|  |  |  |  |
| --- | --- | --- | --- |
| МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені Тараса Шевченка  ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  **Кафедра програмних систем і технологій**  Дисципліна  **«Структури данних, аналіз і**  **алгоритми ком’ютерної**  **обробки інформаціїї»**  **Лабораторна робота № 1**  **на тему:** | | | |
| **Виконав:** | Аврущенко Дмитро Мансоровича | **Перевірила**: | Бичков Олексій Сергійович |
| Група | ІПЗ-12(1) | Дата перевірки |  |
| Форма навчання | денна | Оцінка |  |
| Спеціальність | 121 |
| 2022 | | | |

**Умова лабораторної:**

Написати програму мовою C# з можливістю вибору різних алгоритмів пошуку. Продемонструвати роботу (ефективність, час виконання) програм на різних структурах даних (масив, лінійний зв’язаний список), з різними умовами, що забезпечують зменшення часу виконання. Навести аналіз отриманих результатів.

Реалізувати алгоритми:

* пошуку перебором елемента масиву, що дорівнює заданому значенню.
* пошуку з бар'єром елемента масиву, що дорівнює заданому значенню.
* бінарного пошуку елемента масиву рівного заданому значенню.
* бінарного пошуку елемента масиву, рівного заданому значенню, в якій нове значення індексу m визначалося б не як середнє значення між L і R, а згідно з правилом золотого перерізу.

**Аналіз**

Спочатку нам необхідно створити масив та однозв’язний список з однаковою кількістю та позицією елементів. Потім за допомогою класу Stopwatch виміряємо кількість затраченого часу на кожний алгоритм. Так як це клас і він має свої методи ми будемо зустрічати деякі не співпадіння у часі між однаковими алгоритмами, але у середньому ми зможемо побачити, який алгоритм пошуку краще та з якою структурою даних. Також відображений час у цьому класі залежить і від ресурсів комп’ютера.

Спочатку виконаємо пошук перебором, тобто пройдемо по кожному елементу до поки не знайдемо шукане число або не вийдемо за границю масиву чи списку.

Пошук з бар’єром допомагає не перевіряти вихід за границі, але він потребує додавання одного елементу, що для масиву є дуже складним, на відміну від списку.

Бінарний пошук потребує відсортованих елементів, тому спочатку відсортуємо масив та список і будемо виконувати пошук.

Бінарний пошук за правилом золотої середина також як і попередній потребує відсортованого масиву чи списку.

Так як бінарний пошук працює лише у відсортованих даних, то масив та список необхідно відсортувати, але час сортування до загального часу пошуку додавати не будемо.

**Структура основних вхідних та вихідних даних**

* Вхідні дані:

Вхідні дані – створенні псевдовипадкові числа за допомогою класу Random та згруповані до масиву чи списку

* Вихідні дані:

Вихідними даними є повідомлення про те знайдено елемент чи ні. Якщо елемент є, то виводимо індекс цього елемента, якщо немає, то виводимо повідомлення про його відсутність. Також виводимо використаний час.

**Алгоритм розв’язання задачі:**

Спочатку створимо масив та зв’язний список. Вони будуть на 1000 елементів. Тепер ініціюємо псевдовипадковими числами масив та цими ж числами ініціюємо список. Далі за допомогою switch обираємо тип алгоритму пошуку.

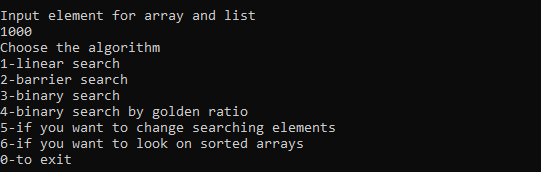
1. Лінійний пошук. Пройдемо по кожному елементу до поки не знайдемо шукане число або не вийдемо за границю масиву чи списку. Реалізуймо його за допомогою циклу while
2. Пошук із бар’єром. Для його реалізації необхідно створити новий масив на 1 елемент більше та скопіювати його. Також на останню позицію поставити шуканий елемент. Для списку необхідно лише створити «вузол» та прив’язати його до всього списку.
3. Бінарний пошук. Спочатку необхідно відсортувати масив та список. Потім за правилом бінарного пошуку знайдемо індекс елемента
4. Бінарний пошук за правилом золотого перерізу. Також як і попередній пункт, але замість пошуку центрального елемента, ми шукаємо елемент за золотим перерізом, тобто first + (end -first)/1.61803398

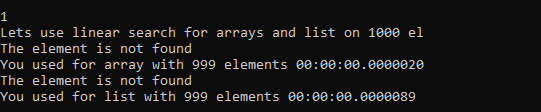
**Набір тестів**

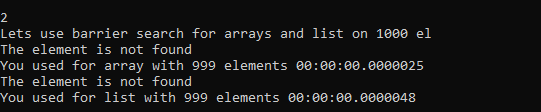
* Для початку введемо числа, які не належать масиву та спискам.
* Потім введемо числа, що належать масиву та списку. Та подивимося, яка швидкість пошуку кожного пошуку.

