# Хеш-таблиці

# Таблиця

*Таблиця* – це структура даних, яка дозволяє зберігати пари виду "ключ-значення (дані)".

## Основні операції:

- додавання нової пари ключ-значення;
- пошук значення по ключу;
- видалення пари ключ-значення по ключу.

| індекс | key | value 1 | … | value k |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| 2 | | | | |
| … | | | | |
| | | | | |

# Таблиця з прямою адресацією

index = key

| key | value 1 | … | value k |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

*DirectAddressSearch*(T, key)
    return T[key]

*DirectAddressInsert*(T, x)
    return T[x.key]=x

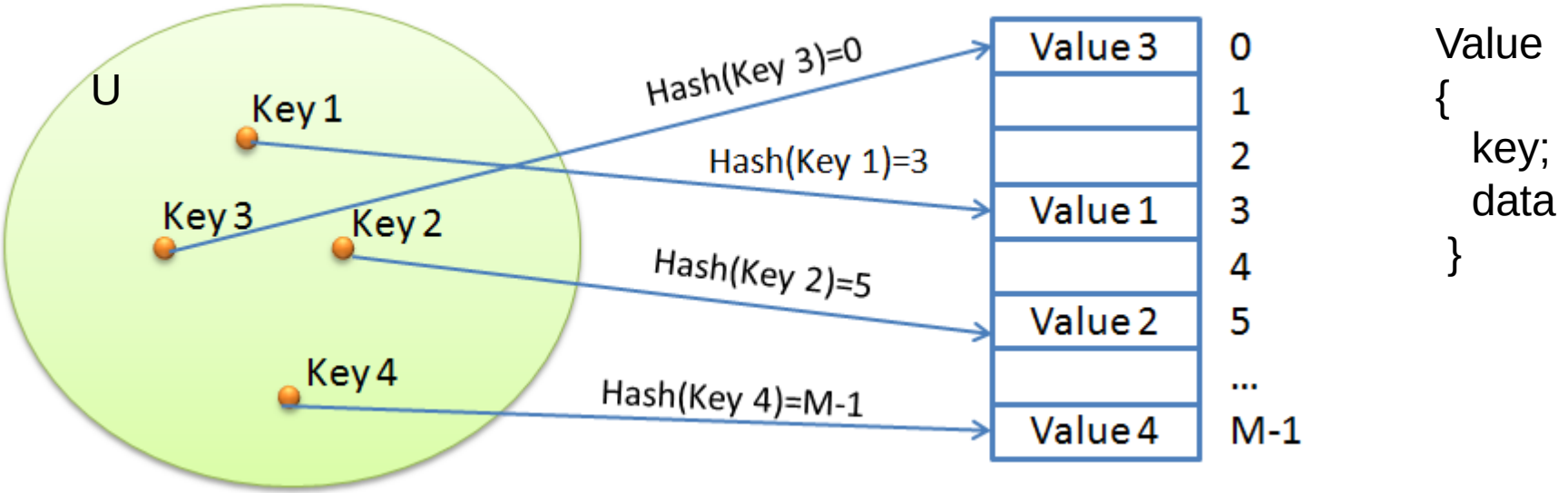
*DirectAddressDelete*(T, x)
    return T[x.key]=null

# Хеш-таблиця

index = Hash(key)

$$Hash : U \to \{0, 1, 2, ..., M-1\}$$

$$M < |U|$$



# Хеш-функція

*Хеш-функція (Hash function)* – це функція, що перетворює значення ключа (наприклад: строки, числа, файла) в ціле число. Значення, що повертає хеш-функція називається хеш-кодом (hash code), контрольною сумою (hash sum) або хешем (hash) і застосовується в якості індекса хеш-таблиці. Весь процес отримання індексів хеш-таблиці називається *хешуванням*.

# Основні вимоги до хеш-функцій:

*Швидке обчислення* хеш-коду по значенню ключа. Складність обчислення хеш-коду не повинна залежати від N – кількості елементів в хеш-таблиці.

*Детермінованість.* Для заданого значення ключа хеш-функція завжди повинна повертати одне й те саме значення.

