**нМіністерство 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)**

(шифр, прізвище, ім'я, по батькові)

*ІТ-04 Гавриленко Ян Сергійович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2021

Зміст

[1 Мета лабораторної роботи 3](#_Toc81070524)

[2 Завдання 4](#_Toc81070525)

[3 Виконання 7](#_Toc81070526)

[3.1 Псевдокод алгоритмів 7](#_Toc81070527)

[3.2 Часова складність пошуку 7](#_Toc81070528)

[3.3 Програмна реалізація 7](#_Toc81070529)

[3.3.1 Вихідний код 7](#_Toc81070530)

[3.3.2 Приклади роботи 7](#_Toc81070531)

[3.4 Тестування алгоритму 8](#_Toc81070532)

[3.4.1 Часові характеристики оцінювання 8](#_Toc81070533)

[Висновок 9](#_Toc81070534)

[Критерії оцінювання 10](#_Toc81070535)

# Мета лабораторної роботи

Мета роботи – вивчити основні підходи проектування та обробки складних структур даних.

# 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 |
| 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, бінарний пошук |
| 18 | B-дерево t=25, бінарний пошук |
| 19 | B-дерево t=50, бінарний пошук |
| 20 | B-дерево t=100, 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 |

# Виконання

## Псевдокод алгоритмів

Func addLine (data):

Key = generate key

findIndexStart()

[] index = readIndex()

Index = bin\_search(index, key)

write(index, key)

findMainStart()

count = read()

write(count, key)

write(count, data)

Func delLine(data, key)

findIndexStart()

[] index = readIndex()

Index = bin\_search(index, key)

findMainStart()

index2 = Index.lineNumber

del(index2)

Del(Index)

Func getLine(key)

findIndexStart()

[] index = readIndex()

Index = bin\_search(index, key)

findMainStart()

index2 = Index.lineNumber

return read(index2)

## Часова складність пошуку

…

## Програмна реалізація

### Вихідний код

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace laba2

{

class Program

{

static void Main(string[] args)

{

FileSettings sts = new FileSettings()

{

LZ = 2,

LK = 3,

KZ = 100,

LB = 41

};

using (LABFile fl = new LABFile("labfile.lab", sts))

{

//AddDataAndReadAllExample(fl);

//ReadAllDataExample(fl);

//GetOneLineExample(fl);

//AddDataReadAndDeleteExample(fl);

}

}

private static void AddDataAndReadAllExample(LABFile fl)

{

int cnt = 0;

while (true)

{

if (!fl.AddLine(new Line(RandomString(2)))) break;

cnt++;

}

Console.WriteLine($"Added {cnt} random values to the base. Output:");

ReadAllDataExample(fl);

}

private static void AddDataReadAndDeleteExample(LABFile fl)

{

var line = new Line("PA");

var line1 = new Line("AP");

fl.AddLine(line);

fl.AddLine(line1);

ReadAllDataExample(fl);

fl.DeleteLine(line.Key);

ReadAllDataExample(fl);

Console.WriteLine();

Console.WriteLine(fl.ReadRawData());

}

private static void ReadAllDataExample(LABFile fl)

{

Console.WriteLine("Readling data from file:");

foreach (var item in fl.GetAllLines())

{

Console.WriteLine(item);

}

}

private static void GetOneLineExample(LABFile fl)

{

var line = new Line("PA");

var line1 = new Line("PP");

var line2 = new Line("AA");

fl.AddLine(line);

fl.AddLine(line1);

fl.AddLine(line2);

Console.WriteLine(fl.GetLine(line1.Key));

}

private static Random random = new Random();

public static string RandomString(int length)

{

const string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

return new string(Enumerable.Repeat(chars, length)

.Select(s => s[random.Next(s.Length)]).ToArray());

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

namespace laba2

{

internal class LABFile : IDisposable

{

public LABFile()

{

}

public LABFile(string path, FileSettings settings)

{

meta = new byte[4];

meta[0] = settings.LZ;

meta[1] = settings.LK;

meta[2] = settings.KZ;

meta[3] = settings.LB;

if (File.Exists(path))

{

byte[] metabt = new byte[24];

