**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 Симоненко Михайло*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2022

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 7](#_Toc114359765)

[3.3 Програмна реалізація 7](#_Toc114359766)

[3.3.1 Вихідний код 7](#_Toc114359767)

[3.3.2 Приклади роботи 7](#_Toc114359768)

[3.4 Тестування алгоритму 8](#_Toc114359769)

[3.4.1 Часові характеристики оцінювання 8](#_Toc114359770)

[Висновок 9](#_Toc114359771)

[Критерії оцінювання 10](#_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** |
| 24 | Файли з не щільним індексом з областю переповнення, метод Шарра |

# Виконання

## Псевдокод алгоритмів

**function** binary\_search(key\_, records):

low = 0

high = length(records) - 1

counter = 0

**while** low <= high:

mid = (low + high) // 2

current\_record = records[mid]

counter += 1

**if** current\_record.get\_key() == to\_int(key\_):

show\_success\_message(current\_record.get\_str(), counter)

return True

**elif** current\_record.get\_key() < to\_int(key\_):

low = mid + 1

**else**:

high = mid - 1

**function** find\_el(key\_):

# Search in the file with a non-dense index

**for** key, records in self.i\_records.items():

**if** length(records) > 0 and records[0].get\_key() <= to\_int(key\_) <= records[-1].get\_key():

**if** binary\_search(key\_, records):

return

# Search in the overflow area

**if** length(self.overflow\_records)>0 and self.overflow\_records[0].get\_key() <= to\_int(key\_) <= self.overflow\_records[-1].get\_key():

**if** binary\_search(key\_, self.overflow\_records):

return

print(“key doesn’t exist”)

**function** update(key\_, value):

# Update in the overflow records

**for** record in self.overflow\_records:

**if** to\_int(record.get\_key()) > to\_int(key\_):

break

**if** to\_int(record.get\_key()) == to\_int(key\_):

record.update(value)

**return**

# Find the appropriate key in the indexed records

keys\_list = list(self.i\_records.keys())

last\_key = keys\_list[0]

**for** k in keys\_list:

**if** to\_int(k) <= to\_int(key\_):

last\_key = k

**else**:

break

# Update in the records associated with the last key

**for** record in self.i\_records[last\_key]:

**if** to\_int(record.get\_key()) == to\_int(key\_):

record.update(value)

**return**

**function** add(key\_, value):

new\_record = Record(to\_int(key\_), value)

# Check if a record with the same key already exists

**for** record in self.overflow\_records:

**if** to\_int(record.get\_key()) == to\_int(key\_):

show\_error\_message("Key already exists.")

**return**

**for** key, records in self.i\_records.items():

**for** record in records:

**if** to\_int(record.get\_key()) == to\_int(key\_):

show\_error\_message("Key already exists.")

return

# Check the ratio of records in indexed and overflow areas

**if** self.count\_i\_records \* self.ALPHA > self.count\_overflow\_records \* (1 - self.ALPHA):

# Add to the overflow area

self.overflow\_records.append(new\_record)

sort\_overflow\_records()

self.count\_overflow\_records += 1

**else**:

# Add to the main area

**if not** self.i\_records:

self.i\_records[to\_int(key\_)] = []

keys\_list = list(self.i\_records.keys())

last\_key = keys\_list[0]

**for** k in keys\_list:

**if** to\_int(k) <= to\_int(key\_):

last\_key = k

**else**:

break

temp\_arr = []

flag = True

**for** record in self.i\_records[last\_key]:

**if** to\_int(record.get\_key()) > to\_int(key\_) and flag:

temp\_arr.append(Record(to\_int(key\_), value))

self.count\_i\_records += 1

flag = False

temp\_arr.append(record)

**if** flag:

temp\_arr.append(Record(to\_int(key\_), value))

self.count\_i\_records += 1

self.i\_records[last\_key] = temp\_arr[:self.count\_page]

**for** record in temp\_arr[self.count\_page:]:

sort\_over\_records(record)

**function** delete(key\_):

# Delete from the overflow area

**for** record in self.overflow\_records:

**if** to\_str(record.get\_key()) == key\_:

self.overflow\_records.remove(record)

self.count\_overflow\_records -= 1

**return**

# Delete from the indexed records

**for** key, records in self.i\_records.items():

**for** record in records:

**if** to\_int(record.get\_key()) == to\_int(key\_):

self.i\_records[key].remove(record)

self.count\_i\_records -= 1

**return**

show\_error\_message("Key doesn't exist.")

