Introduction to Data Structures

The Search Problem

- Find items with keys matching a given search key
  - Given an array A, containing n keys, and a search key x, find the index i such as x=A[i]
  - As in the case of sorting, a key could be part of a large record.

example of a record

Key	**other data**

Hashing

- Aimed towards faster search performance
- Search techniques covered
  - Linear/sequential search
  - Binary search
- When log(*n*) is just too big…
  - air traffic control
  - packet routing
- What hashing does
  - Storage location depends on the item

Hashing: Usefulness

- Some other examples where hashing helps
  - Click count of webpages
  - Number of connections from a url
  - Reservations in a given flight
  - Unique terms from a book
  - List of visited cells/nodes in an application
- Basic idea: item itself determines (narrows down) where the element can be found
  - H:Item domain->set of storage locations
  - i=h(k) is where the item is present
  - (Not always true, but provides starting point)

The Search Problem

- Find items with keys matching a given search key
  - Given an array A, containing n keys, and a search key x, find the index i such as x=A[i]
  - As in the case of sorting, a key could be part of a large record.

example of a record

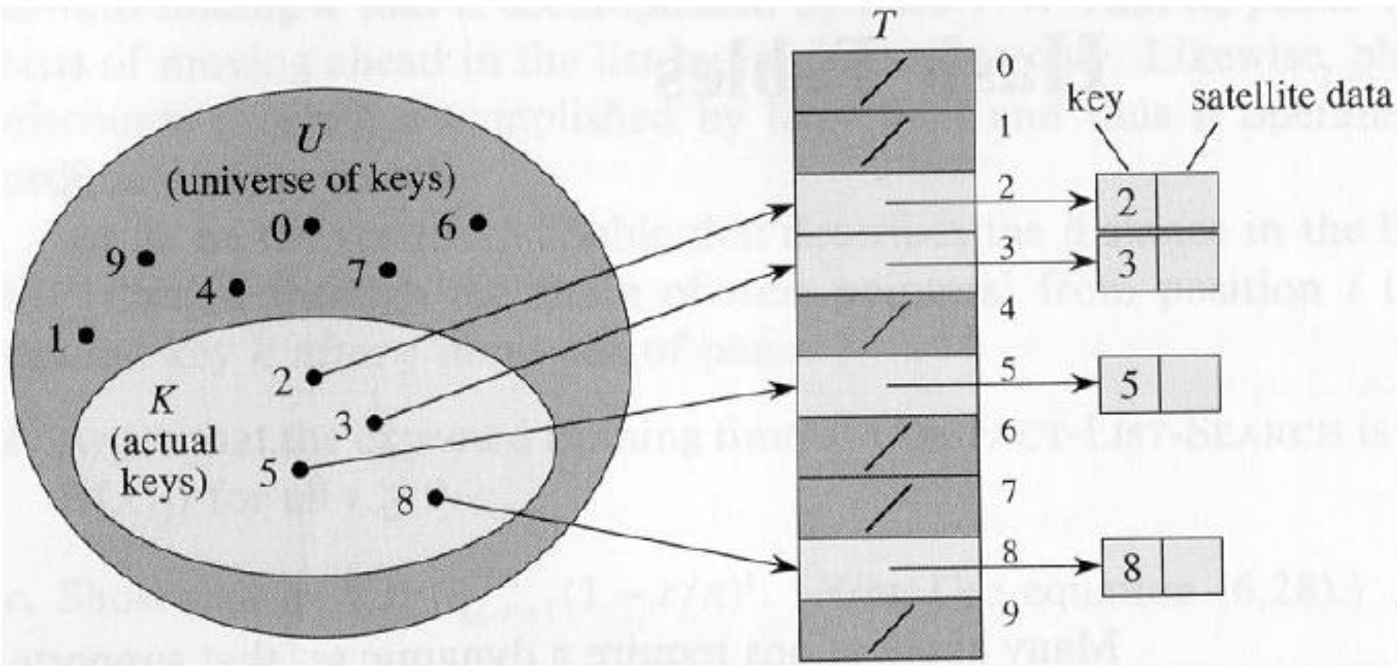
Key	**other data**

Direct Addressing

- Assumptions:
  - Key values are distinct
  - Each key is drawn from a universe U = {0, 1, . . . , m - 1}
- Idea:
  - Store the items in an array, indexed by keys

