Лабораторна робота №4

Робота з хеш-таблицями

Метою виконання лабораторної роботи є набуття практичних навичок із проектування, реалізації, тестування та аналізу алгоритмів створення і обробки хеш-таблиць.

Зміст:

Розглядається хеш-таблиці із відкритим (open hashing) та закритим (closed hashing) хешуванням як структури для побудови словників. *Відкрите хешування* також називають *хешуванням з роздільними ланцюжками* (separate chaining), а *закрите хешування* також називають *хешуванням з відкритою адресацією* (open addressing) Розглядаються властивості цих структур для виконання словникових операцій, а саме вставки, видалення та пошуку елемента.

1.Основні відомості про хеш-таблиці

Хеш-таблиця – це структура даних, що дозволяє ефективно реалізовувати словники (часто середній час пошуку елементу O(1)). Узагальнює поняття масиву. Наявність прямої індексації елементів масиву забезпечує доступ до довільної позиції у масиві за час O(1). Використання хеш-таблиці дозволяє забезпечити середній час пошуку елемента O(1) за умови рівномірного розподілу ключів, хоча вимоги до пам'яті залишаються пропорційними кількості збережених елементів, тобто O(n).

При використанні хеш-таблиці елемент з ключом k зберігається у комірці масиву $h(k)$ за рахунок використання *хеш-функції* h, що обчислює комірку для заданого ключа k.

Хеш-функція – це така функція $h: U \to \{0,1,\dots,m-1\}$, що відображає сукупність ключів U на комірки хеш-таблиці $T[0..m-1]$. Кажуть, що елемент з ключем k хешується в комірку $h(k)$, а величину $h(k)$ називають *хеш-значенням* ключа k. Звичайно розмір хеш-таблиці значно менший U . Хеш-функція зменшує робочий діапазон індексів масиву і тому замість розміру U значень обходимося масивом розміром m (рис. 4.1)

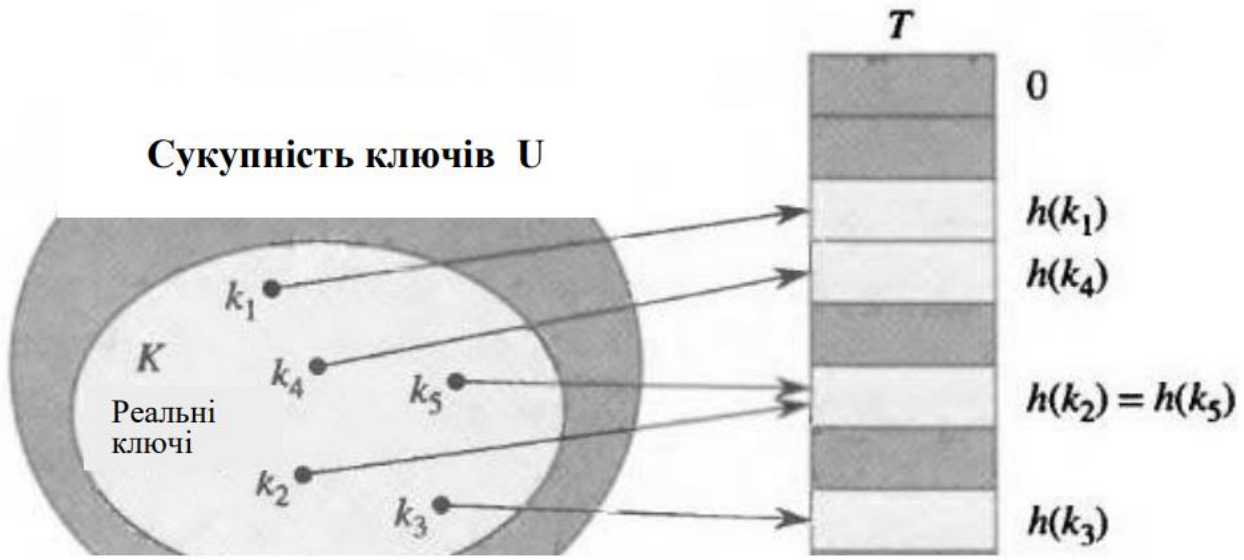

Рисунок 4.1 – Хешування до хеш-таблиці

Ситуацію, коли два ключа хешуються в одну комірку називають *колізією*. Є різні технології вирішення колізій. За допомогою підходящого вибору хеш-функції намагаються їх мінімізувати. Але так як $|U| > m$, то повинно існувати не менше двох ключів, що мають однакове хеш-значення.

