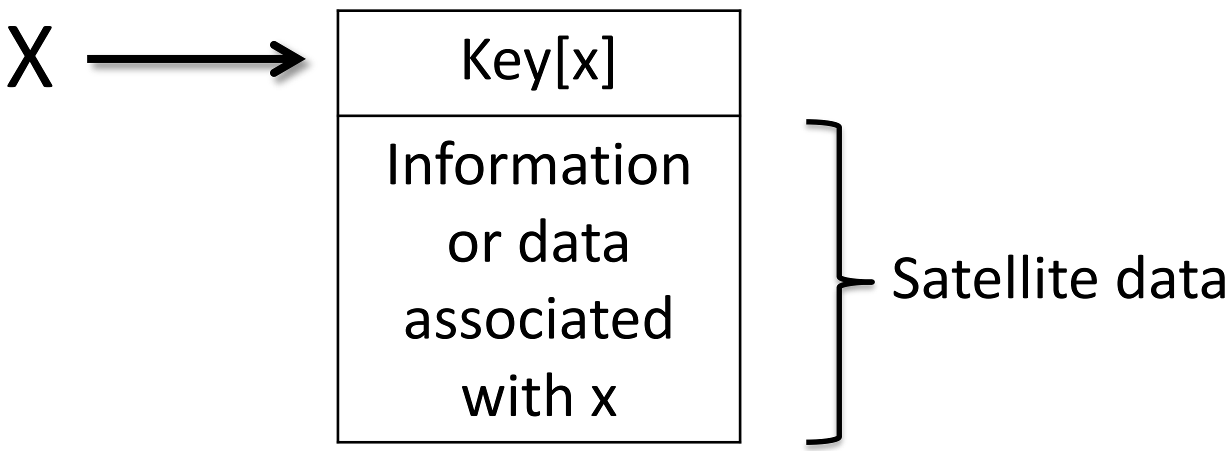
COMP251: Hashing

Jérôme Waldispühl & Giulia Alberini
School of Computer Science
McGill University

Based on (Cormen *et al.*, 2002)

Problem Definition

Table S with *n* records *x*:



We want a data structure to store and retrieve these data.

Operations:

- $insert(S, x) : S \leftarrow S \cup \{x\}$
- $delete(S, x) : S \leftarrow S \setminus \{x\}$

