Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Навчально-науковий інститут комп’ютерних систем
Кафедра інформаційних систем

Лабораторна робота № 4

з дисципліни: «Теорія алгоритмів»

Тема: «Робота з хеш-таблицями»

Варіант № 6

Виконав:
Студент групи АІ-243
Гаврилов О.В.
Перевірили:
Смик С. Ю.
Арсірій О.О.

Одеса 2025

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

Завдання:

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
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
5. 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

Номер варіанту: 6;
Вхідні дані (послідовність):
«Скільки вовка не годуй, а він усе в ліс дивиться.».

Результати виконання завдання:
Розглянемо наступний список із 10 слів

key	СКІЛЬКИ	ВОВКА	НЕ	ГОДУЙ	А	ВІН	УСЕ	В	ЛІС	ДИВИТЬСЯ
№ слова	1	2	3	4	5	6	7	8	9	10

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

№ 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 – 4.2 Номера позиції української букви

Тоді отримаємо

h(А) = (1) mod 13 = 1

h(ВОВКА) = (3+19+3+15+1) mod 13 = 2

h(ГОДУЙ) = (4+19+6+24+14) mod 13 = 2

h(ВІН) = (3+12+18) mod 13 = 7

h(УСЕ) = (24+22+7) mod 13 = 1

h(В) = (3) mod 13 = 3

h(ЛІС) = (16+12+22) mod 13 = 11

h(СКІЛЬКИ) = (22+15+12+16+15+12+11) mod 13 = 12

h(НЕ) = (18+7) mod 13 = 12

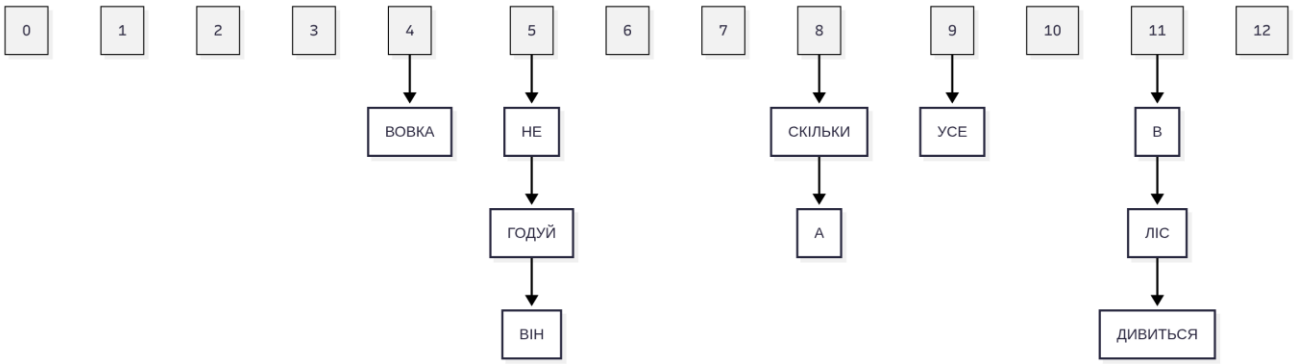
h(ДИВИТЬСЯ) = (6+12+3+12+23+31+22+33) mod 13 = 12

Таблиця 4.1 Значення хеш-функції

key	СКІЛЬКИ	ВОВКА	НЕ	ГОДУЙ	А	ВІН	УСЕ	В	ЛІС	ДИВИТЬСЯ
№ слова	1	2	3	4	5	6	7	8	9	10
h(key)	12	2	12	2	1	7	1	3	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



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

```
# --- ЗАВДАННЯ: Хешування прислів'я українською мовою ---

# Константа розміру таблиці
M = 13

# Список вхідних слів (транслітерація прислів'я для коректності):
# Використовуємо слова українською
WORDS_UA = ["СКІЛЬКИ", "ВОВКА", "НЕ", "ГОДУЙ", "А", "ВІН", "УСЕ", "В",
"ЛІС", "ДИВИТЬСЯ"]

# Словник позицій українських букв (згідно Рисунку 4.3)
LETTER_POSITIONS = {
    'А': 1, 'Б': 2, 'В': 3, 'Г': 4, 'Ґ': 5, 'Д': 6, 'Е': 7, 'Є': 8, 'Ж':
9, 'З': 10, 'И': 11, 'І': 12, 'Ї': 13, 'Й': 14,
    'К': 15, 'Л': 16, 'М': 17, 'Н': 18, 'О': 19, 'П': 20, 'Р': 21, 'С':
22, 'Т': 23, 'У': 24, 'Ф': 25, 'Х': 26, 'Ц': 27,
    'Ч': 28, 'Ш': 29, 'Щ': 30, 'Ь': 31, 'Ю': 32, 'Я': 33
}

def simple_hash_from_map(key: str) -> int:
    """
    Хеш-функція: h(k) = (сума позицій букв українського алфавіту) mod 13.
    """
    sum_of_positions = 0

    # Обчислення суми позицій
    for char in key.upper():
        # 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) за методом ділення.
    """
    # 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: ---
# 1. Побудова таблиці
hash_table = build_open_hash_table(WORDS_UA, M)

# 2. Виведення результату
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: ['А', 'УСЕ']
Індекс 02: ['ВОВКА', 'ГОДУЙ']
Індекс 03: ['В']
Індекс 04: []
Індекс 05: []
Індекс 06: []
Індекс 07: ['ВІН']
Індекс 08: []
Індекс 09: []
Індекс 10: []
Індекс 11: ['ЛІС']
Індекс 12: ['СКІЛЬКИ', 'НЕ', 'ДИВИТЬСЯ']
```

