HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

DATA STRUCTURES & ALGORITHMS

HASHING

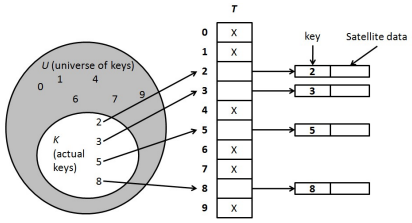
ONE LOVE. ONE FUTURE.

HASHING

- Mapping: a data structure storing objects/pairs (key, value)
  - put(*k*,*v*): Store an object having key *k* and value *v* to the data structure
  - get(*k*): return the object having key *k*
- Implementation
  - Binary Search Trees
  - Hash tables

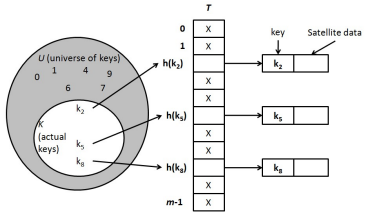
HASHING

- Direct addressing
  - Value of the key *k* is the address indicate the place in the table storing the pair (k,v)
  - Advantages: simple, fast lookup
  - Disadvantages: memory usage might be ineffective



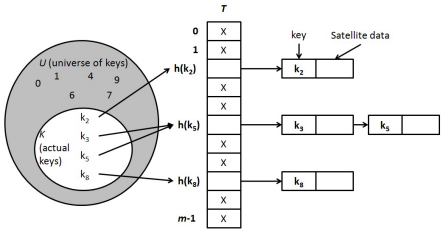
HASHING

- Hash function $h(k)$ specifies the address where the pair (k, *value*) is stored
- $h(k)$ should be simple and easy to compute



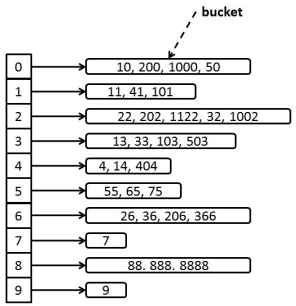
HASHING

- Collision: Two different keys have the same value of the hash function (hash code):
- Resolution:
  - Chaining: group keys having the same hash code into buckets
  - Open Addressing



HASHING: collision resolution by chaining

- Modulo: $h(k) = k \bmod m$ where m is the size of the hash table



HASHING: collision resolution by chaining

- Modulo: $h(k) = k \bmod m$ where m is the size of the hash table

HASHING: collision resolution by open addressing

- Pairs (key, value) are stored in the table itself
- Operations put(k, v) and get(k) need to probe the table until the desired slot found
  - put(k, v): probe for finding a free slot for storing (k, v)
  - get(k): probe for finding the slot where the key k is stored
  - Probing order: $h(k, 0)$, $h(k, 1)$, $h(k, 2)$, ..., $h(k, m\text{-}1)$
  - Methods
    - Linear probing: $h(k, i) = (h_1(k) + i) \bmod m$ where h_1 is normal hash function
    - Quadratic probing: $h(k, i) = (h_1(k) + c_1 i + c_2 i^2) \bmod m$ where h_1 is normal hash function
    - Double hashing: $h(k, i) = (h_1(k) + i * h_2(k)) \bmod m$ where h_1 and h_2 are normal hash functions

HASHING: collision resolution by open addressing

- get(k) and put(k, v) operations:

```
get(k)
{
  // T: the table
  i = 0;
  while(i < m) {
    j = h(k,i);
    if(T[j].key = k) {
      return T[j];
    }
    i = i + 1;
  }
  return NULL;
}
```

```
put(k, v)
{
  // T: the table
  x.key = k; x.value = v;
  i = 0;
  while(i < m) {
    j = h(k,i);
    if(T[j] = NULL) {
      T[j] = x; return j;
    }
    i = i + 1;
  }
  error("Hash table overflow");
}
```

HASHING: collision resolution by open addressing

- Exercise: A table has m slots, apply the open addressing method in which $h(k, i)$ has the form:

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

- Initialization, the table is free, present the status of the table after inserting following sequence of keys 7, 8, 6, 17, 4, 28 into the table with m = 10

HASHING: collision resolution by open addressing

- Exercise: A table has m slots, apply the open addressing method in which $h(k, i)$ has the form:

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

- Initialization, the table is free, present the status of the table after inserting following sequence of keys 7, 8, 6, 17, 4, 28 into the table with m = 10

0	1	2	3	4	5	6	7	8	9
28	x	x	x	4	x	6	7	8	17

HASHING: hash functions

- Key k is an integer
  - $h(k)$ = mod m
- Key is a string
  - $k = s[0..n\text{-}1] \rightarrow h(k) = (s[0]*256^{n-1} + s[1]*256^{n-2} + \ldots + s[n\text{-}1]*256^{0}) \bmod m$

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !