**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)**

(шифр, прізвище, ім'я, по батькові)

*ІП-21 Загребельний О. А.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2023

Зміст

[1 Мета лабораторної роботи 3](#_Toc114359761)

[2 Завдання 4](#_Toc114359762)

[3 Виконання 7](#_Toc114359763)

[3.1 Псевдокод алгоритмів 7](#_Toc114359764)

[3.2 Часова складність пошуку 10](#_Toc114359765)

[3.3 Програмна реалізація 11](#_Toc114359766)

[3.3.1 Вихідний код 11](#_Toc114359767)

[3.3.2 Приклади роботи 17](#_Toc114359768)

[3.4 Тестування алгоритму 19](#_Toc114359769)

[3.4.1 Часові характеристики оцінювання 19](#_Toc114359770)

[Висновок 20](#_Toc114359771)

[Критерії оцінювання 21](#_Toc114359772)

# 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 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 |

# Виконання

**Варіант 12.** Файли з щільним індексом з областю переповнення, однорідний бінарний пошук.

## Псевдокод алгоритмів

**Homogeneus binary search**

function Search(key\_to\_search, index = int.MinValue, delta = 1):

if index is not equal to int.MinValue:

data = new string[2]

if index > size:

Set key to the maximum value

else if index < 1:

Set key to the minimum value

else:

Get the key and pointer from the index file line of index number

if key is equal to key\_to\_search:

return pointer

if delta is equal to 0:

return -1 (value not found)

delta = delta / 2

if key < key\_to\_search:

index = index + delta + 1

return Search(key, index, delta)

else:

index = index - delta - 1

return Search(key, index, delta)

else:

delta = size / 2

index = delta + 1

return Search(key, index, delta)

**Add**

function Add(key, value, index = int.MinValue, delta = 1):

if index is not equal to int.MinValue:

data = new string[2]

if index > size:

Set key to the maximum value

else if index < 1:

Set key to the minimum value

else:

Get the key and pointer from the index file line of index number

if key is equal to key\_to\_search:

// Duplicate key, handle accordingly (e.g., throw an error or update the existing value)

// In this example, we're assuming no duplicate keys are allowed

return pointer

if delta is equal to 0:

// Reached the leaf node, add the new key-value pair

pointer = size + 1 // Incremental pointer assignment

Append key-value pair to the end of the index file

Append key-pointer pair to the end of the overflow area

Append key-value pair to the end of the database

return pointer

delta = delta / 2

if key < key\_to\_search:

index = index + delta + 1

return Add(key, value, index, delta)

else:

index = index - delta - 1

return Add(key, value, index, delta)

else:

// Initialize the process for adding a new key-value pair

delta = size / 2

index = delta + 1

return Add(key, value, index, delta)

**Delete**

function Delete(key\_to\_delete, index = int.MinValue, delta = 1):

if index is not equal to int.MinValue:

data = new string[2]

if index > size:

Set key to the maximum value

else if index < 1:

Set key to the minimum value

else:

Get the key and pointer from the index file line of index number

if key is equal to key\_to\_delete:

// Found the key to delete

// Implement deletion logic here

// For example, mark the entry as deleted in the index file

return pointer

if delta is equal to 0:

return -1 (value not found)

delta = delta / 2

if key < key\_to\_delete:

index = index + delta + 1

return Delete(key\_to\_delete, index, delta)

else:

index = index - delta - 1

return Delete(key\_to\_delete, index, delta)

else:

// Key not found

return -1

**Edit**

function Edit(key\_to\_edit, new\_value, index = int.MinValue, delta = 1):

if index is not equal to int.MinValue:

data = new string[2]

if index > size:

Set key to the maximum value

else if index < 1:

Set key to the minimum value

else:

Get the key and pointer from the index file line of index number

if key is equal to key\_to\_edit:

// Found the key to edit

// Implement editing logic here

// For example, update the value in the index file

return pointer

if delta is equal to 0:

return -1 (value not found)

delta = delta / 2

if key < key\_to\_edit:

index = index + delta + 1

return Edit(key\_to\_edit, new\_value, index, delta)

else:

index = index - delta - 1

return Edit(key\_to\_edit, new\_value, index, delta)

else:

// Key not found

return -1

## Часова складність пошуку

## Найбільш ефективним алгоритмом пошуку на впорядкованому масиві є логарифмічний, або бінарний, пошук. Максимальна кількість кроків пошуку визначається двійковим логарифмом від загального числа елементів в шуканому просторі пошуку:

Tn = log2N, де N — число елементів.

Однак в нашому випадку є суттєвим тільки число звернень до диска при пошуку запису по заданому значенню первинного ключа. Пошук відбувається в індексній області, де застосовується двійковий алгоритм пошуку індексного запису, а потім шляхом прямої адресації ми звертаємося до основної області вже по конкретному номеру запису. Для того щоб оцінити максимальний час доступу, нам треба визначити кількість звернень до диска для пошуку довільного запису.