Таблиця 4.4 Візуалізацію створення відкритої хеш-таблиці

	хеш-таблиця з відкритою адресацією												
Insert key	0	1	2	3	4	5	6	7	8	9	10	11	12
1													СКІЛЬКИ
2			ВОВКА										СКІЛЬКИ
3	НЕ		ВОВКА										СКІЛЬКИ
4	НЕ		ВОВКА	ГОДУЙ									СКІЛЬКИ

5	НЕ	А	ВОВКА	ГОДУЙ									СКІЛЬКИ
6	НЕ	А	ВОВКА	ГОДУЙ				ВІН					СКІЛЬКИ
7	НЕ	А	ВОВКА	ГОДУЙ	УСЕ			ВІН					СКІЛЬКИ
8	НЕ	А	ВОВКА	ГОДУЙ	УСЕ	В		ВІН					СКІЛЬКИ
9	НЕ	А	ВОВКА	ГОДУЙ	УСЕ	В		ВІН					СКІЛЬКИ

Лістинг 4.5 – Python-код реалізації алгоритмів хешуванням з роздільними ланцюжками та візуалізації за закритої хеш-таблиці

```
# --- КОНСТАНТИ ---
M = 13
A = (5**0.5 - 1) / 2 # ~0.6180339887 для методу множення

WORDS_UA = ["СКІЛЬКИ", "ВОВКА", "НЕ", "ГОДУЙ", "А", "ВІН", "УСЕ", "В", "ЛІС",
"ДИВИТЬСЯ"]

LETTER_POSITIONS_UA = {
    'А': 1, 'Б': 2, 'В': 3, 'Г': 4, 'Ґ': 5, 'Д': 6, 'Е': 7, 'Є': 8, 'Ж': 9, 'З':
10, 'И': 11, 'І': 12, 'Ї': 13, 'Й': 14,
    'К': 15, 'Л': 16, 'М': 17, 'Н': 18, 'О': 19, 'П': 20, 'Р': 21, 'С': 22, 'Т':
23, 'У': 24, 'Ф': 25, 'Х': 26, 'Ц': 27,
    'Ч': 28, 'Ш': 29, 'Щ': 30, 'Ь': 31, 'Ю': 32, 'Я': 33
}

# --- ДОПОМІЖНА ФУНКЦІЯ: Обчислення K (суми позицій) ---
def calculate_k(key: str) -> int:
    """Обчислює числовий ключ K як суму позицій букв."""
    k = 0
    for char in key.upper():
        k += LETTER_POSITIONS_UA.get(char, 0)
    return k

# --- ХЕШ-ФУНКЦІЇ З ОСНОВНИМ ЗОНДУВАННЯМ ---

def hash_division_probe(k: int, i: int) -> int:
    """h(K) = (K mod M + i) mod M"""
    return (k % M + i) % M

def hash_multiplication_probe(k: int, i: int) -> int:
    """h(k) = (floor(M * (k*A mod 1)) + i) mod M"""
    h0 = int(M * ((k * A) % 1))
    return (h0 + i) % M

# --- АЛГОРИТМ ПОБУДОВИ ТАБЛИЦІ ---

def build_closed_hash_table(words: list, m: int, probe_func) -> (list, list):
    """
    Будує закриту хеш-таблицю (відкрита адресація) з лінійним зондуванням.
    Повертає таблицю та список довжин зондування.
    """
    hash_table = [None] * m
    probe_lengths = []

    for word in words:
        k = calculate_k(word)
        i = 0

        # Лінійне зондування: шукаємо наступну вільну комірку
        while i < m:
```

```
            address = probe_func(k, i)

            if hash_table[address] is None:
                hash_table[address] = word
                probe_lengths.append((word, i + 1)) # i+1 - довжина зондування
                break

            i += 1
        else:
            print(f"Помилка: Таблиця переповнена, не вдалося вставити {word}")

    return hash_table, probe_lengths

def display_closed_hash_table(table: list, title: str):
    """Виводить хеш-таблицю у зручному форматі."""
    print(f"\n--- {title} (Таблиця M={M}) ---")

    # Заголовки індексів
    print("Індекс:", end=" ")
    for i in range(len(table)):
        print(f"{i:02d}", end=" ")
    print()

    # Вміст комірок
    print("Вміст:", end="  ")
    for item in table:
        print(f"{item[:2] if item else '--'}", end=" ") # Відображаємо лише перші
2 символи
    print("\n")


# --- ВИКОНАННЯ ---

# 1. Метод Ділення
hash_table_div,     lengths_div     =     build_closed_hash_table(WORDS_UA,     M,
hash_division_probe)
display_closed_hash_table(hash_table_div,    "Результат    Закритої    Хеш-таблиці
(Ділення)")

# 2. Метод Множення
hash_table_mult,    lengths_mult    =    build_closed_hash_table(WORDS_UA,    M,
hash_multiplication_probe)
display_closed_hash_table(hash_table_mult,   "Результат   Закритої   Хеш-таблиці
(Множення)")

# Аналіз довжин зондування
print("\n--- Аналіз Довжин Зондування (Кількість кроків для вставки) ---")
print("Метод Ділення:")
for word, length in lengths_div:
    print(f"  {word}: {length} кроків")

print("\nМетод Множення:")
for word, length in lengths_mult:
    print(f"  {word}: {length} кроків")
```

