**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 3 з дисципліни

«Проектування алгоритмів»

„ **Проектування структур даних**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

Присяжний А. О.

**Перевірив**

(прізвище, ім'я, по батькові)

*Ахаладзе І. Е..*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc114359761)

[2 Завдання 4](#_Toc114359762)

[3 Виконання 7](#_Toc114359763)

[3.1 Псевдокод алгоритмів 7](#_Toc114359764)

[3.2 Часова складність пошуку 7](#_Toc114359765)

[3.3 Програмна реалізація 7](#_Toc114359766)

[3.3.1 Вихідний код 7](#_Toc114359767)

[3.3.2 Приклади роботи 7](#_Toc114359768)

[3.4 Тестування алгоритму 8](#_Toc114359769)

[3.4.1 Часові характеристики оцінювання 8](#_Toc114359770)

[Висновок 9](#_Toc114359771)

[Критерії оцінювання 10](#_Toc114359772)

# Мета лабораторної роботи

Мета роботи – вивчити основні підходи проектування та обробки складних структур даних.

# Завдання

Відповідно до варіанту (таблиця 2.1), записати алгоритми пошуку, додавання, видалення і редагування запису в структурі даних за допомогою псевдокоду (чи іншого способу по вибору).

Записати часову складність пошуку в структурі в асимптотичних оцінках.

Виконати програмну реалізацію невеликої СУБД з графічним (не консольним) інтерфейсом користувача (дані БД мають зберігатися на ПЗП), з функціями пошуку (алгоритм пошуку у вузлі структури згідно варіанту таблиця 2.1, за необхідності), додавання, видалення та редагування записів (запис складається із ключа і даних, ключі унікальні і цілочисельні, даних може бути декілька полів для одного ключа, але достатньо одного рядка фіксованої довжини). Для зберігання даних використовувати структуру даних згідно варіанту (таблиця 2.1).

Заповнити базу випадковими значеннями до 10000 і зафіксувати середнє (із 10-15 пошуків) число порівнянь для знаходження запису по ключу.

Зробити висновок з лабораторної роботи.

Таблиця 2.1 – Варіанти алгоритмів

|  |  |
| --- | --- |
| **№** | **Структура даних** |
| 1 | Файли з щільним індексом з перебудовою індексної області, бінарний пошук |
| 2 | Файли з щільним індексом з областю переповнення, бінарний пошук |
| 3 | Файли з не щільним індексом з перебудовою індексної області, бінарний пошук |
| 4 | Файли з не щільним індексом з областю переповнення, бінарний пошук |
| 5 | АВЛ-дерево |
| 6 | Червоно-чорне дерево |
| 7 | B-дерево t=10, бінарний пошук |
| 8 | B-дерево t=25, бінарний пошук |
| 9 | B-дерево t=50, бінарний пошук |
| 10 | B-дерево t=100, бінарний пошук |
| 11 | Файли з щільним індексом з перебудовою індексної області, однорідний бінарний пошук |
| 12 | Файли з щільним індексом з областю переповнення, однорідний бінарний пошук |
| 13 | Файли з не щільним індексом з перебудовою індексної області, однорідний бінарний пошук |
| 14 | Файли з не щільним індексом з областю переповнення, однорідний бінарний пошук |
| 15 | АВЛ-дерево |
| 16 | Червоно-чорне дерево |
| 17 | B-дерево t=10, однорідний бінарний пошук |
| 18 | B-дерево t=25, однорідний бінарний пошук |
| 19 | B-дерево t=50, однорідний бінарний пошук |
| 20 | B-дерево t=100, однорідний бінарний пошук |
| 21 | Файли з щільним індексом з перебудовою індексної області, метод Шарра |
| 22 | Файли з щільним індексом з областю переповнення, метод Шарра |
| 23 | Файли з не щільним індексом з перебудовою індексної області, метод Шарра |
| 24 | Файли з не щільним індексом з областю переповнення, метод Шарра |
| 25 | АВЛ-дерево |
| 26 | Червоно-чорне дерево |
| 27 | B-дерево t=10, метод Шарра |
| 28 | B-дерево t=25, метод Шарра |
| 29 | B-дерево t=50, метод Шарра |
| 30 | B-дерево t=100, метод Шарра |
| 31 | АВЛ-дерево |
| 32 | Червоно-чорне дерево |
| 33 | B-дерево t=250, бінарний пошук |
| 34 | B-дерево t=250, однорідний бінарний пошук |
| 35 | B-дерево t=250, метод Шарра |

# Виконання

## Псевдокод алгоритмів

**Search(key)** {

left = 0;

right = records.size() - 1;

**while** (left <= right) **do**

mid = left + (right - left) / 2;

**if** (records[mid].key == key) **do**

**return** records[mid].key;

**if** (records[mid].key < key) **do**

left = mid + 1;

**else**

right = mid - 1;

**end while**

**if** поточний вузол - листок

**повернути** помилку, такого ключа немає

**return** children[left]->search(key);

}

## Часова складність пошуку

Часова складність бінарного пошуку – O(log(N)).

## Програмна реалізація

### Вихідний код

**data\_converters.h**

#pragma once

#include <string>

#include <msclr\marshal\_cppstd.h>

using namespace std;

using namespace System;

using namespace Windows::Forms;

string String\_to\_string(String^ data);

String^ string\_to\_String(string data);

const char\* String\_to\_char\_string(String^ data);

String^ char\_string\_into\_String(const char\* data);

**data\_converters.cpp**

#include "Data\_converters.h"

string String\_to\_string(String^ data) {

return msclr::interop::marshal\_as<string>(data);

}

String^ string\_to\_String(string data) {

return msclr::interop::marshal\_as<String^>(data);

}

const char\* String\_to\_char\_string(String^ data) {

string str\_data = String\_to\_string(data);

return str\_data.c\_str();

}

String^ char\_string\_into\_String(const char\* data) {

return msclr::interop::marshal\_as<String^>(data);

}

**data\_validators.h**

#pragma once

#include "Data\_converters.h"

#include <fstream>

bool is\_number(const string& input);

int get\_input(TextBox^ source);

bool is\_empty\_file(ifstream& file);

**data\_validators.cpp**

#include "Data\_validators.h"

bool is\_number(const string& input) {

for (char ch : input) {

if (!isdigit(ch))

return false;

}

return true;

}

int get\_input(TextBox^ source) {

string data = String\_to\_string(source->Text);

if (data == "")

throw "You must enter a key";

if (!is\_number(data))

throw "You must enter a positive integer number";

if (data.length() > 1 && data[0] == '0')

throw "Number can't start with 0";

return stoi(data);

}

bool is\_empty\_file(ifstream& file) {

file.seekg(0, ios::end);

bool result = file.tellg() == 0;

file.seekg(0, ios::beg);

return result;

}

**DBForm.h**

#pragma once

#include <fstream>

#include "../Lab3\_code/validation\_functions.h"

#include "DBManagement.h"

namespace Lab3 {

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;

/// <summary>

/// Summary for DBForm

/// </summary>

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

{

public:

DBForm(void)

{

InitializeComponent();

//

//TODO: Add the constructor code here

//

}

protected:

/// <summary>

/// Clean up any resources being used.

