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 *ІП-12 Басараб Олег Андрійович* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов О. О.* _____
(прізвище, ім'я, по батькові)

Київ 2023

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ 3

2 ЗАВДАННЯ 4

3 ВИКОНАННЯ 7

3.1 Псевдокод алгоритмів 7

3.2 Часова складність пошуку 9

3.3 Програмна реалізація 9

3.3.1 Вихідний код 9

3.3.2 Приклади роботи 16

3.4 Тестування алгоритму 17

3.4.1 Часові характеристики оцінювання 17

ВИСНОВОК 18

КРИТЕРІЇ ОЦІНЮВАННЯ 19

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

Варіант 2 – Файли з щільним індексом з областю переповнення, бінарний пошук

3.1 Псевдокод алгоритмів

Бінарний пошук:

```
procedure doBinarySearch (block, key):
    lo := 0
    hi := block.size () – 1
    while (hi – lo > 1):
        mid := (hi + lo) / 2
        if (getKey (block [mid]) < key):
            lo := mid + 1
        else:
            hi := mid
    if getKey (block [lo] == key):
        return lo
    elif getKey (block [hi] == key):
        return hi
    else:
        return -1
```

Видалення запису за ключем:

```
procedure deleteRecordByKey (key):
    currBlock := indexArea [key % numOfBlocks]
    inBlockIndex := doBinarySearch (currBlock, key)
    if inBlockIndex == -1:
        inOverflowIndex := doBinarySearch (overflowArea, key)
        if inOverflowIndex == -1:
            raise error message
```

else:

deleteFromDB (getMainIndex (overflowArea [inOverflowIndex]))

deleteFromOverflow (inOverflowIndex)

else:

deleteFromDB (getMainIndex (currBlock [inBlockIndex]))

deleteFromIndexBlock (currBlock, inBlockIndex)

Оновлення запису за ключем:

procedure updateRecordByKey (key, value):

currBlock := indexArea [key % numOfBlocks]

inBlockIndex := doBinarySearch (currBlock, key)

if inBlockIndex == -1:

inOverflowIndex := doBinarySearch (overflowArea, key)

if inOverflowIndex == -1:

raise error message

else:

updateMainDB (getMainIndex (overflowArea [inOverflowIndex]), value)

else:

updateMainDB (getMainIndex (currBlock [inBlockIndex]), value)

Пошук запису за ключем:

procedure findRecordByKey (key):

currBlock := indexArea [key % numOfBlocks]

inBlockIndex := doBinarySearch (currBlock, key)

if inBlockIndex == -1:

inOverflowIndex := doBinarySearch (overflowArea, key)

if inOverflowIndex == -1:

raise error message

else:

value = mainDB [getMainIndex (overflowArea [inOverflowIndex])].value

else:

value = mainDB [getMainIndex (currBlock [inBlockIndex])].value

return value

Додавання нового запису:

procedure addRecord (value):

if indexArea [currRecordId % numOfBlocks].size () < numberInBlock:

indexArea [currRecordId % numOfBlocks].add (currRecordId, value)

else:

overflowArea.add (currRecordId, value)

mainDB.add (currRecordId, value)

currRecordId += 1

3.2 Часова складність пошуку

Часова складність пошуку в даній структурі даних складає:

$$O(\log N) + O(\log M),$$

де N – розмір кожного індексного блоку, а M – розмір області переповнення.

3.3 Програмна реалізація

3.3.1 Вихідний код

Файл DenseIndexDatabase.h:

```
#pragma once
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>
#include <ctime>
#include <vector>
using namespace std;

class DenseIndexDatabase {
private:
	vector<vector<string>> indexArea_;
	vector<string> overflowArea_;
	vector<string> mainDatabase_;
	static const int numberInBlock_ = 100;
	static const int numOfBlocks_ = 100;
	int currRecordId_;
```

```
	void getIndexFromFile();
	void getMainFromFile();
	void getOverflowFromFile();
	void createMainFile(int numOfRecords);
	void createIndexFile();
	void addRecord();
	void deleteRecordByKey();
	void findRecordByKey();
	void updateRecordByKey();
	void mainUpdate();
	void indexUpdate();
	void overflowUpdate();

public:
	static const char blockDelimiter = '-';
	static const char inRecordDelimiter = ',';
	static const string indexAreaFileName;
	static const string overflowAreaFileName;
	static const string mainDatabaseFileName;

	void runUI();
	DenseIndexDatabase() {
		getIndexFromFile();
		getMainFromFile();
		getOverflowFromFile();
		currRecordId_ = mainDatabase_.size();
	}
	DenseIndexDatabase(int numOfRecords) {
		createMainFile(numOfRecords);
		getMainFromFile();
		currRecordId_ = mainDatabase_.size();
		createIndexFile();
		getOverflowFromFile();
		getIndexFromFile();
	}
};
```

