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-15 Борисик Владислав Тарасович _____
(шифр, прізвище, ім'я, по батькові)

Перевірив Головченко М.Н. _____
(прізвище, ім'я, по батькові)

Київ 2022

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

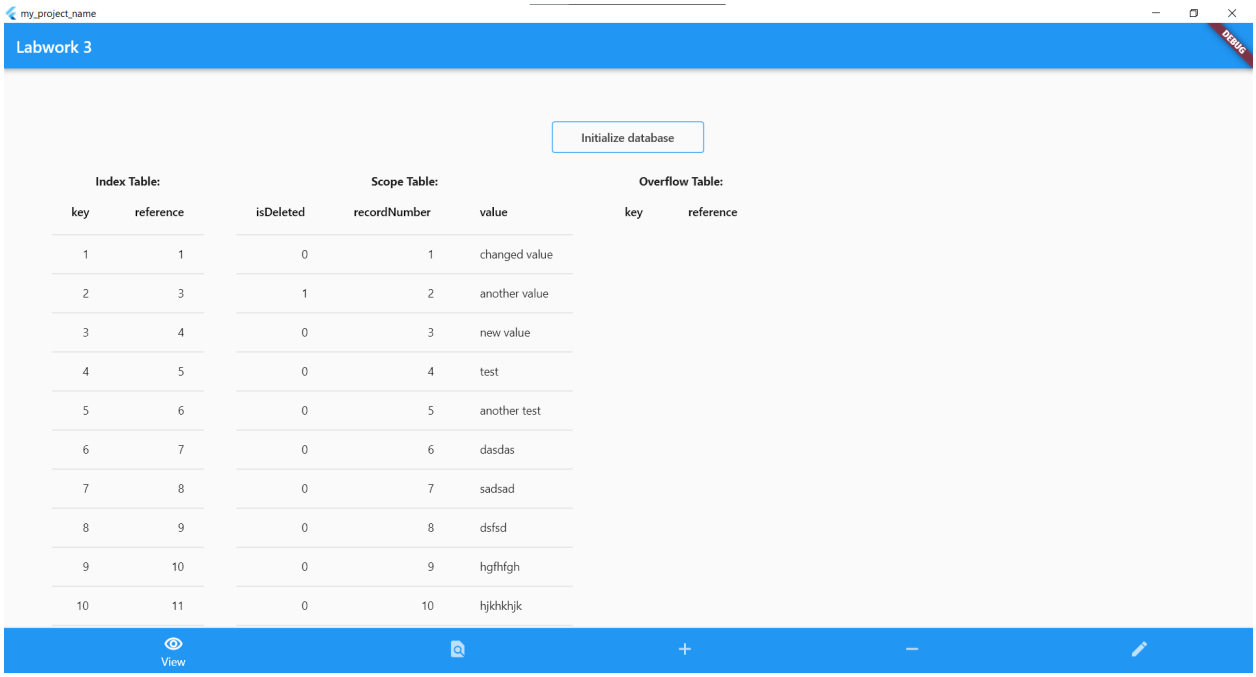
3 ВИКОНАННЯ

Варіант 2

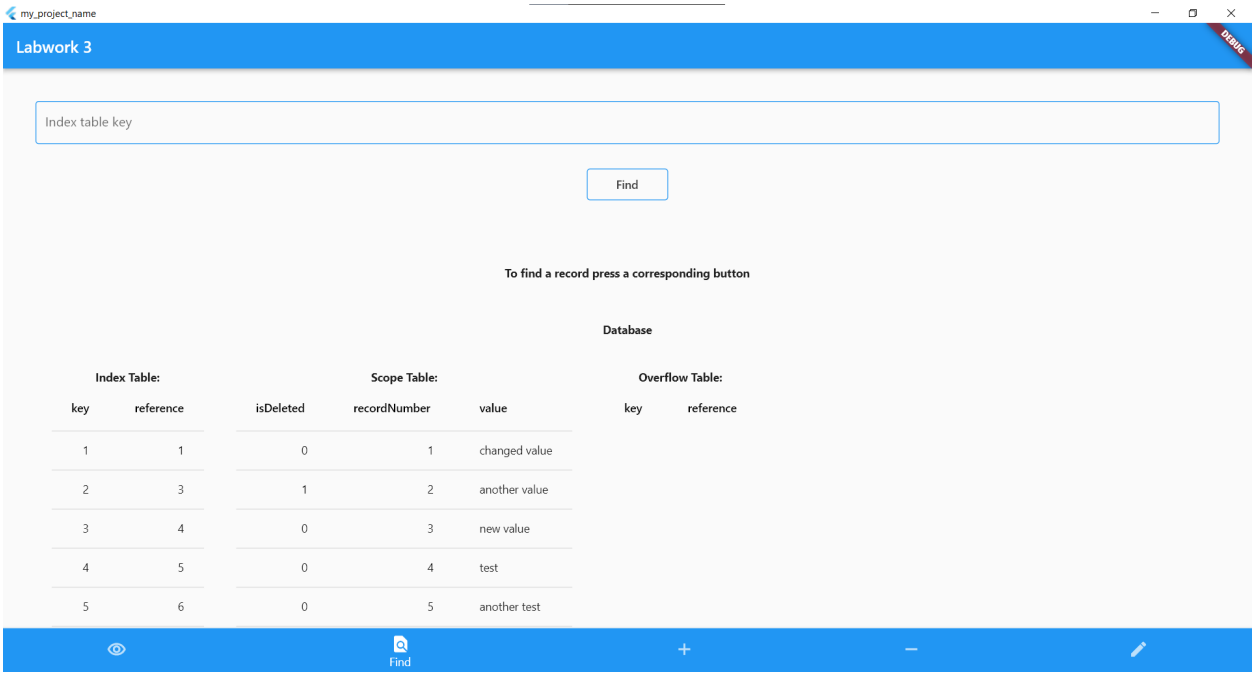
Файли з щільним індексом з областю переповнення, бінарний пошук

Програма має наступний вигляд

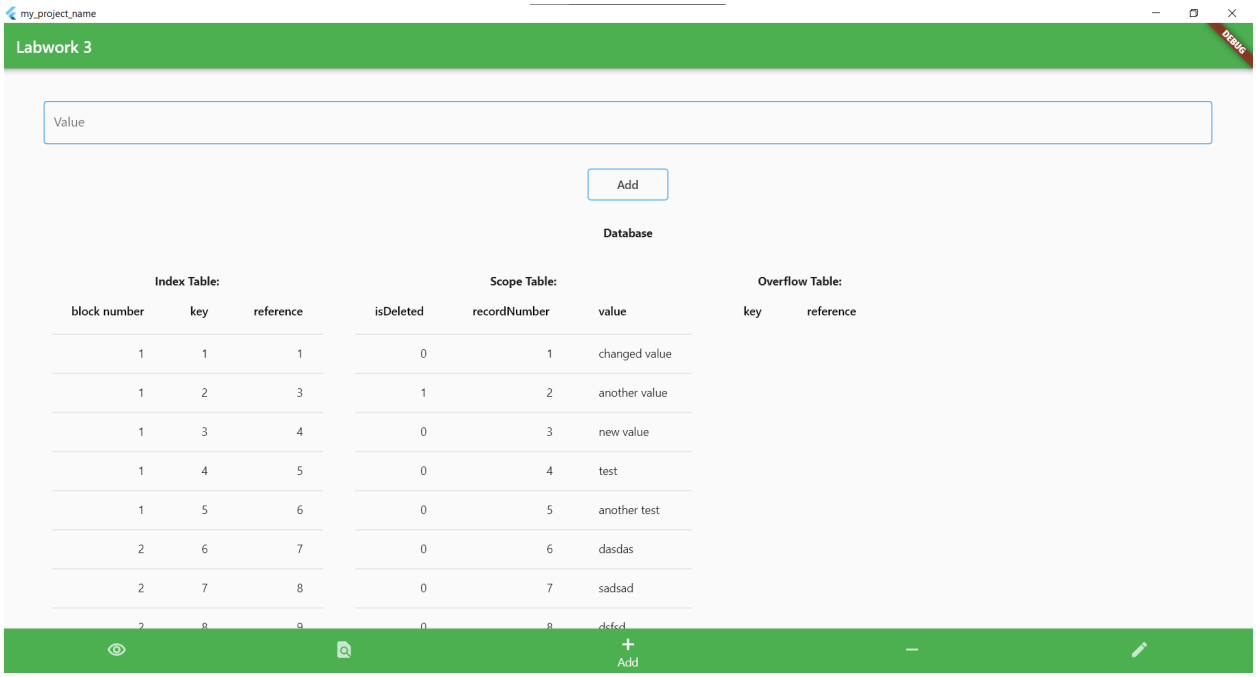
Меню з переглядом бази даних:



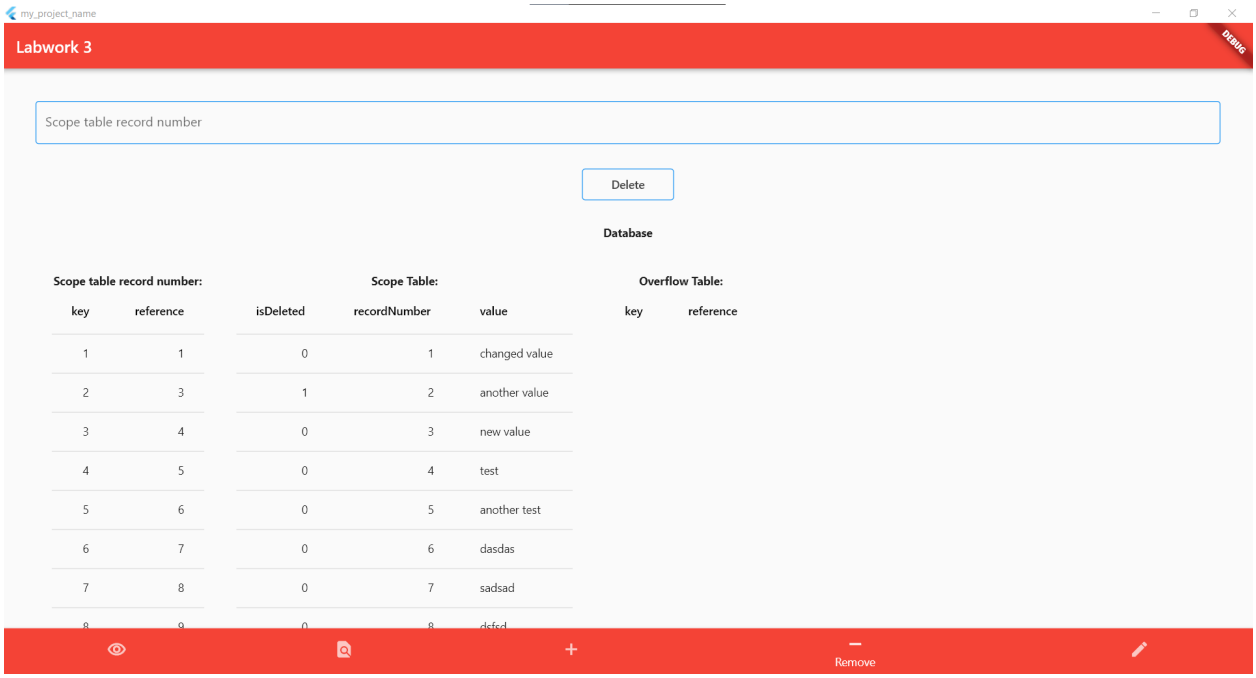
Меню зі знаходженням потрібного запису:



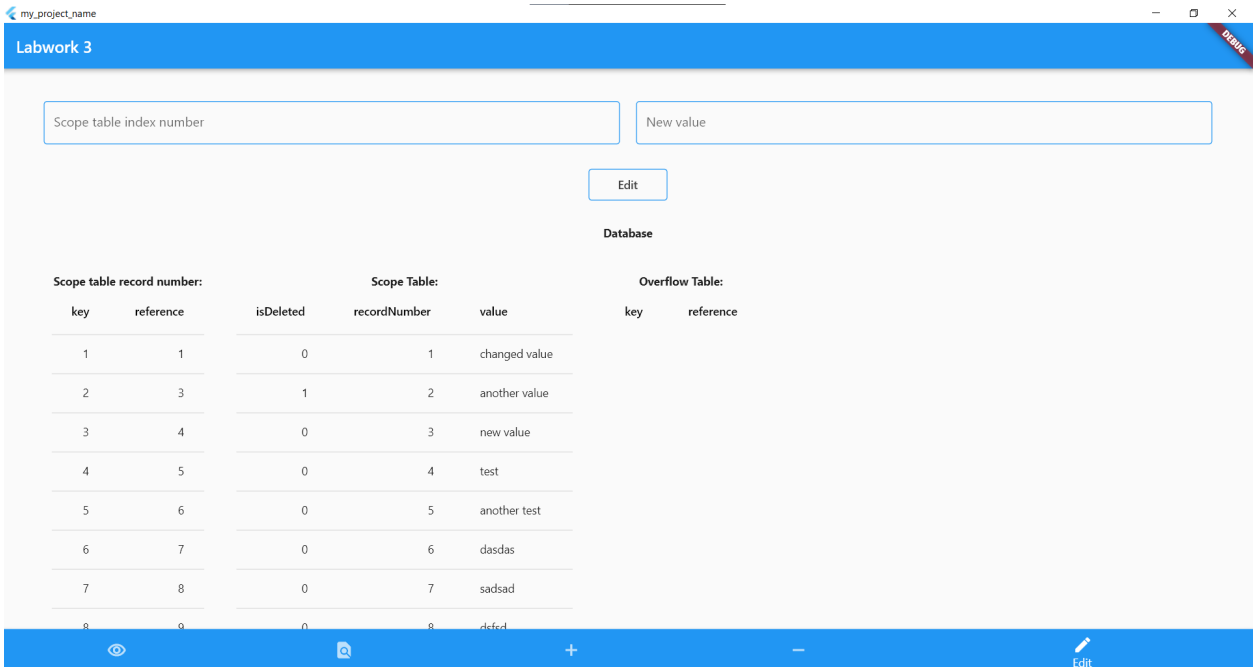
Меню з додаванням запису:



Меню з видаленням запису:



Меню зі зміною запису:



Для створення програми я використовував мову програмування Dart та фреймворк Flutter для побудови графічного інтерфейсу.

Усі операції над базою даних ми будемо виконувати через клас DBManager (він використовує патерн Singleton, тобто клас матиме тільки один екземпляр, і ми будемо мати глобальну точку доступу до цього екземпляра). Він знаходиться в папці lib/classes/DBManager.dart. Псевдокод та код його методів наведемо далі.