/// </summary>

~DBForm()

{

if (components)

{

delete components;

}

}

private:

void open\_db\_management\_window(String^ file\_path, bool open);

private: System::Windows::Forms::Button^ DB\_creation\_btn;

private: System::Windows::Forms::Button^ DB\_opening\_btn;

protected:

protected:

private:

/// <summary>

/// Required designer variable.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

void InitializeComponent(void)

{

this->DB\_creation\_btn = (gcnew System::Windows::Forms::Button());

this->DB\_opening\_btn = (gcnew System::Windows::Forms::Button());

this->SuspendLayout();

//

// DB\_creation\_btn

//

this->DB\_creation\_btn->Location = System::Drawing::Point(67, 38);

this->DB\_creation\_btn->Name = L"DB\_creation\_btn";

this->DB\_creation\_btn->Size = System::Drawing::Size(148, 75);

this->DB\_creation\_btn->TabIndex = 0;

this->DB\_creation\_btn->Text = L"Create new DB";

this->DB\_creation\_btn->UseVisualStyleBackColor = true;

this->DB\_creation\_btn->Click += gcnew System::EventHandler(this, &DBForm::DB\_creation\_btn\_Click);

//

// DB\_opening\_btn

//

this->DB\_opening\_btn->Location = System::Drawing::Point(67, 148);

this->DB\_opening\_btn->Name = L"DB\_opening\_btn";

this->DB\_opening\_btn->Size = System::Drawing::Size(148, 75);

this->DB\_opening\_btn->TabIndex = 1;

this->DB\_opening\_btn->Text = L"Open existing DB";

this->DB\_opening\_btn->UseVisualStyleBackColor = true;

this->DB\_opening\_btn->Click += gcnew System::EventHandler(this, &DBForm::DB\_opening\_btn\_Click);

//

// DBForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);

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

this->ClientSize = System::Drawing::Size(286, 285);

this->Controls->Add(this->DB\_opening\_btn);

this->Controls->Add(this->DB\_creation\_btn);

this->FormBorderStyle = System::Windows::Forms::FormBorderStyle::FixedSingle;

this->MaximizeBox = false;

this->Name = L"DBForm";

this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;

this->Text = L"BTree Database";

this->Load += gcnew System::EventHandler(this, &DBForm::DBForm\_Load);

this->ResumeLayout(false);

}

#pragma endregion

private: System::Void DBForm\_Load(System::Object^ sender, System::EventArgs^ e) {

}

private: Void DB\_creation\_btn\_Click(Object^ sender, EventArgs^ e);

Void DB\_opening\_btn\_Click(Object^ sender, EventArgs^ e);

};

}

**DBForm.cpp**

#include "DBForm.h"

using namespace Lab3;

using namespace std;

Void DBForm::DB\_creation\_btn\_Click(Object^ sender, EventArgs^ e) {

SaveFileDialog^ file\_dialog = gcnew SaveFileDialog();

file\_dialog->Filter = "My data base file | \*.mdb";

file\_dialog->Title = "Create data base";

file\_dialog->InitialDirectory = Application::StartupPath;

if (file\_dialog->ShowDialog() == System::Windows::Forms::DialogResult::OK)

open\_db\_management\_window(file\_dialog->FileName, false);

}

Void DBForm::DB\_opening\_btn\_Click(Object^ sender, EventArgs^ e) {

OpenFileDialog^ file\_dialog = gcnew OpenFileDialog();

file\_dialog->Filter = "My data base file | \*.mdb";

file\_dialog->Title = "Open data base file";

file\_dialog->InitialDirectory = Application::StartupPath;

if (file\_dialog->ShowDialog() == System::Windows::Forms::DialogResult::OK)

open\_db\_management\_window(file\_dialog->FileName, true);

}

void DBForm::open\_db\_management\_window(String^ file\_path, bool open) {

string extension = ".mdb";

string file\_name = String\_to\_string(file\_path);

validate\_file\_path(file\_name, extension);

DBManagement^ db\_management\_window = gcnew DBManagement(file\_name, open);

db\_management\_window->Show();

}

**DBManagement.h**

#pragma once

#include "../Lab3\_code/B\_Tree.h"

#include "Data\_validators.h"

#include <string>

#include "random\_generators.h"

namespace Lab3 {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Summary for DBManagement

/// </summary>

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

{

public:

DBManagement(void)

{

InitializeComponent();

//

//TODO: Add the constructor code here

//

}

DBManagement(std::string file\_path\_to\_save, bool open);

protected:

/// <summary>

/// Clean up any resources being used.

/// </summary>

~DBManagement();

private:

int minimum\_degree;

std::string\* file\_path\_to\_save;

private: System::Windows::Forms::Button^ records\_addition\_btn;

BTree\* tree;

void display();

void remove\_row(int key);

void edit\_row(int key, char\* data);

void disable\_edit\_delete\_search();

void enable\_edit\_delete\_search();

void disable\_records\_addition();

void enable\_records\_addition();

private: System::Windows::Forms::DataGridView^ data\_table;

protected:

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

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

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

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

private: System::Windows::Forms::Button^ insertion\_btn;

private: System::Windows::Forms::TextBox^ key\_to\_insert;

private: System::Windows::Forms::TextBox^ data\_to\_insert;

private: System::Windows::Forms::TextBox^ key\_to\_delete;

private: System::Windows::Forms::Button^ deletion\_btn;

private: System::Windows::Forms::Label^ label3;

private: System::Windows::Forms::TextBox^ new\_data;

private: System::Windows::Forms::TextBox^ key\_to\_edit;

private: System::Windows::Forms::Button^ editing\_btn;

private: System::Windows::Forms::Label^ label5;

private: System::Windows::Forms::Label^ label6;

private: System::Windows::Forms::TextBox^ key\_to\_find;

private: System::Windows::Forms::Button^ find\_btn;

private: System::Windows::Forms::Label^ label8;

protected:

private:

/// <summary>

/// Required designer variable.

/// </summary>

System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

void InitializeComponent(void)

{

this->data\_table = (gcnew System::Windows::Forms::DataGridView());

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

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

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

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

this->insertion\_btn = (gcnew System::Windows::Forms::Button());

this->key\_to\_insert = (gcnew System::Windows::Forms::TextBox());

this->data\_to\_insert = (gcnew System::Windows::Forms::TextBox());

this->key\_to\_delete = (gcnew System::Windows::Forms::TextBox());

this->deletion\_btn = (gcnew System::Windows::Forms::Button());

this->label3 = (gcnew System::Windows::Forms::Label());

this->new\_data = (gcnew System::Windows::Forms::TextBox());

this->key\_to\_edit = (gcnew System::Windows::Forms::TextBox());

this->editing\_btn = (gcnew System::Windows::Forms::Button());

this->label5 = (gcnew System::Windows::Forms::Label());

this->label6 = (gcnew System::Windows::Forms::Label());

this->key\_to\_find = (gcnew System::Windows::Forms::TextBox());

this->find\_btn = (gcnew System::Windows::Forms::Button());

this->label8 = (gcnew System::Windows::Forms::Label());

this->records\_addition\_btn = (gcnew System::Windows::Forms::Button());

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

this->SuspendLayout();