2.Відкрите хешування (хешуванням з роздільними ланцюжками)

Найпростішим методом вирішення колізій є метод ланцюжків (separate chaining), за яким розташовують всі елементи, що хешовані до одної комірки, у зв'язаний список. Це відкрите (зовнішнє) хешування. Тоді, наприклад, комірка *j* містить покажчик на заголовок списку всіх елементів, хеш-значення яких дорівнює *j*, а якщо таких елементів немає, комірка містить значення NIL.
CHAINED-HASH-SEARCH(T, k)
Пошук елементу з ключом k у списку $T[h(k)]$
CHAINED-HASH-INSERT(T, x)
Вставка x до заголовку списку $T[h(x, key)]$
CHAINED-HASH-DELETE(T, x)
Видалення x із списку $T[h(x, key)]$

Припустимо, що елемент, що вставляємо до таблиці, відсутній в ній. Тоді час, необхідний для вставки – O(1). Інакше перевіряємо наявність, що тягне за собою додаткову вартість виконання пошуку елементу з відповідним ключом перед вставкою. Тоді час роботи буде пропорційним довжині списку. Вилучення елементу матиме вартість таку ж часову вартість.
Проведемо більш докладний аналіз алгоритмічної складності хешування з ланцюжками. Припустимо, маємо хеш-таблицю T з m комірками, в яких зберігаються n елементів. Тоді коефіцієнтом заповнення таблиці буде $\alpha = n/m$, який може бути менше, дорівнювати або більше 1. У найгіршому випадку всі n ключів можуть бути хешовані в одну комірку. Тоді маємо

список довжиною *n*. Тому час пошуку для найгіршого випадку буде O(*n*) плюс час на обчислення хеш-функції, а тоді використання хеш-функції буде невигідним. У середньому випадку продуктивність хешування залежить від того, як добре хеш-функція розподіляє множину ключів, що зберігається, за *m* комірками у середньому.

3. Приклад побудови відкритої хеш-таблиці

Розглянемо наступний список із 8 слов (англійське прислів'я, використане як приклад Дональдом Кнутом)

key	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
№ слова	1	2	3	4	5	6	7	8

В якості хеш-функції будемо використовувати просту хеш-функцію для строк – тобто підсумувати номер позиції букви в англійському алфавіті (див. рис. 4.2) та обчислювати залишок від ділення цієї суми на 13 (Табл.4.1)

$A-1$ $J-10$ $S-19$
$B-2$ $K-11$ $T-20$
$C-3$ $L-12$ $U-21$
$D-4$ $M-13$ $V-22$
$E-5$ $N-14$ $W-23$
$F-6$ $O-15$ $X-24$
$G-7$ $P-16$ $Y-25$
$H-8$ $Q-17$ $Z-26$
$I-9$ $R-18$

Рисунок – 4.2 Номера позиції англійської букви в експерименті Кнута

Тоді отримаємо
h(A)= 1 mod 13 = 1
h(FOOL) = (6+15+15+12) mod 13=9
h(AND)=6
h(HIS)=10
h(MONEY)=7
h(ARE) = (1+18+5) mod 13= 11
h(SOON) = (19+15+15+14) mod 13= 11
h(PARTED)=12

Таблиця 4.1 Значення хеш-функції

key	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
№ слова	1	2	3	4	5	6	7	8
h(key)	1	9	6	10	7	11	11	12

Тоді побудована хеш-таблиця з роздільними ланцюжками буде виглядати наступним чином (Табл. 4.2)