*Рівномірність.* Хеш-функція повинна рівномірно заповнювати масив. Бажано, щоб усі хеш-коди формувалися з однаковою ймовірністю.

# Приклади хеш-функцій

Метод ділення: $h(k) = k \bmod M$

Важливо правильно обрати M. Зазвичай обирають просте число, далеке від степені двійки.

Метод множення: $h(k) = [M \cdot \{k \cdot A\}]$,

де { } – дробна частина,

[ ] – ціла частина,

A – дійсне число, $0 < A < 1$,

Кнут запропонував в якості A використовувати число, зворотнє до золотого перетину:

$$A = \phi^{-1} = (\frac{\sqrt{5}-1}{2}) = 0,6180339887...$$

# Приклади хеш-функцій (строки)

Нехай строка s містить символи $s_0$, $s_1$, …, $s_{n-1}$

Варіант 1: $h_1(s) = (s_0 + s_1 a + s_2 a^2 + \ldots + s_{n-1} a^{n-1}) \bmod M$

Варіант 2: $h_2(s) = (s_0 a^{n-1} + s_1 a^{n-2} + \ldots + s_{n-2} a + s_{n-1}) \bmod M$

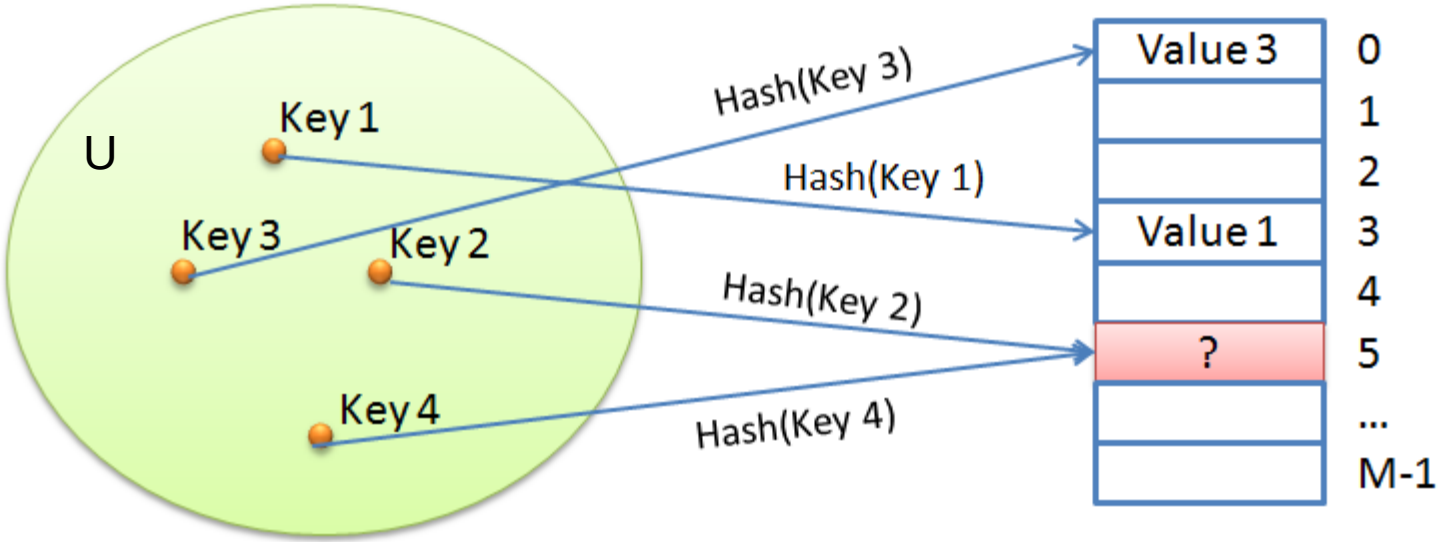
В якості a рекомендують обирати просте число, що приблизно дорівнює кількості символів у вхідному алфавіті

# Універсальне хешування

(застосування набору хеш-функцій)

# Колізія

*Колізія* – це збіг значень хеш-функції для двох різних ключів.




