Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра ІПІ

Звіт

з лабораторної роботи №7 з дисципліни
«Алгоритми та структури даних 2. Структури даних»

„ **Проектування і аналіз алгоритмів пошуку**”

Виконав ІП-45 Янов Богдан Євгенійович

Перевірив Соколовський Владислав Володимирович

Київ 2025

Лабораторна робота №7

Проектування і аналіз алгоритмів пошуку

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

Псевдокод алгоритмів

```
struct Entry {
	key
	value
}

struct HashTable {
	buckets
	size
}

pjw_hash(s) {
	hash = 0

	for c in s.chars() {
		hash = (hash << 4) + (c as u64)
		let test = hash & 0xF0000000
		if test != 0 {
			hash ^= test >> 24
			hash &= !test
		}
	}

	hash
}

insert(table, key, value) {
	let mut idx = first hash of key
	let step = second hash of key

	loop {
		match table.buckets[idx] {
			Some(entry) if entry.key == key => {
			entry.value = value
			return
			}
			None => {
			table.buckets[idx] = Entry { key, value }
			return
			}
			_ => {
			idx = (idx + step) % self.size
			}
```

```
            }
      }
}

search(table, key) {
      comparisons = 0
      idx = first hash of file
      step = second hash of key

      loop {
            comparisons += 1
            match table.buckets[idx] {
                  Some(entry) if entry.key == key => {
                  return (&entry.value, comparisons)
                  }
                  Some(_) => {
                  idx = (idx + step) % self.size
                  }
                  None => {
                  return (None, comparisons)
                  }
            }
      }
}
```

Вихідний код

```
use rand::{Rng, distr::Alphanumeric};

fn pjw_hash(s: &str) -> u64 {
    let mut hash = 0;

    for c in s.chars() {
        hash = (hash << 4) + (c as u64);
        let test = hash & 0xF0000000;
        if test != 0 {
            hash ^= test >> 24;
            hash &= !test;
        }
    }

    hash
}

#[derive(Clone, Debug)]
```

```rust
struct Entry {
    key: String,
    value: String,
}

#[derive(Debug)]
pub struct HashTable {
    buckets: Vec<Option<Entry>>,
    size: usize,
}

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

    fn h1(&self, key: &str) -> usize {
        (pjw_hash(key) as usize) & (self.size - 1)
    }

    fn h2(&self, key: &str) -> usize {
        1 + ((pjw_hash(key) as usize) % (self.size - 1))
    }

    pub fn insert(&mut self, key: String, value: String) {
        let mut idx = self.h1(&key);
        let step = self.h2(&key);

        loop {
            match &mut self.buckets[idx] {
                Some(entry) if entry.key == key => {
                    entry.value = value;
                    return;
```

```
                }
                None => {
                    self.buckets[idx] = Some(Entry { key, value });
                    return;
                }
                _ => {
                    idx = (idx + step) % self.size;
                }
            }
        }
    }

    pub fn search(&self, key: &str) -> (Option<&str>, usize) {
        let mut comparisons = 0;
        let mut idx = self.h1(key);
        let step = self.h2(key);

        loop {
            comparisons += 1;
            match &self.buckets[idx] {
                Some(entry) if entry.key == key => {
                    return (Some(&entry.value), comparisons);
                }
                Some(_) => {
                    idx = (idx + step) % self.size;
                }
                None => {
                    return (None, comparisons);
                }
            }
        }
    }
}

fn generate_random_string(length: usize) -> String {
    rand::rng()
```

