table->NewRow();  row[0] = gcnew System::String(root->data.c\_str());  row[1] = root->count;  table->Rows->Add(row);  }  to\_datagrid(root->right, table, len);  }  };  /\*\*  \* Метод, рекурсивно заполняющий заданный вектор значениями из дерева  \*  \* @param указатель на корень дерева  \* @param вектор указателей на листы  \* @param (необязательный) фильтр длины  \*/  template <class T>  void SearchTree<T>::to\_vector(Node\*& root, std::vector<Node\*>& vec, int& len) {  if (root != nullptr) {  to\_vector(root->left, vec, len);  if (len) {  // Добавление значений только с заданной длинной  if (root->data.size() == len)  vec.push\_back(root);  }  else vec.push\_back(root);  to\_vector(root->right, vec, len);  }  };  /\*\*  \* Метод, добавляющий значение в дерево  \*  \* @param добавляемое значение  \*/  template <class T>  void SearchTree<T>::add(T key) {  Node\* value = nullptr;  search(this->root, value, key); // Поиск существующего  if (value == nullptr) create(this->root, key);  else value->count++;  };  /\*\*  \* Метод поиска по дереву (интерфейс для рекурсивного search())  \*  \* @param искомое значение  \*/  template <class T>  typename const SearchTree<T>::Node\* SearchTree<T>::search(T key) {  Node\* result = nullptr;  search(this->root, result, key);  return result;  }  /\*  \* Метод, заполняющий заданную таблицу значениями из дерева  \* (интерфейс для рекурсивного to\_datagrid())  \*  \* @param CLR таблица (System::Data::DataTable)  \* @param (необязательный) фильтр длины  \*/  template <class T>  void SearchTree<T>::to\_datagrid(System::Data::DataTable^ table, int len) {  to\_datagrid(this->root, table, len);  }  /\*\*  \* Метод, заполняющий заданный вектор значениями из дерева  \* (интерфейс для рекурсивного to\_vector())  \*  \* @param вектор указателей на листы  \* @param (необязательный) фильтр длины  \*/  template <class T>  void SearchTree<T>::to\_vector(std::vector<Node\*>& vec, int len) {  to\_vector(this->root, vec, len);  }  /\*\*  \* Метод задающий фильтр для сортировки листов дерева.  \* Сравнивает два листа либо по количеству, либо по значению, если  \* количество одинаково.  \*  \* @param указатель на первый лист  \* @param указатель на второй лист  \* @return булево значение (больше ли первый второго)  \*/  template <class T>  bool SearchTree<T>::sorter(SearchTree<T>::Node\* a, SearchTree<T>::Node\* b) {  if (a->count != b->count) {  return a->count > b->count;  }  return a->data > b->data;  } |
| MyForm.cpp |
| #include "MyForm.h"  using namespace System;  using namespace System::Windows::Forms;  /\*  \* Метод, удаляющий из строки ненужные символы  \*/  String^ SearchTree4::Searcher::strip(String^ word) {  String^ delims = L" .,!?:;=+-&#%$[](){}/\\\0\n\r\"\'«»„“”…°—––\_     0123456789";  for each (wchar\_t d in delims)  word = word->Replace(d.ToString(), "");  return word;  }  /\*  \* Метод, преобразующий тип System::String в std::wstring  \*/  std::wstring SearchTree4::Searcher::to\_string(String^ string) {  return msclr::interop::marshal\_as<std::wstring>(string);  }  /\*  \* Метод, сбрасывающий элементы интерфейса в изначальный вид  \*/  System::Void SearchTree4::Searcher::reset() {  ((DataTable^) dataGridViewAlphabet->DataSource)->Rows->Clear();  ((DataTable^)dataGridViewFreq->DataSource)->Rows->Clear();  ((DataTable^)dataGridViewFilter->DataSource)->Rows->Clear();  textBoxResult->Text = "";  textBoxSearch->Text = "";  delete s\_tree;  s\_tree = new SearchTree<std::wstring>();  }  /\*\*  \* Метод, вызываемый по нажатию кнопки открытия файла, построчно  \* делящий текстовый файл на слова, добавляя их в дерево поиска  \*/  System::Void SearchTree4::Searcher::buttonOpen\_Click(System::Object^ sender, System::EventArgs^ e) {  openFileDialog1->FileName = "";  