Pseudorandom number, Universal Hashing, Chaining and Linear-Probing

COMP 480/580

10th Jan 2019

How are sequence of random numbers generated?

- Natural Experiments Based
  - Start flipping real coins.
  - Measure some physical phenomenon that is expected to be random and then compensates for possible biases in the measurement process. Example sources include measuring atmospheric noise, thermal noise, and other external electromagnetic and quantum phenomena.

- Pseudorandom Number Generator (PNG)
  - Seed based. Deterministic and will give the same sequence if the seed is same.

Idea behind PNG

- Middle-Square method (John von Neumann in 1949)
  - Start with n-digit seed. (Say 1111)
  - Square it (1234321 = 01234321) . (will be 2n digits, is not pad leading zeros.)
  - Report the middle n numbers as the next sequence. (2343)

- Issues?
  - N needs to be even.

Based on Weyl's Sequence

- Idea
  - The sequence of all multiples of such an integer integer k,
    - 0, k, 2k, 3k, 4k, … is equi-distributed modulo m if m is relatively prime to k.
  - That is, the sequence of the remainders of each term when divided by m will be uniformly distributed in the interval [0, m).

- The following C code is an example
  - d += 362437;
  - In this case, the seed 362437 is an odd integer.
- (Why?)

Simple Interview Question

- Given an array of n integers, remove duplicates

- Comparison Based:
  - O(n^2)

- Sorting Based
  - O(n log n)

- Hashing Based.
  - O(n). Hash tables. (if we can map n integers uniquely to [0-n-1] then in O(n) memory)

Hashing Basics

- Hash Function: Takes an object O and maps it to some discrete range $h(O) \in [0 - N]$.

- **Perfect Hash Function:**
  - if $O_1 \neq O_2 \quad then \quad h(O_1) \neq h(O_2)$
  - Guaranteed. O(1) search.
  - How to construct such functions.

- Given a dynamic collection of objects, allow for efficient searching and addition.

Universal Hashing (Approximate)

- Starting point
  - Design h: $objects \rightarrow [0 - N]$ such that
  - If $O_1 \neq O_2 \ \ then\ h(O_1) \neq h(O_2)$ most of the time.
  - h is cheap to compute and requires almost no memory.

- Or probability of h(O_1) = h(O_2). If h is perfectly random, then it is 1/N
  - Starting point of the definition.

2-Universal Hashing Family.

- A set (family) of hash function H is 2-universal if
  - for all x, y ∈ U such that $x \neq y$ we have

$$\Pr_{\{h \sim H\}}(h(x) = h(y)) \leq 1/\mathrm{N}$$

  where h ~ H means that h is selected uniformly at random from H.

The hash family H

- Choose a prime P > R
- Let, $h_{\{a,b\}} = \big((ax + b) mod\ P\big) mod\ N$

- The family is
  - H = $\{h_{\{a,b\}} \mid a \in \{1,2, \dots, P-1\}$ and $b \in \{0,1,2, \dots, P-1\}\}$

- Claim: For any x, y $\Pr_{\{h \sim H\}}(h(x) = h(y)) < 1/N$

- How about $\Pr_{\{h \sim H\}}(h(x) = h(y) = h(z))$

How to sample h uniformly from H

- Sample and fix.
  - Randomly generate a and b independently and uniformly and store them.
  - $h(x) = ax + b \; mod \; P \; mod \; R$
  - Allows us to compute h(x) and h(y) just via communication of a, b, P

- **Principle of deferred decision**
  - A probabilistic algorithm makes several stochastic choices based on coins (random outcomes). The following are equivalent
    - Either flip the coins during the experiments. (deferred)
    - Or flip the coins in advance and reveal their outcomes to algorithm when needed.

Class Exercise: Mod operation is slow

- $h(x) = ax + b \; mod \; P \; mod \; R$
  - a should not divide P
- Alternatives?

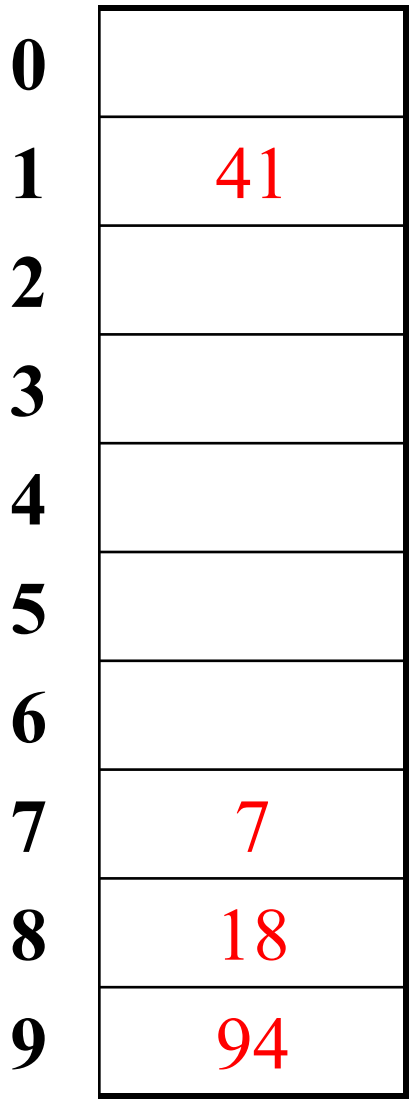
Back to the problem of searching.

Example

- key space = integers
- TableSize = 10

- **h**(K) = K mod 10

- **Insert**: 7, 18, 41, 94

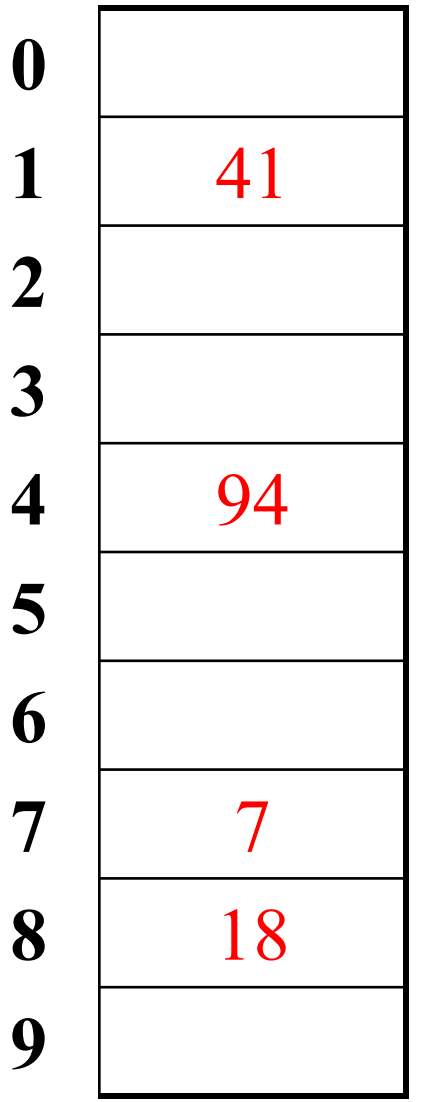


Example

- key space = integers
- TableSize = 10

- **h**(K) = K mod 10

- **Insert**: 7, 18, 41, 94



Collision Resolution

Collision: When two keys map to the same location in the hash table.

Two ways to resolve collisions:

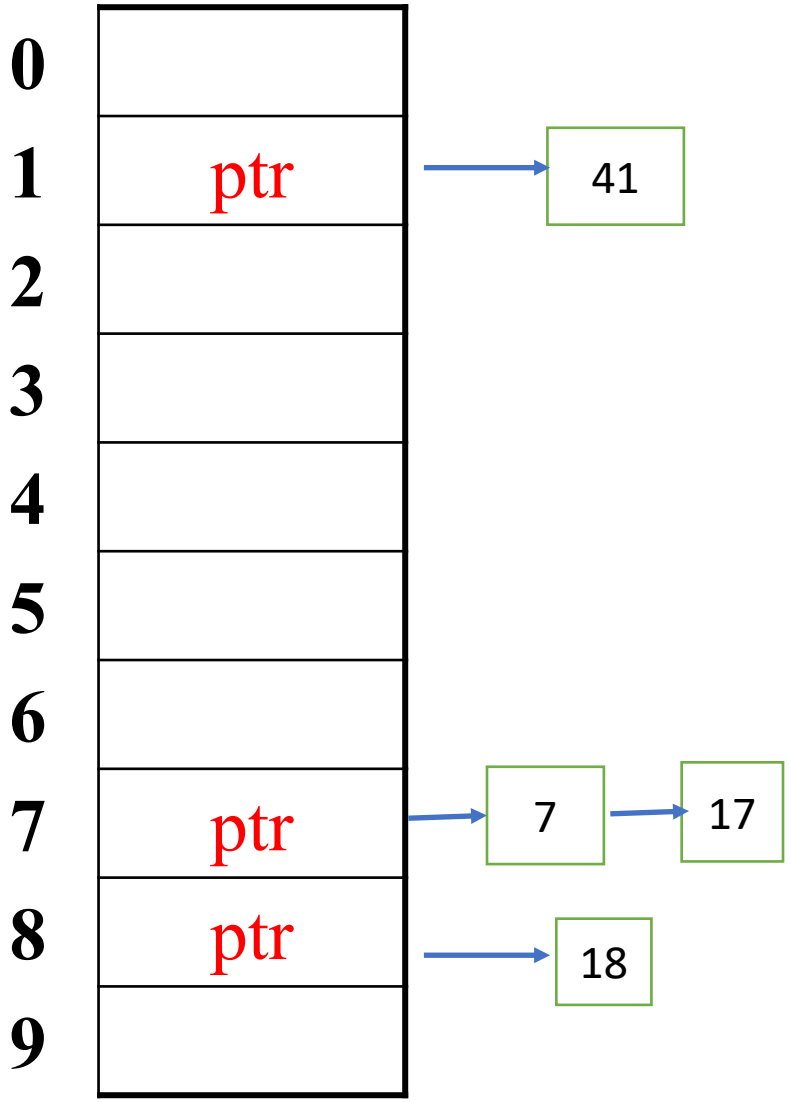
1. Separate Chaining
2. Open Addressing (linear probing, quadratic probing, double hashing)

Example: Separate Chaining.

- key space = integers
- TableSize = 10

- **h**(K) = K mod 10

- **Insert**: 7, 41,18, 17



Chaining with 2-universal hashing?

- We insert m objects into array of size N using 2-universal hashing.

- What is the worst case searching time of an object?

- What is the expected value of searching time with N array size and m objects inserted?
    - $< 1 + m-1/N$

Issues with Chaining?

- Linked List

Idea behind Probing

F(i)

- Probe sequence:

  0^{th} probe = h(k) mod TableSize
  1^{th} probe = (h(k) + f(1)) mod TableSize
  2^{th} probe = (h(k) + f(2)) mod TableSize
  . . .
  i^{th} probe = (h(k) + i) mod TableSize

- Searching
  - Keep probing until you find empty spot.
  - Return if match

Linear Probing

$$F(i) = i$$

- Probe sequence:

  0^{th} probe = h(k) mod TableSize

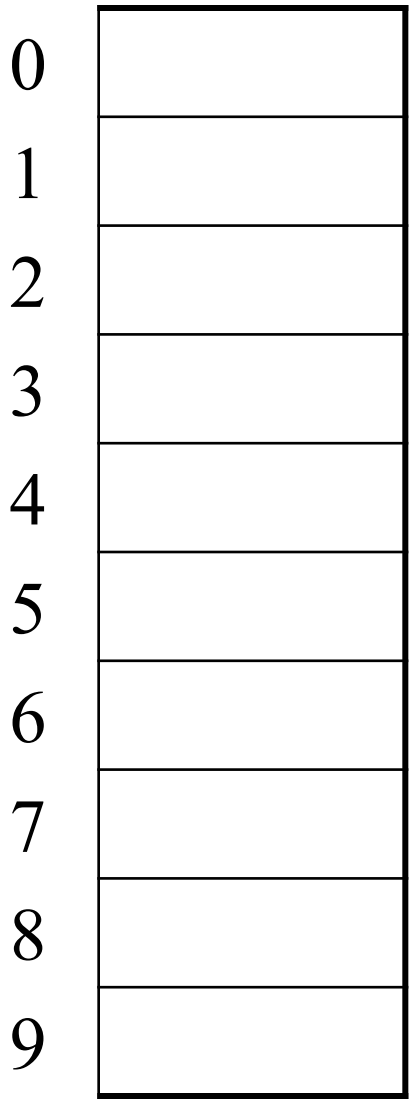
  1^{th} probe = (h(k) + 1) mod TableSize

  2^{th} probe = (h(k) + 2) mod TableSize

  . . .

  i^{th} probe = (h(k) + i) mod TableSize

Probing or Open Addressing



Insert:
38
19
8
109
10

- **Linear Probing**: after checking spot h(k), try spot h(k)+1, if that is full, try h(k)+2, then h(k)+3, etc.

Probing or Open Addressing

0	8
1	109
2	10
3	
4	
5	
6	
7	
8	38
9	19

Insert:
38
19
8
109
10

- **<u>Linear Probing</u>**: after checking spot h(k), try spot h(k)+1, if that is full, try h(k)+2, then h(k)+3, etc.

Quadratic Probing

Less likely to encounter Primary Clustering

$F(i) = i^2$

- Probe sequence:

  0^{th} probe = h(k) mod TableSize

  1^{th} probe = (h(k) + 1) mod TableSize

  2^{th} probe = (h(k) + 4) mod TableSize

  3^{th} probe = (h(k) + 9) mod TableSize

  . . .

  i^{th} probe = (h(k) + i^2) mod TableSize

Double Hashing

$$f(i) = i * g(k)$$

where g is a second hash function

- Probe sequence:

  0^{th} probe = h(k) mod TableSize

  1^{th} probe = (h(k) + g(k)) mod TableSize

  2^{th} probe = (h(k) + 2*g(k)) mod TableSize

  3^{th} probe = (h(k) + 3*g(k)) mod TableSize

  . . .

  i^{th} probe = (h(k) + i*g(k)) mod TableSize