**class** File2:

self.count\_page = 10 # кількість записів на сторінці

self.ALPHA = 0.4

self.i\_records = {}

self.overflow\_records = []

self.index\_records = []

self.count\_i\_records = 0

self.count\_overflow\_records = 0

## Часова складність пошуку

Пошук у файлі з нещільним індексом та області переповнення використовує метод бінарного пошуку. Таким чином, часова складність цих частин коду буде **O(logN),** де N - кількість записів у відповідній області (індексованій або переповненій).

## Програмна реалізація

### Вихідний код

import tkinter as tk  
from tkinter import filedialog, messagebox  
from File2 import File2  
import random  
import string  
from Record import Record  
  
file\_object = File2()  
  
  
def find():  
 key = entry1.get()  
 file\_object.find\_el(int(key))  
  
  
def add():  
 key = entry1.get()  
 new\_value = entry2.get()  
 file\_object.add(key, new\_value)  
  
 large\_window.delete(1.0, tk.END)  
 large\_window.insert(tk.END, file\_object.print\_info())  
  
  
def delete():  
 key = entry1.get()  
 file\_object.delete(key)  
  
 large\_window.delete(1.0, tk.END)  
 large\_window.insert(tk.END, file\_object.print\_info())  
 entry1.delete(0, tk.END)  
 entry2.delete(0, tk.END)  
  
def update():  
 key = entry1.get()  
 new\_value = entry2.get()  
  
 file\_object.update(key, new\_value)  
 large\_window.delete(1.0, tk.END)  
 large\_window.insert(tk.END, file\_object.print\_info())  
 entry1.delete(0, tk.END)  
 entry2.delete(0, tk.END)  
  
def save():  
 text\_content = large\_window.get("1.0", tk.END)  
 with open("data.txt", "w") as file:  
 file.write(text\_content)  
  
def load():  
 file\_object.all\_delete()  
 with open('data.txt', 'r') as file:  
 content = file.read()  
 lines = content.splitlines()  
 counter = 0  
 is\_main = True  
 for line in lines:  
 if line.startswith("=== Primary Area ===") or line.startswith(  
 "Page"):  
 continue  
 if line.startswith("=== Overflow Area ==="):  
 is\_main = False  
 continue  
 if line.startswith("=== Indexes ==="):  
 break  
 line = line.strip()  
 line = line.split()  
 if len(line)==0:  
 continue  
 rec = Record(int(line[0]), line[2])  
 file\_object.load\_record(rec, is\_main)  
 counter += 1  
 print(file\_object.print\_info())  
 large\_window.delete(1.0, tk.END)  
 large\_window.insert(tk.END, file\_object.print\_info())  
 print(counter)  
  
  
def generate\_random\_string(max\_length):  
 all\_letters = string.ascii\_letters  
 length = random.randint(1, max\_length)  
 random\_string = ''.join(random.choice(all\_letters) for \_ in range(length))  
 return random\_string  
  
def generate():  
 file\_object.all\_delete()  
 for i in range(20):  
 temp\_val = generate\_random\_string(10)  
 record = Record(i+1, temp\_val)  
 file\_object.add\_record(record)  
  
 large\_window.delete(1.0, tk.END)  
 large\_window.insert(tk.END, file\_object.print\_info())  
  
  
main\_window = tk.Tk()  
main\_window.title("My GUI")  
  
label1 = tk.Label(main\_window, text="Key:", font=("Arial", 12, "bold"))  
label2 = tk.Label(main\_window, text="Value:", font=("Arial", 12, "bold"))  
  
entry1 = tk.Entry(main\_window)  
entry2 = tk.Entry(main\_window)  
  
add\_button = tk.Button(main\_window, text="Add", width=15, command=add)  
search\_button = tk.Button(main\_window, text="Search", width=15, command=find)  
delete\_button = tk.Button(main\_window, text="Delete", width=15, command=delete)  
update\_button = tk.Button(main\_window, text="Update", width=15, command=update)  
save\_button = tk.Button(main\_window, text="Save", width=15, command=save)  
load\_button = tk.Button(main\_window, text="Load", width=15, command=load)  
generate\_button = tk.Button(main\_window, text="Generate", width=15, command=generate)  
  
  
def on\_scroll(\*args):  
 large\_window.yview(\*args)  
large\_window = tk.Text(main\_window, height=20, width=75)  
  
  
label1.grid(row=0, column=0, sticky="w", padx=10, pady=10)  
entry1.grid(row=0, column=1, padx=10, pady=10)  
label2.grid(row=0, column=2, sticky="w", padx=10, pady=10)  
entry2.grid(row=0, column=3, padx=10, pady=10)  
add\_button.grid(row=1, column=0, pady=10)  
search\_button.grid(row=1, column=1, pady=10)  
delete\_button.grid(row=1, column=2, pady=10)  
update\_button.grid(row=1, column=3, pady=10)  
save\_button.grid(row=2, column=0, pady=10)  
load\_button.grid(row=2, column=1, pady=10)  
generate\_button.grid(row=2, column=2, pady=10)  
large\_window.grid(row=3, column=0, columnspan=4, padx=10, pady=10)  
  
