

**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-12 Сімчук Андрій Володимирович*  
(шифр, прізвище, ім'я, по батькові)

**Перевірів**

*Сонов Олексій Олександрович*  
(прізвище, ім'я, по батькові)

Київ 2022

## ЗМІСТ

<b>1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ .....</b>	<b>3</b>
<b>2 ЗАВДАННЯ.....</b>	<b>4</b>
<b>3 ВИКОНАННЯ .....</b>	<b>5</b>
3.1 ПСЕВДОКОД АЛГОРИТМІВ.....	5
3.2 ЧАСОВА СКЛАДНІСТЬ ПОШУКУ .....	6
3.3 ПРОГРАМНА РЕАЛІЗАЦІЯ .....	6
3.3.1 Вихідний код .....	6
3.3.2 Приклади роботи .....	14
3.4 ТЕСТУВАННЯ АЛГОРИТМУ .....	15
3.4.1 Часові характеристики оцінювання.....	15
<b>ВИСНОВОК .....</b>	<b>16</b>
<b>КРИТЕРІЇ ОЦІНЮВАННЯ .....</b>	<b>17</b>

## 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
21	Файли з щільним індексом з перебудовою індексної області, метод Шарпа

## 3.1 Псевдокод алгоритмів

```

k =  $\log_2(\text{size} - 1)$ 
i =  $2^k$ 
b =  $2^k$ 
j = b + 1
l =  $\log_2(\text{size} - 2^k)$ 
x = 2
flag = true
kk = 0
if main_data[k][1] == value do
    print main_data[k][0]
end if
else
    while true do
        kk ++
        b = b / 2
        if main_data[j][1] < value and size - 1 <  $2^k$  do
            j = j + b + 1
        end if
        else if main_data[j][1] > value and size - 1 <  $2^k$  do
            j = j - (b + 1)
        end else if
        else if flag do
            j = size -  $2^1$ 
            b =  $2^{1-1}$ 
            flag = false
        end else if
        else do
            b =  $2^{1-x}$ 
            if main_data[j][1] < value do
                j = j + b + 1
            end if
            else do
                j = j - (b + 1);
            end else
            x++
        end else
        if main_data[j][1] == value do
            print main_data[j][0]
            stop while

```

```

        end if
    end while
    print kk
end else

```

## 3.2 Часова складність пошуку

В середньому алгоритм працює за  $O(\log \log n)$ .

## 3.3 Програмна реалізація

### 3.3.1 Вихідний код

#### lab3.cpp

```

#include "files.h"
using namespace std;

int main() {
    srand(time(0));
    Index_files a;

    ofstream index_out("index.txt");
    ofstream data_out("data.txt");

    int key, data;
    a.index = new int* [a.proc * a.amount];
    for (int i = 0; i < a.proc * a.amount; i++) {
        a.index[i] = new int[2];
    }

    for (int i = 0; i < a.proc * a.amount; i++) {
        a.index[i][0] = INT_MAX;
        a.index[i][1] = INT_MAX;
    }

    a.filled = new int[a.amount];
    for (int i = 0; i < a.amount; i++) {
        a.filled[i] = 0;
    }

    for (int i = 0; i <= 1000; i++) {
        int in_key = i;
        int in_data = i + 1;
        int s = a.search(in_key);
        if (s == -1) a.insert(in_key, in_data);
    }

    for (int i = 0; i < a.proc * a.amount; i++) {

```

```

        index_out << a.index[i][0] << " " << a.index[i][1]
<< endl;
    }

    for (int i = 0; i < a.size; i++) {
        data_out << a.main_data[i][0] << " " <<
a.main_data[i][1] << endl;
    }

    int menu;
    while (true) {
        cout << "Choose:\n1 to insert, 2 to search by
index, 3 to search by data, 4 to remove, 5 to change, 6 to
update, 0 to exit" << endl;
        cin >> menu;

        if (menu == 1) {
            int in_key, in_data;
            cin >> in_key >> in_data;
            if (cin.fail()) {
                cout << "Incorrect entry. Try again:" <<
endl;

                cin.clear();
                cin.ignore();
                continue;
            }
            int s = a.search(in_key);
            if (s == -1) a.insert(in_key, in_data);
        }
        else if (menu == 2) {
            int s_key, s;
            cin >> s_key;
            if (cin.fail()) {
                cout << "Incorrect entry. Try again:" <<
endl;

                cin.clear();
                cin.ignore();
                continue;
            }
            s = a.search(s_key);
            if (s != -1) cout <<
a.main_data[a.index[s][1]][1] << endl;
            else cout << -1 << endl;
        }
        else if (menu == 3) {
            int s_value, s;
            cin >> s_value;
            if (cin.fail()) {
                cout << "Incorrect entry. Try again:" <<
endl;

                cin.clear();
                cin.ignore();
                continue;
            }

