Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ КОМП'ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Катедра «Комп'ютерна інженерія та програмування»

ЗВІТ

про виконання лабораторної роботи №9

з навчальної дисципліни «Алгоритми та структури даних»

Варіант 5

Виконав студент:

Омельніцький Андрій Миколайович
Група: КН-1023б

Перевірив:

Старший викладач

Бульба Сергій Сергійович

Харків - 2024

Зміст

1 Мета роботи

Набути навичок та закріпити знання при виконанні операцій пошуку.

Теми для попередньої роботи:

- набори даних: масиви, лінійні списки;
- алгоритми пошуку за числовомим ключом: лінійний, лінійний з бар'єром, двійковий, експоненційний, інтерполяційний;
- алгоритми пошуку зразка в тексті: прямий, КМП, БМ алгоритми.

2 Завдання

Розробити та налагодити програму, в якій реалізувати два алгоритми пошуку відповідно до завдання. Порівняти алгоритми за часом роботи.

На етапі тестування для кожного з алгоритмів (варіанти 4 – 15) визначити кількість порівнянь у наборі даних з різною кількістю елементів (20, 100, 1000, 10000) визначити час пошуку, заповнити таблицю за формою табл. 9.1, побудувати графіки, зробити висновки. При роботі з текстом (варіанти 1 – 3) змінювати довжину текста та зразка.

УВАГА! У завданнях 4 – 15 передбачена робота з цілими числами; наступні 12 завдань (номери 16 – 27) вимагають роботу з дійсними числами.

Завдання за варіантом (5): Двійковий та лінійний пошуки у лінійному списку.

3 Хід виконання

Для виконання завдання було обрано мову Rust. Увесь код також додатково був розміщений в GitHub репозитарії: https://github.com/blackgolyb/algos-labs.

3.1 Лінійний пошук

```rust
use crate::libs::list::double_linked_list::List;

use super::super::logger::Logger;

pub fn linear_search<T: PartialEq>(
    list: &mut List<T>,
    elemet: T,
    logger: &mut Logger,
) -> Option<usize> {
    logger.start();

    let mut i: usize = 0;
    unsafe {
        let mut node = list.head;
        while !node.is_null() {
            let e = &mut (*node).value;
            logger.log_compare();
            if elemet.eq(e) {
                logger.end();
                return Some(i);
            }
            logger.log_shift();
            node = (*node).next;
            i += 1;
        }
    }
    logger.end();
    None
}
```

3.2 Двійковий пошук

```rust
use crate::libs::list::double_linked_list::List;

use super::super::logger::Logger;

pub fn binary_search<T: Ord + PartialEq>(
    list: &mut List<T>,
    element: T,
    logger: &mut Logger,
) -> Option<usize> {
    logger.start();
    unsafe {
        let mut left = list.head;

        let mut l = 0;
        let mut r = list.len() as i64 - 1;

        while l <= r {
            let m = (l + r) / 2;
            let mut mid = left;
            for _ in 0..(m - l) {
                logger.log_shift();
                mid = (*mid).next;
            }

            let e = &mut (*mid).value;
            logger.log_compare();
            if element.eq(e) {
                logger.end();
                return Some(m as usize);
            } else if element.gt(e) {
                logger.log_compare();
                logger.log_shift();
                l = m + 1;
                left = (*mid).next;
            } else {
                logger.log_compare();
                r = m - 1;
            }
        }
    }
    logger.end();
    None
}
```

