

**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)**

IT-01 Бардін В. Д.  
(шифр, прізвище, ім'я, по батькові)

**Перевірив**

Камінська П.А.  
(прізвище, ім'я, по батькові)

Київ 2021

## ЗМІСТ

<b>1</b>	<b>МЕТА ЛАБОРАТОРНОЇ РОБОТИ .....</b>	<b>3</b>
<b>2</b>	<b>ЗАВДАННЯ .....</b>	<b>4</b>
<b>3</b>	<b>ВИКОНАННЯ .....</b>	<b>5</b>
3.1	ПСЕВДОКОД АЛГОРИТМІВ.....	5
3.2	ЧАСОВА СКЛАДНІСТЬ ПОШУКУ.....	5
3.3	ПРОГРАМНА РЕАЛІЗАЦІЯ .....	6
3.3.1	<i>Вихідний код .....</i>	<i>6</i>
3.3.2	<i>Приклади роботи .....</i>	<i>13</i>
3.4	ТЕСТУВАННЯ АЛГОРИТМУ .....	14
3.4.1	<i>Часові характеристики оцінювання.....</i>	<i>14</i>
	<b>ВИСНОВОК .....</b>	<b>16</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 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

## 3.1 Псевдокод алгоритмів

```
function binary_search(A, n, T) is
  L := 0
  R := n - 1
  while L ≤ R do
    m := floor((L + R) / 2)
    if A[m] < T then
      L := m + 1
    else if A[m] > T then
      R := m - 1
    else:
      return m
  return unsuccessful
```

## 3.2 Часова складність пошуку

За теоремою Мастерса рекурентне відношення має формулу:

$T(N) = aT\left(\frac{N}{b}\right) + f(N)$ , тоді для бінарного пошуку ця формула матиме

вигляд —  $T(N) = T\left(\frac{N}{2}\right) + O(1)$ , бо  $a = 1, b = 2$ , тоді:  $\log_a b = 1$ . Окрім

цього:  $f(N) = n^c(\log(n))^k$ , де  $k = 0, c = \log_b(a)$ . Отже,  $T(N) =$

$O(n^c(\log n)^{k+1}) = O(\log(n))$ .

### 3.3 Програмна реалізація

```
private Index<TKey> BinarySearch<TKey>(  
    List<Index<TKey>> indexes,  
    TKey value) where TKey : IComparable  
{  
    var lo = 0;  
    var hi = indexes.Count - 1;  
    while (lo <= hi)  
    {  
        var i = lo + ((hi - lo) >> 1);  
        var order = indexes[i].Key.CompareTo(value);  
        switch (order)  
        {  
            case 0:  
                return indexes[i];  
            case < 0:  
                lo = i + 1;  
                break;  
            default:  
                hi = i - 1;  
                break;  
        }  
    }  
    return indexes[~lo];  
}
```

#### 3.3.1 Вихідний код СУБД

##### Core

```
internal readonly struct ConnectionInfo  
{  
    public string FilePath { get; }  
    public FusionAccessMode AccessMode { get; }  
    public ConnectionInfo(string filePath, string accessMode)  
    {  
        FilePath = filePath;  
        AccessMode = Enum.Parse<FusionAccessMode>(accessMode);  
    }  
}  
  
internal class DataPiece<TKey> where TKey : IComparable  
{  
    public TKey Key { get; set; }  
    public byte[] Value { get; set; }  
    public static DataPiece<TKey> Create<TVal>(TKey key, TVal value)  
    {  
        return new DataPiece<TKey>  
        {  
            Key = key,  
            Value = Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(value)),  
        };  
    }  
    public static DataPiece<TKey> ToDataPiece(string str)  
    {  

