11. Hash Tables

Many applications require a **dynamic set** that supports only the **directory operations** `INSERT`, `SEARCH` and `DELETE`.

A hash table is a generalization of the simpler notion of an ordinary array.

Directly addressing into an ordinary array makes effective use of our ability to examine an arbitrary position in the array in $O(1)$ time, independent of the size of array n.

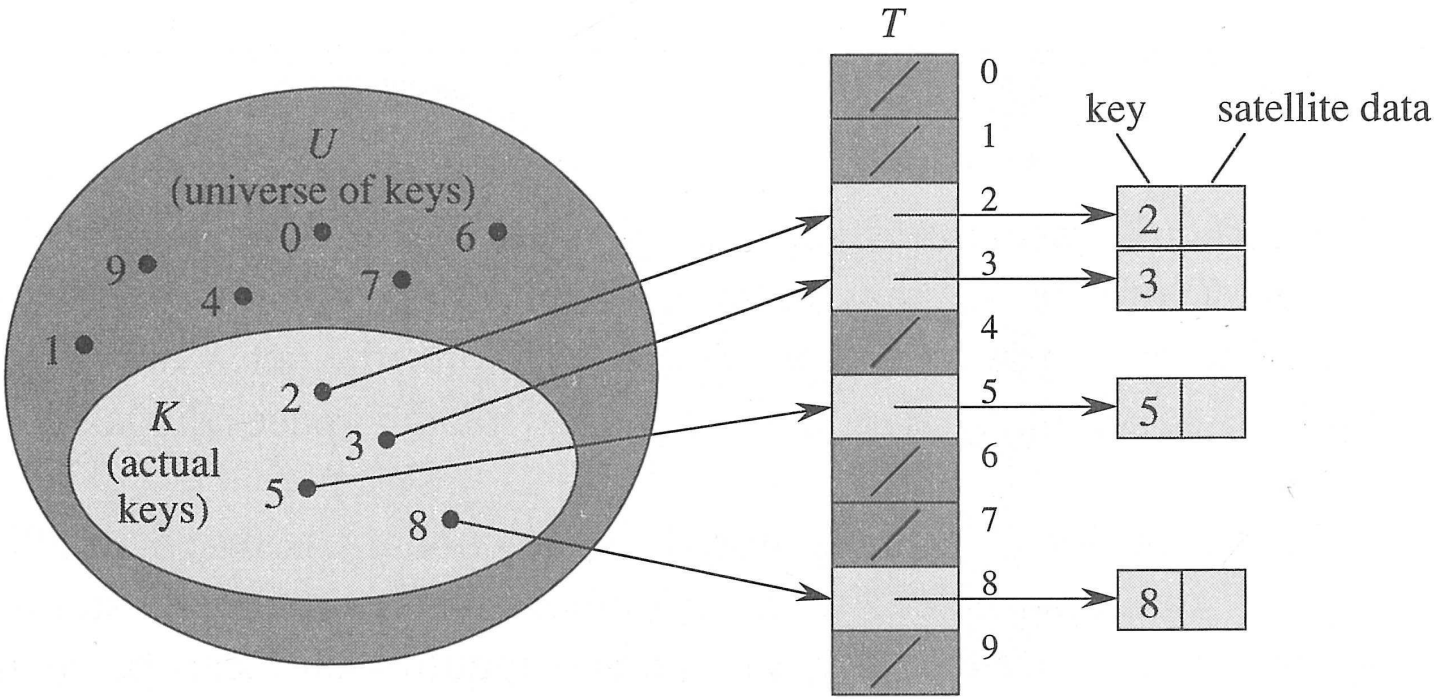
Direct addressing is a simple technique that works well when the universe U of the keys (all possible values of k) is reasonably small,

$$U = \{0, 1, \ldots, m-1\},$$

where

- m is not too large
- no two elements share the same key.

11.1. Direct-address tables



Direct-Addressing (Cormen et al., p254)

11.1 Operations on direct-address tables

To represent a dynamic set that has insertion, deletion, and searching operations, we use a direct-address table, denoted by $T[0..m-1]$, in which each position, or **slot**, corresponds to a key in the universe U.