Файл DenseIndexDatabase.cpp:

```
#include "DenseIndexDatabase.h"

const string DenseIndexDatabase::indexAreaFileName = "index.txt";
const string DenseIndexDatabase::mainDatabaseFileName = "main.txt";
const string DenseIndexDatabase::overflowAreaFileName = "overflow.txt";

void DenseIndexDatabase::getIndexFromFile()
{
	vector<vector<string>> vec(numOfBlocks_);
	ifstream inFile(DenseIndexDatabase::indexAreaFileName);
	string temp;
	int block = 0;
	while (inFile >> temp) {
		if (temp[0] == blockDelimiter) {
			block++;
		}
		else {
			vec[block].push_back(temp);
		}
	}
	inFile.close();
	indexArea_ = vec;
}

void DenseIndexDatabase::getMainFromFile()
{
	vector<string> vec;
	ifstream inFile(DenseIndexDatabase::mainDatabaseFileName);
```

```
	string temp;
	while (inFile >> temp) {
		vec.push_back(temp);
	}
	inFile.close();
	mainDatabase_ = vec;
}

void DenseIndexDatabase::getOverflowFromFile()
{
	vector<string> vec;
	ifstream inFile(DenseIndexDatabase::overflowAreaFileName);
	string temp;
	while (inFile >> temp) {
		vec.push_back(temp);
	}
	inFile.close();
	overflowArea_ = vec;
}

void DenseIndexDatabase::createMainFile(int numberOfRecords)
{
	srand(time(0));
	ofstream outFile(DenseIndexDatabase::mainDatabaseFileName);
	for (int i = 0; i < numberOfRecords; i++) {
		string value = "";
		for (int j = 0; j < 10; j++) {
			value += char(65 + rand() % 25);
		}
		outFile << i << inRecordDelimiter << value << inRecordDelimiter << 1
<< endl;
	}
	outFile.close();
}

void DenseIndexDatabase::createIndexFile()
{
	string temp;
	vector<vector<string>> tempIndex(numOfBlocks_);
	for (int i = 0; i < mainDatabase_.size(); ++i) {
		temp = mainDatabase_[i];
		int delimPos = temp.find(inRecordDelimiter);
		int key = stoi(temp.substr(0, delimPos));
		temp.erase(0, delimPos + 1);
		delimPos = temp.find(inRecordDelimiter);
		string value = temp.substr(0, delimPos);
		temp.erase(0, delimPos + 1);
		bool isNotDeleted = stoi(temp);
		int blockToPaste = key % numOfBlocks_;
		if (tempIndex[blockToPaste].size() < numberInBlock_) {
			tempIndex[blockToPaste].push_back(to_string(key) +
inRecordDelimiter + to_string(i));
		}
		else {
			overflowArea_.push_back(to_string(key) + inRecordDelimiter +
to_string(i));
		}
	}
	indexArea_ = tempIndex;
	indexUpdate();
	overflowUpdate();
}

void DenseIndexDatabase::mainUpdate() {
	ofstream outFile(DenseIndexDatabase::mainDatabaseFileName);
	for (int i = 0; i < mainDatabase_.size(); i++) {
		outFile << mainDatabase_[i] << endl;
```

```
	}
	outFile.close();
}
void DenseIndexDatabase::indexUpdate()
{
	ofstream outFile(DenseIndexDatabase::indexAreaFileName);
	for (int i = 0; i < indexArea_.size(); i++) {
		for (int j = 0; j < indexArea_[i].size(); j++) {
			outFile << indexArea_[i][j] << endl;
		}
		outFile << blockDelimiter << endl;
	}
	outFile.close();
}
void DenseIndexDatabase::overflowUpdate()
{
	ofstream outFile(DenseIndexDatabase::overflowAreaFileName);
	for (int i = 0; i < overflowArea_.size(); i++) {
		outFile << overflowArea_[i] << endl;
	}
	outFile.close();
}

int getKey(string temp) {
	int delimPos = temp.find(DenseIndexDatabase::inRecordDelimiter);
	int key = stoi(temp.substr(0, delimPos + 1));
	return key;
}

int getMainIndex(string temp) {
	int delimPos = temp.find(DenseIndexDatabase::inRecordDelimiter);
	int index = stoi(temp.substr(delimPos + 1, temp.length() - delimPos));
	return index;
}

void DenseIndexDatabase::addRecord() {
	cout << "Please, enter the value of the record you want to add: ";
	string value;
	cin >> value;
	if (indexArea_[currRecordId_ % numOfBlocks_].size() < numberInBlock_) {
		indexArea_[currRecordId_ %
numOfBlocks_].push_back(to_string(currRecordId_) + ',' +
to_string(mainDatabase_.size()));
		indexUpdate();
	}
	else {
		overflowArea_.push_back(to_string(currRecordId_) + inRecordDelimiter
+ to_string(mainDatabase_.size()));
		overflowUpdate();
	}

	mainDatabase_.push_back(to_string(currRecordId_) + inRecordDelimiter +
value + inRecordDelimiter + "1");
	mainUpdate();
	cout << "The record has been successfully added to the database!\n";
	currRecordId_++;
}

int doBinarySearch(vector<string> block, int key, int& comparisonNum) {
