МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М.Є. Жуковського

«Харківський авіаційний інститут»

Факультет радіоелектроніки, комп’ютерних систем та інфокомунікацій

Кафедра комп’ютерних систем, мереж і кібербезпеки (503)

Лабораторна робота № *4*

|  |  |
| --- | --- |
|  | *Дослідження простих алгоритмів пошуку* |
|  | (назва лабораторної роботи) |
| з дисципліни | *Моделі та структури даних* |
|  | (шифр)  ХАІ**.**503**.**525a**.**03О**.**123-Комп'ютерна інженерія**,** ПЗ №9629619 |

|  |  |  |
| --- | --- | --- |
| Виконав студент гр. | 525а | *Литвиненко А.В.* |
| 16.11.2022 | (№ групи) | (П.І.Б.) |
| (підпис, дата) |  |  |
| Перевірив | канд. техн. наук, доцент | |
|  |  | *А. В. Шостак* |
| (підпис, дата) |  | (П.І.Б.) |

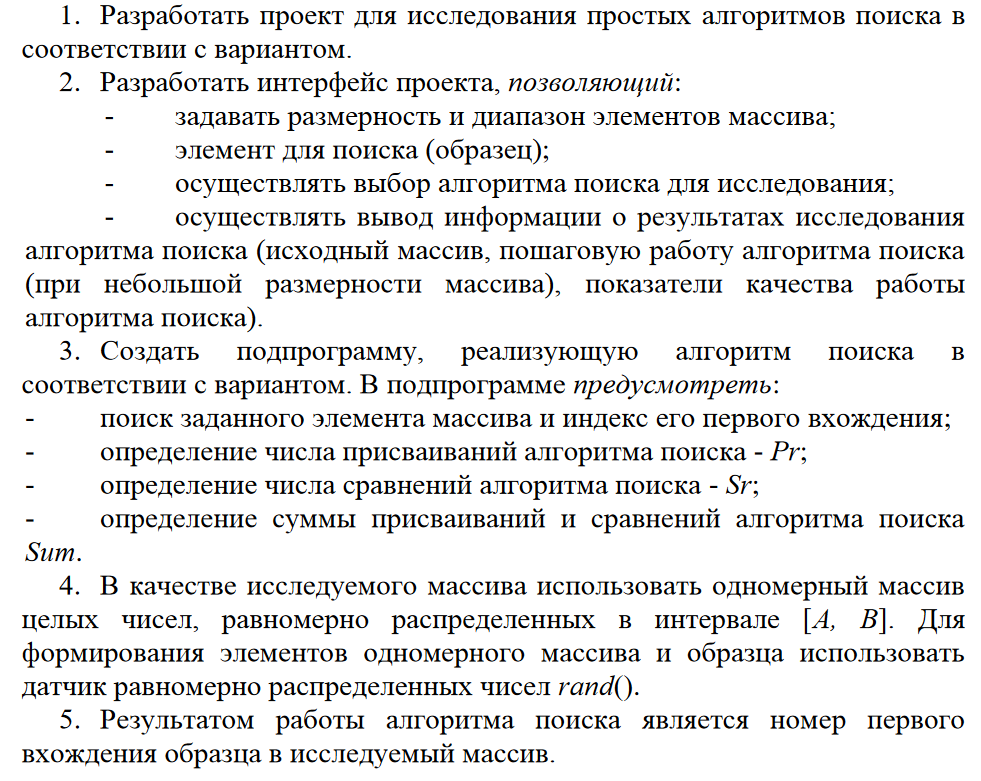
Харків – 2022

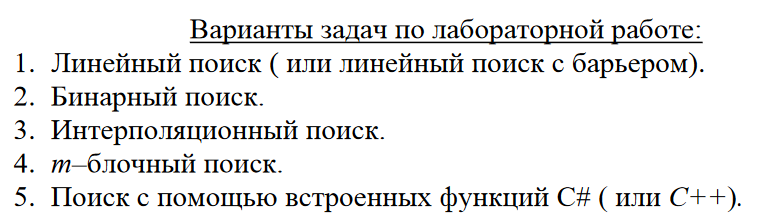
**Варіант 5**

**Задача 1**

**Частина 1**. Постановка завдання

**Умова:**





**Умова з додатка:**



**Частина 2**. Схема класу

На основі постановки завдання розроблений алгоритм, представлений на рисунку 1.

