**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**”

Варіант 1

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-з21 Стихун М.В.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2023

Зміст

1. Мета лабораторної роботи 3
2. Завдання 4
3. Виконання 7
   1. Псевдокод алгоритмів 7
   2. Часова складність пошуку 7
   3. Програмна реалізація 7
      1. Вихідний код 7
      2. Приклади роботи 17
   4. Тестування алгоритму 18
      1. Часові характеристики оцінювання 18

Висновок 19

# 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

## Псевдокод алгоритмів

Function binary\_search(list, target\_element):

left\_bound = 0

right\_bound = size(list) - 1

while left\_bound <= right\_bound:

middle = (left\_bound + right\_bound) / 2

middle\_element = list[middle]

if middle\_element == target\_element:

return middle # Found the target element

if middle\_element < target\_element:

left\_bound = middle + 1

else:

right\_bound = middle - 1

return -1 # Element not found in the list

## Часова складність пошуку

Часова складність бінарного пошуку є O(log n), де "n" - це кількість елементів у відсортованому масиві (або списку). Основна ідея бінарного пошуку полягає в тому, що на кожному кроці пошуку кількість можливих елементів для перевірки зменшується вдвічі.

## Програмна реалізація

### Вихідний код

├── app

│ ├── db

│ │ ├──tables

│ │ └──indexes

│ ├── file\_system

│ │ ├──crud.py

│ │ └──index .py

│ ├── lang

│ │ ├──\_base.py

│ │ ├──\_commander.py

│ │ ├──create\_table.py

│ │ ├──select.py

│ │ └──inser.py

│ └── main.py

└──

from lang.\_commander import LangCommander

def server():

while True:

if raw\_command := input("naviSql : "):

LangCommander.run\_command(raw\_command)

def main():

server()

if \_\_name\_\_ == '\_\_main\_\_':

main()

from typing import Optional

from file\_system.crud import CRUD

class String:

@classmethod

def between(cls, command: str, start: str, finish: str) -> str:

between\_str = ((command.split(start))[1].split(finish)[0]).strip()

return between\_str

@classmethod

def right(cls, command: str, word: str) -> str:

values\_i = command.rfind(word)

text = command[values\_i + len(word):].strip()

return text

class BaseCommand:

fs = CRUD

find = String

@classmethod

def deconstruct(cls, stringed\_data: str, mapper: Optional[dict[str, callable]] = None) -> dict:

query = {}

if not mapper:

return query

for i in stringed\_data.split(" "):

try:

query[i] = mapper[i](stringed\_data)

except KeyError:

pass

return query

@classmethod

def exec(cls):

raise NotImplementedError

from lang.create\_table import CreateTable

from lang.insert import Insert

from lang.select import Select

class LangCommander:

@classmethod

def run\_command(cls, stringed\_data: str) -> any:

commands = {

"insert": Insert,

"select": Select,

"create\_table": CreateTable,

}

try:

command = stringed\_data[:stringed\_data.index(" ")]

ins = commands[command](stringed\_data)

print(ins.exec() or "no result")

except Exception as exc:

print(exc)

from typing import Optional

from lang.\_base import BaseCommand

class CreateTable(BaseCommand):

def \_\_init\_\_(self, stringed\_data: str):

mapped\_q = self.deconstruct(stringed\_data)

self.table\_name: str = mapped\_q["create\_table"]

self.table\_columns: list = mapped\_q["columns"]

def exec(self) -> any:

self.fs.create\_index(self.table\_name)

return self.fs.create\_table(self.table\_name, self.table\_columns)

@classmethod

def sql\_create\_table(cls, command: str) -> str:

table\_name = cls.find.between(command, "create\_table", "columns")

return table\_name

@classmethod

def sql\_columns(cls, command: str) -> list:

columns = cls.find.right(command, "columns")

return columns.split(",")

@classmethod

def deconstruct(cls, stringed\_data: str, mapper: Optional[dict[str, callable]] = None) -> dict:

commands\_mapper = {

"create\_table": cls.sql\_create\_table,

"columns": cls.sql\_columns,

}

return super().deconstruct(stringed\_data, commands\_mapper)

from typing import Optional

from lang.\_base import BaseCommand

class Insert(BaseCommand):

def \_\_init\_\_(self, stringed\_data: str):

mapped\_q = self.deconstruct(stringed\_data)

self.table\_name = mapped\_q["insert"]

self.obj\_data = mapped\_q["values"]

def exec(self) -> any:

return self.fs.write(self.table\_name, self.obj\_data)

