Алгоритмы и структуры данных

Курсовой проект на тему «Разработка программы для автодополнения текста при вводе при помощи дерева префиксов»

Подготовил мельников никита, епи-2-16

2018

Оглавление

[Задание на курсовую работу 2](#_Toc518100831)

[Краткая теория по теме курсовой работы 3](#_Toc518100832)

[Метод решения 5](#_Toc518100833)

[Руководство программиста 6](#_Toc518100834)

[Общее руководство пользователя 8](#_Toc518100835)

[Комбинации клавиш 8](#_Toc518100836)

[Анализ результатов работы программы 9](#_Toc518100837)

[Листинг программы 10](#_Toc518100838)

[main.cpp 10](#_Toc518100839)

[mainwindow.hpp 10](#_Toc518100840)

[mainwindow.cpp 11](#_Toc518100841)

[textedit.hpp 16](#_Toc518100842)

[textedit.cpp 17](#_Toc518100843)

[trie.hpp 18](#_Toc518100844)

[trie.cpp 19](#_Toc518100845)

[suggestionslist.hpp 21](#_Toc518100846)

[suggestionslist.cpp 22](#_Toc518100847)

[textedit\_p.hpp 23](#_Toc518100848)

[textedit\_p.cpp 23](#_Toc518100849)

# Задание на курсовую работу

Создать программу, поддерживающую ввод текста с автодополнением по словарю.

При запуске программа должна составить дерево префиксов на основе словаря – текстового файла.

Впоследствии программа должна позволять пользователю вводить текст, состоящий из слов и символов-разделителей (знаки препинания, пробелы, символы табуляции и новой строки). Если при вводе слова пользователь нажмёт определённую клавишу или сочетание клавиш, то программа должна предложить ему меню с вариантами завершения вводимого слова. Если пользователь ввёл слово, отсутствующее в словаре – программа должна предоставить ему выбор – добавлять или не добавлять введённое слово в словарь.

Перед завершением работы программа должна сохранять текущее состояние словаря в тот же текстовый файл, из которого он был прочитан.

При сравнении слов регистр символов должен игнорироваться.

# Краткая теория по теме курсовой работы

Префиксное дерево (также бор, луч, нагруженное дерево, англ. trie) — структура данных, позволяющая хранить ассоциативный массив, ключами которого являются строки. Представляет собой корневое дерево, каждое ребро которого помечено каким-то символом так, что для любого узла все рёбра, соединяющие этот узел с его сыновьями, помечены разными символами. Некоторые узлы префиксного дерева выделены (на рисунке они подписаны цифрами) и считается, что префиксное дерево содержит данную строку-ключ тогда и только тогда, когда эту строку можно прочитать на пути из корня до некоторого (кстати, единственного для этой строки) выделенного узла. В некоторых приложениях удобно считать все узлы дерева выделенными.

Таким образом, в отличие от бинарных деревьев поиска, ключ, идентифицирующий конкретный узел дерева, не явно хранится в данном узле, а неявно задаётся положением данного узла в дереве. Получить ключ можно выписыванием подряд символов, помечающих рёбра на пути от корня до узла. Ключ корня дерева — пустая строка. Часто в выделенных узлах хранят дополнительную информацию, связанную с ключом, и обычно выделенными являются только листья и, возможно, некоторые внутренние узлы.

Выделяют три основные операции над префиксным деревом: проверка наличия ключа в дереве, удаление ключа из дерева и вставка нового ключа (возможно, с какой-то дополнительной связанной информацией). Каждая из этих операций очевидным образом реализуется с помощью спуска по дереву из корня, но эффективность такой реализации напрямую зависит от организации навигации по узлам. Для последующего анализа различных подходов к этой проблеме обозначим через n длину строки, которую запрашивают/удаляют/вставляют, а через sigma обозначим размер алфавита, то есть количество различных символов на рёбрах данного префиксного дерева. Пусть данный узел x имеет k сыновей (при этом, очевидно, k <= sigma). Обозначим через x1, x2, … xk ссылки на этих сыновей, а через a1, a2, … ak — символы, которые помечают рёбра, соединяющие x с соответствующими сыновьями.

# Метод решения

Для реализации дерева префиксов мне понадобились 2 класса: класс Node\_p, реализующий узел дерева и класс Trie для хранения и работы с узлами, реализующий дерево. В Node\_p имеются поля: isEnd – признак конца слова, d – массив указателей на потомков узла и parent – родительский узел. Так же Node\_p имеет методы: set(char, Node\_p), который устанавливает дочерний узел по символу, at(char), который возвращает дочерний узел по символу. Класс Trie имеет поле root – корневой узел, и методы: words(preffix) – возвращает все слова по префиксу, addWord(word) – добавляет слово в дерево, contains(word) – проверяет наличие слова в дереве, remove(word) – удаляет слово из дерева. Так же имеются 2 функции для вычисления индекса в массиве d по символу и для вычисления символа по индексу.

Алгоритм:

Для добавления слова необходимо спуститься по дереву до необходимого узла, попутно создавая необходимые узлы, после спуска нужно установить текущему узлу признак конца слова.

Для сбора всех слов по префиксу необходимо спуститься по дереву до конца префикса, а затем пройтись по всем дочерним узлам, попутно собирая в список слова (слово – последовательность потомков, последний из которых имеет признак конца слова).

Для проверки на наличие слова необходимо спуститься по дереву до конца слова и проверить имеет ли конечный узел признак конца слова.

Для удаления слова необходимо спуститься по дереву до конца слова, удалить признак конца слова, а затем подняться до первого узла без потомков и без признака конца слова, попутно удаляя пустые узлы.

# Руководство программиста

static const int CAP = 70; //максимальное количество потомков узла

class Node\_p

{

friend class Trie;

#ifdef QT\_QTTEST\_MODULE\_H

friend class TrieTest;

#endif

//STATIC

static int **p\_indexHelper**(const QChar &c); //вычисляет индекс по символу

static QChar **p\_letterHelper**(int p\_indexHelper); //вычисляет символ по индексу

//FIELDS

Node\_p \*m\_parent; //указатель на родительский узел

Node\_p \*d[CAP] = {0x0}; //массив указателей на дочерние узлы

bool isEnd; //признак конца слова

//METHODS

Node\_p(Node\_p \*parent = 0x0); //конструктор, устанавливает указатель на родительский узел

Node\_p \***parent**() const;//возвращает указатель на родительский узел

Node\_p \***at**(const QChar &c) const; //возвращает указатель на дочерний узел по символу

bool **isEmpty**() const; //возвращает признак конца слова

void **set**(const QChar &c, Node\_p \*val); //устанавливает дочерний узел по символу, удаляя перед этим дочерний узел по данному символу

void **setParent**(Node\_p \*parent); //устанавливает родителя для узла

public:

virtual ~***Node\_p***();//удаляет всех своих потомков

};

class Trie

{

#ifdef QT\_QTTEST\_MODULE\_H

friend class TrieTest;

#endif

//STATIC

static void **p\_wordsHelper**(Node\_p \*node, QString word, QStringList &list, int n); //помощник для обхода дерева и сбора ключей по префиксу

static Trie s\_trie; //статический объект, обращение к которому идёт через статически метод Trie::obj(), реализует паттерн “Singleton”

public:

static Trie &**obj**();//возвращает единственный, для программы, объект класса Trie

private:

