**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)** *ІП-02, Ткачук Святослав*
(шифр, прізвище, ім'я, по батькові)

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

Київ 2021

# 1 ЗАВДАННЯ

| | |
|---|---|
| 24 | Файли з не щільним індексом з областю переповнення, метод Шарра |

# 2 ВИКОНАННЯ

## 2.1 Псевдокод алгоритмів

**Пошук елемента**

```
find(key):
    area = self.get_area_data(key)
    main_area_search = self.find_in_area(area, key)
    if main_area_search != None:
        return main_area_search
    elif self.occupancy(area) >= self.area_size:
        area = self.get_area_data('overflow')
        overflow_area_search = self.find_in_area(area, key)
        return overflow_area_search
    else:
        return None


find_in_area(array, K):
    self.number_of_comparisons = 0
    k = int(math.log2(N))
    i = 2 ** k
    if K < int(array[i - 1][:5]) or int(array[i - 1][:5]) == 0:
        j = 0
        delta = 2 ^ (k - j)
        while delta > 0:
            j += 1
            if K < int(array[i - 1][:5]) or int(array[i - 1][:5]) == 0:
                i -= delta // 2 + 1
                i = max(i, 1)
```

```
        elif K > int(array[i - 1][:5]):
            i = max(i, 1)
            i += delta // 2 + 1

        if int(array[i - 1][:5]) == K:
            return array[i - 1]

        if delta > 1:
            delta = 2 ** (k - j)
        else:
            delta = 0
    elif K > int(array[i - 1][:5]):
        l = int(math.log2(N - 2 ** k + 1))
        i = N + 1 - 2 ** l
        j = 1
        delta = 2 ** (l - j)
        if int(array[i - 1][:5]) == K:
            return array[i - 1]

        while delta > 0:
            j += 1
            if K < int(array[i - 1][:5]) or int(array[i - 1][:5]) == 0:
                i -= delta // 2 + 1
            elif K > int(array[i - 1][:5]):
                i += delta // 2 + 1

            if int(array[i - 1][:5]) == K:
                return array[i - 1]
```

```
            if delta > 1:
                delta = 2 ** (1 - j)
            else:
                delta = 0
        else:
            return array[i - 1]
```

**Додавання елемента**

```
    add(key, data):
        new_line = self.create_line(key, data)

        area = self.get_area_data(key)

        if self.occupancy(area)<self.area_size or self.find_in_area(area, key) != None:
            area = self.add_line(area, new_line)
            self.write_area(key, area)
        else:
            area = self.get_area_data('overflow')
            area = self.add_line(area, new_line)
            self.write_area('overflow', area)
        return 'added'


add_line(linses, line):
        for i in range(len(linses)):
            if int(linses[i][:5]) == int(line[:5]):
                linses[i] = line
```

```
            return linses
        if i < len(linses) - 1:
            if int(linses[i + 1][:5]) != 0:
                if int(line[:5]) > int(linses[i][:5]) and int(line[:5]) < int(linses[i + 1][:5]):
                    linses.insert(i + 1, line)
                    break
            elif int(linses[i][:5]) != 0:
                if int(line[:5]) > int(linses[i][:5]):
                    linses.insert(i + 1, line)
                    break
        if i == 0:
            if int(line[:5]) < int(linses[i][:5]) or int(linses[i][:5]) == 0:
                linses.insert(i, line)
                break
    return linses[:-1]
```

**Видалення**

```
remove(self, key):
    area = self.get_area_data(key)

    if self.occupancy(area) < self.area_size or self.find_in_area(area, key) != None:
        if self.find_in_area(area, key) == None:
            return -1
        area = self.remove_in_area(key, area)
        self.write_area(key, area)
    else:
        area = self.get_area_data('overflow')
```

```
        if self.find_in_area(area, key) == None:
            return -1
        area = self.remove_in_area(key, area)
        self.write_area(area)
    return 'removed'


remove_in_area(key, area):
    for i in range(len(area)):
        if int(area[i][:5]) != 0:
            if int(area[i][:5]) == key:
                del area[i]
                break
        else:
            break
    area.append(‘')
    return area
```

### 2.2 Часова складність пошуку

Оцінимо час доступу до довільного запису для файлів з нещільним індексом. Алгоритм розв'язання задачі аналогічний.