\_mainStr = File.Open(path, FileMode.Open, FileAccess.ReadWrite);

\_mainStr.Seek(-24, SeekOrigin.End);

\_mainStr.Read(metabt);

indexStart = BitConverter.ToInt64(metabt[0..8]);

overflowStart = BitConverter.ToInt64(metabt[8..16]);

mainStart = BitConverter.ToInt64(metabt[16..24]);

}

else

{

\_mainStr = File.Open(path, FileMode.OpenOrCreate, FileAccess.ReadWrite);

\_mainStr.Write(createStructure());

}

indexStart = 4;

}

FileStream \_mainStr;

byte[] meta;

long indexStart,

overflowStart,

mainStart;

byte[] createStructure()

{

List<byte> fin = new List<byte>();

byte[] bts = new byte[1] { 0 };

byte indexLineLength = (byte)(meta[1] + 1);

\_mainStr.Seek(0, SeekOrigin.Begin);

fin.AddRange(meta);

indexStart = fin.Count;

for (int i = 0; i < meta[3] / indexLineLength; i++)

{

fin.AddRange(bts);

fin.AddRange(new byte[40]);

}

fin.AddRange(bts);

overflowStart = fin.Count;

fin.AddRange(new byte[80]);

mainStart = fin.Count;

fin.AddRange(new byte[500]);

fin.AddRange(Encoding.UTF8.GetBytes("$$$"));

fin.AddRange(BitConverter.GetBytes(indexStart));

fin.AddRange(BitConverter.GetBytes(overflowStart));

fin.AddRange(BitConverter.GetBytes(mainStart));

return fin.ToArray();

}

public Line GetLine(byte[] key)

{

var EA = getBlockNumberAddressByKey(key);

\_mainStr.Seek(EA, SeekOrigin.Begin);

byte blockCount = (byte)\_mainStr.ReadByte();

var lines = new List<IndexLine>();

byte[] crntbytes = new byte[4];

for (int i = 0; i < blockCount; i++)

{

\_mainStr.Read(crntbytes, 0, 4);

lines.Add(new IndexLine(crntbytes));

}

var index = lines

.ToList()

.BinarySearch

(new IndexLine(key.Concat(new byte[] { 255 })

.ToArray()), new IndexLineComparer());

var indexLine = lines[index];

\_mainStr.Seek(mainStart, SeekOrigin.Begin);

var linesCount = \_mainStr.ReadByte();

if (indexLine.LineNum + 1 > linesCount)

throw new Exception("Seeking for line number, that exceeds the file maximum capacity");

\_mainStr.Seek(indexLine.LineNum \* 5, SeekOrigin.Current);

var mainBts = new byte[5];

\_mainStr.Read(mainBts);

return new Line(mainBts);

}

public IEnumerable<Line> GetAllLines()

{

\_mainStr.Seek(mainStart, SeekOrigin.Begin);

var lineCount = \_mainStr.ReadByte();

byte[] crnt = new byte[5];

for (int i = 0; i < lineCount; i++)

{

\_mainStr.Read(crnt);

if(crnt[1] == 0)

{

i--;

continue;

}

yield return new Line(crnt);

}

}

public bool AddLine(Line value)

{

var indexLine = new IndexLine(value.Key, value.cntCur);

return InsertIndex(indexLine) && InsertMain(value);

}

public bool DeleteLine(byte[] key)

{

var mainIndex = DeleteIndex(key);

DeleteMain(mainIndex);

return true;

}

public string ReadRawData()

{

byte[] bts = new byte[\_mainStr.Length];

\_mainStr.Read(bts);

string result = "";

foreach (var item in bts)

{

result += item.ToString();

result += " ";

}

return result;

}

public void Dispose()

{

\_mainStr.Close();

\_mainStr.Dispose();

}

#region private

private bool addToOverflow(IndexLine indexLine)

{

\_mainStr.Seek(overflowStart, SeekOrigin.Begin);

var cnt = \_mainStr.ReadByte();

if (cnt > 80 / (meta[3] + 1))

return false;

\_mainStr.Seek(overflowStart + cnt \* (meta[3] + 1), SeekOrigin.Begin);

\_mainStr.Write(indexLine.ToBytes());

return true;