//FIELDS

Node\_p \*root; //корневой узел дерева

//METHODS

Trie() : root(new Node\_p()) {}

Trie(const Trie &){}

Trie(Trie &){}

public:

virtual ~***Trie***();//удаляет корневой узел, а тот в свою очередь удаляет всех своих потомков

bool **isEmpty**() const; //проверка на пустоту

QStringList **words**(const QString &preffix, int n = -1) const; //возвращает все слова по префиксу

bool **contains**(const QString &word) const; //проверяет наличие слова в дереве

void **addWord**(const QString &word); //добавляет слово в дерево

Trie &**operator** <<(const QString &word); //добавляет слово в дерево

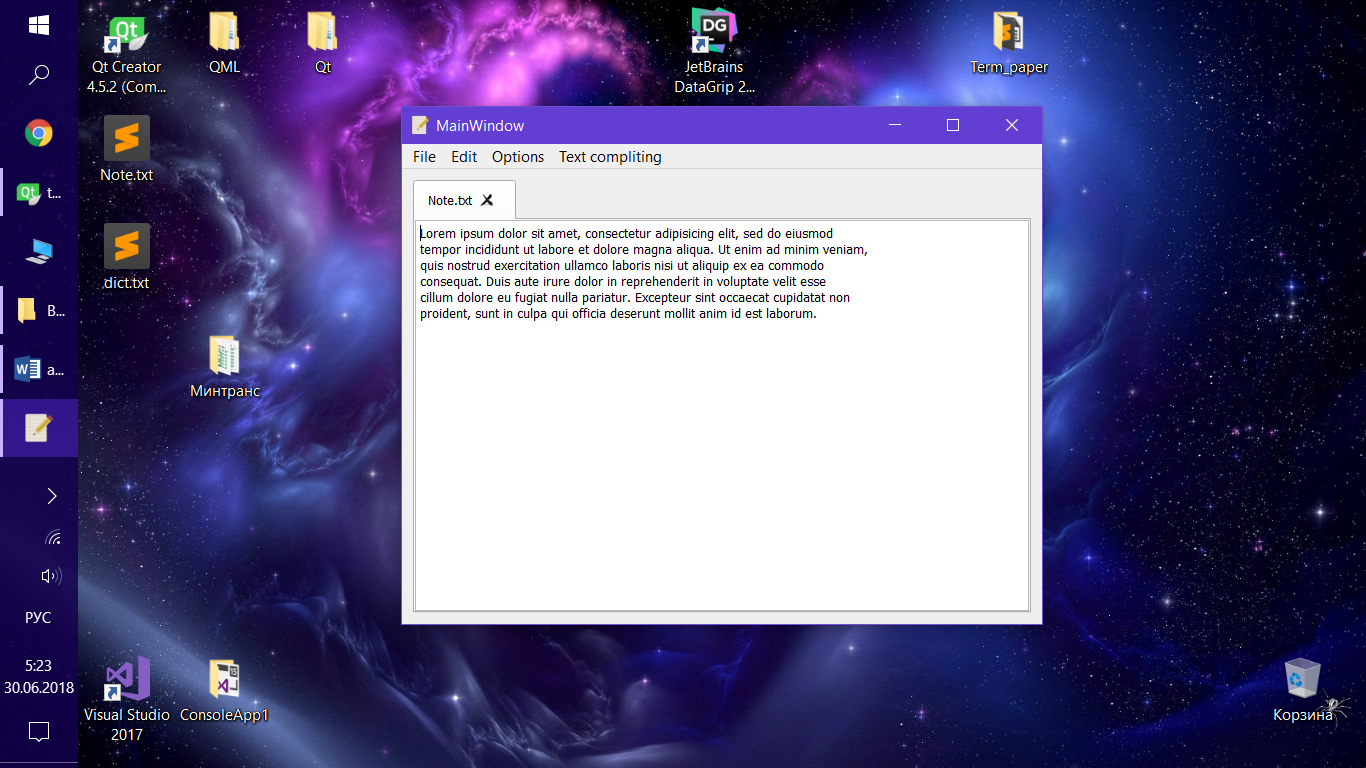
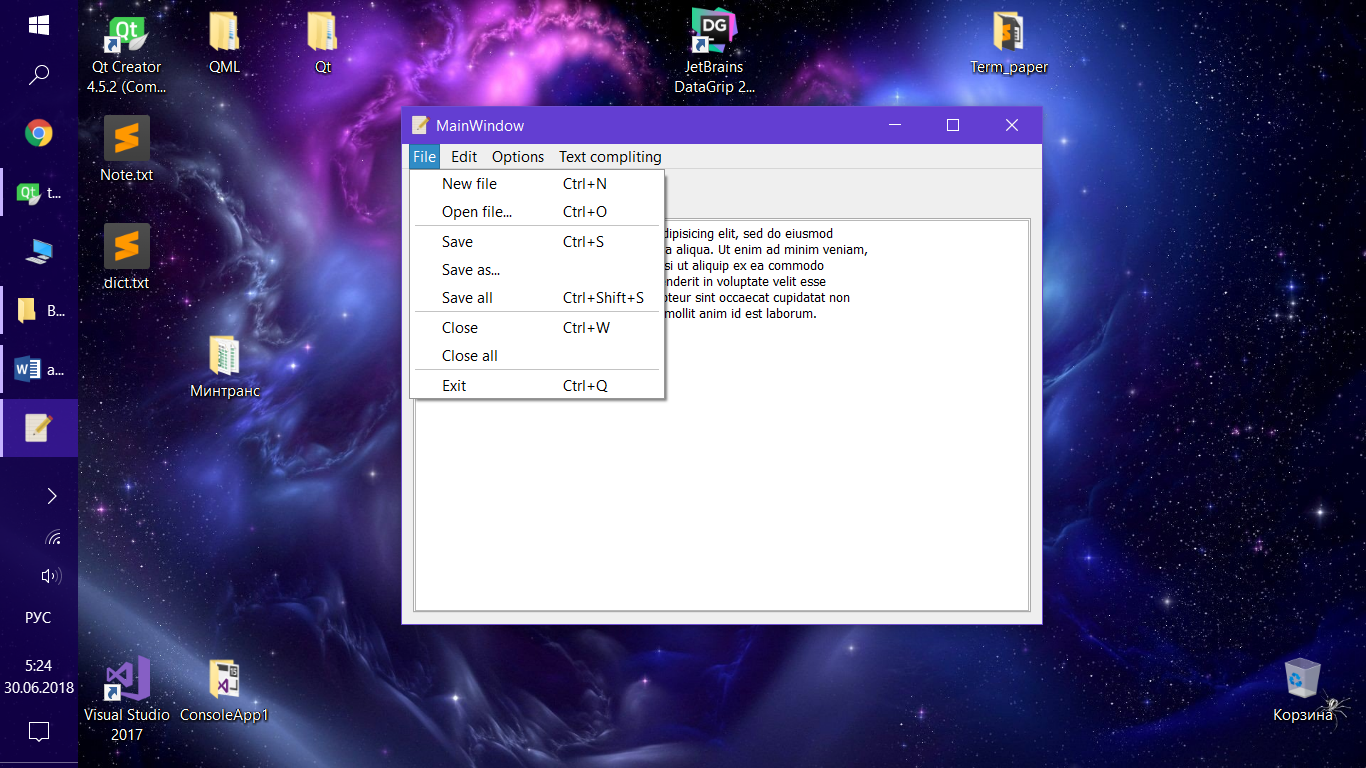
Trie &**operator** <<(const QStringList &dict); //добавляет список слов в дерево

