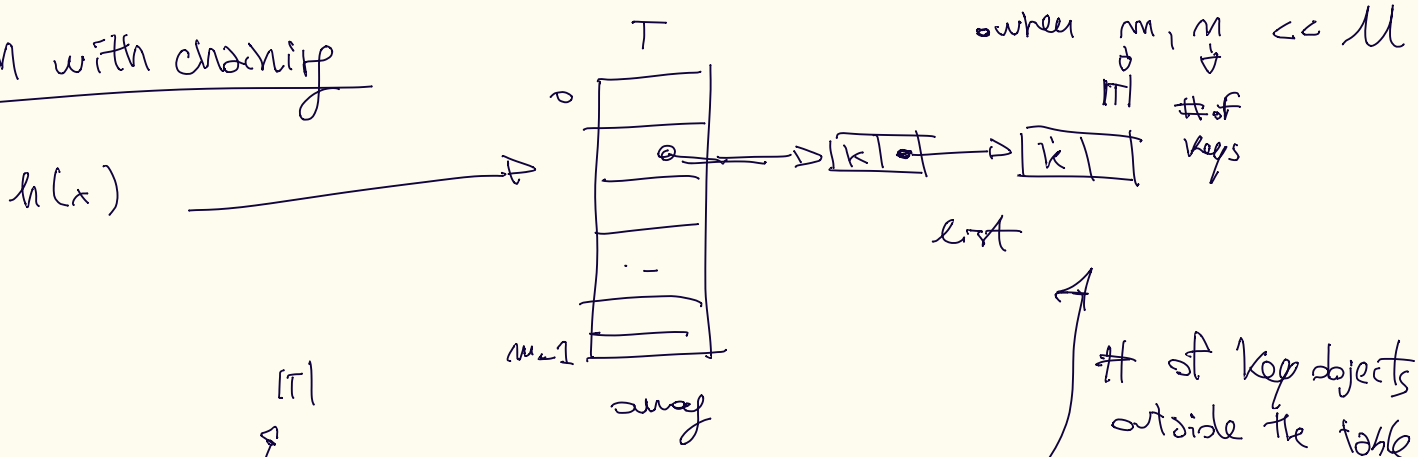
# Hash with chaining




- Total space: $m + n$ + satellite data

$|T|$ → pointer → $m$ = $m \log_2 n$ bits ($m \log_2 64$ bits) = $m \cdot 6$ bits

$n$ → pointer & keys → $n = n \cdot [L + 2 \cdot 64]$ (2 pointers)

# of key objects outside the table

keys table pointer

space: $n \lg U + n \lg n \in n \lg n$ Bits

- INSERTION = $O(1)$ w.c.
- SEARCH = AVG length of the list $O(1+\alpha)$ =>
- DELETION = // => depends on search $O(1+\alpha)$

hash $f(x)$ that distributes the keys in table entries uniformly distributed at random

- If simple uniform hash (UNREALISTIC): $\forall$ key $k$, $P(h(k) = i) = \frac{1}{m}$ (some position $i$ in T (slot))
  => BUT computationally UNFEASIBLE TO MAKE

**Theorem 8.1** *Under the hypothesis of simple uniform hashing, there exists a hash table with chaining, of size $m$, in which the operation* `Search`$(k)$ *over a dictionary of $n$ objects takes $\Theta(1 + n/m)$ expected time. The value $\alpha = n/m$ is often called the* load factor *of the hash table.*

proof:

$$\mathbb{E}[\text{search}] = 1 + \mathbb{E}[\text{length of the list}] = 1 + n \cdot \frac{1}{m} = 1 + \frac{n}{m}$$

(1: constant time computation of $h(x)$)

$$= 1 + \sum_{k \in D} 1 \cdot P(h(k) = i) = 1 + \frac{n}{m}$$

$\frac{n}{m} = \alpha$ = LOAD FACTOR ($n$: # of keys, $|T| = m$)

=> $O(1 + \frac{n}{m})$ time

=> $O(1 + \alpha)$ search

- if size of table $m = n$ $\sqrt{}$ cost becomes constant $(\frac{n}{n})$

BUT THE PROBLEM is (n)?

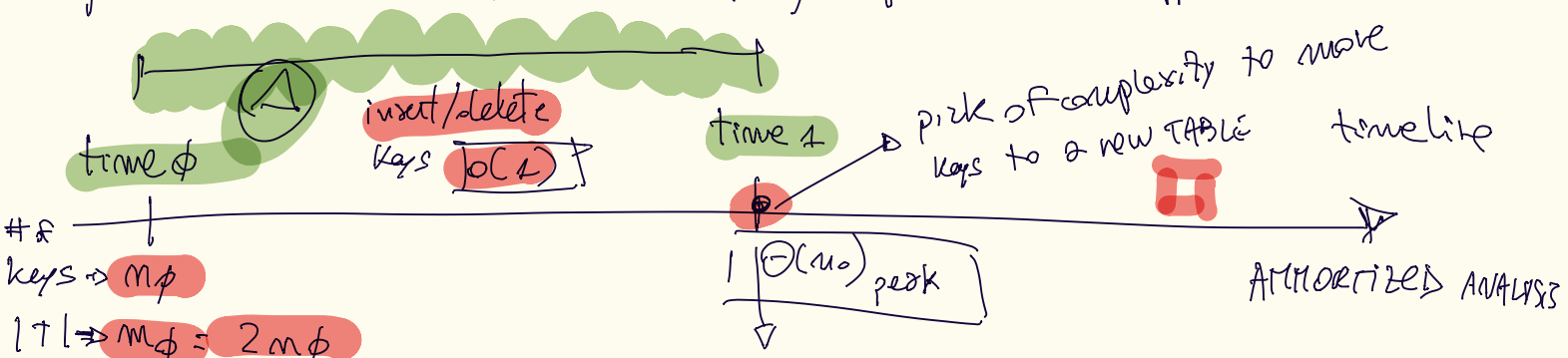
IF 1) (M) is static (fixed # of keys) => (dx)

BUT considering insertion() and deletion()

2) (M) is dynamic (changes) => managed by

**GLOBAL REBUILDING**

- Want: ~~constant time operation~~ when (M) is dynamic (# of keys)
  - at time $\phi$, we have some # of keys $n_\phi$ and size of table $m_\phi = 2n_\phi$
  - keys are added to the table, look how complexity change to the hash approach




$$\alpha = \frac{n}{m} = \frac{n_\phi}{2n_\phi} = \frac{1}{2}$$

$\frac{1}{2}$ => constant

=> insertion & deletion

$$\Rightarrow O\left(1 + \frac{n}{m}\right) = \frac{1}{2} = \text{CONSTANT TIME}$$

