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 №10

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

Закріпити знання про алгоритми пошуку, що потребують додаткової пам’яті; набути навичок виконання операцій пошуку із використанням таблиць прямого доступу, довідників та хешованих таблиць.

Теми для попередньої роботи:

- масиви та списки;
- файли;
- прості алгоритми пошуку;
- алгоритми пошуку із застосуванням таблиць прямого доступу;
- поняття про хеш-таблиці, функції хешування, колізії;
- алгоритми розв’язання колізій.

2 Завдання

Початкові дані містяться у текстовому файлі. Прочитати файл, створити таблицю прямого доступу або хеш-таблицю відповідно до завдання з табл. 10.1. Перевірити працездатність створених таблиць на прикладі операцій пошуку.

Порівняти час пошуку із використанням створених таблиць та простих алгоритмів пошуку з лабораторної роботи 4. Для кожного з алгоритмів визначити кількість порівнянь у наборі даних з різною кількістю елементів (20, 1000, 5000, 10000, 50000), визначити час пошуку, заповнити таблицю за формою табл. 10.2, побудувати графіки, зробити висновки.

№	Вміст початкових даних	Таблиця	Хеш-функція	Ключ пошуку
1	2	3	4	5
5	Мережева адреса станції, операційна система, мережевий протокол	Хеш-таблиця з розподіленими ланцюжками переповненнь	Функція перетворення системи числення та ділення за модулем	Мережева адреса станції

Рис. 1. Завдання за варіантом (5)

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 Хеш-функція

Розглянемо спочатку нашу функцію: функція перетворення системи числення та ділення за модулем. Така функція буде мати дві змінні, які будуть впливати на кількість колізій - це параметр системи числення (base) та число, по якому модулю ми будемо брати, що своєю чергою є розміром (size) нашої хеш-таблиці. Тому перед подальшою роботою треба провести дослідження цієї функції та знайти оптимальні параметри.

Для цього напишемо програму, яка на певній рівномірно розподіленій вибірці, значення якої лежить в діапазоні $[0, 2^{63} - 1]$, розмір вибірки візьмемо $n = 1000$ і для кожної пари $(base, size)$ будемо рахувати помилку (error) за таким алгоритмом:

1. Створюємо масив розміром size
2. Знаходимо цільову кількість елементів на кожен backet за формулою: $target = n/size$
3. Для кожного елемента в вибірці рахуємо його хеш
4. За знайденим хешем додаємо до масиву 1
5. Після обробки усієї вибірки рахуємо MSE між нашим масивом та target

Для подальшого досліду візьмемо значення base в діапазоні $[2, 257]$, а значення size в діапазоні $[8, n]$

Щоб оцінити, як функція поводить себе при різних значеннях, побудуємо графік помилки нашої функції за вище описаним алгоритмом. Використаємо Python та бібліотеки для візуалізації для побудови цього графіку.