@classmethod

def sql\_insert(cls, command: str) -> str:

table\_name = cls.find.between(command, "insert", "values")

return table\_name

@classmethod

def sql\_values(cls, command: str) -> list:

values = cls.find.right(command, "values")

return values.split(",")

@classmethod

def deconstruct(cls, stringed\_data: str, mapper: Optional[dict[str, callable]] = None) -> dict:

commands\_mapper = {

"insert": cls.sql\_insert,

"values": cls.sql\_values,

}

return super().deconstruct(stringed\_data, commands\_mapper)

from typing import Optional

from lang.\_base import BaseCommand

class Select(BaseCommand):

def \_\_init\_\_(self, stringed\_data: str):

mapped\_q = self.deconstruct(stringed\_data)

self.columns: list = mapped\_q["select"]

self.table\_name: str = mapped\_q["from"]

self.obj\_id: int = mapped\_q["where"]

def exec(self) -> any:

return self.fs.read(self.table\_name, self.obj\_id, self.columns)

@classmethod

def sql\_select(cls, command: str) -> list:

args = cls.find.between(command, "select", "from")

return args.split(",")

@classmethod

def sql\_from(cls, command: str) -> str:

table\_name = cls.find.between(command, "from", "where")

return table\_name

@classmethod

def sql\_where(cls, command: str) -> int:

return int(cls.find.right(command, "where"))

@classmethod

def deconstruct(cls, stringed\_data: str, mapper: Optional[dict[str, callable]] = None) -> dict:

commands\_mapper = {

"select": cls.sql\_select,

"from": cls.sql\_from,

"where": cls.sql\_where,

}

return super().deconstruct(stringed\_data, commands\_mapper)

import re

from bisect import bisect\_left

class Index:

DB\_SOURCE = "db/indexes"

@classmethod

def get\_full\_index\_name(cls, table\_name: str) -> str:

return f"{cls.DB\_SOURCE}/{table\_name}"

@classmethod

def create\_index(cls, table\_name: str) -> None:

with open(cls.get\_full\_index\_name(table\_name), "w") as f:

pass

@classmethod

def get\_column\_indexes(cls, selected: list, full: list) -> list:

res = []

for i\_f, val\_f in enumerate(full):

for val\_s in selected:

if val\_f == val\_s:

res.append(i\_f)

return res

@classmethod

def get\_index(cls, table\_name: str, obj\_id: int) -> int:

with open(cls.get\_full\_index\_name(table\_name), "r") as f:

index = cls.binary\_search(f.readlines(), obj\_id) + 1

return index

@classmethod

def update\_index(cls, table\_name: str, obj\_id: int, last\_el: int):

with open(cls.get\_full\_index\_name(table\_name), "r+") as f:

data = f.read().splitlines()

data.append(f"{obj\_id}\_\_\_{last\_el}")

data.sort(key=lambda s: int(re.search(r'\d+', s).group()))

f.seek(0)

f.writelines([line + "\n" for line in data])

@classmethod

def binary\_search(cls, obj\_list: list, obj\_id: int) -> int:

i = bisect\_left([obj.split("\_\_\_")[0] for obj in obj\_list], str(obj\_id))

if i != len(obj\_list) and obj\_list[i].startswith(str(obj\_id)):

result = re.search('\_\_\_(.\*)\n', obj\_list[i])

return int(result.group(1))

else:

return -1

import json

from typing import Optional

from file\_system.index import Index

class CRUD:

index = Index

DB\_SOURCE = "db/tables"

@classmethod

def get\_full\_table\_name(cls, table\_name: str) -> str:

return f"{cls.DB\_SOURCE}/{table\_name}"

@classmethod

def create\_index(cls, table\_name: str) -> None:

cls.index.create\_index(table\_name)

@classmethod

def create\_table(cls, table\_name: str, obj\_columns: list) -> None:

with open(cls.get\_full\_table\_name(table\_name), "w") as f:

f.write(f'{"\_\_\_".join(obj\_columns)}\n')