ESCAPE TIME

1) (*) # of keys $\geq 2n_\phi$ (fill the table)
or
2) # of keys $\leq \frac{n_\phi}{2}$

if 1 or 2 one TIME => DO SOMETHING

NOT in 1 or 2

$$\frac{n_\phi}{2} < \#\text{keys} < 2n_\phi$$

$\frac{n_\phi}{4}$ $\qquad$ $m_\phi$

- in this range $|T|$ at the beginning is proportional to the # of keys (operation executed between to RT is CONSTANT)

$n_\phi = 6$

$m_\phi = 2 \cdot 6 = 12$

$$\alpha = \frac{n_\phi}{2n_\phi} = \frac{1}{2}$$

$$\frac{n_0}{2} = \frac{6}{2} = 3$$

- In cases 1 or 2 # keys are too large or too small

=> in those cases: 1 table DOUBLES, 2 table HALVES

=> RESIZE THE TABLE IN 1 or 2

- in case (1) => TABLE size gets $(M_1) = 2M_\phi = 4M_\phi$ (DOUBLES)
- in case (2) => TABLE size gets $(M_1) = \frac{M_\phi}{2} = \frac{2M_\phi}{2} = M_\phi$ (halves)

(4) In that time frame, between $T_0$ & $T_1$ is IMPORTANT that is guaranteed that # of keys ($n$) is PROPORTIONAL (for a given constant) to the size of the table
=> operations cost constant time $O(1)$

(1) How much it costs! (REBUILDING at $T_1$?)

CASE (1): keys DOUBLES
=> • move keys into the new table => I'm moving $2M_\phi$ keys and insertion is still constant $O(2M_\phi \cdot 1)$
• destroy old table

1) Moving $2M_\phi$ keys => insertion: $O(2M_\phi \cdot 1)$
2) Destroy old table $T_0$ => $O(M_\phi) = O(2M_\phi)$ (ITS size $M_\phi$) $\Downarrow$ $O(M_\phi)$
3) Create new table $T_1$ => $O(M_1) = O(4M_\phi)$ (ITS size $M_1$)
=> AMMORTIZED $\frac{O(M_\phi)}{M_\phi} = O(1)$ => CONSTANT

+ destroying old table
+ constructing new table

$$=> O(2M_\phi \cdot 1 + M_\phi + M_1) =$$

($2M_\phi$, $4M_\phi$)

$$= \boxed{O(M_\phi)}$$ take a cost proportional to the keys

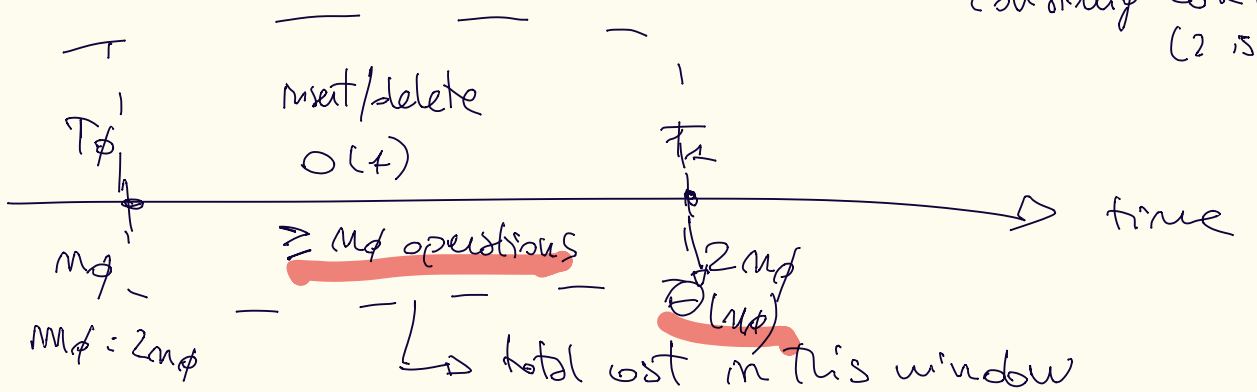
=> so at $T_1$ on the line I'm getting $\Theta(n_0)$ (peak)
exactly this cost

**AMMORTIZED ANALYSIS**: not counting (constant, $n_0$, constant ...) point-wise complexity.

• but I'm counting complexity over a sequence of operations

- consider window $T_0$ —— $T_1$

Consider case ①
(② is analogue)




↳ total cost in this window

→ start with $n_0$, when arrive to $T_1$

# keys gets $2n_\phi$

⇒ at least I did $n_\phi$ operations (consider insertions)

$\geq M_\phi$ operations

(each costs $O(1)$)

→ So in the window

I pay at least

$(\geq)$ $M_\phi + M_\phi = 2n_\phi$ so asymptotically $\Theta(n_0)$ (including at least $M_\phi$ operations)

⇒ Spread total cost to each ($n_\phi$ operation)

(like a loan)

⇒ $\frac{\text{WORST CASE COST of a sequence of operations}}{\text{\# of operations}}$ like $\frac{\text{WC one}}{\text{AVG one}}$

⇒ $\frac{\Theta(n_0)}{n_0}$ ⇒ CONSTANT $O(1)$

⇒ COST OF HASHING IS AMORTIZED CONSTANT

⇒ so all (5k) if I have simple uniform hashing

⇓

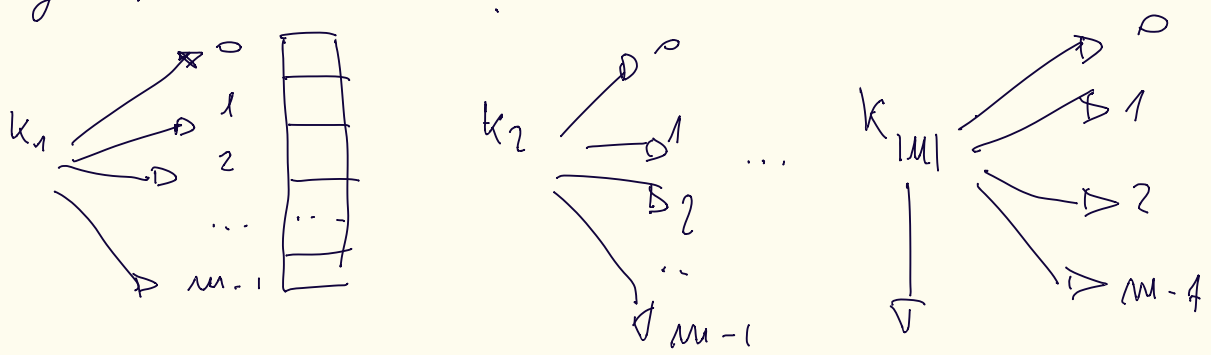
NOT POSSIBLE

- SIMPLE UNIFORM HASHING is NOT possible!