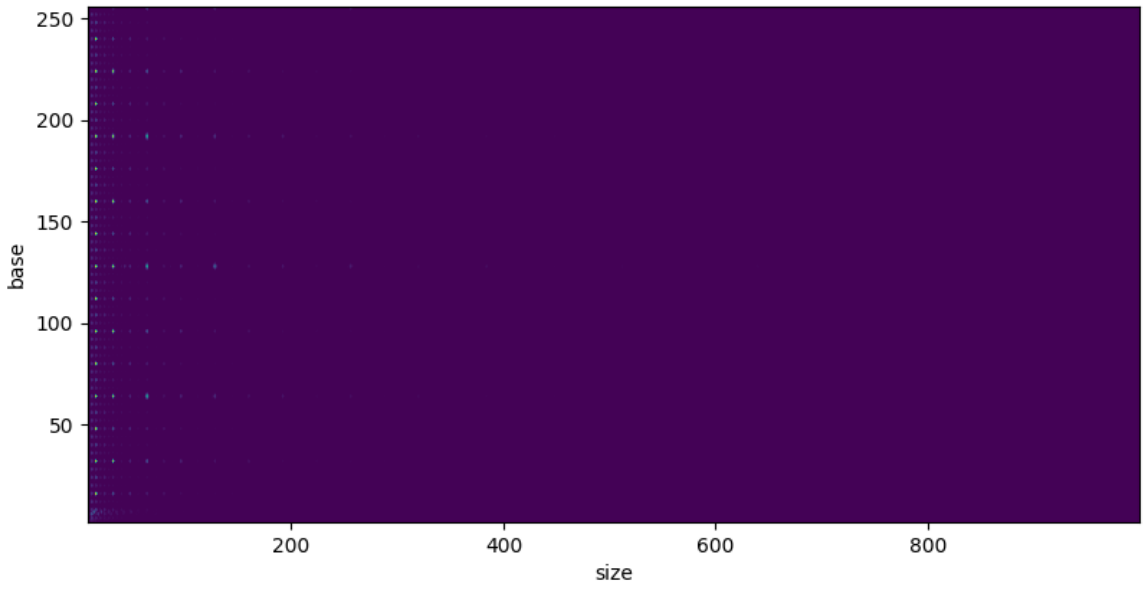

Рис. 2. Графік функції помилки

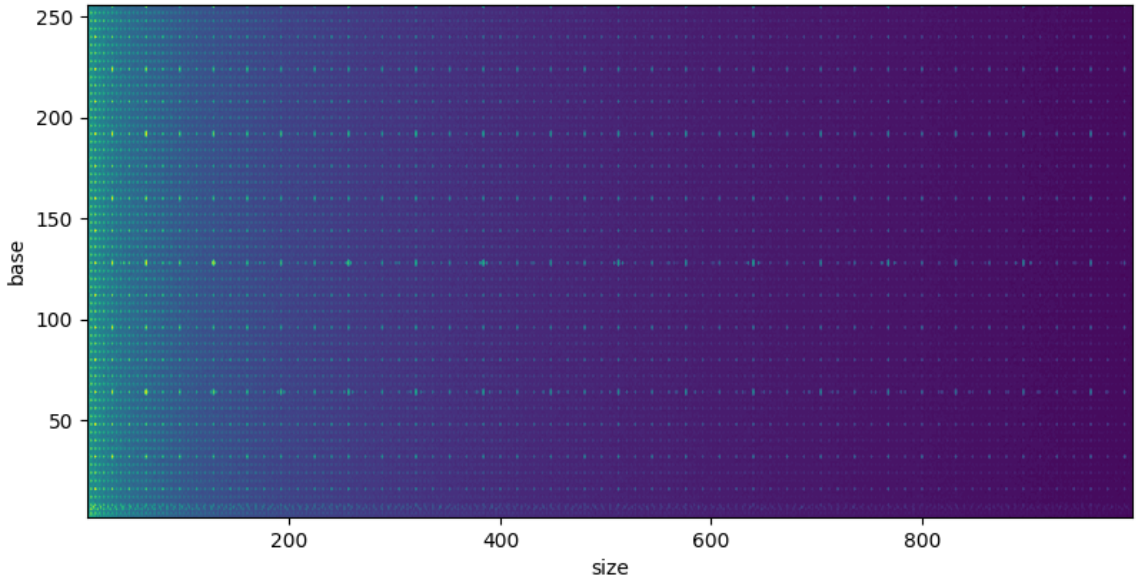

Рис. 3. Графік функції помилки з логарифмованим значенням помилки для згладження піків

Як можна побачити з вищенаведених графіків хеш-функція за парних значень набуває більшого значення помилки, а особливо на значеннях коли base та size є ступенем двійки. Також в ході дослідження було знайдено, що найкраще себе показує функція коли значення base та size є простими, але різниця не настільки суттєва якщо порівнювати з непарними числами. Тому надалі для обрання параметрів base та size будемо використовувати непарні числа.

Також хочеться зазначити, що таку хеш-функцію краще не використовувати в реальній роботі, бо вона: повільно працює, дуже перебірлива до вхідних параметрів та не підходить до будь-яких даних. Тому на цей час найкращий вибір буде xxHash, бо вона: працює на граничній швидкості оперативної пам’яті, дає дуже гарний розподіл хешу та працює не представленням даних, а з набором байтів.

