# Хеш-таблиці

Сидоренко М.О.

- **Хеш-таблиця** - це структура даних, яка впроваджує інтерфейс асоціативного масиву, а саме, вона дозволяє зберігати пари (ключ, значення) і виконувати три операції: операцію додавання нової пари, операцію пошуку і операцію видалення пари по ключу.
- Існують два основні варіанти хеш-таблиць: з ланцюжками і відкритою адресацією. Хеш-таблиця містить деякий масив , елементи якого є пари (хеш-таблиця з відкритою адресацією) або списки пар (хеш-таблиця з ланцюжками).

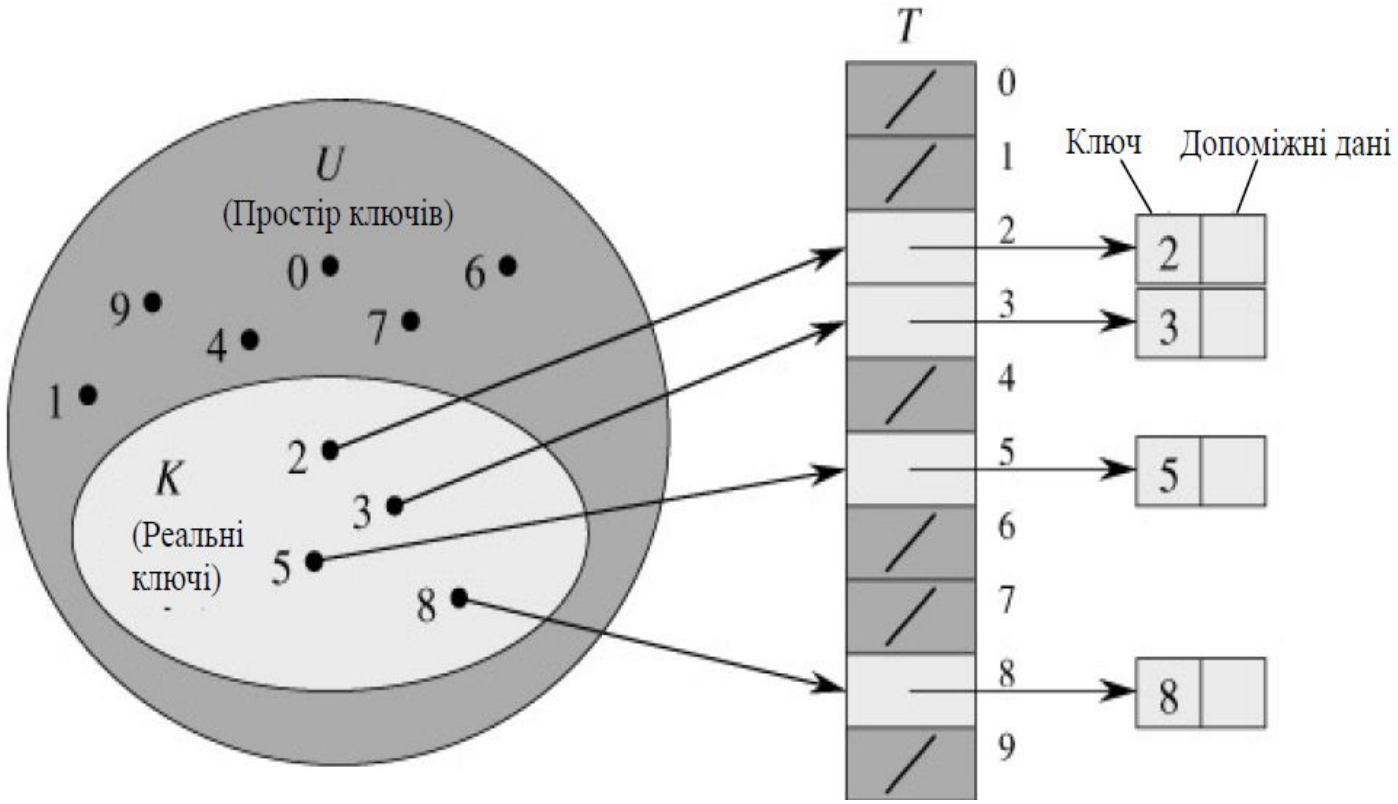
# Таблиці з прямою адресацією

- Припустимо, що застосуванню потрібна динамічна множина, кожний елемент якої має ключ з множини $U = \{0, 1, \ldots, m - 1\}$, де $m$ не дуже велике. Крім того, припускається, що жодні два елементи не мають однакових ключів.
- Для представлення динамічної множини використовується масив, або таблиця з прямою адресацією, який позначається як $T[0\ldots m - 1]$, кожна позиція, або комірка, якого відповідає ключу з простору ключів $U$.

