Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

**Лабораторна робота №5**

З дисципліни «Алгоритми та структури даних-2»

Тема:  «Дослідження структури даних: хеш-таблиця (hash table)»

Виконали: Перевірив:

студенти групи ІТ-03 Смолій В. В.

Митєв А. Ю.

Яремчук Д. В.

Шевчук Д. Д.

Дата здачі: 25.04.21

Захищено з балом \_\_\_\_\_\_\_\_

Київ 2021

**Тема**: «Дослідження структури даних: хеш-таблиця (hash table)»

**Мета**: Закріпити знання з механізмів функціонування хеш-таблиць, реалізувати на практиці відповідні користувацькі структури даних та дослідити ефективність їх практичної реалізації.

**Завдання:**

Визначити склад, методи та інструменти для реалізації інтерфейсу та відповідного класу об'єктів, визначити алгоритми функціонування складових компонентів, реалізувати відповідне програмне забезпечення.

Окреме питання приділити методам реалізації хеш-функцій та розв'язання колізій.

Визначити які хеш-функції і чому можна вважати «*гарними*» для реалізації у поточній задачі, обґрунтувати використання обраного варіанту.

1. Реалізувати наступні методи для операцій:

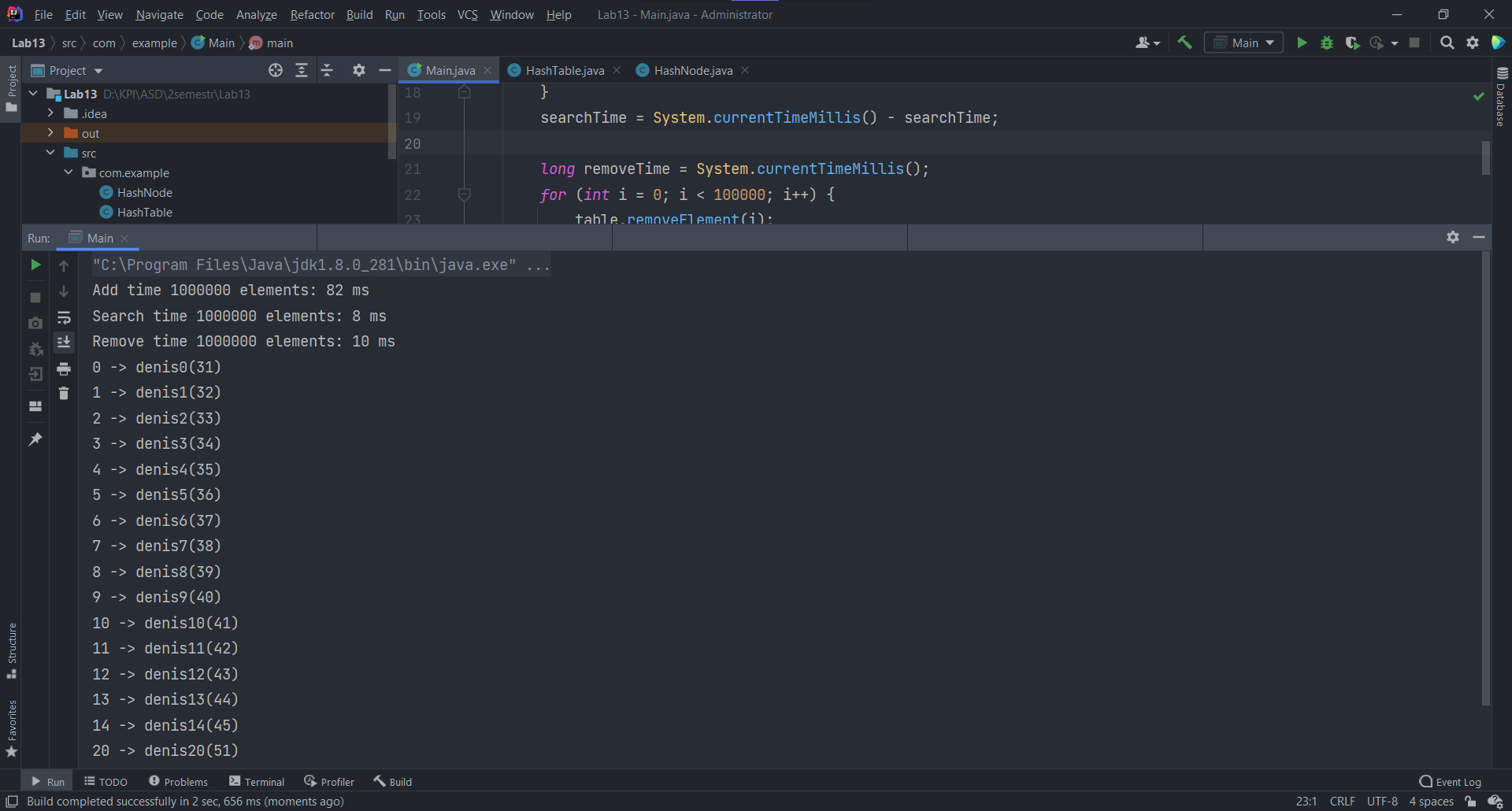
* вставки;
* видалення елементу;
* пошуку елементу за значенням (існуючого та не існуючого у структурі);
* отримання впорядкованої послідовності елементів.