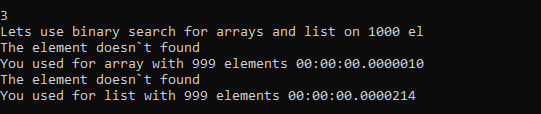
**Результати тестування**

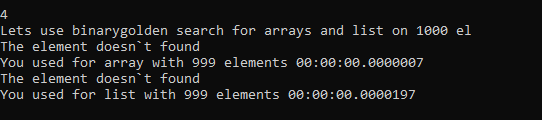
Знайдемо елемент 1000, який у генераторі випадкових чисел знаходиться за межою(вибраний мною діапазон значень (-999, 999)).



****

****

****

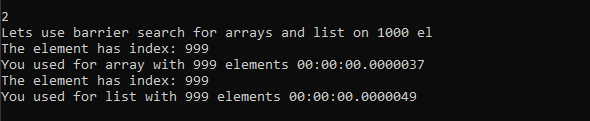
****

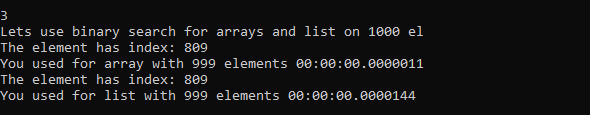
Тепер знайдемо якесь ще число, яке буде знаходитися майже у кінці початкового та відсортованого масиву/списку

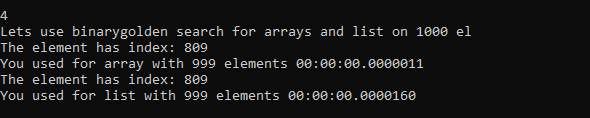
****

**Изображение выглядит как текст

Автоматически созданное описание**

****

****

****

**Аналіз отриманих результатів**

Тепер знайдемо середнє арифметичне для того, щоб оцінити кожний час у середньому та вибрати найшвидший алгоритм пошуку елемента.

|  |  |  |
| --- | --- | --- |
| Назва Сортування | Масив | Список |
| Лінійний пошук | 23.5 | 74 |
| Лінійний пошук з бар’єром | 31 | 48.5 |
| Бінарний пошук | 10.5 | 179 |
| Бінарний з золотим перерізом | 9 | 178.5 |

Як ми можемо побачити найшвидшим пошуком у масиві є бінарний пошук з золотим перерізом зі швидкістю 9 ticks. Найшвидшим пошуком у списках, на моє здивування, став лінійний пошук з бар’єром зі швидкістю 48.5 ticks. Якщо загалом проаналізувати пошук, то у середньому пошук у масиві є швидшим.

**Аналіз помилок**

Перша помилка, яка в мене з’явилася, то це як реалізувати список на практиці. Хоча теоретично я розумів як це зробити. Трохи по експериментувавши, я зрозумів як він працює та зміг його реалізувати.

Також на мою думку є помилкою дуже криве індексування(на мою думку + я не придумав іншого способу), яке збільшує час пошуку в багато разів, але тепер набагато простіше користуватися списками.

Ще одна помилка, яка в мене була виявилася на стадії оптимізування мого коду. Саме лінійний пошук та пошук з бар’єром були у мене жахливо великими а саме 3500 ticks та 3300 відповідно. Але як ви бачите, виправивши код, я зміг досягнути такого пришвидшення. Також можу сказати, що на цьому ж етапі я зіткнувся з виходами за кінцеві елементи та з неправильними порівняннями умов виходу із циклів.

Ще можу сказати, що мій бінарний пошук переходив у безкінечний пошук, бо я не задав умови виходу( тобто якщо елемент знайдено або перевірено всі елементи), також я забув сортувати масиви та списки, але вже потім я все виправив коли зайшов у дебаг та побачив, що елементи не відсортовані.

**Висновок**

Під час цієї лабораторної роботи я навчився використовувати списки та реалізувати їх у «польових умовах». Але все ж таки списки, що пропонуються у вбудованих бібліотеках C# є більш ефективнішими за ті, що написав я.