3.2 Хеш-таблиця

Реалізуємо хеш-таблицю де ключем може бути будь-який тип який можна хешувати та порівняти, а значенням будь-який тип.

```
use std::{
    cell::RefCell,
    fmt,
    hash::{Hash, Hasher},
    rc::Rc,
};

type EntryRef<K, V> = Rc<RefCell<Entry<K, V>>>;

struct Entry<K, V>
where
    K: Hash + Eq,
{
    key: K,
    value: Rc<RefCell<V>>,
    next: Option<EntryRef<K, V>>,
}

pub struct HashTable<K, V>
where
    K: Hash + Eq,
{
    buckets: Vec<Option<EntryRef<K, V>>>,
}

impl<K, V> HashTable<K, V>
where
    K: Hash + Eq,
{
    pub fn new() -> Self {
        Self::new_with_capacity(255)
    }

    pub fn new_with_capacity(capacity: usize) -> Self {
        Self {
            buckets: vec![None; capacity],
        }
    }

    pub fn capacity(&self) -> usize {
        self.buckets.len()
    }

    fn new_element(key: K, value: V) -> EntryRef<K, V> {
        Rc::new(RefCell::new(Entry {
            key,
            value: Rc::new(RefCell::new(value)),
            next: None,
        }))
    }

    pub fn get_index(&self, key: &K) -> usize {
        let mut hasher = std::collections::hash_map::DefaultHasher::new();
        key.hash(&mut hasher);
        let mut initial_hash = hasher.finish();

        let base: u64 = 3;
        let mut hash = 0;
        let mut i = 0;

        while initial_hash > 0 {
            hash += (initial_hash % 10) * base.pow(i);
            initial_hash /= 10;
            i += 1;
        }

        (hash % self.capacity() as u64) as usize
    }

```

```
70      pub fn contains(&self, key: K) -> bool {
71          let index = self.get_index(&key);
72          let elem = &self.buckets[index];
73          if elem.is_none() {
74              return false;
75          }
76
77          let mut elem = elem.clone();
78
79          loop {
80              let e = elem.as_ref().unwrap();
81              if e.borrow().key == key {
82                  return true;
83              }
84
85              if e.borrow().next.is_none() {
86                  return false;
87              }
88
89              let next = e.borrow_mut().next.clone();
90              elem = next;
91          }
92      }
93
94      pub fn insert(&mut self, key: K, value: V) {
95          let index = self.get_index(&key);
96          let elem = &mut self.buckets[index];
97
98          if elem.is_none() {
99              self.buckets[index] = Some(Self::new_element(key, value));
100             return;
101         }
102
103         let mut elem = elem.clone();
104
105         loop {
106             let e = elem.as_ref().unwrap();
107             if e.borrow().key == key {
108                 *e.borrow_mut().value.borrow_mut() = value;
109                 return;
110             }
111
112             if e.borrow().next.is_none() {
113                 e.borrow_mut().next = Some(Self::new_element(key, value));
114                 return;
115             }
116
117             let next = e.borrow_mut().next.clone();
118             elem = next;
119         }
120     }
121
122     pub fn get(&self, key: K) -> Option<Rc<RefCell<V>>> {
123         let index = self.get_index(&key);
124         let elem = &self.buckets[index];
125         if elem.is_none() {
126             return None;
127         }
128
129         let mut elem = elem.clone();
130
131         loop {
132             let e = elem.as_ref().unwrap();
133             if e.borrow().key == key {
134                 return Some(e.borrow().value.clone());
135             }
136
137             if e.borrow().next.is_none() {
138                 return None;
139             }
140
141             let next = e.borrow_mut().next.clone();
142             elem = next;
143         }
144     }
145 }
```

```
146
147 impl<K, V> fmt::Display for HashTable<K, V>
148 where
149     K: Hash + Eq,
150     K: fmt::Display,
151     V: fmt::Display,
152 {
153     fn fmt(&self, f: &mut fmt::Formatter<'_>) -> fmt::Result {
154         for i in 0..self.buckets.len() {
155             if self.buckets[i].is_none() {
156                 continue;
157             }
158             let mut elem = (&self.buckets[i]).clone();
159
160             let mut pos = 0;
161
162             loop {
163                 let e = elem.as_ref().unwrap();
164                 write!(
165                     f,
166                     "{key} => {value}   | [ buckets id: {i}; \t linked list position: {pos} ]\n",
167                     key = e.borrow().key,
168                     value = e.borrow().value.borrow()
169                 )?;
170
171                 if e.borrow().next.is_none() {
172                     break;
173                 }
174
175                 let next = e.borrow_mut().next.clone();
176                 elem = next;
177                 pos += 1;
178             }
179         }
180         Ok(())
181     }
182 }
```