# Таблиці з прямою адресацією

- Комірка k вказує на елемент множини з ключем k. Якщо множина не містить елементу з таким ключем, то T[k] = NIL. На рисунку кожний ключ простору U = {0, 1, …, 9} відповідає індексу таблиці. Множина реальних ключів K = {2, 3, 5, 8} визначає комірки таблиці, які містять покажчики на елементи. Решта комірок містять значення NIL.

# Динамічна множина із використанням таблиці з прямою адресацією



# Процедури, які реалізують операції роботи з масивами.

- DirectAddressSearch(T, k)

  return T[k]

- DirectAddressInsert(T, x)

  T[key[x]] ← x

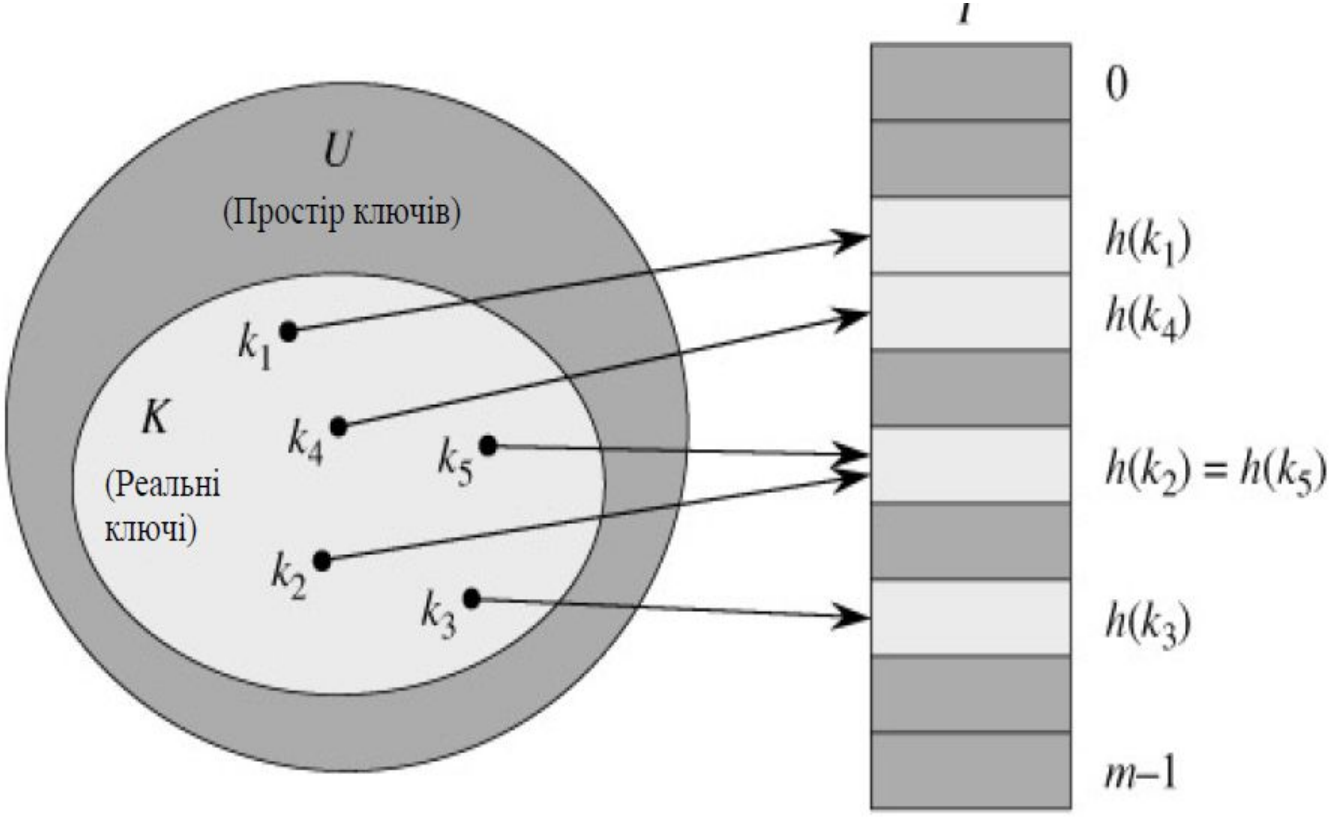
- DirectAddressDelete(T, x)