- **Direct-address table** representation:
  - An array T[0 . . . m - 1]
  - Each **slot**, or position, in T corresponds to a key in U
  - For an element x with key k, a pointer to x (or x itself) will be placed in location T[k]
  - If there are no elements with key k in the set, T[k] is empty, represented by NIL

Direct Addressing (cont'd)



(insert/delete in O(1) time)

Operations

Alg.: DIRECT-ADDRESS-SEARCH(T, k)

  return $T[k]$

Alg.: DIRECT-ADDRESS-INSERT(T, x)

  $T[key[x]] \leftarrow x$

Alg.: DIRECT-ADDRESS-DELETE(T, x)

  $T[key[x]] \leftarrow NIL$

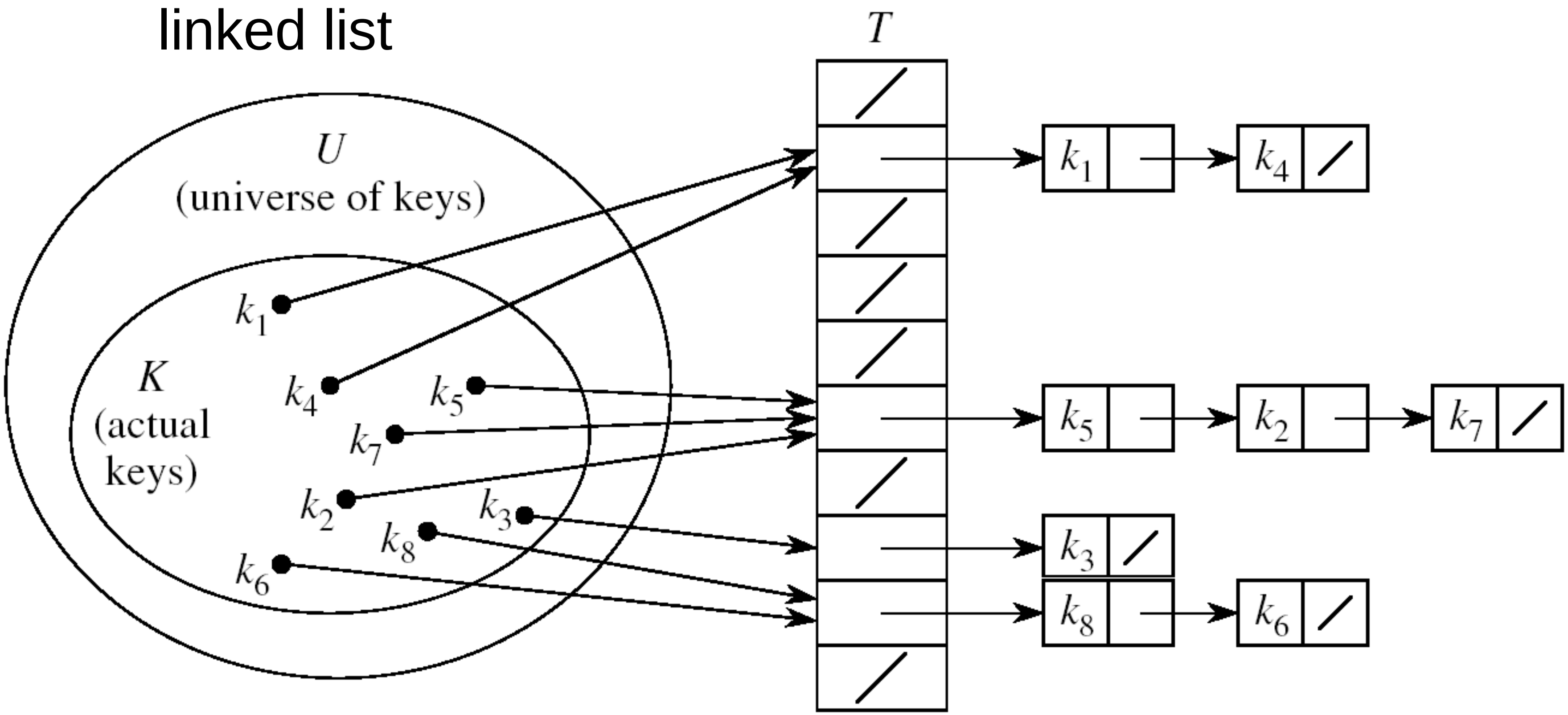
- Running time for these operations: $O(1)$

Handling Collisions

- We will review the following methods:
  - Open addressing
    - Linear probing
    - Quadratic probing
    - Double hashing
  - Chaining

Handling Collisions Using Chaining

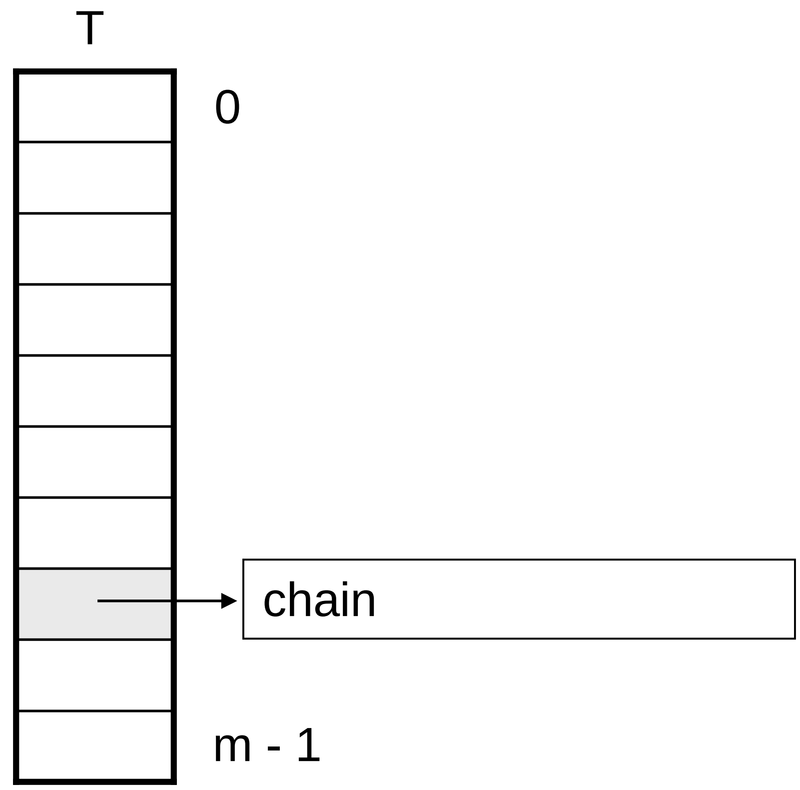
- **Idea**:
  - Put all elements that hash to the same slot into a linked list



  - Slot j contains a pointer to the head of the list of all elements that hash to j

Analysis of Hashing with Chaining: Worst Case

- How long does it take to search for an element with a given key?
- Worst case:
  - All n keys hash to the same slot
  - Worst-case time to search is $\Theta(n)$, plus time to compute the hash function
  - Expected time?



Successful Search

$$X_{ij} = I\{x_i \text{ and } x_j \text{ hash to same bucket}\}$$

$$\begin{aligned}
\mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n} X_{ij}\right)\right] \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n} \mathrm{E}\left[X_{ij}\right]\right) \quad \text{(by linearity of expectation)} \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n} \frac{1}{m}\right) \\
&= 1+\frac{1}{nm}\sum_{i=1}^{n}(n-i) \\
&= 1+\frac{1}{nm}\left(\sum_{i=1}^{n} n-\sum_{i=1}^{n} i\right) \\
&= 1+\frac{1}{nm}\left(n^2-\frac{n(n+1)}{2}\right) \quad \text{(by equation (A.1))} \\
&= 1+\frac{n-1}{2m} \\
&= 1+\frac{\alpha}{2}-\frac{\alpha}{2n}\,.
\end{aligned}$$

Successful Search

Successful search: $\Theta(1 + \frac{a}{2})$=$\Theta(1 + a)$ time on the average

(search half of a list of length a plus $O(1)$ time to compute $h(k)$)

Analysis of Search in Hash Tables

- If m (# of slots) is proportional to n (# of elements in the table):
- $n = O(m)$
- $\alpha = n/m = O(m)/m = O(1)$

$\Rightarrow$ Searching takes constant time on average