```

```

        return JsonConvert.DeserializeObject<DataPiece<TKey>>(str);
    }
    public override string ToString()
    {
        return JsonConvert.SerializeObject(this, Formatting.None);
    }
}

internal enum FusionAccessMode
{
    ReadWrite,
    ReadOnly,
}

internal class Index<TKey>
{
    public TKey Key { get; set; }
    public int Payload { get; set; }
    public Index()
    {
    }
    public Index(TKey key, int payload)
    {
        Key = key;
        Payload = payload;
    }
    public static Index<TKey> FromString(string indexStr)
    {
        return JsonConvert.DeserializeObject<Index<TKey>>(indexStr);
    }
    public override string ToString()
    {
        return JsonConvert.SerializeObject(this, Formatting.None);
    }
}

```

## Exceptions

```

public class InvalidConnectionStringException : Exception
{
    public InvalidConnectionStringException(string message)
        : base(message)
    {
    }
}

public class RecordNotFoundException : Exception
{
    public RecordNotFoundException(string message) : base(message)
    {
    }
}

```

## Internal

```

internal class ConnectionStringParser
{
    private static readonly string[] RequiredParts =
    {
        "Db", "Access"
    };
    internal static ConnectionInfo ParseConnectionString(string connectionString)
    {
    }
}

```

```

        if (string.IsNullOrEmpty(connectionString) ||
            string.IsNullOrWhiteSpace(connectionString))
        {
            throw new InvalidConnectionStringException("Connection string can't be
null, empty or whitespace.");
        }
        var connectionParts = connectionString.Split(';');
        foreach (var requiredPart in RequiredParts)
        {
            if (!connectionParts.Any(x => x.Contains(requiredPart)))
            {
                throw new InvalidConnectionStringException(
                    "The given connection string isn't contains a required part: " +
requiredPart);
            }
        }
        var filePath = connectionParts.First(x =>
x.Contains("Db=")).AsSpan()[3..].ToString();
        var accessMode = connectionParts.First(x =>
x.Contains("Access=")).AsSpan()[7..].ToString();
        if (!File.Exists(filePath))
        {
            throw new FileNotFoundException("The specified database file isn't
exists.");
        }
        return new ConnectionInfo(filePath, accessMode);
    }
    internal static string GetFilePath(string connectionString)
    {
        return connectionString.Split(';')
            .First(x => x.Contains("Db="))
            .AsSpan()[3..]
            .ToString();
    }
}

internal class FileProcessor
{
    private readonly ConnectionInfo _connectionInfo;
    private readonly IndexProcessor _indexProcessor;
    internal FileProcessor(ConnectionInfo connectionInfo)
    {
        _connectionInfo = connectionInfo;
        _indexProcessor = new IndexProcessor();
    }
    internal void WriteDataPiece<TKey>(DataPiece<TKey> dataPiece) where TKey :
IComparable
    {
        var lines = ReadAllLines();
        var newDataPieceIndex = _indexProcessor.CreateIndex(dataPiece.Key, lines);
        var indexes = _indexProcessor.GetIndexes<TKey>(lines);
        indexes.Add(newDataPieceIndex);
        indexes = indexes.OrderBy(x => x.Key).ToList();
        var dataSegment = GetDataSegment<TKey>(lines);
        dataSegment.Add(dataPiece);
        RewriteFile(indexes, dataSegment);
    }
    internal DataPiece<TKey> ReadDataPiece<TKey>(TKey key) where TKey : IComparable
    {
        var lines = ReadAllLines();
        var index = _indexProcessor.GetIndexByKey(lines, key);
        return GetDataPieceByLineNumber<TKey>(lines, index.Payload);
    }
}