- DIRECT_ADDRESS_SEARCH(T, k)
  return $T[k]$

- DIRECT_ADDRESS_INSERT(T, x)
  $T[key[x]] \leftarrow x$

- DIRECT_ADDRESS_DELETE(T, x)
  $T[key[x]] \leftarrow NIL$

11.2 Hash tables

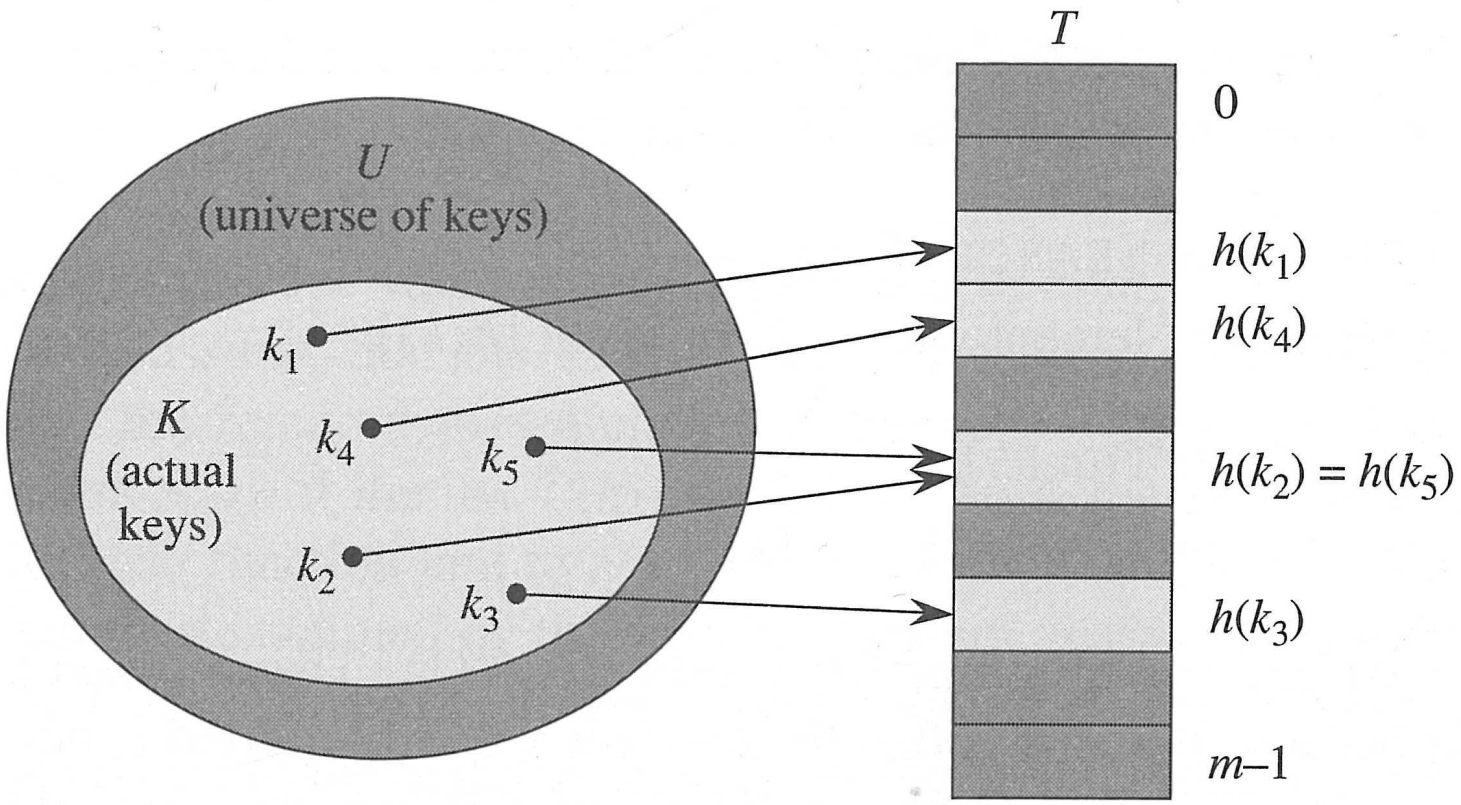
With hashing, the element is stored in slot $h(k)$,
i.e., we use a **hash function** h to compute the slot for the element using key k,
where h maps the universe U of keys into the slots of a **hash table** $T[0..m-1]$.

$$h: U \to \{0, 1, \ldots, m-1\}.$$

- We say that an element with key k hashes to slot $h(k)$.
- We also say that $h(k)$ is the hash value of key k.