openFileDialog1->ShowDialog();  if (openFileDialog1->FileName == "") return;  reset();  StreamReader^ reader = gcnew StreamReader(openFileDialog1->FileName, Encoding::GetEncoding("windows-1251"));  do {  String^ line = reader->ReadLine();  array<wchar\_t>^ delims = { ' ', '.', ',' , '!', '?', ':', ';',  '=', '+', '-', '&', '#', '%', '$', '[', ']', '(', ')', '{',  '}', '/', '\\', '\0', '\n', '\r', '\"', '\'', '«', '»', '„',  '“', '…', '—', '–', ' ', ' ' };  for each (String ^ word in line->Split(delims)) {  std::wstring s\_word = to\_string(strip(word->ToLower()));  if (s\_word != L"")  s\_tree->add(s\_word);  }  } while (reader->Peek() != -1);  reader->Close();  DataTable^ dt\_a = ((DataTable^)dataGridViewAlphabet->DataSource)->Clone();  delete dataGridViewAlphabet->DataSource;  s\_tree->to\_datagrid(dt\_a);  dataGridViewAlphabet->DataSource = dt\_a;  std::vector<SearchTree<std::wstring>::Node\*> words = std::vector<SearchTree<std::wstring>::Node\*>();  s\_tree->to\_vector(words);  std::sort(words.begin(), words.end(), s\_tree->sorter);  DataTable^ dt\_f = ((DataTable^)dataGridViewFreq->DataSource)->Clone();  delete dataGridViewFreq->DataSource;  for each (SearchTree<std::wstring>::Node \* word in words) {  System::Data::DataRow^ row = dt\_f->NewRow();  row[0] = gcnew System::String(word->data.c\_str());  row[1] = word->count;  dt\_f->Rows->Add(row);  }  dataGridViewFreq->DataSource = dt\_f;  }  /\*\*  \* Метод, вызываемый по нажатию кнопки сброса.  \* Является обёрткой к функции reset()  \*/  System::Void SearchTree4::Searcher::buttonReset\_Click(System::Object^ sender, System::EventArgs^ e) {  reset();  }  /\*\*  \* Метод, вызываемый по нажатию Enter в строке поиска  \* и вызывающий поиск слова в дереве поиска  \*/  System::Void SearchTree4::Searcher::textBoxSearch\_Enter(System::Object^ sender, System::Windows::Forms::KeyPressEventArgs^ e) {  if (e->KeyChar == (char)Keys::Return) {  auto t1 = std::chrono::steady\_clock::now();  const SearchTree<std::wstring>::Node\* node = s\_tree->search(to\_string(strip(textBoxSearch->Text->ToLower())));  auto t2 = std::chrono::steady\_clock::now();  std::chrono::duration<double, std::milli> dur = t2 - t1;  textBoxResult->Text = "Найдено: ";  if (node != nullptr)  textBoxResult->Text += node->count;  else  textBoxResult->Text += "0";  textBoxResult->Text += " за " + dur.count() + "ms";  }  }  /\*\*  \* Метод, вызываемый по нажатию Enter в строке фильтра и  \* вызывающий построение таблицы по фильтру длины слова  \*/  System::Void SearchTree4::Searcher::textBoxLen\_Enter(System::Object^ sender, System::Windows::Forms::KeyPressEventArgs^ e) {    if (e->KeyChar == (char)Keys::Return) {  DataTable^ dt = ((DataTable^)dataGridViewFilter->DataSource)->Clone();  delete dataGridViewFilter->DataSource;  try {  int len = Int32::Parse(textBoxLen->Text);  s\_tree->to\_datagrid(dt, len);  dataGridViewFilter->DataSource = dt;  }  catch (FormatException^ fe) {  /\*textBoxLen->Text = "";  int count = dataGridViewFilter->RowCount++;  dataGridViewFilter->Rows[count]->Cells[0]->Value = fe->Message;\*/  }  }  }  /\*\*  \* Метод, вызываемый при загрузке формы. Инициализирует данные в таблицах  \*/  System::Void SearchTree4::Searcher::Searcher\_Load(System::Object^ sender, System::EventArgs^ e) {  DataTable^ dt = gcnew DataTable("Frequency");  dt->Columns->Add(gcnew DataColumn("Слово"));  dt->Columns->Add(gcnew DataColumn("Количество"));  dataGridViewAlphabet->DataSource = dt;  dataGridViewFreq->DataSource = dt->Clone();  dataGridViewFilter->DataSource = dt->Clone();  }  [STAThread]  int main(array<String^>^ args)  {  Application::EnableVisualStyles();  Application::SetCompatibleTextRenderingDefault(false);  SearchTree4::Searcher form;  Application::Run(% form);  } |