3.1 Псевдокод алгоритмів

```
// читаємо індексний файл
 Функція _readIndexFile(){
  // результат (ліст з об'єктами IndexRecord)
  List<IndexRecord> result = [];

  // файл
  File file = File(_indexFilePath);
  // вміст файлу
  String fileContent = file.readAsStringSync();
  // розбиваємо файл на лінії
  List<String> fileContentLines = fileContent.split('\n');

  // для кожного елементу fileContentLines
  Повторити
   Для кожного element в fileContentLines
    // якщо лінія != "key : reference"
    Якщо(!element.contains("key : reference")){
     // парсимо ключ
     int indexKey = int.parse(element.split(" : ")[0]);
     // парсимо посилання
     int indexReference = int.parse(element.split(" : ")[1]);

     // створюємо об'єкт IndexRecord
     IndexRecord record = IndexRecord(indexKey, indexReference);

     // додаємо об'єкт в ліст result
     result.add(record);
    Все якщо
  Все повторити

  повернути result;
 }
```

```
// читаємо файл переповнення (аналогічно до індексного файлу)
Функція readOverflowFile(){
  List<IndexRecord> result = [];

  File file = File(_overflowFilePath);
  var fileContent = file.readAsStringSync();
  var fileContentLines = fileContent.split('\n');
   Повторити
   Для кожного element в fileContentLines
    Якщо(!element.contains("key : reference")){
     int indexKey = int.parse(element.split(" : ")[0]);
     int indexReference = int.parse(element.split(" : ")[1]);

     IndexRecord record = IndexRecord(indexKey, indexReference);

     result.add(record);
    Все якщо
   Все повторити

  Повернути result;
}
```

```
// читаємо файл основної області (аналогічно до індексного файлу)

Функція readScopeFile(){
  List<ScopeRecord> result = [];

  File file = File(_scopeFilePath);
  var fileContent = file.readAsStringSync();
  var fileContentLines = fileContent.split('\n');

   Повторити
   Для кожного element в fileContentLines
    if(!element.contains("isDeleted : recordNumber : value")){
     // парсимо поле, чи видалений елемент
     int isDeleted = int.parse(element.split(" : ")[0]);
     // парсимо номер запису
     int recordNumber = int.parse(element.split(" : ")[1]);
     // парсимо значення
     String value = element.split(" : ")[2];

     result.add(ScopeRecord(isDeleted, recordNumber, value));
    Все якщо
   Все повторити

  Повернути result;
}
```

```
// "відсіюємо" порожні записи із ліста IndexRecord
  Функція _cutEmptyRecords(List<IndexRecord> list){
  List<IndexRecord> cuttedList = [];

  повторити
   для record в list
    якщо (record.key != 0 && record.reference != 0)
     cuttedList.add(record);
    все якщо
  все повторити

  повернути cuttedList;
 }
```

```
// за допомогою бінарного пошуку шукаємо запис в індексній області
 Функція _binarySearchIndexScopeByKey(List<IndexRecord> list, int index){
  // відсіяний ліст (не має порожніх записів)
  List<IndexRecord> cuttedList = _cutEmptyRecords(list);

  // лівий індекс
  int left = 0;
  // правий індекс
  int right = cuttedList.length - 1;
  // центральний індекс
  int middle;

  повторити
   поки left <= right
   // рахуємо центральний індекс
   middle = ((right - left)/2 + left).round();

   // якщо заданий індекс == індексу шуканого елементу
   якщо (index == cuttedList[middle].key){
    // повертаємо шуканий елемент
    повернути cuttedList[middle];
   }
   // якщо заданий індекс менший індексу шуканого елементу
   інакше якщо (index < cuttedList[middle].key){
    // змінюємо правий індекс
    right = middle - 1;
   }
   інакше{
    // змінюємо лівий індекс
    left = middle + 1;
   }
  все повторити

  //якщо елемент не знайдено, то повертаємо порожній IndexRecord
  повернути IndexRecord(0, 0);
 }
```

```
 // знаходимо IndexRecord за посиланням
Функція _findIndexRecordByReference(List<IndexRecord> list, int ref){
 повторити
  для record в list)
   якщо (record.reference == ref)
    повернути record;
   все якщо
 все повторити

 //якщо елемент не знайдено, то повертаємо порожній IndexRecord
 повернути IndexRecord(0, 0);
}



// заповнюємо ліст порожніми елементами
// (наприклад, якщо в лісті 5 елементів,
// але в нас 3 блоки по 5 елементів = загалом 15 елементів,
// то потрібно додати ще 10 порожніх елементів)
Функція _fillEmptyRecords(List<IndexRecord> list){
 int overallCapacity = blockSize * _blocksAmount;

 повторити (int i = list.length; i < overallCapacity; i += 1)
  list.add(IndexRecord(0, 0));
 все повторити
}
```

```
// за подомогою бінарного пошуку видаляємо IndexRecord
 // (аналогічно до функції _binarySearchIndexScopeByKey)
 Функція _binaryDeleteIndexRecord(List<IndexRecord> list, IndexRecord record){
   List<IndexRecord> cuttedList = _cutEmptyRecords(list);

   int left = 0;
   int right = cuttedList.length - 1;
   int middle;

   повторити
    поки left <= right
     middle = ((right - left)/2 + left).round();

     якщо (record.key == cuttedList[middle].key){
       // замінюємо елемент на порожній
       cuttedList[middle] = IndexRecord(0, 0);
     }
     інакше якщо (record.key < cuttedList[middle].key){
       right = middle - 1;
     }
     інакше{
       left = middle + 1;
     }
   все повторити

   // знову видаляємо порожні елементи (тому що в нас з'явився ще один)
   // наприклад, якщо в нас є записи 1:2, 2:3 і 3:5 і потрібно видалити запис з ключем 1,
   // то після видалення отримаємо такі записи: 0:0, 2:3 і 3:5,
   // тому потрібно додатково видалити цей перший запис, який є порожнім
   cuttedList = _cutEmptyRecords(cuttedList);
   // заповнюємо ліст порожніми елементами
   _fillEmptyRecords(cuttedList);

   повернути cuttedList;
 }
```

```
// шукаємо елемент в основній області
 Функція _searchMainScope(List<ScopeRecord> list, int index){
  повторити
   для record в list
    якщо (record.recordNumber == index){
     повернути record;
    все якщо
   все повторити
   //return null;
   повернути ScopeRecord(0, 0, "Empty");
 }


// знаходимо елемент ScopeRecord по ключу IndexRecord
 функція findRecord(int indexKey){
   // отримуємо всі елементи індексної області
   var indexFile = readIndexFile();
   // знаходимо елемент індексної області, який містить потрібний ключ
   var foundIndexRecord = _binarySearchIndexScopeByKey(indexFile, indexKey);

   // отримуємо всі елементи основної області
   var scopeFile = readScopeFile();
   // знаходимо елемент основної області, який містить потрібний ключ
   // (тобто посилання із індексної області)
   var foundScopeRecord = _searchMainScope(scopeFile, foundIndexRecord.reference);

   повернути foundScopeRecord;
 }
```

```
// записуємо елементи в файл індексної області
функція _writeIndexFile(List<IndexRecord> list){
  // файл
  File file = File(_indexFilePath);
  // вміст, який потрібно записати
  String contentToWrite = "key : reference\n";

  повторити
   для record в list
    contentToWrite += "${record.toString()}\n";
  все повторити

  // видаляємо останній символ \n
  contentToWrite = contentToWrite.substring(0, contentToWrite.length - 1);

  // записуэмо вміст у файл
  file.writeAsStringSync(contentToWrite);
}


// записуємо елементи в файл основної області (аналогічно до індексної)
функція _writeScopeFile(List<ScopeRecord> list){
  File file = File(_scopeFilePath);
  String contentToWrite = "isDeleted : recordNumber : value\n";

  повторити
   для record в list
    contentToWrite += "${record.toString()}\n";
   все повторити

  contentToWrite = contentToWrite.substring(0, contentToWrite.length - 1);

  file.writeAsStringSync(contentToWrite);
}
```

```
// записуємо елементи в файл області переповнення (аналогічно до індексної)

функція _writeOverflowFile(List<IndexRecord> list){
  File file = File(_overflowFilePath);
  String contentToWrite = "key : reference\n";

  повторити
   для record в list
    contentToWrite += "${record.toString()}\n";
   все повторити

  contentToWrite = contentToWrite.substring(0, contentToWrite.length - 1);

  file.writeAsStringSync(contentToWrite);
 }



// отримуємо індекс останнього елементу основної області

функція getLastScopeFileIndex(){
  var scopeFileElements = readScopeFile();

  // якщо основна область не порожня
  якщо (scopeFileElements.length > 0){
   повернути scopeFileElements[scopeFileElements.length - 1].recordNumber;
  }
  // якщо основна область порожня
  інакше{
   повернути 0;
  }
 }


// рахуємо номер блоку, в який потрібно записати елемент

функція _defineBlock(int index){
  повернути index % blockSize == 0
  ? ((index / blockSize) - 1).toInt()
  : (index / blockSize).floor();
 }
```

```
// розбиваємо ліст з елементами IndexRecord на блоки (2D ліст/матрицю)

функція _divideIndexFileToBlocks(List<IndexRecord> records){
  List<List<IndexRecord>> dividedList = [];

  List<IndexRecord> tempDividedList = [];
  повторити (int i = 0; i < records.length; i += 1){
    якщо (i % blockSize != 0 || i == 0){
      tempDividedList.add(records[i]);
    }
    інакше{
      dividedList.add(tempDividedList);
      tempDividedList = [];

      tempDividedList.add(records[i]);
    }
  }
  dividedList.add(tempDividedList);

  повернути dividedList;
}
```

```
// додаємо елемент у блок
  функція   _addItemToBlock(List<List<IndexRecord>>   dividedIndexList,   int   blockNumber,
IndexRecord item){
  // рахуємо його позицію всередині блоку
  int positionInsideBlock = (item.key - 1) % blockSize;
  // вставляємо елемент
  dividedIndexList[blockNumber][positionInsideBlock] = item;
}


// записуємо блоки індекснох області у файл
функція _writeIndexBlocks(List<List<IndexRecord>> blocks){
  File file = File(_indexFilePath);
  String contentToWrite = "key : reference\n";

  повторити (var block in blocks) {
   повторити(var item in block){
    contentToWrite += "${item.toString()}\n";
   }
  }

  // видаляємо останній символ \n
  contentToWrite = contentToWrite.substring(0, contentToWrite.length - 1);

  file.writeAsStringSync(contentToWrite);
}


// отримуємо останній запис індексної області
функція _getLastIndexRecord(){
  var indexFileElements = readIndexFile();

  повторити (int i = indexFileElements.length - 1; i >= 0; i -= 1){
   якщо (indexFileElements[i].key != 0 && indexFileElements[i].reference != 0){
    повернути indexFileElements[i];
   }
  }

  повернути IndexRecord(0, 0);
}
```

```
  // перевіряємо чи індексна область заповнена
функція _isIndexTableFull(List<IndexRecord> indexFileElements){
  // загальна місткість індексної області
  int overallCapacity = blockSize * _blocksAmount;
  // кількість непустих елементів
  int nonEmptyRecordCount = 0;

  повторити (var record in indexFileElements) {
    // якщо елемент не пустий
    якщо (record.key != 0 && record.reference != 0){
      nonEmptyRecordCount += 1;
    }
  }
  // повертаємо true, якщо кількість непустих елементів == місткості індексної області,
  // інакше повертаємо false
  повернути nonEmptyRecordCount == overallCapacity ? true : false;
}
```

```
// додаємо запис
функція addRecord(ScopeRecord record){
  // елементи індексної області
  var indexFileElements = readIndexFile();
  // елементи основної області
  var scopeFileElements = readScopeFile();

  // створюємо об'єкт ScopeRecord
  var scopeRecordToAdd = ScopeRecord(record.isDeleted, record.recordNumber, record.value);
  // додаємо цей об'єкт в scopeFileElements
  scopeFileElements.add(scopeRecordToAdd);
  // записуємо scopeFileElements в файл основної області
  _writeScopeFile(scopeFileElements);

  // генеруємо ключ в залежності від того, що заповнена основна область
  int key = _isIndexTableFull(indexFileElements)
  // якщо основна область заповнена
  // генеруємо ключ із області переповнення
  ? _getLastIndexRecordOverflowTable().key + 1
  // якщо основна область не заповнена
  // генеруємо ключ із індексної області
  : _getLastIndexRecord().key + 1;
  int reference = record.recordNumber;
  var indexRecordToAdd = IndexRecord(key, reference);

  // якщо індексна область не заповнена
  якщо (_isIndexTableFull(indexFileElements) == false){
    // розбиваємо область на блоки
    var dividedIndexList = _divideIndexFileToBlocks(indexFileElements);
    // отримуємо блок в який потрібно запести запис
    int blockToAdd = _defineBlock(indexRecordToAdd.key);
    // заносимо запис у блок
    _addItemToBlock(dividedIndexList, blockToAdd, indexRecordToAdd);
    // записуємо блоки у файл
    _writeIndexBlocks(dividedIndexList);
  }
  // якщо індексна область заповнена
  інакше{
    // отримуємо записи з області переповнення
    var overflowItems = readOverflowFile();
    // додамо туди запис
    overflowItems.add(indexRecordToAdd);
    // записуємо все в область переповнення
    _writeOverflowFile(overflowItems);
  }
```

```
// помічаємо елемент як видалений
 функція _markRecordAsRemoved(List<ScopeRecord> records, int recordNumber){
  повторити (var record in records) {
   якщо (record.recordNumber == recordNumber){
    record.isDeleted = 1;
   }
  }
 }


// видаляємо елемент
 функція deleteRecord(int recordNumber){

  // елементи індексної області
  var indexFileElements = readIndexFile();
  // елементи основної області
  var scopeFileElements = readScopeFile();

  // помічаємо елемент як видалений
  _markRecordAsRemoved(scopeFileElements, recordNumber);
  // записуємо scopeFileElements в основну область
  _writeScopeFile(scopeFileElements);

  // отримуємо IndexRecord який потрібно видалити
  IndexRecord indexRecordToDelete = _findIndexRecordByReference(indexFileElements,
recordNumber);
  // ліст з видаленим IndexRecord
  var removedIndexRecordList = _binaryDeleteIndexRecord(indexFileElements,
indexRecordToDelete);
  // записуємо removedIndexRecordList в індексну область
  _writeIndexFile(removedIndexRecordList);
 }


 // змінюємо значення запису
 функція _changeRecordValue(List<ScopeRecord> records, int recordNumber, String newValue){
  повторити (var record in records) {
   якщо (record.recordNumber == recordNumber){
    record.value = newValue;
   }
  }
 }
```

```
 // змінюємо запис
функція editRecord(int recordNumber, String newValue){
 // елементи основної області
 var scopeFileElements = readScopeFile();

 // змінюємо значення запису
 _changeRecordValue(scopeFileElements, recordNumber, newValue);

 // записуємо scopeFileElements в основну область
 _writeScopeFile(scopeFileElements);
}
```

```
  // ініціалізуємо базу даних
  функція initializeDatabase(){
    // файл індексної області
    File indexFile = File(_indexFilePath);
    // файл основної області
    File scopeFile = File(_scopeFilePath);
    // файл області переповнення
    File overflowFile = File(_overflowFilePath);

    // записуємо "шапку" в основну область
    scopeFile.writeAsStringSync("isDeleted : recordNumber : value");
    // записуємо "шапку" в область переповнення
    overflowFile.writeAsStringSync("key : reference");

    // вміст, який потрібно записати
    String indexFileStringToWrite = "key : reference\n";
    // місткість індексної області
    int overallCapacity = _blocksAmount * blockSize;
    // формуємо "пусті" елементи
    повторити (int i = 0; i < overallCapacity; i += 1){
      IndexRecord record = IndexRecord(0, 0);
      indexFileStringToWrite += record.toString() + '\n';
    }
    // видаляємо останній символ "\n"
    indexFileStringToWrite = indexFileStringToWrite.substring(0,
indexFileStringToWrite.length - 1);
    // записуємо indexFileStringToWrite в індексну область
    indexFile.writeAsStringSync(indexFileStringToWrite);
  }
```