//

// data\_table

//

this->data\_table->AllowUserToAddRows = false;

this->data\_table->BackgroundColor = System::Drawing::SystemColors::ButtonHighlight;

this->data\_table->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->data\_table->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(2) { this->Key, this->Data });

this->data\_table->GridColor = System::Drawing::SystemColors::ButtonHighlight;

this->data\_table->Location = System::Drawing::Point(-1, 183);

this->data\_table->Name = L"data\_table";

this->data\_table->RowHeadersVisible = false;

this->data\_table->RowHeadersWidth = 51;

this->data\_table->RowTemplate->Height = 24;

this->data\_table->Size = System::Drawing::Size(1008, 354);

this->data\_table->TabIndex = 0;

//

// Key

//

this->Key->AutoSizeMode = System::Windows::Forms::DataGridViewAutoSizeColumnMode::Fill;

this->Key->HeaderText = L"Key";

this->Key->MinimumWidth = 6;

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

this->Key->ReadOnly = true;

//

// Data

//

this->Data->AutoSizeMode = System::Windows::Forms::DataGridViewAutoSizeColumnMode::Fill;

this->Data->HeaderText = L"Data";

this->Data->MinimumWidth = 6;

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

this->Data->ReadOnly = true;

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(12, 9);

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

this->label1->Size = System::Drawing::Size(85, 16);

this->label1->TabIndex = 1;

this->label1->Text = L"Key to insert :";

//

// label2

//

this->label2->AutoSize = true;

this->label2->Location = System::Drawing::Point(12, 35);

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

this->label2->Size = System::Drawing::Size(91, 16);

this->label2->TabIndex = 2;

this->label2->Text = L"Data to insert :";

//

// insertion\_btn

//

this->insertion\_btn->Location = System::Drawing::Point(72, 63);

this->insertion\_btn->Name = L"insertion\_btn";

this->insertion\_btn->Size = System::Drawing::Size(102, 40);

this->insertion\_btn->TabIndex = 3;

this->insertion\_btn->Text = L"Insert";

this->insertion\_btn->UseVisualStyleBackColor = true;

this->insertion\_btn->Click += gcnew System::EventHandler(this, &DBManagement::insertion\_btn\_Click);

//

// key\_to\_insert

//

this->key\_to\_insert->Location = System::Drawing::Point(103, 6);

this->key\_to\_insert->MaxLength = 6;

this->key\_to\_insert->Name = L"key\_to\_insert";

this->key\_to\_insert->Size = System::Drawing::Size(130, 22);

this->key\_to\_insert->TabIndex = 4;

//

// data\_to\_insert

//

this->data\_to\_insert->Location = System::Drawing::Point(103, 35);

this->data\_to\_insert->MaxLength = 30;

this->data\_to\_insert->Name = L"data\_to\_insert";

this->data\_to\_insert->Size = System::Drawing::Size(130, 22);

this->data\_to\_insert->TabIndex = 5;

//

// key\_to\_delete

//

this->key\_to\_delete->Location = System::Drawing::Point(354, 18);

this->key\_to\_delete->MaxLength = 6;

this->key\_to\_delete->Name = L"key\_to\_delete";

this->key\_to\_delete->Size = System::Drawing::Size(130, 22);

this->key\_to\_delete->TabIndex = 9;

//

// deletion\_btn

//

this->deletion\_btn->Location = System::Drawing::Point(323, 63);

this->deletion\_btn->Name = L"deletion\_btn";

this->deletion\_btn->Size = System::Drawing::Size(102, 40);

this->deletion\_btn->TabIndex = 8;

this->deletion\_btn->Text = L"Delete";

this->deletion\_btn->UseVisualStyleBackColor = true;

this->deletion\_btn->Click += gcnew System::EventHandler(this, &DBManagement::deletion\_btn\_Click);

//

// label3

//

this->label3->AutoSize = true;

this->label3->Location = System::Drawing::Point(257, 21);

this->label3->Name = L"label3";

this->label3->Size = System::Drawing::Size(91, 16);

this->label3->TabIndex = 6;

this->label3->Text = L"Key to delete :";

//

// new\_data

//

this->new\_data->Location = System::Drawing::Point(603, 32);

this->new\_data->MaxLength = 30;

this->new\_data->Name = L"new\_data";

this->new\_data->Size = System::Drawing::Size(130, 22);

this->new\_data->TabIndex = 15;

//

// key\_to\_edit

//

this->key\_to\_edit->Location = System::Drawing::Point(603, 3);

this->key\_to\_edit->MaxLength = 6;

this->key\_to\_edit->Name = L"key\_to\_edit";

this->key\_to\_edit->Size = System::Drawing::Size(130, 22);

this->key\_to\_edit->TabIndex = 14;

//

// editing\_btn

//

this->editing\_btn->Location = System::Drawing::Point(572, 60);

this->editing\_btn->Name = L"editing\_btn";

this->editing\_btn->Size = System::Drawing::Size(102, 40);

this->editing\_btn->TabIndex = 13;

this->editing\_btn->Text = L"Edit";

this->editing\_btn->UseVisualStyleBackColor = true;

this->editing\_btn->Click += gcnew System::EventHandler(this, &DBManagement::editing\_btn\_Click);

//

// label5

//

this->label5->AutoSize = true;

this->label5->Location = System::Drawing::Point(512, 32);

this->label5->Name = L"label5";

this->label5->Size = System::Drawing::Size(70, 16);

this->label5->TabIndex = 12;

this->label5->Text = L"New data :";

//

// label6

//

this->label6->AutoSize = true;

this->label6->Location = System::Drawing::Point(512, 6);

this->label6->Name = L"label6";

this->label6->Size = System::Drawing::Size(75, 16);

this->label6->TabIndex = 11;

this->label6->Text = L"Key to edit :";

//

// key\_to\_find

//

this->key\_to\_find->Location = System::Drawing::Point(842, 18);

this->key\_to\_find->MaxLength = 6;

this->key\_to\_find->Name = L"key\_to\_find";

this->key\_to\_find->Size = System::Drawing::Size(130, 22);

this->key\_to\_find->TabIndex = 19;

//

// find\_btn

//

this->find\_btn->Location = System::Drawing::Point(822, 63);

this->find\_btn->Name = L"find\_btn";

this->find\_btn->Size = System::Drawing::Size(102, 40);

this->find\_btn->TabIndex = 18;

this->find\_btn->Text = L"Find";

this->find\_btn->UseVisualStyleBackColor = true;

this->find\_btn->Click += gcnew System::EventHandler(this, &DBManagement::find\_btn\_Click);

//

// label8

//

this->label8->AutoSize = true;

this->label8->Location = System::Drawing::Point(762, 21);

this->label8->Name = L"label8";

this->label8->Size = System::Drawing::Size(74, 16);

this->label8->TabIndex = 16;

this->label8->Text = L"Key to find :";

//

// records\_addition\_btn

//

this->records\_addition\_btn->Location = System::Drawing::Point(872, 125);

this->records\_addition\_btn->Name = L"records\_addition\_btn";

this->records\_addition\_btn->Size = System::Drawing::Size(100, 52);

this->records\_addition\_btn->TabIndex = 20;

this->records\_addition\_btn->Text = L"Add 1000 records";

this->records\_addition\_btn->UseVisualStyleBackColor = true;

this->records\_addition\_btn->Click += gcnew System::EventHandler(this, &DBManagement::records\_addition\_btn\_Click);