main\_window.mainloop()

from tkinter import messagebox  
  
class File2:  
 def \_\_init\_\_(self):  
 self.count\_page = 10 # кількість записів на сторінці  
 self.ALPHA = 0.4  
 self.i\_records = {}  
 self.overflow\_records = []  
 self.index\_records = []  
 self.count\_i\_records = 0  
 self.count\_overflow\_records = 0  
  
 def all\_delete(self):  
 self.i\_records = {}  
 self.overflow\_records = []  
 self.count\_i\_records = 0  
 self.count\_overflow\_records = 0  
  
 def add\_record(self, rec):  
 if self.count\_i\_records\*self.ALPHA>self.count\_overflow\_records\*(1-self.ALPHA):  
 self.overflow\_records.append(rec)  
 self.count\_overflow\_records += 1  
 else:  
 if not self.i\_records:  
 self.i\_records[1] = []  
 last\_key = max(self.i\_records)  
 if len(self.i\_records[last\_key])==self.count\_page:  
 self.i\_records[rec.get\_key()] = []  
 last\_key = max(self.i\_records)  
 self.i\_records[last\_key].append(rec)  
 self.count\_i\_records += 1  
  
 def load\_record(self, rec, is\_main):  
 if is\_main:  
 if not self.i\_records:  
 self.i\_records[1] = []  
 last\_key = max(self.i\_records)  
 if len(self.i\_records[last\_key])==self.count\_page:  
 self.i\_records[rec.get\_key()] = []  
 last\_key = max(self.i\_records)  
 self.i\_records[last\_key].append(rec)  
 self.count\_i\_records += 1  
 else:  
 self.overflow\_records.append(rec)  
 self.count\_overflow\_records += 1  
  
  
  
 def print\_info(self):  
 result\_str = "=== Primary Area ===\n"  
 counter = self.count\_page  
 c=1  
 index\_text = "\n=== Indexes ===\n"  
 for page\_number, record in self.i\_records.items():  
 index\_text += record[0].get\_str()  
 result\_str += f"Page {c}:\n"  
 c += 1  
 for i in record:  
 result\_str += " " + i.get\_str()  
 counter += 1  
  
 result\_str += "=== Overflow Area ===\n"  
 for record in self.overflow\_records:  
 result\_str += record.get\_str()  
 result\_str += index\_text  
  
 return result\_str  
  
 def delete(self, key\_):  
 for i in self.overflow\_records:  
 if str(i.get\_key())==key\_:  
 self.overflow\_records.remove(i)  
 self.count\_overflow\_records -= 1  
 return  
 for key, value in self.i\_records.items():  
 for i in value:  
 if int(i.get\_key())==int(key\_):  
 self.i\_records[key].remove(i)  
 self.count\_i\_records -= 1  
 return  
 messagebox.showinfo("Error", "Key doesn't exists.")  
  
 def add(self, key\_, value):  
 new\_record = Record(int(key\_), value)  
  
 # Перевірка, чи запис із таким ключем вже існує  
 for i in self.overflow\_records:  
 if int(i.get\_key()) == int(key\_):  
 messagebox.showinfo("Error", "Key already exists.")  
 return  
 for key, records in self.i\_records.items():  
 for i in records:  
 if i.get\_key() == int(key\_):  
 messagebox.showinfo("Error", "Key already exists.")  
 return  
 print(self.count\_i\_records, self.count\_overflow\_records)  
 if self.count\_i\_records \* self.ALPHA > self.count\_overflow\_records \* (1 - self.ALPHA):  
 # Додати в область переповнення  
 self.overflow\_records.append(new\_record)  
 self.overflow\_records.sort(key=lambda x: x.get\_key())  
 self.count\_overflow\_records += 1  
 else:  
  
  
 # Додати в основну область  
 if not self.i\_records:  
 self.i\_records[int(key\_)] = []  
 keys\_list = list(self.i\_records.keys())  
 last\_key = keys\_list[0]  
 for k in keys\_list:  
 if k<=int(key\_):  
 last\_key = k  
 else:  
 break  
  
  
 temp\_arr = []  
 flag = True  
 for i in self.i\_records[last\_key]:  
 if i.get\_key()>int(key\_) and flag:  
 temp\_arr.append(Record(int(key\_), value))  
 self.count\_i\_records += 1  
 flag = False  
 temp\_arr.append(i)  
 if flag:  
 temp\_arr.append(Record(int(key\_), value))  
 self.count\_i\_records += 1  
 self.i\_records[last\_key] = temp\_arr[:self.count\_page]  
 for i in temp\_arr[self.count\_page:]:  
 self.sort\_over\_records(i)  
 print("===\n", self.print\_info())  
  