↳ every key can go everywhere in the table (uniformly distributed at random)

- $h: U \to [m]$

- to have $h$ simple uniform, $h$ must be selected from a class of hash functions ($h \in \mathcal{H}$) such that is able to map every key $k$ in $U$ in every slot of the table $T$.




in universe $U$ I've $|U|$ keys

mappings: $m \cdot m \cdot \ldots \cdot m$

=> ~~# of simple hash f(x)s~~ $|\mathcal{H}| = m^{|U|}$

=> # of mappings: $m^{|U|}$

How big is $\mathcal{H}$? => $|\mathcal{H}|$ must include all $m^{|U|}$ mappings

=> $|\mathcal{H}| = \#\text{mappings} = m^{|U|}$

- To represent each $h \in \mathcal{H}$ need at least $\geq \log_2 m^{|U|}$ bits ($\geq$)

↓

bits to represent something: $\log_2$ SOMETHING

=> $\Omega(\log_2 m^{|U|}) = \Omega(|U| \log_2 m)$ bits

(bits)

=> every function $h$ requires $|U| \log_2 m$ bits

ex. keys of 8 bytes, $U = 2^{8\text{ bytes}} = 2^{8 \cdot 8\text{ bits}} = 2^{64}$ just size of the universe

=> NOT AFFORDABLE

# UNIVERSAL HASHING

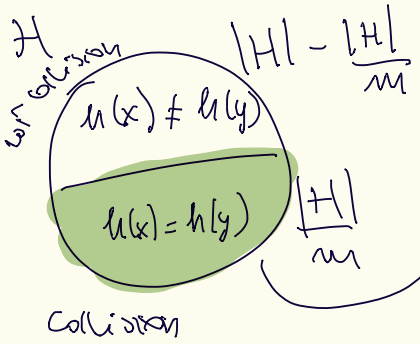
→ Class $\mathcal{H}$ of functions $h: U \to [m]$ is called universal hashing iff

$\forall$ pair of keys $x \neq y \in U$ the $\left(\#\text{ of } h : h(x) = h(y)\right) \leq \frac{|\mathcal{H}|}{m}$

such that you have a collision

to good hash fns

upper bounded




so

given $x, y \in U$ $(x \neq y)$

so $\mathbb{P}$ (given $x, y \in U, x \neq y$ I get $h(x) = h(y)$)

$= \frac{\frac{|\mathcal{H}|}{m}}{|\mathcal{H}|} = \frac{|\mathcal{H}|}{m} \cdot \frac{1}{|\mathcal{H}|} = \boxed{\frac{1}{m}}$

- reasoning with $U$ is too much

⇒ OK reasoning with pairs of objects $(x, y)$

- $\forall$ pair of keys, I've a collision with probability $\frac{1}{m}$

## Universal Hashing with chaining

- assume universal hash class exists
- pick $h \in \mathcal{H}$ at random
- want to see AVG list length

$\mathbb{E}$[list length of cell $[i]$] $= \sum_{k' \in D,\ k' \neq k} 1 \cdot \mathbb{P}(h(k) = h(k'))$

$h(k)$ ← $[i]$

$h \in \mathcal{H}$ picked at random

collision

by the definition of universal hash function

$\frac{1}{m} = \frac{\frac{|\mathcal{H}|}{m}}{|\mathcal{H}|} = \frac{|\mathcal{H}|}{m} \cdot \frac{1}{|\mathcal{H}|} = \frac{1}{m}$

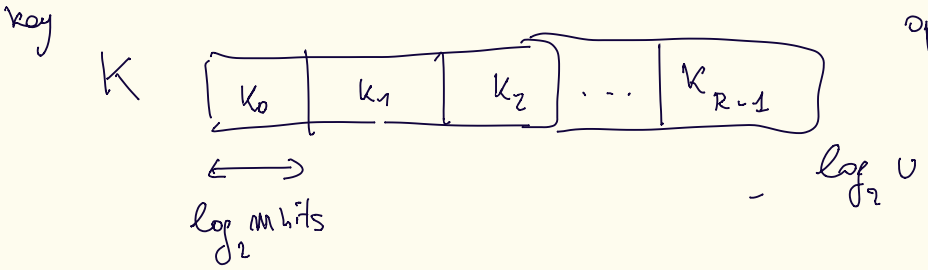
$\leq \sum_{k' \in D} \frac{1}{m} \cdot 1 = \frac{|D|}{m} = \frac{n}{m} = \alpha$

same as SIMPLE UNIFORM HASH

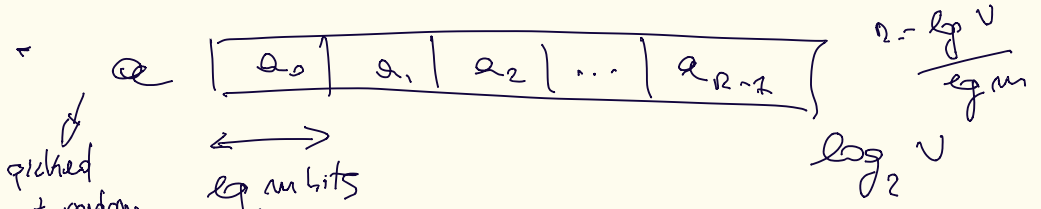
BUT EXISTS?

# example of universal hash function

- class of universal H
- $h: U \to [m]$ => key is $\log_2 |U|$ bits
- every key k taken from U consists $\log_2 u$ bits, $u = |U|$
- take the key, divide it into blocks: $k_0, k_1, k_2, \dots, k_{R-1}$ where $R = \frac{\log_2 u}{\log_2 m}$ # of blocks
- every block is $\log_2 m$ bits, $\log_2 m$ bits can't $\forall k_i$ $0 \le k_i < m$
  - $\forall k_i$: for all blocks
  - every block is at most the size of codomain




- construct hash function taken from a class, => I'll have a parameter a that defines the hash functions.
  - picked at random
  - of size $\log_2 U$




$0 < a < U$

I multiply a · key (don't want ∅)

$0 \le k_i < U$
$a_i$

- where
  - $0 \le k_i < U$, $a_i$
  - $0 < a < U$

Class H:

- hash function:

$$h_a(k) = \sum_{i=0}^{R-1} a_i \cdot k_i \bmod m$$

where $a =>$

- $a_i$: item
- $k_i$: key
- m: prime
- R: # blocks

$|H| = \#a = U - 1$ wt $0 < a \le u$

- |H|: how many hash functions I've?
- a: parameter

To be a universal hash function CLASS I've to proof that: $\forall$ given $x, y$; $x \neq y$ $\quad \#h_a : [h_a(x) = h_a(y)] \leq \frac{|H|}{m}$