Алгоритм работы:

Программа ожидает от пользователя текстовый файл. После того как файл указан, происходит его построчное чтение и разделение строк на слова. Каждое отделённое слово добавляется в дерево поиска.

При добавлении в дереве поиска происходит поиск уже существующего значения и если значение найдено поле количества увеличивается, если же не найдено происходит поиск свободного места в дереве. При этом место в дереве ищется с условием: если добавляемое значение меньше значения в текущем поддереве, то поиск свободного места продолжается в левом поддереве, если же добавляемое значение больше, то поиск свободного места продолжается в правом поддереве.

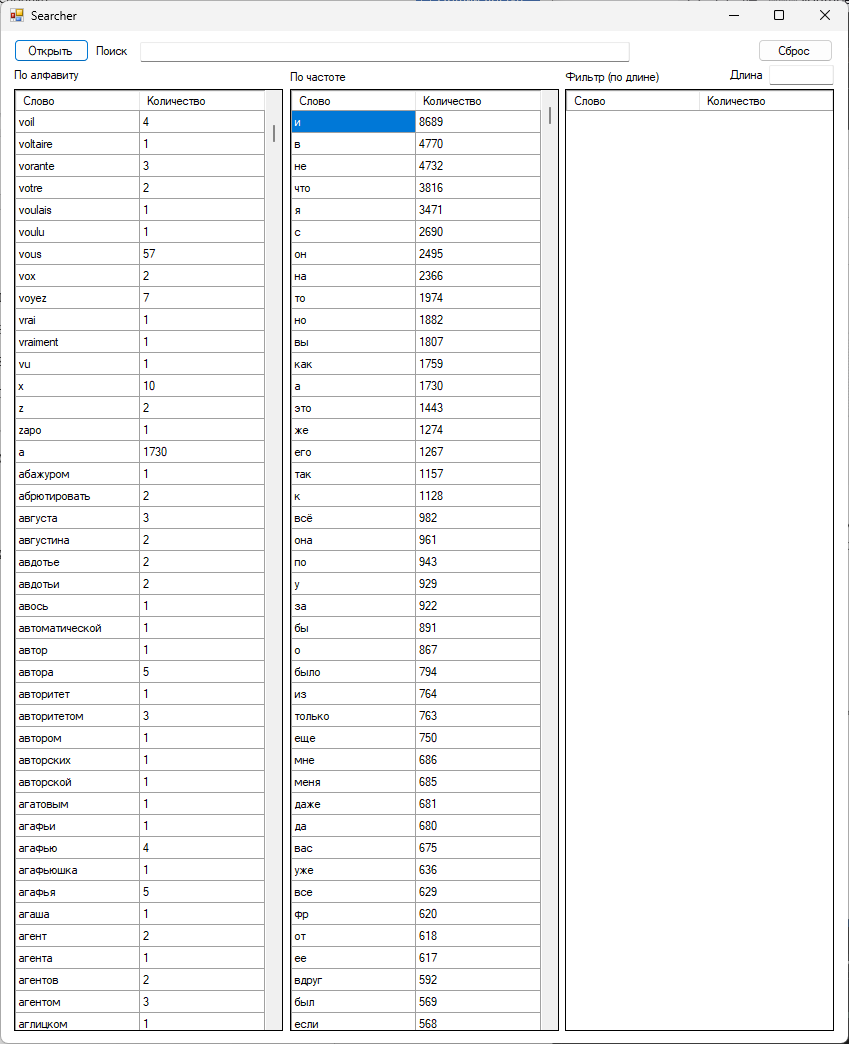
После того как дерево создано строятся таблицы частот. Первая таблица отсортирована по алфавиту – производится симметричный обход дерева и добавление в таблицу его значений на каждом шаге. Вторая таблица отсортирована по количеству – использован метод sort() для std::vector, для которого была создана собственная функция определяющая больше или меньше конкретный лист дерева.