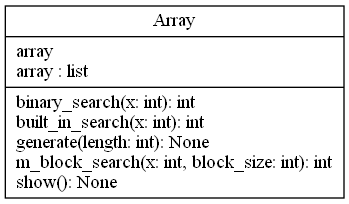


Рисунок 1 - Алгоритм перетворення

**Частина 3**. Заміри показників

Таблиця 1 – Показники якості бінарного пошуку

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | 10 | 100 | 400 | 800 | 2000 |
| Sr | 2 | 6 | 9 | 10 | 10 |
| Time | 0.01536 | 0.02083 | 0.01092 | 0.02143 | 0.02114 |

Таблиця 2 – Показники якості m-блочного пошуку

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 10 | | | 100 | | | 400 | | | 800 | | | 2000 | |
| m | 2 | 10 | | 2 | 10 | | 2 | 10 | | 2 | 10 | | 2 | 10 |
| Sr | 15 | 15 | | 146 | 146 | | 669 | 669 | | 1071 | 1071 | | 2523 | 2523 |
| Time | 0.0211 | 0.0219 | | 0.0204 | 0.0210 | | 0.0190 | 0.0199 | | 0.0186 | 0.0213 | | 0.0226 | 0.0204 |
| Avg  Sr | 15 | | 146 | | | 669 | | | 1071 | | | 2523 | | |
| Avg  Time | 0.0215 | | 0.0207 | | | 0.0194 | | | 0.0199 | | | 0.0215 | | |

Таблиця 3 – Показники якості вбудованого пошуку

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | 10 | 100 | 400 | 800 | 2000 |
| Sr | 0 | 0 | 0 | 0 | 0 |
| Time | 0.01934 | 0.01967 | 0.02139 | 0.02132 | 0.01803 |

**Частина 4**. Текст програми

Відповідно до розробленого алгоритму в середовищі Microsoft Visual Studio була написана програма, яка наведена нижче.

import time

from random import randint

class Array:

def \_\_init\_\_(self, arr: list):

self.\_array = arr.copy()

@property

def array(self):

return self.\_array

@array.setter

def array(self, value):

self.\_array = value.copy()

def \_\_len\_\_(self):

return len(self.\_array)

def \_\_getitem\_\_(self, index):

return self.\_array[index]

def show(self) -> None:

for i in self.array:

print('%d ' % i, end='')

print()

def generate(self, length: int) -> None:

MIN, MAX = 0, 1000

new = []

for i in range(length):

new.append(randint(MIN,MAX))

self.array = new

def binary\_search(self, x: int) -> int:

# array must be sorted!

op\_count = 0

arr = sorted(self.array)

low = 0

high = len(arr) - 1

mid = 0

while low <= high:

op\_count += 1

mid = (high + low) // 2

if arr[mid] < x:

low = mid + 1

elif arr[mid] > x:

high = mid - 1

else:

print(op\_count)

return 1

print(op\_count)

return 0

def m\_block\_search(self, x: int, block\_size: int = 1) -> int:

# array must be sorted!

op\_count = 0

arr = sorted(self.array)

i = len(arr) - 1

j = 0

s = 0

b = []

while 1:

op\_count += 1

b.append(block\_size)

s += b[j]

if(s >= i): break

j = 0

k = b[j] - 1

j += 1

while(arr[k] < x and k <= i):

op\_count += 1

# print(j, b[j])

k += b[j]

j += 1

if(k > j):

print(op\_count)

return -1

z = k - b[j - 1]

while(z <= k):

op\_count += 1

z += 1

if(arr[z] == x):

print(op\_count)

return 1

print(op\_count)

return -1

def built\_in\_search(self, x: int) -> int:

print('[INFO] OPERATION COUNT CAN\'NT BE DEFINED!')

try:

a = self.array.index(x)

a = 1

except ValueError:

a = 0

return a

def runtime\_decorator(func, blocksize: int = 1):

arg = int(input('[VALUE] >> '))

start\_time = time.time()

if func is Array.m\_block\_search:

result = func(arg, blocksize)

else:

result = func(arg)

if result in [0, -1]:

print("Not found")

else:

print("Found")

end\_time = time.time()

print('Runtime: ', round(end\_time - start\_time, 5))

def menu() -> None:

a = """Menu:

- 0. Show this menu

- 1. Enter an array

- 2. Generate an array

- 3. # Binary search

- 4. # M-block search

- 5. # Build-in search

- 6. Show

- 7. Show random element and its position (faster)

- 8. Exit"""

print(a)

op = 0

arr = Array([])

while 1:

match op:

case 0:

menu()

case 1:

t = map(int, input("[ARR] >> ").split(" "))

arr = list(t)

print('[OK]')

case 2:

l = int(input('[LEN] >> '))

arr.generate(l)

print('[OK]')

case 3:

runtime\_decorator(arr.binary\_search)

case 4:

m = input('[Block size] >> ')

runtime\_decorator(arr.m\_block\_search, m)

case 5:

runtime\_decorator(arr.built\_in\_search)

case 6:

arr.show()

print('[OK]')

case 7:

r = randint(0, len(arr)-1)

print(r)

print(f'Element: {arr[r]}\nIndex: {r}')

case 8:

print("[OK] Exiting...")

exit()

# Else

case \_:

print("[ERR] An invalid operation!")

op = int(input('>> '))

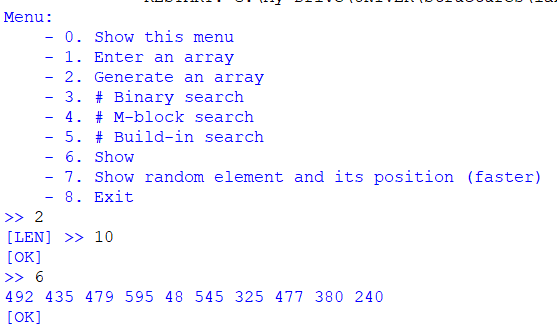
**Частина 5**. Скриншоти роботи на прикладі малого масиву

Довжина: 10

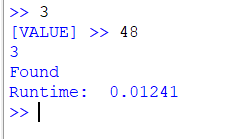
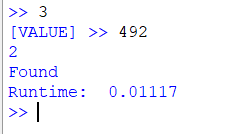
Масив: [492 435 479 595 48 545 325 477 380 240]

Відсортований масив: [48 240 325 380 435 477 479 492 545 595]

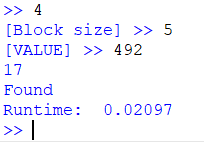
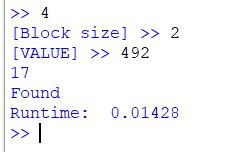
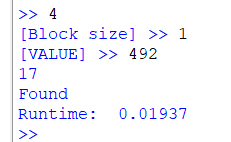
Числа для пошуку: 492, 48

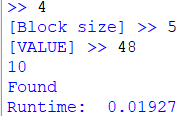
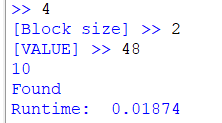
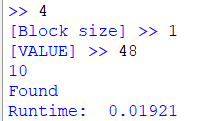


Бінарний пошук

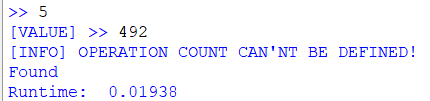


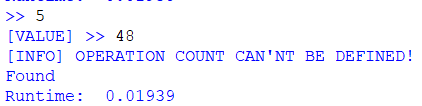
M-блочний пошук





Вбудований пошук





**Висновки**

Під час цієї лабораторної роботи я вивчав різні алгоритми пошуку та реалізовував їх на практиці, прочитавши багато матеріалу я зміг зрозуміти принципи їх роботи та особливості кожного. Наприклад бінарний пошук є дуже швидким, але потребує відсортованого масиву, як і m-блочний пошук.

Моє розуміння про бінарний пошук: він бере мінімальну, максимальну позицію і вираховує їх середнє значення, якщо шукане значення не дорівнює середньому і більше за нього, то змінюється мінімальна позиція і виконується розрахунок середньої, якщо ж менше – змінюється максимальна позиція і так до тих пір поки не знайдеться елемент або поки не позиції не почнуть перекривати одна одну.

Моє розуміння про m-блочний пошук: масив розбивається на блоки довжиною в m, масив пробігається по цим блокам і порівнює значення. Плюс цього пошуку є його швидкість, якщо правильно підібрати довжину блоку.