//

// DBManagement

//

this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);

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

this->ClientSize = System::Drawing::Size(1004, 536);

this->Controls->Add(this->records\_addition\_btn);

this->Controls->Add(this->key\_to\_find);

this->Controls->Add(this->find\_btn);

this->Controls->Add(this->label8);

this->Controls->Add(this->new\_data);

this->Controls->Add(this->key\_to\_edit);

this->Controls->Add(this->editing\_btn);

this->Controls->Add(this->label5);

this->Controls->Add(this->label6);

this->Controls->Add(this->key\_to\_delete);

this->Controls->Add(this->deletion\_btn);

this->Controls->Add(this->label3);

this->Controls->Add(this->data\_to\_insert);

this->Controls->Add(this->key\_to\_insert);

this->Controls->Add(this->insertion\_btn);

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

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

this->Controls->Add(this->data\_table);

this->FormBorderStyle = System::Windows::Forms::FormBorderStyle::FixedSingle;

this->MaximizeBox = false;

this->Name = L"DBManagement";

this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;

this->Text = L"DBManagement";

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

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

private: Void insertion\_btn\_Click(Object^ sender, EventArgs^ e);

Void deletion\_btn\_Click(Object^ sender, EventArgs^ e);

Void editing\_btn\_Click(Object^ sender, EventArgs^ e);

Void find\_btn\_Click(Object^ sender, EventArgs^ e);

Void records\_addition\_btn\_Click(Object^ sender, EventArgs^ e);

};

}

**DBManagement.cpp**

#include "DBManagement.h"

#define MINIMUM\_DEGREE 100;

using namespace Lab3;

DBManagement::DBManagement(string file\_path\_to\_save, bool open) {

InitializeComponent();

this->minimum\_degree = MINIMUM\_DEGREE;

this->file\_path\_to\_save = new string(file\_path\_to\_save);

string a = \*this->file\_path\_to\_save;

this->tree = new BTree(this->minimum\_degree);

if (!open) {

disable\_edit\_delete\_search();

return;

}

ifstream file(file\_path\_to\_save, ios::binary);

if (!file) {

MessageBox::Show("Can't open file");

return;

}

if (is\_empty\_file(file)) {

disable\_edit\_delete\_search();

return;

}

disable\_records\_addition();

tree->open(file);

display();

file.close();

}

Void DBManagement::insertion\_btn\_Click(Object^ sender, EventArgs^ e) {

Record rec;

try {

rec.key = get\_input(key\_to\_insert);

}

catch (const char\* er) {

MessageBox::Show(char\_string\_into\_String(er));

return;

}

strcpy(rec.data, String\_to\_string(data\_to\_insert->Text).c\_str());

try {

tree->insert(rec);

key\_to\_insert->Text = "";

data\_to\_insert->Text = "";

display();

}

catch (const char\* er) {

MessageBox::Show(char\_string\_into\_String(er));

}

}

Void DBManagement::deletion\_btn\_Click(Object^ sender, EventArgs^ e) {

int key;

try {

key = get\_input(key\_to\_delete);

}

catch (const char\* er) {

MessageBox::Show(char\_string\_into\_String(er));

return;

}

try {

tree->remove(key);

key\_to\_delete->Text = "";

remove\_row(key);

}

catch (string er) {

MessageBox::Show(string\_to\_String(er));

}

}

Void DBManagement::editing\_btn\_Click(Object^ sender, EventArgs^ e) {

Record new\_record;

try {

new\_record.key = get\_input(key\_to\_edit);

}

catch (const char\* er) {

MessageBox::Show(char\_string\_into\_String(er));

return;

}

strcpy(new\_record.data, String\_to\_string(new\_data->Text).c\_str());

try {

tree->edit(new\_record.key, new\_record.data);

key\_to\_edit->Text = "";

new\_data->Text = "";

display();

}

catch (const char\* er) {

MessageBox::Show(char\_string\_into\_String(er));

}

}

Void DBManagement::find\_btn\_Click(Object^ sender, EventArgs^ e) {

int key;

try {

key = get\_input(key\_to\_find);

}

catch (const char\* er) {

MessageBox::Show(char\_string\_into\_String(er));

return;

}

try {

int amount\_of\_conmarisons = 0;

Record rec = tree->search(key, amount\_of\_conmarisons);

String^ message = "Key: " + Convert::ToString(key) + ", Data: " + char\_string\_into\_String(rec.data) + ", Comparisons: " + Convert::ToString(amount\_of\_conmarisons);

MessageBox::Show(message);

key\_to\_find->Text = "";

}

catch (const char\* er) {

MessageBox::Show(char\_string\_into\_String(er));

}

}

Void DBManagement::records\_addition\_btn\_Click(Object^ sender, EventArgs^ e) {

srand(time(nullptr));

int amount\_of\_records = 1000;

int data\_length = data\_to\_insert->MaxLength;

vector<int> keys;

for (int i = 1; i <= amount\_of\_records; ++i)

keys.push\_back(i);

int i = 0;

while(i < amount\_of\_records) {

int index = generate\_number\_in\_range(0, keys.size()-1);

int key = keys.at(index);

keys.erase(keys.begin() + index);

Record rec;

rec.key = key;

strcpy(rec.data, generate\_string(data\_length));

tree->insert(rec);

++i;

}

display();

}

void DBManagement::display() {

vector<Record> records;

tree->traverse(records);

if (!records.empty()) {

enable\_edit\_delete\_search();

disable\_records\_addition();

}

int amount\_of\_iterations = records.size() - data\_table->Rows->Count;

for(int i = 0; i < amount\_of\_iterations; ++i)

data\_table->Rows->Add();

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

data\_table->Rows[i]->Cells[0]->Value = records.at(i).key;

data\_table->Rows[i]->Cells[1]->Value = char\_string\_into\_String(records.at(i).data);

}

records.clear();

}

void DBManagement::remove\_row(int key) {

for (int i = 0; i < data\_table->Rows->Count; ++i) {

int row\_key = Convert::ToInt16(data\_table->Rows[i]->Cells[0]->Value);

if (row\_key == key) {

data\_table->Rows->RemoveAt(i);

break;

}

}

if (data\_table->Rows->Count == 0) {

disable\_edit\_delete\_search();

enable\_records\_addition();

}

}

void DBManagement::edit\_row(int key, char\* data) {

for (int i = 0; i < data\_table->Rows->Count; ++i) {

int row\_key = Convert::ToInt16(data\_table->Rows[i]->Cells[0]->Value);

if (row\_key == key) {

data\_table->Rows[i]->Cells[1]->Value = char\_string\_into\_String(data);

break;

}

}

}

void DBManagement::disable\_edit\_delete\_search() {

key\_to\_edit->Enabled = false;

key\_to\_delete->Enabled = false;

key\_to\_find->Enabled = false;

new\_data->Enabled = false;

deletion\_btn->Enabled = false;

find\_btn->Enabled = false;

editing\_btn->Enabled = false;