На диску записи файлів зберігаються в блоках. Розмір блоку визначається фізичними особливостями дискового контролера і операційною системою. В одному блоці можуть розміщуватися декілька записів. Тому нам треба визначити кількість індексних блоків, який буде потрібно для розміщення всіх необхідних індексних записів, а тому максимальне число звернень до диска дорівнюватиме двійковому логарифму від заданого числа блоків плюс одиниця.

Навіщо потрібна одиниця? Після пошуку номера запису в індексному області ми повинні ще звернутися до основної області файлу. Тому формула для обчислення максимального часу доступу в кількості звернень до диска виглядає наступним чином:

Tn = log2Nбл.інд. + 1.

## Програмна реалізація

### Вихідний код

**Search**

FileManager.cs

public string Search(int key)

{

comparisons = 0;

int row = index.Search(key);

comparisons += index.comparisons;

if (row == -1)

{

row = area.Search(key);

comparisons += area.comparisons;

}

return row == -1 ? null : File.ReadLines(db.filePath).Skip(row - 1).Take(1).First().Split(' ')[1];

}

Index.cs

public int Search(int key, int index = int.MinValue, int delta = 1)

{

if (index != int.MinValue)

{

string[] data = new string[2];

if (index > size)

{

data[0] = int.MaxValue.ToString();

}

else if (index < 1)

{

data[0] = int.MinValue.ToString();

}

else

{

string line = File.ReadLines(filePath).Skip(index - 1).Take(1).First();

data = line.Split(' ');

}

comparisons++;

if (int.Parse(data[0]) == key)

{

return int.Parse(data[1]);

}

if (delta == 0)

{

return -1;

}

delta /= 2;

if (int.Parse(data[0]) < key)

{

index += delta + 1;

return Search(key, index, delta);

}

else

{

index -= delta + 1;

return Search(key, index, delta);

}

}

else

{

comparisons = 0;

delta = size / 2;

index = delta + 1;

return Search(key, index, delta);

}

}

OverflowArea.cs

public int Search(int key)

{

comparisons = 0;

StreamReader reader = new StreamReader(filePath);

while (!reader.EndOfStream)

{

string[] data = reader.ReadLine().Split(' ');

comparisons++;

if (int.Parse(data[0]) == key)

return int.Parse(data[1]);

}

reader.Close();

return -1;

}

**Add**

FileManager.cs

public void Add(string value)

{

int key = db.valuesNum + 1;

if (index.isFull())

{

key = area.Add(key);

}

else

{

key = index.Add(key, db.valuesNum + 1);

}

db.AddRecord(key, value);

}

Index.cs

public int Add(int key, int db\_size, int index = int.MinValue, int delta = 1)

{

if (index == int.MinValue)

{

delta = size / 2;

if (deletedIndexes.set.Count != 0)

{

key = deletedIndexes.set.First();

deletedIndexes.set.Remove(key);

}

index = delta + 1;

return Add(key, db\_size, index, delta);

}

else

{

int current;

int next;

if (index > size)

{

current = int.MaxValue;

}

else if (index < 1)

{

current = int.MinValue;

}

else

{

string line = File.ReadLines(filePath).Skip(index - 1).Take(1).First();

current = int.Parse(line.Split(' ')[0]);

}

if (index + 1 > size)

{

next = int.MaxValue;

}

else if (index + 1 < 1)

{

next = int.MinValue;

}

else

{

string line = File.ReadLines(filePath).Skip(index).Take(1).First();

next = int.Parse(line.Split(' ')[0]);

}

if (current < key && next > key)

{

using (StreamReader sr = new StreamReader(filePath))

using (StreamWriter sw = new StreamWriter("temp\_index.txt"))

{

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

{

sw.WriteLine(sr.ReadLine());

}

sw.WriteLine($"{key} {db\_size}");

while (!sr.EndOfStream)

{

sw.WriteLine(sr.ReadLine());

}

}

size++;

File.Delete(filePath);

File.Move("temp\_index.txt", filePath);

return key;

}

if (current < key)

{

delta /= 2;

index += delta + 1;

return Add(key, db\_size, index, delta);

}

else

{

index -= delta + 1;

return Add(key, db\_size, index, delta);

}

}

}

OverflowArea.cs

public int Add(int key)

{

int db\_size = key;

if (deletedIndexes.set.Count != 0)

{

key = deletedIndexes.set.First();

deletedIndexes.set.Remove(key);

}

File.AppendAllText(filePath, $"{key} {db\_size}\n");

size++;

return key;

}

Database.cs

public void AddRecord(int key, string value)

{

File.AppendAllText(filePath, $"{key} {value}\n");

size++;

valuesNum++;

}

**Delete**

FileManager.cs

public bool Delete(int key)

{

int row = index.Delete(key);

if (row == -1)

{

row = area.Delete(key);

if (row == -1)