  T[key[x]] ← NIL

# Хеш-таблиці

- У випадку прямої адресації елемент з ключем k зберігається у комірці k. При хешуванні цей елемент зберігається в комірці h(k), тобто тут використовується хеш-функція h для обчислення комірки для даного ключа k. Функція h відображає простір ключів U на комірки хеш-таблиці T[0…m – 1]: h: U → {0, 1, …, m – 1}.
- Ми говоримо, що елементи з ключем k хешується в комірку h(k); величина h(k) називається хеш-значенням ключа k.
- Мета хеш-функції полягає в тому, щоб зменшити робочий діапазон індексів масиву.

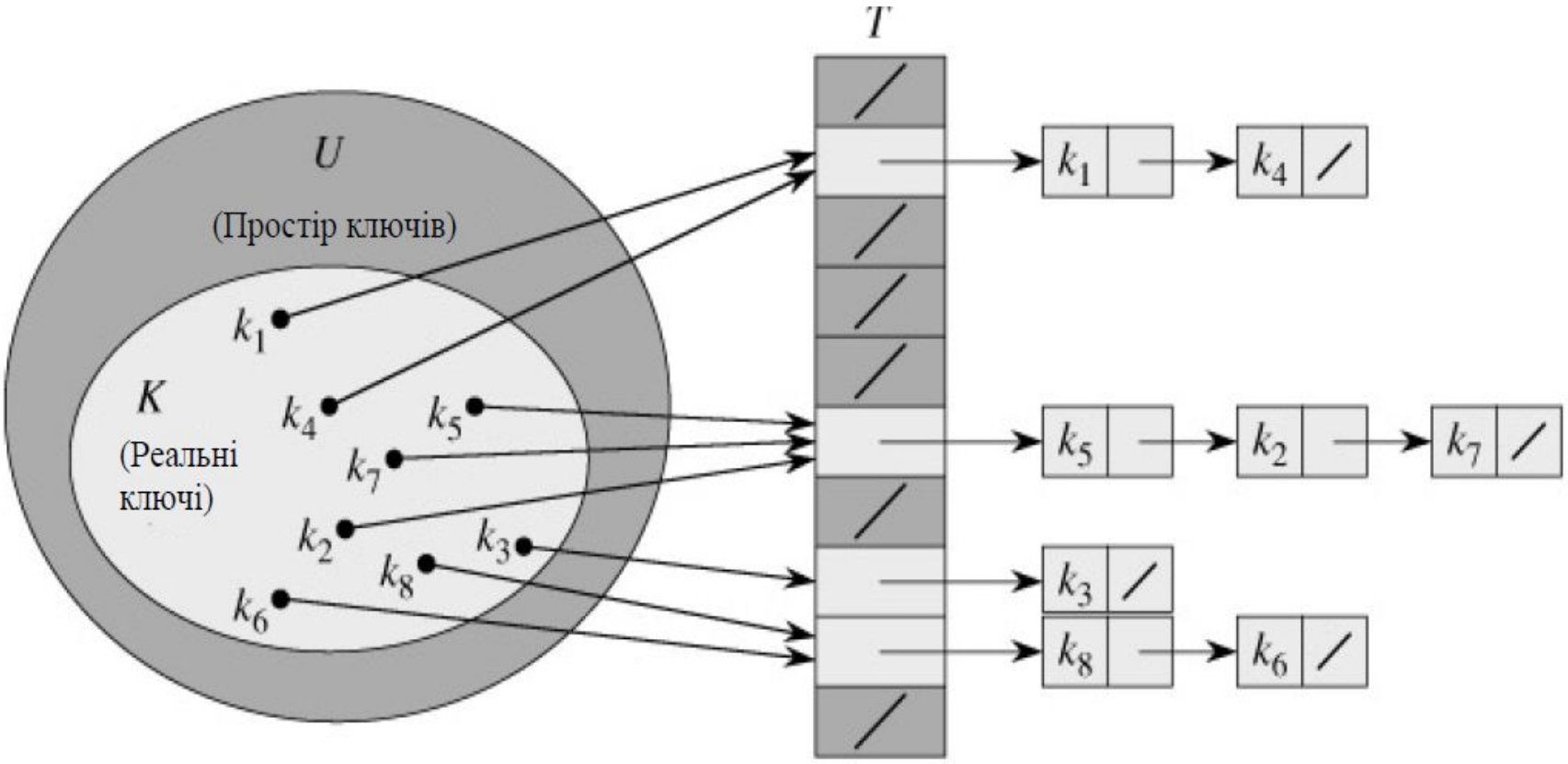
# Використання хеш-функції для відображення ключів у комірки хеш-таблиці