# of hash (A) COUNT THIS NUMBER

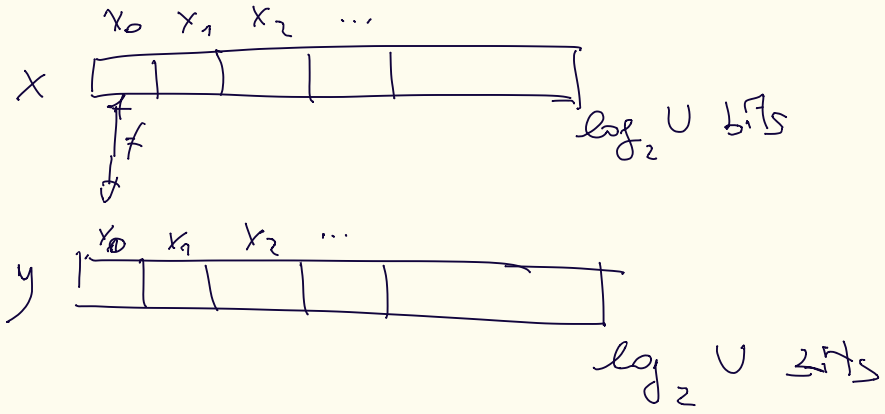
$\Rightarrow$

- in order to have a collision

WANT TO COUNT $\rightarrow$ $\#h_a : \sum_{i=0}^{R-1} a_i \cdot x_i \bmod m = \sum_{i=0}^{R-1} a_i \cdot y_i \bmod m$.

$\Leftarrow$ SO THE NUMBER OF HASH FUNCTIONS SUCH THAT THIS EQUALITY IS TRUE (*)

- assume $x \neq y$ $\Rightarrow$ at least there is 1 pair of blocks that is different es. $x_0 \neq y_0$




(*)

$\Rightarrow \#h_a : \sum_{i=0}^{R-1} a_i \cdot x_i \equiv \sum_{i=0}^{R-1} a_i \cdot y_i \pmod m$

1) I know just $x_0$ and $y_0$ are different (takes $x_0$ out and $y_0$ out) $\quad x_i$ are different

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

$$\Rightarrow a_0 \cdot x_0 - a_0 y_0 \equiv \sum_{i=1}^{R-1} a_i \cdot x_i - \sum_{i=1}^{R-1} a_i \cdot x_i \pmod m$$

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

$(x_0 - y_0) \neq 0$ driven by (1) the assumption they're different

m prime (2)

SINCE (1) & (2) $\Rightarrow \exists$ an inverse $\Rightarrow \exists$ an integer $V = (x_0 - y_0)^{-1} \bmod m$

$\Rightarrow$ $\exists$ an integer such that $(x_0 - y_0)^{-1} \cdot (x_0 - y_0) \equiv 1 \bmod m$

$\Rightarrow$ multiply all sides by this integer: $(x_0 - y_0)^{-1}$

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

#a are fixed by

$a_0$ , $a_1$ , $a_2$ ... $a_{R-1}$

$a_0$ : ?

$a_1$ : $m$ : $\log_2 m$ bits

$a_2$ : $m$ : idem

$\dots$ : $m$ (A)

$\Rightarrow$ Can be chosen $m$-ways in $m$-ways

in this way (fixed)

To have a collision $\Rightarrow$ $a_0$ must not be chosen as I want it)

$\Rightarrow$ How many #a induce the collision? (A)

$$\left( \#h_a : h_a(x) = h_a(y) \leq \frac{|H|}{m} \right.$$

$\Rightarrow$ $m^{R-1} - 1$

↳ Brute force

$$2^{\log m^R} = 2^{R \log m}$$

$$m^{R-1} - 1 = m^R : m^1 - 1 = \frac{m^R}{m} - 1 = \frac{2^{R \cdot \log m}}{m} - 1 =$$

NB: $R = \frac{\log U}{\log m}$

$$= \frac{2^{\frac{\log U}{\log m} \cdot \log m}}{m} - 1 = \frac{2^{\log_2 U}}{m} - 1$$

$$= \frac{U}{m} - 1 = \frac{U - m}{m} \leq \frac{U - 1}{m} =$$

$$= \left( \frac{|H|}{m} \right) \checkmark$$

Cardinality of $|H|$ = # of hash functions = $U - 1$

just another example

$$\Rightarrow \text{so } h_a(k) = \left[ \sum_{i=0}^{R-1} a_i \cdot k_i \bmod p \right] \bmod m$$

$p$: prime, $p > m$

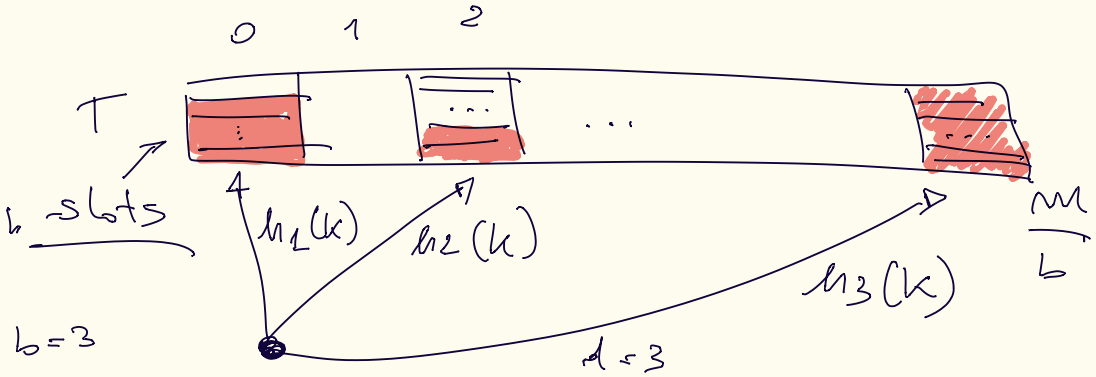
$m$: power of 2

## D-LEFT HASHING

- Want avoid scanning list } of hash with chaining
- " reduce # of pointers }

=> idea

- take the TABLE
- PARTITION IT INTO BEANS
- EVERY BEAN has $d$-slots => table is $\frac{m}{d}$
- $d$ = # of hash functions
- es. $d = 3$




/// : filled

IDEA: Given a key, since I've 3 hash fs, I check 3 slots

=> AMONG SLOTS, TAKE THE MOST EMPTY (2)

- Search is CONSTANT $O(3 \cdot d)$

$d = 2$ best choice

$d \geq 2$

$$\mathbb{E}[\text{length of LONGEST list}] = \frac{\log \log n}{\log d} + O(1)$$

- In $d=1$ we don't have choice => we insert directy (sort of chasing)
  ↳ complexity

$\Downarrow$ $\mathbb{E}[\text{lenght of longest list}] = \frac{\log n}{\log \log n}$

POWER OF 2 CHOICES

=> with 1 choice is chopping
=> with going for 2 choices
=> LENGTH OF THE LONGEST LIST DROPS.