{

return false;

}

db.Delete(row);

return true;

}

else

{

db.Delete(row);

return true;

}

}

Index.cs

public int Delete(int key, int index = int.MinValue, int delta = 1)

{

if (index != int.MinValue)

{

string[] data = new string[2];

if (index > size)

{

data[0] = int.MaxValue.ToString();

}

else if (index < 1)

{

data[0] = int.MinValue.ToString();

}

else

{

string line = File.ReadLines(filePath).Skip(index - 1).Take(1).First();

data = line.Split(' ');

}

if (int.Parse(data[0]) == key)

{

deletedIndexes.set.Add(key);

size--;

using (StreamReader r = new StreamReader(filePath))

using (StreamWriter w = new StreamWriter("temp\_index.txt"))

{

string line;

for (int i = 0; i < index - 1; i++)

{

w.WriteLine(r.ReadLine());

}

r.ReadLine();

while (!r.EndOfStream)

{

w.WriteLine(r.ReadLine());

}

}

File.Delete(filePath);

File.Move("temp\_index.txt", filePath);

return int.Parse(data[1]);

}

if (delta == 0)

{

return -1;

}

delta /= 2;

if (int.Parse(data[0]) < key)

{

index += delta + 1;

return Delete(key, index, delta);

}

else

{

index -= delta + 1;

return Delete(key, index, delta);

}

}

else

{

delta = size / 2;

index = delta + 1;

return Delete(key, index, delta);

}

}

OverflowArea.cs

public int Delete(int key)

{

size--;

StreamReader reader = new StreamReader(filePath);

StreamWriter writer = new StreamWriter("temp\_overflow.txt");

string line;

int row = -1;

while (!reader.EndOfStream)

{

line = reader.ReadLine();

string[] data = line.Split(' ');

if (int.Parse(data[0]) == key)

{

row = int.Parse(data[1]);

deletedIndexes.set.Add(key);

}

else

{

writer.WriteLine(line);

}

}

writer.Close();

reader.Close();

File.Delete(filePath);

File.Move("temp\_overflow.txt", filePath);

return row;

}

Database.cs

public void Delete(int row)

{

valuesNum--;

using (StreamReader r = new StreamReader(filePath))

using (StreamWriter w = new StreamWriter("temp\_db.txt"))

{

string line;

for (int i = 0; i < row - 1; i++)

{

w.WriteLine(r.ReadLine());

}

r.ReadLine();

w.WriteLine();

while (!r.EndOfStream)

{

w.WriteLine(r.ReadLine());

}

}

File.Delete(filePath);

File.Move("temp\_db.txt", filePath);

}

**Edit**

FileManager.cs

public bool Edit(int key, string value)

{

int row = index.Search(key);

if (row == -1)

{

row = area.Search(key);

if (row == -1)

{

return false;

}

else

{

db.Edit(row, value);

return true;

}

}

else

{

db.Edit(row, value);

return true;

}

}

Database.cs

public void Edit(int row, string newValue)

{

string[] lines = File.ReadAllLines(filePath);

string[] data = lines[row - 1].Split(' ');

lines[row - 1] = $"{data[0]} {newValue}";

File.WriteAllLines(filePath, lines);

}

### Приклади роботи

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

Зображення, що містить текст, знімок екрана, Шрифт, програмне забезпечення

Автоматично згенерований опис

Рисунок 3.1 –Додавання запису

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

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 | 14 |
| 2 | 12 |
| 3 | 14 |
| 4 | 10 |
| 5 | 14 |
| 6 | 12 |
| 7 | 14 |
| 8 | 13 |
| 9 | 11 |
| 10 | 13 |
| 11 | 14 |
| 12 | 12 |
| 13 | 14 |
| 14 | 9 |
| 15 | 14 |
| 16 | 12 |
| 17 | 14 |
| 18 | 13 |
| 19 | 11 |
| 20 | 13 |
| Середнє значення: | 12.65 |

Висновок

У результаті виконання лабораторної роботи було записано алгоритми пошуку, додавання, видалення і редагування запису в індексному файлі зі щільним індексом та областю переповнення.

Було виконано програмну реалізацію невеликої СУБД з графічним інтерфейсом з функцією однорідного бінарного пошуку, додавання, видалення та редагування записів.

База була заповнена випадковими значеннями до 10000 і зафіксовано середнє з 20 пошуків число порівнянь для знаходження запису по ключу.

Критерії оцінювання

За умови здачі лабораторної роботи до 26.11.2023 включно максимальний бал дорівнює – 5. Після 26.11.2023 максимальний бал дорівнює – 4,5.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 10%;
* аналіз часової складності – 5%;
* програмна реалізація алгоритму – 50%;
* робота з гіт – 20%
* тестування алгоритму – 10%;
* висновок – 5%.

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

+1 додатковий бал можна отримати за виконання та захист роботи до 19.11.2023.