В программе предусмотрен вывод таблицы со значениями определённой длины – таблица формируется тем же способом что и таблица частот, отсортированная по алфавиту с единственным отличием что в методе добавления в таблицу проверяется длина значения.

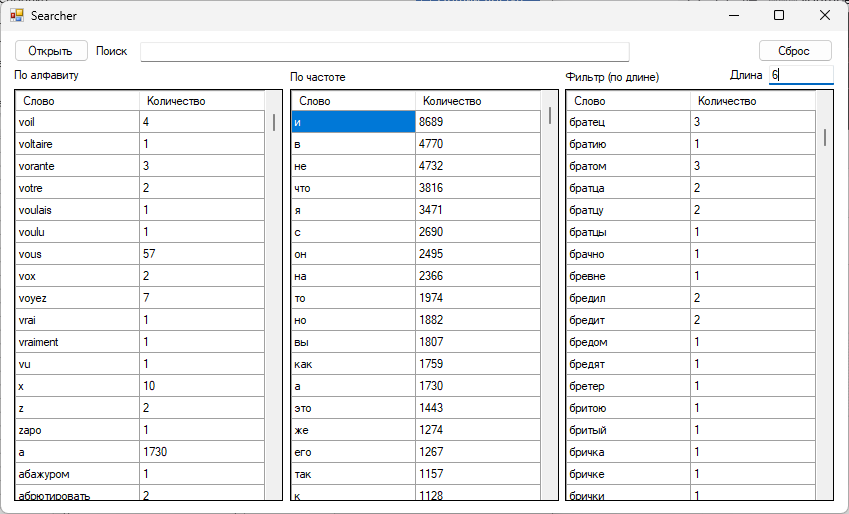
В программе также предусмотрен поиск слова в дереве. Поиск значения производится обходом дерева в зависимости от искомого значения: если оно меньше, чем значение текущего поддерева, то обход продолжается в левое поддерево, если значение больше, обход продолжается в правом поддереве.

Результат работы:

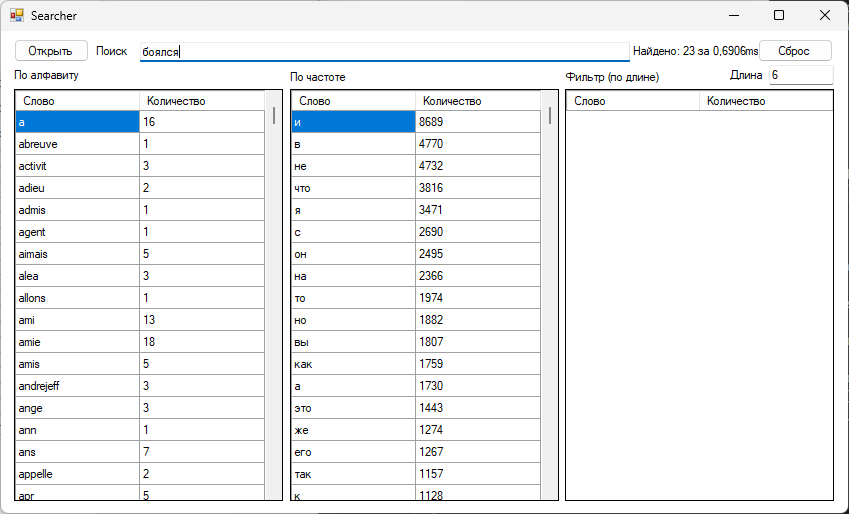
* Достоевский Федор. Бесы - royallib.ru.txt