# CUCKOO:

THEO: $\forall$ pair of positions $i, j$, $\forall c > 1$ ; if $m \geq 2 \cdot c \cdot n$

$$\Rightarrow \mathbb{P}(i \overset{2}{-} j) = \frac{1}{m \cdot c^2}$$

$$\Rightarrow \alpha = \frac{n}{m} \leq \frac{m}{2c} \cdot \frac{1}{m} = \frac{1}{2c} < \frac{1}{2}$$

$\forall c > 1$

## proof by induction

BASE CASE $L = 1$

$$\mathbb{P}(i - j) = \mathbb{P}\left(\exists \text{ a key } k : \begin{matrix} h_1(k) = i \wedge h_2(k) = j \\ +(\vee) \\ h_1(k) = j \wedge h_2(k) = i \end{matrix}\right) \leq$$

union bound

$$\leq n \cdot \mathbb{P}(k : \text{satisfy}) = n \cdot \left(\mathbb{P}(h_1(k) = i \wedge h_2(k) = j) \overset{\rightarrow \frac{1}{m}}{+} \mathbb{P}(h_1(k) = j \wedge h_2(k) = i)\right) =$$

$$= n \cdot \frac{2}{m^2} \leq \frac{m}{c} \cdot \frac{1}{m^2} = \frac{1}{cm} \quad (\checkmark)$$

INDUCTIVE STEP (THEO FOR $L-1$, proof for $L$)

$$\mathbb{P}(i \overset{L}{-} j) = \mathbb{P}(\exists z : i \overset{L-1}{-} j \overset{L-1}{-} z) \leq$$

(scelto tra $m$)

$$\leq m \cdot \mathbb{P}(i \overset{L-1}{-} j \overset{1}{-} z) \leq m \cdot \left(\frac{1}{m \cdot c^{L-1}} \cdot \frac{1}{c \cdot m}\right) =$$

(inductive hypothesis)

$m$ = # of nodes ($n$ = # edges)

$$= \frac{1}{m \cdot c^L} \quad (\checkmark)$$

REPEAT proof

THEO: $\forall i, j, \forall c > 1$ if $m \geq 2 \cdot c \cdot n$ $\mathbb{P}((i)-(j)) = \frac{1}{m c^l}$

inductive step ($l = 1$)

$$\mathbb{P}((i)-(j)) = \mathbb{P}\left(\exists k : \begin{array}{l} h_1(k) = i \wedge h_2(k) = j \\ \vee \\ h_2(k) = j \wedge h_2(k) = i \end{array}\right) \leq$$

union bound

$$\leq m \cdot \mathbb{P}(\exists k : \text{satisfy that condition}) =$$

$$= m \cdot \frac{2}{m^2} \leq \frac{m}{c} \cdot \frac{1}{m} = \frac{1}{m \cdot c}$$

inductive step: ok FOR $l-1$, proof for $l$

$$\mathbb{P}((i) \xrightarrow{l} (j)) = \mathbb{P}(\exists z : (i) \xrightarrow{l-1} (j) \xrightarrow{1} (z)) \leq$$

$z$ chosen from the $m$

$$\leq m \cdot \mathbb{P}((i) \xrightarrow{l-1} (j) \xrightarrow{1} (z)) \leq m \cdot \left(\frac{1}{m \cdot c^{l-1}} \cdot \frac{1}{m \cdot c}\right) =$$

hypothesis

$$= \frac{1}{m \cdot c^l} \quad \checkmark$$

**WHERE CUCKOO HASHING FAILS** $\Rightarrow$ cycle

$$\mathbb{P}((i) \circlearrowright \quad (j) \circlearrowright) \leq \mathbb{P}((i) \circlearrowright) =$$

$$= \mathbb{P}(\exists i, \exists l : (i) \circlearrowright^{2l}) = \sum_{i} \sum_{l \geq 1} \frac{1}{m \cdot c^{2l}} =$$

node (over $m$)

$$= m \cdot \sum_{l \geq 1} \frac{1}{m \cdot c^{2l}} = \sum_{l \geq 1} \left(\frac{1}{c}\right)^l = \frac{1}{c-1} \Rightarrow$$

$=$ small know serves that converges to

PROBABILITY of FAILURE WHILE BUILDING THE GRAPH

COROLLARIO 1: if $c = 3$ $\mathbb{P}$ (&FAILS during construction) $\leq \frac{1}{2}$

COROLLARIO 2: $\mathbb{P}\left(① \xrightarrow{\text{is connected}} ② \text{ (aka } i, j \text{ collide)}\right) = \mathbb{P}(\exists$ a path $\overset{L}{\text{from}}$ ① to ② aka ① $\overset{L}{—}$ ②$) \leq$

$$\leq \sum_{L \geq 1} \frac{1}{m \cdot c^L} = \frac{1}{m} \sum \frac{1}{c^L} = \frac{1}{m} \cdot \frac{1}{c-1} = \frac{1}{m(c-1)} =$$

well known ↗

$$= \frac{1}{m} \text{ if } c \geq 2$$

$\Rightarrow \mathbb{P}(\text{collision}) \leq \frac{1}{m}$

AVG # keys colliding with same key k: $\frac{n}{m} < 1$

but $n \leq 6m$

$\Rightarrow$ AVG COST OF INSERTION $O(1)$ time

MINIMAL ORDERED PERFECT HASHING , Given a Dictionary (1)

- is STATIC
- cool for strings (for the rank)

NO COLLISIONS

if $k_1 < k_2$
then $h(k_1) < h(k_2)$

$h_1, h_2$ : $n \to m$ , $f$ : $m \to n$

↳ Codomain is $[n]$ (# of keys)

$D = \{AA, BB, BD, CD\}$ sorted

returns: 0, 1, 2, 3 (RANK)

want create an hash function that

- $h_1, h_2$ universal hash function

$h$ → WANT TO GET

| | $h_1$ | $h_2$ | $h$ |
|---|---|---|---|
| AA | 4 | 2 | 0 |
| BB | 1 | 4 | 1 |
| BD | 3 | 6 | 2 |
| CD | 6 | 0 | 3 |

$h_1(xy) = 3\,\text{rank}(x) + \text{rank}(y) \bmod 7$

$h_2(xy) = \text{rank}(x) + \text{rank}(y) \bmod 7$

$(m)_7 > 4$ ✓ codomain must be > than number of strings = n

TO GET $h$:

$h(t) = g(h_1(t)) + g(h_2(t)) \bmod n$

1st example $g(4) + g(2) \bmod 4 = 0$

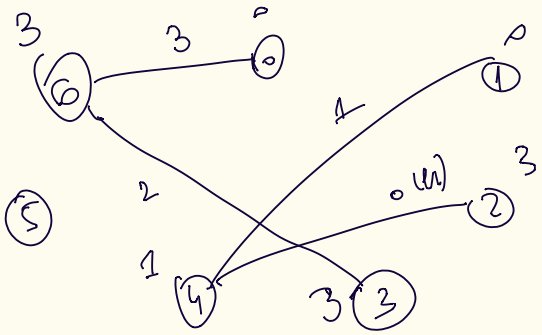
represent $g$ as an array

$g$

| 0 | 0 | 3 | 3 | 1 | 0 | 3 |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 = m−1 |

$|g| = m = 7$

- construct graph: nodes = $|g| = m$ edges: values $h_1 \sim h_2 = n$ = # of strings




mod 4

$P(\text{cycle})$ is small

$m > n$

$m = n \log n$, $m$ prime

graph is SPARSE

$g(10) + g(6) \bmod 4 = 3$

$g(6) + g(3) \bmod 4 = 2$

$\quad\;\; \| \qquad \| \qquad$

$\quad\;\; 3 \qquad 3$

$f(1) + f(4) \bmod 4 = 1$

$\quad\;\; \| \qquad \| \qquad$

$\quad\;\; 0 \qquad 1$

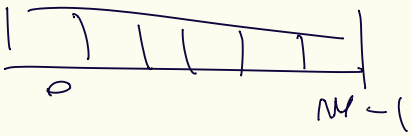
- if I'm a cycle => redraw others $h_1$ & $h_2$
- SPACE: store $h_1, h_2$ (universal hashes)
  store $f$ => $m$ cells
  $m \simeq n \log n$

Bloom Filter

- membership (k) } O(r) times
- insertion (k)
- no deletion

- r: hash functions

Binary Array B




- insertion (k):

$\forall i = 1, 2, \dots, r \quad B[h_i(k)] = 1$

- membership (k): yes iff $\forall i = 1, 2, \dots, r \quad B[h_i(k)] = 1$ (FALSE POSITIVE)

no iff $\forall i = 1, 2, \dots, r \quad B[h_i(k)] = 0$

- prob of ERRORS:

→ r-hash functions

$$P(B[j] = 0) = \left(\frac{M-1}{M}\right)^{r \cdot n} = \left(1 - \frac{1}{M}\right)^{r \cdot n} = \left(1 - \frac{1}{M}\right)^{\frac{r \cdot n}{M} \cdot M} =$$

$$= \left[\left(1 - \frac{1}{M}\right)^{M}\right]^{\frac{r \cdot n}{M}} \simeq e^{-\frac{r \cdot n}{M}}$$

so $P(B[j] = 1) = 1 - e^{-\frac{rn}{M}}$

$$\Rightarrow P(\text{errors}) = P(k \notin D, B[h_i(k) = 1] \ \forall i = 1, 2, \dots, r) \simeq$$

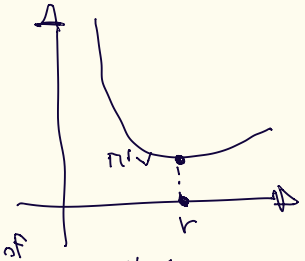
$$\simeq \left(1 - e^{-\frac{rn}{M}}\right)^r \checkmark$$

- # hash functions that minimize the error: $\frac{M}{n} \ln 2$

- r = # hash functions

- r optimal = $\frac{M}{n} \ln 2$

- $\varepsilon_{(\text{error})} = \left(\frac{1}{2}\right)^{r \text{ optimal}}$




- $P(\text{error}) = f(n, M, r) = \left(1 - e^{-\frac{rn}{M}}\right)^r$

- THEOREM: SPACE

every DS that makes an error $\varepsilon$ over $n$ keys (occupying $n$ space) must uses at least $m$-bits such that:

$$m \geq n \log_2 \frac{1}{\varepsilon}$$

$\varepsilon = \left(\frac{1}{2}\right)^r$ optimal

$$\varepsilon = \left(\frac{1}{2}\right)^{\frac{m}{n} \ln 2}$$

$$\frac{1}{\varepsilon} = 2^{\frac{m}{n} \ln 2}$$

$$\log \frac{1}{\varepsilon} = \log_2 2^{\frac{m}{n} \ln 2}$$

$$\log \frac{1}{\varepsilon} = \frac{m}{n} \ln 2$$

$\simeq 1{,}44$

$$\log \frac{1}{\varepsilon} \cdot n \cdot \frac{1}{\ln 2} = m$$

$$m = n \frac{\log_2(1/\epsilon)}{\ln 2} \approx 1.44\, n \log_2(1/\epsilon).$$

$$\Rightarrow \quad m = \frac{n \cdot \log_2 \frac{1}{\varepsilon}}{\ln 2} = 1{,}44\, n \log_2 \frac{1}{\varepsilon}$$

---

SPACES (store a key | pointer | table)

- HASH WITH CHAINING: $n \log U + n \log n + m \log n$ BITS
- cuckoo: $\underbrace{n \log U}_{\text{key}} + \text{NO pointers} + \underbrace{m \log n}_{\text{TABLE}}$ BITS
- Bloom filter:

$m$ bits $\quad m = c \cdot n$

$\Rightarrow$ $\forall$ key I pay $c$-bits

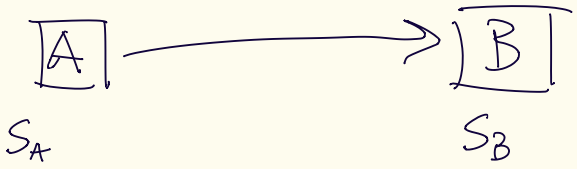
example of BF

- $P(\text{error}) = \left(\frac{1}{2}\right)^{\varepsilon}$ - optimal
- $\gamma$ - optimal $= \frac{m}{n} \ln 2$

↓ # hash fs

- PPP
- A, B want to exchange data (film, music)
- estimate $A \cap B \Rightarrow S_A \cap S_B$ (with and without errors)

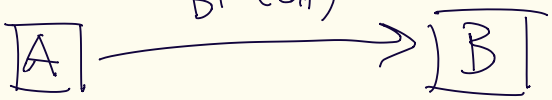
- $S_A$ and $S_B$ are set of items of the respectively machines




- A sends to B its set and B computes the intersection
  => TOO COSTLY!
  => send bits really needed

