Hashing - II

Instructor: Ashok Singh Sairam

Lecture Plan

- The hash function
- Methods of generating hash function
  - Division method
  - Multiplication method
  - Universal Hashing
- Designing Universal class of hash function
- Types of hashing
  - Closed Hashing/Open addressing
  - Open hashing/Chainging

Hash Functions

- A hash function transforms a key into a table address
- Properties of good hash function

  (1) Easy to compute

  (2) Approximates a random function: for every input, every output is equally likely (simple uniform hashing)

  (Possible if we know in advance the probability distribution that keys are drawn from)

Interpreting keys as natural numbers

- Assume universe of keys is $\mathbb{N} = \{0, 1, 2 \ \ldots\}$
- A character can be interpreted as integer expressed in suitable radix notation
- For example, interpret **pt**
  - ASCII value of p=112, t = 116
  - Expressed as a radix-128 integer

$(pt)_{128} = (112 \times 128 + 116) = 14452$

The Division Method

- **Idea:**
  - Map a key k into one of the m slots by taking the remainder of k divided by m

$$h(k) = k \bmod m$$

- **Advantage**:
  - fast, requires only one operation
- **Disadvantage**:
  - Certain values of m are bad, e.g.,
    - power of 2
    - non-prime numbers

Division method: Example

- If $m = 2^p$, then $h(k)$ is just the least significant p bits of k
  - $p = 1 \Rightarrow m = 2$
    $\Rightarrow h(k) = \{0, 1\}$, least significant 1 bit of k
  - $p = 2 \Rightarrow m = 4$
    $\Rightarrow h(k) = \{0, 1, 2, 3\}$, least significant 2 bits of k
- Good Heuristic: Choose m to be a prime, not close to a power of 2 or 10
  - Column 2: k mod 97
  - Column 3: k mod 100

	m 97	m 100
16838	57	38
5758	35	58
10113	25	13
17515	55	15
31051	11	51
5627	1	27
23010	21	10
7419	47	19
16212	13	12
4086	12	86
2749	33	49
12767	60	67
9084	63	84
12060	32	60
32225	21	25
17543	83	43
25089	63	89
21183	37	83
25137	14	37
25566	55	66
26966	0	66
4978	31	78
20495	28	95
10311	29	11
11367	18	67

The Multiplication Method

Idea:

- Multiply key k by a constant A, where $0 < A < 1$, i.e. kA
- Extract the fractional part of kA ,
  $(kA \ mod \ 1) = kA \ - \ \lfloor kA \rfloor$
- Multiply the fractional part by m
- Take the floor of the result

$$h(k) = \lfloor m \ (k \ A \bmod 1) \rfloor$$

- **Disadvantage:** Slower than division method
- **Advantage:** Value of m is not critical, e.g., typically 2^p

Example – Multiplication Method

- $m=2^p$ (Typically choose m to be a power of 2)
- Computer has w-bit word

  $h(k) = (A.k \bmod 2^w) >>(w-p)$
- A is an odd integer $2^{w-1}<A<2^w$
- $m=2^3$, w=6
- h(110101)=011

.101101 (A)
110101 (k)

1001010.0110011 (kA)
discard 1001010
shift .0110011 by 3 bits 011.0011
take integer part: 011

Universal Hashing

- For any choice of hash function, ∃ worst case keys, that all hash into the same slot
- How to beat the adversary?
  - Select a hash function at random, independent from the keys. This approach is called universal hashing
- Universal hashing: Select hash function at random from a carefully designed class of hash functions

Universal Hashing: Definition

- H: Finite collection of hash function that maps
  $$U \to \{0,1,..,m-1\}$$
- H is said to be universal if $\forall x, y \in U$, where x ≠ y
  $$|\{h \in H, h(x) = h(y)\}| = \frac{|H|}{m}$$
- If *h* is chosen randomly from *H*, the probability of collision between *x* and *y* is $1/m$
  $$\Pr\big(h(x) = h(y)\big) = \frac{|H|/m}{H} = 1/m$$

#keys hashing into same slot

- Theorem: If you choose *h* randomly from *H* (set of universal hash function), for n arbitrary distinct keys

$$E[\#keys\ hash\ into\ same\ slot] \leq 1 + \propto$$

Designing a Universal Class of Hash Functions

- Choose a prime number p large enough so that every possible key k is in the range $[0 \ldots p - 1]$

  $Z_p = \{0, 1, \ldots, p - 1\}$ and $Z_p^* = \{1, \ldots, p - 1\}$
- Define the following hash function

  $h_{a,b}(k) = ((ak + b) \bmod p) \bmod m,$

  $\forall\ a \in Z_p^*$ and $b \in Z_p$
- The family of all such hash functions is

  $\mathcal{H}_{p,m} = \{h_{a,b} : a \in Z_p^* \text{ and } b \in Z_p\}$
- a , b: chosen randomly at the beginning of execution

  $|\mathcal{H}_{p,m}| = p(p-1)$

Universal Hash Functions: Example

$p = 17, m = 6$

$$h_{a,b}(k) = ((ak + b) \bmod p) \bmod m$$

$$h_{3,4}(8) = ((3 \cdot 8 + 4) \bmod 17) \bmod 6$$

$$= (28 \bmod 17) \bmod 6$$

$$= 11 \bmod 6$$

$$= 5$$

Hashing Types

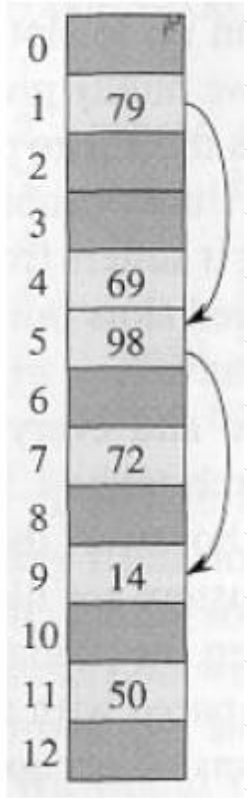
- Two broad types

1. Closed Hashing/Open Addressing
2. Open Hashing/Chaining

Open Addressing

- No chaining, all elements occupy the hash table itself
- Notion of probing
  - **Insertion**: If the slot is full, try (probe) another slot until an empty slot is found
  - The hash function specifies the order of slots to probe for a key
  - Follow the same sequence during **search** and **delete**
- Search time depends on the length of the probe sequence!

e.g., insert 14



Generalize hash function notation

- A hash function contains two arguments now:
  (i) Key value, and (ii) Probe number

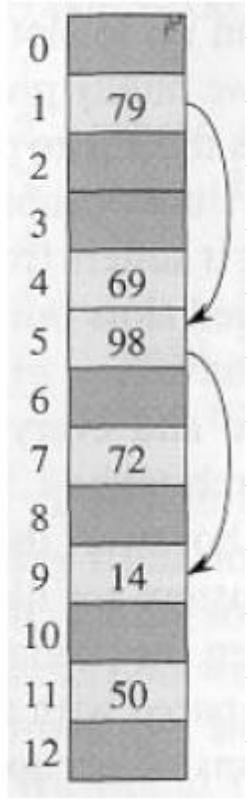
  $h(k,p)$, $p=0,1,\ldots,m-1$

- Probe sequences

  $\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$

  - Must be a permutation of $\langle 0,1,\ldots,m-1 \rangle$
  - There are $m!$ possible permutations
  - Good hash functions should be able to produce all $m!$ probe sequences

e.g., insert 14



Example

<1, 5, 9>

Insertion

- Insert key k in the hash table T
  - Assume the keys are with no satellite information

```
HASH-INSERT(T, k)
1  i = 0
2  repeat
3       j = h(k, i)
4       if T[j] == NIL
5            T[j] = k
6            return j
7       else i = i + 1
8  until i == m
9  error "hash table overflow"
```

Search

- Takes as input hash table T and key *k*, returns *j* the slot that contains key k, or NIL if key not present

```
HASH-SEARCH(T, k)
1  i = 0
2  repeat
3       j = h(k, i)
4       if T[j] == k
5            return j
6       i = i + 1
7  until T[j] == NIL or i == m
8  return NIL
```

Deletion

- Encounters an empty that was previously occupied, it will fail incorrectly
- Solve by putting a flag “Deleted”
- Modify Insert to treat such a slot as empty
- Search need not be modified

Open addressing methods

- We will examine three common techniques used to compute the probe sequences required for open addressing

1. Linear probing
2. Quadratic probing
3. Double hashing

- Note: None of these methods can generate more than m^2 different probing sequences!