```

```

        }
        a.search_data(s_value);
    }
    else if (menu == 4) {
        int r_key;
        cin >> r_key;
        if (cin.fail()) {
            cout << "Incorrect entry. Try again:" <<
endl;

            cin.clear();
            cin.ignore();
            continue;
        }
        a.delete_key(r_key);
    }
    else if (menu == 5) {
        int in_key, in_data;
        cin >> in_key >> in_data;
        if (cin.fail()) {
            cout << "Incorrect entry. Try again:" <<
endl;

            cin.clear();
            cin.ignore();
            continue;
        }
        a.change(in_key, in_data);
    }
    else if (menu == 6) {
        index_out.seekp(index_out.beg);
        for (int i = 0; i < a.proc * a.amount; i++) {
            index_out << a.index[i][0] << " " <<
a.index[i][1] << endl;
        }
        data_out.seekp(data_out.beg);
        for (int i = 0; i < a.size; i++) {
            data_out << a.main_data[i][0] << " " <<
a.main_data[i][1] << endl;
        }
    }
    else if (menu == 0) break;
}
return 0;
}

```

## methods.cpp

```

#include "files.h"

Index_files::Index_files() {
    proc = 5, amount = 10; // proc of zapas, amount of
blocks
    block_interval = 1000;
    size = 0;
}

```



```

    }

    Index_files::~~Index_files() {
        for (int i = 0; i < proc * amount; i++) {
            delete[] index[i];
        }
        delete[] index;

        for (int i = 0; i < size; i++) {
            delete[] main_data[i];
        }
        delete[] main_data;

        delete[] filled;
    }

    void Index_files::insert(int key, int data) {
        if (filled[key / block_interval] >= proc) {

            Index_files::temp_index = new int* [(proc + 1) *
amount];

            for (int i = 0; i < (proc + 1) * amount; i++) {
                Index_files::temp_index[i] = new int[2];
            }

            for (int i = 0; i < proc * amount; i++) {
                Index_files::temp_index[i + i / proc][0] =
index[i][0];
                Index_files::temp_index[i + i / proc][1] =
index[i][1];
            }

            for (int i = 0; i < proc * amount; i++) {
                delete[] index[i];
            }
            delete[] index;

            proc++;

            index = new int* [proc * amount];
            for (int i = 0; i < proc * amount; i++) {
                index[i] = new int[2];
            }

            for (int i = 0; i < proc * amount; i++) {
                index[i][0] = Index_files::temp_index[i][0];
                index[i][1] = Index_files::temp_index[i][1];
            }

            for (int i = 1; i <= amount; i++) {
                index[i * proc - 1][0] = INT_MAX;
                index[i * proc - 1][1] = INT_MAX;
            }

```

```

        for (int i = 0; i < proc * amount; i++) {
            delete[] Index_files::temp_index[i];
        }
        delete[] Index_files::temp_index;
    }

    if (filled[key / block_interval] == 0) {
        index[proc * (key / block_interval)][0] = key;
        index[proc * (key / block_interval)][1] = size;
    }
    else {

        int curr = 0;
        for (int i = proc * (key / block_interval); i <
proc * (key / block_interval + 1); i++) {
            curr = i;
            if (index[i][0] > key) {
                break;
            }
        }

        for (int i = proc * (key / block_interval + 1) -
1; i > curr; i--) {
            index[i][0] = index[i - 1][0];
            index[i][1] = index[i - 1][1];
        }

        index[curr][0] = key;
        index[curr][1] = size;

    }

    if (size == 0) {
        main_data = new int* [1];
        main_data[0] = new int[2];
        main_data[0][1] = data;
        main_data[0][0] = key;
        size++;
    }
    else {
        Index_files::temp_main = new int* [++size];
        for (int i = 0; i < size; i++) {
            Index_files::temp_main[i] = new int[2];
        }

        for (int i = 0; i < size - 1; i++) {
            Index_files::temp_main[i][0] =
main_data[i][0];
            Index_files::temp_main[i][1] =
main_data[i][1];
        }
    }

```

```

        for (int i = 0; i < size - 1; i++) {
            delete[] main_data[i];
        }
        delete[] main_data;

        main_data = new int* [size];
        for (int i = 0; i < size; i++) {
            main_data[i] = new int[2];
        }

        for (int i = 0; i < size; i++) {
            main_data[i][0] =
Index_files::temp_main[i][0];
            main_data[i][1] =
Index_files::temp_main[i][1];
        }

        for (int i = 0; i < size; i++) {
            delete[] Index_files::temp_main[i];
        }
        delete[] Index_files::temp_main;

        main_data[size - 1][0] = key;
        main_data[size - 1][1] = data;

        filled[key / block_interval]++;
    }
}

int Index_files::search(int key) {
    int count = 0;
    int mid;
    int left = (key / block_interval) * proc;
    int right = (key / block_interval) * proc + proc - 1;

    while (index[left][0] <= key && index[right][0] >= key)
    {
        mid = left + ((key - index[left][0]) * (right -
left)) / (index[right][0] - index[left][0]);
        count++;

        if (index[mid][0] < key) left = mid + 1;
        else if (index[mid][0] > key) right = mid - 1;
        else {
            return mid;
        }
    }
    if (index[left][0] == key) {
        return left;
    }
    else return -1;
}