3.3 Опис даних

Напишемо структуру даних як описана у завданні та напишемо для неї серіалі затор та десереалізатор. А також функції для генерації випадкових значень цих даних.

```rust
use std::cmp::Ordering;
use std::io::prelude::*;
use std::{fmt, str::FromStr};

use rand::distributions::{Distribution, Standard};
use rand::Rng;
use serde::{Deserialize, Deserializer, Serialize, Serializer};

#[derive(Serialize, Deserialize, Debug, Clone, Copy)]
pub enum OS {
    #[serde(rename = "Windows")]
    Windows,
    #[serde(rename = "Debian")]
    Debian,
    #[serde(rename = "NixOS")]
    NixOS,
    #[serde(rename = "Ubuntu")]
    Ubuntu,
    #[serde(rename = "Fedora")]
    Fedora,
    #[serde(rename = "CentOS")]
    CentOS,
    #[serde(rename = "Arch")]
    Arch,
    #[serde(rename = "MacOS")]
    MacOS,
    #[serde(rename = "FreeBSD")]
    FreeBSD,
    #[serde(rename = "OpenBSD")]
    OpenBSD,
}

#[derive(Serialize, Deserialize, Debug, Clone, Copy)]
pub enum Protocol {
    #[serde(rename = "TCP")]
    TCP,
    #[serde(rename = "UDP")]
    UDP,
    #[serde(rename = "HTTP")]
    HTTP,
    #[serde(rename = "QUIC")]
    QUIC,
    #[serde(rename = "SMTP")]
    SMTP,
    #[serde(rename = "FTP")]
    FTP,
}

#[derive(Debug, Hash, PartialEq, Eq, PartialOrd, Ord, Copy, Clone)]
pub struct Ipv4Address {
    octets: [u8; 4],
}

impl Distribution<OS> for Standard {
    fn sample<R: Rng + ?Sized>(&self, rng: &mut R) -> OS {
        match rng.gen_range(0..=10) {
            0 => OS::Windows,
            1 => OS::Debian,
            2 => OS::NixOS,
            3 => OS::Ubuntu,
            4 => OS::Fedora,
            5 => OS::CentOS,
            6 => OS::Arch,
            7 => OS::MacOS,
            8 => OS::FreeBSD,
            _ => OS::OpenBSD,
        }
```

```
    }
}

impl Distribution<Protocol> for Standard {
    fn sample<R: Rng + ?Sized>(&self, rng: &mut R) -> Protocol {
        match rng.gen_range(0..=6) {
            0 => Protocol::TCP,
            1 => Protocol::UDP,
            2 => Protocol::HTTP,
            3 => Protocol::QUIC,
            4 => Protocol::SMTP,
            _ => Protocol::FTP,
        }
    }
}

impl fmt::Display for Ipv4Address {
    fn fmt(&self, f: &mut fmt::Formatter<'_>) -> fmt::Result {
        write!(
            f,
            "{:03}.{:03}.{:03}.{:03}",
            self.octets[0], self.octets[1], self.octets[2], self.octets[3]
        )
    }
}

pub struct ParseIpAdrrError {
    // TODO: add more information
}

impl fmt::Display for ParseIpAdrrError {
    fn fmt(&self, f: &mut fmt::Formatter<'_>) -> fmt::Result {
        write!(f, "Failed to parse ip addres")
    }
}

impl FromStr for Ipv4Address {
    type Err = ParseIpAdrrError;

    fn from_str(s: &str) -> Result<Self, Self::Err> {
        let octets: Result<Vec<u8>, _> = s.split('.').map(|part| part.parse()).collect();

        match octets {
            Ok(octets) if octets.len() == 4 => Ok(Ipv4Address {
                octets: [octets[0], octets[1], octets[2], octets[3]],
            }),
            _ => Err(ParseIpAdrrError {}),
        }
    }
}

// Custom serialization for Ipv4Address
impl Serialize for Ipv4Address {
    fn serialize<S>(&self, serializer: S) -> Result<S::Ok, S::Error>
    where
        S: Serializer,
    {
        serializer.serialize_str(&self.to_string())
    }
}

// Custom deserialization for Ipv4Address
impl<'de> Deserialize<'de> for Ipv4Address {
    fn deserialize<D>(deserializer: D) -> Result<Self, D::Error>
    where
        D: Deserializer<'de>,
    {
        let s = String::deserialize(deserializer)?;
        s.parse().map_err(serde::de::Error::custom)
    }
}

impl Distribution<Ipv4Address> for Standard {
    fn sample<R: Rng + ?Sized>(&self, rng: &mut R) -> Ipv4Address {
        Ipv4Address {
            octets: [rng.gen(), rng.gen(), rng.gen(), rng.gen()],
```

```
        }
    }
}

#[derive(Serialize, Deserialize, Debug, Clone, Copy)]
pub struct Data {
    pub ip: Ipv4Address,
    pub os: OS,
    pub protocol: Protocol,
}

impl fmt::Display for Data {
    fn fmt(&self, f: &mut fmt::Formatter<'_>) -> fmt::Result {