3.2 Часова складність пошуку

За умовою завдання потрібно використовувати бінарний пошук, отже, розглянемо його часову складність:

Найкращий випадок – у бінарному пошуку найкращий випадок виникає, коли шуканий елемент знайдено під час першого порівняння, тобто коли перший елемент сам є елементом, якого шукають. Найкраща часова складність бінарного пошуку становить **O(1)**.

Середній випадок - середня складність бінарного пошуку становить **O(logn)**.

Найгірший випадок складності - у бінарному пошуку найгірший випадок виникає, коли нам потрібно скорочувати простір пошуку, поки в ньому не залишиться лише один елемент. Найгірша часова складність бінарного пошуку – **O(logn)**.

3.3 Програмна реалізація

3.3.1 Вихідний код

records.dart:

```dart
// запис індексної області
class IndexRecord{
  // ключ
  final int key;
  // посилання на recordNumber з основної області
  final int reference;

  // конструктор
  IndexRecord(this.key, this.reference);

  @override
  String toString(){
    return "${key} : ${reference}";
  }
}

// запис основної області
class ScopeRecord{
  // чи видалений запис (0 - ні, 1 - так)
  int isDeleted;
  // номер запису
  final int recordNumber;
  // значення запису
  String value;

  // конструктор
  ScopeRecord(this.isDeleted, this.recordNumber, this.value);

  @override
  String toString(){
    return "${isDeleted} : ${recordNumber} : ${value}";
  }
}
```