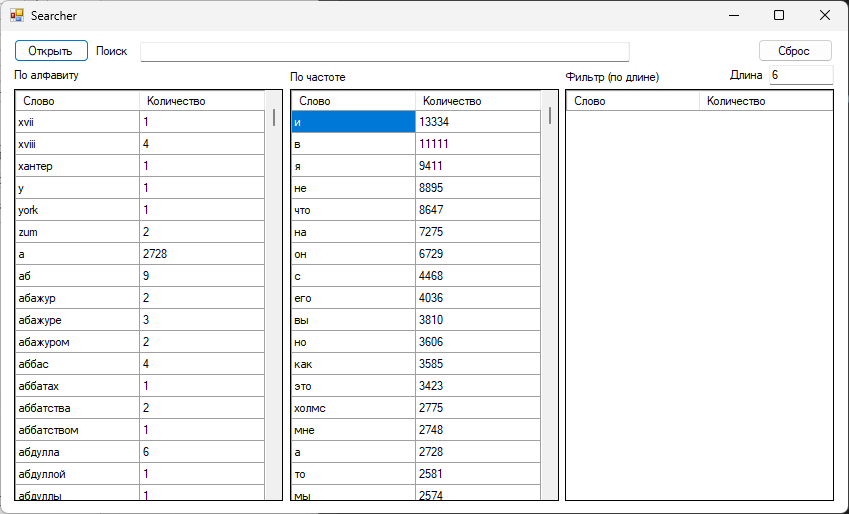
Слова с длиной 6:



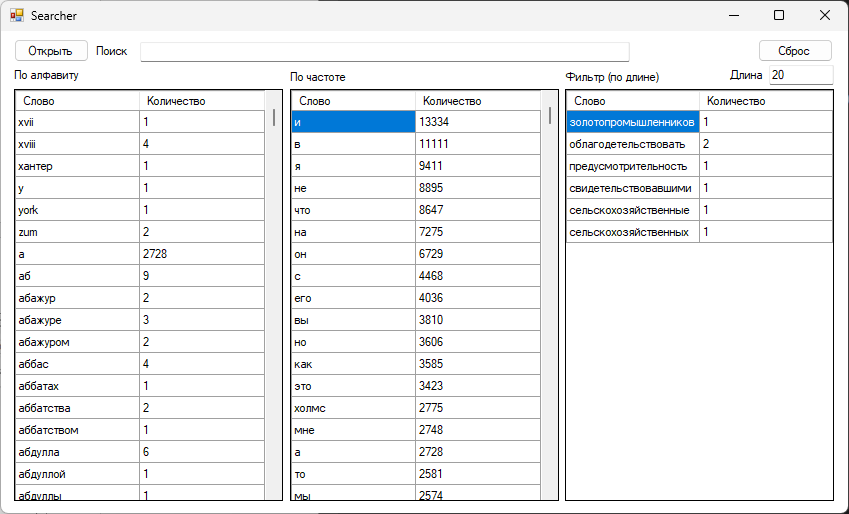
Поиск слова «боялся»:



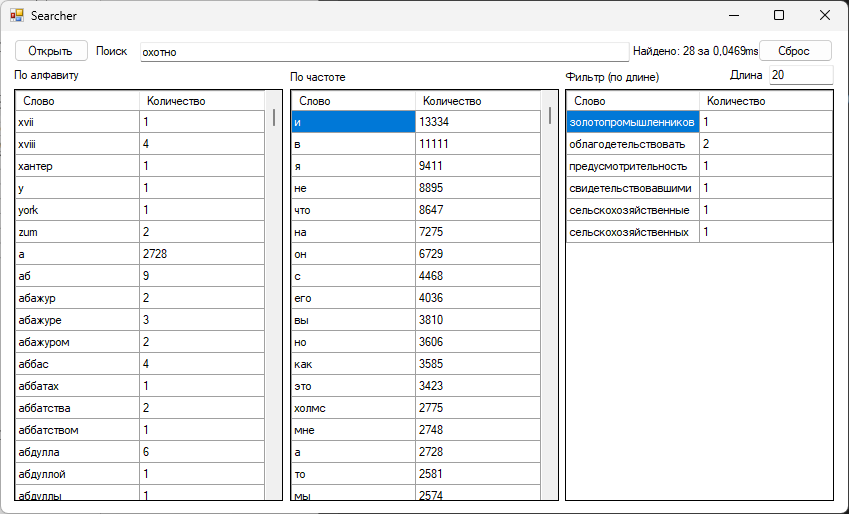
* Дойль Артур Конан. Шерлок Холмс. Большой сборник - royallib.com.txt



Слова с длиной 20:



Поиск слова «охотно»



Поиск слова «амршгнцкар»