        write!(
            f,
            "Data␣{{␣ip:␣{},\tos:␣{:?},\tprotocol:␣{:?}\t␣}}",
            self.ip, self.os, self.protocol
        )
    }
}

impl PartialEq for Data {
    fn eq(&self, other: &Self) -> bool {
        self.ip == other.ip
    }
}

impl Eq for Data {}

impl PartialOrd for Data {
    fn partial_cmp(&self, other: &Self) -> Option<Ordering> {
        Some(self.cmp(other))
    }
}

impl Ord for Data {
    fn cmp(&self, other: &Self) -> Ordering {
        self.ip.cmp(&other.ip)
    }
}

impl Distribution<Data> for Standard {
    fn sample<R: Rng + ?Sized>(&self, rng: &mut R) -> Data {
        Data {
            ip: rng.gen(),
            os: rng.gen(),
            protocol: rng.gen(),
        }
    }
}

pub fn generate_unique_ip_addresses(n: usize) -> Vec<Ipv4Address> {
    let mut rng = rand::thread_rng();
    let mut res = Vec::new();

    while res.len() < n {
        let addr: Ipv4Address = rng.gen();
        if !res.contains(&addr) {
            res.push(addr);
        }
    }

    res
}

pub fn generate_unique_data(n: usize) -> Vec<Data> {
    let mut rng = rand::thread_rng();
    generate_unique_ip_addresses(n)
        .into_iter()
        .map(|ip| Data {
            ip,
            os: rng.gen(),
            protocol: rng.gen(),
        })
        .collect::<Vec<Data>>()
}
```

```
220
221 pub fn default_unique_data(n: usize) -> Vec<Data> {
222     let mut i = 0;
223     let mut vec = Vec::<Data>::new();
224     for a in 204..=255 {
225         for b in 111..=255 {
226             for c in 0..=255 {
227                 for d in 0..=255 {
228                     if i >= n {
229                         return vec;
230                     }
231
232                     let ip = Ipv4Address {
233                         octets: [a, b, c, d],
234                     };
235                     let data = Data {
236                         ip,
237                         os: OS::Arch,
238                         protocol: Protocol::TCP,
239                     };
240                     vec.push(data);
241                     i += 1;
242                 }
243             }
244         }
245     }
246     vec
247 }
248
249 pub fn generate_test_file(file: &mut std::fs::File, n: usize) {
250     let data = generate_unique_data(n);
251     let serialized_data = serde_json::to_string(&data).unwrap();
252
253     file.write_all(serialized_data.as_bytes())
254         .expect("Unable␣write␣into␣file");
255 }
256
257 pub fn read_data_from_file(file: &mut std::fs::File) -> Vec<Data> {
258     let mut contents = String::new();
259     file.read_to_string(&mut contents)
260         .expect("Unable␣read␣from␣file");
261
262     serde_json::from_str(&contents).unwrap()
263 }
```

```
[
    { "ip": "245.181.224.156", "os": "Ubuntu", "protocol": "FTP" },
    { "ip": "106.149.27.208", "os": "Windows", "protocol": "HTTP" },
    { "ip": "117.33.232.223", "os": "FreeBSD", "protocol": "FTP" },
    { "ip": "94.80.10.83", "os": "CentOS", "protocol": "TCP" },
    { "ip": "49.187.234.231", "os": "FreeBSD", "protocol": "FTP" },
    { "ip": "73.27.8.77", "os": "OpenBSD", "protocol": "FTP" },
    { "ip": "10.9.182.166", "os": "MacOS", "protocol": "UDP" },
    { "ip": "162.17.217.219", "os": "CentOS", "protocol": "TCP" },
    { "ip": "169.238.43.52", "os": "Debian", "protocol": "SMTP" },
    { "ip": "192.21.25.122", "os": "FreeBSD", "protocol": "FTP" },
    { "ip": "71.64.226.188", "os": "NixOS", "protocol": "HTTP" },
    { "ip": "102.202.182.48", "os": "MacOS", "protocol": "UDP" },
    { "ip": "255.181.97.192", "os": "NixOS", "protocol": "SMTP" },
    { "ip": "73.145.52.114", "os": "Ubuntu", "protocol": "UDP" },
    { "ip": "71.25.51.193", "os": "NixOS", "protocol": "FTP" },
    { "ip": "139.54.31.21", "os": "CentOS", "protocol": "HTTP" },
    { "ip": "74.48.191.236", "os": "NixOS", "protocol": "TCP" },
    { "ip": "19.244.41.22", "os": "Arch", "protocol": "QUIC" },
    { "ip": "35.61.205.97", "os": "MacOS", "protocol": "HTTP" },
    { "ip": "171.188.179.151", "os": "Debian", "protocol": "HTTP" }
]
```