DBManager,dart:

```dart
import 'dart:io';
import '../classes/records.dart';

class DBManager{
  // використовуємо патерн Singleton
  // (клас матиме тільки один екземпляр,
  // і ми будемо мати глобальну точку доступу до цього екземпляра)
  static final DBManager instance = DBManager._init();
  DBManager._init();

  // шлях до індексного файлу
  final String _indexFilePath = "F:/Projects/Flutter/Labwork_3/lib/files/index_file.txt";
  // шлях до файлу переповнення
  final String _overflowFilePath =
"F:/Projects/Flutter/Labwork_3/lib/files/overflow_file.txt";
  // шлях до файлу основної області
  final String _scopeFilePath = "F:/Projects/Flutter/Labwork_3/lib/files/scope_file.txt";

  // розмір блоку (включаючи область переповнення)
  final int blockSize = 5;
  // кількість блоків
  final int _blocksAmount = 5;

  // читаємо індексний файл
  List<IndexRecord> readIndexFile(){
    // результат (ліст з об'єктами IndexRecord)
    List<IndexRecord> result = [];

    // файл
    File file = File(_indexFilePath);
    // вміст файлу
    var fileContent = file.readAsStringSync();
    // розбиваємо файл на лінії
    var fileContentLines = fileContent.split('\n');

    // для кожного елементу fileContentLines
    fileContentLines.forEach((element) {
      // якщо лінія != "key : reference"
      if(!element.contains("key : reference")){
        // парсимо ключ
        int indexKey = int.parse(element.split(" : ")[0]);
        // парсимо посилання
        int indexReference = int.parse(element.split(" : ")[1]);
```

```
      // створюємо об'єкт IndexRecord
      IndexRecord record = IndexRecord(indexKey, indexReference);

      // додаємо об'єкт в ліст result
      result.add(record);
    }
  },);

  return result;
 }

 // читаємо файл переповнення (аналогічно до індексного файлу)
 List<IndexRecord> readOverflowFile(){
  List<IndexRecord> result = [];

  File file = File(_overflowFilePath);
  var fileContent = file.readAsStringSync();
  var fileContentLines = fileContent.split('\n');

  fileContentLines.forEach((element) {
   if(!element.contains("key : reference")){
    int indexKey = int.parse(element.split(" : ")[0]);
    int indexReference = int.parse(element.split(" : ")[1]);

    IndexRecord record = IndexRecord(indexKey, indexReference);

    result.add(record);
   }
  },);

  return result;
 }

 // читаємо файл основної області (аналогічно до індексного файлу)
 List<ScopeRecord> readScopeFile(){
  List<ScopeRecord> result = [];

  File file = File(_scopeFilePath);
  var fileContent = file.readAsStringSync();
  var fileContentLines = fileContent.split('\n');

  fileContentLines.forEach((element) {
   if(!element.contains("isDeleted : recordNumber : value")){
    // парсимо поле, чи видалений елемент
```

```
      int isDeleted = int.parse(element.split(" : ")[0]);
      // парсимо номер запису
      int recordNumber = int.parse(element.split(" : ")[1]);
      // парсимо значення
      String value = element.split(" : ")[2];

      result.add(ScopeRecord(isDeleted, recordNumber, value));
    }
  },);

  return result;
}

// "відсіюємо" порожні записи із ліста IndexRecord
List<IndexRecord> _cutEmptyRecords(List<IndexRecord> list){
  List<IndexRecord> cuttedList = [];

  for (var record in list) {
    if(record.key != 0 && record.reference != 0){
      cuttedList.add(record);
    }
  }

  return cuttedList;
}

// за допомогою бінарного пошуку шукаємо запис в індексній області
IndexRecord _binarySearchIndexScopeByKey(List<IndexRecord> list, int index){
  // відсіяний ліст (не має порожніх записів)
  List<IndexRecord> cuttedList = _cutEmptyRecords(list);

  // лівий індекс
  int left = 0;
  // правий індекс
  int right = cuttedList.length - 1;
  // центральний індекс
  int middle;

  while(left <= right){
    // рахуємо центральний індекс
    middle = ((right - left)/2 + left).round();

    // якщо заданий індекс == індексу шуканого елементу
    if(index == cuttedList[middle].key){
      // повертаємо шуканий елемент
```

```
      return cuttedList[middle];
    }
    // якщо заданий індекс менший індексу шуканого елементу
    else if (index < cuttedList[middle].key){
      // змінюємо правий індекс
      right = middle - 1;
    }
    else{
      // змінюємо лівий індекс
      left = middle + 1;
    }
  }

  //якщо елемент не знайдено, то повертаємо порожній IndexRecord
  return IndexRecord(0, 0);
}

// знаходимо IndexRecord за посиланням
IndexRecord _findIndexRecordByReference(List<IndexRecord> list, int ref){
  for (var record in list) {
    if(record.reference == ref){
      return record;
    }
  }

  //якщо елемент не знайдено, то повертаємо порожній IndexRecord
  return IndexRecord(0, 0);
}

// заповнюємо ліст порожніми елементами
// (наприклад, якщо в лісті 5 елементів,
// але в нас 3 блоки по 5 елементів = загалом 15 елементів,
// то потрібно додати ще 10 порожніх елементів)
void _fillEmptyRecords(List<IndexRecord> list){
  int overallCapacity = blockSize * _blocksAmount;

  for(int i = list.length; i < overallCapacity; i += 1){
    list.add(IndexRecord(0, 0));
  }
}

// за подомогою бінарного пошуку видаляємо IndexRecord
// (аналогічно до функції _binarySearchIndexScopeByKey)
List<IndexRecord> _binaryDeleteIndexRecord(List<IndexRecord> list, IndexRecord record){
  List<IndexRecord> cuttedList = _cutEmptyRecords(list);
```

```
  int left = 0;
  int right = cuttedList.length - 1;
  int middle;

  while(left <= right){
   middle = ((right - left)/2 + left).round();

   if(record.key == cuttedList[middle].key){
    // замінюємо елемент на порожній
    cuttedList[middle] = IndexRecord(0, 0);
   }
   else if (record.key < cuttedList[middle].key){
    right = middle - 1;
   }
   else{
    left = middle + 1;
   }
  }

  // знову видаляємо порожні елементи (тому що в нас з'явився ще один)
  // наприклад, якщо в нас є записи 1:2, 2:3 і 3:5 і потрібно видалити запис з ключем 1,
  // то після видалення отримаємо такі записи: 0:0, 2:3 і 3:5,
  // тому потрібно додатково видалити цей перший запис, який є порожнім
  cuttedList = _cutEmptyRecords(cuttedList);
  // заповнюємо ліст порожніми елементами
  _fillEmptyRecords(cuttedList);

  return cuttedList;
 }

 // шукаємо елемент в основній області
 ScopeRecord _searchMainScope(List<ScopeRecord> list, int index){
  for (var record in list) {
   if(record.recordNumber == index){
    return record;
   }
  }
  //return null;
  return ScopeRecord(0, 0, "Empty");
 }

 // знаходимо елемент ScopeRecord по ключу IndexRecord
 ScopeRecord findRecord(int indexKey){
  // отримуємо всі елементи індексної області
```

```
  var indexFile = readIndexFile();
  // знаходимо елемент індексної області, який містить потрібний ключ
  var foundIndexRecord = _binarySearchIndexScopeByKey(indexFile, indexKey);

  // отримуємо всі елементи основної області
  var scopeFile = readScopeFile();
  // знаходимо елемент основної області, який містить потрібний ключ
  // (тобто посилання із індексної області)
  var foundScopeRecord = _searchMainScope(scopeFile, foundIndexRecord.reference);

  return foundScopeRecord;
 }

 // записуємо елементи в файл індексної області
 void _writeIndexFile(List<IndexRecord> list){
  // файл
  File file = File(_indexFilePath);
  // вміст, який потрібно записати
  String contentToWrite = "key : reference\n";

  for (var record in list) {
   contentToWrite += "${record.toString()}\n";
  }

  // видаляємо останній символ \n
  contentToWrite = contentToWrite.substring(0, contentToWrite.length - 1);

  // записуэмо вміст у файл
  file.writeAsStringSync(contentToWrite);
 }

 // записуємо елементи в файл основної області (аналогічно до індексної)
 void _writeScopeFile(List<ScopeRecord> list){
  File file = File(_scopeFilePath);
  String contentToWrite = "isDeleted : recordNumber : value\n";

  for (var record in list) {
   contentToWrite += "${record.toString()}\n";
  }

  contentToWrite = contentToWrite.substring(0, contentToWrite.length - 1);

  file.writeAsStringSync(contentToWrite);
 }
```

```
// записуємо елементи в файл області переповнення (аналогічно до індексної)
