**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Звіт**

з лабораторної роботи № 6

„ **Проектування і аналіз алгоритмів пошуку**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-21 Скрипець Ольга Олександрівна*

**Перевірив**

(прізвище, ім'я, по батькові)

Київ 2023

**Зміст**

[1 мета лабораторної роботи 3](#_Toc509037345)

[2 Завдання 4](#_Toc509037346)

[3 Виконання 8](#_Toc509037347)

[3.1 Псевдокод алгоритму 8](#_Toc509037349)

[3.2 Аналіз часової складності 8](#_Toc509037350)

[3.3 Програмна реалізація алгоритму 8](#_Toc509037351)

[3.3.1 Вихідний код 8](#_Toc509037352)

[3.3.2 Приклади роботи 8](#_Toc509037353)

[3.4 тестування алгоритму 9](#_Toc509037354)

[3.4.1 Часові характеристики оцінювання 9](#_Toc509037355)

[3.4.2 Графіки залежності часових характеристик оцінюваняя від розміру структури 10](#_Toc509037356)

[Висновок 11](#_Toc509037357)

[Критерії оцінювання 12](#_Toc509037358)

# Мета лабораторної роботи

Мета роботи – вивчити основні підходи аналізу обчислювальної складності алгоритмів пошуку оцінити їх ефективність на різних структурах даних.

# Завдання

Згідно варіанту (таблиця 2.1), написати алгоритм пошуку за допомогою псевдокоду (чи іншого способу за вибором).

Провести аналіз часової складності пошуку в гіршому, кращому і середньому випадках і записати часову складність в асимптотичних оцінках.

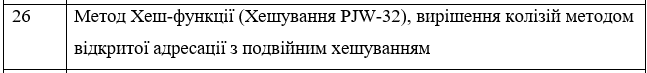
Виконати програмну реалізацію алгоритму на будь-якій мові програмування для пошуку індексу елемента по заданому ключу в масиві і двохзв'язному списку з фіксацією часових характеристик оцінювання (кількість порівнянь).

Для варіантів з **Хеш-функцією** замість масиву і двохзв'язного списку використати безіндексну структуру даних розмірності *n*, що містить пару ключ-значення рядкового типу. Ключ – унікальне рядкове поле до 20 символів, значення – рядкове поле до 200 символів. Виконати пошук значення по заданому ключу. Розмірність хеш-таблиці регулювати відповідно потребам, а початкову її розмірність обрати самостійно.

Провести ряд випробувань алгоритму на структурах різної розмірності (100, 1000, 5000, 10000, 20000 елементів) і побудувати графіки залежності часових характеристик оцінювання від розмірності структури.

Для проведення випробувань у варіантах з хешуванням рекомендується розробити генератор псевдовипадкових значень полів структури заданої розмірності.

Зробити висновок з лабораторної роботи.



# Виконання

## Аналіз часової складності

Метод хеш-функції для пошуку елемента в масиві має середню асимптотичну складність O(1). Час виконання алгоритму не залежить від розміру масиву або кількості елементів в ньому. Однак, це стосується середнього випадку, коли відсутні колізії (два різних ключа мають однаковий хеш). У найгіршому випадку, коли всі елементи масиву мають однаковий хеш або виникає багато колізій, складність може стати O(n), де n - розмір масиву. Проте, у нашому випадку в середньому метод працює дуже ефективно зі складністю O(1).

## Програмна реалізація алгоритму

### Вихідний код

import random

import matplotlib.pyplot as plt

class HashTable:

def \_\_init\_\_(self, size):

self.size = size

self.table = [None] \* size

self.comparison\_count = 0

def hash\_function(self, key):

h = 0

for char in key:

h = (h << 4) + ord(char)

g = h & 0xF0000000

if g != 0:

h ^= g >> 24

h &= ~g

return h % self.size

def double\_hash(self, key, i):

hash\_1 = self.hash\_function(key)

hash\_2 = 1 + (self.hash\_function(key) % (self.size - 1))

return (hash\_1 + i \* hash\_2) % self.size

def insert(self, key, value):

i = 0

while i < self.size:

index = self.double\_hash(key, i)

if self.table[index] is None:

self.table[index] = (key, value)

return True

i += 1

return False

def get(self, key):

i = 0

while i < self.size:

index = self.double\_hash(key, i)

self.comparison\_count += 1

if self.table[index] is not None and self.table[index][0] == key:

return self.table[index][1]

i += 1

return None

def generate\_random\_string(length):

chars = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890'

return ''.join(random.choice(chars) for \_ in range(length))

def print\_hash\_table(hash\_table):

for i, item in enumerate(hash\_table.table):

if item is not None:

print(f'Index {i}: Key: {item[0]}, Value: {item[1]}')

else:

print(f'Index {i}: Empty')

average\_comparisons = []

size = []

table\_size = int(input("Enter the size of the hash table: "))

# Створення хеш-таблиці

hash\_table = HashTable(table\_size \* 2)

key = []

# Заповнення хеш-таблиці згенерованими випадковими значеннями

for i in range(table\_size):

key.append(generate\_random\_string(20))

value = generate\_random\_string(200)

hash\_table.insert(key[i], value)

# Виведення хеш-таблиці на екран

print\_hash\_table(hash\_table)