@classmethod

def read(cls, table\_name: str, obj\_id: int, columns: list) -> Optional[str]:

obj\_index = cls.index.get\_index(table\_name, obj\_id)

if not obj\_index:

return

with open(cls.get\_full\_table\_name(table\_name), "r") as f:

lines = f.readlines()

labels = lines[0].replace("\n", "").split("\_\_\_")

res = {}

obj = lines[obj\_index].replace("\n", "").split("\_\_\_")

column\_indexes = cls.index.get\_column\_indexes(columns, labels)

for index in column\_indexes:

res[labels[index]] = obj[index]

return json.dumps(res)

@classmethod

def count(cls, table\_name: str) -> int:

with open(cls.index.get\_full\_index\_name(table\_name), "r") as f:

return len(f.readlines())

@classmethod

def write(cls, table\_name: str, data: list):

with open(cls.get\_full\_table\_name(table\_name), "a") as f:

f.write(f'{"\_\_\_".join(data)}\n')

cls.index.update\_index(table\_name, data[0], cls.count(table\_name))

### Приклади роботи

На рисунках 3.1 ,3.2 та 3.3 показані приклади роботи програми для додавання і пошуку запису.

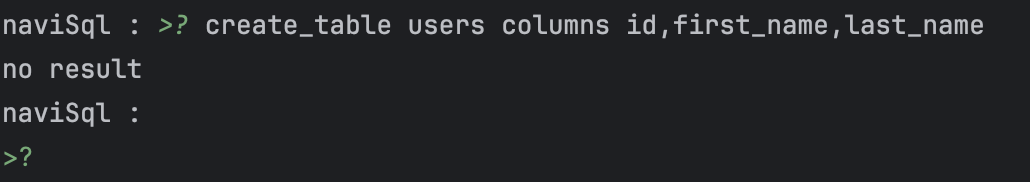
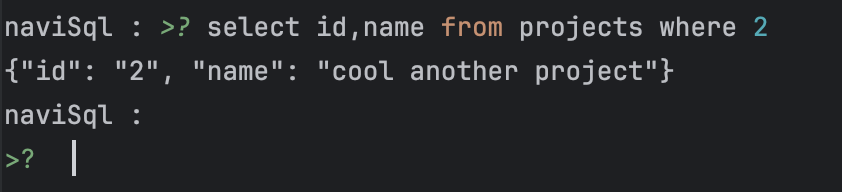
Рисунок 3.1 –Створення таблиці

Рисунок 3.2 –Додавання запису

Рисунок 3.3 – Пошук запису

## Тестування алгоритму

### Часові характеристики оцінювання

Індекс - у простій формі це додаткова структура, яка має відсортований список записів і посилання на їх реальне розташування.

За його принципом ми створюємо окрему таблицю, де зберігаються відсортовані по id пари \*\*obj\_id\*\*\_\_\_\*\*obj\_position\*\*. Це дозволяє в майбутньому використовувати алгоритми швидкого пошуку.

Впровадження індексів дозволяє істотно підвищити швидкість читання, але за рахунок швидкості запису, оскільки кожного разу при додаванні нового елемента ми перезаписуємо всі файли індексів (у нашій базі даних). У реальних базах даних все робиться по-іншому.

Реалізацій за цим принципом дуже багато. В одних випадках індекси знаходяться у файловій системі, в інших - в оперативній пам'яті або в комбінованих варіантах.

Тут під час додавання нового запису CRUD надсилає команду для перерахунку файлу індексу.

Висновок

У ході виконання цієї лабораторної роботи я успішно створив власну базу даних, використовуючи бінарний пошук як ефективний алгоритм для оптимізації пошуку записів. Цей процес не тільки дозволив мені глибше розібратися з принципами роботи баз даних, але й дав змогу вивчити практичні аспекти їхньої реалізації.

**Досягнуті результати та навчання:**

* **Створення та оптимізація бази даних:** Успішно реалізована можливість створення таблиць та операції вставки даних.
* **Використання бінарного пошуку:** Засвоєння принципів бінарного пошуку для швидкого та ефективного доступу до даних за їхнім унікальним ідентифікатором.
* **Індексація для покращення швидкості:** Вивчив, як створити просту систему індексації для покращення швидкості пошуку, особливо застосовуючи бінарний пошук.
* **Програмування на Python:** Поглиблено вивчив можливості мови програмування Python, використовуючи її для створення функціоналу бази даних.