void _writeOverflowFile(List<IndexRecord> list){
  File file = File(_overflowFilePath);
  String contentToWrite = "key : reference\n";

  for (var record in list) {
    contentToWrite += "${record.toString()}\n";
  }

  contentToWrite = contentToWrite.substring(0, contentToWrite.length - 1);

  file.writeAsStringSync(contentToWrite);
}

// отримуємо індекс останнього елементу основної області
int getLastScopeFileIndex(){
  var scopeFileElements = readScopeFile();

  // якщо основна область не порожня
  if(scopeFileElements.length > 0){
    return scopeFileElements[scopeFileElements.length - 1].recordNumber;
  }
  // якщо основна область порожня
  else{
    return 0;
  }
}

// рахуємо номер блоку, в який потрібно записати елемент
int _defineBlock(int index){
  return index % blockSize == 0
  ? ((index / blockSize) - 1).toInt()
  : (index / blockSize).floor();
}

// розбиваємо ліст з елементами IndexRecord на блоки (2D ліст/матрицю)
List<List<IndexRecord>> _divideIndexFileToBlocks(List<IndexRecord> records){
  List<List<IndexRecord>> dividedList = [];

  List<IndexRecord> tempDividedList = [];
  for(int i = 0; i < records.length; i += 1){
    if(i % blockSize != 0 || i == 0){
      tempDividedList.add(records[i]);
    }
    else{
```

```
      dividedList.add(tempDividedList);
      tempDividedList = [];

      tempDividedList.add(records[i]);
    }
   }
   dividedList.add(tempDividedList);

   return dividedList;
  }

  // додаємо елемент у блок
  void _addItemToBlock(List<List<IndexRecord>> dividedIndexList, int blockNumber,
IndexRecord item){
   // рахуємо його позицію всередині блоку
   int positionInsideBlock = (item.key - 1) % blockSize;
   // вставляємо елемент
   dividedIndexList[blockNumber][positionInsideBlock] = item;
  }

  // записуємо блоки індекснох області у файл
  void _writeIndexBlocks(List<List<IndexRecord>> blocks){
   File file = File(_indexFilePath);
   String contentToWrite = "key : reference\n";

   for (var block in blocks) {
    for(var item in block){
     contentToWrite += "${item.toString()}\n";
    }
   }

   // видаляємо останній символ \n
   contentToWrite = contentToWrite.substring(0, contentToWrite.length - 1);

   file.writeAsStringSync(contentToWrite);
  }

  // отримуємо останній запис індексної області
  IndexRecord _getLastIndexRecord(){
   var indexFileElements = readIndexFile();

   for(int i = indexFileElements.length - 1; i >= 0; i -= 1){
    if(indexFileElements[i].key != 0 && indexFileElements[i].reference != 0){
     return indexFileElements[i];
    }
```

```
    }

    return IndexRecord(0, 0);
  }

  // отримуємо останній запис області переповнення
  IndexRecord _getLastIndexRecordOverflowTable(){
    var indexFileElements = readOverflowFile();

    for(int i = indexFileElements.length - 1; i >= 0; i -= 1){
      if(indexFileElements[i].key != 0 && indexFileElements[i].reference != 0){
        return indexFileElements[i];
      }
    }

    return IndexRecord(0, 0);
  }

  // перевіряємо чи індексна область заповнена
  bool _isIndexTableFull(List<IndexRecord> indexFileElements){
    // загальна місткість індексної області
    int overallCapacity = blockSize * _blocksAmount;
    // кількість непустих елементів
    int nonEmptyRecordCount = 0;

    for (var record in indexFileElements) {
      // якщо елемент не пустий
      if(record.key != 0 && record.reference != 0){
        nonEmptyRecordCount += 1;
      }
    }

    // повертаємо true, якщо кількість непустих елементів == місткості індексної області,
    // інакше повертаємо false
    return nonEmptyRecordCount == overallCapacity ? true : false;
  }

  // додаємо запис
  void addRecord(ScopeRecord record){
    // елементи індексної області
    var indexFileElements = readIndexFile();
    // елементи основної області
    var scopeFileElements = readScopeFile();

    // створюємо об'єкт ScopeRecord
```

```
  var scopeRecordToAdd = ScopeRecord(record.isDeleted, record.recordNumber, record.value);
  // додаємо цей об'єкт в scopeFileElements
  scopeFileElements.add(scopeRecordToAdd);
  // записуємо scopeFileElements в файл основної області
  _writeScopeFile(scopeFileElements);

  // генеруємо ключ в залежності від того, що заповнена основна область
  int key = _isIndexTableFull(indexFileElements)
  // якщо основна область заповнена
  // генеруємо ключ із області переповнення
  ? _getLastIndexRecordOverflowTable().key + 1
  // якщо основна область не заповнена
  // генеруємо ключ із індексної області
  : _getLastIndexRecord().key + 1;
  int reference = record.recordNumber;
  var indexRecordToAdd = IndexRecord(key, reference);

  // якщо індексна область не заповнена
  if(_isIndexTableFull(indexFileElements) == false){
   // розбиваємо область на блоки
   var dividedIndexList = _divideIndexFileToBlocks(indexFileElements);
   // отримуємо блок в який потрібно запести запис
   int blockToAdd = _defineBlock(indexRecordToAdd.key);
   // заносимо запис у блок
   _addItemToBlock(dividedIndexList, blockToAdd, indexRecordToAdd);
   // записуємо блоки у файл
   _writeIndexBlocks(dividedIndexList);
  }
  // якщо індексна область заповнена
  else{
   // отримуємо записи з області переповнення
   var overflowItems = readOverflowFile();
   // додамо туди запис
   overflowItems.add(indexRecordToAdd);
   // записуємо все в область переповнення
   _writeOverflowFile(overflowItems);
  }
 }

 // помічаємо елемент як видалений
 void _markRecordAsRemoved(List<ScopeRecord> records, int recordNumber){
  for (var record in records) {
   if(record.recordNumber == recordNumber){
    record.isDeleted = 1;
   }
```

```
  }
 }

 // видаляємо елемент
 void deleteRecord(int recordNumber){

  // елементи індексної області
  var indexFileElements = readIndexFile();
  // елементи основної області
  var scopeFileElements = readScopeFile();

  // помічаємо елемент як видалений
  _markRecordAsRemoved(scopeFileElements, recordNumber);
  // записуємо scopeFileElements в основну область
  _writeScopeFile(scopeFileElements);

  // отримуємо IndexRecord який потрібно видалити
  IndexRecord indexRecordToDelete = _findIndexRecordByReference(indexFileElements,
recordNumber);
  // ліст з видаленим IndexRecord
  var removedIndexRecordList = _binaryDeleteIndexRecord(indexFileElements,
indexRecordToDelete);
  // записуємо removedIndexRecordList в індексну область
  _writeIndexFile(removedIndexRecordList);
 }

 // змінюємо значення запису
 void _changeRecordValue(List<ScopeRecord> records, int recordNumber, String newValue){
  for (var record in records) {
   if(record.recordNumber == recordNumber){
    record.value = newValue;
   }
  }
 }

 // змінюємо запис
 void editRecord(int recordNumber, String newValue){
  // елементи основної області
  var scopeFileElements = readScopeFile();

  // змінюємо значення запису
  _changeRecordValue(scopeFileElements, recordNumber, newValue);

  // записуємо scopeFileElements в основну область
  _writeScopeFile(scopeFileElements);
```

```
}

// ініціалізуємо базу даних
void initializeDatabase(){
  // файл індексної області
  File indexFile = File(_indexFilePath);
  // файл основної області
  File scopeFile = File(_scopeFilePath);
  // файл області переповнення
  File overflowFile = File(_overflowFilePath);

  // записуємо "шапку" в основну область
  scopeFile.writeAsStringSync("isDeleted : recordNumber : value");
  // записуємо "шапку" в область переповнення
  overflowFile.writeAsStringSync("key : reference");

  // вміст, який потрібно записати
  String indexFileStringToWrite = "key : reference\n";
  // місткість індексної області
  int overallCapacity = _blocksAmount * blockSize;
  // формуємо "пусті" елементи
  for(int i = 0; i < overallCapacity; i += 1){
    IndexRecord record = IndexRecord(0, 0);
    indexFileStringToWrite += record.toString() + '\n';
  }
  // видаляємо останній символ "\n"
  indexFileStringToWrite = indexFileStringToWrite.substring(0,
indexFileStringToWrite.length - 1);
  // записуємо indexFileStringToWrite в індексну область
  indexFile.writeAsStringSync(indexFileStringToWrite);
 }
}
```