```

```

void Index_files::search_data(int value) {
    int k = log2(size - 1);
    int i = pow(2, k);
    int b = pow(2, k);
    int j = b + 1;
    int l = log2(size - pow(2, k));
    int x = 2;
    bool flag = true;
    int left = 0, right = 0;
    if (main_data[k][1] == value) cout << main_data[k][0]
<< endl;
    else{
        while (true) {
            if (main_data[j][1] < value) right =
main_data[j][1];
            if (main_data[j][1] > value) left =
main_data[j][1];
            b = b / 2;
            if (main_data[j][1] < value && ((size - 1) <
pow(2, k))) j = j + b + 1;
            else if ((main_data[j][1] > value) && ((size
- 1) < pow(2, k))) j = j - (b + 1);
            else if (flag) {
                j = size - pow(2, l);
                b = pow(2, l - 1);
                flag = false;
            }
            else {
                b = pow(2, l - x);
                if (main_data[j][1] < value) j = j + b +
1;
                else j = j - (b + 1);
                x++;
            }

            if (main_data[j][1] == value) {
                cout << main_data[j][0] << endl;
                break;
            }
        }
    }
}

void Index_files::change(int key, int new_data) {
    main_data[index[search(key)][1]][1] = new_data;
}

void Index_files::delete_key(int key) {
    int block = key / block_interval;
    int index_to_delete = search(key);

    if (index_to_delete != proc * (block + 1) - 1) {

```

```

        for (int i = index_to_delete; i < proc * (block +
1) - 1; i++) {
            index[i][0] = index[i + 1][0];
            index[i][1] = index[i + 1][1];
        }
        index[proc * (block + 1) - 1][0] = INT_MAX;
        index[proc * (block + 1) - 1][1] = INT_MAX;
    }
    else {
        index[index_to_delete][0] = INT_MAX;
        index[index_to_delete][1] = INT_MAX;
    }
}

```

## files.h

```

#include <iostream>
#include <fstream>
#include <ctime>
#include <limits.h>
#include <cmath>
using namespace std;

class Index_files {
public:
    int** main_data;
    int** index;
    int size;
    int proc, amount; // percentage of expansion, amount of
blocks
    int block_interval;
    int* filled;

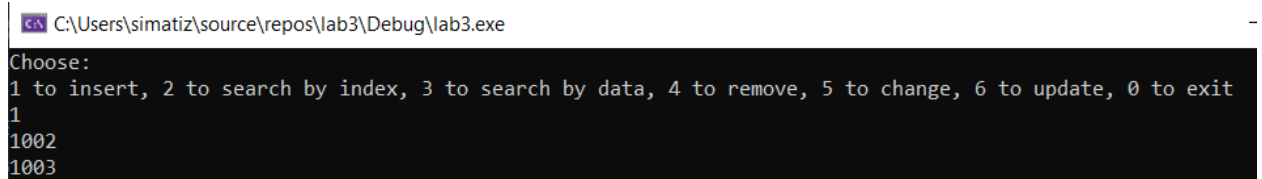
    Index_files();
    ~Index_files();
    void insert(int key, int data);
    int search(int key);
    void search_data(int value);
    void change(int key, int new_data);
    void delete_key(int key);

private:
    int** temp_index;
    int** temp_main;
};

```

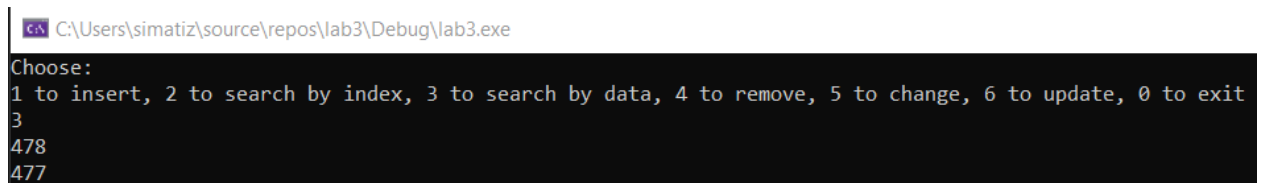
### 3.3.2 Приклади роботи

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



```
C:\Users\simatiz\source\repos\lab3\Debug\lab3.exe
Choose:
1 to insert, 2 to search by index, 3 to search by data, 4 to remove, 5 to change, 6 to update, 0 to exit
1
1002
1003
```

Рисунок 3.1 –Додавання запису



```
C:\Users\simatiz\source\repos\lab3\Debug\lab3.exe
Choose:
1 to insert, 2 to search by index, 3 to search by data, 4 to remove, 5 to change, 6 to update, 0 to exit
3
478
477
```

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	168
2	275
3	518
4	565
5	53
6	24
7	127
8	607
9	327
10	125
11	55
12	428
13	415
14	388
15	70

## ВИСНОВОК

В рамках лабораторної роботи було вивчено основні підходи проектування та обробки складних структур даних на прикладі файлів з щільним індексом з перебудовою індексної області, використовуючи метод Шарра.



## 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.