```



```

    }
    internal DataPiece<TKey> UpdateDataPiece<TKey>(TKey key, DataPiece<TKey>
newValue)
    where TKey : IComparable
    {
        if (key.CompareTo(newValue.Key) != 0)
        {
            throw new InvalidOperationException("Record and key aren't consistent!");
        }
        var lines = ReadAllLines();
        RewriteDataPiece(lines, newValue);
        return newValue;
    }
    internal void DeleteDataPiece<TKey>(TKey key) where TKey : IComparable
    {
        var lines = ReadAllLines();
        var indexes = _indexProcessor.GetIndexes<TKey>(lines);
        var dataSegment = GetDataSegment<TKey>(lines);
        indexes.RemoveAll(x => key.CompareTo(x.Key) == 0);
        dataSegment.RemoveAll(x => key.CompareTo(x.Key) == 0);
        RewriteFile(indexes, dataSegment);
    }
    private List<string> ReadAllLines()
    {
        var lines = new List<string>();
        using var reader = new StreamReader(_connectionInfo.FilePath);
        while (!reader.EndOfStream)
        {
            lines.Add(reader.ReadLine());
        }
        return lines;
    }
    private void RewriteDataPiece<TKey>(
        List<string> lines,
        DataPiece<TKey> newValue) where TKey : IComparable
    {
        var dataSegment = GetDataSegment<TKey>(lines);
        var existedDataPiece = dataSegment
            .FirstOrDefault(x => x.Key.CompareTo(newValue.Key) == 0);
        if (existedDataPiece is null)
        {
            throw new RecordNotFoundException(
                "Record with key " + newValue.Key + " wasn't found!");
        }
        existedDataPiece.Value = newValue.Value;
        var indexes = _indexProcessor.GetIndexes<TKey>(lines);
        RewriteFile(indexes, dataSegment);
    }
    private void RewriteFile<TKey>(
        IEnumerable<Index<TKey>> indexes,
        IEnumerable<DataPiece<TKey>> dataSegment) where TKey : IComparable
    {
        var res = new List<string> { FusionConstants.IndexAreaBeginMarker };
        res.AddRange(indexes.Select(x => x.ToString()));
        res.Add(FusionConstants.IndexAreaEndMarker);
        res.Add(FusionConstants.DataAreaBeginMarker);
        res.AddRange(dataSegment.Select(x => x.ToString()));
        using var writer = new StreamWriter(_connectionInfo.FilePath, append: false);
        foreach (var line in res)
        {
            writer.WriteLine(line);
        }
    }

```

```

        writer.Flush();
    }
    private List<DataPiece<TKey>> GetDataSegment<TKey>(List<string> lines) where TKey
: IComparable
    {
        var dataSegment = new List<DataPiece<TKey>>();
        var isDataSegment = false;
        using var dataIterator = lines.GetEnumerator();
        while (dataIterator.MoveNext())
        {
            var line = dataIterator.Current;
            if (line is FusionConstants.DataAreaBeginMarker) isDataSegment = true;
            if (IsMarkLine(line)) continue;
            if (isDataSegment)
            {
                dataSegment.Add(DataPiece<TKey>.ToDataPiece(line));
            }
        }
        return dataSegment;
    }
    private DataPiece<TKey> GetDataPieceByLineNumber<TKey>(
        List<string> lines,
        int lineNumber) where TKey : IComparable
    {
        using var dataIterator = lines.GetEnumerator();
        var isDataSegment = false;
        while (dataIterator.MoveNext())
        {
            var line = dataIterator.Current;
            if (line is FusionConstants.DataAreaBeginMarker) isDataSegment = true;
            if (IsMarkLine(line)) continue;
            if (!isDataSegment) continue;
            // Iterating should start from 1 not from 0.
            // Because lines at the database starts from 1
            for (var i = 1; i <= lineNumber; i++)
            {
                if (i == lineNumber) return DataPiece<TKey>.ToDataPiece(line);
                if (!dataIterator.MoveNext())
                {
                    throw new RecordNotFoundException(
                        "An error occurred. Record at line " + lineNumber + " wasn't
found.");
                }
                line = dataIterator.Current;
            }
        }
        throw new RecordNotFoundException(
            "An error occurred. Record at line " + lineNumber + " wasn't found.");
    }
    private static bool IsMarkLine(string str) =>
        str is FusionConstants.DataAreaBeginMarker or
        FusionConstants.IndexAreaBeginMarker or
        FusionConstants.IndexAreaEndMarker
        || str.Contains(FusionConstants.IndexAreaEndMarker);
}
internal class IndexProcessor
{
    internal Index<TKey> CreateIndex<TKey>(TKey key, List<string> lines)
    {
        return new Index<TKey>(key, GetRecordsAmount(lines) + 1);
    }
    internal List<Index<TKey>> GetIndexes<TKey>(List<string> lines)

