**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)

*ІП-14 Громов Владислав*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2022

Зміст

[**1**](#_gjdgxs) **Мета лабораторної роботи 3**

[**2**](#_30j0zll) **Завдання 4**

[**3**](#_3znysh7) **Виконання 7**

[3.1](#_2et92p0) Псевдокод алгоритмів 7

[3.2](#_tyjcwt) Часова складність пошуку 7

[3.3](#_3dy6vkm) Програмна реалізація 7

[*3.3.1*](#_1t3h5sf) *Вихідний код 7*

[*3.3.2*](#_4d34og8) *Приклади роботи 7*

[3.4](#_2s8eyo1) Тестування алгоритму 8

[*3.4.1*](#_17dp8vu) *Часові характеристики оцінювання 8*

[**Висновок 9**](#_3rdcrjn)

[**Критерії оцінювання 10**](#_26in1rg)

# 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

## int Height(Node? node)

## {

## return node?.height ?? 0;

## }

## int BalanceFactor(Node? node)

## {

## return Height(node.nodeR) - Height(node.nodeL);

## }

## void FixHeight(Node? node)

## {

## var hL = Height(node.nodeL);

## var hR = Height(node.nodeR);

## node.height = (hL > hR ? hL : hR) + 1;

## }

## Node? RotateRight(Node? node)

## {

## var s = node.nodeL;

## node.nodeL = s.nodeR;

## s.nodeR = node;

## FixHeight(node);

## FixHeight(s);

## return s;

## }

## 

## Node? RotateLeft(Node? node)

## {

## var s = node.nodeR;

## node.nodeR = s.nodeL;

## s.nodeL = node;

## FixHeight(node);

## FixHeight(s);

## return s;

## }

## 

## Node? Balance(Node? node)

## {

## FixHeight(node);

## if(BalanceFactor(node) == 2)

## {

## if(BalanceFactor(node.nodeR) < 0)

## node.nodeR = RotateRight(node.nodeR);

## return RotateLeft(node);

## }

## if(BalanceFactor(node) == -2)

## {

## if(BalanceFactor(node.nodeL) > 0)

## node.nodeL = RotateLeft(node.nodeL);

## return RotateRight(node);

## }

## return node;

## }

## Node? Insert(Node? node, int key)

## {

## if (node == null) return new Node(key);

## if (key < node.key)

## node.nodeL = Insert(node.nodeL, key);

## else

## node.nodeR = Insert(node.nodeR, key);

## return Balance(node);

## }

## Node? FindMin(Node? node)

## {

## return node.nodeL != null ? FindMin(node.nodeL) : node;

## }

## Node? RemoveMin(Node? node)

## {

## if (node.nodeL == null)

## return node.nodeR;

## node.nodeL = RemoveMin(node.nodeL);

## return Balance(node);

## }

## Node? FillRandom(Node? x, int? num)

## {

## var rnd = new Random();

## for (int i = 0; i < num; i++)

## {

## x = Insert(x, rnd.Next(10000));

## }

## return x;

## }

## Node? Remove(Node? node, int key)

## {

## if (node == null) return null;

## if (key < node.key)

## node.nodeL = Remove(node.nodeL, key);

## else if (key > node.key)

## node.nodeR = Remove(node.nodeR, key);

## else

## {

## var l = node.nodeL;

## var r = node.nodeR;

## if (r == null) return l;

## var min = FindMin(r);

## min.nodeR = RemoveMin(r);

## min.nodeL = l;

## return Balance(min);

## }

## return Balance(node);

## }

## 

## string PrintTree(Node root, int space, int height, string dir) {

## if (root == null) {

## return null;

## }

## var ss = "";

## space += height;

## ss += PrintTree(root.nodeR, space - 2, height, "right");

## if (dir.Equals("main"))

## {

## ss += "\n";

## ss += (new string(' ', (space) \* 2 - 4));

## ss += (root.key);

## ss += "\n";

## }

## else

## {

## ss += "\n";

## ss += (new string(' ', (space - height) \* 2));

## if (dir.Equals("right")) ss += ("┌");

## if(dir.Equals("left")) ss += ("└");

## ss += (new string('-', height - 1));

## ss += (root.key);

## ss += "\n";

## }

## ss += PrintTree(root.nodeL, space - 2, height, "left");

## return ss;

## }

## 

## void SaveTree(Node node, StreamWriter sw)

## {

## if (node == null) sw.WriteLine("#");

## else

## {

## sw.WriteLine(node.key);

## sw.WriteLine(node.height);

## SaveTree(node.nodeL, sw);

## SaveTree(node.nodeR, sw);

## }

## }

## Node ReadTree(Node node, StreamReader sr)

## {

## var el = sr.ReadLine();

## Console.WriteLine(el);

## if (el.Equals("#") || el.Equals("")) return null;

## var height = sr.ReadLine();

## node = new Node(int.Parse(el), int.Parse(height));

## node.nodeL = ReadTree(node.nodeL, sr);

## node.nodeR = ReadTree(node.nodeR, sr);

## Console.WriteLine(node.key);

## return node;

## }

## 

## Часова складність пошуку

## O(log n)

## Програмна реалізація

### Вихідний код

**AVLTree.cs**

1. using System;  
   using System.IO;  
     
   namespace Lab3PA.Tree;  
     
   public static class AVLTree  
   {  
    public static int Height(Node? node)  
    {  
    return node?.height ?? 0;  
    }  
     
    public static int BalanceFactor(Node? node)  
    {  
    return Height(node.nodeR) - Height(node.nodeL);  
    }  
     
    public static void FixHeight(Node? node)  
    {  
    var hL = Height(node.nodeL);  
    var hR = Height(node.nodeR);  
    node.height = (hL > hR ? hL : hR) + 1;  
    }  
     
    public static Node? RotateRight(Node? node)  
    {  
    var s = node.nodeL;  
    node.nodeL = s.nodeR;  
    s.nodeR = node;  
    FixHeight(node);  
    FixHeight(s);  
    return s;  
    }  
      
    public static Node? RotateLeft(Node? node)  
    {  
    var s = node.nodeR;  
    node.nodeR = s.nodeL;  
    s.nodeL = node;  
    FixHeight(node);  
    FixHeight(s);  
    return s;  
    }  
      
    public static Node? Balance(Node? node)  
    {  
    FixHeight(node);  
    if(BalanceFactor(node) == 2)  
    {  
    if(BalanceFactor(node.nodeR) < 0)  
    node.nodeR = RotateRight(node.nodeR);  
    return RotateLeft(node);  
    }  
    if(BalanceFactor(node) == -2)  
    {  
    if(BalanceFactor(node.nodeL) > 0)  
    node.nodeL = RotateLeft(node.nodeL);  
    return RotateRight(node);  
    }  
    return node;  
    }  
     
    public static Node? Insert(Node? node, int key)  
    {  
    if (node == null) return new Node(key);  
    if (key < node.key)  
    node.nodeL = Insert(node.nodeL, key);  
    else  
    node.nodeR = Insert(node.nodeR, key);  
    return Balance(node);  
    }  
     
    public static Node? FindMin(Node? node)  
    {  
    return node.nodeL != null ? FindMin(node.nodeL) : node;  
    }  
     
    public static Node? RemoveMin(Node? node)  
    {  
    if (node.nodeL == null)  
    return node.nodeR;  
    node.nodeL = RemoveMin(node.nodeL);  
    return Balance(node);  
    }  
     
    public static Node? FillRandom(Node? x, int? num)  
    {  
    var rnd = new Random();  
    for (int i = 0; i < num; i++)  
    {  
    x = Insert(x, rnd.Next(10000));  
    }  
     
    return x;  
    }  
     
    public static Node? Remove(Node? node, int key)  
    {  
    if (node == null) return null;  
    if (key < node.key)  
    node.nodeL = Remove(node.nodeL, key);  
    else if (key > node.key)  
    node.nodeR = Remove(node.nodeR, key);  
    else  
    {  
    var l = node.nodeL;  
    var r = node.nodeR;  
    if (r == null) return l;  
    var min = FindMin(r);  
    min.nodeR = RemoveMin(r);  
    min.nodeL = l;  
    return Balance(min);  
    }  
     
    return Balance(node);  
    }  
      
    public static string PrintTree(Node root, int space, int height, string dir) {  
    if (root == null) {  
    return null;  
    }  
     
    var ss = "";  
    space += height;  
    ss += PrintTree(root.nodeR, space - 2, height, "right");  
    if (dir.Equals("main"))  
    {  
    ss += "\n";  
    ss += (new string(' ', (space) \* 2 - 4));  
    ss += (root.key);  
    ss += "\n";  
    }  
    else  
    {  
    ss += "\n";  
    ss += (new string(' ', (space - height) \* 2));  
    if (dir.Equals("right")) ss += ("┌");  
    if(dir.Equals("left")) ss += ("└");  
    ss += (new string('-', height - 1));  
    ss += (root.key);  
    ss += "\n";  
    }  
    ss += PrintTree(root.nodeL, space - 2, height, "left");  
    return ss;  
    }  
      
     
    public static void SaveTree(Node node, StreamWriter sw)  
    {  
    if (node == null) sw.WriteLine("#");  
    else  
    {  
    sw.WriteLine(node.key);  
    sw.WriteLine(node.height);  
    SaveTree(node.nodeL, sw);  
    SaveTree(node.nodeR, sw);  
    }  
    }  
     
    public static Node ReadTree(Node node, StreamReader sr)  
    {  
    var el = sr.ReadLine();  
    Console.WriteLine(el);  
    if (el.Equals("#") || el.Equals("")) return null;  
    var height = sr.ReadLine();  
    node = new Node(int.Parse(el), int.Parse(height));  
    node.nodeL = ReadTree(node.nodeL, sr);  
    node.nodeR = ReadTree(node.nodeR, sr);  
    Console.WriteLine(node.key);  
    return node;  
    }  
      
      
   }

### Приклади роботи

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

Рисунок 3.1 –Додавання запису

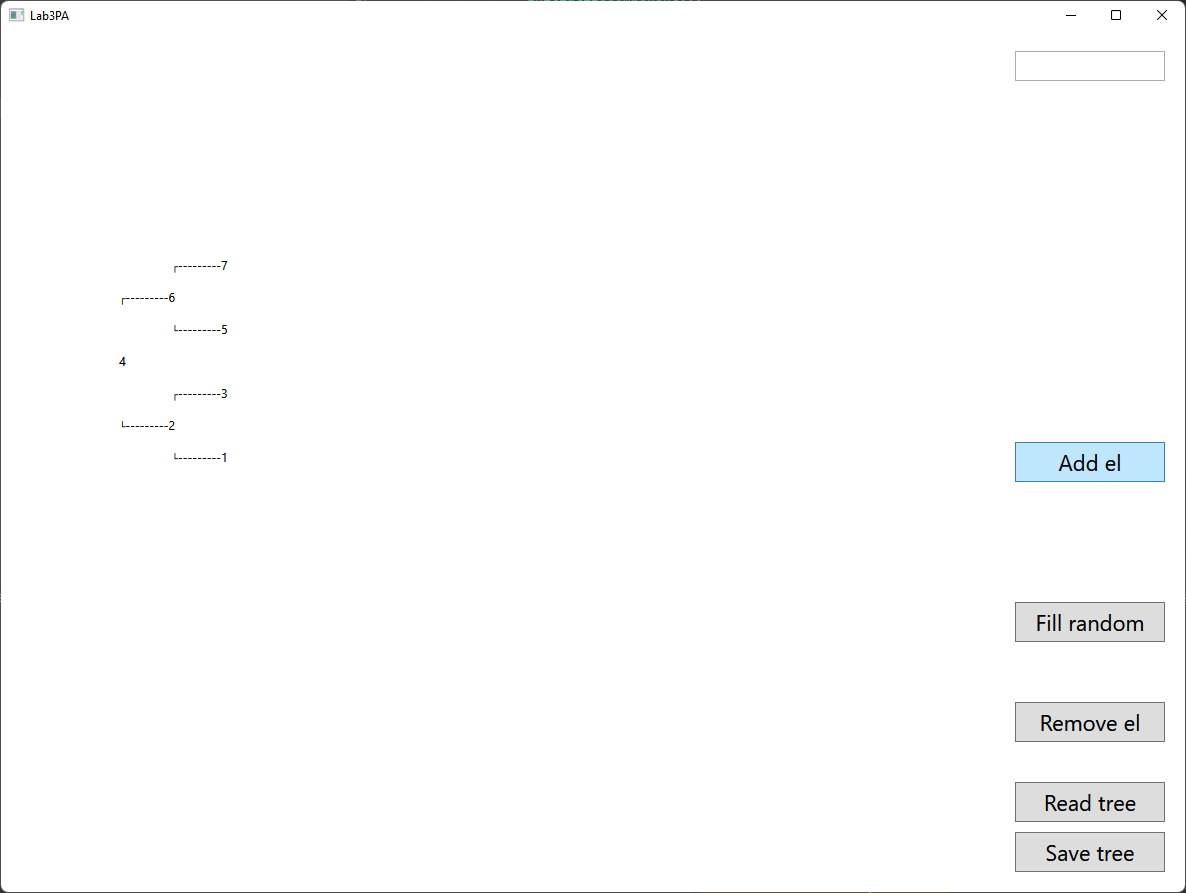
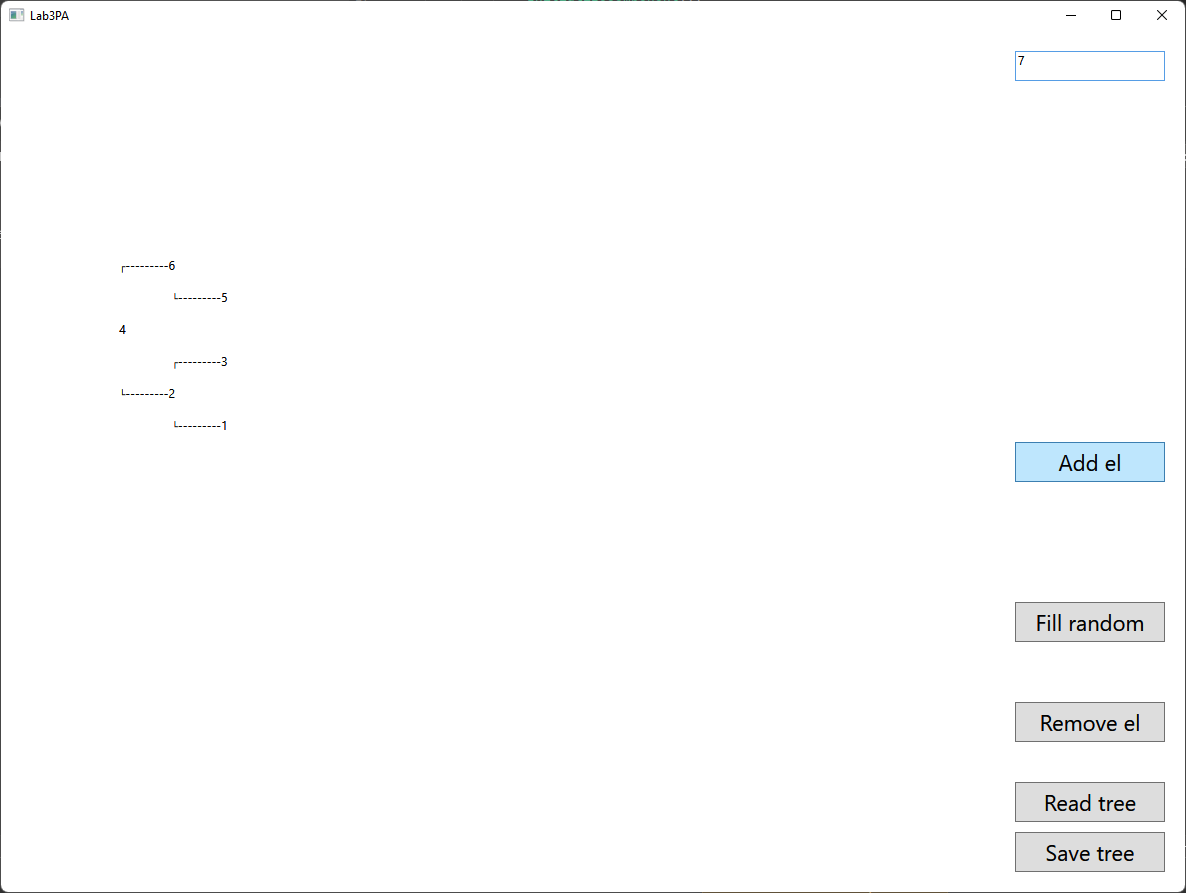


Рисунок 3.2 – Пошук запису

## 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 | 21 |
| 2 | 32 |
| 3 | 12 |
| 4 | 5 |
| 5 | 4 |
| 6 | 45 |
| 7 | 34 |
| 8 | 21 |
| 9 | 43 |
| 10 | 56 |
| 11 | 43 |
| 12 | 56 |
| 13 | 12 |
| 14 | 2 |
| 15 | 8 |

Висновок

У рамках цієї лабораторної роботи я навчився працювати з АВЛ-деревами, а також зрозумів принцип їхньої роботи. Також я навчився створювати десктопні додатки з графічної оболонкою.

Критерії оцінювання

За умови здачі лабораторної роботи до 13.11.2022 включно максимальний бал дорівнює – 5. Після 13.11.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 15%;
* аналіз часової складності – 5%;
* програмна реалізація алгоритму – 65%;
* тестування алгоритму – 10%;
* висновок – 5%.

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