Hash(Key 2) = Hash(Key 4)

# Способи вирішення колізій:

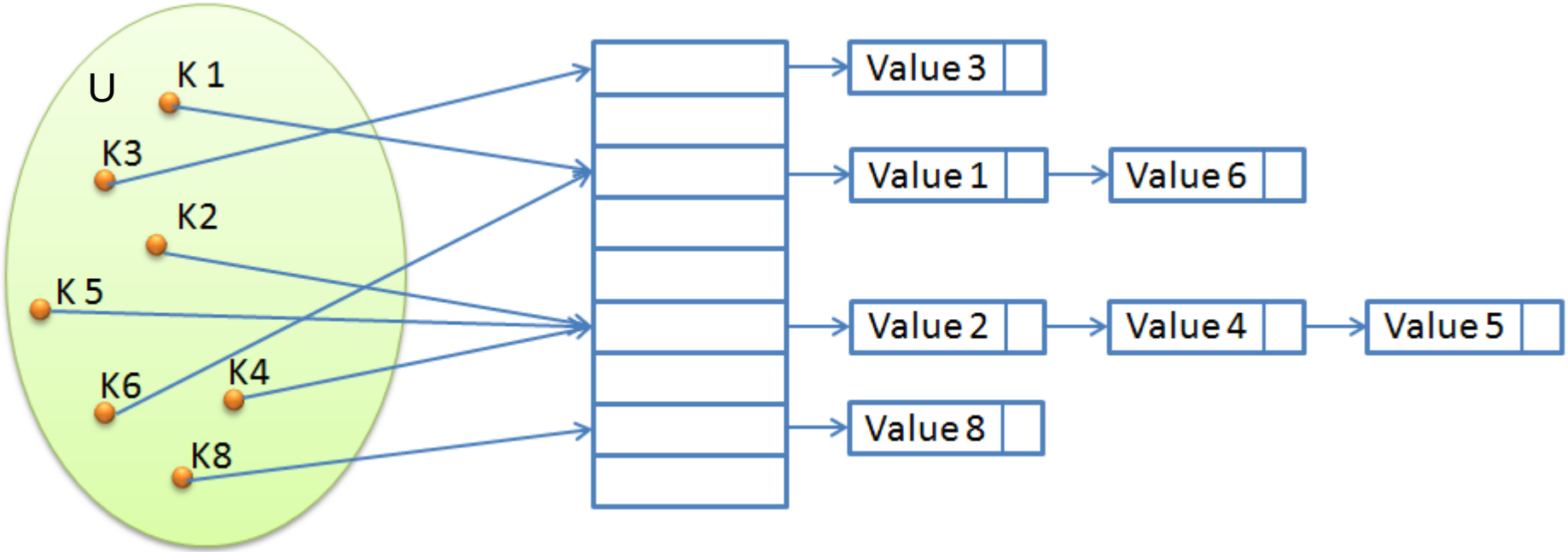
*Метод ланцюжків (Chaining, відкрите хешування).*

Елементи з однаковим значенням хеш-функції об’єднують у зв’язний список. Покажчик на список зберігають у відповідній чарунці хеш-таблиці.

