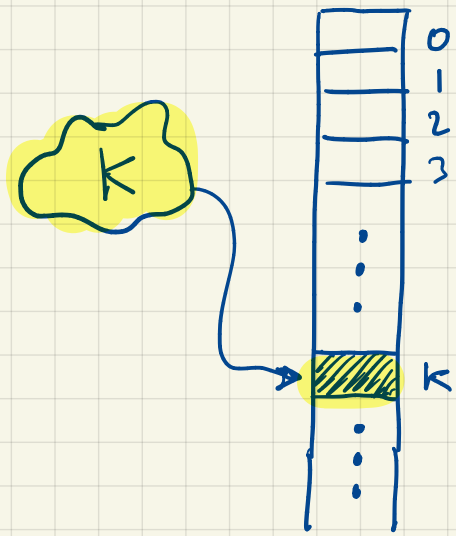
Hash tables

Motivation: Need to insert & search keys in constant time

Direct addressing: Store key k in $A[k]$

(Assuming all keys are integers, but that' ok)



Problem: Range of keys very large even if actual number of keys n is much smaller! (similar problem to counting sort)

Idea: Use a table with only m entries $0, 1, 2, \ldots, m-1$.

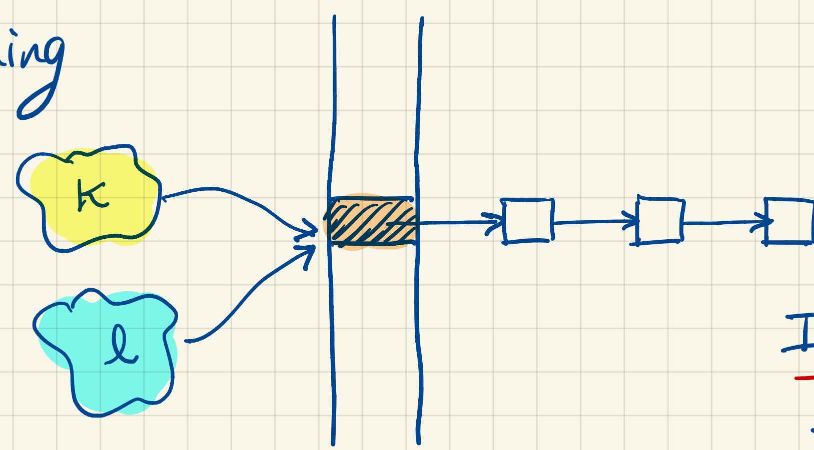
- Map key k to $h(k) \in \{0, 1, 2, \ldots, m-1\}$
- h is a hash function.

Problem: Keys can hash into the same slot

(pigeon hole principle, not too many slots)

Typical Solution: (but other solutions also exist)

use chaining



Each slot has a linked list of keys that hash to it.

Insert: Always at head of list $\Rightarrow \Theta(1)$ time.

What about search?

Analysis of search time.

Assume Simple Uniform Hashing

- Every key hashes into any slot with equal probability $\frac{1}{m}$
- keys hash independently!

This is a strong assumption:

- Hard to guarantee, but
- Several common techniques work well in practice
- Can be relaxed.

Let
$$X_{ij} = \begin{cases} 1 & i^{th} \text{ key hashes to } j^{th} \text{ slot} \\ 0 & \text{otherwise} \end{cases}$$

What does Simple Uniform Hashing tell us?

- $P(X_{ij}=1) = \frac{1}{m}$ (any slot for i^{th} key is equally likely)
- Moreover, if we know $h(k) = j$

then
$$P(X_{ij}=1) = \begin{cases} \text{still } \frac{1}{m} & \text{key}(i) \neq k \text{ (independence)} \\ 1 & \text{key}(i) = k \end{cases}$$

Let n_j = length of list j = $\sum_{i=1}^{n} X_{ij}$

unsuccessful seach: If $h(k) = j$

$$\underbrace{O(1}_{\text{compute } h} + \underbrace{E[n_j]}_{\text{go through entire list}})$$

$$E[n_j] = E\left[\sum_{i=1}^{n} x_{ij}\right] = \sum_{i=1}^{n} E[x_{ij}] = \sum_{i=1}^{n} \frac{1}{m} = \frac{n}{m}$$

(all keys are $\neq K$)

So an unsuccessful search costs $O(1+\alpha)$

where $\alpha = \frac{n}{m}$ [loading factor]

if $n = O(m)$, then this is $O(1)$.