3.3 Приклад роботи програми

Для перевірки працездатності напишемо програму яка буде порівнювати двійковий та лінійний пошук у лінійному списку. Порівнювати будемо за такими критеріями: кількість порівнянь, кількість переходів до наступного вузла та час пошуку.
Код програми для перевірки:

```rust
use csv::Writer;

use crate::libs::list::double_linked_list::List;

use super::logger::{Logger, Metrics};
use super::variants::binary_search;
use super::variants::linear_search;

fn test(n: usize, elem: i64) -> (Metrics, Option<usize>, Metrics, Option<usize>) {
    let mut list = List::<i64>::new();
    let mut logger = Logger::new();

    for i in 0..n as i64 {
        list.push(i);
    }

    let res_linear = linear_search(&mut list, elem, &mut logger);
    let metrics_linear = logger.get_metrics();
    let res_binary = binary_search(&mut list, elem, &mut logger);
    let metrics_binary = logger.get_metrics();

    (metrics_linear, res_linear, metrics_binary, res_binary)
}

pub fn main() {
    let out_file = "perf_lab9.csv".to_string();
    let test_cases: Vec<usize> = vec![20, 100, 1000, 10000, 100000, 1000000, 10000000, 100000000];
    let elem_rate = 0.66666666;
    // let elem_rate = 0.1;

    let mut wtr = Writer::from_path(out_file).unwrap();
    wtr.write_record(&[
        "n",
        "linear_compares",
        "linear_shifts",
        "linear_time",
        "binary_compares",
        "binary_shifts",
        "binary_time",
    ])
    .unwrap();

    for n in test_cases {
        let elem = (elem_rate * (n as f64)).round() as i64;
        let (m1, r1, m2, r2) = test(n, elem);
        println!("Test Case    n = {n}    element = {elem}");
        println!("Linear Search: Found: {r1:?} — {m1}");
        println!("Binary Search: Found: {r2:?} — {m2}\n");

        let Metrics(c1, s1, t1) = m1;
        let Metrics(c2, s2, t2) = m2;
        wtr.write_record(&[
            n.to_string(),
            c1.to_string(),
            s1.to_string(),
            t1.as_nanos().to_string(),
            c2.to_string(),
            s2.to_string(),
            t2.as_nanos().to_string(),
        ])
        .unwrap();
    }
}
```

```
Test Case   n = 20   element = 13
Linear Search: Found: Some(13) -- Comparisons: 14, Shifts: 13, Execution Time: 483ns
Binary Search: Found: Some(13) -- Comparisons: 9, Shifts: 17, Execution Time: 887ns

Test Case   n = 100   element = 67
Linear Search: Found: Some(67) -- Comparisons: 68, Shifts: 67, Execution Time: 1.638µs
Binary Search: Found: Some(67) -- Comparisons: 7, Shifts: 91, Execution Time: 2.37µs

Test Case   n = 1000   element = 667
Linear Search: Found: Some(667) -- Comparisons: 668, Shifts: 667, Execution Time: 14.709µs
Binary Search: Found: Some(667) -- Comparisons: 19, Shifts: 992, Execution Time: 22.475µs

Test Case   n = 10000   element = 6667
Linear Search: Found: Some(6667) -- Comparisons: 6668, Shifts: 6667, Execution Time: 74.452µs
Binary Search: Found: Some(6667) -- Comparisons: 27, Shifts: 9992, Execution Time: 117.595µs

Test Case   n = 100000   element = 66667
Linear Search: Found: Some(66667) -- Comparisons: 66668, Shifts: 66667, Execution Time: 716.043µs
Binary Search: Found: Some(66667) -- Comparisons: 31, Shifts: 99986, Execution Time: 1.195942ms

Test Case   n = 1000000   element = 666667
Linear Search: Found: Some(666667) -- Comparisons: 666668, Shifts: 666667, Execution Time: 8.5304ms
Binary Search: Found: Some(666667) -- Comparisons: 37, Shifts: 999986, Execution Time: 14.288922ms

Test Case   n = 10000000   element = 6666667
Linear Search: Found: Some(6666667) -- Comparisons: 6666668, Shifts: 6666667, Execution Time: 89.561249ms
Binary Search: Found: Some(6666667) -- Comparisons: 45, Shifts: 9999981, Execution Time: 145.253679ms

Test Case   n = 100000000   element = 66666666
Linear Search: Found: Some(66666666) -- Comparisons: 66666667, Shifts: 66666666, Execution Time: 924.153559ms
Binary Search: Found: Some(66666666) -- Comparisons: 51, Shifts: 99999981, Execution Time: 1.568060855s
```