---

- Compute BF for $S_A$ : define size and # h functions




→ • $S_A$

→ • $BF(S_A)$

• $S_B$ => B receives $BF(S_A)$

• B computes $S_A \cap S_B$
  => SCAN $S_B$
  - $\forall$ element of $S_B$ check $BF(S_A)$
  if $r$ NO => element is surely not in the intersection

if size $m = c \cdot n \rightarrow (0,6185)^c$ error

and $r = \frac{m}{n} \ln 2$ hash functions

n: keys

=> size • $m_A = c \cdot |S_A|$

→ • $r_A = \frac{m_A}{n_A} \ln 2$

if YES => maybe
- we take the element as belong to intersection (maybe correct, maybe error)

(X) yes: → element surely in $S_A \cap S_B$

→ extra elements (BF says yes but $\notin S_A \cap S_B$)

↗ $k \to BF(k) = yes$ but $k \notin S_A \cap S_B$

CANNOT AVOID ⇒ compute the AVG

⇒ PROBABILITY of false $\varepsilon$

⇒ $\mathbb{E}[\#\text{errors}] = P(\text{error}) \cdot \#\text{of checks I do} =$

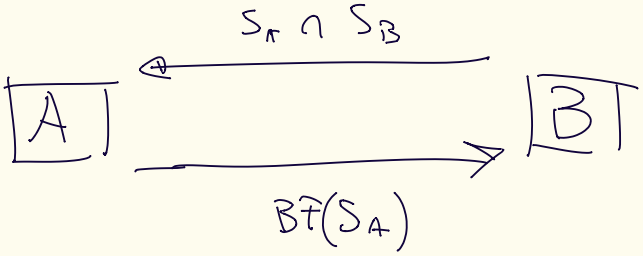
$= \left(\frac{1}{2}\right)^{\text{k-optimal}} \cdot |S_B| =$

$= \left(\frac{1}{2}\right)^{\frac{m}{n}\ln 2} \cdot |S_B| =$

$\hookrightarrow (0,6185)^{\varepsilon} \cdot |S_B|$

⇒ if $c$ is large the prob of error is small

- if I want to be sure, send back the intersection



# SPECTRAL BLOOM FILTER

- BF with counters
- supports deletion
- r hash functions
- insertion(k) => update (increment) the value of the codomain

$$\forall i = 1, 2, \dots, r \; ; \; C[h_i(k)] += 1$$

- deletion(k) => update (decrement) the value of the codomain

$$\forall i = 1, 2, \dots, r \; ; \; C[h_i(k)] -= 1$$

- query(k) : frequency of a key

$$\min\left(C[h_i(k)]\right) \quad \forall i = 1, 2, \dots, r$$

- $P(\text{error}) = P(\text{overflow}) = \left(\frac{1}{2}\right)^{r \cdot \text{opt}} = \left(\frac{1}{2}\right)^{\frac{m}{n}(\ln 2)} \simeq (0,6)^{\frac{m}{n}}$

$$2^{5 \cdot 2} = (2 \cdot 2)^3$$

$$2^6 = 4^3$$

Repeat:

## Hash with chaining

- in presence of a simple uniform hash $\mathcal{H}$ we hash with chaining next support search in $O(1 + \frac{n_{max}}{m})$, $\frac{n}{m} = \alpha$ load factor

$$E[\text{list length}] = 1 + \frac{1}{m} \cdot n = 1 + \frac{n}{m}$$

$$\sum_{i=0}^{m} P(h(k) = i) = n \cdot \frac{1}{m}$$

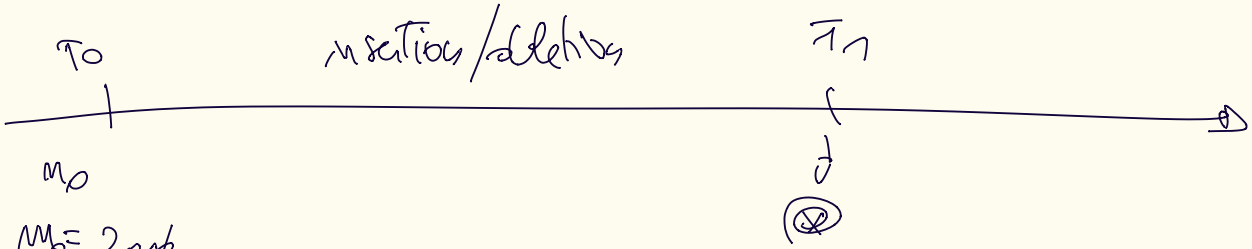
if $m = n$

$\Rightarrow 1 + \frac{1}{1} = 2$

$\Rightarrow$ CONSTANT

## OK if $n$ is static

WANT THE WHEN $n$ IS dynamic




- could happen 2 cases: ① $n$ doubles: $\leq 2n_0$ $n_1 = 2 \cdot 2n_0 = 4n_0$

② $n$ halves $\leq \frac{n_0}{2}$ $n_1 = \frac{2n_0}{2} = n_0$

Case 1 (example for $2n_0$)

① cost of: - moving $n_0$ keys $= O(1 + n_0) = O(n_0)$ +

- build new table $= O(n_1) = O(4n_0) = O(n_0)$ +

- destroy old table $= O(n_0) = O(2n_0) = O(n_0)$

$\Rightarrow O(n_0)$

The search is CONSTANT if I'm considering the amortized cost:

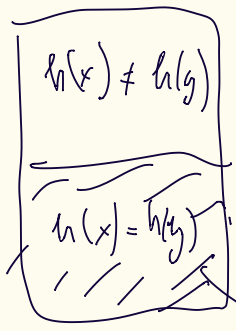
$$\frac{O(n_0)}{n_0} = O(1)$$

• Universal hash

$h: U \to [m]$ ; $h \in H$, $H$ is a class of universal hash functions if ($\#h$ such that $\forall x, y \in U$ ; $x \neq y$

$$h(x) = h(y) \leq \frac{|H|}{m}$$ )

if




$$\Rightarrow P(\text{I'm here}) = \frac{|H|}{m} \cdot \frac{1}{|H|} = \frac{1}{m}$$