# Розв’язання колізій за допомогою ланцюгів

- Два ключа можуть мати одне й те саме хеш-значення.
- Так ситуація називається колізією.
- За допомогою методу ланцюгів всі елементи, які хешуються в одну й ту саму комірку об’єднуються у зв’язаний список.
- Комірка j містить покажчик на заголовок списку всіх елементів, хеш-значення ключа яких дорівнює j; якщо таких елементів немає, комірка містить значення NIL.

# Розв’язання колізій за допомогою ланцюгів



# Словарні операції в хеш-таблиці із використанням ланцюгів

- ChainedHashInsert(T, x)

  Вставити x в заголовок списку T[h(key[x])]

- ChainedHashSearch(T, k)

  Пошук елементу з ключем k в списку T[h(k)]

- ChainedHashDelete(T, x)

  Видалення x зі списку T[h(key[x])]

- T- хеш-таблиця з m комірками, в яких зберігаються n елементів.
- Коефіцієнт заповнення таблиці T як $\alpha = n/m$, тобто як середню кількість елементів, які зберігаються в одному ланцюгу.
- Припустимо, що всі елементи хешуються по комірках рівномірно та незалежно, і назвемо це припущення «простим рівномірним хешуванням».

- Позначимо довжини списків T[j] для j = 0, 1, …, m – 1 як $n_j$, так що $n = n_0 + n_1 + n_{m-1}$,а середнє значення $n_j$ дорівнює E[ $n_j$] = α = n/m.

# Хеш-функції

- Якісна хеш-функція задовольняє (наближено) припущення простого рівномірного хешування: для кожного ключа рівно ймовірне розміщення в будь-яку з m комірок
- Наприклад, якщо відомо, що ключі представляють собою випадкові дійсні числа, які рівномірно розподілені в діапазоні 0 ≤ k < 1, то хеш-функція $h(k) = \lfloor km \rfloor$ задовольняє умові простого рівномірного хешування.
- Гарним підходом є підбір функції таким чином, щоб вона ніяк не корелювала із закономірностями, яким можуть підпорядковуватись існуючі дані

# Хеш-функції

- Розглянемо наступні два методи побудови хеш-функцій: метод ділення та метод множення.
- Побудова хеш-функції методом ділення полягає у відображені ключа k в одну з комірок шляхом отримання остачі від ділення k на m, тобто хеш-функція має вигляді: $h(k) = k \bmod m$.
- При використанні даного методу зазвичай намагаються уникнути деяких значень m.
- Наприклад, m не повинно бути степенем 2.
- Часто добрі результати можна отримати, якщо обирати в якості значення m просте число, достатньо далеке від степеня двійки.

# Хеш-функції

- Побудова хеш-функції методом множення виконується в два етапи. Спочатку ключ k помножується на сталу 0 < A < 1 і береться дробова частина отриманого добутку.
- Потім це значення помножується на m і результат замінюється на найближче менше ціле число, тобто: $h(k) = \lfloor m(kA mod 1) \rfloor$, де вираз “kA mod 1” означає отримання дробової частини добутку kA.
- Перевага методу множення полягає в тому, що значення m перестає бути критичним. Зазвичай величина m із міркувань зручності реалізації функції обирається рівною степеню 2.

# Відкрита адресація

- При використанні методу відкритої адресації всі елементи зберігаються безпосередньо в хеш-таблиці, тобто кожний запис таблиці містить або елемент динамічної множини, або значення NIL.
- Для виконання вставки при відкритій адресації ми послідовно перевіряємо комірки хеш-таблиці до тих пір, доки не знайдемо порожню комірку, в яку розміщується новий ключ. Замість фіксованого порядку дослідження комірок 0, 1, …, m – 1 (для чого потрібний час $\Theta(n)$), послідовність досліджуваних комірок залежить від ключа, який вставляється у таблицю.

