COMP 480/580 — Probabilistic Algorithms and Data Structure Sep 11, 2025

Lecture 5

Lecturer: Anshumali Shrivastava Scribe By: Liebin Zhang (pz23), Limei Jiang (lj37)

1 Background from previous class

1.1 k-Universal Hashing Families

Let U be a universe of possible keys, and suppose the hash functions map

$$h : U \to [m], \quad [m] = \{0, 1, \dots, m-1\}.$$

A family of hash functions $\mathcal{H} = \{h : U \to [m]\}$ is called a k-**universal hashing family** if for any k distinct inputs $x_1, x_2, \dots, x_k \in U$ and any k outputs $y_1, y_2, \dots, y_k \in [m]$, we have

$$\Pr_{h \in \mathcal{H}} \big[h(x_1) = y_1,\ h(x_2) = y_2,\ \dots,\ h(x_k) = y_k\big] = \frac{1}{m^k}.$$

That is, when h is chosen uniformly at random from $\mathcal{H}$, the values $h(x_1), h(x_2), \dots, h(x_k)$ are *independent and uniformly distributed* over $[m]$.

1.2 Linear probing

Linear probing is a widely used collision–resolution strategy in open–addressing hash tables. When two keys hash to the same slot, instead of storing them in a separate chain, linear probing searches forward in the array to find the next available slot.

Formally, let $h(k)$ be the hash value of a key k for a table of size m. If slot $h(k)$ is occupied, linear probing inspects the sequence

$$h(k),\ (h(k) + 1) \bmod m,\ (h(k) + 2) \bmod m,\ \dots$$

until an empty slot is found. The same sequence is followed during search and deletion operations.

The advantages of linear probing are its simplicity and cache efficiency, since all elements are stored in a contiguous array. However, it suffers from *primary clustering*, where long runs of occupied slots form and grow, increasing the expected probe length for insertions and searches.

1.3 Expected Cost of Linear Probing

Let a hash table of size m store n keys using open addressing with *linear probing*, and let the load factor be

$$\alpha \ = \ \frac{n}{m} \ < \ 1.$$

Under the standard random hashing assumptions (keys hashed independently and uniformly), the expected number of probes (array inspections) is:

Successful search For a key present in the table, the expected number of probes is

$$\mathbb{E}[\text{probes}_{\text{succ}}] \ = \ \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right).$$

Unsuccessful search (and insertion) For a key not present (or for inserting a new key, which stops at the first empty slot), the expected number of probes is

$$\mathbb{E}[\text{probes}_{\text{unsucc}}] \;=\; \frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right).$$

Implications Both costs are $O(1)$ for any fixed $\alpha < 1$, but they grow rapidly as $\alpha \to 1$ due to primary clustering. In practice, tables are resized before the load factor becomes too high (e.g., when $\alpha \geq 0.7$).

2 Mark and recapture

Goal: understand how randomized estimation process works

2.1 Background

The **Mark and Recapture** method is a classical technique in ecology, originally developed to estimate the size of animal populations (e.g., fish in a lake, turtles on an island). The idea is:

1. Capture and mark a subset of individuals.
2. Release them back into the population.
3. Recapture another sample later.
4. Use the fraction of marked individuals in the second sample to estimate the total population size.

This idea is widely used outside of ecology as well, including in **computer science** (e.g., estimating the number of distinct users, files, or IP addresses seen in a system).

2.2 Intuition

Suppose:

$n =$ true population size (unknown), $\quad K_1 =$ number marked in first capture, $\quad K_2 =$ number in second captur

Then, intuitively:

$$\frac{M}{K_2} \approx \frac{K_1}{n}.$$

Rearranging:

$$n \approx \frac{K_1 K_2}{M}.$$

This is called the **Lincoln–Petersen estimator**.

2.3 Expectation of Recaptures

Each item in the second sample has probability $\frac{K_1}{n}$ of being marked. Define indicator random variables:

$$X_j = \begin{cases} 1 & \text{if the } j\text{-th individual in second capture is marked,} \\ 0 & \text{otherwise.} \end{cases}$$

Then:

$$\Pr(X_j = 1) = \frac{K_1}{n}, \quad M = \sum_{j=1}^{K_2} X_j.$$

By linearity of expectation:

$$\mathbb{E}(M) = \sum_{j=1}^{K_2} \mathbb{E}(X_j) = K_2 \cdot \frac{K_1}{n}.$$

2.4 Distribution of M

- Without replacement (more realistic):

$$M \sim \text{Hypergeometric}(n, K_1, K_2),$$

  with probability mass function

$$\Pr(M = m) = \frac{\binom{K_1}{m}\binom{n-K_1}{K_2-m}}{\binom{n}{K_2}}.$$

- With replacement (approximation when n is large):

$$M \sim \text{Binomial}\left(K_2, \frac{K_1}{n}\right).$$

As $n \to \infty$, the difference between the two distributions disappears.

2.5 Variance

- Hypergeometric variance:

$$\text{Var}(M) = K_2 \cdot \frac{K_1}{n} \cdot \left(1 - \frac{K_1}{n}\right) \cdot \frac{n - K_2}{n - 1}.$$

- Binomial approximation:

$$\text{Var}(M) \approx K_2 \cdot \frac{K_1}{n} \cdot \left(1 - \frac{K_1}{n}\right).$$

2.6 Estimation Formula

From the expectation:

$$\mathbb{E}(M) = \frac{K_1 K_2}{n}.$$

Thus:

$$n \approx \frac{K_1 K_2}{M}.$$

2.7 Example 2

1. **First capture (Mark):**
   - On Day 1, sample a subset of users randomly (say, 10,000 visitors).
   - Mark them by storing their IDs in a hash table.
2. **Second capture (Recapture):**
   - On Day 15, take another random sample of 10,000 users.
   - Count how many of these are already present in the stored hash table (say, 2,000).
3. **Estimation:**
   - By analogy, the population can be estimated as:

     $$n \approx \frac{K_1 K_2}{M}$$

     where $K_1 = 10{,}000$, $K_2 = 10{,}000$, and $M = 2{,}000$.
   - So:

     $$n \approx \frac{10{,}000 \times 10{,}000}{2{,}000} = 50{,}000.$$

   - Thus the estimated population size is about 50,000.

This is exactly the same math as counting turtles. In computer science applications, instead of physical tags, we "mark" users with a data structure (e.g., a hash table).

2.8 Applications Beyond Ecology

- **Computer networks:** estimate the number of unique users or flows.
- **Databases:** approximate distinct values in massive tables.
- **Web traffic analysis:** estimate daily/weekly active users without tracking everyone.
- **Epidemiology:** estimate disease spread using test-retest methods.

In CS, instead of physical tags, we use hash tables, bitmaps, or Bloom filters to "mark" elements efficiently.

Key takeaway: Mark and Recapture leverages random sampling and probability theory to estimate large unknown populations using only a few samples. Its accuracy depends on sample sizes, randomness of capture, and independence of samples.