Dynamic set

- $search(S, k)$

Direct Address Table

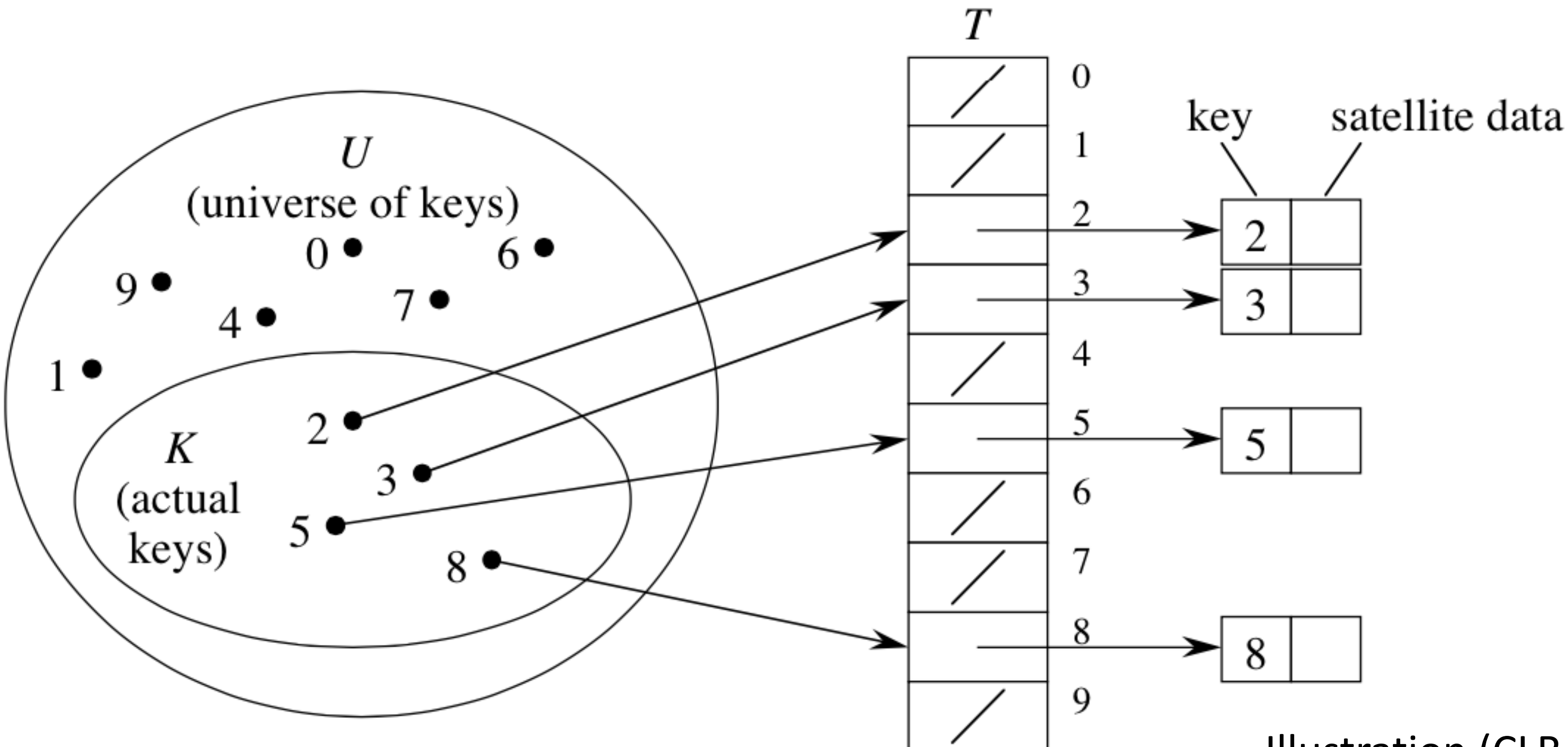


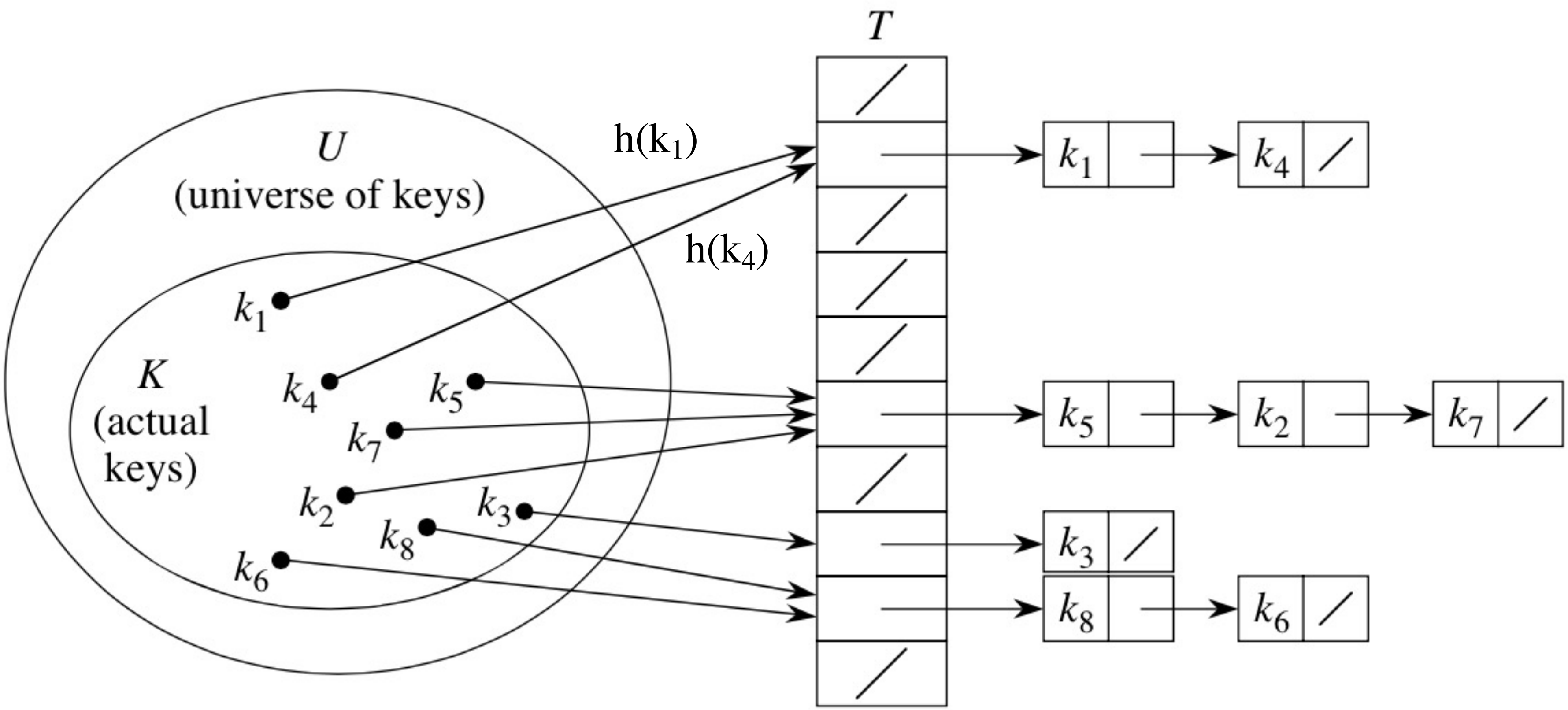
Illustration (CLR, 2005)