# Приклад

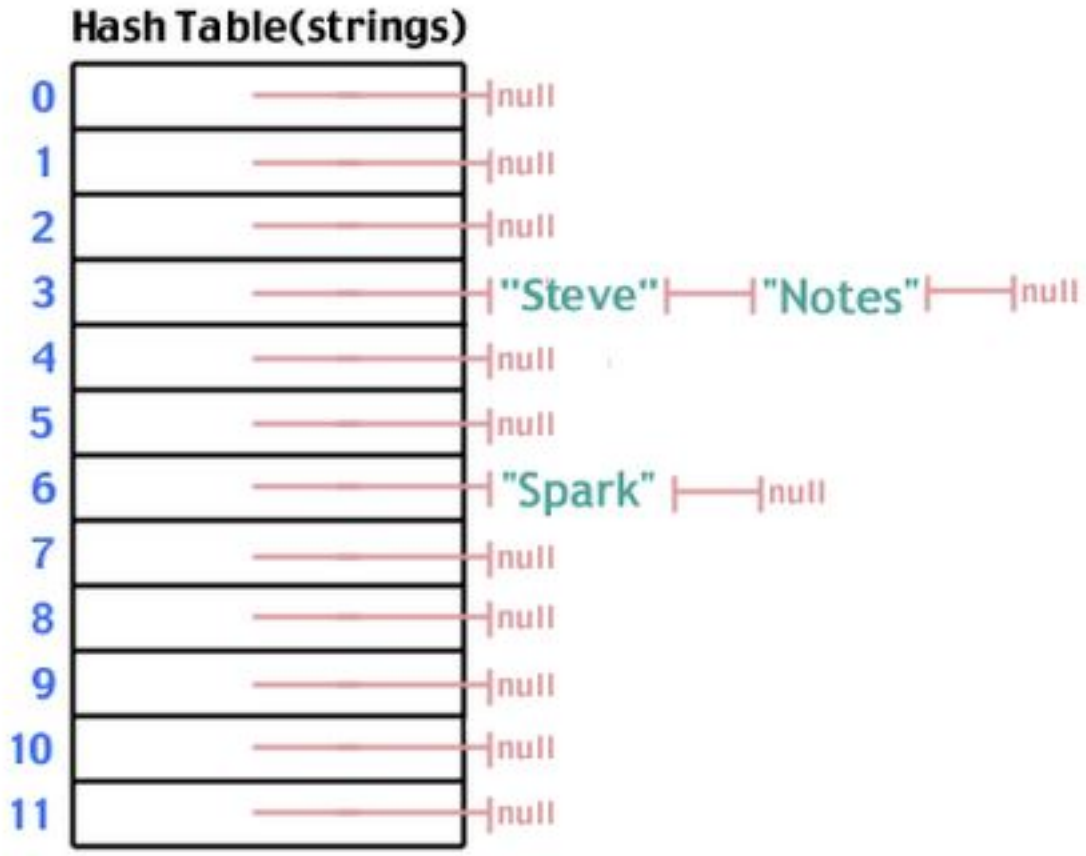

Hash Table(strings)
0 (null)
1 (null)
2 (null)
3 (null)
4 (null)
5 (null)
6 (null)
7 (null)
8 (null)
9 (null)
10 (null)
11 (null)

# Приклад колізії

```
int hash(char *str, int table_size)
{
      int sum;

      /* Make sure a valid string passed in */
      if (str==NULL) return -1;

      /* Sum up all the characters in the string */
      for( ; *str; str++) sum += *str;

      /* Return the sum mod the table size */
      return sum % table_size;
}
```

# Приклад колізії

# вирішення колізії за допомогою ланцюгів



# Представлення структури hash-table на мові C

```
typedef struct _list_t_ {
    char *str;
    struct _list_t_ *next;
} list_t;

typedef struct _hash_table_t_ {
    int size;       /* the size of the table */
    list_t **table; /* the table elements */
} hash_table_t;
```

# ініціалізація Хеш-таблиці

```
hash_table_t *create_hash_table(int size)
{
    hash_table_t *new_table;

    if (size<1) return NULL; /* invalid size for table */

    /* Attempt to allocate memory for the table structure */
    if ((new_table = malloc(sizeof(hash_table_t))) == NULL) {
        return NULL;
    }

    /* Attempt to allocate memory for the table itself */
    if ((new_table->table = malloc(sizeof(list_t *) * size)) == NULL) {
        return NULL;
    }

    /* Initialize the elements of the table */
    for(int i=0; i<size; i++) new_table->table[i] = NULL;

    /* Set the table's size */
    new_table->size = size;

    return new_table;
}
```