}

private bool InsertMain(Line value)

{

\_mainStr.Seek(mainStart, SeekOrigin.Begin);

var lineCount = \_mainStr.ReadByte();

if (lineCount > meta[2])

return false;

var absPosition = mainStart + 1 + lineCount \* (meta[1] + meta[0]);

\_mainStr.Seek(absPosition, SeekOrigin.Begin);

\_mainStr.Write(value.ToBytes());

\_mainStr.Seek(mainStart, SeekOrigin.Begin);

\_mainStr.WriteByte((byte)(++lineCount));

return true;

}

private bool DeleteMain(int lineNumber)

{

\_mainStr.Seek(mainStart, SeekOrigin.Begin);

var lineCount = \_mainStr.ReadByte();

\_mainStr.Seek(lineNumber \* 5, SeekOrigin.Current);

\_mainStr.Write(new byte[5], 0, 5);

\_mainStr.Seek(mainStart, SeekOrigin.Begin);

\_mainStr.WriteByte((byte)(--lineCount));

return true;

}

private bool InsertIndex(IndexLine indexLine)

{

long EA = getBlockNumberAddressByKey(indexLine.Key);

//should be on the count\* byte

\_mainStr.Seek(EA, SeekOrigin.Begin);

byte blockCount = (byte)\_mainStr.ReadByte();

if (blockCount >= meta[3] / (meta[1] + 1))

{

if (addToOverflow(indexLine)) return true;

return false;

}

var lines = new List<IndexLine>();

byte[] crntbytes = new byte[4];

for (int i = 0; i < blockCount; i++)

{

\_mainStr.Read(crntbytes, 0, 4);

lines.Add(new IndexLine(crntbytes));

}

lines = insertIndex(lines.ToList(), indexLine);

\_mainStr.Seek(EA + 1, SeekOrigin.Begin);

\_mainStr.Write(new byte[meta[3]]);

\_mainStr.Seek(EA + 1, SeekOrigin.Begin);

\_mainStr.Write(lines

.Select(n => n.ToBytes())

.Aggregate((a, b) => a.Concat(b).ToArray())

.ToArray());

\_mainStr.Seek(EA, SeekOrigin.Begin);

\_mainStr.WriteByte(++blockCount);

return true;

}

private int DeleteIndex(byte[] key)

{

long EA = getBlockNumberAddressByKey(key);

\_mainStr.Seek(EA,SeekOrigin.Begin);

byte blockcount = (byte)\_mainStr.ReadByte();

var lines = new List<IndexLine>();

byte[] crntbytes = new byte[4];

for (int i = 0; i < blockcount; i++)

{

\_mainStr.Read(crntbytes, 0, 4);

lines.Add(new IndexLine(crntbytes));

}

var index = lines

.ToList()

.BinarySearch

(new IndexLine(key.Concat(new byte[] { 255 })

.ToArray()), new IndexLineComparer());

if (index < 0) throw new ArgumentException("Such line couldn't be found");

var deletedLineNumber = lines[index].LineNum;

lines.RemoveAt(index);

\_mainStr.Seek(EA + 1, SeekOrigin.Begin);

\_mainStr.Write(new byte[meta[3]]);

\_mainStr.Seek(EA + 1, SeekOrigin.Begin);

if(lines.Count != 0)

\_mainStr.Write(lines

.Select(n => n.ToBytes())

.Aggregate((a, b) => a.Concat(b).ToArray())

.ToArray());

\_mainStr.Seek(EA, SeekOrigin.Begin);

\_mainStr.WriteByte(--blockcount);

return deletedLineNumber;

}

private List<IndexLine> insertIndex(List<IndexLine> lines, IndexLine indexLine)

{

var index = lines.BinarySearch(indexLine, new IndexLineComparer());

if (index < 0)

lines.Insert(~index , indexLine);

else

lines.Insert(index, indexLine);

return lines;

}

private long getBlockNumberAddressByKey(byte[] key)

{

var iLinesInBlock = meta[3] / (meta[1] + 1);

var blockNumber = key[0] % iLinesInBlock;

return indexStart + blockNumber \* meta[3];