}

void DBManagement::enable\_edit\_delete\_search() {

key\_to\_edit->Enabled = true;

key\_to\_delete->Enabled = true;

key\_to\_find->Enabled = true;

new\_data->Enabled = true;

deletion\_btn->Enabled = true;

find\_btn->Enabled = true;

editing\_btn->Enabled = true;

}

void DBManagement::disable\_records\_addition() {

records\_addition\_btn->Enabled = false;

}

void DBManagement::enable\_records\_addition() {

records\_addition\_btn->Enabled = true;

}

DBManagement::~DBManagement() {

if (components)

{

delete components;

}

ofstream file\_to\_save(\*file\_path\_to\_save, ios::binary);

if(!file\_to\_save)

MessageBox::Show("Can't save tree");

else

tree->save(file\_to\_save);

file\_to\_save.close();

delete tree;

delete file\_path\_to\_save;

}

**random\_generators.h**

#pragma once

#include <ctime>

#include <stdlib.h>

#include <string>

using namespace std;

int generate\_number\_in\_range(int low, int top);

char\* generate\_string(int length);

**random\_generators.cpp**

#include "random\_generators.h"

int generate\_number\_in\_range(int low, int top) {

return low + rand() % (top - low + 1);

}

char\* generate\_string(int length) {

string symbols = "abcdefghijklmnopqrstuvwxyz";

char\* result = new char[length+1];

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

result[i] = symbols[generate\_number\_in\_range(0, symbols.length())];

result[length] = '\0';

return result;

}

**main.cpp**

#include "DBForm.h"

using namespace System;

using namespace Windows::Forms;

using namespace Lab3;

[STAThreadAttribute]

int main() {

Application::SetCompatibleTextRenderingDefault(false);

Application::EnableVisualStyles();

DBForm^ welcome\_form = gcnew DBForm;

Application::Run(welcome\_form);

return 0;

}

**BTree.h**

#pragma once

#include <iostream>

#include <vector>

#include <string>

#include <fstream>

#include <cstring>

using namespace std;

#define DATA\_SIZE 35

struct Record {

int key;

char data[DATA\_SIZE];

};

class BTree {

class Node {

vector<Record> records;

int minimum\_degree;

vector<Node\*> children;

int amount\_of\_records;

bool is\_leaf;

public:

Node(int minimum\_degree, bool is\_leaf);

void traverse(vector<Record>& destination) const;

Record search(int key, int& amount\_of\_comparisons);

void insert\_in\_non\_full(Record record);

void split\_child(int index);

int find\_index\_of\_first\_greater\_or\_equal\_key(int key) const;

void remove(int key);

void remove\_from\_leaf(int key\_index);

void remove\_from\_non\_leaf(int key\_index);

void replace\_record\_by\_child\_record(int record\_index\_to\_replace, int child\_index\_to\_take\_record, Record record);

Record get\_predecessor(int record\_index);

Record get\_successor(int record\_index);

void fill\_child(int child\_index);

void borrow\_record\_from\_previous\_child(int child\_index);

void borrow\_record\_from\_next\_child(int child\_index);

void merge\_child(int child\_index);

bool is\_full() const;

bool contains\_minimal\_allowed\_amount\_of\_records() const;

void save(ofstream& destination);

void edit(int key, char new\_data[DATA\_SIZE]);

friend class BTree;

};

Node\* root;

int minimum\_degree;

BTree::Node\* read\_node(ifstream& source);

public:

BTree(int minimum\_degree);

void traverse(vector<Record>& destination) const;

Record search(int key, int& amount\_of\_comparisons);

void insert(Record record);

void edit(int key, char new\_data[DATA\_SIZE]);

void remove(int key);

void save(ofstream& destination);

void open(ifstream& source);

bool is\_empty() const;

};

**BTree.cpp**

#include "B\_Tree.h"

BTree::BTree(int minimum\_degree) : minimum\_degree(minimum\_degree) {

root = nullptr;

}

bool BTree::is\_empty() const {

return root == nullptr;

}

void BTree::traverse(vector<Record>& destination) const {

if (root)

root->traverse(destination);

}

Record BTree::search(int key, int& amount\_of\_comparisons) {

if (!root)

throw "The tree is empty";

return root->search(key, amount\_of\_comparisons);

}

void BTree::edit(int key, char\* new\_data) {

if (!root)

throw "The tree is empty";

root->edit(key, new\_data);

}

void BTree::insert(Record record) {

if (!root) {

root = new Node(minimum\_degree, true);

root->records.push\_back(record);

root->amount\_of\_records = 1;

return;

}

bool contains\_record\_with\_such\_key = true;

try {

int not\_used = 0;

search(record.key, not\_used);

}

catch (const char\* er) {

contains\_record\_with\_such\_key = false;

}

if (contains\_record\_with\_such\_key)

throw "There is a record with such key";

if (root->is\_full()) {

Node\* new\_root = new Node(minimum\_degree, false);

new\_root->children.push\_back(root);

new\_root->split\_child(0);

int i = 0;

if (new\_root->records.at(0).key < record.key)

++i;

new\_root->children.at(i)->insert\_in\_non\_full(record);

root = new\_root;

}

else

root->insert\_in\_non\_full(record);

}

void BTree::remove(int key) {

if (!root)

throw "This tree is empty";

root->remove(key);

if (root->amount\_of\_records == 0) {

BTree::Node\* temp = root;

if (root->is\_leaf)

root = nullptr;

else

root = root->children[0];

delete temp;

}

}

BTree::Node::Node(int minimum\_degree, bool is\_leaf) : minimum\_degree(minimum\_degree), is\_leaf(is\_leaf) {

amount\_of\_records = 0;

}

void BTree::Node::traverse(vector<Record>& destination) const {

int i;

for (i = 0; i < amount\_of\_records; ++i) {

if (!is\_leaf)

children.at(i)->traverse(destination);

destination.push\_back(records.at(i));

}

if (!is\_leaf)

children.at(i)->traverse(destination);

}

Record BTree::Node::search(int key, int& amount\_of\_comparisons) {

int left = 0;

int right = records.size() - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (records.at(mid).key == key) {

++amount\_of\_comparisons;

return records.at(mid);

}

if (records.at(mid).key < key)

left = mid + 1;

else

right = mid - 1;

++amount\_of\_comparisons;

}

if (is\_leaf)

throw "There is no such element";

return children.at(left)->search(key, amount\_of\_comparisons);

}

void BTree::Node::edit(int key, char\* new\_data) {

int i = 0;

while (i < amount\_of\_records && key > records.at(i).key)

++i;

if (i < amount\_of\_records && records.at(i).key == key) {

strcpy\_s(records.at(i).data, new\_data);

return;

}

if (is\_leaf)

throw "There is no element with such key";

children.at(i)->edit(key, new\_data);

}

void BTree::Node::insert\_in\_non\_full(Record record) {

int i = amount\_of\_records - 1;

if (is\_leaf) {

while (i >= 0 && records.at(i).key > record.key)

--i;

records.insert(records.begin() + (i + 1), record);

++amount\_of\_records;

return;

}

while (i >= 0 && records.at(i).key > record.key)

--i;

if (children.at(i + 1)->is\_full()) {

split\_child(i + 1);

if (records.at(i + 1).key < record.key)

++i;

}

children.at(i + 1)->insert\_in\_non\_full(record);