Рис. 1. Приклад роботи

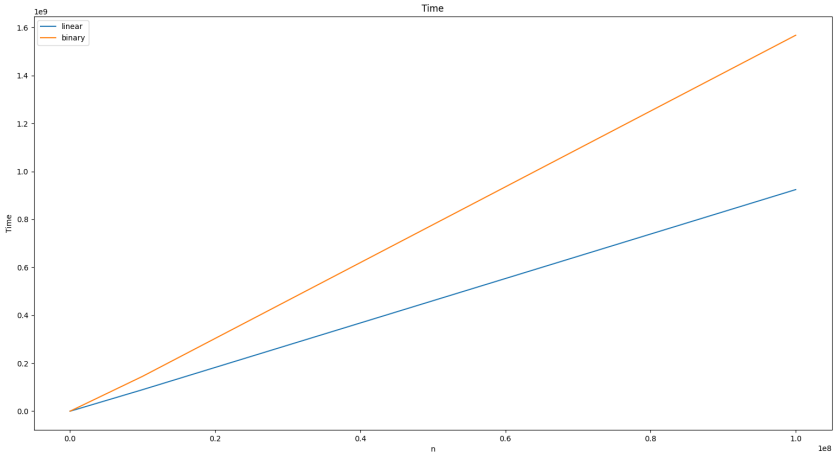

Рис. 2. Залежність часу пошуку від кількості елементів

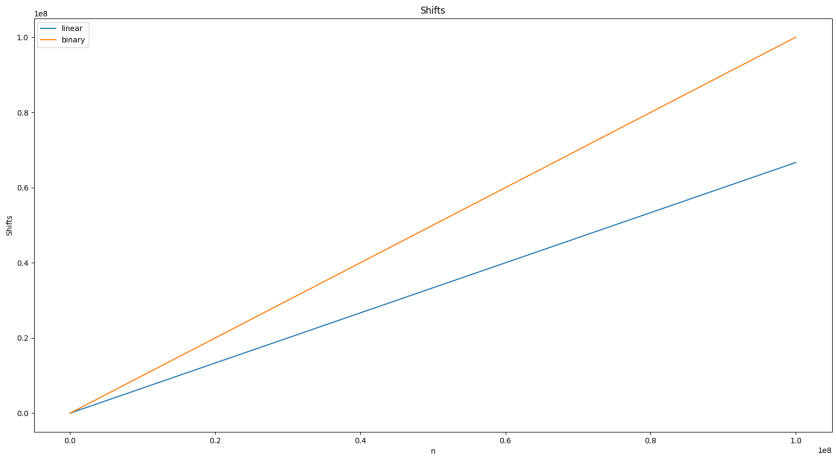

Рис. 3. Залежність переходів до наступного вузла від кількості елементів

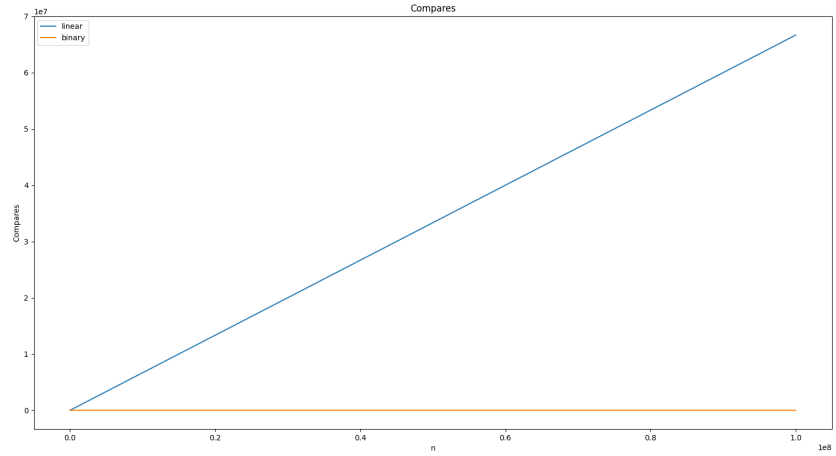

Рис. 4. Залежність кількості порівнянь від кількості елементів

4 Висновки

В ході виконання лабораторної робити було створено лінійний та бінарний пошук у лінійному списку. За результатами порівняння було виявлено, що лінійний пошук займає менше часу і менше кількості переходів до наступного вузла, проте робить набагато більше порівнянь. Бінарний пошук займає більше часу і більше переходів до наступного вузла, але виконує менше порівнянь. Також явно прослідковується кореляція часу та переходів до наступного вузла, а внесок меншої кількості порівнянь не значно впливає на час всього пошуку. Що є логічним, бо операція порівняння цілих чисел значно швидша за операцію разадресації яка виконується при кожному переході до наступного вузла. Тому за умови що операція разадресації на порядок довша за порівняння, тоді лінійний пошук буде кращім рішенням для зв’язного списку.