- Each slot, or position, corresponds to a key in U .
- If there is an element x with key k, then T[k] contains a pointer to x.
- If T [k] is empty, represented by NIL.

All operations in O(1), but if n (#keys) < m (#slots), lot of wasted space.

Hash Tables

- Reduce storage to O(n) keys.
- Resolve conflicts by chaining.
- Search time in $O(1)$ time in average, but not the *worst case*.



Hash function: $h: U \to \{0,1,\ldots,m-1\}$

Analysis of Hashing with Chaining

Insertion: O(1) time (Insert at the beginning of the list).

Deletion: Search time + O(1) if we use a double linked list.

Search:

- **Worst case:** Worst search time to is O(n).

  Search time = time to compute hash function +

  time to search the list.

  Assuming the time to compute the hash function is O(1).

  Worst time happens when all keys go the same slot (list of size n), and we need to scan the full list => O(n).

- **Average case:** It depends how keys are distributed among slots.

Average case Analysis

Assume a **simple uniform hashing**: n keys are distributed uniformly at random among m slots.

Let n be the number of keys, and m the number of slots.

Average number of element per linked list?

Load factor: $\alpha = \frac{n}{m}$

Theorem:

The expected time of a search is $\Theta(1 + \alpha)$.

Note: $\Theta(1)$ if α is a constant, but $\Theta(n)$ if α is $\Theta(n)$.

Note: The average case analysis is an estimation of the computer resources averaged over a distribution of all possible inputs.

Background

Expectation & Indicators

Expectation

- Average or mean
- The expected value of a discrete random variable X is $E[X] = \sum_x x \Pr\{X=x\}$
- Linearity of Expectation
  - $E[X+Y] = E[X]+E[Y]$, for all X, Y
  - $E[aX+Y] = a\,E[X] + E[Y]$, for constant a and all X, Y
- For mutually independent random variables $X_1, \ldots, X_n$
  - $E[X_1 X_2 \ldots X_n] = E[X_1] \cdot E[X_2] \cdot \ldots \cdot E[X_n]$

Expectation – Example

- Let X be the RV denoting the value obtained when a fair die is thrown. What will be the mean of X, when the die is thrown n times.
  - Let X_1, X_2, …, X_n denote the values obtained during the n throws.
  - The mean of the values is $(X_1+X_2+\ldots+X_n)/n$.
  - Since the probability of getting values 1 to 6 is (1/6) in average, we can expect each of the 6 values to show up $(1/6)n$ times.
  - So, the numerator in the expression for mean can be written as $(1/6)n\cdot 1+(1/6)n\cdot 2+\ldots+(1/6)n\cdot 6$
  - The mean, hence, reduces to $(1/6)\cdot 1+(1/6)\cdot 2+\ldots(1/6)\cdot 6$, which is what we get if we apply the definition of expectation.

