Hash tables

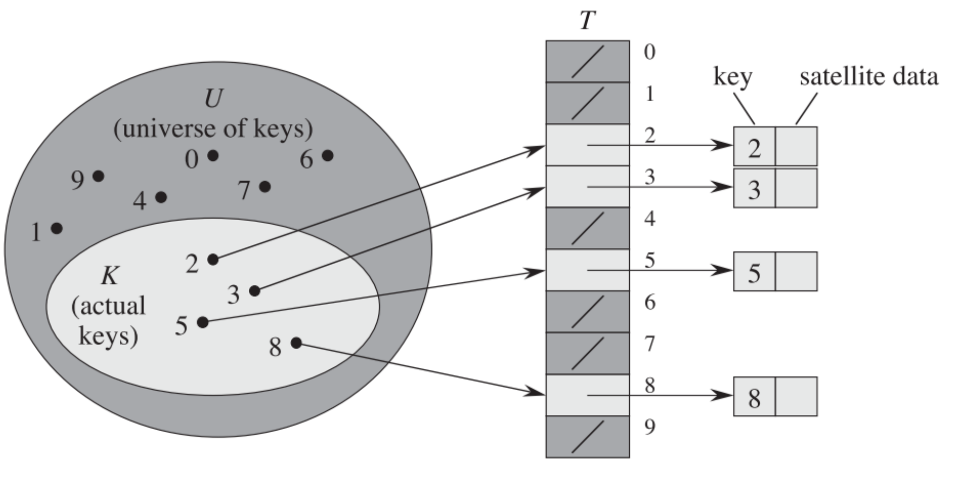
Yulia Burkatovskaya

Outline

- Direct-address tables
- Hash tables
- Hash functions
- Open addressing

Direct-address tables

- Direct addressing is a simple technique that works well when the universe U of keys is reasonably small.
- No two elements have the same key.
- We use an array, or ***direct-address table,*** T[0,…,m-1]. Each position, or slot, corresponds to a key.

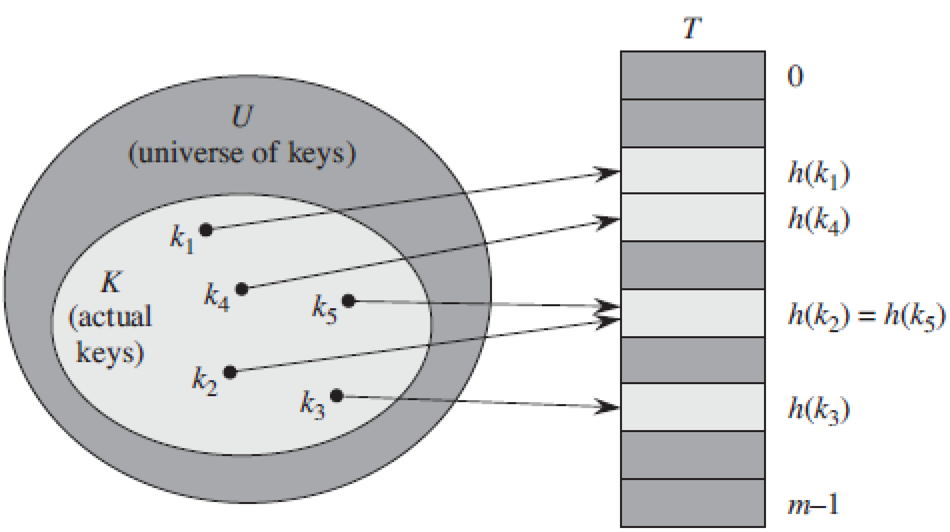


Direct-address tables

	Find	Insert	Delete
Unsorted array	N	1	N
Sorted array	log N	N	N
Linked list	N	1	1
File	N	N	N
Direct-address table	1	1	1

