Dictionaries & Hashing

Dictionary Example

- Suppose you want to provide enhanced caller ID capability (eg Truecaller) for entire COL100 class:
  - –given a phone number, return the caller's name,
  - –may store additional info eg roll no, Lab Group No etc
  - –phone numbers are in the range 0 to $R = 10^{10}-1$
  - –n is the number of phone numbers used
  - –want to support inserts & deletes of items reasonably efficiently
  - –want to do this as efficiently as possible

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**

Dictionaries

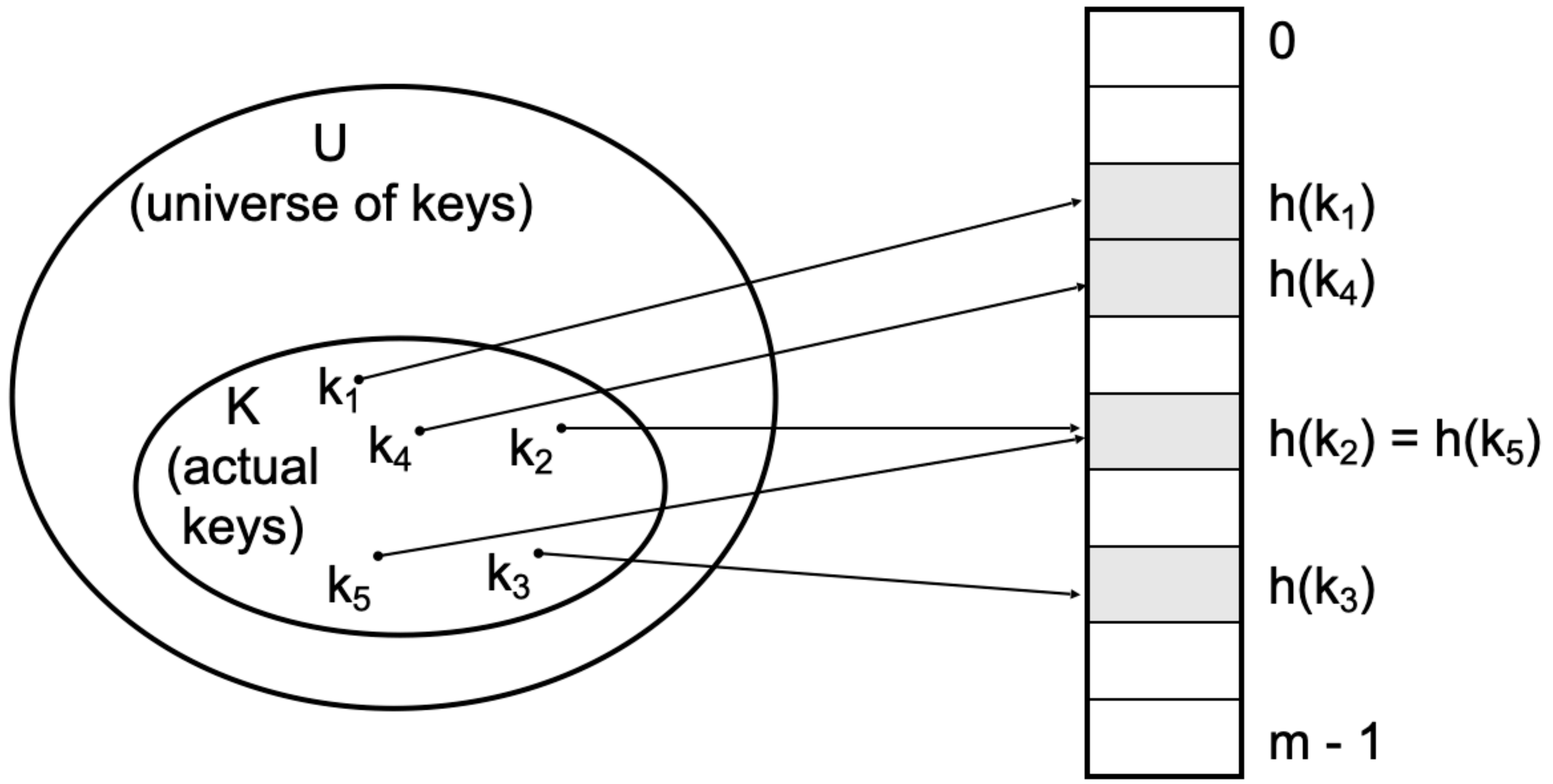
- **Dictionary** = data structure that supports mainly two basic operations: *insert* a new item and *return an item with a given key*
- Queries: return information about the set S:
  - Search (S, k)
- Modifying operations: change the set
  - Insert (S, k)
  - Delete (S, k) – not very often — desirable