	if (block.size() == 0) {
		return -1;
	}
	else {
		int lo = 0, hi = block.size() - 1;

		comparisonNum++;
		while (hi - lo > 1) {
```

```
                        int mid = (hi + lo) / 2;
                        comparisonNum++;
                        if (getKey(block[mid]) < key) {
                                lo = mid + 1;
                        }
                        else {
                                hi = mid;
                        }

                        comparisonNum++;
                }

                comparisonNum++;
                if (getKey(block[lo]) == key) {
                        return lo;
                }
                else if (getKey(block[hi]) == key) {
                        comparisonNum++;
                        return hi;
                }
                else {
                        return -1;
                }
        }
}

void DenseIndexDatabase::deleteRecordByKey() {
        cout << "Please, enter the key of the record you want to delete: ";
        int key;
        cin >> key;
        if (key < 0) {
                cout << "Unfortunately, the database does not contain the record with
entered key\n";
        }
        else {
                vector<string> currIndexBlock = indexArea_[key % numOfBlocks_];
                int indexComparisonsNumber = 0;
                int inBlockIndex = doBinarySearch(currIndexBlock, key,
indexComparisonsNumber);
                if (inBlockIndex == -1) {
                        int overflowComparisonsNumber = 0;
                        int indexInBlock = doBinarySearch(overflowArea_, key,
overflowComparisonsNumber);
                        if (indexInBlock == -1) {
                                cout << "Unfortunately, the database does not contain the
record with entered key\n";
                        }
                        else {
                                string temp =
mainDatabase_[getMainIndex(overflowArea_[indexInBlock])];
                                temp[temp.length() - 1] = '0';
                                mainDatabase_[getMainIndex(overflowArea_[indexInBlock])]
= temp;
                                mainUpdate();
                                vector<string> newOverflow;
                                for (int i = 0; i < overflowArea_.size(); i++) {
                                        if (getKey(overflowArea_[i]) != key) {
                                                newOverflow.push_back(overflowArea_[i]);
                                        }
                                }
                                overflowArea_ = newOverflow;
                                overflowUpdate();
                                cout << "The record has been successfully deleted!\n";
                                cout << "The number of comparisons is: " <<
indexComparisonsNumber + overflowComparisonsNumber << "\n";
                        }
                }
```

```
            else {
                  vector<vector<string>> newIndex(numOfBlocks_);
                  for (int i = 0; i < indexArea_.size(); i++) {
                        for (int j = 0; j < indexArea_[i].size(); j++) {
                              if (getKey(indexArea_[i][j]) != key) {
                                    newIndex[i].push_back(indexArea_[i][j]);
                              }
                        }
                  }
                  indexArea_ = newIndex;
                  indexUpdate();
                  string temp =
mainDatabase_[getMainIndex(currIndexBlock[inBlockIndex])];
                  temp[temp.length() - 1] = '0';
                  mainDatabase_[getMainIndex(currIndexBlock[inBlockIndex])] =
temp;
                  mainUpdate();
                  cout << "The record has been successfully deleted!\n";
                  cout << "The number of comparisons is: " <<
indexComparisonsNumber << "\n";
            }
      }
}

void DenseIndexDatabase::findRecordByKey() {
      cout << "Please, enter the key of the record you want to find: ";
      int key;
      cin >> key;
      if (key < 0) {
            cout << "Unfortunately, the database does not contain the record with
entered key\n";