Таблиця 4.2 Вигляд хеш-таблиці при хешуванні з ланцюжками

0	1	2	3	4	5	6	7	8	9	10	11	12
	↓ A					↓ AND	↓ MONEY		↓ FOOL	↓ HIS	↓ ARE ↓ SOON	↓ PARTED

Припустимо, що ми хочемо знайти слово KID обчислюємо його хеш-функцію h(KID)= (11+9+4) mod 13=11. Оскільки список, який під'єднано до комірки 11 не є порожнім – то він може містити відповідний ключ. Однак із-за можливих колізій ми не знаємо точної відповіді поки не знайдемо його в списку або не переглянемо список до кінця. Після порівняння KID із ARE, а потім із SOON можемо сказати, що KID в таблиці відсутній.
Додавання та видалення елементів виконується аналогічним чином. Додавання виконується кінець списку, а видалення відбувається шляхом пошуку елемента з подальшим видаленням із списку. Таким чином ефективність роботи пошуку в хеш-таблиці залежить від довжина зв'язних списків, яка в свою чергу залежить від довжини словника, хеш-таблиці та вигляду хеш-функції.

4. Реалізація мовою Python алгоритму хешуванням з роздільними ланцюжками та візуалізації відкритої хеш-таблиці

Особливості реалізації алгоритмів хешуванням з роздільними ланцюжками `build_open_hash_table(words: list, m: int)` та візуалізації відкритої хеш-таблиці `display_hash_table(table: list)` мовою Python показано на Лістингу 4.1.

Лістинг 4.1 – Python-код реалізації алгоритмів хешуванням з роздільними ланцюжками та візуалізації відкритої хеш-таблиці

```
# Константа розміру таблиці
M = 13
# Список вхідних слів
WORDS = ["A", "FOOL", "AND", "HIS", "MONEY", "ARE", "SOON", "PARTED"]

# Словник позицій
LETTER_POSITIONS = {
    'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5, 'F': 6, 'G': 7,
    'H': 8, 'I': 9, 'J': 10, 'K': 11, 'L': 12, 'M': 13,
    'N': 14, 'O': 15, 'P': 16, 'Q': 17, 'R': 18, 'S': 19,
    'T': 20, 'U': 21, 'V': 22, 'W': 23, 'X': 24, 'Y': 25, 'Z': 26
}

def simple_hash_from_map(key: str) -> int:
    """
    Хеш-функція: h(k) = (сума позицій букв) mod 13.
    Використовує пряме відображення позицій з LETTER_POSITIONS.
    """
    sum_of_positions = 0

    # Ключі в словнику позицій
    for char in key:
        # Отримання позиції. Якщо символ не знайдено (наприклад, не літера),
повертаємо 0.
        position = LETTER_POSITIONS.get(char, 0)
        sum_of_positions += position

    # Обчислення фінальної хеш-адреси
    hash_address = sum_of_positions % M
    return hash_address
```

```
def build_open_hash_table(words: list, m: int) -> list:
    """
    Будує хеш-таблицю з ланцюжками (списками).
    """
    # 1. Ініціалізація таблиці: M порожніх ланцюжків
    hash_table = [[] for _ in range(m)]

    # 2. Обробка кожного слова
    for word in words:
        address = simple_hash_from_map(word)
        # Додавання слова до відповідного ланцюжка
        hash_table[address].append(word)

    return hash_table

def display_hash_table(table: list):
    """Виводить хеш-таблицю у зручному форматі."""
    print("\n--- Результат хешування (Таблиця M=13) ---")
    for i, chain in enumerate(table):
        print(f"Індекс {i:02d}: {chain}")

# Виконання:
hash_table = build_open_hash_table(WORDS, M)
display_hash_table(hash_table)
```

Результати виконання Python-коду, який наведено в Лістінгу 4. 1 показано в таблиці 4.3

Таблиця 4.3 Результати побудови відкритої хеш-таблиці

```
--- Результат хешування (Таблиця M=13) ---
Індекс 00: []
Індекс 01: ['A']
Індекс 02: []
Індекс 03: []
Індекс 04: []
Індекс 05: []
Індекс 06: ['AND']
Індекс 07: ['MONEY']
Індекс 08: []
Індекс 09: ['FOOL']
Індекс 10: ['HIS']
Індекс 11: ['ARE', 'SOON']
Індекс 12: ['PARTED']
```