}

void BTree::Node::split\_child(int index) {

BTree::Node\* child\_to\_split = children.at(index);

BTree::Node\* additional\_node = new BTree::Node(child\_to\_split->minimum\_degree, child\_to\_split->is\_leaf);

additional\_node->amount\_of\_records = minimum\_degree - 1;

for (int i = 0; i < minimum\_degree - 1; ++i) {

additional\_node->records.push\_back(child\_to\_split->records.at(minimum\_degree));

child\_to\_split->records.erase(child\_to\_split->records.begin() + minimum\_degree);

}

if (!child\_to\_split->is\_leaf) {

for (int i = 0; i < minimum\_degree; ++i) {

additional\_node->children.push\_back(child\_to\_split->children.at(minimum\_degree));

child\_to\_split->children.erase(child\_to\_split->children.begin() + minimum\_degree);

}

}

child\_to\_split->amount\_of\_records = minimum\_degree - 1;

children.insert(children.begin() + index + 1, additional\_node);

records.insert(records.begin() + index, child\_to\_split->records.at(minimum\_degree - 1));

child\_to\_split->records.erase(child\_to\_split->records.begin() + minimum\_degree - 1);

++amount\_of\_records;

}

bool BTree::Node::is\_full() const {

return amount\_of\_records == 2 \* minimum\_degree - 1;

}

bool BTree::Node::contains\_minimal\_allowed\_amount\_of\_records() const {

return amount\_of\_records == (minimum\_degree - 1);

}

int BTree::Node::find\_index\_of\_first\_greater\_or\_equal\_key(int key) const {

auto it = find\_if(records.begin(), records.end(), [key](Record element) {return element.key >= key; });

return (it - records.begin());

}

void BTree::Node::remove(int key) {

int index = find\_index\_of\_first\_greater\_or\_equal\_key(key);

if (index < amount\_of\_records && records.at(index).key == key) {

if (is\_leaf)

remove\_from\_leaf(index);

else

remove\_from\_non\_leaf(index);

return;

}

if (is\_leaf)

throw "The record with key " + to\_string(key) + " isn't in the tree";

bool is\_key\_present\_in\_last\_child\_subtree = (index == amount\_of\_records);

if (children[index]->amount\_of\_records < minimum\_degree)

fill\_child(index);

if (is\_key\_present\_in\_last\_child\_subtree && index > amount\_of\_records)

children.at(index - 1)->remove(key);

else

children.at(index)->remove(key);

}

void BTree::Node::remove\_from\_leaf(int key\_index) {

records.erase(records.begin() + key\_index);

--amount\_of\_records;

}

void BTree::Node::remove\_from\_non\_leaf(int key\_index) {

int key = records.at(key\_index).key;

if (!children.at(key\_index)->contains\_minimal\_allowed\_amount\_of\_records())

replace\_record\_by\_child\_record(key\_index, key\_index, get\_predecessor(key\_index));

else if (!children.at(key\_index + 1)->contains\_minimal\_allowed\_amount\_of\_records())

replace\_record\_by\_child\_record(key\_index, key\_index + 1, get\_successor(key\_index));

else {

merge\_child(key\_index);

children.at(key\_index)->remove(key);

}

}

void BTree::Node::replace\_record\_by\_child\_record(int record\_index\_to\_replace, int child\_index\_to\_take\_record, Record record) {

records.at(record\_index\_to\_replace) = record;

children.at(child\_index\_to\_take\_record)->remove(record.key);

}

Record BTree::Node::get\_predecessor(int key\_index) {

BTree::Node\* curr = children.at(key\_index);

while (!curr->is\_leaf)

curr = curr->children.at(curr->amount\_of\_records);

return curr->records.at(curr->amount\_of\_records - 1);

}

Record BTree::Node::get\_successor(int key\_index) {

BTree::Node\* curr = children.at(key\_index + 1);

while (!curr->is\_leaf)

curr = curr->children.at(0);

return curr->records.at(0);

}

void BTree::Node::fill\_child(int child\_index) {

if (child\_index != 0 && !children.at(child\_index - 1)->contains\_minimal\_allowed\_amount\_of\_records())

borrow\_record\_from\_previous\_child(child\_index);

else if (child\_index != amount\_of\_records && !children.at(child\_index + 1)->contains\_minimal\_allowed\_amount\_of\_records())

borrow\_record\_from\_next\_child(child\_index);

else {

if (child\_index != amount\_of\_records)

merge\_child(child\_index);

else

merge\_child(child\_index - 1);

}

}

void BTree::Node::borrow\_record\_from\_previous\_child(int child\_index) {

BTree::Node\* child\_which\_borrows = children.at(child\_index);

BTree::Node\* sibling\_child\_from\_which\_borrow = children.at(child\_index - 1);

child\_which\_borrows->records.insert(child\_which\_borrows->records.begin(), records.at(child\_index - 1));

if (!child\_which\_borrows->is\_leaf) {

child\_which\_borrows->children.insert(child\_which\_borrows->children.begin(), sibling\_child\_from\_which\_borrow->children.at(sibling\_child\_from\_which\_borrow->amount\_of\_records));

sibling\_child\_from\_which\_borrow->children.erase(sibling\_child\_from\_which\_borrow->children.begin() + sibling\_child\_from\_which\_borrow->amount\_of\_records);

}

records.at(child\_index - 1) = sibling\_child\_from\_which\_borrow->records.at(sibling\_child\_from\_which\_borrow->amount\_of\_records - 1);

sibling\_child\_from\_which\_borrow->records.erase(sibling\_child\_from\_which\_borrow->records.begin() + sibling\_child\_from\_which\_borrow->amount\_of\_records - 1);

++child\_which\_borrows->amount\_of\_records;

--sibling\_child\_from\_which\_borrow->amount\_of\_records;

}

void BTree::Node::borrow\_record\_from\_next\_child(int child\_index) {

BTree::Node\* child\_which\_borrows = children.at(child\_index);

BTree::Node\* sibling\_child\_from\_which\_borrow = children.at(child\_index + 1);

child\_which\_borrows->records.insert(child\_which\_borrows->records.begin() + child\_which\_borrows->amount\_of\_records, records.at(child\_index));

if (!child\_which\_borrows->is\_leaf) {

child\_which\_borrows->children.insert(child\_which\_borrows->children.begin() + child\_which\_borrows->amount\_of\_records + 1, sibling\_child\_from\_which\_borrow->children.at(0));

sibling\_child\_from\_which\_borrow->children.erase(sibling\_child\_from\_which\_borrow->children.begin());

}

records.at(child\_index) = sibling\_child\_from\_which\_borrow->records.at(0);

sibling\_child\_from\_which\_borrow->records.erase(sibling\_child\_from\_which\_borrow->records.begin());

++child\_which\_borrows->amount\_of\_records;

--sibling\_child\_from\_which\_borrow->amount\_of\_records;

}

void BTree::Node::merge\_child(int child\_index) {

BTree::Node\* child = children.at(child\_index);

BTree::Node\* sibling\_which\_merges = children.at(child\_index + 1);

child->records.push\_back(records.at(child\_index));

records.erase(records.begin() + child\_index);

child->records.insert(child->records.end(), sibling\_which\_merges->records.begin(), sibling\_which\_merges->records.end());

sibling\_which\_merges->records.clear();

if (!child->is\_leaf) {

child->children.insert(child->children.end(), sibling\_which\_merges->children.begin(), sibling\_which\_merges->children.end());

sibling\_which\_merges->children.clear();