      }
      else {
            vector<string> currIndexBlock = indexArea_[key % numOfBlocks_];
            int indexComparisonsNumber = 0;
            int inBlockIndex = doBinarySearch(currIndexBlock, key,
indexComparisonsNumber);
            if (inBlockIndex == -1) {
                  int overflowComparisonsNumber = 0;
                  inBlockIndex = doBinarySearch(overflowArea_, key,
overflowComparisonsNumber);
                  if (inBlockIndex == -1) {
                        cout << "Unfortunately, the database does not contain the
record with entered key\n";
                  }
                  else {
                        string temp =
mainDatabase_[getMainIndex(overflowArea_[inBlockIndex])];
                        temp.erase(0, temp.find(inRecordDelimiter) + 1);
                        temp.erase(temp.find(inRecordDelimiter), temp.length() -
temp.find(inRecordDelimiter));
                        cout << "The record has been successfully found, its
value is: " << temp << "\n";
                        cout << "The number of comparisons is: " <<
indexComparisonsNumber + overflowComparisonsNumber << "\n";
                  }
            }
            else {
                  string temp =
mainDatabase_[getMainIndex(currIndexBlock[inBlockIndex])];
                  temp.erase(0, temp.find(inRecordDelimiter) + 1);
                  temp.erase(temp.find(inRecordDelimiter), temp.length() -
temp.find(inRecordDelimiter));
                  cout << "The record has been successfully found, its value is:
" << temp << "\n";
                  cout << "The number of comparisons is: " <<
indexComparisonsNumber << "\n";
```

```
			}
		}
}

void DenseIndexDatabase::updateRecordByKey()
{
		cout << "Please, enter the key of the record you want to update: ";
		int key;
		cin >> key;
		cout << "Enter the new value for the selected record: ";
		string val = "";
		cin >> val;
		if (key < 0) {
			cout << "Unfortunately, the database does not contain the record with
entered key\n";
		}
		else {
			vector<string> blockToFind = indexArea_[key % numOfBlocks_];
			int indexComparisonsNumber = 0;
			int indexInBlock = doBinarySearch(blockToFind, key,
indexComparisonsNumber);
			if (indexInBlock == -1) {
				int overflowComparisonsNumber = 0;
				int indexInBlock = doBinarySearch(overflowArea_, key,
overflowComparisonsNumber);
				if (indexInBlock == -1) {
					cout << "Unfortunately, the database does not contain the
record with entered key\n";
				}
				else {
					string newStr =
to_string(getMainIndex(overflowArea_[indexInBlock])) + "," + val + "," + "1";
					mainDatabase_[getMainIndex(overflowArea_[indexInBlock])]
= newStr;
					mainUpdate();
					cout << "The record has been successfully updated!\n";
					cout << "The number of comparisons is: " <<
indexComparisonsNumber + overflowComparisonsNumber << "\n";
				}
			}
			else {
				string newStr =
to_string(getMainIndex(blockToFind[indexInBlock])) + "," + val + "," + "1";
				mainDatabase_[getMainIndex(blockToFind[indexInBlock])] =
newStr;
				mainUpdate();
				cout << "The record has been successfully updated!\n";
				cout << "The number of comparisons is: " <<
indexComparisonsNumber << "\n";
			}
		}
}

void DenseIndexDatabase::runUI()
{
		int commandId;
		int continueCommandId = 1;
		while (continueCommandId) {

			cout << "Please, select command:\t1 - Add record;\n\t\t\t2 - Delete
record by key;\n\t\t\t3 - Find record by key;\n\t\t\t4 - Update record by
key.\nYour input: ";
			cin >> commandId;
			switch (commandId) {
			case 1:
				addRecord();
				break;
```

```
                case 2:
                        deleteRecordByKey();
                        break;
                case 3:
                        findRecordByKey();
                        break;
                case 4:
                        updateRecordByKey();
                        break;
                default:
                        cout << "Error: Unknown command!\n";
                }

                cout << "\nContinue?\t\t1 - Yes;\n\t\t\t0 - No.\nYour input: ";
                cin >> continueCommandId;
                cout << "\n";
        }
}
```