```

```

{
    using var indexesIterator = lines.GetEnumerator();
    var indexes = new List<Index<TKey>>();
    var indexSectionBeginMarkerFound = false;
    while (indexesIterator.MoveNext())
    {
        var line = indexesIterator.Current;
        if (line is FusionConstants.IndexAreaBeginMarker)
        {
            indexSectionBeginMarkerFound = true;
            continue;
        }
        if (line is FusionConstants.IndexAreaEndMarker) break;
        if (indexSectionBeginMarkerFound)
        {
            indexes.Add(Index<TKey>.FromString(line));
        }
    }
    return indexes;
}

internal Index<TKey> GetIndexByKey<TKey>(List<string> lines, TKey key) where TKey
: IComparable
{
    try
    {
        var indexes = GetIndexes<TKey>(lines);
        return BinarySearch(indexes, key);
    }
    catch (ArgumentOutOfRangeException)
    {
        throw new RecordNotFoundException("Record with key " + key + " wasn't
found!");
    }
}

private Index<TKey> BinarySearch<TKey>(
    IReadOnlyList<Index<TKey>> indexes,
    TKey value) where TKey : IComparable
{
    var steps = 0;
    var lo = 0;
    var hi = indexes.Count - 1;
    while (lo <= hi)
    {
        steps++;
        var i = lo + ((hi - lo) >> 1);
        var order = indexes[i].Key.CompareTo(value);
        switch (order)
        {
            case 0:
                Console.WriteLine("Index with key " + value + " found in " +
steps + " steps");
                return indexes[i];
            case < 0:
                lo = i + 1;
                break;
            default:
                hi = i - 1;
                break;
        }
    }
    return indexes[~lo];
}

```

```

private int GetRecordsAmount(List<string> lines)
{
    using var recordsIterator = lines.GetEnumerator();
    var records = 0;
    var dataBlockStarted = false;
    while (recordsIterator.MoveNext())
    {
        var line = recordsIterator.Current;
        if (line is FusionConstants.DataAreaBeginMarker)
        {
            dataBlockStarted = true;
            continue;
        }
        if (dataBlockStarted) records++;
    }
    return records;
}
}

```

## Public Interface

```

/// <summary>
/// <b>This is an outer interface that allows to manipulate with a database file</b>
/// </summary>
public class FusionConnection
{
    // Valid connection string example: "Db=dbfile.fdb;Access=ReadOnly;"
    private readonly FileProcessor _fileProcessor;
    // ReSharper disable once ParameterOnlyUsedForPreconditionCheck.Local
    public FusionConnection(string connectionString, bool createIfNotExists = false)
    {
        ConnectionInfo connectionInfo;
        try
        {
            connectionInfo =
                ConnectionStringParser.ParseConnectionString(connectionString);
        }
        catch (FileNotFoundException)
        {
            if (!createIfNotExists) throw;
        }

        File.Create(ConnectionStringParser.GetFilePath(connectionString)).Dispose();
        connectionInfo =
            ConnectionStringParser.ParseConnectionString(connectionString);
        _fileProcessor = new FileProcessor(connectionInfo);
    }
    public Record<TKey, TVal> ReadRecord<TKey, TVal>(TKey key) where TKey :
    IComparable
    {
        var dataPiece = _fileProcessor.ReadDataPiece(key);
        return Record<TKey, TVal>.ToRecord(dataPiece);
    }
    public TVal Read<TKey, TVal>(TKey key) where TKey : IComparable
    {
        var dataPiece = _fileProcessor.ReadDataPiece(key);
        return Record<TKey, TVal>.ToRecord(dataPiece).Value;
    }
    public Record<TKey, TVal> Write<TKey, TVal>(TKey key, TVal value) where TKey :
    IComparable
    {
        _fileProcessor.WriteDataPiece(DataPiece<TKey>.Create(key, value));
        return new Record<TKey, TVal>(key, value);
    }
}

