Data Structures and Algorithms

Lecture 19

Aniket Basu Roy

BITS Pilani Goa Campus

2026-03-06 Fri

Agenda

- Data Structures
- Recap: Hashing with Chaining
- Hash Functions
- Open Addressing
- Probing Strategies

Recap: Hash Tables

Use a hash function $h : U \to \{0, 1, \ldots, m-1\}$.

- Element with key k goes in slot h(k).
- Table size $m \ll |U|$ $\Rightarrow$ space $\Theta(m)$.
- Problem: Two distinct keys may map to the same slot, aka collision.

Recap: Hashing with Chaining

- Hash function $h : U \to \{0, \ldots, m-1\}$.
- Collisions resolved by linked lists at each slot.
- Load factor $\alpha = n/m$.
- Under SUHA: $O(1+\alpha)$ expected search.
- Universal hashing: guarantees $O(1+\alpha)$ expected for any input.

Expected Search Times

Under SUHA:

Unsuccessful search: examine entire chain at $T[h(k)]$.

$$\text{Expected cost} = \Theta(1 + \alpha)$$

Successful search: examine on average half the chain beyond target.

$$\text{Expected cost} = \Theta(1 + \alpha)$$

Key Takeaway

If $n = O(m)$ (i.e., $\alpha = O(1)$), all operations take $O(1)$ expected time.

Hash Functions

A good hash function approximates SUHA.

Hash Functions

A good hash function approximates SUHA.
Interpretation: Any key can be viewed as a non-negative integer (e.g., a string treated as a base-128 number).

Hash Functions

A good hash function approximates SUHA.
Interpretation: Any key can be viewed as a non-negative integer (e.g., a string treated as a base-128 number).

Division Method

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

Choose m = prime, not close to a power of 2 or 10.

Multiplication Method

$$h(k) = \lfloor m \cdot (kA \bmod 1) \rfloor, \quad 0 < A < 1$$

Multiplication Method

$$h(k) = \lfloor m \cdot (kA \bmod 1) \rfloor, \quad 0 < A < 1$$

- $m = 2^p$ works fine.
- Knuth: $A \approx (\sqrt{5} - 1)/2 \approx 0.618\ldots$

Multiplication Method

$$h(k) = \lfloor m \cdot (kA \bmod 1) \rfloor, \quad 0 < A < 1$$

- $m = 2^p$ works fine.
- Knuth: $A \approx (\sqrt{5} - 1)/2 \approx 0.618\ldots$

Implementation: Let $s = \lfloor A \cdot 2^w \rfloor$ (w = word size). Compute $k \cdot s$; take the high-order p bits of the lower \$w\$-bit word.

The Problem with Fixed Hash Functions

For any fixed h, an adversary can choose n keys all hashing to one slot $\Rightarrow \Theta(n)$ per operation.

The Problem with Fixed Hash Functions

For any fixed h, an adversary can choose n keys all hashing to one slot $\Rightarrow \Theta(n)$ per operation.

Solution: Universal Hashing

Choose h at random from a universal family $\mathcal{H}$:

$$\forall k_1 \neq k_2 \in U: \quad \Pr_{h \in \mathcal{H}}\big[h(k_1) = h(k_2)\big] \leq \frac{1}{m}$$

The Problem with Fixed Hash Functions

For any fixed h, an adversary can choose n keys all hashing to one slot $\Rightarrow \Theta(n)$ per operation.

Solution: Universal Hashing

Choose h at random from a universal family $\mathcal{H}$:

$$\forall k_1 \neq k_2 \in U: \quad \Pr_{h \in \mathcal{H}}\big[h(k_1) = h(k_2)\big] \leq \frac{1}{m}$$

Consequence: For any n keys, expected collisions per key $\leq \alpha$.
All operations: $O(1+\alpha)$ expected — regardless of input.

A Universal Family

For prime $p \geq |U|$, let

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

with $a \in \{1, \ldots, p-1\}$, $b \in \{0, \ldots, p-1\}$ chosen uniformly at random.

A Universal Family

For prime $p \geq |U|$, let

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

with $a \in \{1, \ldots, p-1\}$, $b \in \{0, \ldots, p-1\}$ chosen uniformly at random.
The family $\mathcal{H}_{pm} = \{h_{ab}\}$ is universal.

A Universal Family

For prime $p \geq |U|$, let

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

with $a \in \{1, \ldots, p-1\}$, $b \in \{0, \ldots, p-1\}$ chosen uniformly at random.
The family $\mathcal{H}_{pm} = \{h_{ab}\}$ is universal.
Idea: For distinct k_1, k_2, the value
$(ak_1 + b) - (ak_2 + b) \bmod p$ is uniformly distributed $\Rightarrow$ the chance of collision mod m is $\leq 1/m$.