Hashing

- A ***Hash Table*** is an alternative solution with O(1) expected query time and O(n + N) space, where N is the size of the table
- Like an array, but with a function to map the large range of keys into a smaller one
  - -e.g., take the original key, ***mod*** the size of the table, and use that as an index
- Insert item (9018637639, Rahul) into a table of size 5
- -9018637639 mod 5 = 4, so item (9018637639, Rahul) is stored in slot 4 of the table
- A lookup uses the same process: map the key to an index, then check the array cell at that index

Collision

- Insert (9018639350, Aditya) => slot 0
- now insert (9018632234, Devinder) => slot 4. We have a ***collision!***



- How to deal with two keys which map to the same cell of the array?

Collisions

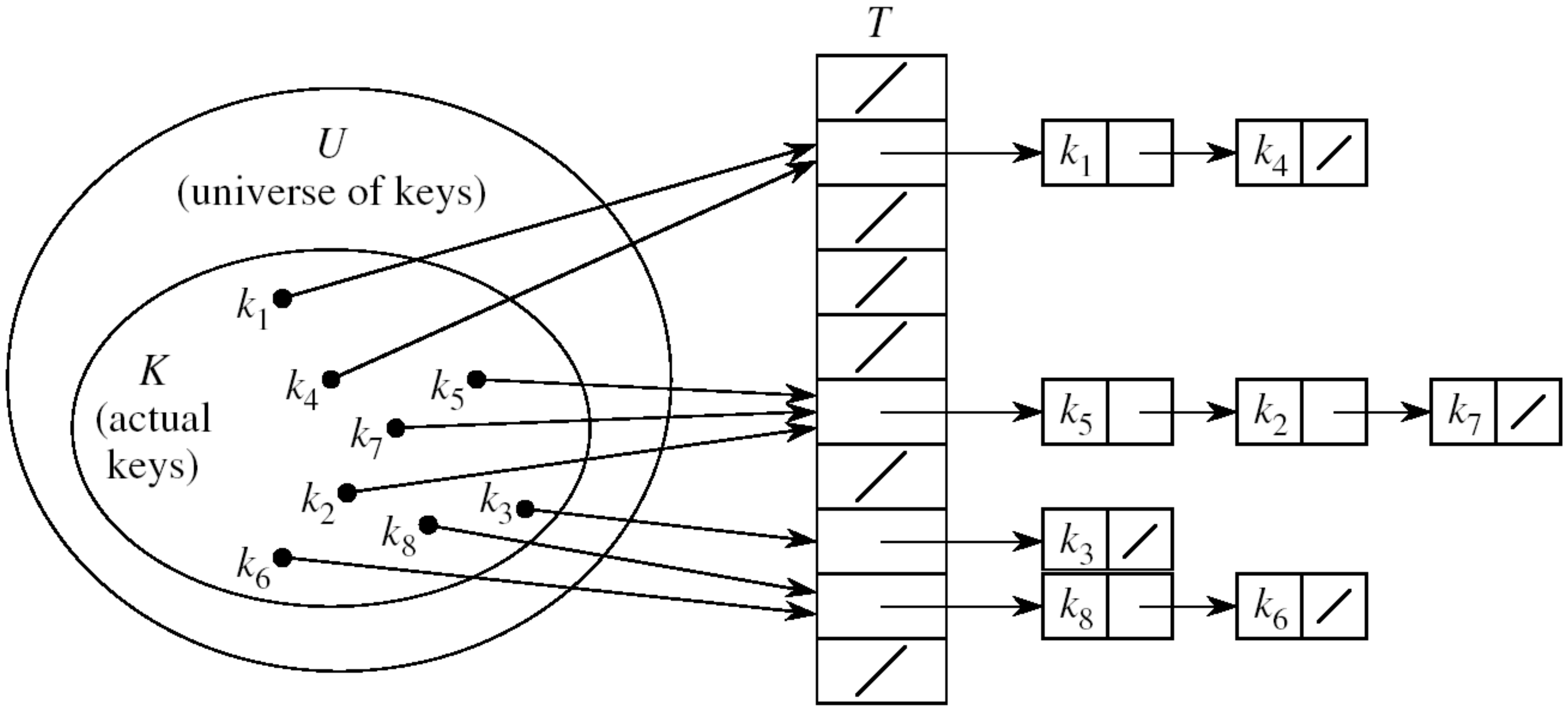
- Two or more keys hash to the same slot!!
- For a given set K of keys
  - If $|K| \leq m$, collisions may or may not happen, depending on the hash function
  - If $|K| > m$, collisions will definitely happen (i.e., there must be at least two keys that have the same hash value)

Avoiding collisions completely is hard, even with a good hash function

- The expected search/insertion time is O(n/m), provided the indices are uniformly distributed
- The performance of the data structure can be fine-tuned by changing the table size

Collision

- How to deal with two keys which map to the same cell of the array?
- Use ***chaining to set up lists*** of items with the same index — Slot j contains a list of all elements that hash to j



Hash function:

- The mapping of keys to indices of a hash table is called a ***hash function***
- A hash function is usually the composition of two maps, a ***hash code map*** and a ***compression map.***
  - –An essential requirement of the hash function is to map equal keys to equal indices
  - –A "good" hash function minimizes the probability of collisions

  - ***hash code map***: key → integer
  - ***compression map***: integer → $[0, N-1]$

- widely used way to define the hash of a string of length n using ***polynomial rolling hash function***

$$\begin{aligned}\text{hash}(s) &= s[0] + s[1] \cdot p + s[2] \cdot p^2 + \ldots + s[n-1] \cdot p^{n-1} \mod m \\ &= \sum_{i=0}^{n-1} s[i] \cdot p^i \mod m,\end{aligned}$$

- where p is a prime and m are some chosen, positive numbers

Popular Compression Maps

- Division: $h(k) = |k| \bmod N$
  - the table size N is usually chosen as a prime number
- Multiply, Add, and Divide: $h(k) = |ak + b| \bmod N$
  - same formula used in linear congruential (pseudo) random number generators

Python Implementation

```
def hashElem(e):
    global m,p
    if type(e) == int:
        val = e
    if type(e) == str:
        #Convert e to an int
        val = 0
        shift = 1
        for c in e:
            val = val + shift*ord(c)
            shift *= p
    return val%m
```

Searching in Hash Tables

Alg.: CHAINED-HASH-SEARCH(T, k)

search for an element with key k in list $T[h(k)]$

- Running time is proportional to the length of the list of elements in slot $h(k)$

Insertion in Hash Tables

Alg.: CHAINED-HASH-INSERT(T, x)