Property of Expectation

When X takes values in set of natural numbers:

$$E[X] = \sum_{i=0}^{\infty} i \cdot P(X = i)$$

$$= \sum_{i=0}^{\infty} i \cdot (P(X \geq i) - P(X \geq i + 1))$$

$$= \sum_{i=0}^{\infty} P(X \geq i)$$

Pr(X≥i) is added in i times and subtracted out i-1 times

Indicator Random Variables

- A simple yet powerful technique for computing the expected value of a random variable.
- Convenient method for converting between probabilities and expectations.
- Helpful in situations in which there may be dependence.
- Takes only 2 values, 1 and 0.
- Indicator Random Variable for an event A of a sample space is defined as:

$$I\{A\} = \begin{cases} 1 & \text{if } A \text{ occurs,} \\ 0 & \text{if } A \text{ does not occur.} \end{cases}$$

Indicator Random Variable

Lemma 5.1

Given a sample space S and an event A in the sample space S, let $X_A = \mathrm{I}\{A\}$. Then $\mathrm{E}[X_A] = \Pr\{A\}$.

Proof:

Let $\bar{A} = S - A$ (Complement of A)

Then,

$$\begin{aligned}\mathrm{E}[X_A] &= \mathrm{E}[\mathrm{I}\{A\}] \\ &= 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{\bar{A}\} \\ &= \Pr\{A\}\end{aligned}$$

Indicator RV – Example

Problem: Determine the expected number of heads in n coin flips.

Method 1 (without indicator random variables)

Let X be the random variable for the number of heads in n flips.

Then, $E[X] = \sum_{k=0..n} k \cdot \Pr\{X=k\}$

We can solve this with a lot of math.

Indicator RV – Example

Method 2 (with Indicator Random Variables)

- Define n indicator random variables, X_i, $1 \leq i \leq n$.
- Let X_i be the indicator random variable for the event that the i^{th} flip results in a Head.

  $\Rightarrow X_i = \mathrm{I}\{\text{the } i^{th} \text{ flip results in } H\}$
- Then $X = X_1 + X_2 + \ldots + X_n = \sum_{i=1..n} X_i$.
- By Lemma 5.1, $E[X_i] = \Pr\{H\} = ½$, $1 \leq i \leq n$.
- Expected number of heads is $E[X] = E[\sum_{i=1..n} X_i]$.
- By linearity of expectation, $E[\sum_{i=1..n} X_i] = \sum_{i=1..n} E[X_i]$.
- $E[X] = \sum_{i=1..n} E[X_i] = \sum_{i=1..n} ½ = n/2$.

Average case Analysis

Theorem:

The expected time of a search is $\Theta(1 + \alpha)$.

Proof?

Distinguish two cases:

- search is unsuccessful
- search is successful

Unsuccessful search

- Assume that we can compute the hash function in $O(1)$ time.
- An unsuccessful search requires to scan all the keys in the list.

Average search time = O(1 + average length of lists)

Let n_i be the length of the list attached to slot i.

Average value of n_i? $E(n_i) = \alpha = \dfrac{n}{m}$ (Load factor)

$$\Rightarrow O(\ 1\ ) + O(\ \alpha\ ) = O(\ 1 + \alpha\ )$$

Successful search

- Assume the position of the searched key x is equally likely to be any of the elements stored in the list.
- New keys inserted at the head of the list ⇒ Keys scanned *after* finding *x* have been inserted in the hash table before *x*.
- We will use an indicator to count the number of collisions:

$$X_{ij} = I\left\{h(k_i) = h(k_j)\right\};\ E(X_{ij}) = \frac{1}{m} \quad \text{(probability of a collision)}$$

The analysis of a successful search is more complicated because the search may stop before scanning the full list.

We are interested in collisions because it represents the number of keys that are stored in the same list.

Successful search

The keys in front of x in the list have been inserted after x.

We use the indicator to count the number of keys in front of x

$$number\ of\ keys\ inserted\ in\ the\ slot\ after\ x = 1 + \sum_{j=i+1}^{n} X_{ij}$$

$$expected\ number\ of\ scanned\ keys = E\left[\frac{1}{n}\sum_{i=1}^{n}\left(1 + \sum_{j=i+1}^{n} X_{ij}\right)\right]$$

$$E\left[\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n} X_{ij}\right)\right] = \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n} E\left[X_{ij}\right]\right)$$

All keys have the same probability to be searched for.

$$= \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\frac{1}{m}\right)$$

$$= 1 + \frac{\alpha}{2} - \frac{\alpha}{2n}$$

$$\boxed{\text{Search time: } \Theta\left(1+1+\frac{\alpha}{2}-\frac{\alpha}{2n}\right) = \Theta(1+\alpha)}$$