}

children.erase(children.begin() + child\_index + 1);

--amount\_of\_records;

child->amount\_of\_records += sibling\_which\_merges->amount\_of\_records + 1;

delete sibling\_which\_merges;

}

void BTree::save(std::ofstream& destination) {

if (!root)

return;

root->save(destination);

}

void BTree::Node::save(ofstream& destination) {

destination.write(reinterpret\_cast<const char\*>(&is\_leaf), sizeof(is\_leaf));

destination.write(reinterpret\_cast<const char\*>(&amount\_of\_records), sizeof(amount\_of\_records));

for (int i = 0; i < amount\_of\_records; ++i)

destination.write(reinterpret\_cast<const char\*>(&records.at(i)), sizeof(Record));

for (const auto& child : children)

child->save(destination);

}

void BTree::open(std::ifstream& source) {

root = read\_node(source);

}

BTree::Node\* BTree::read\_node(ifstream& source) {

bool is\_node\_leaf;

source.read(reinterpret\_cast<char\*>(&is\_node\_leaf), sizeof(is\_node\_leaf));

BTree::Node\* node = new BTree::Node(minimum\_degree, is\_node\_leaf);

source.read(reinterpret\_cast<char\*>(&node->amount\_of\_records), sizeof(node->amount\_of\_records));

for (int i = 0; i < node->amount\_of\_records; ++i) {

Record rec;

source.read(reinterpret\_cast<char\*>(&rec), sizeof(Record));

node->records.push\_back(rec);

}

if (!is\_node\_leaf) {

for (int i = 0; i < node->amount\_of\_records + 1; ++i)

node->children.push\_back(read\_node(source));

}

return node;

}

**validation\_functions.h**

#pragma once

#include <string>

void validate\_file\_path(std::string& path, std::string extension);

**validation\_functions.cpp**

#include "validation\_functions.h"

void validate\_file\_path(std::string& path, std::string extension) {

std::string file\_extension = path.substr(path.length() - extension.length(), extension.length());

if (file\_extension != extension)

path += extension;

}

**Lab3\_tests.cpp**

#include "pch.h"

#include "CppUnitTest.h"

#include "../Lab3\_code/B\_Tree.h"

#include "../Lab3\_code/validation\_functions.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace Lab3tests

{

TEST\_CLASS(BTree\_tests)

{

BTree create\_BTree(int amount\_of\_records, int minimal\_degree) {

BTree tree(minimal\_degree);

for (int i = 0; i < amount\_of\_records; ++i) {

Record record;

record.key = i + 1;

strcpy\_s(record.data, "data");

strcat\_s(record.data, (to\_string(i + 1)).c\_str());

tree.insert(record);

}

return tree;

}

public:

TEST\_METHOD(search\_present\_one\_node) {

BTree tree = create\_BTree(6, 3);

int key\_to\_search = 1;

int not\_used = 0;

Record result = tree.search(key\_to\_search, not\_used);

Assert::AreEqual(result.key, key\_to\_search);

char expected\_data[250] = "data";

strcat\_s(expected\_data, (to\_string(key\_to\_search)).c\_str());

Assert::AreEqual(result.data, expected\_data);

}

TEST\_METHOD(search\_present\_two\_nodes\_first) {

BTree tree = create\_BTree(6, 3);

int key\_to\_search = 1;

int not\_used = 0;

Record result = tree.search(key\_to\_search, not\_used);

Assert::AreEqual(result.key, key\_to\_search);

char expected\_data[250] = "data";

strcat\_s(expected\_data, (to\_string(key\_to\_search)).c\_str());

Assert::AreEqual(result.data, expected\_data);

}

TEST\_METHOD(search\_present\_two\_nodes\_median) {

BTree tree = create\_BTree(6, 3);

int key\_to\_search = 3;

int not\_used = 0;

Record result = tree.search(key\_to\_search, not\_used);

Assert::AreEqual(result.key, key\_to\_search);

char expected\_data[250] = "data";

strcat\_s(expected\_data, (to\_string(key\_to\_search)).c\_str());

Assert::AreEqual(result.data, expected\_data);

}

TEST\_METHOD(search\_present\_two\_nodes\_last) {

BTree tree = create\_BTree(6, 3);

int key\_to\_search = 6;

int not\_used = 0;

Record result = tree.search(key\_to\_search, not\_used);

Assert::AreEqual(result.key, key\_to\_search);

char expected\_data[250] = "data";

strcat\_s(expected\_data, (to\_string(key\_to\_search)).c\_str());

Assert::AreEqual(result.data, expected\_data);

}

TEST\_METHOD(search\_absent) {

BTree tree = create\_BTree(6, 3);

int key\_to\_search = 7;

int not\_used = 0;

auto func = [&tree, key\_to\_search, &not\_used] { tree.search(key\_to\_search, not\_used); };

Assert::ExpectException<const char\*>(func);

}

TEST\_METHOD(insert\_already\_present) {

BTree tree = create\_BTree(6, 3);

int key\_to\_insert = 3;

Record record = { key\_to\_insert, "test data" };

auto func = [&tree, &record] { tree.insert(record); };

Assert::ExpectException<const char\*>(func);

}

TEST\_METHOD(delete\_absent) {

BTree tree = create\_BTree(6, 3);

int key\_to\_delete = 7;

auto func = [&tree, key\_to\_delete] { tree.remove(key\_to\_delete); };

Assert::ExpectException<string>(func);

}

TEST\_METHOD(delete\_present\_first) {

BTree tree = create\_BTree(6, 3);

int key\_to\_delete = 1;

tree.remove(key\_to\_delete);

int not\_used = 0;

auto func = [&tree, key\_to\_delete, &not\_used] { tree.search(key\_to\_delete, not\_used); };

Assert::ExpectException<const char\*>(func);

}

TEST\_METHOD(delete\_present\_median) {

BTree tree = create\_BTree(6, 3);

int key\_to\_delete = 3;

tree.remove(key\_to\_delete);

int not\_used = 0;

auto func = [&tree, key\_to\_delete, &not\_used] { tree.search(key\_to\_delete, not\_used); };

Assert::ExpectException<const char\*>(func);

}

TEST\_METHOD(edit\_present\_first) {

BTree tree = create\_BTree(6, 3);

int key\_to\_edit = 1;

char\* new\_data = "new data";

tree.edit(key\_to\_edit, new\_data);

int not\_used = 0;

Record result = tree.search(key\_to\_edit, not\_used);

Assert::AreEqual(result.key, key\_to\_edit);

Assert::AreEqual(result.data, new\_data);

}

TEST\_METHOD(edit\_present\_median) {

BTree tree = create\_BTree(6, 3);

int key\_to\_edit = 3;

char\* new\_data = "new data";

tree.edit(key\_to\_edit, new\_data);

int not\_used = 0;

Record result = tree.search(key\_to\_edit, not\_used);

Assert::AreEqual(result.key, key\_to\_edit);

Assert::AreEqual(result.data, new\_data);