2. Для кожної операції визначити найкращий та найгірший (якщо є такі показники) час виконання (на великому наборі числових значень, не менш ніж 10 000) та пояснити отримані результати.

3. Визначити кількість пам’яті, яку займає структура даних.

4. Теоретично пояснити способи розв’язання колізій

**Хід роботи**

1. Я вибрав ланцюговий метод розв’язання колізій. Він полягає в тому, що кожен елемент списку це, по суті, однозв’язний список і, якщо відбувається колізія (якщо хешкоди 2 об’єктів збігаються), то елемент добавляється в кінець цього однозв’язного списку.
2. Хешкод ми розраховуємо по ключу за допомогою вбудованого метода Objects.hash(). Індекс в списку визначається остачею від ділення хешкоду на кількість, так званих, бакетів (buckets). В кінці кожної операції додавання елемента в таблицю ми перевіряємо чи не дуже заповнений список, і якщо він заповнений більш ніж треба, то ми створюємо нову хештаблицю (з вдвічі більшою кількістю бакетів) і додаємо туди всі елементи з оригінальної таблиці.
3. Час додавання 100000 елементів + сортування:

Оскільки тип даних можу бути будь-який, то ми сортуємо за хешкодом (на картинці він відображений в дужках). Сортування відбувається за

O(n\*log(n)), оскільки ми сортуємо за допомогою Arrays.sort().

Найгірший час всіх операцій – O(n), найкращий – O(1). Оскільки loadFactor – 0.75, то час виконання ближчий до O(1). Тобто додавання, видалення та пошук n елементів буде в середньому та найкращому випадку відбуватися за O(n), в найгіршому – O(n^2). В нашому випадку, швидше за все, всі операції відбуватимуться по середньому або найкращому випадках, тобто за O(n). Додавання буде відбуватися трішки довше, адже ми в певні моменти будемо змінювати розмір, тому час буде O(n\*lg(n)).

1. K – ключ (тут це ціле число), V – значення (рядок)

Розмір об’єкта HashNode<K,V> - K + V + 8 (посилання на наступний елемент) = 4 (ціле число) + 8 (рядок) + 8 = 20 байтів

Розмір об’єкта HashTable<K, V> - 4 (змінна кількості бакетів) + 4 (змінна кількості елементів) + 4 (змінна заповненості списка) + 20\*n (всі елементи) = 20\*n + 12 байтів

1. Посилання на весь код - <https://github.com/Den-droid/ASD_Lab4>

***Висновки***

Ми ознайомилися з такою структурою даних як хештаблиця, дізналися її основні операції та час їхнього виконання.