CHAPTER 8
HASHING

Iris Hui-Ru Jiang Fall 2008

Hashing

- **Contents**
  - Static hashing
    - Hash tables
    - Hash functions
    - Overflow handling
- **Readings**
  - Chapter 8

Dictionary (Dynamic Set) Revisited

- **Definition: A dictionary is**
  - A collection of pairs: key and an associated element
  - To support operations:
    - Queries: search, min, max, rank, successor, predecessor
    - Modifications: insert, delete
- **Implementation: binary search tree vs. hash**
  - Binary search tree is good for a dictionary if tree height h is well controlled
  - Hash table is good for dictionary that ordering is not important

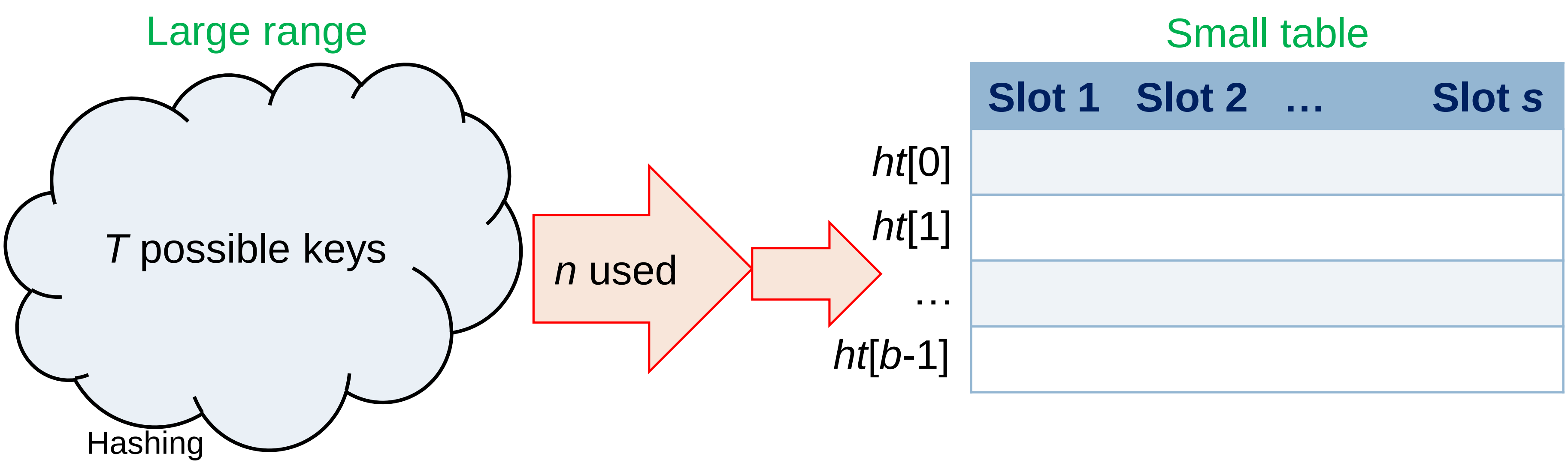
Operation	Binary Search Tree	Hash Table
Search	O(h)	O(1)
Insert	O(h)	O(1)
Delete	O(h)	O(1)

4

Hash Tables

Static Hashing

- **The dictionary pairs are stored in a table *ht* (hash table)**
  - b buckets: $ht[0]$, $ht[1]$, …, $ht[b-1]$
  - s slots in a bucket
- **Definition:**
  - Key density: n/T: very small for reasonable applications
  - Loading density / factor: $\alpha = n/(sb)$
- **What if each possible key occupies one distinct bucket?**
  - Be very fast but waste memory