Спочатку визначимо розмір індексного запису. Якщо раніше посилання розраховувалася виходячи з того, що було потрібно посилатися на 100 000 записів, то тепер нам потрібно посилатися лише на 12 500 блоків, тому для посилання досить двох байт. Тоді довжина індексного запису буде дорівнює:

LI = LK + 2 = 14 + 2 = 14 байт.

Тоді кількість індексних записів в одному блоці дорівнюватиме:

KIZB = LB/LI = 1024/14 = 73 індексних записів в одному блоці.

Визначимо кількість індексних блоків, потрібних для зберігання необхідних індексних записів:
KIB = KBO/KZIB = 12500/73 = 172 блока.
Тоді час доступу за минулою формулою буде визначатися:
Тпоиска = log2KIB + 1 = log2172 + 1 = 8 + 1 = 9 звернень до диска.
Ми бачимо, що при переході до нещільного індексу час доступу зменшився практично в півтора рази. Тому можна визнати, що організація нещільного індексу дає виграш в швидкості доступу.

Кількість звернень до диска при додаванні нового запису дорівнює кількості звернень, необхідних для пошуку відповідного блоку плюс одне звернення, яке потрібно для занесення зміненого блоку на старе місце.
Тдодавання = log2N +1 + 1 звернень.

## 2.3 Програмна реалізація

### 2.3.1 Вихідний код

```
import csv
import random
import string
import math

class DataBase:
    def __init__(self):
        self.keys_in_area = 2000
        self.area_size = 1000
        self.main_area = open('main_area.csv', '+r')
        self.index_area = open("index_area.csv")
```

```
    self.number_of_comparisons = 0

def create_main_area(self):
    file = open('main_area.csv', 'w')
    for _ in range(11000):
        file.write('00000,             \n')
    file.close()

def generate_random_string(self):
    length = random.randint(1, 12)
    letters = string.ascii_lowercase
    rand_string = ''.join(random.choice(letters) for _ in range(length))
    return rand_string

def generate(self, num=10000):
    for i in range(num):
        print(i)
        key = random.randint(1, 20000)
        data = self.generate_random_string()
        self.add(key, data)

def to_index_area_key(self, key):
    res = self.keys_in_area
    while key > res:
        res += self.keys_in_area
    return res

def open_area(self, key):
    address = self.get_address(key)
```

```
    self.main_area.seek(address * 21)

def get_area_data(self, key):
    self.open_area(key)
    area = self.main_area.read(self.area_size * 20)
    area = area.split(sep='\n')
    return area[:-1]

def get_address(self, key):
    self.index_area.seek(0)
    index_data = []
    try:
        file_reader = csv.reader(self.index_area, delimiter=",")
        for row in file_reader:
            index_data.append(row)
    except IOError:
        return None

    area = ''
    if type(key) == type(1000):
        index_area_key = self.to_index_area_key(key)
    else:
        index_area_key = 'overflow'

    for row in index_data:
        if row[0] == str(index_area_key):
            area = row[1]
    return int(area)
```

```
def find(self, key):
    area = self.get_area_data(key)
    main_area_search = self.find_in_area(area, key)
    if main_area_search != None:
        return main_area_search
    elif self.occupancy(area) >= self.area_size:
        area = self.get_area_data('overflow')
        overflow_area_search = self.find_in_area(area, key)
        return overflow_area_search
    else:
        return None

def num_to_len_5(self, num):
    str_num = str(num)
    if len(str_num) > 5:
        return -1

    return '0' * ((5 - len(str_num))) + str_num

def str_to_len_n(self, s, n):
    return s + (' ' * (n - len(s)))

def create_line(self, key, data):
    return self.num_to_len_5(key) + ',' + self.str_to_len_n(data, 13)

def add_line(self, linses, line):
    print(len(linses))
    for i in range(len(linses)):
        if int(linses[i][:5]) == int(line[:5]):
```

```
                linses[i] = line
                return linses
            if i < len(linses) - 1:
                if int(linses[i + 1][:5]) != 0:
                    if int(line[:5]) > int(linses[i][:5]) and int(line[:5]) <
int(linses[i + 1][:5]):
                        linses.insert(i + 1, line)
                        break
                elif int(linses[i][:5]) != 0:
                    if int(line[:5]) > int(linses[i][:5]):
                        linses.insert(i + 1, line)
                        break
            if i == 0:
                if int(line[:5]) < int(linses[i][:5]) or int(linses[i][:5]) == 0:
                    linses.insert(i, line)
                    break
        return linses[:-1]

    def find_in_area(self, array, K):
        self.number_of_comparisons = 0
        N = len(array)
        k = int(math.log2(N))
        i = 2 ** k

        self.number_of_comparisons += 1
        if K < int(array[i - 1][:5]) or int(array[i - 1][:5]) == 0:
            j = 0
            delta = 2 ** (k - j)
```

```
    while delta > 0:
        self.number_of_comparisons += 1
        j += 1
        self.number_of_comparisons += 1
        if K < int(array[i - 1][:5]) or int(array[i - 1][:5]) == 0:
            i -= delta // 2 + 1
            i = max(i, 1)
        elif K > int(array[i - 1][:5]):
            i = max(i, 1)
            i += delta // 2 + 1

        if int(array[i - 1][:5]) == K:
            return array[i - 1]

        if delta > 1:
            delta = 2 ** (k - j)
        else:
            delta = 0
elif K > int(array[i - 1][:5]):
    l = int(math.log2(N - 2 ** k + 1))
    i = N + 1 - 2 ** l
    j = 1
    delta = 2 ** (l - j)
    self.number_of_comparisons += 1
    if int(array[i - 1][:5]) == K:
        return array[i - 1]

    while delta > 0:
        j += 1
```

```
            self.number_of_comparisons += 1
            if K < int(array[i - 1][:5]) or int(array[i - 1][:5]) == 0:
                i -= delta // 2 + 1
            elif K > int(array[i - 1][:5]):
                i += delta // 2 + 1

            if int(array[i - 1][:5]) == K:
                return array[i - 1]

            if delta > 1:
                delta = 2 ** (1 - j)
            else:
                delta = 0
        else:
            return array[i - 1]

    def write_area(self, key, area):
        address = self.get_address(key)
        self.main_area.seek(address * 21)

        united_area = ''
        for line in area:
            united_area += self.str_to_len_n(line, 19) + '\n'

        self.main_area.write(united_area)

    def add(self, key, data):
        new_line = self.create_line(key, data)
```

```
        area = self.get_area_data(key)

        if self.occupancy(area) < self.area_size or self.find_in_area(area,
key) != None:
            area = self.add_line(area, new_line)
            self.write_area(key, area)
        else:
            area = self.get_area_data('overflow')
            area = self.add_line(area, new_line)
            self.write_area('overflow', area)
        return 'added'

    def remove_in_area(self, key, area):
        for i in range(len(area)):
            if int(area[i][:5]) != 0:
                if int(area[i][:5]) == key:
                    del area[i]
                    break
            else:
                break
        area.append('00000,             ')
        return area

    def remove(self, key):
        area = self.get_area_data(key)

        if self.occupancy(area) < self.area_size or self.find_in_area(area,
key) != None:
            if self.find_in_area(area, key) == None:
```

```
            return -1
        area = self.remove_in_area(key, area)
        self.write_area(key, area)
    else:
        area = self.get_area_data('overflow')
        if self.find_in_area(area, key) == None:
            return -1
        area = self.remove_in_area(key, area)
        self.write_area(area)
    return 'removed'

def occupancy(self, area):
    k = 0
    for line in area:
        if int(line[:5]) == 0:
            return k
        k += 1
    return k

def __del__(self):
    self.main_area.close()
    self.index_area.close()

def get_number_of_comparisons(self):
    return self.number_of_comparisons
```

### 2.3.2 Приклади роботи

```
In[2]: from data_base import DataBase
In[3]: db = DataBase()
In[4]: db.find(2583)
Out[4]: '02583,rvsjmfsvtwyi '
In[5]: db.add(2578, 'dfgdgf')
1000
Out[5]: 'added'
In[6]: db.remove(2578)
Out[6]: 'removed'
```

## 2.4 Тестування алгоритму

### 2.4.1 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 | 19 |
| 2 | 19 |
| 3 | 21 |
| 4 | 15 |
| 5 | 19 |
| 6 | 21 |
| 7 | 17 |
| 8 | 21 |
| 9 | 10 |
| 10 | 10 |
| 11 | 15 |
| 12 | 21 |

| | |
|---|---|
| 13 | 21 |
| 14 | 19 |
| 15 | 10 |

# ВИСНОВОК

В рамках лабораторної роботи ми розглянули файли з нещільним індексом з областю переповнення та метод Шарра. Також було проведено аналіз часової складності та тестування алгоритмів. Дані алгоритми дозволяють дуже швидко виконувати такі оперції як пошук, додавання та видалення елемента у файлі.