Рис. 4. Приклад збережених даних

3.4 Приклад роботи програми

Для перевірки працездатності напишемо програму, яка буде порівнювати хеш-таблицю, двійковий та лінійний пошук у лінійному списку. А також програму для виводу вмісту таблиці.
Код програми для перевірки:

```
use csv::Writer;

use super::data::*;
use crate::libs::hash_table::HashTable;
use crate::libs::list::double_linked_list::List;
use crate::libs::search::binary_search;
use crate::libs::search::linear_search;
use crate::libs::search::logger::{Logger,        Metrics};

pub fn test()   {
    let  n = 10;
    let  file_name = format!("data_{n}.json");

    //  let  mut file  = std::fs::File::create(&file_name).expect("Unable       to  create  file");
    //  generate_test_file(&mut   file,   n);

    //  let  mut file  = std::fs::File::open(&file_name).expect("Unable       to  create  file");
    //  let  mut data_vec = read_data_from_file(&mut file);

    let  mut data_vec = generate_unique_data(n);
    //  let  mut data_vec = default_unique_data(n);
    println!("generated\n");

    let  mut hash_table = HashTable::<Ipv4Address, Data>::new_with_capacity(n + 1);
    let  mut list  = List::<Data>::new();

    data_vec.sort();
```

```
28
29      for data in data_vec {
30          hash_table.insert(data.ip,      data);
31          list.push(data);
32      }
33
34      println!("{}",    hash_table);
35  }
36
37  fn perf_test(n:   usize,   elem_rate:  f64)  -> (Metrics,   Metrics,   Metrics)  {
38      let  elem_id = (n  as  f64  * elem_rate)  as  usize;
39      let  mut data_vec = generate_unique_data(n);
40
41      let  mut hash_table  = HashTable::<Ipv4Address,  Data>::new_with_capacity(n  + 1);
42      let  mut list   = List::<Data>::new();
43
44      data_vec.sort();
45      let  elem = data_vec[elem_id];
46
47      for data in data_vec {
48          hash_table.insert(data.ip,      data);
49          list.push(data);
50      }
51
52      let  mut logger = Logger::new();
53
54      let  res1  = linear_search(&mut list,    elem,  &mut logger);
55      let  m1 = logger.get_metrics();
56
57      let  res2  = binary_search(&mut list,    elem,  &mut logger);
58      let  m2 = logger.get_metrics();
59
60      logger.start();
61      let  res3  = hash_table.get(elem.ip);
62      logger.end();
63      let  m3 = logger.get_metrics();
64
65      (m1,  m2, m3)
66  }
67
68  fn perf()  {
69      let  out_file   = "perf_lab10.csv".to_string();
70      let  test_cases:   Vec<usize> = vec![20,   1000,  5000,  10000,  50000];