# Пошук значення за ключем, введеним користувачем

key\_to\_find = input('Enter a key to search: ')

value = hash\_table.get(key\_to\_find)

if value:

print(f'Value for key {key\_to\_find}: {value}')

else:

print(f'Key {key\_to\_find} not found.')

print(f'Number of comparisons: {hash\_table.comparison\_count}')

table\_size = [100, 500, 1000, 2000, 5000, 10000, 15000, 20000]

# Отримання розміру хеш-таблиці від користувача

for cnt in range(8):

# Створення хеш-таблиці

hash\_table = HashTable(table\_size[cnt] \* 2)

key = []

# Заповнення хеш-таблиці згенерованими випадковими значеннями

for i in range(table\_size[cnt]):

key.append(generate\_random\_string(20))

value = generate\_random\_string(200)

hash\_table.insert(key[i], value)

for i in range(table\_size[cnt]):

key\_to\_find = key[i]

value = hash\_table.get(key\_to\_find)

total\_comparisons = hash\_table.comparison\_count

average = total\_comparisons / table\_size[cnt]

print(f'{table\_size[cnt]}:{average }')

size.append(table\_size[cnt])

average\_comparisons.append(average)

plt.plot(size, average\_comparisons)

plt.xlabel('Size of Hash Table')

plt.ylabel('Number of Comparisons')

plt.title('HashTable')

plt.show()

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми для пошуку індекса елемента за ключем для масиву на 100 елементів і двохзв'язного списку на 1000 елементів.



Рисунок 3.1 – Пошук елемента в масиві на 100 елементів

## Графіки залежності часових характеристик оцінювання від розмірності структури

На рисунку 3.3 показані графіки залежності часових характеристик від розмірності структур:

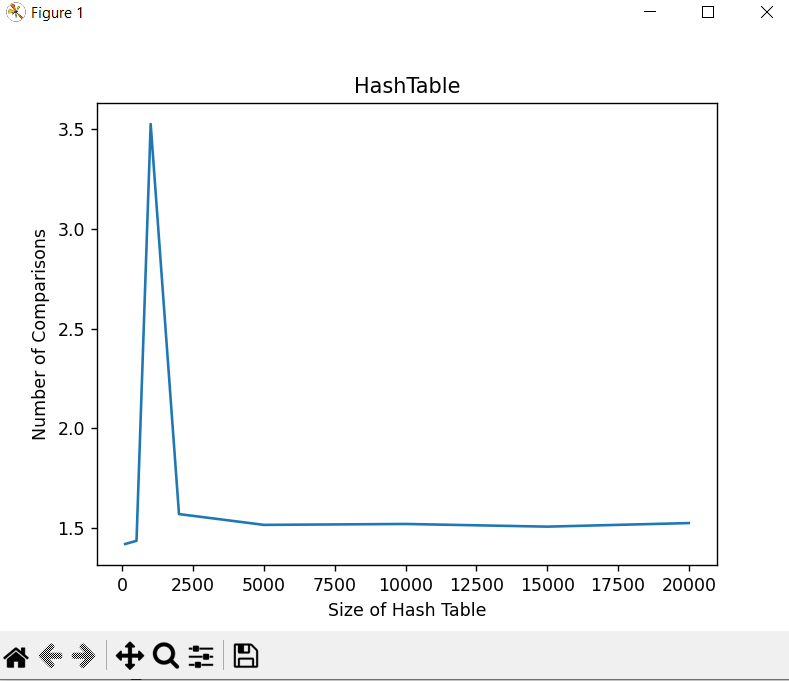


Рисунок 3.3 – Графіки залежності часових характеристик оцінювання

Висновок

В рамках виконання даної лабораторної роботи було реалізовано алгоритм пошуку елемента за ключем в безіндексній хеш-таблиці з вирішенням колізій методом відкритої адресації з подвійним хешуванням.

Алгоритм використовує дві хеш-функції для обчислення хешу ключа, і використовується подвійне хешування для вирішення колізій. При вставці елемента, він обчислюється хеш першою функцією, а якщо в цій позиції вже є елемент, використовується друга хеш-функція для знаходження нової позиції. Якщо ця позиція також зайнята, алгоритм переходить до наступної позиції згідно зі змінною "step". Так продовжується до тих пір, поки не буде знайдений елемент з заданим ключем або буде досягнуто пустої позиції.

Аналізуючи часову складність, в гіршому випадку, коли всі елементи мають однаковий хеш або виникає багато колізій, пошук в безіндексній хеш-таблиці з використанням методу відкритої адресації з подвійним хешуванням може мати складність O(n), де n - розмірність таблиці. Це відбувається, оскільки алгоритм може перебрати багато позицій у таблиці, перш ніж знайде елемент з заданим ключем або порожню позицію.

У кращому випадку, коли немає колізій і ключ знаходиться на своєму місці, часова складність стає O(1), оскільки потрібно лише обчислити хеш ключа і звернутися безпосередньо до відповідної позиції в таблиці.

У середньому випадку, часова складність також може бути оцінена як O(1), оскільки ми припускаємо, що розподіл ключів у таблиці рівномірний і відсутні багато колізій. Тому в середньому випадку алгоритм працює ефективно.

Завершуючи висновок, можна сказати, що реалізація пошуку елемента в безіндексній хеш-таблиці з використанням методу відкритої адресації з подвійним хешуванням є ефективним способом пошуку елемента за ключем, особливо в середньому та кращому випадках.