Як бачимо результат побудови відкритої хеш-таблиці за допомогою Python-коду (див. Табл 4.3) співпадає із таблицею побудованою в ручному режимі. (Див. Таблицю 4.2)

5. Закрите хешування (хешування з відкритою адресацією)

При використанні цього методу безпосередньо в хеш-таблиці зберігаються всі елементи, а якщо комірка пуста, то вона має містити значення NULL. Тому таблиця може стати повністю заповненою (коефіцієнт заповнення α не може перевищувати 1) і елемент вставити буде неможливо. При виконанні пошуку елементу перевіряються всі комірки таблиці, доки не знайдемо шуканий елемент або не впевнимось у відсутності елементу в хеш-таблиці. Перевагою методу економія пам’яті за рахунок відмови від покажчиків. Використовують методику повторного хешування. За нею для виконання вставки елементу послідовно перевіряють (досліджують) комірки

таблиці, доки не знайдеться пуста комірка, в яку й розміщують ключ. Тобто, замість фіксованого порядку дослідження комірок 0, 1, …, m – 1 (що дає найгірший час O(m)), послідовність досліджуваних комірок залежить від ключа, що вставляють до хеш-таблиці. З цією метою використовується *розширене* визначення хеш-функції, де в якості другого аргументу включений номер дослідження (який починається з 0): $h:U \times\{0,1,...,m-1\}\rightarrow\{0,1,...,m-1\}$. В методі відкритої адресації потрібно, щоб для кожного ключа k послідовність досліджень $h(k,0),h(k,1),...,h(k, m-1)$ являла собою перестановку множини 0,1,...,m-1 такої, щоб могли бути переглянутими всі комірки таблиці. Для випадку, коли елементи в хеш-таблиці T є ключами без супутньої інформації й ключ k тотожній елементу, що містить ключ k, *процедура вставки* ключа k може мати наступний вигляд:

```
HASH-INSERT(T, k)
1 i = 0
2       repeat
3               j = h(k, i)
4               if T[j] == NIL
5                       T[j] = k
6               return j
7               else i = i + 1
8       until i == m
9 error «переповнення хеш-таблиці»
```

У *процедурі пошуку* ключа k досліджується та ж послідовність комірок, що й при його вставці. Але, якщо зустрічається пуста комірка, то пошук завершується невдачею (якщо припустимо, що елементи не видалялися):

```
HASH-SEARCH(T, k)
1 i = 0
2       repeat
3               j = h(k, i)
4               if T[j] == k
5               return j
6               else i = i + 1
7       until T[j] == NIL або i == m
8       return NIL
```

Процедура видалення з хеш-таблиці більш складна. При видаленні ключа з комірки і хеш-таблиці її не можна просто помітити пустим значенням NIL(це негативно впливає на пошук елементу), таку комірку помічають спеціальним значенням DELETED. Крім того, потрібно дещо змінити наведену процедуру вставки, щоб в ній комірка із значенням DELETED розглядалася як пуста. Але час пошуку тоді перестає залежати від коефіцієнта заповнення α і тому при необхідності видалення з таблиці користуються методом ланцюжків в якості методу вирішення колізій.

6. Приклад побудови закритої хеш-таблиці

В випадку закритого хешування всі ключі зберігаються безпосередньо в хеш-таблиці, що призводить до вимоги про розмір хеш-таблиці, який повинен дорівнювати кількості ключів. Для вирішення колізій використовують лінійне дослідження (linear probing) коли в випадку виникнення колізій перевіряються всі комірки одна за одною, поки елемент не буде знайдено. Якщо в процесі перевірки досягається кінець таблиці то пошук переходить до першої комірки таблиці, яку розглядаємо як циклічний масив. Для попереднього прикладу (Табл 4.1) побудова хеш-таблиці з відкритою адресацією виглядає наступним чином:

Таблиця 4.4 Вигляд хеш-таблиці з відкритою адресацією

	хеш-таблиця з відкритою адресацією												
Insert key	0	1	2	3	4	5	6	7	8	9	10	11	12
1		a											
2		a								fool			
3		a					and			fool			
4		a					and			fool	his		
5		a					and	money		fool	his		
6		a					and	money		fool	his	are	
7		a					and	money		fool	his	are	
7		a					and	money		fool	his	are	soon
8		a					and	money		fool	his	are	soon
9	parted	a					and	money		fool	his	are	soon