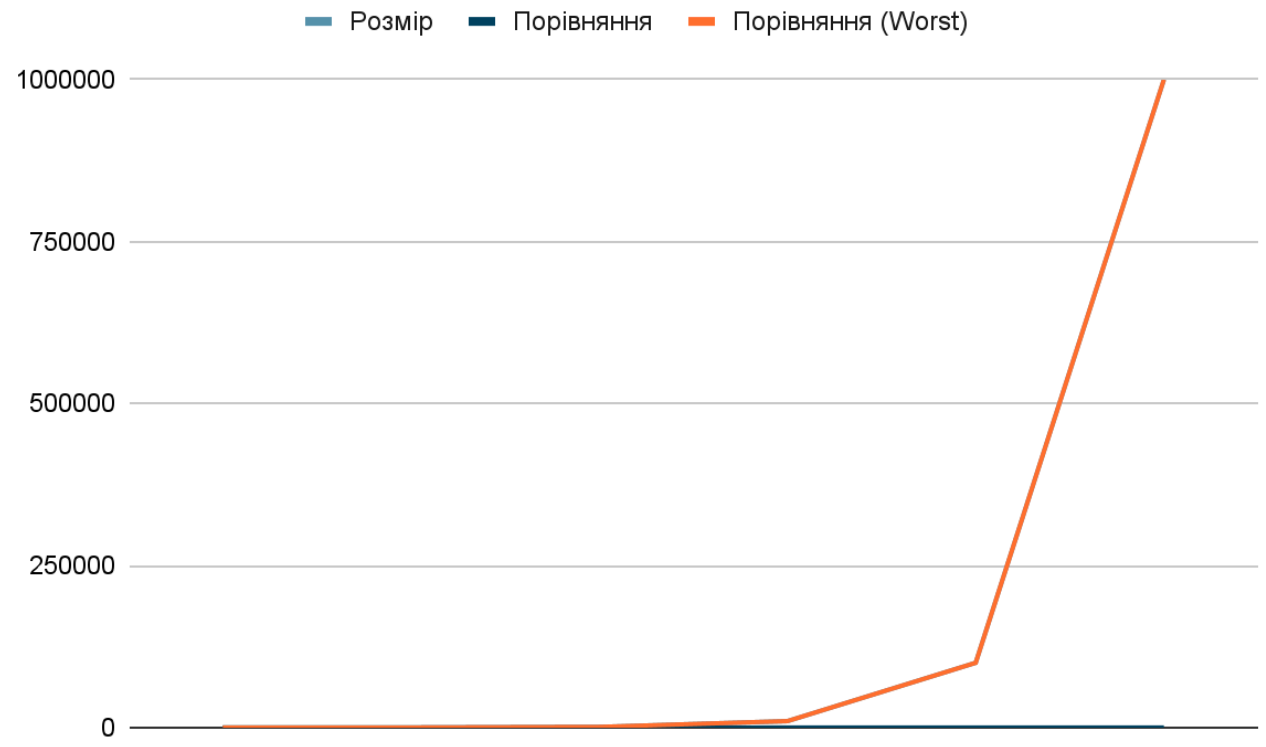
```
        .sample_iter(&Alphanumeric)
        .take(length)
        .map(char::from)
        .collect()
}

fn main() {
    let len = 1_000_000;
    let mut ht = HashTable::new(len * 5);

    for i in 0..len {
        let key = format!("key_{}", i);
        let value = generate_random_string(5);
        ht.insert(key, value);
    }

    let key = "key_42";
    let (res, comps) = ht.search(key);
    match res {
        Some(val) => {
            println!("Found `{}`: \"{}\" (comparisons: {})", key,
val, comps);
        }
        None => {
            println!("Key `{}` not found (comparisons: {})", key,
comps);
        }
    }
}
```

Приклад роботи

```
λ Partur lab7 → λ git master* → cargo run --release
    Finished `release` profile [optimized] target(s) in 0.01s
     Running `target/release/lab7`
Found `key_42`: "C08d5" (comparisons: 1)
λ Partur lab7 → λ git master* →
```

Часові характеристики

Розмір	Кількість порівнянь	Кількість порівнянь (Worst)
10	1	10
100	1	100
1 000	1	1 000
10 000	1	10 000
100 000	1	100 000
1 000 000	1	1 000 000



Висновок

У ході виконання роботи було реалізовано хеш-таблицю з використанням хеш-функції PJW-32 та методом розв’язання колізій відкритою адресацією з подвійним хешуванням. Реалізація дозволяє зберігати та ефективно шукати пари ключ-значення рядкового типу. Проведені тести показали, що кількість порівнянь при пошуку значно менша, ніж у звичайному масиві. Таким чином, обрана структура забезпечує добру продуктивність і може бути рекомендована для задач із великим обсягом даних та вимогами до швидкого доступу за ключем.