Supplementary material

$$
\begin{aligned}
\mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}X_{ij}\right)\right]
&= \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\mathrm{E}\left[X_{ij}\right]\right)\\
&= \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\frac{1}{m}\right)\\
&= 1+\frac{1}{nm}\sum_{i=1}^{n}(n-i)\\
&= 1+\frac{1}{nm}\left(\sum_{i=1}^{n}n-\sum_{i=1}^{n}i\right)\\
&= 1+\frac{1}{nm}\left(n^2-\frac{n(n+1)}{2}\right)\\
&= 1+\frac{n-1}{2m}\\
&= 1+\frac{\alpha}{2}-\frac{\alpha}{2n}\,.
\end{aligned}
$$

Designing a hash function

Properties:

1. Uniform distribution of keys into slots
2. Regularity in key disturb should not affect uniformity.

List of functions:

- Division method
- Multiplication methods
- Open addressing:
  - Linear probing
  - Quadratic probing
  - Double hashing

Binary Numbers (reminder)

Each integer x accepts a unique decomposition $x = \sum_i a_i \cdot 2^i$
where $0 \le a_i < 2$

Example: $x = 11 = 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3$

The binary number representation of an integer x is its (reversed) sequence of a's.

Example: $x = 11 \rightarrow$

2^3	2^2	2^1	2^0
1	0	1	1

$\rightarrow 1011$

Binary number operations:

101101 >> 1 = 10110 (right shift) : quotient of division by 2^k

101101 << 2 = 10110100 (left shift) : multiplication by 2^k

101101 mod 2^2 = 01 (modulo 2^k) : remainder of division by 2^k

Division Method

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

***d* must be chosen carefully!**

Example 1: *d* = 2 and all keys are even?

Odd slots are never used…

Example 2: *d* = 2^r

k = 10001011010110101011

r = 2 -> 11

r = 3 -> 011

r = 4 -> 1011

keeps only r last bits…

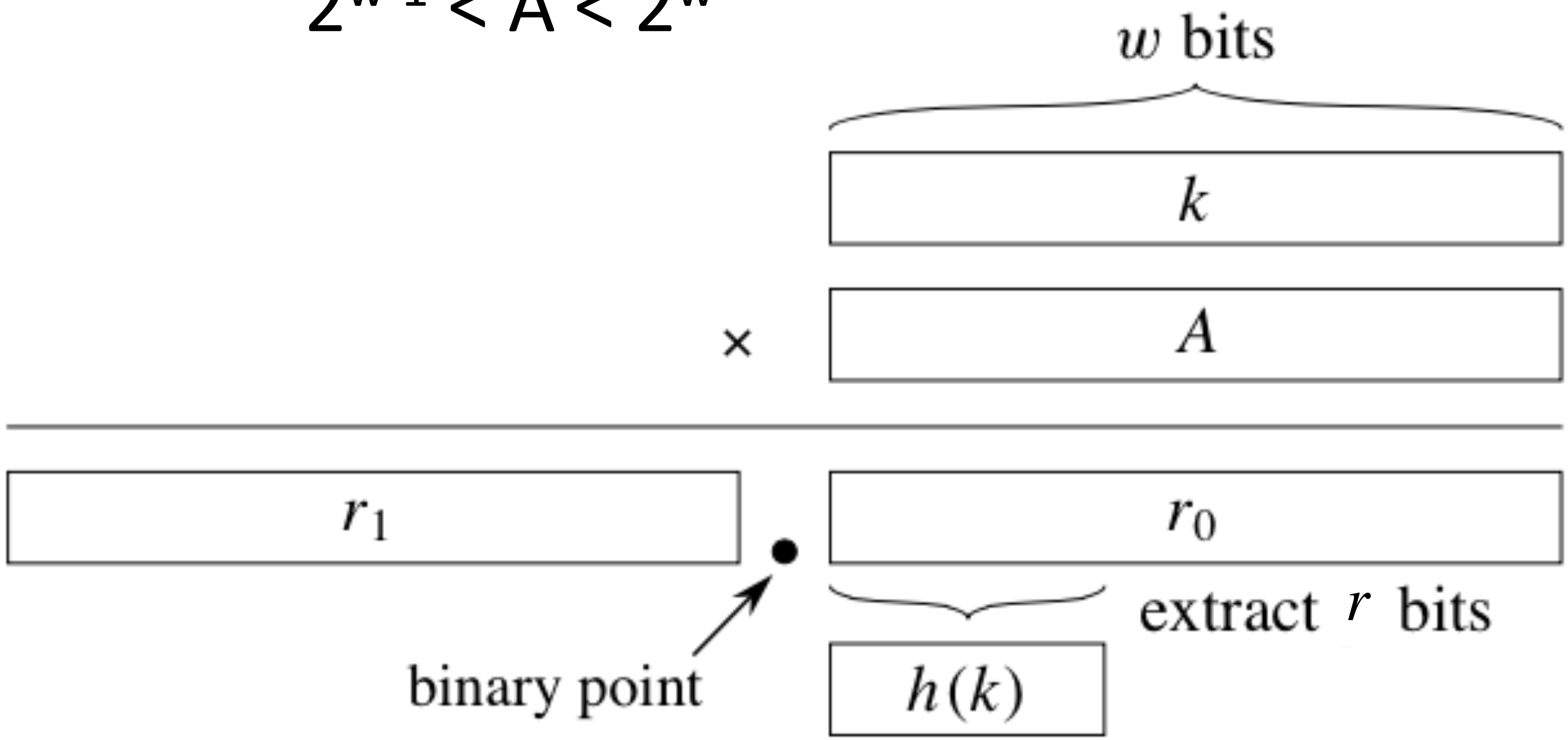
Good heuristic: Choose *d* prime not too close from a power of 2.