}

TEST\_METHOD(edit\_present\_last) {

BTree tree = create\_BTree(6, 3);

int key\_to\_edit = 6;

char\* new\_data = "new data";

tree.edit(key\_to\_edit, new\_data);

int not\_used = 0;

Record result = tree.search(key\_to\_edit, not\_used);

Assert::AreEqual(result.key, key\_to\_edit);

Assert::AreEqual(result.data, new\_data);

}

TEST\_METHOD(edit\_absent) {

BTree tree = create\_BTree(6, 3);

int key\_to\_edit = 7;

char\* new\_data = "new data";

auto func = [&tree, key\_to\_edit, new\_data] { tree.edit(key\_to\_edit, new\_data); };

Assert::ExpectException<const char\*>(func);

}

TEST\_METHOD(is\_empty\_empty) {

BTree tree(3);

Assert::IsTrue(tree.is\_empty());

}

TEST\_METHOD(is\_empty\_non\_empty) {

BTree tree = create\_BTree(6, 3);

Assert::IsFalse(tree.is\_empty());

}

};

TEST\_CLASS(validation\_functions\_tests) {

TEST\_METHOD(validate\_file\_path\_correct) {

string initial\_path = "test.mdb";

string expected\_result = "test.mdb";

string extension = ".mdb";

validate\_file\_path(initial\_path, extension);

Assert::AreEqual(initial\_path, expected\_result);

}

TEST\_METHOD(validate\_file\_path\_not\_correct1) {

string initial\_path = ".exe";

string expected\_result = ".exe.mdb";

string extension = ".mdb";

validate\_file\_path(initial\_path, extension);

Assert::AreEqual(initial\_path, expected\_result);

}

TEST\_METHOD(validate\_file\_path\_not\_correct2) {

string initial\_path = "test.exe";

string expected\_result = "test.exe.mdb";

string extension = ".mdb";

validate\_file\_path(initial\_path, extension);

Assert::AreEqual(initial\_path, expected\_result);

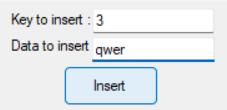
}

};

}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми для додавання і пошуку запису.



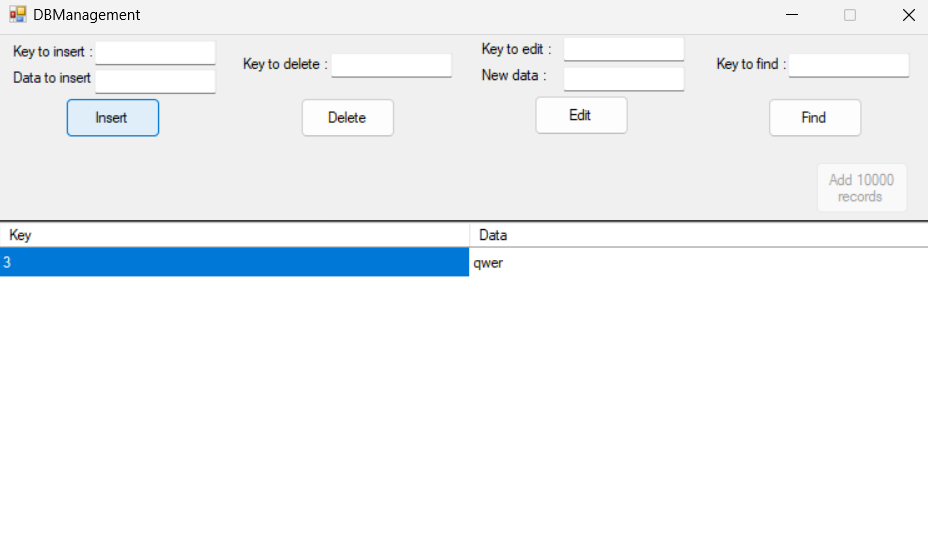


Рисунок 3.1 –Додавання запису

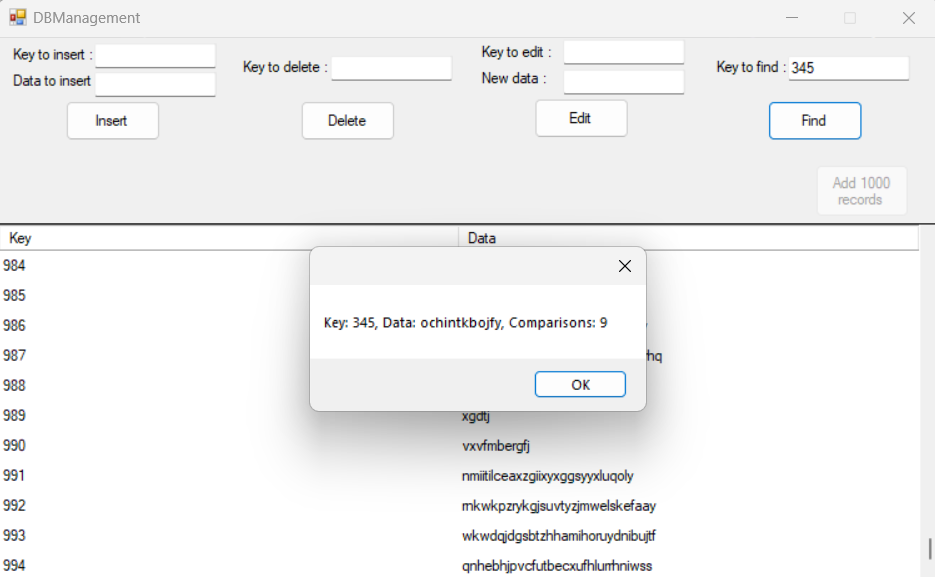


Рисунок 3.2 – Пошук запису

## Тестування алгоритму

### Часові характеристики оцінювання

В таблиці 3.1 наведено кількість порівнянь для 15 спроб пошуку запису по ключу.

Таблиця 3.1 – Число порівнянь при спробі пошуку запису по ключу

|  |  |
| --- | --- |
| Номер спроби пошуку | Число порівнянь |
| 1 | 9 |
| 2 | 10 |
| 3 | 9 |
| 4 | 11 |
| 5 | 10 |
| 6 | 6 |
| 7 | 10 |
| 8 | 9 |
| 9 | 8 |
| 10 | 10 |
| 11 | 9 |
| 12 | 10 |
| 13 | 10 |
| 14 | 9 |
| 15 | 9 |

Висновок

В рамках лабораторної роботи я реалізував структуру даних В-Дерево та розробив графічний інтерфейс на мові програмування С++ для взаємодії з нею. Основні алгоритми я описав за допомогою псевдокоду. Провів випробування цієї структури для пошуку запису, коли всього в дереві є 1000 записів. Результати випробування занесено до таблиці.

Критерії оцінювання

За умови здачі лабораторної роботи до 26.11.2023 включно максимальний бал дорівнює – 5. Після 26.11.2023 максимальний бал дорівнює – 4,5.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 10%;
* аналіз часової складності – 5%;
* програмна реалізація алгоритму – 50%;
* робота з гіт – 20%
* тестування алгоритму – 10%;
* висновок – 5%.

+1 додатковий бал можна отримати за реалізацію графічного відображення структури ключів.

+1 додатковий бал можна отримати за виконання та захист роботи до 19.11.2023.