  
 def sort\_over\_records(self, rec):  
 temp\_arr = []  
 flag = True  
 for i in self.overflow\_records:  
 if i.get\_key()>rec.get\_key() and flag:  
 temp\_arr.append(rec)  
 self.count\_overflow\_records += 1  
 flag = False  
 temp\_arr.append(i)  
 self.overflow\_records = temp\_arr  
  
 def update(self, key\_, value):  
 for i in self.overflow\_records:  
 if int(i.get\_key())>int(key\_):  
 break  
 if int(i.get\_key())==int(key\_):  
 i.update(value)  
 return  
  
 keys\_list = list(self.i\_records.keys())  
 last\_key = keys\_list[0]  
 for k in keys\_list:  
 if k <= int(key\_):  
 last\_key = k  
 else:  
 break  
 for i in self.i\_records[last\_key]:  
 if i.get\_key()==int(key\_):  
 i.update(value)  
 return  
 messagebox.showinfo("Error", "Key already exists.")  
  
 def binary\_search(self, key\_, records):  
 low = 0  
 high = len(records) - 1  
 counter = 0  
 while low <= high:  
 mid = (low + high) // 2  
 current\_record = records[mid]  
 counter += 1  
 if current\_record.get\_key() == int(key\_):  
 messagebox.showinfo("Success", current\_record.get\_str()+"\nCount of comparisons: "+str(counter))  
 return True  
 elif current\_record.get\_key() < int(key\_):  
 low = mid + 1  
 else:  
 high = mid - 1  
  
  
 def find\_el(self, key\_):  
 # Пошук у файлі з нещільним індексом  
 for key, records in self.i\_records.items():  
 if len(records) > 0 and records[0].get\_key() <= int(key\_) <= records[-1].get\_key():  
 if self.binary\_search(key\_, records):  
 return  
  
 # Пошук у області переповнення  
 if len(self.overflow\_records) > 0 and self.overflow\_records[0].get\_key() <= int(key\_) <= self.overflow\_records[  
 -1].get\_key():  
 if self.binary\_search(key\_, self.overflow\_records):  
 return  
  
 messagebox.showinfo("Error", "Key doesn't exists.")

### Приклади роботи

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

Изображение выглядит как текст, снимок экрана, дисплей, диаграмма

Автоматически созданное описание

Рисунок 3.1 –Додавання запису

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 3.2 – Пошук запису

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 3.3 – Пошук запису з неіснуючим ключем

Изображение выглядит как текст, снимок экрана, дисплей, диаграмма

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, дисплей, диаграмма

Автоматически созданное описание

Рисунок 3.4 – Оновлення існуючого запису

Изображение выглядит как текст, снимок экрана, дисплей, диаграмма

Автоматически созданное описание

Рисунок 3.5 – Видалення 5-ого 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 | 11 |
| 2 | 10 |
| 3 | 12 |
| 4 | 12 |
| 5 | 13 |
| 6 | 11 |
| 7 | 14 |
| 8 | 12 |
| 9 | 10 |
| 10 | 11 |
| 11 | 10 |
| 12 | 9 |
| 13 | 12 |
| 14 | 11 |
| 15 | 12 |

Висновок

Під час виконання даної лабораторної роботи було вивчено основні підходи до проектування та обробки складних структур даних. Згідно з варіантом, використовувалася структура даних "Файли з нещільним індексом з областю переповнення".

Були розроблені алгоритми пошуку, додавання, видалення і редагування записів у даній структурі даних. Ці алгоритми були виражені за допомогою псевдокоду, який відображав суть операцій, що виконуються.

Часова складність пошуку оцінюється як O(logN).

Для оцінки ефективності операції пошуку було проведено серію експериментів, заповнивши базу даних випадковими значеннями до 10000 записів. Зафіксовано середнє число порівнянь для знаходження запису по ключу під час 15 пошукових операцій: 11.(3). Зазвичай пошук записів ефективніший при пошуку в основній області завдяки індексам, проте це може залежати від відношення розмірів основної області та області переповнення.

Часова складність пошуку оцінюється як O(logN).

Також була розроблена програмна реалізація невеликої системи управління базою даних (СУБД) з графічним інтерфейсом користувача. Система дозволяє виконувати операції пошуку, додавання, видалення та редагування записів у базі даних, яка зберігається на постійному запам'ятовувальному пристрої.

На основі проведених експериментів можна зробити висновок про ефективність використаної структури даних та оптимальність алгоритмів обробки. Також, можна розглядати можливості оптимізації та покращення алгоритмів для підвищення продуктивності системи.

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.