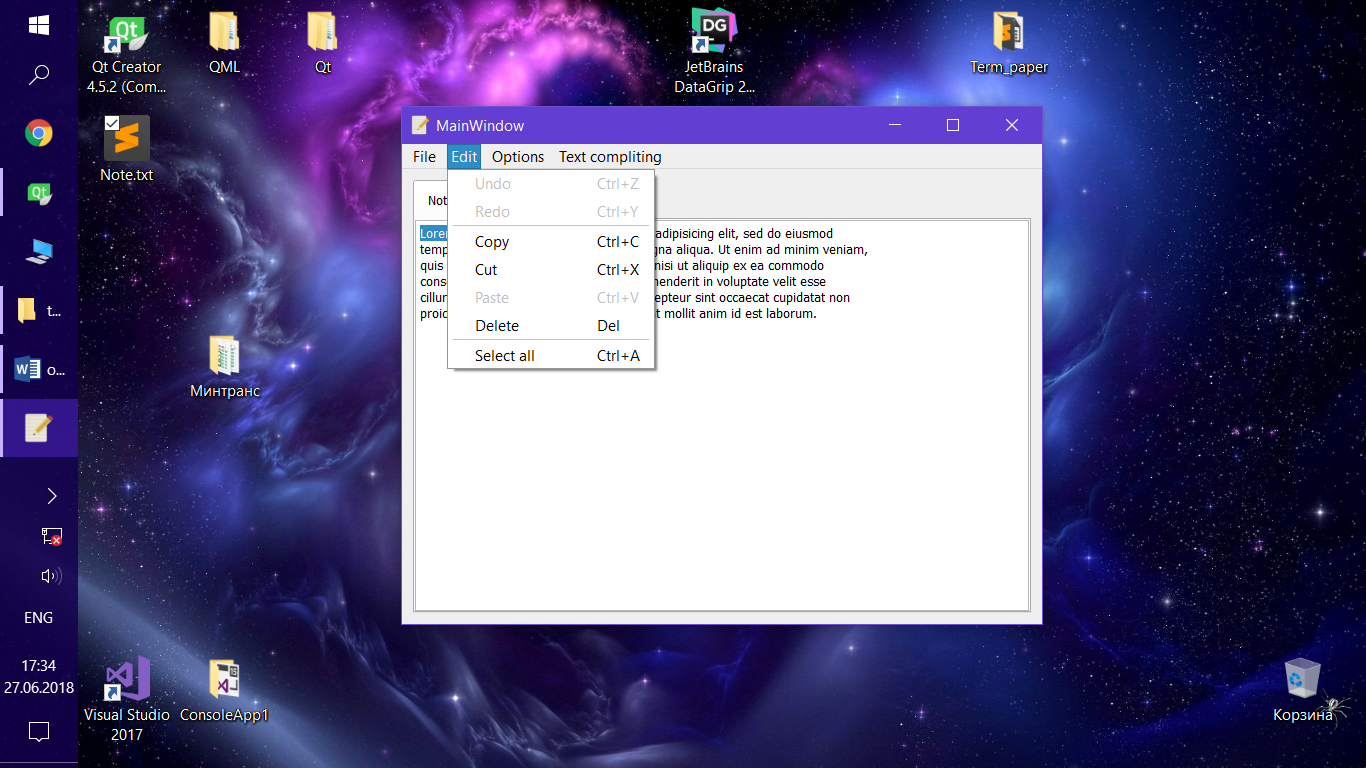
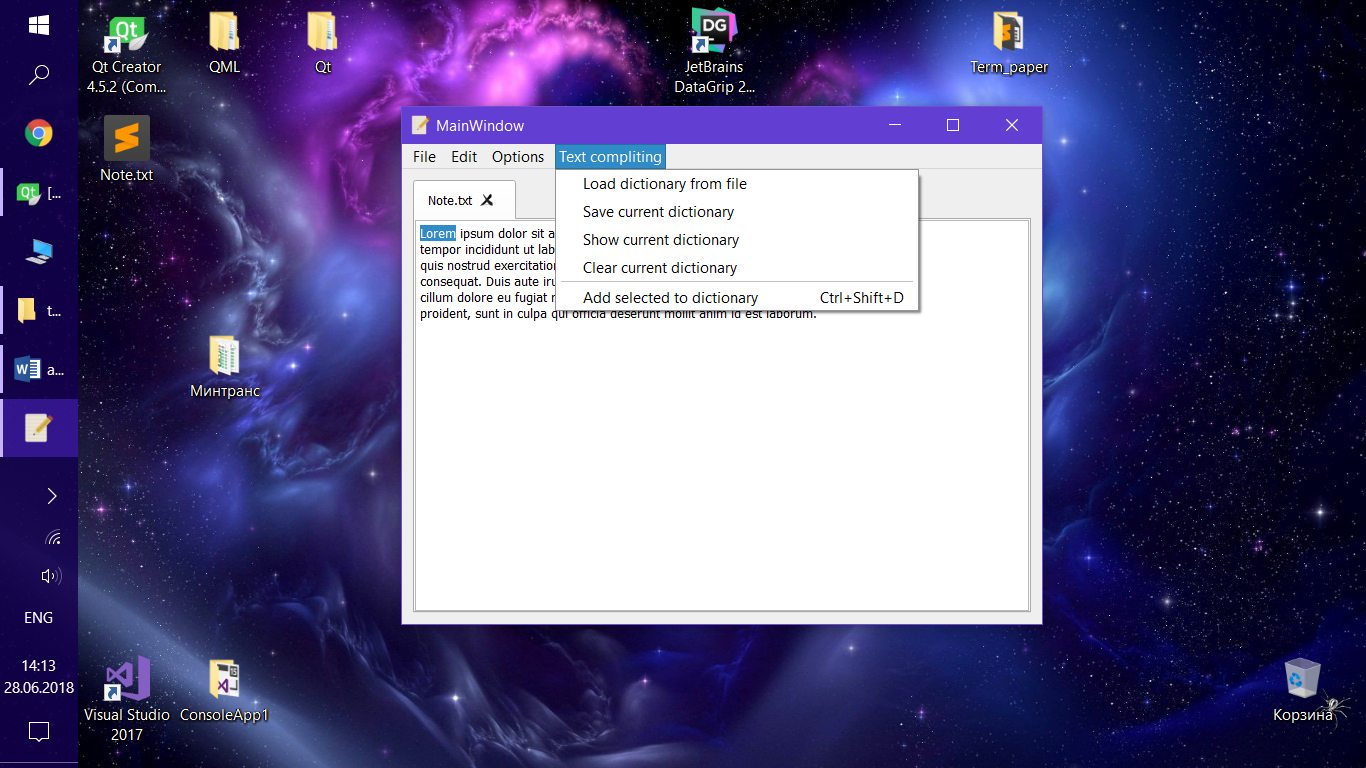
void **remove**(const QString &word); //удаляет слово из дерева

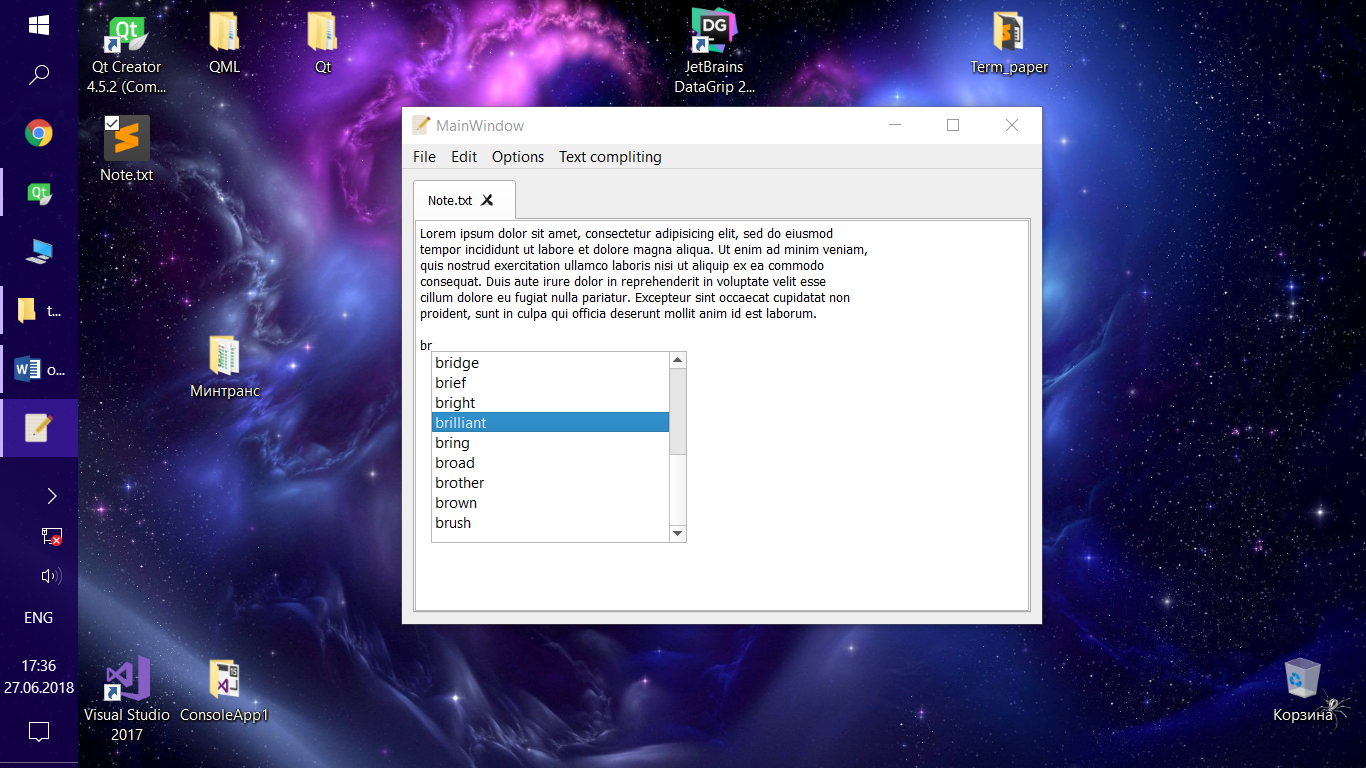
void **clear**();//удаляет все слова из дерева

};

# Общее руководство пользователя



## Комбинации клавиш

Ctrl + N – создать новый файл

Ctrl + O – открыть файл

Ctrl + S – сохранить файл

Ctrl + Shift + S – сохранить все файлы

Ctrl + W – закрыть файл

Ctrl + Q – выйти

Ctrl + Z – отменить

Ctrl + Y – повторить

Ctrl + C – копировать

Ctrl + X – вырезать

Ctrl + V – вставить

Del – удалить выделенное

Ctrl + A – выделить всё

Ctrl + Shift + D – добавить слово в словарь

# Анализ результатов работы программы

При реализации дерева при помощи массива для хранения потомков узлов, словарь с миллионом слов возвращает список всех слов примерно за 30 секунд, а при реализации с помощью красно-черного дерева эта операция занимает около 45 секунд, при реализации с помощью хэш-таблицы эта операция занимает недопустимо долгое время.

# Листинг программы

## main.cpp

#include "mainwindow.hpp"

#include <QApplication>

#include <QStyleFactory>

int main(int argc, char \*argv[])

{

QApplication::setStyle(QStyleFactory::create("Fusion"));

QApplication a(argc, argv);

MainWindow w;

w.setWindowIcon(QIcon(":/icon.ico"));

w.show();

return a.exec();

}

## mainwindow.hpp

#ifndef MAINWINDOW\_HPP

#define MAINWINDOW\_HPP

#include <QMainWindow>

#include <QDir>

#include <QFileDialog>

#include <QMessageBox>

#include <QCloseEvent>

#include <TextEdit/textedit.hpp>

#include "Trie/trie.hpp"

#include <QDebug>

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

bool isDictSaved;

QString lastDictPath;

protected:

void ***closeEvent***(QCloseEvent \*event) override;

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

private slots:

void **on\_actionExit\_triggered**();

void **on\_actionUndo\_triggered**();

void **on\_actionRedo\_triggered**();

void **on\_actionCopy\_triggered**();

void **on\_actionCut\_triggered**();

void **on\_actionPaste\_triggered**();

void **on\_actionDelete\_triggered**();

void **on\_actionSelect\_all\_triggered**();

void **on\_actionOpen\_file\_triggered**();

void **on\_actionSave\_triggered**();

void **on\_actionSave**(int index, bool newflname = false);

void **on\_actionSave\_all\_triggered**();

void **on\_actionClose\_triggered**();

void **on\_actionClose\_all\_triggered**();

void **on\_actionSave\_as\_triggered**();

void **on\_actionNew\_file\_triggered**();

void **on\_tabWidget\_tabCloseRequested**(int index);

void **on\_currentTabChanged**(int index);

void **on\_curentDocumentChanged**();

void **on\_actionAdd\_selected\_to\_dict\_triggered**();

void **on\_actionLoad\_dict\_from\_file\_triggered**();

void **on\_actionShow\_current\_dict\_triggered**();

void **on\_undoEnable**(bool enable);

void **on\_redoEnable**(bool enable);

void **on\_actionClear\_current\_dict\_triggered**();

void **on\_curentDocumentTextChanged**();

void **on\_actionSave\_current\_dict\_triggered**();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_HPP

## mainwindow.cpp

#include "mainwindow.hpp"

#include "ui\_mainwindow.h"

void MainWindow::closeEvent(QCloseEvent \*event)

{

if ( (ui->tabWidget->count() == 0 && ui->actionClose\_if\_no\_tabs->isChecked()) ||

QMessageBox::question(this, "Close", "Want to close?") == QMessageBox::Yes)

{

on\_actionClose\_all\_triggered();

if (!isDictSaved &&

!Trie::obj().isEmpty() &&

QMessageBox::question(this, "Save dictionary", "Dictionary changed. Want to save it?") == QMessageBox::Yes)

{

on\_actionSave\_current\_dict\_triggered();

}

QMainWindow::closeEvent(event);

}

else event->ignore();

}

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

isDictSaved(true),

lastDictPath(QDir::currentPath()),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

connect(ui->tabWidget, &QTabWidget::currentChanged, this, &MainWindow::on\_currentTabChanged);

connect(ui->actionAdd\_selected\_to\_dict, &QAction::triggered, this, &MainWindow::on\_curentDocumentChanged);

ui->actionClose\_if\_no\_tabs->setChecked(true);

ui->actionAdd\_selected\_to\_dict->setDisabled(true);

ui->actionCopy->setDisabled(true);

ui->actionCut->setDisabled(true);

ui->actionUndo->setDisabled(true);

ui->actionRedo->setDisabled(true);

ui->actionPaste->setDisabled(true);

ui->actionDelete->setDisabled(true);

ui->actionSelect\_all->setDisabled(true);

ui->actionSave->setDisabled(true);

ui->actionShow\_current\_dict->setDisabled(Trie::obj().isEmpty());

ui->actionClear\_current\_dict->setDisabled(Trie::obj().isEmpty());

on\_actionNew\_file\_triggered();

on\_curentDocumentChanged();

QFile f(":/dict.txt");

if (f.open(QIODevice::ReadOnly))

{

QString dict(f.readAll());

dict.remove('\r');

Trie::obj() << dict.split('\n', QString::SkipEmptyParts);

if (!Trie::obj().isEmpty())

{

ui->actionShow\_current\_dict->setEnabled(true);

ui->actionClear\_current\_dict->setEnabled(true);

}

f.close();

}

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_actionExit\_triggered()

{

close();

}

void MainWindow::on\_actionUndo\_triggered()

{

static\_cast<TextEdit \*>(ui->tabWidget->currentWidget())->undo();

}

void MainWindow::on\_actionRedo\_triggered()

{

static\_cast<TextEdit \*>(ui->tabWidget->currentWidget())->redo();

}

void MainWindow::on\_actionCopy\_triggered()

{

static\_cast<TextEdit \*>(ui->tabWidget->currentWidget())->copy();

}

void MainWindow::on\_actionCut\_triggered()

{

static\_cast<TextEdit \*>(ui->tabWidget->currentWidget())->cut();

}

void MainWindow::on\_actionPaste\_triggered()

{

static\_cast<TextEdit \*>(ui->tabWidget->currentWidget())->paste();

}

void MainWindow::on\_actionDelete\_triggered()

{

static\_cast<TextEdit \*>(ui->tabWidget->currentWidget())->textCursor().removeSelectedText();

}

void MainWindow::on\_actionSelect\_all\_triggered()

{

static\_cast<TextEdit \*>(ui->tabWidget->currentWidget())->selectAll();

}

void MainWindow::on\_actionOpen\_file\_triggered()

{

QString file = QFileDialog::getOpenFileName(this, "Open file", QDir::homePath());

if (file.isEmpty()) return;

TextEdit \*textEdit;

if (ui->tabWidget->count() == 0)

{

textEdit = new TextEdit();

ui->tabWidget->addTab(textEdit, "Untilted");

}

else

{

textEdit = static\_cast<TextEdit \*>(ui->tabWidget->currentWidget());

}

if (!textEdit->toPlainText().isEmpty() || !textEdit->fileName().isEmpty())

{

textEdit = new TextEdit();

ui->tabWidget->addTab(textEdit, QFileInfo(file).fileName());

}

if (textEdit->open(file))

{

ui->tabWidget->setTabText( ui->tabWidget->indexOf(textEdit), QFileInfo(file).fileName());

}

else

{

QMessageBox::warning(this, "Open file", QString("Failed to open file: \"%1\"").arg(file));

}

}

void MainWindow::on\_actionSave\_triggered()

{

on\_actionSave(ui->tabWidget->currentIndex());

}

void MainWindow::on\_actionSave(int index, bool newflname)

{

if (ui->tabWidget->count() > 0)

{

TextEdit \*textEdit = static\_cast<TextEdit \*>(ui->tabWidget->widget(index));

if (textEdit->fileName().isEmpty() || newflname)

{

QString file = QFileDialog::getSaveFileName(this, "Open file", QDir::homePath());

if (file.isEmpty()) return;

textEdit->setFileName(file);

QStringList t = file.split('/', QString::SkipEmptyParts);

if (!t.isEmpty()) ui->tabWidget->setTabText(index, t[t.size()-1]);

}

if (!textEdit->save(textEdit->fileName()))

{

QMessageBox::warning(this, "Open file", QString("Failed to save file: \"%1\"").arg(textEdit->fileName()));

}

}

}

void MainWindow::on\_actionSave\_all\_triggered()

{

for (int i = 0; i < ui->tabWidget->count(); i++)

{

TextEdit \*textEdit = static\_cast<TextEdit \*>(ui->tabWidget->widget(i));

if (!textEdit->saved())

{

on\_actionSave(i);

}

}

}

void MainWindow::on\_actionClose\_triggered()

{

on\_tabWidget\_tabCloseRequested(ui->tabWidget->currentIndex());

}

void MainWindow::on\_actionClose\_all\_triggered()

{

int n = ui->tabWidget->count();

while(n--)

{

ui->tabWidget->setCurrentWidget(0);

on\_tabWidget\_tabCloseRequested(0);

}

}

void MainWindow::on\_actionSave\_as\_triggered()

{

on\_actionSave(ui->tabWidget->currentIndex(), true);

}

void MainWindow::on\_actionNew\_file\_triggered()

{

int index = ui->tabWidget->addTab(new TextEdit(), "Untilted");

ui->tabWidget->setCurrentIndex(index);

ui->actionSave\_all->setEnabled(true);

ui->actionSave\_as->setEnabled(true);

ui->actionClose->setEnabled(true);

ui->actionClose\_all->setEnabled(true);

}

void MainWindow::on\_tabWidget\_tabCloseRequested(int index)

{

if (ui->tabWidget->count() > 0)

{

TextEdit \*textEdit = static\_cast<TextEdit \*>(ui->tabWidget->widget(index));

if (!textEdit->saved())

{

int ans = QMessageBox::question(this,

"File save",

QString("Save changes to file \"%1\" before closing?")