Файл main.cpp:

```
#include <iostream>
#include "DenseIndexDatabase.h"

using namespace std;

int main()
{
        DenseIndexDatabase db(10000);
        db.runUI();
}
```

3.3.2 Приклади роботи

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

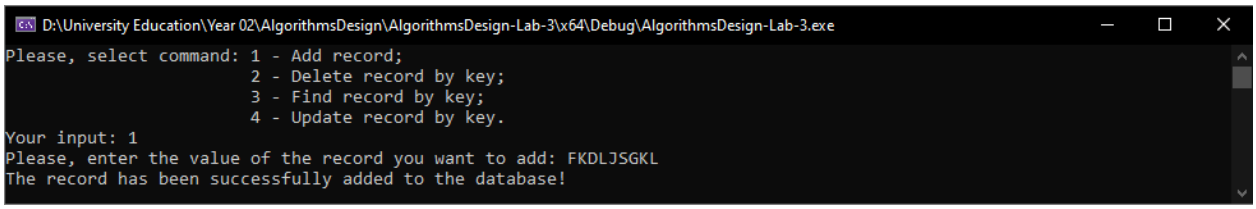

Рисунок 3.1 – Додавання запису

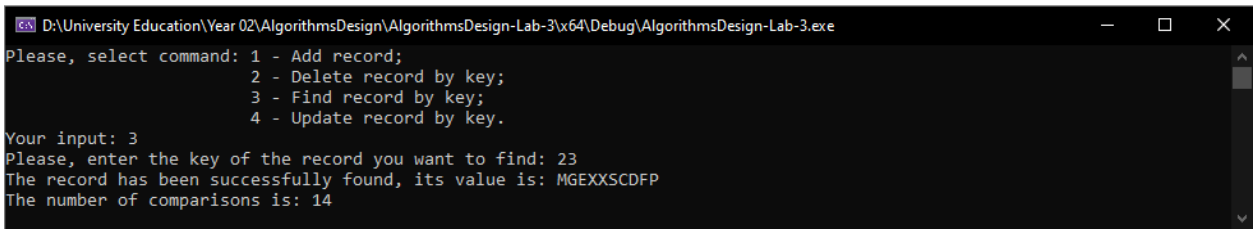

Рисунок 3.2 – Пошук запису

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	Ключ	Число порівнянь
1	1	14
2	5000	14
3	2345	15
4	8492	14
5	47	14
6	9934	14
7	6666	14
8	4235	15
9	7193	14
10	1234	14
11	500	15
12	5500	15
13	8943	15
14	2500	15
15	9043	14

ВИСНОВОК

В рамках лабораторної роботи було опрацьовано та здійснено програмну реалізацію простої бази даних, яка ґрунтується на файлах зі щільним індексом та областю переповнення мовою C++. Для пошуку ключів в реалізованій структурі було використано бінарний пошук. Було проведено тестування реалізованого алгоритму пошуку та занесено його результати до таблиці. Можна стверджувати, що пошук в даній структурі даних здійснюється з часовою складністю $O(\log N) + O(\log M)$, де N – розмір кожного індексного блоку, а M – розмір області переповнення.

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.