**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)**

(шифр, прізвище, ім'я, по батькові)

*ІП-02, Ткачук Святослав*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2021

# Завдання

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

# Виконання

## 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 \*\* (l - 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

## 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:

Tпоиска = 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.

Tдодавання = log2N +1 + 1 звернень.

## Програмна реалізація

### 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 \*\* (l - 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):

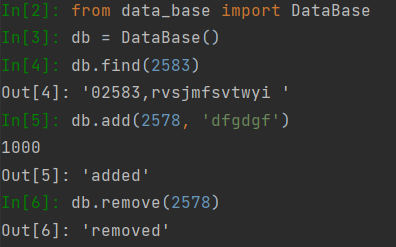
self.main\_area.close()

self.index\_area.close()

def get\_number\_of\_comparisons(self):

return self.number\_of\_comparisons

### Приклади роботи



## 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 | 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.