Для прикладу знайдемо слово LIT в таблиці. Розраховуємо хеш-функцію h(LIT) = (12+9+20) mod 13= 2. Звертаємось до комірки за номером 2 – вона порожня, тобто слово відсутнє. В випадку пошуку слова KID обчислюємо його хеш-функцію h(KID)= (11+9+4) mod 13=11. Звертаємось до комірки 11 (ARE), ключ не співпадає, звертаємось до наступної комірки 12 (SOON), не має співпадіння, циклічно переходимо до комірки 0 (PARTED) не має, порівнюємо із 1 (A) і тільки коли потрапляємо до порожньої комірки робимо висновок про відсутність слова KID.

Операція видалення є більш складною. Якщо необхідно видалити ключ ARE (11) – комірка стає порожньою, то ми втратимо доступ ключа SOON (12). Тому необхідно «відкладене видалення», тобто комірку 11 в випадку зі ключом ARE відмічають спеціальним чином (DELETED), щоб відрізнити від комірок, які взагалі не були зайняти.

7. Реалізація мовою Python алгоритмів хешуванням з відкритою адресацією та візуалізації закритої хеш-таблиці

Реалізація алгоритмів хешуванням з з відкритою адресацією `build_closed_hash_table(words: list, m: int)` та візуалізації закритої хеш-

таблиці `display_hash_table(table: list)` мовою Python показано за допомогою Лістингу 4.2.

Лістинг 4.2 – Python-код реалізації алгоритмів хешуванням з відкритою адресацією та візуалізації закритої хеш-таблиці

```
# Константа розміру таблиці
M = 13
# Список вхідних слів
WORDS = ["A", "FOOL", "AND", "HIS", "MONEY", "ARE", "SOON", "PARTED"]

# Словник позицій (на основі наданого малюнка)
LETTER_POSITIONS = {
    'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5, 'F': 6, 'G': 7,
    'H': 8, 'I': 9, 'J': 10, 'K': 11, 'L': 12, 'M': 13,
    'N': 14, 'O': 15, 'P': 16, 'Q': 17, 'R': 18, 'S': 19,
    'T': 20, 'U': 21, 'V': 22, 'W': 23, 'X': 24, 'Y': 25, 'Z': 26
}

def primary_hash(key: str) -> int:
    """h(k) = (сума позицій букв) mod M. Первинна хеш-функція."""
    sum_of_positions = 0
    for char in key:
        position = LETTER_POSITIONS.get(char, 0)
        sum_of_positions += position

    return sum_of_positions % M

def build_closed_hash_table(words: list, m: int) -> list:
    """
    Будує хеш-таблицю з відкритою адресацією, використовуючи лінійне дослідження.
    """
    # 1. Ініціалізація таблиці: M порожніх слотів (використовуємо None як "порожній")
    hash_table = [None] * m
    inserted_count = 0

    # 2. Обробка кожного слова
    for word in words:
        # Крок 2a: Обчислення початкової адреси
        start_address = primary_hash(word)
        address = start_address

        # Крок 2b: Лінійне дослідження (Linear Probing)
        # Цикл гарантує, що ми не будемо шукати нескінченно довго у повній таблиці
        for i in range(m):
            # h(k, i) = (h(k) + i) mod M
            address = (start_address + i) % m

            # Перевірка, чи комірка вільна
            if hash_table[address] is None:
                # Вставлення ключа
                hash_table[address] = word
                inserted_count += 1
                break

            # Якщо комірка зайнята, продовжуємо цикл (наступна ітерація і збільшить крок на 1)

        else:
```

```
            # Цей блок виконується, якщо цикл завершився без 'break'
(таблиця повна)
            print(f"Помилка: Таблиця заповнена. Не вдалося додати
слово: {word}")

    return hash_table

def display_hash_table(table: list):
    """Виводить хеш-таблицю у зручному форматі."""
    print("\n--- Хеш-таблиця (Відкрита адресація, M=13) ---")
    print("Індекс | Слово")
    print("-------|-------")
    for i, item in enumerate(table):
        # Виводимо ключі або позначку, що комірка порожня
        value = item if item is not None else "(NULL)"
        print(f"{i:02d}     | {value}")

# Виконання:
hash_table = build_closed_hash_table(WORDS, M)
display_hash_table(hash_table)
```

