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) ________Присяжний А. О.________ ________
(шифр, прізвище, ім'я, по батькові)

Перевірив ________*Ахаладзе І. Е..*________ ________
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

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

2 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	В-дерево t=10, бінарний пошук
8	В-дерево t=25, бінарний пошук
9	В-дерево t=50, бінарний пошук
10	В-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	В-дерево t=10, однорідний бінарний пошук
18	В-дерево t=25, однорідний бінарний пошук
19	В-дерево t=50, однорідний бінарний пошук
20	В-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	В-дерево t=10, метод Шарра
28	В-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

3 ВИКОНАННЯ

3.1 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);

}
```

3.2 Часова складність пошуку

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

3.3 Програмна реалізація

3.3.1 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::AutoS
ize;
                    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";
```

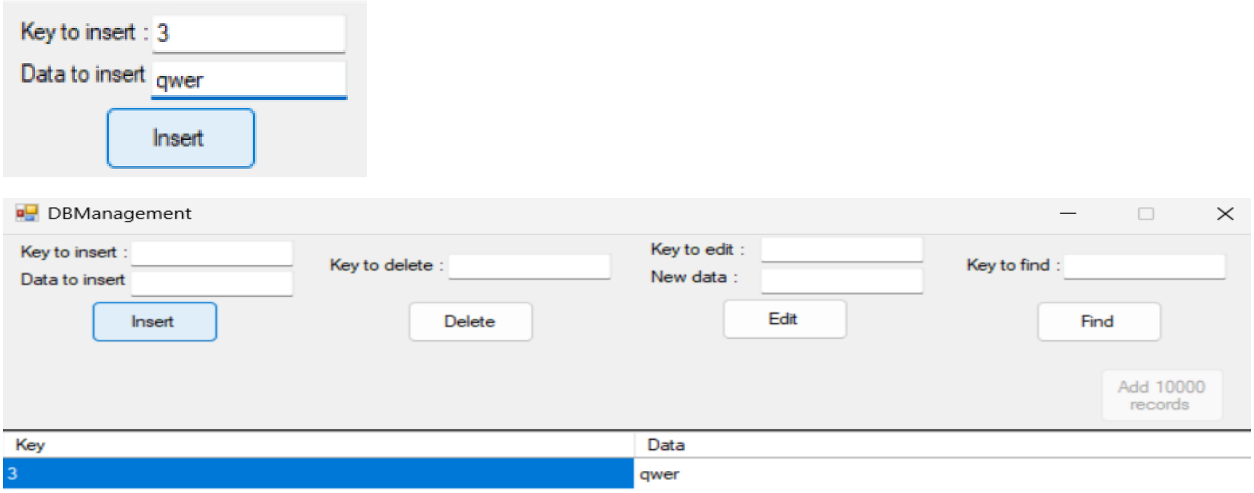
```
				strcat_s(expected_data,
(to_string(key_to_search)).c_str());
				Assert::AreEqual(result.data, expected_data);
			}
			TEST_METHOD(search_present_two_nodes_median) {
				BTree tree = create_BTree(6, 3);
				int key_to_search = 3;

				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);
            }
        };
    }
```

3.3.2 Приклади роботи

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

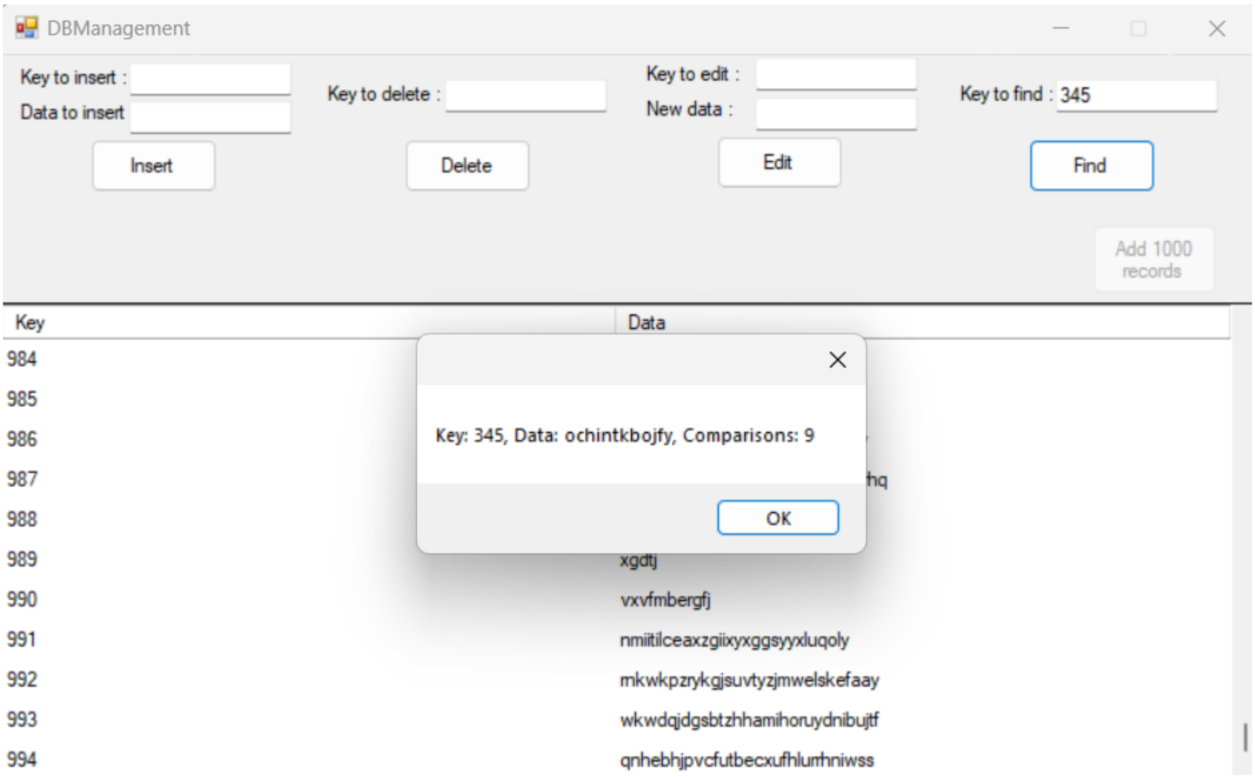


Key to insert : 3

Data to insert qwer

Key	Data
3	qwer

Рисунок 3.1 –Додавання запису



Key to find : 345

Key: 345, Data: ochintkbojfy, Comparisons: 9

Key	Data
984	
985	
986	
987	hq
988	
989	xgdtj
990	vxvfmbergfj
991	nmiitilceaxzgiixyxggsyyxluqoly
992	mkwkpzrykgjsuvtyzjmwelskefaay
993	wkwdqjdgsbtzhhamihoruydnibujtf
994	qnhebhjpvcfutbecxufhlurhniwss

Рисунок 3.2 – Пошук запису

3.4 Тестування алгоритму

3.4.1 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 B-Дерево та розробив графічний інтерфейс на мові програмування C++ 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.