.arg(

textEdit->fileName().isEmpty()

?

ui->tabWidget->tabText(index)

:

textEdit->fileName()),

QMessageBox::No,

QMessageBox::Cancel,

QMessageBox::Yes);

if (ans == QMessageBox::Cancel) return;

if (ans == QMessageBox::Yes) on\_actionSave(index);

}

ui->tabWidget->removeTab(index);

textEdit->deleteLater();

if (ui->tabWidget->count() == 0)

{

if (ui->actionClose\_if\_no\_tabs->isChecked()) close();

else

{

ui->actionAdd\_selected\_to\_dict->setDisabled(true);

ui->actionCopy->setDisabled(true);

ui->actionCut->setDisabled(true);

ui->actionUndo->setDisabled(true);

ui->actionRedo->setDisabled(true);

ui->actionPaste->setDisabled(true);

ui->actionDelete->setDisabled(true);

ui->actionSelect\_all->setDisabled(true);

ui->actionSave->setDisabled(true);

ui->actionSave\_all->setDisabled(true);

ui->actionSave\_as->setDisabled(true);

ui->actionClose->setDisabled(true);

ui->actionClose\_all->setDisabled(true);

}

}

}

}

void MainWindow::on\_currentTabChanged(int index)

{

if (ui->tabWidget->count() > 0)

{

TextEdit \*te;

for (int i = 0; i < ui->tabWidget->count(); i++)

{

te = static\_cast<TextEdit \*>(ui->tabWidget->widget(i));

disconnect(te, &TextEdit::selectionChanged,

this, &MainWindow::on\_curentDocumentChanged);

disconnect(te, &TextEdit::undoAvailable,

this, &MainWindow::on\_undoEnable);

disconnect(te, &TextEdit::redoAvailable,

this, &MainWindow::on\_redoEnable);

disconnect(te, &TextEdit::textChanged,

this, &MainWindow::on\_curentDocumentTextChanged);

}

te = static\_cast<TextEdit \*>(ui->tabWidget->widget(index));

connect(te, &TextEdit::selectionChanged,

this, &MainWindow::on\_curentDocumentChanged);

connect(te, &TextEdit::undoAvailable,

this, &MainWindow::on\_undoEnable);

connect(te, &TextEdit::redoAvailable,

this, &MainWindow::on\_redoEnable);

connect(te, &TextEdit::textChanged,

this, &MainWindow::on\_curentDocumentTextChanged);

}

}

void MainWindow::on\_curentDocumentChanged()

{

TextEdit \*te = static\_cast<TextEdit \*>(ui->tabWidget->currentWidget());

QTextCursor cursor = te->textCursor();

bool selected = cursor.selectionStart() != cursor.selectionEnd();

ui->actionAdd\_selected\_to\_dict->setEnabled(selected);

ui->actionCopy->setEnabled(selected);

ui->actionCut->setEnabled(selected);

ui->actionDelete->setEnabled(selected);

ui->actionPaste->setEnabled(te->canPaste());

ui->actionSave->setDisabled(te->saved());

ui->actionShow\_current\_dict->setDisabled(Trie::obj().isEmpty());

}

void MainWindow::on\_actionAdd\_selected\_to\_dict\_triggered()

{

QString text = static\_cast<TextEdit \*>(ui->tabWidget->currentWidget())->textCursor().selectedText();

if (!text.isEmpty())

{

Trie::obj().addWord(text);

isDictSaved = false;

}

if (!Trie::obj().isEmpty())

{

ui->actionShow\_current\_dict->setEnabled(true);

ui->actionClear\_current\_dict->setEnabled(true);

}

}

void MainWindow::on\_actionLoad\_dict\_from\_file\_triggered()

{

if (!isDictSaved &&

!Trie::obj().isEmpty() &&

QMessageBox::question(this, "Save dictionary", "Dictionary changed. Want to save it?") == QMessageBox::Yes)

{

on\_actionSave\_current\_dict\_triggered();

}

QString filename = QFileDialog::getOpenFileName(this, "Open dictionary", lastDictPath);

if (!filename.isEmpty())

{

QFile f(filename);

if (f.open(QIODevice::ReadOnly))

{

QString dict(f.readAll());

dict.remove('\r');

Trie::obj() << dict.split('\n', QString::SkipEmptyParts);

if (!Trie::obj().isEmpty())

{

ui->actionShow\_current\_dict->setEnabled(true);

ui->actionClear\_current\_dict->setEnabled(true);

ui->actionSave\_current\_dict->setEnabled(true);

}

lastDictPath = filename;

isDictSaved = true;

f.close();

}

else

{

QMessageBox::warning(this, "Open dictionary", QString("Dictionary, named \"%1\" not loaded!").arg(filename));

}

}

}

void MainWindow::on\_actionShow\_current\_dict\_triggered()

{

if (Trie::obj().isEmpty())

{

QMessageBox::warning(this, "Open dictionary", "Dictionary is empty!");

}

else

{

QStringList dict = Trie::obj().words(QString());

on\_actionNew\_file\_triggered();

TextEdit \*te = static\_cast<TextEdit \*>(ui->tabWidget->currentWidget());

ui->tabWidget->setTabText(ui->tabWidget->currentIndex(), "Dictionary");

foreach (const QString &word, dict)

{

te->append(QString("<span>%1</span>").arg(word));

}

}

}

void MainWindow::on\_undoEnable(bool enable)

{

ui->actionUndo->setEnabled(enable);

}

void MainWindow::on\_redoEnable(bool enable)

{

ui->actionRedo->setEnabled(enable);

}

void MainWindow::on\_actionClear\_current\_dict\_triggered()

{

if (!Trie::obj().isEmpty()

&&

QMessageBox::question(this, "Clear curent dictionary", "Are you sure want to clear curent dictionary?", QMessageBox::No, QMessageBox::Yes) == QMessageBox::Yes)

{

Trie::obj().clear();

ui->actionShow\_current\_dict->setDisabled(true);

ui->actionSave\_current\_dict->setDisabled(true);

ui->actionClear\_current\_dict->setDisabled(true);

isDictSaved = false;

}

}

void MainWindow::on\_curentDocumentTextChanged()

{

ui->actionSave->setEnabled(true);

ui->actionSelect\_all->setDisabled(static\_cast<TextEdit \*>(ui->tabWidget->currentWidget())->toPlainText().isEmpty());

}

void MainWindow::on\_actionSave\_current\_dict\_triggered()

{

QString dictPath = QFileDialog::getSaveFileName(this, "Save dictionary", lastDictPath);

if (!dictPath.isEmpty())

{

QFile f(dictPath);

if (f.open(QIODevice::WriteOnly))

{

QStringList dict = Trie::obj().words(QString());

foreach (const QString &word, dict)

{

f.write(QString("%1\n").arg(word).toUtf8());

}

f.close();

lastDictPath = dictPath;

isDictSaved = true;

}

else

{

QMessageBox::warning(this, "Save dictionary", QString("Failed to save dictionary to file \"%1\"").arg(dictPath));

}

}

}

## textedit.hpp

#ifndef TEXTEDIT\_HPP

#define TEXTEDIT\_HPP

#include <QTextEdit>

#include <QListWidget>

#include <QFileDialog>

#include <QMessageBox>

#include <QShortcut>

#include <QApplication>

#include <QDesktopWidget>

#include <QWindow>

#include <QAction>

#include <QContextMenuEvent>

#include <QMenu>

#include "Trie/trie.hpp"

#include "suggestionslist.hpp"

#include <QDebug>

class TextEdit : public QTextEdit

{

Q\_OBJECT

int last\_pos;

SuggestionsList \*suggestions;

bool m\_saved;

Q\_PROPERTY(QString fileName READ fileName WRITE setFileName NOTIFY fileNameChanged)

QShortcut \*callSuggestionsShortcut;

QShortcut \*hideSuggestionsShortcut;