Note: Easy to implement, but division is slow…

Multiplication method

$$h(k) = \left(A \cdot k \bmod 2^w\right) >> (w - r)$$

$2^{w-1} < A < 2^w$



Slower to compute but less sensitive to the choice of variables.

Open addressing

No storage for multiple keys on single slot (i.e., no chaining).

Idea: Probe the table.

- Insert if the slot if empty,
- Try another hash function otherwise.

h: U x { 0, … , m-1 } -> { 1, … , m }

Universe of keys   probe number   slot

Constraints:

- $n \leq m$ (i.e. more slots than keys to store)
- Deletion is difficult

Challenge: How to build the hash function?

Open addressing

Illustration: Where to store key 282?

Full!

h(282,0)=3

h(282,1)=1

h(282,2)=5

index	key
1	355
2	
3	567
4	233
5	282
6	799
7	

The probe number is used to determine the order you will use to try the slots.

Important: Search must use the same probe sequence.

Linear & Quadratic probing

Linear probing:

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

Note: tendency to create clusters.

Quadratic probing:

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

Remarks:

- We must ensure that we have a full permutation of ⟨ 0, … , m-1 ⟩.
- **Secondary clustering**: 2 distinct keys have the same h′ value, if they have the same probe sequence.

Double hashing

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

Must have $h_2(k)$ be “relatively” prime to m to guarantee that the probe sequence is a full permutation of $\langle 0, 1, \ldots, m - 1\rangle$.

Examples:

- m power of 2 and h_2 returns odd numbers
- m prime number and $1 < h_2(k) < m$

Analysis of open-addressing

We assume **uniform hashing**: Each key equally likely to have anyone of the m's permutations as its probe sequence, independently of other keys.

Theorem 1: The expected number of probes in an unsuccessful search is at most $\frac{1}{1-\alpha}$.

Theorem 2: The expected number of probes in a successful search is at most $\frac{1}{\alpha} \cdot \log\left(\frac{1}{1-\alpha}\right)$

Reminder: $\alpha = \frac{n}{m}$ is the load factor

Proof for unsuccessful searches

Initial state: *n* keys are already stored in *m* slots.

Probability 1st slot is occupied: n/m.

Probability 2nd slot is occupied knowing 1st is too: (n-1)/(m-1).

Probability 3rd slot is occupied knowing 1st & 2nd are too : (n-2)/(m-2).

Let *X* be the number of unsuccessful probes.

$$\Pr\{X \geq i\} = \underbrace{\frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdots \frac{n-i+2}{m-i+2}}_{\text{i-1 factors}}$$

$n < m \Rightarrow (n-j)/(m-j) \leq n/m$, for $0 \leq j \leq n$

$\Pr\{X \geq i\} \leq (n/m)^{i-1} = \alpha^{i-1}$

$$E[X] = \sum_{i=1}^{\infty} \Pr\{X \geq i\} \leq \sum_{i=1}^{\infty} \alpha^{i-1} = \sum_{i=0}^{\infty} \alpha^{i} = \frac{1}{1-\alpha}$$

We use the same upper bound for all terms in the product.