71      let  elem_rate  = 0.66666666;
72      //  let  elem_rate  = 0.1;
73
74      let  mut wtr = Writer::from_path(out_file).unwrap();
75      wtr.write_record(&[
76          "n",
77          "linear_compares",
78          "linear_time",
79          "binary_compares",
80          "binary_time",
81          "hash_compares",
82          "hash_time",
83      ])
84      .unwrap();
85
86      for n in test_cases  {
87          let  (m1,  m2, m3) = perf_test(n,    elem_rate);
88          println!("Linear␣Search:␣{m1}");
89          println!("Binary␣Search:␣{m2}");
90          println!("Hash␣table:␣{m3}\n");
91
92          let  Metrics(c1,   s1,  t1)  = m1;
93          let  Metrics(c2,   s2,  t2)  = m2;
94          let  Metrics(c3,   s3,  t3)  = m3;
95          wtr.write_record(&[
96              n.to_string(),
97              c1.to_string(),
98              t1.as_nanos().to_string(),
99              c2.to_string(),
100             t2.as_nanos().to_string(),
101             c3.to_string(),
102             t3.as_nanos().to_string(),
103         ])
```

```
104             .unwrap();
105     }
106 }
107
108 pub fn main() {
109     test();
110     //  perf();
111 }
```

```
214.175.089.249 ⇒ Data { ip: 214.175.089.249,  os: CentOS,    protocol: UDP   }  |  [ buckets id: 0;  linked list position: 0 ]
033.005.216.015 ⇒ Data { ip: 033.005.216.015,  os: OpenBSD,  protocol: TCP   }  |  [ buckets id: 1;  linked list position: 0 ]
031.169.207.112 ⇒ Data { ip: 031.169.207.112,  os: OpenBSD,  protocol: HTTP  }  |  [ buckets id: 2;  linked list position: 0 ]
175.153.160.227 ⇒ Data { ip: 175.153.160.227,  os: CentOS,    protocol: FTP   }  |  [ buckets id: 2;  linked list position: 1 ]
031.086.136.121 ⇒ Data { ip: 031.086.136.121,  os: Windows,   protocol: UDP   }  |  [ buckets id: 3;  linked list position: 0 ]
112.201.136.084 ⇒ Data { ip: 112.201.136.084,  os: Arch,      protocol: FTP   }  |  [ buckets id: 5;  linked list position: 0 ]
179.233.161.057 ⇒ Data { ip: 179.233.161.057,  os: OpenBSD,  protocol: FTP   }  |  [ buckets id: 6;  linked list position: 0 ]
201.176.007.180 ⇒ Data { ip: 201.176.007.180,  os: Windows,   protocol: UDP   }  |  [ buckets id: 6;  linked list position: 1 ]
232.126.023.129 ⇒ Data { ip: 232.126.023.129,  os: Arch,      protocol: FTP   }  |  [ buckets id: 6;  linked list position: 2 ]
005.103.072.108 ⇒ Data { ip: 005.103.072.108,  os: FreeBSD,   protocol: FTP   }  |  [ buckets id: 8;  linked list position: 0 ]
```