QString m\_fileName;

protected:

void ***contextMenuEvent***(QContextMenuEvent \*e) override;

private slots:

void **on\_callSuggestions**(int n = 3);

void **on\_textCursorChanged**();

public slots:

bool **open**(const QString &file);

bool **save**(const QString &filename);

void **setFileName**(const QString &fileName);

public:

explicit TextEdit(QWidget \*parent = nullptr);

bool **saved**() const;

QString **fileName**() const;

signals:

void **fileNameChanged**(QString fileName);

};

#endif // TEXTEDIT\_HPP

## textedit.cpp

#include "textedit.hpp"

#include "textedit\_p.hpp"

static QString p\_firstWordFind\_Helper(QTextDocument \*doc, int pos)

{

pos--;

QString word;

while (pos >= 0)

{

QChar ch = doc->characterAt(pos).toLower();

if (p\_isLetterOrNumber(ch)) word.prepend(ch);

else break;

pos--;

}

return word;

}

bool TextEdit::saved() const

{ return m\_saved; }

QString TextEdit::fileName() const

{ return m\_fileName; }

void TextEdit::contextMenuEvent(QContextMenuEvent \*e)

{

QMenu \*menu = createStandardContextMenu();

menu->addSeparator();

menu->addAction("Add word to dict",

[this](){

QString text = textCursor().selectedText();

if (!text.isEmpty()) Trie::obj().addWord(text);

})->setDisabled(textCursor().selectionStart() == textCursor().selectionEnd());

menu->exec(e->globalPos());

delete menu;

}

void TextEdit::on\_callSuggestions(int n)

{

QString preffix = p\_firstWordFind\_Helper(document(), textCursor().anchor());

if (preffix.size() >= n || suggestions->isVisible())

{

QStringList words = Trie::obj().words(preffix);

if (words.isEmpty()) suggestions->hideSuggestions();

else suggestions->showSuggestions(words);

}

}

void TextEdit::on\_textCursorChanged()

{

int pos = textCursor().anchor();

if (pos - last\_pos == 1) on\_callSuggestions();

else suggestions->hideSuggestions();

last\_pos = pos;

}

bool TextEdit::open(const QString &file)

{

if (!file.isEmpty())

{

QFile f(file);

if(f.open(QIODevice::ReadOnly))

{

setText(QString(f.readAll()));

f.close();

setFileName(file);

m\_saved = true;

return true;

}

}

return false;

}

bool TextEdit::save(const QString &filename)

{

if (filename.isEmpty()) return false;

QFile f(filename);

if (f.open(QIODevice::WriteOnly))

{

f.write(toPlainText().toUtf8());

f.close();

return m\_saved = true;

}

return false;

}

void TextEdit::setFileName(const QString &fileName)

{

if (m\_fileName == fileName)

return;

m\_saved = false;

m\_fileName = fileName;

emit fileNameChanged(m\_fileName);

}

TextEdit::TextEdit(QWidget \*parent) :

QTextEdit(parent),

last\_pos(textCursor().anchor()),

suggestions(new SuggestionsList(this)),

m\_saved(true),

callSuggestionsShortcut(new QShortcut(QKeySequence(Qt::CTRL | Qt::Key\_Space), this)),

hideSuggestionsShortcut(new QShortcut(QKeySequence(Qt::Key\_Escape), this))

{

connect(this, &TextEdit::textChanged, [this](){ m\_saved = false; } );

connect(this, &TextEdit::cursorPositionChanged, this, &TextEdit::on\_textCursorChanged);

connect(callSuggestionsShortcut, &QShortcut::activated, [=]() {

on\_callSuggestions(0);

});

connect(hideSuggestionsShortcut, &QShortcut::activated,

suggestions, &SuggestionsList::hideSuggestions);

}

## trie.hpp

#ifndef TRIE\_HPP

#define TRIE\_HPP

#include <QChar>

#include <stdexcept>

#include <QDebug>

static const int CAP = 70;

class Node\_p

{

friend class Trie;

#ifdef QT\_QTTEST\_MODULE\_H

friend class TrieTest;

#endif

//STATIC

static int **p\_indexHelper**(const QChar &c);

static QChar **p\_letterHelper**(int p\_indexHelper);

//FIELDS

Node\_p \*m\_parent;

Node\_p \*d[CAP] = {0x0};

bool isEnd;

//METHODS

Node\_p(Node\_p \*parent = 0x0);

Node\_p \***parent**() const;

Node\_p \***at**(const QChar &c) const;

bool **isEmpty**() const;

void **set**(const QChar &c, Node\_p \*val);

void **setParent**(Node\_p \*parent);

public:

virtual ~***Node\_p***();

};

class Trie

{

#ifdef QT\_QTTEST\_MODULE\_H

friend class TrieTest;

#endif

//STATIC

static void **p\_wordsHelper**(Node\_p \*node, QString word, QStringList &list, int n);

static Trie s\_trie;

public:

static Trie &**obj**();

private:

//FIELDS

Node\_p \*root;

//METHODS

Trie() : root(new Node\_p()) {}

Trie(const Trie &){}

Trie(Trie &){}

public:

virtual ~***Trie***();

bool **isEmpty**() const;

QStringList **words**(const QString &preffix, int n = -1) const;

bool **contains**(const QString &word) const;

void **addWord**(const QString &word);

Trie &**operator** <<(const QString &word);

Trie &**operator** <<(const QStringList &dict);

void **remove**(const QString &word);

void **clear**();

};

#endif // TRIE\_HPP

## trie.cpp

#include "trie.hpp"

int Node\_p::**p\_indexHelper**(const QChar &c)

{

QChar ch = c.toLower();

int index = ( ch.row() << 8 ) | ch.cell();

if ('a' <= index && index <= 'z') return index - 'a';

else if (1072 <= index && index <= 1103) return (index - 1072) + 26; //ru а-я

else if (index == 1105) return 58; //ru ё

else if('0' <= index && index <= '9') return (index - '0') + 59;

else if (index == '-') return 69;

else return -1;

}

QChar Node\_p::**p\_letterHelper**(int index)

{

if (0 <= index && index <= 25) return QChar(index + 'a');

else if (26 <= index && index <= 57) return QChar(index - 26 + 1072); //ru а-я

else if (index == 58) return QChar(1105); //ru ё

else if (59 <= index && index <= 68) return QChar(index - 59 + '0');

else if (index == 69) return QChar('-');

else return QChar();

}

Node\_p::**Node\_p**(Node\_p \*parent) :

m\_parent(parent),

isEnd(false)

{

}

Node\_p \*Node\_p::**parent**() const

{ return m\_parent; }

Node\_p \*Node\_p::**at**(const QChar &c) const

{

int index = p\_indexHelper(c);

if (index < 0) throw std::range\_error("in \"Node\_p \*Node\_p::operator [](const QChar &c)\" range error");

return d[index];

}

bool Node\_p::**isEmpty**() const

{

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

{

if (d[i] != 0x0) return false;

}

return !isEnd;

}

void Node\_p::**set**(const QChar &c, Node\_p \*node)

{

int index = p\_indexHelper(c);

if (index < 0) throw std::range\_error("in \"void Node\_p::set(const QChar &c, Node\_p \*node)\" range error");

delete d[index];

d[index] = node;

}

void Node\_p::**setParent**(Node\_p \*parent)

{ m\_parent = parent; }

Node\_p::~***Node\_p***()