Результати виконання Python-коду, який наведено в Лістінгу 4.5 показано в таблиці 4.6

Таблиця 4.6 Результати побудови закритої хеш-таблиці

```
--- Результат Закритої Хеш-таблиці (Ділення) (Таблиця M=13) ---
Індекс: 00 01 02 03 04 05 06 07 08 09 10 11 12
Вміст:  -- А ВО ГО УС СК В ВІ -- -- ДИ ЛІ НЕ


--- Результат Закритої Хеш-таблиці (Множення) (Таблиця M=13) ---
Індекс: 00 01 02 03 04 05 06 07 08 09 10 11 12
Вміст:  ДИ -- -- -- ВО СК НЕ ГО А ВІ УС В ЛІ


--- Аналіз Довжин Зондування (Кількість кроків для вставки) ---
Метод Ділення:
  СКІЛЬКИ: 1 кроків
  ВОВКА: 1 кроків
  НЕ: 1 кроків
  ГОДУЙ: 2 кроків
  А: 1 кроків
  ВІН: 1 кроків
  УСЕ: 4 кроків
  В: 4 кроків
  ЛІС: 1 кроків
  ДИВИТЬСЯ: 1 кроків

Метод Множення:
  СКІЛЬКИ: 1 кроків
  ВОВКА: 1 кроків
  НЕ: 2 кроків
  ГОДУЙ: 3 кроків
  А: 1 кроків
  ВІН: 5 кроків
  УСЕ: 2 кроків
  В: 1 кроків
  ЛІС: 2 кроків
  ДИВИТЬСЯ: 8 кроків
```

Таблиця 4.7 Візуалізацію створення закритої хеш-таблиці

	хеш-таблиця з відкритою адресацією												
Insert key	0	1	2	3	4	5	6	7	8	9	10	11	12
1													СКІЛЬКИ
2			ВОВКА										СКІЛЬКИ
3	НЕ		ВОВКА										СКІЛЬКИ
4	НЕ		ВОВКА	ГОДУЙ									СКІЛЬКИ
5	НЕ	А	ВОВКА	ГОДУЙ									СКІЛЬКИ
6	НЕ	А	ВОВКА	ГОДУЙ				ВІН					СКІЛЬКИ
7	НЕ	А	ВОВКА	ГОДУЙ	УСЕ			ВІН					СКІЛЬКИ
8	НЕ	А	ВОВКА	ГОДУЙ	УСЕ	В		ВІН					СКІЛЬКИ

9	НЕ	А	ВОВКА	ГОДУЙ	УСЕ	В		ВІН				ЛІС	СКІЛЬКИ

Таблиця 4.8 Зведене Порівняння Ефективності Алгоритмів (Сортування та Хешування)

Назва алгоритму	Реалізація	Складність (Теорія)	Порівняння	Присвоювання / Обміни	Додаткові лічильники
Selection Sort (ЛР1	Ітеративна	$O(N^2)$	44	40	-
Insertion Sort (ЛР1)	Ітеративна	$O(N^2)$	24	58	-
Heap Sort	Ітеративна (Sink)	*O(N log N)*	45	18	-
QuickSort (ЛР2)	Рекурсивна	$O(N \log N)_{(Avg)}$	65	103	Рекурсивних викликів: 8
Merge Sort (ЛР2)	Рекурсивна	*O(N log N)*	110	119	Рекурсивних викликів: 17
Merge Sort (ЛР2)	Ітеративна	*O(N log N)*	99	206	-
Хешування з Ланцюжками (Пошук)	Ітеративна	*O(1)*	3 (макс. L_{max} у вашій табл.)	1.5 (середнє порівняння)	Потрібна пам'ять на ланцюжки
Закрита (Ділення)	Ітеративна	*O(1)*	4 (макс. зондування у вашій табл.)	1.7 (середнє порівняння)	Макс. кластеризація: 4 кроки
Закрита (Множення)	Ітеративна	*O(1)*	8 (макс. зондування у вашій табл.)	2.6 (середнє порівняння)	Макс. кластеризація: 8 кроків