Hash Functions

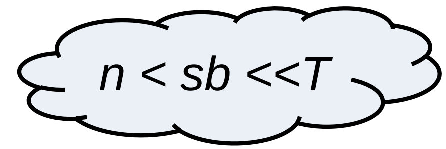
Division

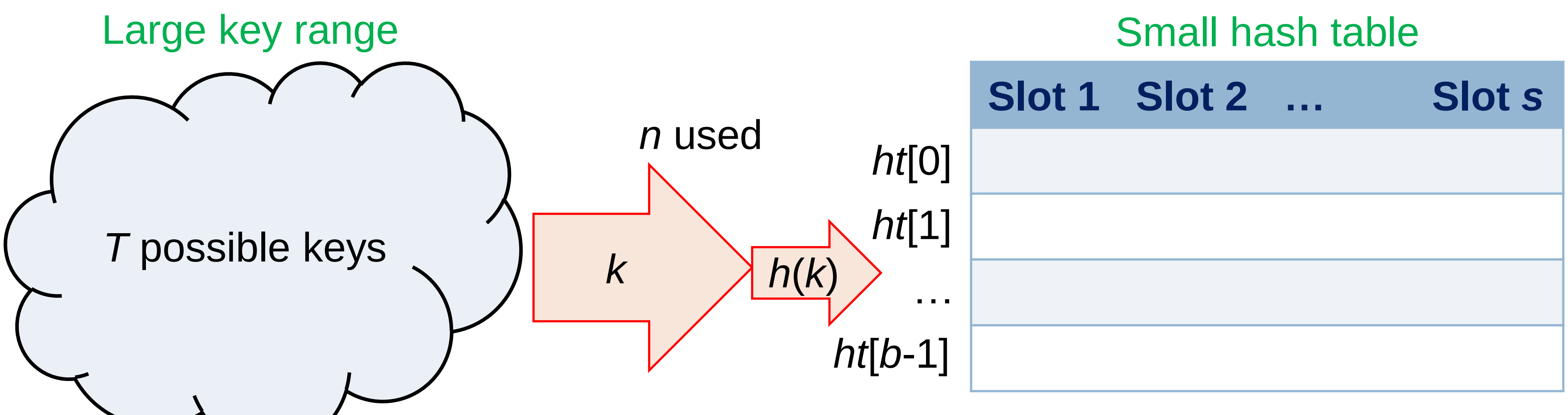
Mid-square

Folding

Where to Store?

- **Use a hash function to map keys into hash table buckets**
  - Is easy to compute
  - Minimize the # of collisions
    - Collision: two keys map to the same home bucket: $h(k_1) = h(k_2)$
    - Overflow: a key maps to a full bucket
- **Uniform hash function: a random key has an equal chance of hashing into any of the buckets**
  - Division, mid-square, folding





A Simple but Not Good Example

- **Keys: GA, D, A, G, L, A2, A1, A3, A4, E ⇒ n = 10**
- **Hash table: b = 26, s = 2 ⇒ α = 10/(26*2) = 0.19**
- **Hash function: h maps the first character A~Z to bucket 0~25**
  - Not good
  - Lots of collisions and overflows!

	Slot 1	Slot 2
[0]	A	A2
[1]		
[2]		
[3]	D	
[4]	E	
[5]		
[6]	GA	G
…		
[25]		

Division

- ***h*(*k*) = *k* % *D***
  - Bucket addresses range from 0 to *D*-1
- **The choice of *D* is critical**
  - Bad idea: choose a power of 2 as *D*
  - Good idea: choose a prime number as *D*
    - For real-world dictionaries, the distribution of home buckets is biased whenever *D* has a prime smaller than 20.
- **Most widely used!**

Mid-Square

- **How?**
  1. Square the key
  2. Use an appropriate # of bits from the middle of the square
- **Implementation**
  - Where is middle?
    - Depend on all bits of the key
  - How many bits?
    - r bits for 2^r buckets

Folding

- **How?**
  1. Partition the key into several parts
  2. Add all partitions together
- **How to add?**
  - Example key *k* = 12320324111220 = 123_203_241_112_20
  - Shift folding: *h*(*k*) = 123+203+241+112+20 = 699
  - Folding at the boundaries: *h*(*k*) = 123+302+241+211+20 = 897