Successful search: If $h(k) = j$

now $E[n_j] = E\left[\sum_{i=1}^{n} X_{ij}\right] = \sum_{i=1}^{n} E[X_{ij}] = 1 + \sum_{key(i) \neq k} \frac{1}{m}$

$= 1 + \frac{n-1}{m} < 1 + \alpha$ (one key is k)

So a successful search take $O(1+1+\alpha) = O(1+\alpha)$ time

Note: The successful search does not need to go through the entire list, but only until it locates k. The book assumes that every element is equally likely to be the one searched for and finds $1 + \frac{n-1}{2m}$ instead of $1 + \frac{n-1}{m}$. which can be heuristically explained as going through half of the other keys in the list before finding k.

Relaxing the Simple Uniform Hashing condition.

weaker condition: (called Universal Hashing)

$$\forall k, l: \quad P(h(k) = h(l)) \leq \frac{1}{m}$$

To redo the analysis:

$P(X_{ij} = 1) = $? (don't know without specific context)

and knowing that $h(k) = j$:

$$P(X_{ij} = 1) = \begin{cases} P(h(k) = h(\text{key}(i))) \leq \frac{1}{m} & \text{key}(i) \neq k \\ 1 & \text{key}(i) = k \end{cases}$$

So same bounds can be derived.

[we will see a method to guarantee this condition]

Practical Hash functions

Division method: $h(k) = k \bmod m$ [remainder in div. by m]

Deficiency: If m has a divisor d, then keys congruent modulo d utilize only $\frac{d}{m}$ slots.

So choose m prime?

Ex: $m = 21$, $d = 7$

$21 \equiv 0$
$28 \equiv 7$
$35 \equiv 14$
$42 \equiv 0$
$\vdots$

Another: If strings are numbers in base 2^p, then if $m = 2^p - 1$, any permutation of the characters result in the same hash

$p = 7$

e.g: "saad" and $m = 127$

Ascii: $\underset{s}{115} \times 128^3 + \underset{a}{97} \times 128^2 + \underset{a}{97} \times 128 + \underset{d}{100} \pmod{127}$

$= 28$

Typical solution:

Choose m prime not close to a power of 2

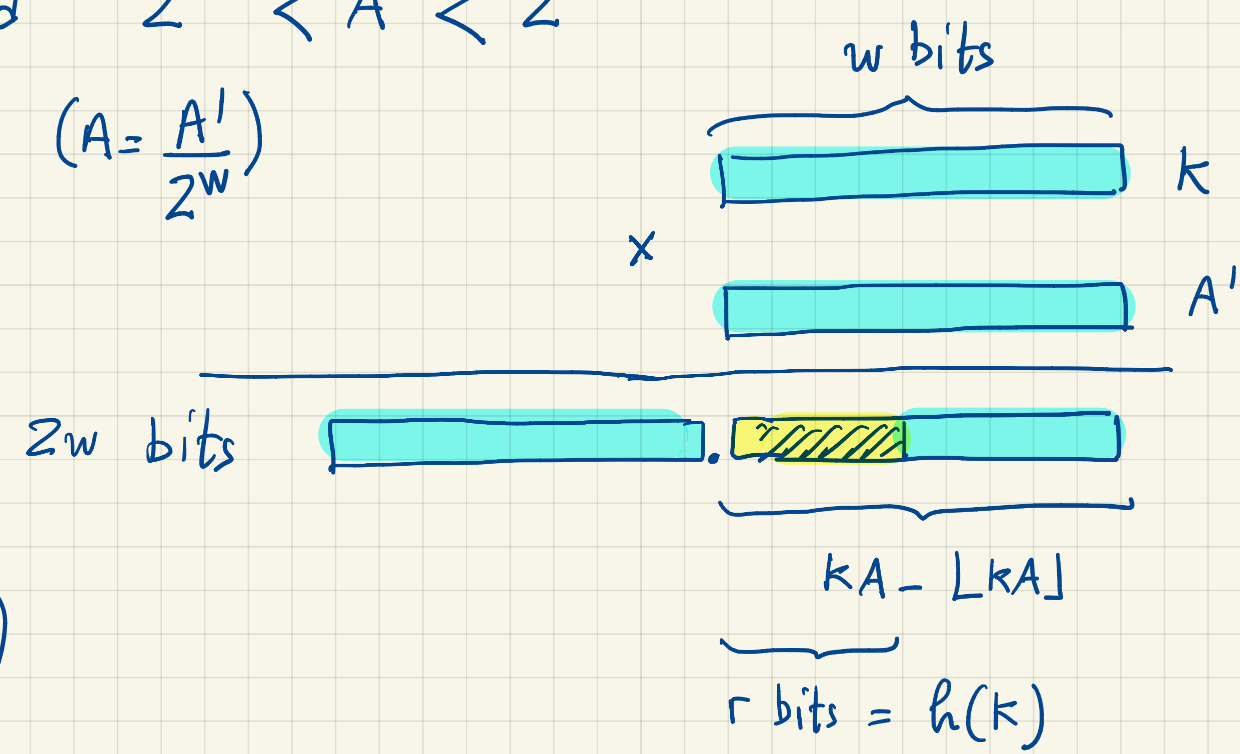
Multiplication method:

$$h(k) = \lfloor m\,(kA - \lfloor kA \rfloor) \rfloor \qquad 0 < A < 1$$

Ex: $A = \dfrac{\sqrt{5}-1}{2} = 0.618$ (golden ratio)

Implementation using w-bit word computer

- let $m = 2^r$ and $2^{w-1} < A' < 2^w$
- Consider $\dfrac{kA'}{2^w}$ $\left(A = \dfrac{A'}{2^w}\right)$



See an example in book end of Sec 11.3.2

Universal Hashing

Consider $\mathcal{H}$ a finite set of hash functions.

It's called universal iff:

$$\forall k, l . \left| \{ h \in \mathcal{H} : h(k) = h(l) \} \right| \leq \frac{|\mathcal{H}|}{m}$$

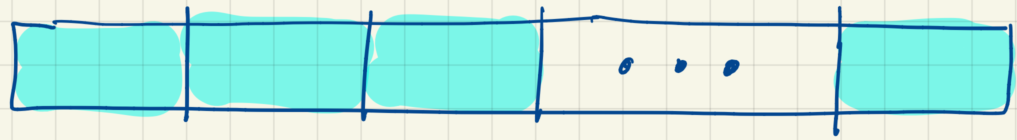
We pick h uniformly at random from $\mathcal{H}$.

How to construct $\mathcal{H}$? Many methods exist.

We will look at one that is easy to analyze.

(the book presents a different one)

Assume key has r parts (treated as integers)



$$k = \langle k_0, k_1, \ldots, k_{r-1} \rangle \qquad 0 \leq k_i < m$$

and m is prime.

Pick $a = \langle a_0, a_1, \ldots, a_{r-1} \rangle$ where each a_i is chosen uniformly at random from $\{0, 1, \ldots, m-1\}$

Then let:

$$h_a(k) = \sum_{i=0}^{r-1} a_i k_i \pmod{m} \qquad |\mathcal{H}| = m^r$$

Given $x \neq y$:

$$h(x) = h(y) \Rightarrow \sum a_i x_i \equiv \sum a_i y_i \pmod m$$

Assume $x_0 > y_0$, then

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

Number theory: m prime $\Rightarrow$ any integer $0 < z < m$ has a multiplicative inverse $z z^{-1} \equiv 1 \pmod m$

So we can solve for a_0. [multiply both sides by $(x_0 - y_0)^{-1}$]

$$\sum_{i=0}^{r-1} a_i x_i \equiv \sum_{i=0}^{r-1} a_i y_i \pmod{m}$$

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

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

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

Example: Multiplicative inverses when $m = 7$

z	1	2	3	4	5	6
z^{-1}	1	4	5	2	3	6

$zz^{-1} \equiv 1 \pmod 7$

For every $\langle a_1, a_2, \ldots, a_{r-1} \rangle$ there is only one a_0 that makes $h(x) = h(y)$. So there are m^{r-1} functions out of m^r functions that make $h(x) = h(y)$.

Therefore

$$\forall x, y. \quad \left| \{ h \in \mathcal{H} : h(x) = h(y) \} \right| = m^{r-1} = \frac{m^r}{m} = \frac{|\mathcal{H}|}{m}$$