Рис. 5. Приклад роботи

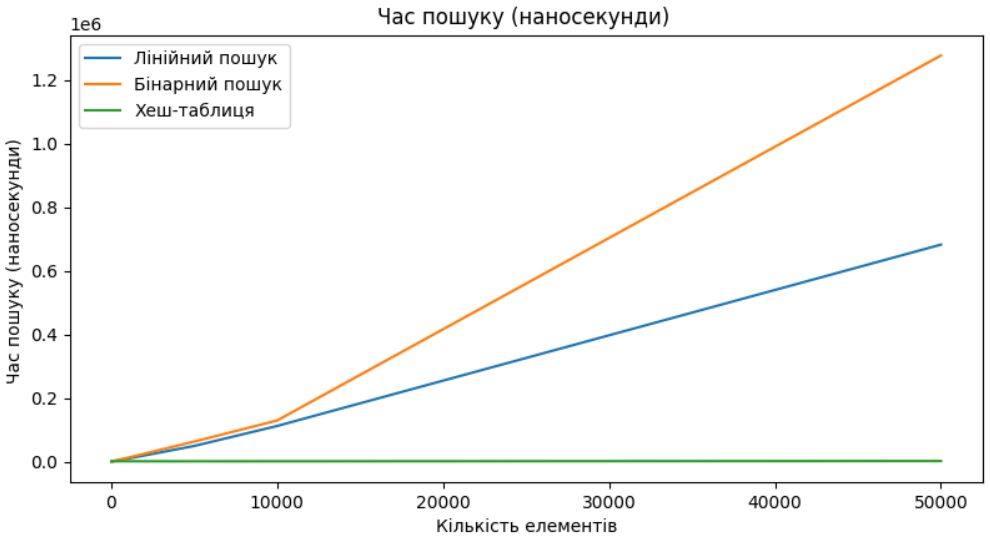

Рис. 6. Залежність часу пошуку від кількості елементів

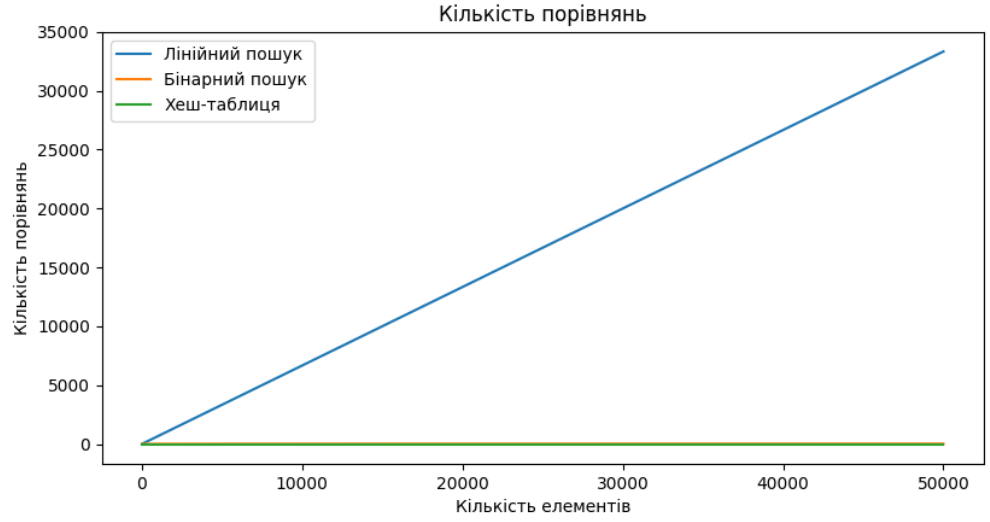

Рис. 7. Залежність кількості порівнянь від кількості елементів

4 Висновки

В ході виконання лабораторної робити було створено хеш-таблицю. Також було порівняно хеш-таблицю з лінійним та бінарним пошуком в списку. За результатами порівняння було виявлено, що хеш-таблиця набагато ефективніше за інші способи а також вона не потребує відсортований набір даних як бінарний пошук.