**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 № 2 з дисципліни

«Проектування алгоритмів»

„ **Проектування структур даних**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІТ-04 Стрільчук Михайло Васильович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2021

Зміст

[1 Мета лабораторної роботи 3](#_Toc81070524)

[2 Завдання 4](#_Toc81070525)

[3 Виконання 7](#_Toc81070526)

[3.1 Псевдокод алгоритмів 7](#_Toc81070527)

[3.2 Часова складність пошуку 7](#_Toc81070528)

[3.3 Програмна реалізація 7](#_Toc81070529)

[3.3.1 Вихідний код 7](#_Toc81070530)

[3.3.2 Приклади роботи 7](#_Toc81070531)

[3.4 Тестування алгоритму 8](#_Toc81070532)

[3.4.1 Часові характеристики оцінювання 8](#_Toc81070533)

[Висновок 9](#_Toc81070534)

[Критерії оцінювання 10](#_Toc81070535)

# 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 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, бінарний пошук |
| 18 | B-дерево t=25, бінарний пошук |
| 19 | B-дерево t=50, бінарний пошук |
| 20 | B-дерево t=100, 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 |

# Виконання

## Псевдокод алгоритмів

**Структура вузла**

**struct** Node

**bool** leaf

**int** n

**int** key[]

**Node** c

**Структура дерева**

**struct** BTree

**int** t

**Node** root

**Вставка ключа в дерево**

**void** B-Tree-Insert(T: **BTree**, k: **int**):

r = T.root

**if** r.n == 2T.t - 1

s = Allocate-Node()

T.root = s

s.leaf = *false*

s.n = 0

s.c[1] = r

B-Tree-Split-Child(s, T.t, 1)

B-Tree-Insert-Nonfull(s, k, T.t)

**else**

B-Tree-Insert-Nonfull(r, k, T.t)

**void** B-Tree-Insert-Nonfull(x: **Node**, k: **int**, t: **int**):

i = x.n

**if** x.leaf

**while** i >= 1 **and** k < x.key[i]

x.key[i+1] = x.key[i]

i = i - 1

x.key[i+1] = k

x.n = x.n + 1

Disk-Write(x)

**else**

**while** i >= 1 **and** k < x.key[i]

i = i - 1

i = i + 1

Disk-Read(x.c[i])

**if** x.c[i].n == 2t - 1

B-Tree-Split-Child(x, t, i)

**if** k > x.key[i]

i = i + 1

B-Tree-Insert-Nonfull(x.c[i], k, t)

**Розбиття вузла**

**void** B-Tree-Split-Child(x: **Node**, t: **int**, i: **int**):

z = Allocate-Node()

y = x.c[i]

z.leaf = y.leaf

z.n = t - 1

**for** j = 1 **to** t - 1

z.key[j] = y.key[j+t]

**if** not y.leaf

**for** j = 1 **to** t

z.c[j] = y.c[j+t]

y.n = t - 1

**for** j = x.n + 1 **to** i + 1

x.c[j+1] = x.c[j]

x.c[i+1] = z

**for** j = x.n **to** i

x.key[j+1] = x.key[j]

x.key[i] = y.key[t]

x.n = x.n + 1

Disk-Write(y)

Disk-Write(z)

Disk-Write(x)

## Часова складність пошуку

Часова складність операцій пошуку, видалення, додавання, редагування

O(logn)

## Програмна реалізація

### Вихідний код

https://github.com/Mishanych/Algorithms/tree/main/laba2

using System;

using System.Diagnostics;

namespace Lab2

{

class Program

{

static void Main(string[] args)

{

BTree t = new BTree(100);

t.Insert(1, "v1");

t.Insert(3, "v3");

t.Insert(7, "v7");

t.Insert(10, "v10");

t.Insert(11, "v11");

t.Insert(13, "v13");

t.Insert(14, "v14");

t.Insert(15, "v15");

t.Insert(4, "v4");

t.Insert(5, "v5");

t.Insert(2, "v2");

t.Insert(12, "v12");

t.Insert(6, "v6");

Console.WriteLine("Traversal of tree constructed is");

t.Traverse();

Console.WriteLine();

t.Remove(6);

Console.WriteLine("Traversal of tree after removing 6");

t.Traverse();

Console.WriteLine();

t.Remove(13);

Console.WriteLine("Traversal of tree after removing 13");

t.Traverse();

Console.WriteLine();

t.Remove(7);

Console.WriteLine("Traversal of tree after removing 7");

t.Traverse();

Console.WriteLine();

t.Remove(4);

Console.WriteLine("Traversal of tree after removing 4");

t.Traverse();

Console.WriteLine();

t.Remove(2);

Console.WriteLine("Traversal of tree after removing 2");

t.Traverse();

Console.WriteLine();

t.Remove(16);

Console.WriteLine("Traversal of tree after removing 16");

t.Traverse();

Console.WriteLine();

int pkey = 15;

Console.WriteLine($"Search key {pkey}: value = {t.Search(pkey)}");

t.Change(15, "Hello world!");

Console.WriteLine("Traversal of tree after removing 16");

t.Traverse();

Console.WriteLine();

t.Save();

t.Load();

t.Traverse();