Результати виконання Python-коду, який наведено в Лістінг 4. 2 показано в таблиці 4.5

Таблиця 4.5 Результати побудови хеш-таблиці з відкритою адресацією

```
Хеш-таблиця (Відкрита адресація, M=13) ---
Індекс | Слово
-------|-------
00     | PARTED
01     | A
02     | (NULL)
03     | (NULL)
04     | (NULL)
05     | (NULL)
06     | AND
07     | MONEY
08     | (NULL)
09     | FOOL
10     | HIS
11     | ARE
12     | SOON
```

Як бачимо результат побудови закритою хеш-таблиці за допомогою Python-коду (див. Табл 4.5) співпадає із таблицею побудованою в ручному режимі. (Див. таблицю 4.4)

8. Хеш-функції. Побудова некриптографічних хеш-функцій

При побудові якісних хеш-функцій дуже корисною є інформація про розподіл ключів. Вірним підходом при побудові є підбір хеш-функції так, щоб вона не корелювала із закономірностями, яким можуть підкорятися існуючі дані. Іноді до хеш-функцій можуть висувати більш суворі вимоги, ніж за простого рівномірного хешування.

Нехай сукупність ключів можна представити множиною цілих невід'ємних чисел N.

1. *Побудова хеш-функції методом ділення*: $h(k) = k \bmod m$ (відображення ключа k до одної з m комірок за допомогою вказаного *обчислення залишку*).

Таке хешування доволі швидке. Але необхідно уникати деяких значень m (не повинно бути степенем 2, так як для $m = 2^p$ значення h(k) буде просто p молодших бітів числа k, або 2^{p-1}). Часто m вибирають простим, досить далеким від числа, що є ступенем 2.

2. *Побудова хеш-функції методом множення*: $h(k)=\lfloor m(kA \bmod 1)\rfloor$, $0 < A<1$. Побудова хеш-функції методом множення виконується в два етапи. Спочатку ми множимо ключ k на константу A й одержуємо дробову частину отриманого добутку. Потім ми множимо отримане значення на m і застосовуємо до нього функцію заокруглення вниз тобто,

$$h(k)=\lfloor m(kA \bmod 1)\rfloor$$

де вираз « kA mod 1» означає одержання дробової частини добутку k, тобто величину $kA - \lfloor kA \rfloor$. Перевага методу множення полягає в тому, що значення m може бути довільним, часто вибирають $m = 2^p$, де p – деяке натуральне, але деякі значення константи А дають кращі результати у порівнянні з іншими. Оптимальне значення константи А залежить від характеристик даних, що хешуються Наприклад. Дональд Кнут запропонував використовувати значення $A \approx (\sqrt{5}-1)/2 \approx 0.6180339887$, що дає непогані результати

Завдання щодо роботи з хеш-таблицями:

Маємо український алфавіт із номерами позицій кожної букви

№ п.п.	Буква	№ п.п.	Буква
1	А а	18	Н н
2	Б б	19	О о
3	В в	20	П п
4	Г г	21	Р р
5	Ґ ґ	22	С с
6	Д д	23	Т т
7	Е е	24	У у
8	Є є	25	Ф ф
9	Ж ж	26	Х х
10	З з	27	Ц ц
11	И и	28	Ч ч
12	І і	29	Ш ш
13	Ї ї	30	Щ щ
14	Й й	31	Ь ь
15	К к		
16	Л л	32	Ю ю
17	М м	33	Я я

Рисунок 4.3 Номера позицій українських букв

1. Спроектувати, реалізувати, протестувати та проаналізувати алгоритм створення відкритою хеш-таблиці при хешування з ланцюжками за варіантом завдання із Таблиці «Варіанти завдань», використовуючи хеш-функції