}

#endregion

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace laba2

{

struct FileSettings

{

public byte LZ, LK, KZ, LB;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace laba2

{

class Line

{

static int cnt = 0;

public int cntCur;

public Line()

{

cntCur = Line.cnt++;

}

public Line(string word)

{

if (word.Length > 2)

throw new ArgumentException("Lines must be 2 bytes long");

cntCur = Line.cnt++;

Data = Encoding.UTF8.GetBytes(word);

Key = keyGen();

}

public Line(byte[] bts)

{

if (bts.Length != 5)

throw new ArgumentException("Wrong number of bytes passed to Line constructor");

cntCur = Line.cnt++;

data = bts[3..5];

key = bts[0..3];

}

byte[] key;

byte[] data;

public byte[] Data {

get {

return data;

}

private set

{

if(value.Length > 2)

throw new ArgumentException("Wrong number of bytes passed to data setter");

data = value;

}

}

public byte[] Key

{

get

{

return key;

}

private set

{

if (value.Length != 3)

throw new ArgumentException("Wrong number of bytes passed to key setter");

key = value;

}

}

private byte[] keyGen()

{

Random rand = new Random();

byte[] key = new byte[3];

key[0] = (byte)rand.Next(0, 9);

key[1] = Encoding.UTF8.GetBytes("-")[0];

key[2] = (byte)rand.Next(1, 257);

return key;

}

public override string ToString()

{

return $"{BitConverter.ToString(Key)} {BitConverter.ToString(Data)} |";

}

public byte[] ToBytes()

{

return key.Concat(data).ToArray();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace laba2

{

class IndexLine

{

public IndexLine(byte[] key, int lineNum)

{

Key = key;

\_lineNum = lineNum;

}

public IndexLine(byte[] bts)

{

if (bts.Length != 4)

throw new ArgumentException("Wrong number of bytes passed to IndexLine constructor");

\_lineNum = bts[3];

key = bts[0..3];

}

byte[] key;

int \_lineNum;

public byte[] Key

{

get

{

return key;

}

private set

{

if (value.Length != 3)

throw new ArgumentException("Wrong number of bytes passed to key setter");

key = value;

}

}

public int LineNum { get { return \_lineNum; } private set { \_lineNum = value; } }

public byte[] ToBytes()

{

return key.ToList().Concat(BitConverter.GetBytes(\_lineNum)).ToArray();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace laba2

{

class IndexLineComparer : IComparer<IndexLine>

{

public int Compare(IndexLine x, IndexLine y)

{

return x.Key[2] - y.Key[2];

}

}

}

### Приклади роботи

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

Рисунок 3.1 –Додавання запису

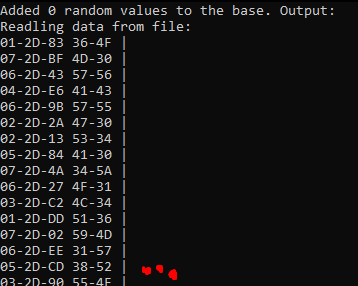


Рисунок 3.2 – Пошук запису



## Тестування алгоритму

### 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 | 5 |
| 3 | 6 |
| 4 | 3 |
| 5 | 4 |
| 6 | 4 |
| 7 | 5 |
| 8 | 3 |
| 9 | 1 |
| 10 | 4 |
| 11 | 1 |
| 12 | 3 |
| 13 | 2 |
| 14 | 1 |
| 15 | 1 |

Висновок

В рамках лабораторної роботи ми вивчили особливості роботи індексного файлу з щільним індексом, вирішення переповнення за допомогою області переповнення, реалізували цю системо програмно та протестували дієздатність на прикладі невеликого обсягу даних

Github repository : https://github.com/yan14171/APLAB2.git

Критерії оцінювання

За умови здачі лабораторної роботи до 15.10.2021 включно максимальний бал дорівнює – 5. Після 15.10.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 20%;
* аналіз часової складності – 5%;
* програмна реалізація алгоритму – 60%;
* тестування алгоритму – 10%;
* висновок – 5%.

+1 додатковий бал можна отримати за реалізацію графічного інтерфейсу.

+1 додатковий бал можна отримати за реалізацію графічного зображення структури ключів.