Console.WriteLine();

var rand = new Random();

Console.WriteLine("Added BTree (t=100). Search 10 random keys:");

for (int i = 0; i < 10000; i++)

{

t.Insert(i, "value\_" + i);

}

for (int i = 0; i < 10; i++)

{

int k = rand.Next(10000);

Console.WriteLine("Search key " + k + " Founded value: " + t.Search(k) + " Amount of passed nodes: " + BTreeNode.AmountOfPassedNodes);

}

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab2

{

public class BTree

{

private BTreeNode \_root;

private int \_minPower;

public BTree(int power)

{

this.\_root = null;

this.\_minPower = power;

}

public void Traverse()

{

if (\_root != null)

{

\_root.Traverse();

}

}

private BTreeNode SearchKey(int key)

{

BTreeNode.AmountOfPassedNodes = 0;

return \_root == null ? null : \_root.SearchNode(key);

}

public void Insert(int key, string value)

{

if (\_root == null)

{

\_root = new BTreeNode(\_minPower, true);

\_root.Keys[0].key = key;

\_root.Keys[0].value = value;

\_root.AmountOfKeys = 1;

}

else

{

if (\_root.AmountOfKeys == 2 \* \_minPower - 1)

{

BTreeNode node = new BTreeNode(\_minPower, false);

node.Сhildren[0] = \_root;

node.SplitChild(0, \_root);

int i = 0;

if (node.Keys[0].key < key)

{

i++;

}

node.Сhildren[i].InsertNotFull(key, value);

\_root = node;

}

else

{

\_root.InsertNotFull(key, value);

}

}

}

public void Remove(int key)

{

if (\_root == null)

{

Console.WriteLine("The tree is empty");

return;

}

\_root.Remove(key);

if (\_root.AmountOfKeys == 0)

{

if (\_root.isLeaf)

{

\_root = null;

}

else

{

\_root = \_root.Сhildren[0];

}

}

}

public string Search(int pkey)

{

BTreeNode.AmountOfPassedNodes = 0;

if (\_root != null)

{

return \_root.SearchValueByKey(pkey);

}

else

{

return "not found";

}

}

public void Change(int pkey, string val)

{

if (SearchKey(pkey) != null)

{

Remove(pkey);

Insert(pkey, val);

Console.WriteLine($"Changed key {pkey}: value = {Search(pkey)}");

}

else

{

Console.WriteLine($"key={pkey} not found! ");

}

}

public void Save()

{

if (\_root != null)

{

string fname = "data.txt";

using (StreamWriter sw = new StreamWriter(fname, false, System.Text.Encoding.Default))

{

sw.WriteLine(\_root.TreeToString());

}

}

}

public void Load()

{

string fname = "data.txt";

string s = "";

using (StreamReader sr = new StreamReader(fname))

{

s = sr.ReadToEnd();

string[] dataS = s.Split("$");

\_root = null;

for (int i = 0; i < dataS.Length - 1; i += 2) Insert(Int32.Parse(dataS[i]), dataS[i + 1]);

}

}

}

}

### Приклади роботи

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

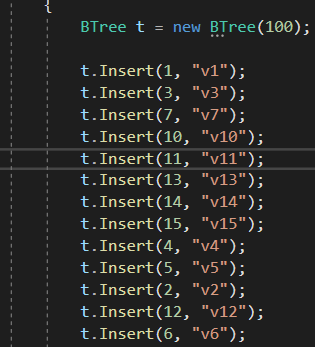


Рисунок 3.1 –Додавання запису

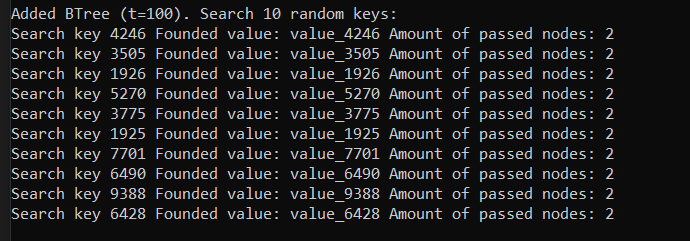


Рисунок 3.2 – Пошук запису

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 | 2 |
| 2 | 2 |
| 3 | 2 |
| 4 | 2 |
| 5 | 2 |
| 6 | 2 |
| 7 | 2 |
| 8 | 2 |
| 9 | 2 |
| 10 | 2 |
| 11 | 2 |
| 12 | 2 |
| 13 | 2 |
| 14 | 2 |
| 15 | 2 |

Висновок

В рамках лабораторної роботи я вивчив основні підходи проектування та обробки складних структур даних. Реалізував програмно B-tree (t=100) з бінарним пошуком з методами вставлення даних, їх редагування, пошук даних та видалення їх. Зробив можливість зберігати дерево в файлі і зчитувати його.

Критерії оцінювання

За умови здачі лабораторної роботи до 15.10.2021 включно максимальний бал дорівнює – 5. Після 15.10.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 20%;
* аналіз часової складності – 5%;
* програмна реалізація алгоритму – 60%;
* тестування алгоритму – 10%;
* висновок – 5%.

+1 додатковий бал можна отримати за реалізацію графічного інтерфейсу.

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