• Universal Hash with chaining

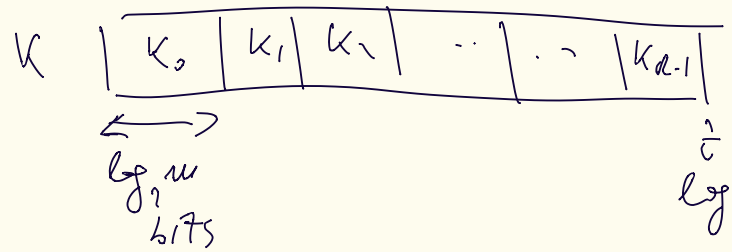
• picking $h \in H$ at random, AUG last length

$$E[\text{LA length}] = \sum_{\substack{k \in D \\ k \neq k'}} 1 \cdot P(h(k) = h(k')) =$$

$$= n \cdot \frac{1}{m} = \frac{n}{m}$$

• Give an example of universal Hash function

- suppose $k$ divided into blocks, ideas a process a




$$R = \frac{\log_2 U}{\log_2 m}$$


$$\Rightarrow h_a = \sum_{i=0}^{r-1} k_i \cdot a_i \bmod m$$

m prime

$$\downarrow\downarrow$$

$$0 \leq \frac{a_i}{k_i} < U$$

$$0 < a < U$$

to be universal both functions, given $x, y \in U$, $x \neq y$

$\#h_{\text{esh functions}} : (h(x) = h(y)) \leq \frac{|H|}{m}$

- to have a collision

$\#a_i : \sum_{i=0}^{R-1} x_i \cdot a_i \bmod m = \sum_{i=0}^{R-1} y_i \cdot a_i \bmod m$

$\#$ of hash functions

$$\sum_{i=0}^{R-1} x_i \cdot a_i \equiv \sum_{i=0}^{R-1} y_i \cdot a_i \bmod m$$

$$a_0 \cdot x_0 + \sum_{i=1}^{R-1} (x_i \cdot a_i) \equiv a_0 \cdot y_0 + \sum_{i=1}^{R-1} y_i \cdot a_i \bmod m$$

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

multiply for the inverse $(x_0 - y_0)^{-1}$

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

To have a collisions

$a_0 \neq a_1 + a_2 + \dots a_{R-1}$ or $a = 0$

$m$ - ways

$$m^{R-1} - 1 \leq \frac{m^R}{m} - 1 = \frac{2^{\log_2 m^R}}{m} - 1 =$$

$$= \frac{2^{\log_2 m^{\frac{\log U}{\log m}}}}{m} - 1 = \frac{2^{\frac{\log U}{\log m} \cdot \log_2 m}}{m} - 1 =$$

$$= \frac{2^{\log_2 U}}{m} - 1 = \frac{U}{m} - 1 = \frac{U - m}{m} \leq \frac{U - 1}{m} = \frac{|H|}{m}$$ ✓

# cuckoo hashing

**Theo** $\forall$ position $i, j$, $\forall c > 1$ : if $m \geq 2 \cdot c \cdot n$

$$P\left(i \overset{L}{-} j\right) = \frac{1}{m \cdot c^L}$$

case base $L = 1$

$$P(i - j) = P(\exists k : h_1(k) = i \wedge h_2(k) = j \vee h_1(k) = j \wedge h_2(k) = i)$$

$\leq$ ...

$P(h_1(k) = j) = \frac{1}{m}$

$$\leq m \cdot P(\text{satisfy}) = m \cdot \frac{2}{m^2}$$

$$\frac{2m}{m^2} \leq \frac{m}{c} \cdot \frac{1}{m^2} = \frac{1}{mc} \qquad \frac{2m}{m^2} \leq \frac{m}{c} \cdot \frac{1}{m^2} = \frac{1}{mc}$$

$\Rightarrow$ Induction step, true for $L-1$

$$P\left(i \overset{L}{-} j\right) = P\left(\exists z : i \overset{L-1}{-} z \overset{L-1}{-} j\right) \leq$$

union bound, $z$ can be $m$

$$\leq m \cdot \left(\frac{1}{m c^{L-1}}\right) \cdot \frac{1}{mc} = \frac{1}{m c^L} \quad \checkmark$$

---

Evaluate the probability that cuckoo hashing fails building the graph

$$P(\text{fails}) \leq P(\text{self-loop}) =$$

$$= P(\exists \text{ node } i : P(\text{self-loop at } i) = \frac{1}{mc^L}) =$$

union bound

$$\leq m \cdot \sum_{L \geq 1} \frac{1}{m c^L} = \sum_{L \geq 1} \frac{1}{c^L} = \frac{1}{c-1}$$

Bloom filter:

Insert(k)
Membership(k) } $O(n)$

$R$ = # of hash functions

e.g.

Insertion(k) => $\forall i = 1, \dots, R \quad h_i(k) = 1$

=> if yes => false positive
no => ok

EVALUATE

$P[\text{error}]$

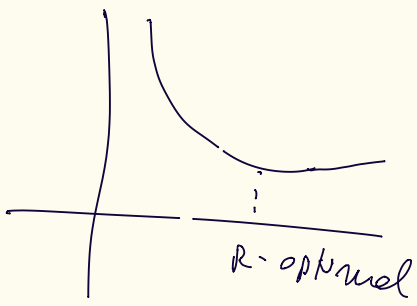
- $P(B[j] = 0) = \left(\frac{m-1}{m}\right)^{R \cdot n} = \left(1 - \frac{1}{m}\right)^{R \cdot n \cdot \frac{m}{m}} =$

$= \left[\left(1 - \frac{1}{m}\right)^{m}\right]^{\frac{R \cdot n}{m}} = e^{-\frac{rn}{m}}$

- $P[B[j] = 1] = 1 - e^{-\frac{rn}{m}}$

$P(\text{error}) = P(k \notin D \text{ but } B[h_i(k)] = 1 \quad \forall i = 1, 2, \dots R) =$

$= \left(1 - e^{-\frac{rn}{m}}\right)^{R}$




$R \text{ optimal} = \frac{m}{n} \ln 2$

$\varepsilon_{opt} = \left(\frac{1}{2}\right)^{R_{opt}} = \left(\frac{1}{2}\right)^{\frac{m}{n} \ln 2} = \ln 2 \left(\frac{1}{2}\right)^{\frac{m}{n}} =$

$= (0{,}6)^{\frac{m}{n}}$

Thm any DS that makes an error $\varepsilon$ over $n$ keys (of size $n$) must uses at least

$$m \geq n \log \frac{1}{\varepsilon} \quad 6.75$$

Proof

$$\varepsilon = \left(\frac{1}{2}\right)^{2 \text{ opt}}$$

$$\varepsilon = \left(\frac{1}{2}\right)^{\frac{m}{n} \ln 2}$$

$$\frac{1}{\varepsilon} = 2^{\frac{m}{n} \ln 2}$$

$$\log_2 \frac{1}{\varepsilon} = \log_2 2^{\frac{m}{n} \ln 2}$$

$$\log_2 \frac{1}{\varepsilon} = \frac{m}{n} \ln 2$$

$$\left(\log_2 \frac{1}{\varepsilon}\right) n = m \ln 2$$

$$m = \log_2 \frac{1}{\varepsilon} \cdot n \cdot \left(\frac{1}{\ln 2}\right) \to 1.44$$