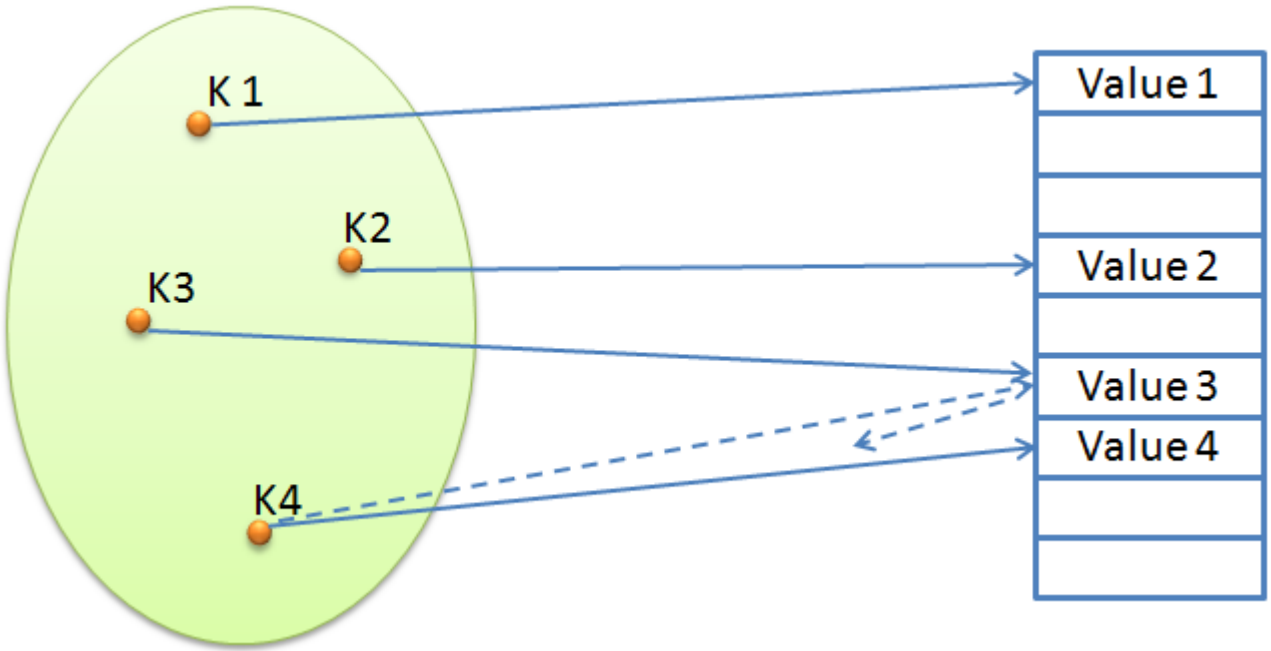
*Відкрита адресація (Open addressing, закрите хешування).*

У кожній чарунці хеш-таблиці зберігають не покажчик на зв’язний список, а один елемент (ключ, значення). Якщо чарунка з індексом hash (key) зайнята, то здійснюють пошук вільної чарунки в наступних позиціях таблиці

# Метод ланцюжків (Chaining)



# Відкрита адресація (Open addressing)




Hash(Key 4) = Hash(Key 3)

# Послідовність проб

- Лінійне хешування (linear probing)

$$h(k,i) = (h'(k) + i) \bmod m$$

- Квадратичне хешування

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

- Подвійне хешування.

$$h(k,i) = (h_1(k) + i h_2(k)) \bmod m$$

```
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"
```

```
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] = NIL або i = m
7  return NIL
```

# Проблема видалення елементів

Застосовуючи відкриту адресацію, у разі видалення ключа із чарунки не можна просто позначити її NIL. Інакше в послідовності проб для деякого ключа з’явиться порожня чарунка, а це призведе до неправильної роботи алгоритму пошуку. Для вирішення даної проблеми слід позначити чарунку, яку видаляють, деяким спеціальним значенням «Deleted», а у методі пошуку продовжувати пошук у разі виявлення цього позначення, у методі вставки – вставляти на його місце.

***Коефіцієнт заповнення хеш-таблиці*** *(load factor, fill factor)* $\alpha$ – відношення числа $N$ елементів хеш-таблиці до розміру таблиці $M$ (середнє число елементів на одну чарунку таблиці)

$$\alpha = N / M.$$

Від цього коефіцієнта залежить середній час виконання операцій додавання, пошуку й видалення елементів.

# Час роботи

| | Лучший случай. | В среднем. Метод цепочек. | В среднем. Метод открытой адресации. | Худший случай. |
|---|---|---|---|---|
| Поиск | $O(1)$ | $O(1+\alpha)$ | $O\left(\frac{1}{1-\alpha}\right)$ | $O(N)$ |
| Вставка | $O(1)$ | $O(1+\alpha)$ | $O\left(\frac{1}{1-\alpha}\right)$ | $O(N)$ |
| Удаление | $O(1)$ | $O(1+\alpha)$ | $O\left(\frac{1}{1-\alpha}\right)$ | $O(N)$ |

# Застосування хеш-таблиць

- Асоціативні масиви та словники в мовах програмування
- Таблиці символів в компіляторах та інтерпретаторах
- Реалізація структури даних множини (Set)
- Індекси в базах даних
- Деякі реалізації кеш-пам’яті – пам’яті зі швидким доступом