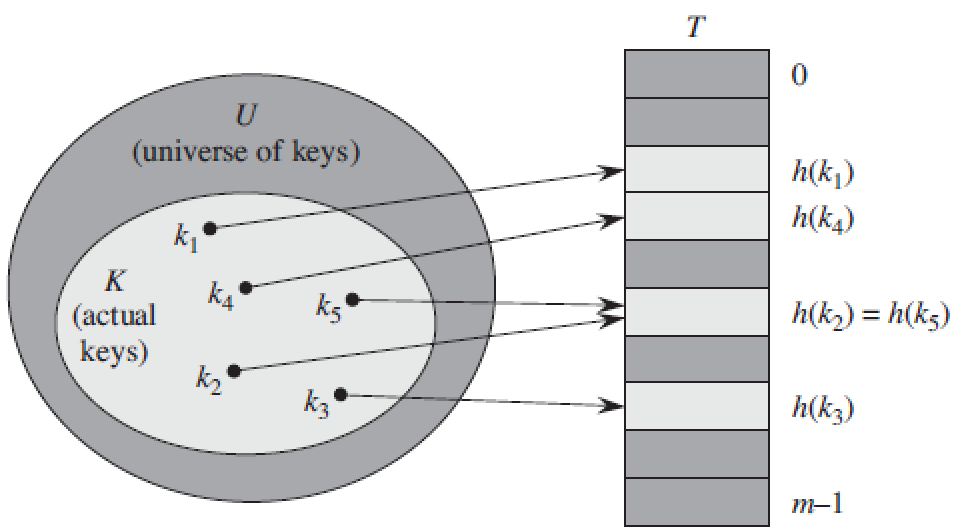
Hash tables

- If universe U is **large** then direct-address tables take a huge memory.
- The set of actually stored keys, say K, could be rather **small**.
- Reduce the storage requirement to m=O(|K|), T[m].
- **Hash function**: h: K→{0,…,m-1}



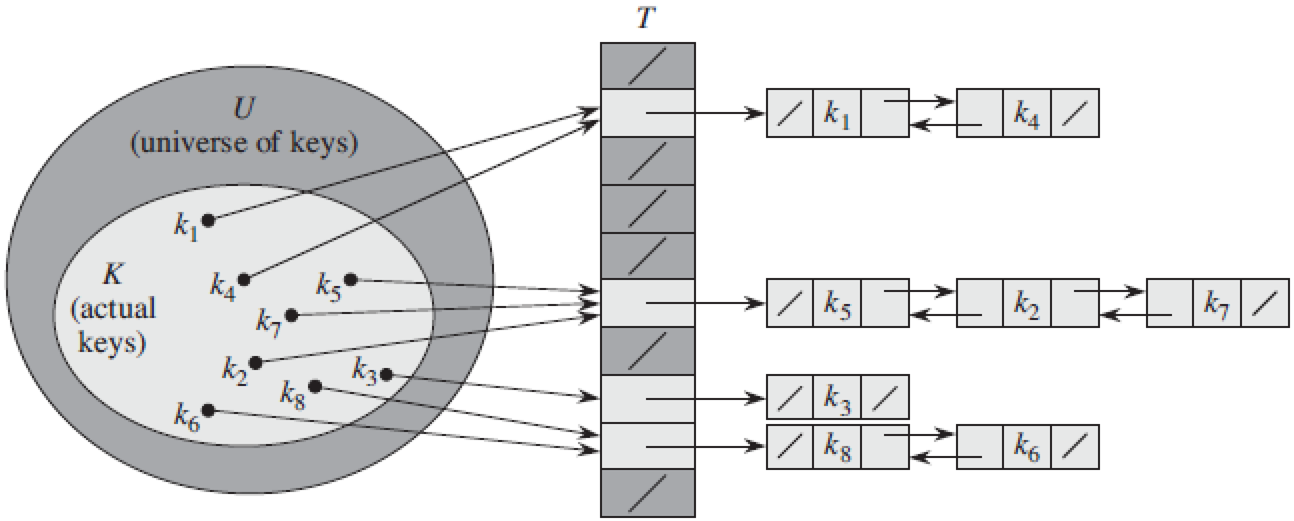
Hash tables

- **Collision**: two keys hash to the same slot.
- Good hash function – **random**, to minimize the number of collisions.
- **Solutions?**



Hash tables

- **Collision resolution by chaining**
- Elements hashing the same slot are placed in a linked list.



Hash tables

- **T: m slots, n elements**
- **Load factor**: $\alpha=n/m$ (the average number of elements stored at one slot)
- **Worst case**: all elements are in the same slot (**terrible**)
- **Average case**: depends on the hash function
- **Simple uniform hashing:** any given element is equally likely to hash into any of the m slots, independently of where any other element has hashed to.
- n_j – the number of elements in slot j
- $N=n_1+\ldots+n_m$

Hash tables

- **Theorem**
- In a hash table in which collisions are resolved by chaining, an unsuccessful search takes average-case time $O(1+\alpha)$, under the assumption of simple uniform hashing
- **Theorem**
- In a hash table in which collisions are resolved by chaining, an unsuccessful search takes average-case time $O(1+\alpha)$, under the assumption of simple uniform hashing.

- A is an average length of a list, 1 – to compute the hash function.

Hash functions

- **What makes a good hash function?**
- A good hash function satisfies (approximately) the assumption of simple uniform hashing.
- Unfortunately, we typically have **no way to check** this condition, since we rarely know the probability distribution from which the keys are drawn. Moreover, the keys might not be drawn independently.

- **Example**. U – integers, h(x) = x mod m.
- **Challenge**: a good hash function for identifiers?

Hash functions

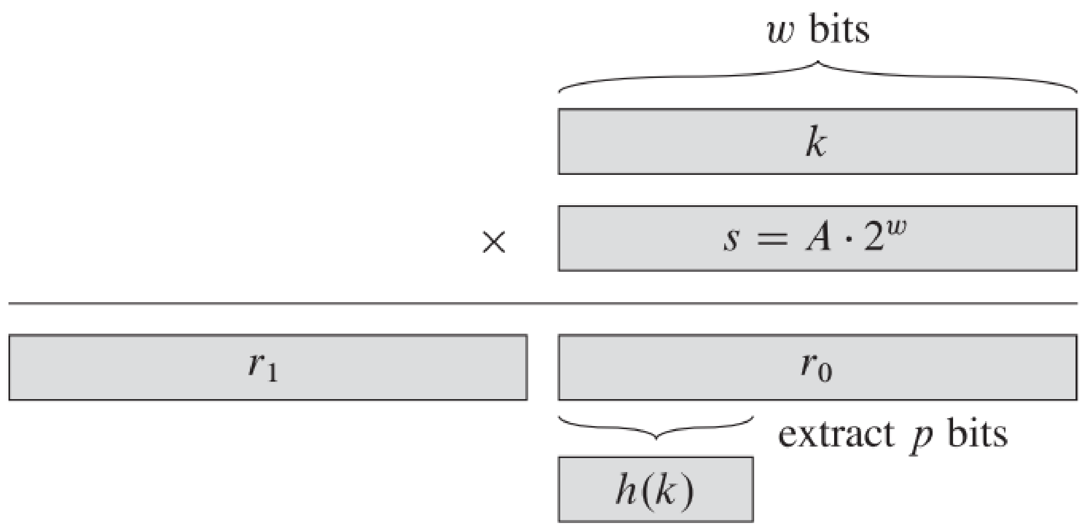
- **The division method**
- $h(x) = x \bmod m$.
- How to choose **m**?
- **Bad choice**: $m=2^p$ ($h(x)$= lowest p bits, it's better when a hash function depends on all bits)
- **Good choice** (usually) a prime not to close to 2^p
  - **Example**: $N=2000$, we don't mind $\alpha=3$; so, $m=701$.

Hash functions

- **The multiplication method**
- First, we multiply the key k by a constant A in the range 0<A<1 and extract the fractional part of kA.
- Then, we multiply this value by m and take the floor of the result.
- The value of m is not critical (2^p)
- $A = \left(\sqrt{5} - 1\right)/2$ (Donald Knuth)
- **Example**. k=103, m=23
  - A=0.037 kA=3.811, kA mod 1 = 0.811
  - m(kA mod 1) = 18.653, floor(18.653)=18

Hash functions

- **The multiplication method**



Hash function

- **Universal hashing**
- **The worst case**: all n keys share the same slot. It can be for any fixed hash function.
- **Idea**: to choose the hash function **randomly** and **independent on the keys**, which are going to be stored.
- **Example**. $h_{ab}(k) = \big((ak + b)\ mod\ p\big)\ mod\ m$
- Number p is prime and large enough, p>m
- If $k \neq l$ then $(ak + b)\ mod\ p \neq (al + b)\ mod\ p$
- $PR\Big\{\big((ak + b)\ mod\ p = (al + b)\ mod\ p\big)\ mod\ m\Big\} \leq \frac{1}{m}$

Hash function

- **Universal hashing**

Suppose that a hash function h is chosen randomly from a universal collection of hash functions and has been used to hash n keys into a table T of size m, using chaining to resolve collisions. If key k is not in the table, then the expected length $\mathrm{E}[n_{h(k)}]$ of the list that key k hashes to is at most the load factor $\alpha = n/m$. If key k is in the table, then the expected length $\mathrm{E}[n_{h(k)}]$ of the list containing key k is at most $1 + \alpha$.