# Відкрита адресація

- В результаті хеш-функція стає наступною: h :U × {0,1,.. m −1 }-> {0,1,0..,m −1 } ,
- У методі відкритої адресації потрібно, що для кожного ключа k послідовність досліджень
- H( k ,0), h (k ,1),( h (k, m − 1)) представляла собою перестановку множини {0, 1, …, m – 1}, щоб в кінцевому випадку можна було переглянути всі комірки хеш-таблиці.
- У наведеному нижче псевдокоді припускається, що елементи в таблиці T представляють собою ключі без додаткової інформації; ключ k тотожній елементу, який містить ключ k. Кожна комірка містить або ключ, або значення NIL

# Процедура додавання елементу до відкритої адресації

```
HashInsert(T, k)
1    i ← 0
2   repeat j ← h(k, i)
3        if T[j] = NIL
4             then T[j] ← k
5                  return j
6             else i ← i + 1
7        until i = m
8    error “Хеш-таблиця переповнена”
```

# Процедура пошуку ключа у відкритій адресації

```
HashSearch(T, k)
1   i ← 0
2   repeat j ← h(k, i)
3         if T[j] = k
4               then return j
5         i ← i + 1
6   until T[j]
```

# Рівномірне хешування

- Ми будемо виходити із припущення рівномірного хешування, тобто ми припускаємо, що для кожного ключа в якості послідовності досліджень рівноймовірні всі m! перестановок множини {0, 1, …, m – 1}.
- Рівномірне хешування представляє собою узагальнення визначеного раніше простого рівномірного хешування, яка полягає в тому, що тепер хеш-функція дає не одне значення, а цілу послідовність досліджень

# Рівномірне хешування

- Для обчислення послідовності досліджень для відкритої адресації зазвичай використовуються три методи: лінійне дослідження, квадратичне дослідження та подвійне хешування.

# Лінійне хешування

- Нехай задана звичайна хеш-функція h′: U → {0, 1, …, m – 1}, яку будемо надалі йменувати допоміжною хеш-функцією. Метод лінійного дослідження для обчислення послідовності досліджень використовує хеш-функцію h(k,i)=(h’(k)+i)mod m, де і приймає значення від 0 до m – 1. Для заданого ключа k першою досліджуваною коміркою є T[h′(k)], тобто комірка, яку дає допоміжна хеш-функція. Далі досліджуються комірки T[h′(k) + 1], T[h′(k) + 2], …, T[m – 1], а потім переходять до T[0], T[1], і далі до T[h′(k) – 1]. Оскільки початкова досліджувана комірка однозначно визначає всю послідовність досліджень в цілому, всього є m різних послідовностей.

# Квадратичне дослідження

- Квадратичне дослідження використовує хеш-функцію вигляду:
  $h(k,i) = (h'(k) + c_1 i + c_2 i^2) mod\ m$ , де h′ – допоміжна хеш-функція, c1 та c2 ≠ 0 – допоміжні константи, а і приймає значення від 0 до m – 1. Початкова досліджувана комірка – T[h′(k)]; решта досліджуваних позицій зміщені відносно неї на величину, які описуються квадратичною залежністю від номеру дослідження і.

# Подвійне хешування

- Подвійне хешування представляє собою один з найкращих методів використання відкритої адресації, оскільки отримані при цьому перестановки мають багато характеристик випадкового згенерованих перестановок. Подвійне хешування використовує хеш-функцію наступного вигляду:
  $h(k,i) = (h_1(k) + ih_2(k)) mod\ m$, де h1 та h2 – допоміжні хеш-функції.
- Початкове дослідження виконується в позиції T[h1(k)], а зміщення кожної з наступних досліджуваних комірок відносно попередньої дорівнює h2(k) по модулю m.

- Для того щоб послідовність досліджень могла охопити всю таблицю, значення h2(k) повинно бути взаємно простим із розміром хеш-таблиці m
- Наприклад, можна обрати просте число в якості m, а хеш-функції такими:
- h1(k) = k mod m ,
- h2(k)=(1+k mod m′) ,
- де m′ повинно бути трохи менше m (наприклад, m – 1).

- Дякую за увагу.