3.3.2 Приклади роботи

На рисунках 3.1, 3.2, 3.3, 3.4 показані приклади роботи програми для додавання і пошуку запису.

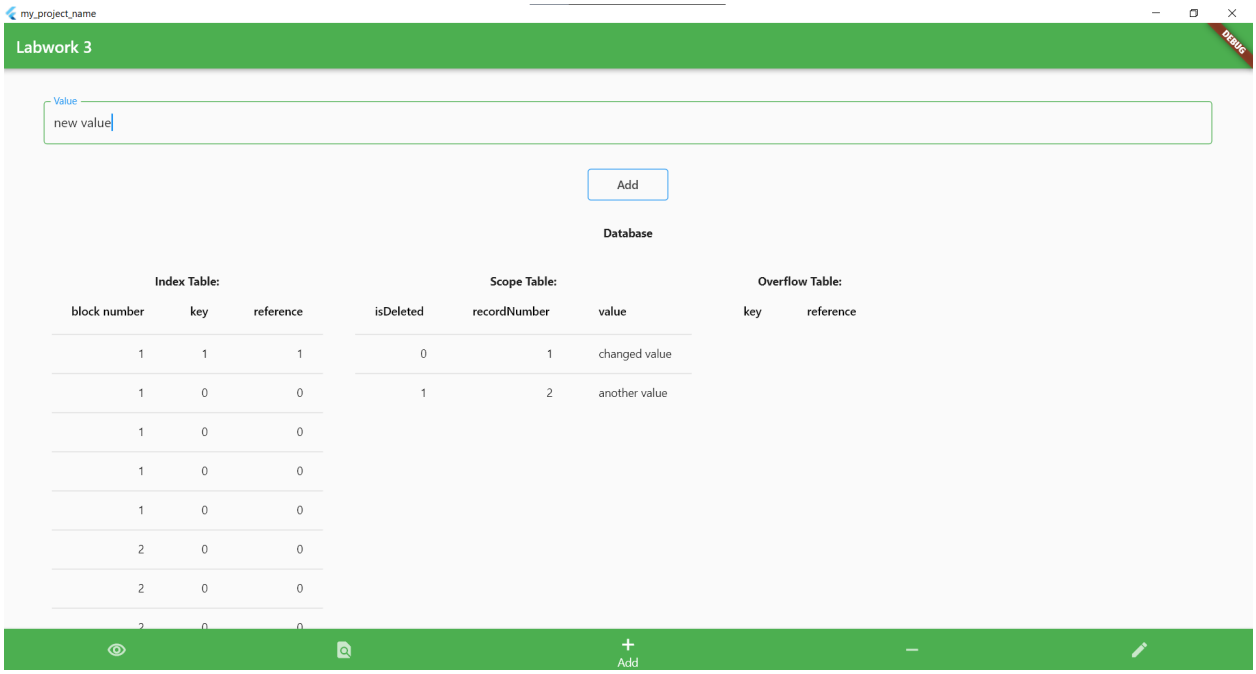


Рисунок 3.1 – База баних до додавання запису

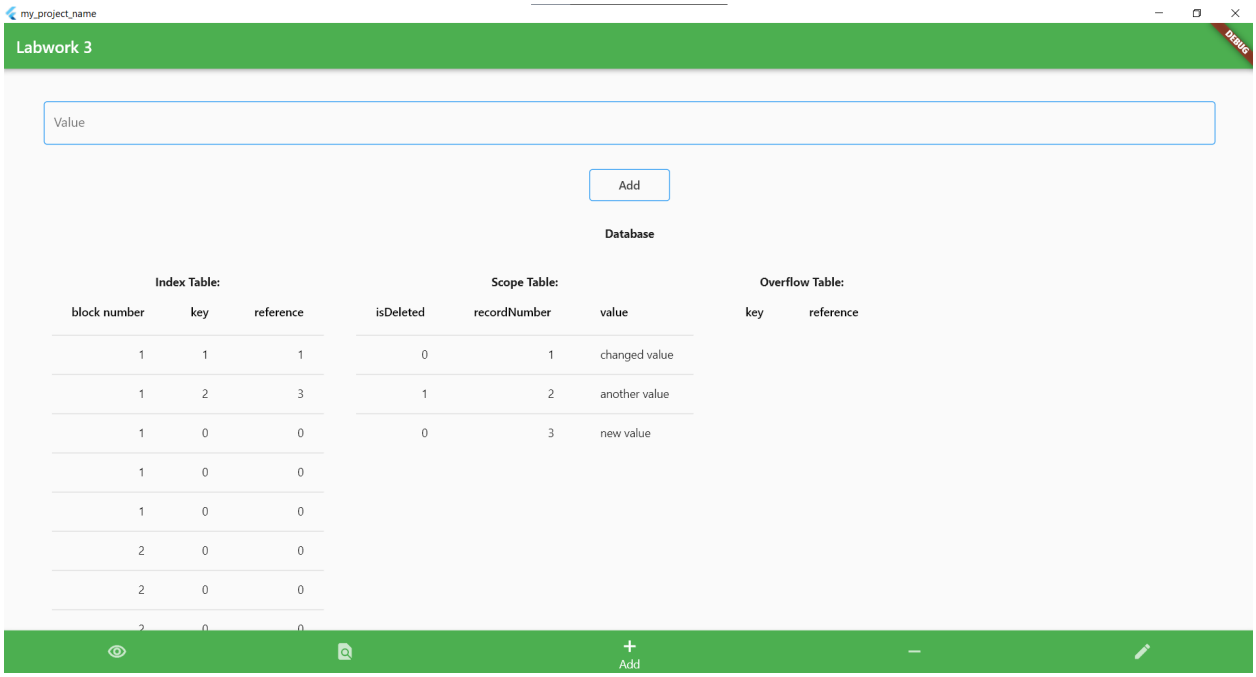


Рисунок 3.2 – База баних після додавання запису

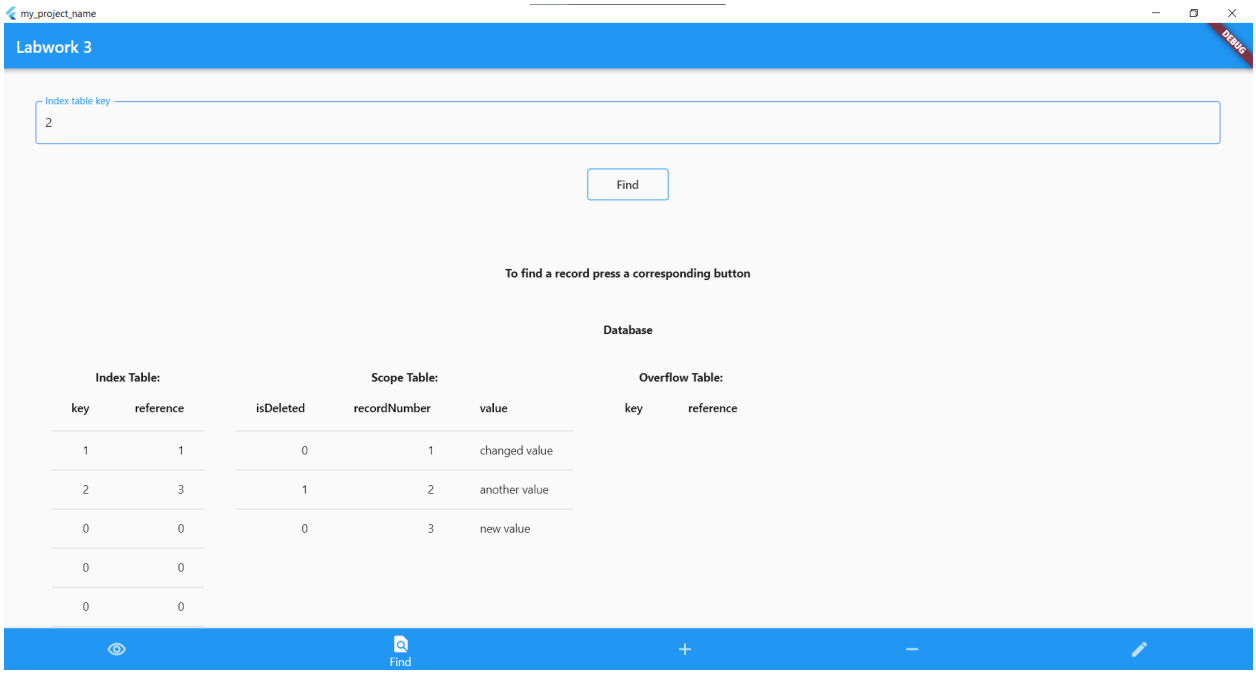


Рисунок 3.3 – База баних до пошуку запису

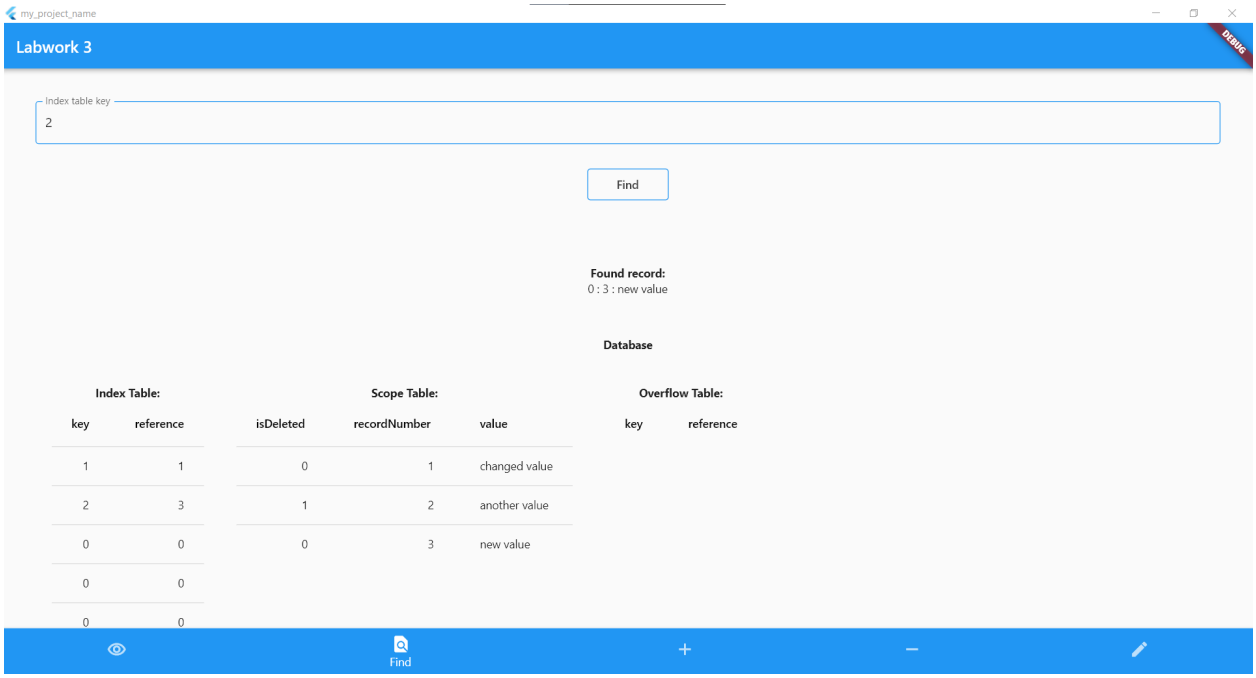


Рисунок 3.4 – База баних після пошуку запису

3.4 Тестування алгоритму

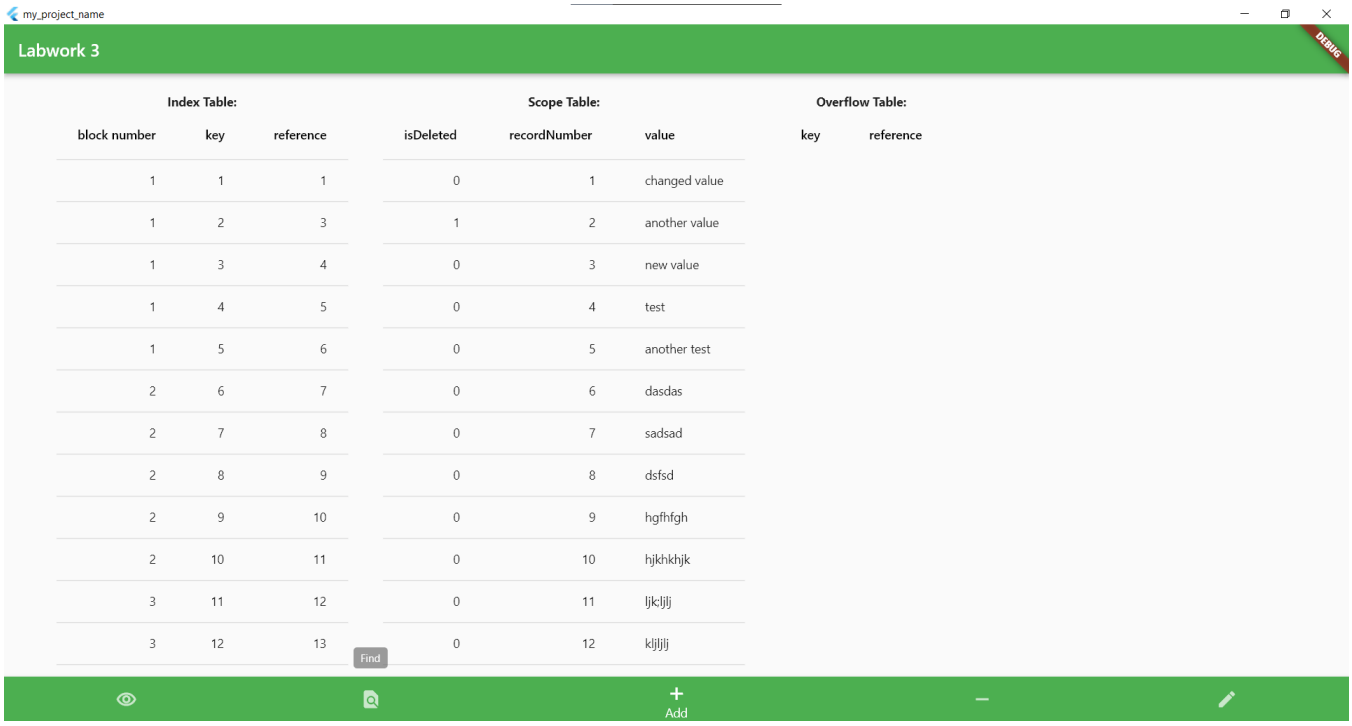
3.4.1 Часові характеристики оцінювання

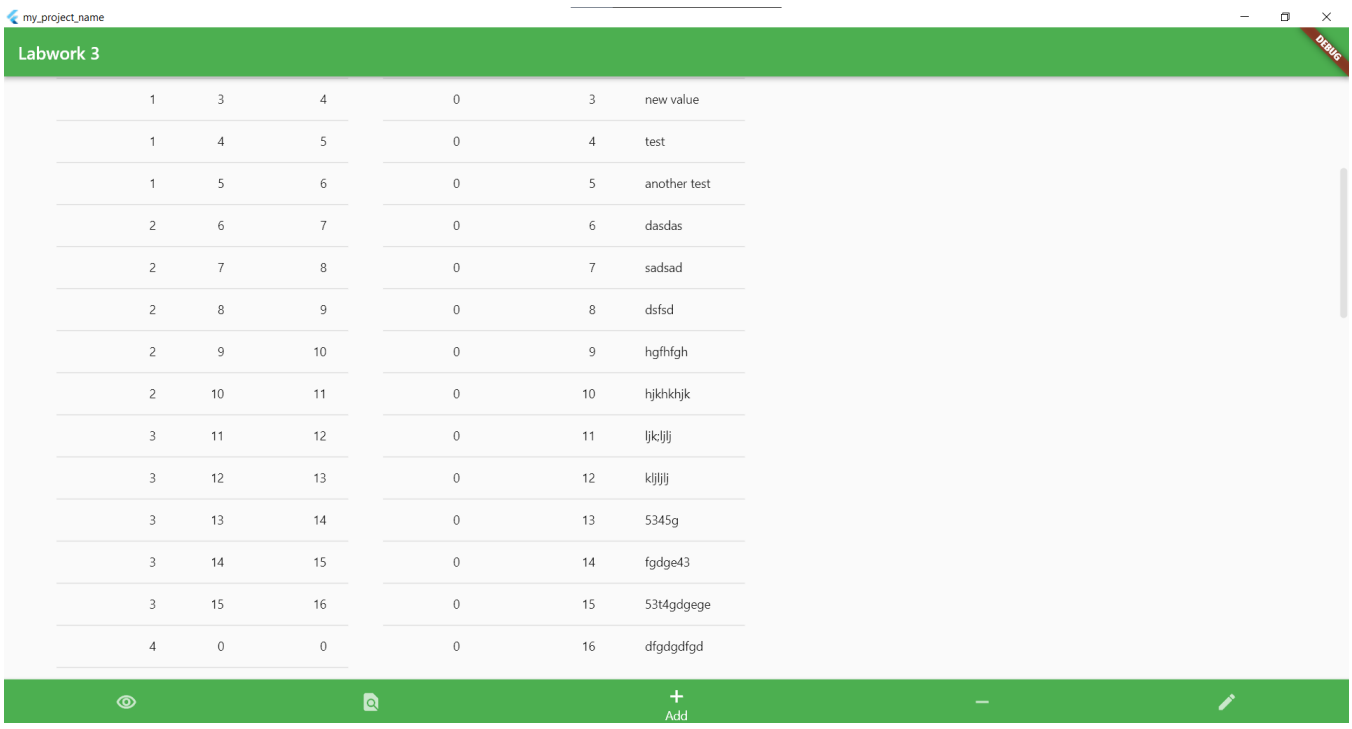
В таблиці 3.1 наведено кількість порівнянь для 15 спроб пошуку запису по ключу.

Таблиця 3.1 – Число порівнянь при спробі пошуку запису по ключу

Номер спроби пошуку	Число порівнянь
1	4
2	3
3	4
4	2
5	4
6	3
7	4
8	1
9	4
10	3
11	4
12	2
13	4
14	3
15	4

Для тестування алгоритму будемо використовувати наступну базу даних:





Для спроби №1 будемо шукати елемент з посиланням 1 (ключ індексної області: 1).
Для спроби №2 будемо шукати елемент з посиланням 3 (ключ індексної області: 2).
Для спроби №3 будемо шукати елемент з посиланням 4 (ключ індексної області: 3).
Для спроби №4 будемо шукати елемент з посиланням 5 (ключ індексної області: 4).