Notice that with direct addressing, an element with key k is stored in slot k, which is a very special hash table.

11.2 Hash tables



Using a hash function (Cormen et al., p256)

11.2 Collision in Hash tables

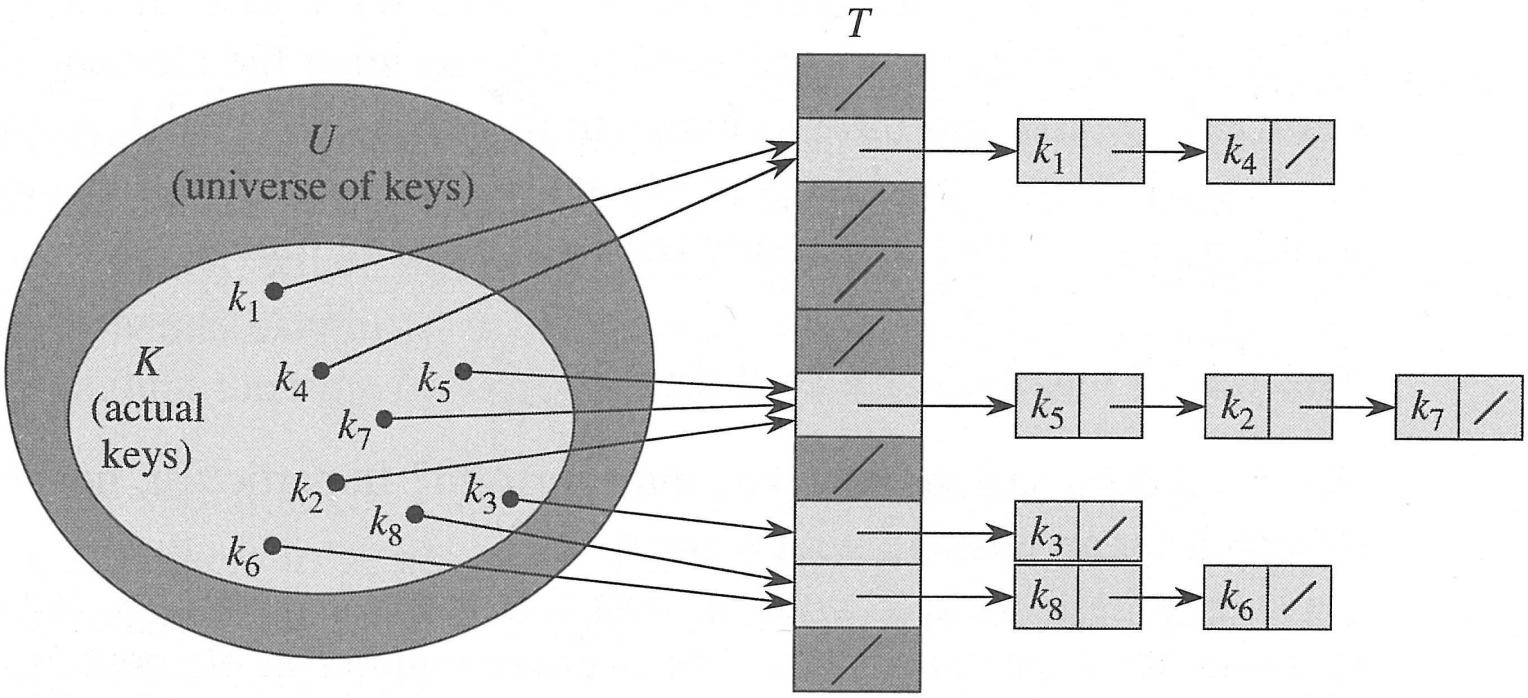
The drawback of any hash tables is the collision when two different keys are mapped to the same slot.