{

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

delete d[i];

}

void Trie::**p\_wordsHelper**(Node\_p \*node, QString word, QStringList &list, int n)

{

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

{

Node\_p \*t = node->d[i];

if (t)

{

QChar c = Node\_p::p\_letterHelper(i);

if (t->isEnd)

{

list << word+c;

n--;

}

if (n) p\_wordsHelper(t, word+c, list, n);

else return;

}

}

}

Trie Trie::s\_trie = Trie();

Trie &Trie::**obj**()

{ return s\_trie; }

Trie::~***Trie***()

{ delete root; }

bool Trie::**isEmpty**() const

{ return root->isEmpty(); }

QStringList Trie::**words**(const QString &preffix, int n) const

{

Node\_p \*node = root;

foreach (const QChar &c, preffix)

{

if ( (node = node->at(c)) == 0x0) return QStringList();

}

QStringList s\_list;

QString str = preffix;

p\_wordsHelper(node, str, s\_list, n);

return s\_list;

}

bool Trie::**contains**(const QString &word) const

{

Node\_p \*node = root;

foreach (const QChar &c, word)

{

if ( (node = node->at(c)) == 0x0) return false;

}

return node->isEnd;

}

void Trie::**addWord**(const QString &word)

{

Node\_p \*node = root;

foreach (QChar c, word)

{

if (Node\_p::p\_indexHelper(c) < 0) c = '-';

Node\_p \*t = node->at(c);

if (t == 0x0)

{

t = new Node\_p(node);

node->set(c, t);

}

node = t;

}

node->isEnd = true;

}

Trie &Trie::**operator** <<(const QString &word)

{

addWord(word);

return \*this;

}

Trie &Trie::**operator** <<(const QStringList &dict)

{

foreach (const QString &word, dict)

{

addWord(word);

}

return \*this;

}

void Trie::**remove**(const QString &word)

{

Node\_p \*node = root;

QString stack;

foreach (const QChar &c, word)

{

if ( (node = node->at(c)) == 0x0) return;

stack.prepend(c);

}

node->isEnd = false;

foreach (const QChar &c, stack)

{

if ( (node = node->parent()) == 0x0) return;

if (!node->isEmpty()) return;

node->set(c, 0x0);

}

}

void Trie::**clear**()

{

delete root;

root = new Node\_p();

}

## suggestionslist.hpp

#ifndef SUGGESTIONSLIST\_HPP

#define SUGGESTIONSLIST\_HPP

#include <QTextEdit>

#include <QListWidget>

#include <QKeyEvent>

#include <QFocusEvent>

#include <QDebug>

class SuggestionsList : public QListWidget

{

Q\_OBJECT

protected:

void ***keyPressEvent***(QKeyEvent \*e) override;

void ***focusOutEvent***(QFocusEvent \*e) override;

private slots:

void **on\_itemActivated**(QListWidgetItem \*item);

public:

SuggestionsList(QTextEdit \*parent);

public slots:

void **showSuggestions**(const QStringList &suggestions);

void **hideSuggestions**();

signals:

void **focusLost**();

};

#endif // SUGGESTIONSLIST\_HPP

## suggestionslist.cpp

#include "suggestionslist.hpp"

#include "textedit\_p.hpp"

void SuggestionsList::***keyPressEvent***(QKeyEvent \*e)

{

int key = e->key();

QString text = e->text();

QTextEdit \*textEdit = static\_cast<QTextEdit \*>(parentWidget());

if (key == Qt::*Key\_Space*)

{

textEdit->textCursor().insertText(QString(QChar::*Space*));

e->ignore();

hideSuggestions();

return;

}

else if (key == Qt::*Key\_Backspace*)

{

textEdit->textCursor().deletePreviousChar();

e->ignore();

return;

}

else if (key == Qt::*Key\_Delete*)

{

textEdit->textCursor().deleteChar();

e->ignore();

return;

}

else if (key == Qt::*Key\_Left*)

{

QTextCursor cursor = textEdit->textCursor();

cursor.movePosition(QTextCursor::*Left*);

textEdit->setTextCursor(cursor);

e->ignore();

return;

}

else if (key == Qt::*Key\_Right*)

{

QTextCursor cursor = textEdit->textCursor();

cursor.movePosition(QTextCursor::*Right*);

textEdit->setTextCursor(cursor);

e->ignore();

return;

}

else if (!text.isEmpty())

{

foreach (QChar ch, text)

{

if (p\_isLetterOrNumber(ch.toLower()))

{

textEdit->textCursor().insertText(QString(ch));

e->ignore();

return;

}

}

}

QListWidget::*keyPressEvent*(e);

}

void SuggestionsList::***focusOutEvent***(QFocusEvent \*e)

{

if (e->lostFocus()) emit focusLost();

QListWidget::*focusOutEvent*(e);

}

void SuggestionsList::**on\_itemActivated**(QListWidgetItem \*item)

{

QString text = item->text();

QTextEdit \*textEdit = static\_cast<QTextEdit \*>(parentWidget());

hideSuggestions();

int p = textEdit->textCursor().anchor() - 1;

while (p >= 0)

{

QChar ch = textEdit->document()->characterAt(p).toLower();

if (p\_isLetterOrNumber(ch))

{

textEdit->textCursor().deletePreviousChar();

p--;

}

else break;

}

textEdit->textCursor().insertText(text);

}

SuggestionsList::**SuggestionsList**(QTextEdit \*parent) :

QListWidget(parent)

{

setTabKeyNavigation(true);

setFocusPolicy(Qt::*StrongFocus*);

setWindowFlags(Qt::*Tool* | Qt::*FramelessWindowHint*);

hide();

connect(this, &SuggestionsList::itemActivated, this, &SuggestionsList::on\_itemActivated);

connect(this, &SuggestionsList::focusLost, this, &SuggestionsList::hideSuggestions);

}

void SuggestionsList::**showSuggestions**(const QStringList &suggestions)

{

clear();

addItems(suggestions);

QTextEdit \*textEdit = static\_cast<QTextEdit \*>(parentWidget());

QPoint s\_point = textEdit->mapToGlobal(textEdit->geometry().topLeft());

QPoint point = s\_point + textEdit->cursorRect().bottomLeft();

setGeometry(QRect(point, *sizeHint*()));

show();

setCurrentRow(0);

textEdit->clearFocus();

setFocus();

activateWindow();

raise();

}

void SuggestionsList::**hideSuggestions**()

{

clearFocus();

static\_cast<QTextEdit \*>(parentWidget())->setFocus();

hide();

clear();

}

## textedit\_p.hpp

#ifndef TEXTEDIT\_P\_HPP

#define TEXTEDIT\_P\_HPP

class QChar;

bool **p\_isLetterOrNumber**(const QChar &ch);

#endif // TEXTEDIT\_P\_HPP

## textedit\_p.cpp

#include "textedit\_p.hpp"

#include <QChar>

bool **p\_isLetterOrNumber**(const QChar &ch)

{

QChar c = ch.toLower();

int d = ( c.row() << 8 ) | c.cell();

return (

('a' <= d && d <= 'z') ||

('0' <= d && d <= '9') ||

(1072 <= d && d <= 1103) ||

(d == 1105) ||

(d == '-')

);

}