```

```

    }
    public void Update<TKey, TVal>(TKey key, TVal newValue) where TKey : IComparable
    {
        var updatedDataPiece = DataPiece<TKey>.Create(key, newValue);
        _fileProcessor.UpdateDataPiece(updatedDataPiece.Key, updatedDataPiece);
    }
    public void Delete<TKey>(TKey key) where TKey : IComparable
    {
        _fileProcessor.DeleteDataPiece(key);
    }
}


public static class FusionConstants
{
    public const string IndexAreaBeginMarker = "//start-indexes";
    public const string IndexAreaEndMarker = "//end-indexes";
    public const string DataAreaBeginMarker = "//data-area";
}

public class Record<TKey, TVal> where TKey : IComparable
{
    public TKey Key { get; set; }
    public TVal Value { get; set; }
    public Record(TKey key, TVal val)
    {
        Key = key;
        Value = val;
    }
    internal static Record<TKey, TVal> ToRecord(DataPiece<TKey> dataPiece)
    {
        return new Record<TKey, TVal>(
            dataPiece.Key,
            JsonConvert.DeserializeObject<TVal>(Encoding.UTF8.GetString(dataPiece.Value)));
    }
}

```

### 3.3.2 Приклади роботи

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



```

dbfile.fdb
C: > Users > User > Desktop > KPI > AD > lab2 > FusionDms > FusionDms.ConsoleClient > bin > Debug >
1 //start-indexes
2 {"Key":1,"Payload":1}
3 {"Key":2,"Payload":2}
4 {"Key":3,"Payload":3}
5 //end-indexes
6 //data-area
7 {"Key":1,"Value":"IjBhZDI4YzMzLTU0ODYtNDIxNS04Nzc5LWVhMDcxYTA4Mzk4YyI="}
8 {"Key":2,"Value":"IjI0OGM4MzQ5LWMyYTktNDFiMi04NTZmLTEzZGYxNWJjNDM4YyI="}
9 {"Key":3,"Value":"ImRhZWU5OWZhLWZmY2UtNGY5My1iNWU0LWI2MjU0ZjRkMzExMCI="}
10

```

Рисунок 3.1 –Додавання запису

```
READ:
Index with key 1 found in 2 steps
Value for record with id 1: (0ad28c33-5486-4215-8779-ea071a08398c)
Index with key 2 found in 1 steps
Value for record with id 2: (248c8349-c2a9-41b2-856f-13df15bc438c)
Index with key 3 found in 2 steps
Value for record with id 3: (daee99fa-ffce-4f93-b5a4-b6254f4d3110)
```

Рисунок 3.2 – Пошук запису

### 3.4 Тестування алгоритму

Для тестування алгоритму було виконано запуск програми, при якому було створено БД та заповнено її 100 записами виду: {int; Guid}. А потім проведено пошук 15 значень за їх ключами з проміжку [0; 100].

#### 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	6
2	11	7
3	21	5
4	31	4
5	41	6
6	51	6
7	61	7
8	71	5
9	81	4
10	91	5
11	50	1

12	75	2
13	90	7
14	25	2
15	15	5

## ВИСНОВОК

В рамках лабораторної роботи було розроблено міні СУБД, яка дозволяє виконувати операції: запису, пошуку, оновлення та видалення елементів. Також, я розібрався як працюють файли з щільним індексом, та як за допомогою алгоритму бінарного пошуку можна ефективно шукати запис за ключем за допомогою індексів. А також дізнався про певні обмеження ОС Windows при роботі з файлами. Як виявилось, під капотом ця ОС, не може: частково перезаписувати файл, гарантувати асинхронної роботи з файлами, навіть якщо у функцію WinApi передати прапорець, який відповідає, за те, щоб функція працювала асинхронно.