Resolving collisions is a key issue in the design of hash functions.

One effective way to resolve collisions is called chaining, which works as follows:

Put all elements that hash to the same slot in a linked list.

11.2 Collision in Hash tables



Avoiding collisions using linked lists (Cormen et al., p257)

11.2 Operations on hash tables

The directory operations on a hash table T are easy to implement when collisions are resolved by chaining.

- CHAINED_HASH_SEARCH(T, k)
  search for an element with key k in the linked list $T[h(k)]$

- CHAINED_HASH_INSERT(T, x)
  insert x at the head of the linked list $T[h(key[x])]$ (Why?)

- CHAINED_HASH_DELETE(T, x)
  delete x from the linked list $T[h(key[x])]$ (How to delete x from the list?)

Hashing with chaining takes $O(n)$ time in the worst case when searching or deleting an element from the hash table.

11.2 Analysis of simple uniform hashing with chaining

Given a hash table with m slots that stores n elements, the load factor of the hash table is

$$\alpha = n/m.$$

A simple uniform hashing assumes that any given element is equally likely to hash into any of the m slots, independently of where any other element has hashed to.

The average behavior of hashing under the simple uniform hashing assumption is much better, which takes $\Theta(1+\alpha)$ time.

Let the hash table contain m slots. For $j = 0, \ldots, m-1$, denote by n_j the length of the linked list $T[j]$, so that $n = n_0 + n_1 + \ldots + n_{m-1}$, and the average value of n_j is $E[n_j] = \alpha = n/m$.

What is the relationship between the load factor α and the time of searching/deletion of an element?

11.2 Analysis of simple uniform hashing with chaining (cont.)

Theorem Collisions in a hash table are resolved by chaining, an unsuccessful search (or a successful search) takes time $\Theta(1+\alpha)$ **in expectation** under the assumption of simple uniform hashing.

Case 1: Unsuccessful search for key k:

The linked list $T(j)$ for hash value $h(k)$ $(= j)$ has to traversed. The expected length of $T(j)$ is $E[n_j] = \alpha = n/m$.

Case 2: Successful search for key k: Let $k_i = key[x_i]$. For keys k_i and k_j, denote by $X_{ij} = I\{h(k_i) = h(k_j)\}$ a random variable. $Pr\{h(k_i) = h(k_j)\} = 1/m$. Thus, $E[X_{ij}] = 1/m$.

Assume that key k_i is hashed to a slot $h(k_i)$, when we retrieve the linked list that contains key k_i is from its head, we can find all keys k_j in front of key k_i with $j > i$ (**Why?**). In other words, the amount of time spent on this linked list (to identify k_i) is proportional to the number of keys before it in the linked list, while the probability of a key k_j in front of key k_i in the linked list with $j > i$ is $1/m$. The average time complexity of successful search for key k_i thus is

11.2 Analysis of simple uniform hashing with chaining (cont.)

$$
\begin{aligned}
E[\frac{1}{n}\sum_{i=1}^{n}(1+\sum_{j=i+1}^{n}X_{ij})] &= \frac{1}{n}(n+E[\sum_{i=1}^{n}\sum_{j=i+1}^{n}X_{ij}]) \\
&= \frac{1}{n}(n+\sum_{i=1}^{n}\sum_{j=i+1}^{n}E[X_{ij}]) \\
&= \frac{1}{n}(n+\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{1}{m}), \quad \text{as } E[X_{ij}] = 1/m \\
&= 1+\frac{1}{n}\sum_{i=1}^{n}\frac{n-i}{m} \\
&= 1+\frac{n-1}{2m} \\
&= 1+(\frac{n}{2m}-\frac{n}{m}\cdot\frac{1}{2n}) \\
&= 1+\alpha/2-\alpha/2n, \quad \text{since } \alpha = n/m \\
&= \Theta(1+\alpha). \qquad (1)
\end{aligned}
$$