Для спроби №5 будемо шукати елемент з посиланням 6 (ключ індексної області: 5).
Для спроби №6 будемо шукати елемент з посиланням 7 (ключ індексної області: 6).
Для спроби №7 будемо шукати елемент з посиланням 8 (ключ індексної області: 7).
Для спроби №8 будемо шукати елемент з посиланням 9. (ключ індексної області: 8)
Для спроби №9 будемо шукати елемент з посиланням 10 (ключ індексної області: 9.
Для спроби №10 будемо шукати елемент з посиланням 11 (ключ індексної області: 10).
Для спроби №11 будемо шукати елемент з посиланням 12 (ключ індексної області: 11).
Для спроби №12 будемо шукати елемент з посиланням 13 (ключ індексної області: 12).
Для спроби №13 будемо шукати елемент з посиланням 14 (ключ індексної області: 13).
Для спроби №14 будемо шукати елемент з посиланням 15 (ключ індексної області: 14).
Для спроби №15 будемо шукати елемент з посиланням 16 (ключ індексної області: 15).

ВИСНОВОК

В рамках виконання цієї лабораторної роботи я вивчив основні підходи проектування та обробки складних структур даних, а саме файлів зі щільним індексом з областю переповнення, в яких використовувався бінарний пошук. В результаті виконання програми я отримав власну систему управління базою даних, яка може показувати дані в базі, знаходити потрібні елементи, додавати елементи в базу даних, видаляти елементи з бази та змінювати їх.