побудовані за методом ділення h (K) =K mod 13 та множення h(k)=⌊16 (kA mod 1)⌋

2. Виконати візуалізацію створення відкритої хеш-таблиці надати пояснення

3. Спроектувати, реалізувати, протестувати та проаналізувати алгоритм створення закритої хеш-таблиці при хешування з відкритою адресацією за варіантом завдання із Таблиці «Варіанти завдань», використовуючи хеш-функції побудовані за методом ділення h (K) =K mod 13 та множення h(k)=⌊16 (kA mod 1)⌋

4. Виконати візуалізацію створення закритої хеш-таблиці надати пояснення

3. Показати на прикладі, що алгоритми хешування з ланцюжками та з відкритою адресацією мають середню ефективність сортування купою є нестійким та порівняти обчислювальну складність алгоритмів сортування купою та швидкого сортування за базовими операціями.

- знайдіть кількість порівнянь при пошуку (видаленні) будь-якого елемента із побудованих хеш-таблиць та вкажіть елемент з максимальною кількістю порівнянь у кожній хеш-таблиці
- підрахуйте середню кількість порівнянь для пошуку (видаленню) елемента в кожній хеш-таблиці

Таблиця Варіанти завдань:

№	Послідовність
1	Щоб рибу їсти, треба в воду лізти
2	Не той урожай, що в полі, а той, що в коморі.
3	Без труда нема плода, а без науки — мудрості.
4	Слово не горобець — вилетить, не спіймаєш
5	Не копай іншому яму, бо сам у неї впадеш
6	Скільки вовка не годуй, а він усе в ліс дивиться.
7	Людина без друзів — як дерево без коріння
8	Хто людям добра бажає, той і собі має
9	Чужу біду руками розведу, а до своєї розуму не дійду
10	Не хвали день до вечора, а хліб до вечері
11	Не той багатий, у кого багато грошей, а той, у кого душа багата
12	Хто дрібним не радіє, той великого не дочекається
13	Людина без рідного краю — як соловей без гнізда
14	Не клади всі яйця в один кошик, щоб не втратити все одразу
15	Хто грошима величається, той без душі зостачається.
16	Багатство не в грошах, а в доброму серці.
17	Більше діла, менше слів — отак добудеш хліб

18	Як поділиш радість — стане подвійною, а горе — вполовину меншим.
19	Шануй батька й матір, то буде тобі в житті добре.
20	Не той друг, хто медом маже, а той, хто правду каже
21	У чужих руках завжди хлібина більша, а праця легша.
22	Як дбаєш, так і маєш: хто землю робить, той хліб добрий робить
23	Як добре вчишся, то легко працюється, а як лінуєшся, то бідується.
24	У долі чужої руки довгі, а в своєї — короткі.
25	Не все те робота, що руки роблять — голова має знати, що робити
26	Як батька з матір'ю шануєш, так і свої діти тебе шануватимуть
27	Дітей виховуєш — про майбутнє дбаєш, самовпевненим станеш — усе втрати
28	Який батько, такий і син, а яка мати, така й дочка.
29	Не питай броду, коли йдеш у воду, і не питай правди в того, хто брехнею живе
30	Довіряй, але перевіряй — щоб слів на вітер не пускати.
31	Щоб знати, чи друг справжній, треба поділити з ним і горе, і радість
32	Не кожному багатство на радість — декого воно до біди доводить
33	Хто дрібницею нехтує, той великого не доб'ється.
34	Не кожна копійка дорога, але кожна з них свідчить про працю
35	Хоч рідна кров, але життя у кожного своє
36	Щоб коня запрягти, треба віз підготувати

5.Вимоги до представлення звіту

Звіт називається та оформлюється за шаблоном, який представлено на сайті Moodle в розділі вимоги до виконання лабораторних робіт та повинен містити:

1. Титульний лист, мету, завдання, результати виконання завдання, висновки (див. шаблон)

2. Результати виконання завдання за ЛР № 3 складаються із:

Сформованих власноруч за результатами сортування купою відповідно до варіанту завдання Лістингів 3.1 якій відповідає псевдокоду із

Таблиці 3.2, а також результатів Таблиць 3.3 - 3.4 і рисунків 3.9-3.10 з поясненнями.

За результатами порівняння обчислювальної складності алгоритмів самостійно сформуйте таблицю порівнянь.