Overflow Handling

Open addressing

Chaining

Overflow Handling

- **Open addressing**
  - Linear probing: linearly scan the adjacent buckets
  - Quadratic probing: quadratically scan the adjacent buckets
  - Rehashing: use a series of hash functions
  - Random hashing: randomly use one of a pool of hash functions
- **Chaining**
  - Chain together

Linear Probing (1/2)

- **If overflow occurs at bucket $h(k)$, find an unfilled bucket from its next bucket in ascending order**
  - $h'(k) = (h(k) + i)\ \%\ b,\ 0 \le i \le b-1$
- **Example:**
  - Keys: GA, D, A, G, L, A2, A1, A3, A4, Z, ZA, E
  - Hash table: $b = 26$, $s = 1$
  - Hash function: h uses the first character
  - Summary for key retrieval:
    - 39 buckets examined
    - On average, 39/12 = 3.25 buckets accessed

[0]	A	1
[1]	A2	2
[2]	A1	3
[3]	D	1
[4]	A3	5
[5]	A4	6
[6]	GA	1
[7]	G	2
[8]	ZA	10
[9]	E	6
[10]		
[11]	L	1
…		…
[25]	Z	1

Linear Probing (2/2)

- **Search key k in a hash table with s = 1, procedure**
  1. Compute $h(k) = i$
  2. Examine buckets in the order $ht[i]$, $ht[i+1]$, …, $ht[(i+j)\%b]$ until:
     - $ht[i+j] == k$: k is found!
     - $ht[i+j]$ is empty: k is not in the table
     - return to the starting point ($ht[i]$): table full and k is not there
- **The expected number of key comparisons to retrieve a key is approximately $(2 - \alpha)/(2 - 2\alpha)$**
  - E.g., $\alpha = 0.47 \Rightarrow 1.5$ for the previous example
  - The worst case can be quite large

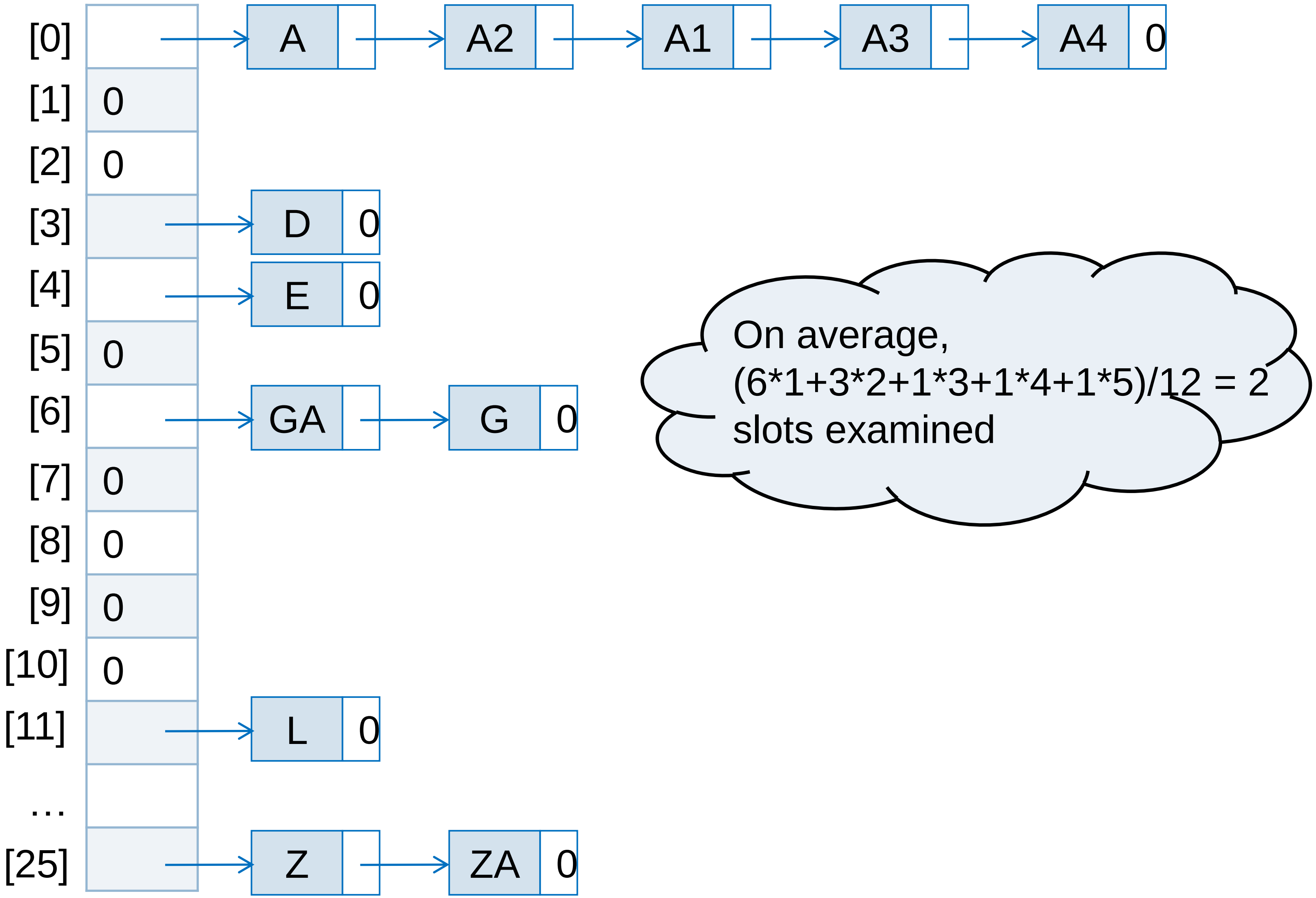
Quadratic Probing

- **If overflow occurs at bucket *h*(*k*), find an unfilled bucket from its adjacent buckets and skip quadratic sizes of buckets**
  - $h'(k) = (h(k) \pm i^2)$ % b, $0 \leq i \leq (b-1)/2$
- **If the bucket size *b* is a prime and of form 4*j*+3, quadratic probing can examine every bucket**
  - e.g., 3, 7, 11, 19, 23, 31, 43, 59, 127, 251, 503, 1019, …

Chaining (1/2)

- **Overflow occurs because the slot size is fixed**
  - Make it dynamic
- **Chaining: each bucket has a linked list**
  - Stores synonyms (keys hashed to the same bucket)
  - Has a variant size
- **The expected key comparisons is approximately $1 + \alpha/2$, where α is *n*/*b***
  - Better search time! (cf. linear probing with $(2 - \alpha)/(2 - 2\alpha)$)

Chaining (2/2)


[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
[10]
[11]
…
[25]
0
A
A2
A1
A3
A4
D
E
GA
G
L
Z
ZA
On average,
(6*1+3*2+1*3+1*4+1*5)/12 = 2
slots examined

Empirical Comparison

- **For a uniform hash function, performance depends only on overflow handling**
  - Best: division + chaining
- **Average # of bucket accesses per key retrieved**

$\alpha = n/b$	0.50		0.75		0.90		0.95	
Hash Function	Chain	Open	Chain	Open	Chain	Open	Chain	Open
division	1.19	4.52	1.31	7.20	1.38	22.42	1.41	25.79
mid square	1.26	1.73	1.40	9.75	1.45	37.14	1.47	37.53
shift fold	1.33	21.75	1.48	65.10	1.40	77.01	1.51	118.57
bound fold	1.39	22.97	1.57	48.70	1.55	69.63	1.51	97.56
digit analysis	1.35	4.55	1.49	30.62	1.52	89.20	1.52	125.59
theoretical	1.25	1.50	1.37	2.50	1.45	5.50	1.48	10.50