Summary

Concept	Key Result
Direct-address table	$\Theta(1)$; needs $O(\|U\|)$ space
Chaining	$O(1+\alpha)$ expected search
Load factor $\alpha = n/m$	Keep $O(1)$ for $O(1)$ expected ops
Division method	$h(k) = k \bmod m$, m prime
Multiplication method	Uses fractional part; $m = 2^p$ ok
Universal hashing	$O(1+\alpha)$ expected, any input

Open Addressing

Store all elements directly in the table — no external storage.

Open Addressing

Store all elements directly in the table — no external storage.
Probe sequence for key k: a permutation of $\{0, \ldots, m-1\}$

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

Open Addressing

Store all elements directly in the table — no external storage.
Probe sequence for key k: a permutation of $\{0, \ldots, m-1\}$

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

Try slots in order; insert into the first empty slot found.

Open Addressing

Store all elements directly in the table — no external storage.
Probe sequence for key k: a permutation of $\{0, \ldots, m-1\}$

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

Try slots in order; insert into the first empty slot found.
Requirement: $\alpha = n/m < 1$.

Open Addressing: Insert

```
HASH-INSERT(T, k)
  i = 0
  repeat
    j = h(k, i)
    if T[j] == NIL:  T[j] = k;  return j
    else i = i + 1
  until i == m
  error "overflow"
```

Open Addressing: Search

```
HASH-SEARCH(T, k)
  i = 0
  repeat
    j = h(k, i)
    if T[j] == k:  return j
    i = i + 1
  until T[j] == NIL or i == m
  return NIL
```

Deletion in Open Addressing

Cannot set $T[j] = \text{NIL}$ — breaks future searches!

Deletion in Open Addressing

Cannot set $T[j] = \text{NIL}$ — breaks future searches!

```
Insert 10 -> slot 2
Insert 22 -> slot 2 occupied -> probe -> slot 3
Delete slot 2 (set to NIL)
Search 22 -> probe slot 2, find NIL -> return NIL
```

Deletion in Open Addressing

Cannot set $T[j] = \text{NIL}$ — breaks future searches!

```
Insert 10 -> slot 2
Insert 22 -> slot 2 occupied -> probe -> slot 3
Delete slot 2 (set to NIL)
Search 22 -> probe slot 2, find NIL -> return NIL
```

Solution: Use special marker DELETED.

- INSERT treats it as empty.
- SEARCH treats it as occupied (continues probing).

Linear Probing

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

Linear Probing

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

Simple and cache-friendly.

Linear Probing

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

Simple and cache-friendly.

Primary Clustering

Long runs of occupied consecutive slots form. A key landing in a run must probe through it — runs grow faster.

```
[_, _, A, B, C, D, _, _, E, _]
New key -> slot 2 -> probes 2,3,4,5,6
```

Quadratic Probing

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

Quadratic Probing

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

Spreads probes more broadly. Choose c_1, c_2, m carefully to visit all slots.

Quadratic Probing

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

Spreads probes more broadly. Choose c_1, c_2, m carefully to visit all slots.

Secondary Clustering

If $h'(k_1) = h'(k_2)$, then k_1 and k_2 always follow the same probe sequence $\Rightarrow$ they keep colliding with each other.

Double Hashing

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

Double Hashing

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

- Probe offset $h_2(k)$ is key-dependent $\Rightarrow$ no secondary clustering.
- Require $\gcd(h_2(k), m) = 1$ to visit all slots.
  - m prime: choose $h_2(k) \in \{1, \ldots, m-1\}$.
  - $m = 2^p$: choose $h_2(k)$ always odd.

Double Hashing

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

- Probe offset $h_2(k)$ is key-dependent $\Rightarrow$ no secondary clustering.
- Require $\gcd(h_2(k), m) = 1$ to visit all slots.
  - m prime: choose $h_2(k) \in \{1, \ldots, m-1\}$.
  - $m = 2^p$: choose $h_2(k)$ always odd.

Example: $h_1(k) = k \bmod m$, $h_2(k) = 1 + (k \bmod (m-1))$

Double Hashing

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

- Probe offset $h_2(k)$ is key-dependent $\Rightarrow$ no secondary clustering.
- Require $\gcd(h_2(k), m) = 1$ to visit all slots.
  - m prime: choose $h_2(k) \in \{1, \ldots, m-1\}$.
  - $m = 2^p$: choose $h_2(k)$ always odd.

Example: $h_1(k) = k \bmod m$, $h_2(k) = 1 + (k \bmod (m-1))$
Best performance in practice. Gives $\Theta(m^2)$ distinct probe sequences.