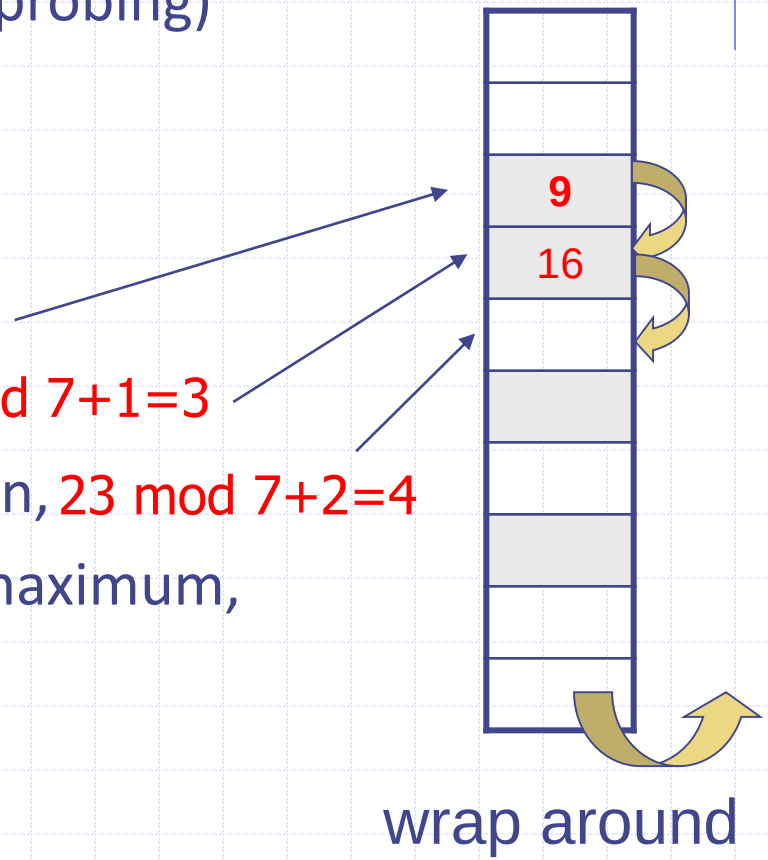
Linear probing: Inserting a key

- Idea: when there is a collision, check the next available position in the table (i.e., probing)

$$h(k,i) = (h_1(k) + i) \bmod m$$

$$i=0,1,2,\ldots$$

- First slot probed: $h_1(k)$, 23 mod 7=2
- Second slot probed: $h_1(k) + 1$, 23 mod 7+1=3
- Third slot probed: $h_1(k)+2$, and so on, 23 mod 7+2=4
- Can generate m probe sequences maximum, why?



Primary clustering problem

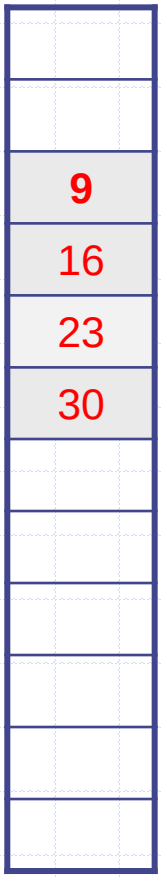
- Linear probing suffers from a problem known as primary clustering
- Some slots become more likely than others
- Long chunks of occupied slots are created

$\Rightarrow$ average search time increases!!

initially, all slots have probability 1/m

- For example, we use the hash fn

$$h(k,i) = (h_1(k) + i) \bmod m$$
$$h_1(k) = k \bmod 7$$



Quadratic probing

- Uses a hash function of the form

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

- The initial position probed is T[h′(k)], later positions probed are offsets that depend in a quadratic manner
- Works better than linear probing
  - Primary clustering due to **similar** keys will not occur
  - But secondary clustering can occur due to **identical** keys