append x at the tail of list $T[h(key[x])]$

- It would take an additional search to check if it was already inserted

Deletion in Hash Tables

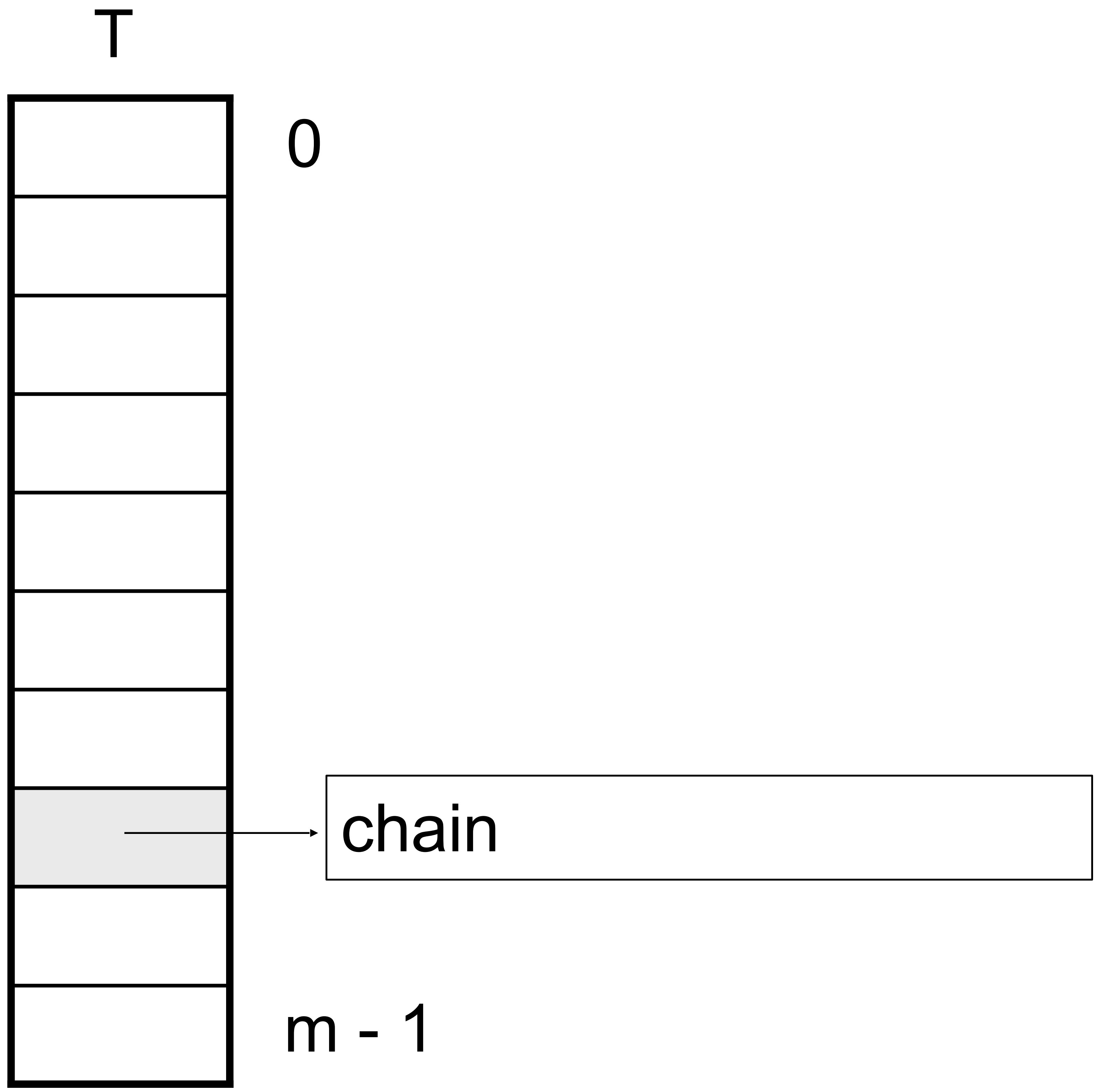
Alg.: CHAINED-HASH-DELETE(T, x)

delete x from the list T[h(key[x])]

- Need to find the element to be deleted.
- Worst-case running time:
  - Deletion depends on searching the corresponding list

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



Average case - Depends on how well the hash function distributes the n keys among the m slots

- **Simple uniform hashing assumption:**
  - Any given element is equally likely to hash into any of the m slots (i.e. probability of collision $Pr(h(x)=h(y))$ is $1/m$)
- Length of a list:

$$T[j] = n_j, \quad j = 0, 1, \ldots, m-1$$

- Number of keys in the table:

$$n = n_0 + n_1 + \cdots + n_{m-1}$$

- Average value of n_j:

$$E[n_j] = \alpha = n/m$$

- The expected, search/insertion/removal time is $O(n/m)$, provided the indices are uniformly distributed
- The performance of the data structure can be fine-tuned by changing the table size m

Uses of Hashing

- Widely used since it provide constant time search, insert and delete operations on average
- Example problems are, distinct elements, counting frequencies of items, finding duplicates, etc.
- **cryptographic Hash Functions**
  - **Passwords**
  - **Message Digests**
  - **One of the common cryptographic hash algorithms is SHA 256. The hash thus computed has a maximum size of 32 bytes.**

Python Dictionaries

Dictionary items are unordered, changeable, and does not allow duplicates. Dictionary items are presented in key:value pairs, and can be referred to by using the key name

```
IPL_team = {
...     'Punjab' : ‘Kings XI’,
...     'Rajasthan'    : 'Royals',
...     'Kolkata': ‘Knight Riders',
...     'Mumbai': 'indians',
...
... }
```

```
print(IPL_team[‘Mumbai’])
```

```
IPL_team[‘Kolkata’]=(‘Knight Riders’,’Brendon Mcullum’)
```

Adding an entry is simply a matter of assigning a new key and value: IPL_team[‘Delhi’]=‘Capitals’

`to delete a key:value from dict:  del IPL_team['Punjab']`