Consequences

Corollary

The expected number of probes to insert is at most $1/(1 - \alpha)$.

Interpretation:

- If α is constant, an unsuccessful search takes $O(1)$ time.

Yet…

- If $\alpha = 0.5$, then an unsuccessful search takes an average of $1/(1 - 0.5) = 2$ probes.
- If $\alpha = 0.9$, takes an average of $1/(1 - 0.9) = 10$ probes.

Proof of Theorem on successful searches: See [CLRS, 2009].

Universal Hashing

- **Set-up: We solve collision by chaining.**
- A malicious adversary who has learned the hash function chooses keys that all map to the same slot, giving worst-case behavior.
- Defeat the adversary using **Universal Hashing**
  - Use a different random hash function each time.
  - Ensure that the random hash function is independent of the keys that are going to be stored.
  - Ensure that the random hash function is "good" by carefully designing a class of functions to choose from:
    - Design a universal class of functions.

Universal Set of Hash Functions

A finite collection of hash functions $\mathbf{H}$ that maps a universe $\mathbf{U}$ of keys into the range $\{0, 1, \ldots, m-1\}$ is **universal** if,

for each pair of distinct keys $x, y \in \mathbf{U}$, the number of hash functions $h \in \mathbf{H}$ for which $h(x)=h(y)$ is $\leq |\mathbf{H}|/m$.

In other words, for a hash function h chosen randomly from $\mathbf{H}$, the chance of a collision is $\leq 1/m$.

Universal hash functions give good hashing behavior.

Cost of Universal Hashing

Theorem:

Using chaining and universal hashing on key k:

- If k is not in the table T, the expected length of the list that k hashes to is $\leq \alpha$.
- If k is in the table T, the expected length of the list that k hashes to is $\leq 1+\alpha$.

Proof:

X_k = # of keys (≠k) that hash to the same slot as k.

$C_{kl} = I\{h(k)=h(l)\}$; $E[C_{kl}] = \Pr\{h(k)=h(l)\} \leq 1/m$.

$$X_k = \sum_{l \in T \setminus \{k\}} C_{kl}, \text{ and } E[X_k] = E\left[\sum_{l \in T \setminus \{k\}} C_{kl}\right] = \sum_{l \in T \setminus \{k\}} E[C_{kl}] \leq \sum_{l \in T \wedge l \neq k} \frac{1}{m}$$

$$\text{If } k \notin T, E[X_k] \leq n/m = \alpha.$$

$$\text{If } k \in T,\ E[X_k] + 1 \leq (n-1)/m + 1 = 1 + \alpha - 1/m < 1 + \alpha.$$

Example of Universal Hashing

- The size of the table *m* is a prime,
- We write a key *x* in bytes s.t. $x = <x_0, \ldots, x_r>$,
- $a = <a_0, \ldots, a_r>$ denotes a sequence of r+1 elements randomly chosen from {0, 1, … , m – 1}.

The class H defined by:

$$H = \bigcup_a \{h_a\} \text{ with } h_a(x) = \sum_{i=0 \text{ to } r} a_i x_i \bmod m$$

is an universal function.

Proof (universal hashing function)

Let $X = \langle x_0, x_1, \ldots, x_r \rangle$ and $Y = \langle y_0, y_1, \ldots, y_r \rangle$ be 2 distinct keys.

They differ at (at least) one position. WLOG let 0 be this position.

For how many h do X and Y collide?

$$\sum_{i=0}^{r} a_i x_i \equiv \sum_{i=0}^{r} a_i y_i \pmod{m}$$

$$\sum_{i=0}^{r} a_i (x_i - y_i) \equiv 0 \pmod{m}$$

$$a_0 (x_0 - y_0) \equiv -\sum_{i=1}^{r} a_i (x_i - y_i) \pmod{m}$$

$$a_0 \equiv \left(-\sum_{i=1}^{r} a_i (x_i - y_i) \right) \cdot (x_0 - y_0)^{-1} \pmod{m}$$

Conclusion:

For any choice of $< a_1, a_2, \ldots, a_r >$ there is **only one choice** of a_0 such that X and Y collide.

#{h that collide} = $m \times m \times \ldots \times m \times 1$

$= m^r = |H|/m$