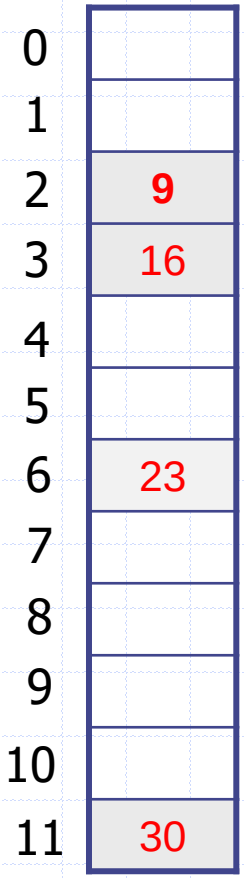
Quadratic probing: example

- The auxiliary hash function is same as before

$$h(k,i) = (h_1(k) + i^2) \bmod m$$
$$h_1(k) = k \bmod 7$$

- The elements to be inserted are

  9, 16, 23, 30



Double Hashing

- One of the best methods available for open addressing

$$h(k,i) = (h_1(k) + i\,h_2(k)) \bmod m$$

  where both h_1 and h_2 are auxiliary hash functions
- Initial probe: $h_1(k)$
- Second probe is offset by $h_2(k)$ mod m, so on ...
- Advantage: avoids clustering
- Disadvantage: harder to delete an element
- Can generate m^2 probe sequences maximum

Double Hashing: Example

$h_1(k) = k \bmod 13$

$h_2(k) = 1 + (k \bmod 11)$

$h(k,i) = (h_1(k) + i\, h_2(k)) \bmod 13$

- Insert key 14:

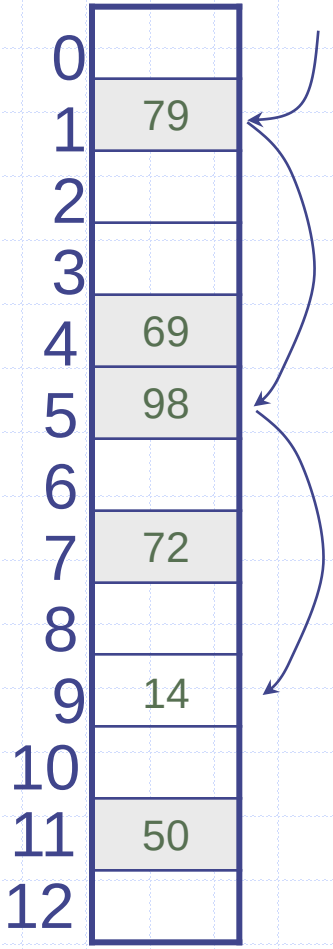
$h_1(14,0) = 14 \bmod 13 = 1$

$h(14,1) = (h_1(14) + h_2(14)) \bmod 13$

$= (1 + 4) \bmod 13 = 5$

$h(14,2) = (h_1(14) + 2\, h_2(14)) \bmod 13$

$= (1 + 8) \bmod 13 = 9$



Analysis of Open Addressing

- In open addressing m $\geq$ n
- Load factor $\alpha = {}^{m}/_{n} \leq 1$
- Thm: Expected #probes for unsuccessful retrieval is $\leq \frac{1}{1-\alpha}$
- Proof:

  Prob(probe hits an occupied cell)= α

  Prob(probe hits an unoccupied cell)= $1-\alpha$

  Prob(probe terminates in 2 trials)= $\alpha(1-\alpha)$

  Prob(probe terminates in k trials)= $\alpha^{k-1}(1-\alpha)$

  E(#steps)=$\sum_{k=1}^{m} k\alpha^{k-1}(1-\alpha) \leq \sum_{k=0}^{\infty} k\alpha^{k-1}(1-\alpha)$

  $=(1-\alpha)\frac{1}{(1-\alpha)^2} = \frac{1}{1-\alpha}$

- **Corollary**: Inserting an element into an open-addressing hash table with load factor α requires at most $1/(1-\alpha)$ probes on average, assuming uniform hashing

Analysis of Open Addressing - contd

- Expected #probes for successful retrieval is $\leq \frac{1}{\alpha} ln \frac{1}{1-\alpha}$

Acknowledgement

- Dr George Bebis, Foundation Professor, Dept of Computer Science and Engineering, University of Nevada Reno