Data Structures and Algorithms

Lecture 18

Aniket Basu Roy

2026-03-02 Mon

Contents

1 Agenda

1.1 Motivation: The Dictionary Problem

1.2 Direct-Address Tables

1.3 Hash Tables and Chaining

1.4 Analysis of Hashing with Chaining

1.5 Hash Functions

2 The Dictionary Problem

We want a data structure that supports:

Operation	Description
INSERT(S, x)	Insert element x into set S
DELETE(S, x)	Delete element x from set S
SEARCH(S, k)	Return element with key k, or NIL

- Keys come from a **universe** $U = \{0, 1, \ldots, |U|-1\}$.
- The set S has n elements at any time; typically $n \ll |U|$.

3 Direct-Address Tables

3.1 Idea

If $|U|$ is small, allocate an array $T[0 : |U| - 1]$. Slot $T[k]$ holds a pointer to the element with key k (or NIL).

3.2 Operations

```
DIRECT-ADDRESS-SEARCH(T, k)
1.  return T[k]

DIRECT-ADDRESS-INSERT(T, x)
1.  T[x.key] = x

DIRECT-ADDRESS-DELETE(T, x)
1.  T[x.key] = NIL
```

All operations take $\Theta(1)$ time.

3.3 Drawback

If $|U|$ is large (e.g., all 64-bit integers), allocating T is infeasible.

4 Hash Tables

4.1 Key Idea

Use a **hash function** $h : U \to \{0, 1, \ldots, m - 1\}$ to map keys to **slots** in a table of size $m \ll |U|$.

Element with key k goes in slot $h(k)$.

4.2 Collisions

Two keys $k_1 \neq k_2$ may satisfy $h(k_1) = h(k_2)$ — a **collision**.

Resolution strategy: **chaining** (this lecture), open addressing (next lecture).

5 Chaining

Store all elements that hash to the same slot in a linked list.

```
T
0 → NIL
1 → [k=49] → [k=86] → NIL
2 → [k=17] → NIL
3 → NIL
4 → [k=52] → [k=30] → NIL
...
```

The list at $T[j]$ holds all elements x with $h(x.key) = j$.

6 Operations: Chaining

```
CHAINED-HASH-INSERT(T, x)
1.  insert x at the head of list T[h(x.key)]

CHAINED-HASH-SEARCH(T, k)
1.  search for element with key k in list T[h(k)]
2.  return the element, or NIL if not found

CHAINED-HASH-DELETE(T, x)
1.  delete x from list T[h(x.key)]
```

- INSERT: $O(1)$ (insert at head; assume x not already present).
- DELETE: $O(1)$ if the list is doubly linked (given a pointer to x).
- SEARCH: proportional to the list length at $T[h(k)]$.

7 Analysis: Load Factor and Simple Uniform Hashing

7.1 Load Factor

$$\alpha = \frac{n}{m}$$

where n = number of elements, m = number of slots. α is the **average** list length.

7.2 Simple Uniform Hashing Assumption (SUHA)

Each key is equally likely to hash to any of the m slots, independently.

Under SUHA, the expected length of each list is $\alpha = n/m$.

8 Analysis: Expected Search Times

8.1 Unsuccessful Search

A search for key k terminates after examining the entire list at $T[h(k)]$.

Expected cost: $\Theta(1+\alpha)$ ($O(1)$ to compute $h(k)$, then traverse list of expected length α).

8.2 Successful Search

A search for key k terminates at element x with $x.key = k$. On average, half the list beyond x is examined.

8.2.1 Expected cost: $\Theta(1+\alpha)$.

- Define a 0/1 random variable Y_{ijk} which is 1 if $h(x_i) = h(x_j) = y_k$, and 0 otherwise, where $i, j \in 1, \ldots, n$ and $k \in 1, \ldots, m$.
- $\mathbb{E}[Y_{ijk}] = \mathbb{P}[h(x_i) = h(x_j) = y_k] = \mathbb{P}[h(x_i) = y_k] \cdot \mathbb{P}[h(x_j) = y_k] = 1/m^2$
- Define a 0/1 random variable $X_{ij} = Y_{ij1} + Y_{ij2} + \ldots + Y_{ijm}$.
- Observe from the linearity of expectations the following holds, $\mathbb{E}[X_{ij}] = \mathbb{E}[Y_{ij1}] + \ldots + \mathbb{E}[Y_{ijm}] = m/m^2 = 1/m$

$$\begin{aligned}
\mathbb{E}\big[\frac{1}{n}\sum_{i=1}^{n}(1+\sum_{j=i+1}^{n}X_{ij})\big] &= \frac{1}{n}\sum_{i=1}^{n}(1+\sum_{j=i+1}^{n}\mathbb{E}[X_{ij}]) \\
&= 1+\frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{1}{m} \\
&= 1+\frac{1}{mn}\sum_{i=1}^{n}\sum_{j=i+1}^{n}1 \\
&= 1+\frac{1}{mn}\sum_{i=1}^{n}(n-i) \\
&= 1+\frac{1}{mn}\big(\sum_{i=1}^{n}n-\sum_{i=1}^{n}i\big) \\
&= 1+\frac{1}{mn}\big(n^2-\frac{n(n+1)}{2}\big) \\
&= 1+\frac{n-1}{2m} \\
&= 1+\frac{\alpha}{2}-\frac{\alpha}{2n}
\end{aligned}$$

8.3 Conclusion

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

9 Hash Functions

A good hash function satisfies (approximately) the SUHA.

9.1 Interpreting Keys as Integers

- Any key (string, struct, etc.) can be interpreted as a non-negative integer.
- E.g., a string $s = s_0 s_1 \cdots s_{l-1}$ can be treated as a base-128 integer.

10 Division Method

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

- Fast: one modulo operation.
- Choose m to be a prime not close to a power of 2 or 10.
- Avoid $m = 2^p$ (only uses low-order bits of k).

11 Multiplication Method

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

where $0 < A < 1$ is a constant and "$x \bmod 1$" means the fractional part of x.

- Value of m is not critical (often take $m = 2^p$).
- Knuth suggests $A \approx (\sqrt{5} - 1)/2 \approx 0.6180339887\ldots$
- Efficient implementation: let $s = \lfloor A \cdot 2^w \rfloor$ where w is word size. Compute $k \cdot s$, take the high-order p bits of the lower w-bit word.

12 Universal Hashing

12.1 The Problem with Fixed Hash Functions

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

12.2 Universal Hashing

Choose h **randomly** at the start from a **universal family** $\mathcal{H}$.

$\mathcal{H}$ is **universal** if for any two distinct keys $k_1, k_2 \in U$:

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

12.3 Consequence

For any fixed set of n keys, the expected number of collisions for any key k is at most $n/m = \alpha$. Hence all operations take $O(1 + \alpha)$ expected time — regardless of the input.

13 A Universal Family

For a prime $p \geq |U|$, define

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

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

The family $\mathcal{H}_{pm} = \{h_{ab}\}$ is universal.