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) *ІП-22 Іщенко К. В.* _____
(шифр, прізвище, ім'я, по батькові)

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

Київ 2023

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	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

3 ВИКОНАННЯ

3.1 Псевдокод алгоритмів

```
Function BTreeSearch(key):
    If root is NULL:
        Return NULL
    Else:
        Set root.comparisonCount to 0
        Return root.search(key)

Function TreeNodeSearch(k):
    Initialize i to 0
    While i < n and k > keys[i]:
        Increment comparisonCount by 1
        Increment i by 1

    If keys[i] equals k:
        Return the current node (this)

    If the current node is a leaf:
        Return NULL

    Set Child[i].comparisonCount to 0
    Return Child[i].search(k)
```

3.2 Часова складність пошуку

O(log n)

3.3 Програмна реалізація

3.3.1 Вихідний код

```
class BTree {
```

```
                TreeNode* root;
                int t;
            public:
                BTree(int);
                TreeNode* search(int);
            };
    TreeNode* BTree::search(int key)
    {
      if (root == NULL) {
        return NULL;
      }
      else {
        root->comparisonCount = 0;
        return root->search(key);
      }
    }

    class TreeNode {
      int* keys;
      std::string* values;
      int t;
      TreeNode** Child;
      int n;
      bool leaf;

    public:
      TreeNode(int, bool);
      int comparisonCount;
      TreeNode* search(int k);
}
TreeNode* TreeNode::search(int k) {
    int i = 0;
    while (i < n && k > keys[i]) {
      comparisonCount++;
      i++;
    }

    if (keys[i] == k) {
```

```
        return this;
    }

    if (leaf == true) {
        return NULL;
    }
    Child[i]->comparisonCount = 0;
    return Child[i]->search(k);
}
```

3.3.2 Приклади роботи

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

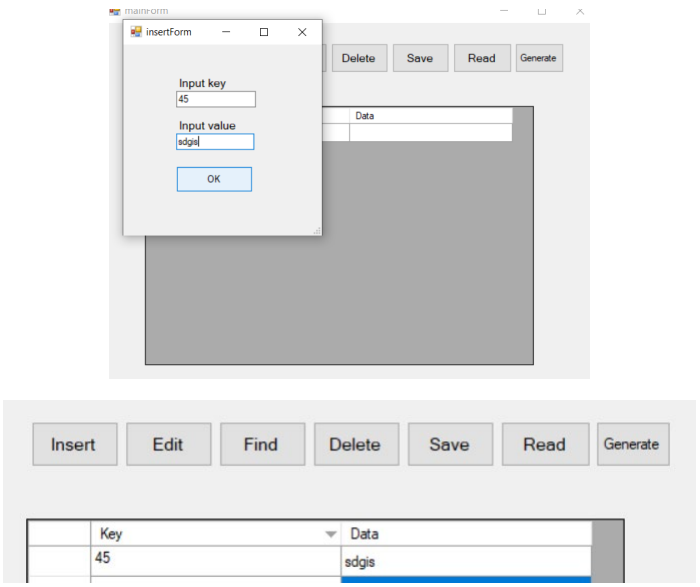

Рисунок 3.1 –Додавання запису

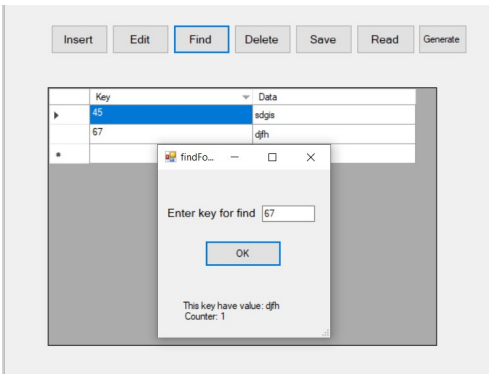

Рисунок 3.2 – Пошук запису

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	1
2	11
3	5
4	9
5	2
6	12
7	21
8	23
9	20
10	11
11	2
12	10
13	0
14	5
15	2

ВИСНОВОК

В рамках лабораторної роботи було розроблено програмну реалізацію структури B-tree. З результатів тестування можемо бачити, що дана структура дозволяє ефективно організувати великі об’єми даних і забезпечити відносно швидкий пошук.

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.