Open addressing

- Each table entry contains either an element of the dynamic set or NIL, no linked lists.
- When searching for an element, we systematically examine table slots until either we find the desired element or we have ascertained that the element is not in the table.
- To perform insertion using open addressing, we successively examine, or **probe**, the hash table until we find an empty slot in which to put the key.
- In open addressing, the hash table can "fill up" so that no further insertions can be made; one consequence is that the load factor α can never exceed 1.

Open addressing

- To determine which slots to probe, we extend the hash function to include the probe number (starting from 0) as a second input
- $h: U \times \{0, \dots, m-1\} \to \{0, \dots, m-1\}$
- The **probe sequence** is a permutation of $\{0, \dots, m-1\}$ **(no repetitions!)**
- $h(k, 0), \dots, h(k, m-1)$

Open addressing

```
HASH-INSERT(T, k)
i = 0
repeat
    j = h(k, i)
    if T[j] == NIL
        T[j] = k
        return j
    else i = i + 1
until i == m
error "hash table overflow"
```

```
HASH-SEARCH(T, k)
i = 0
repeat
    j = h(k, i)
    if T[j] == k
        return j
    i = i + 1
until T[j] == NIL or i == m
return NIL
```

Open addressing

- **Linear probing**
- $h(k, i) = (h'(k) + i) mod\ m$
- **Example**: m=7. 5, 6, 7, 0,…,4
- **Good**: easy to implement
- **Bad**: primary clustering (long sequences of occupied slots)

Open addressing

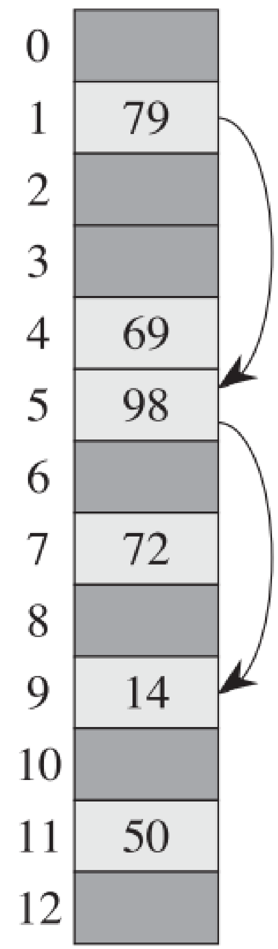
- **Quadratic probing**
- $h(k, i) = (h'(k) + c_1 i + c_2 i^2) mod\ m$
- $(h'(k) + c_1 + c_2) mod\ m, (h'(k) + 2c_1 + 4c_2) mod\ m, \ldots$
- **Good**: better then linear
- **Bad**: secondary clustering (two elements with the same initial slot have the same probe sequence), limitations for constants.

Open addressing

- **Double hashing**
- $h(k, i) = (h_1(k) + ih_2(k))mod\ m$
- **Example**: $h_1(k) = k\ mod\ m, h_2(k) = 1 + k\ mod\ m'$
- **Good**: the probe sequences depends in two ways on k; so, two elements with the same initial slot can have different probe sequences
- $h_2(k)$ should be relatively prime to m

Open addressing

- **Double hashing**
- **Example**: insert 14
  - $h_1(k) = k \ mod \ 13,$
  - $h_2(k) = 1 + k \ mod \ 11$



Open addressing

- Analysis of open addressing

Given an open-address hash table with load factor $\alpha = n/m < 1$, the expected number of probes in an unsuccessful search is at most $1/(1-\alpha)$, assuming uniform hashing.

Given an open-address hash table with load factor $\alpha < 1$, the expected number of probes in a successful search is at most

$$\frac{1}{\alpha} \ln \frac{1}{1-\alpha} ,$$

assuming uniform hashing and assuming that each key in the table is equally likely to be searched for.