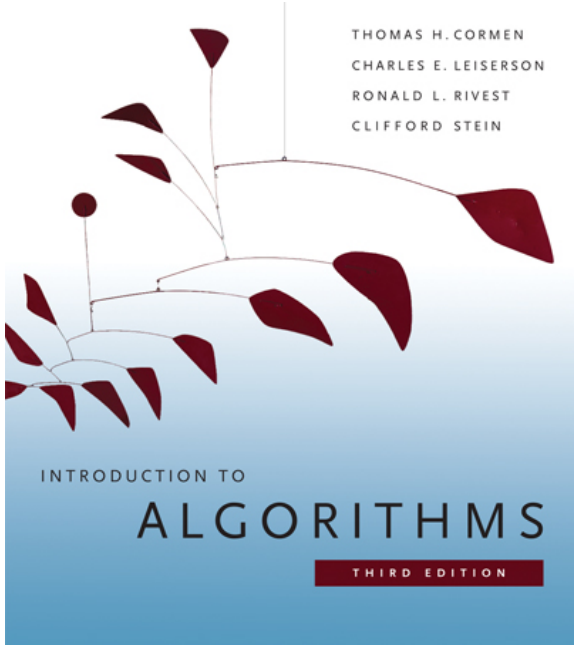
6.006- *Introduction to Algorithms*



Lecture 7

Prof. Silvio Micali

Plan for Today:

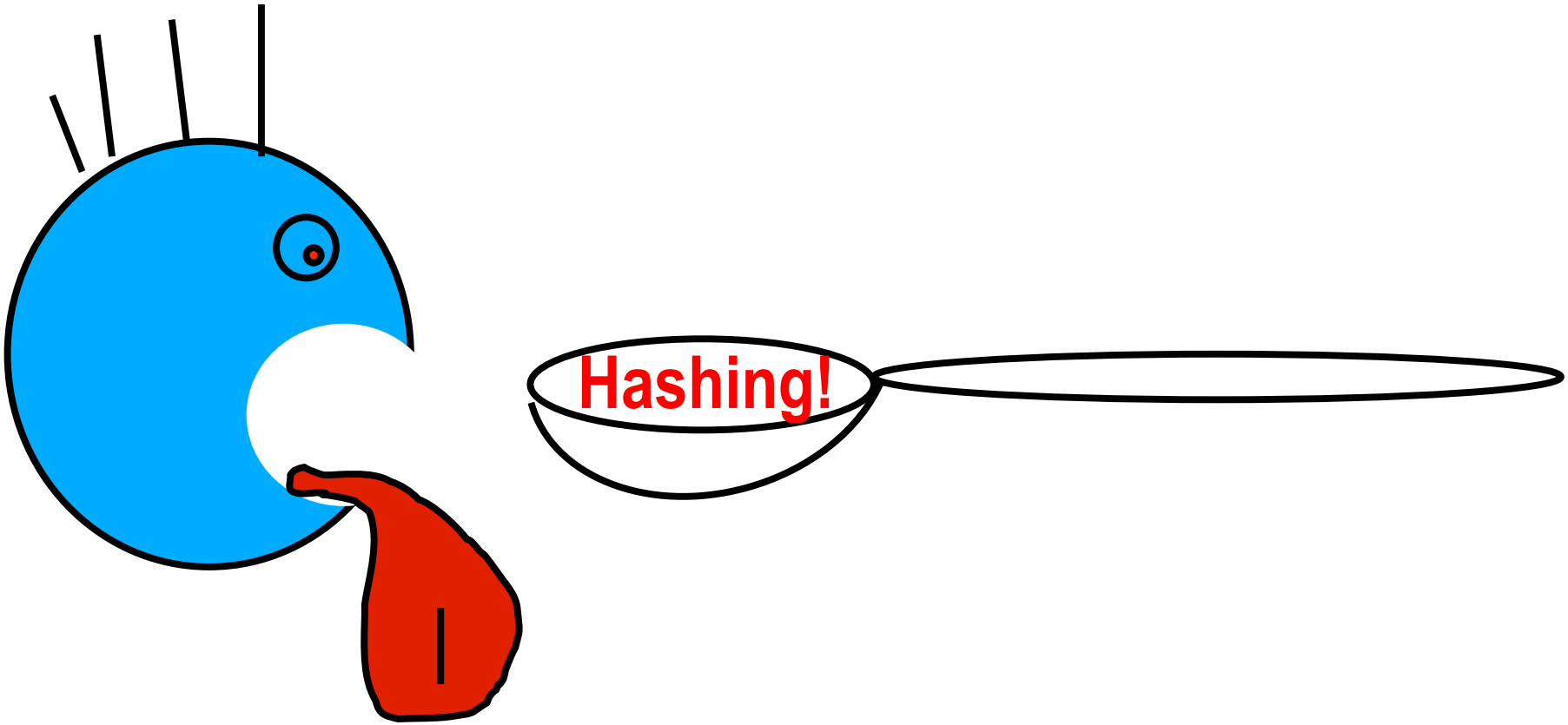
3 new Ideas, 2 of which GREAT!

Congratulations!

Sit down
Focus
Enjoy

Vote at the end…

How to convey these new cool iDEAS?



Idea 1

VIA: DYNAMIC DICTIONARIES

Dynamic Dictionaries

- **So far:** Insert n items in m-size table
- **Now:** *arbitrary* sequence of insert, delete, find n?
- How big a table should we set up?
- What if we guess wrong?

too small ➔ load high, operations slow
too large ➔ high initialization cost, wasted space

Wanted: $m=\Theta(n)$ at all times

potentially more cache-misses

Solution: Resize

- Start with small constant m
- When table too full, make it bigger
- When table too empty, make it smaller

How?

Build a whole new hash table and reinsert items

(Recompute all hashes, Recreate new linked lists)

Time to rebuild: NewSize + #hashes × HashTime

(For simplicity: ignore HashTime)

When to resize?

Approach 1: whenever n > m, $m \leftarrow m+1$

Sequence of n inserts:

- Each insert increases n past m causing rebuild
- Total work: $\Theta(1 + 2 + \ldots + n) = \Theta(n^2)$

Approach 2: Whenever $n \geq 2m$

- ***Costly inserts*:** insert 2^i for all i:

  These cost: $\Theta(1 + 2 + 4 + \ldots + n) = \Theta(n)$

- All other inserts take $O(1)$ time – ***why?***
- Inserting n items takes $O(n)$ time
- Keeps m a power of 2

Amortized Analysis

- If a sequence of n operations takes time T, then each operation has amortized cost T/n

- Some ops are very slow: $\Theta(n)$ for insertion that causes last resize

- But fast amortized cost per operation: $O(1)$

- Often only care about total runtime, so low amortized time is great

Deletions?

- Rebuild table to new size when $n < m$? No: $O(n^2)$
- Rebuild when $n < \frac{m}{2}$

Arbitrary Insertions + Deletions?

Suppose "just rebuilt": $m = n$

- Next rebuild a "grow" ⇒ at least m more inserts before growing table

  Amortized **insert** cost O(2m / m)) = O(1)

- Next rebuild a "shrink" ⇒ at least m/2 more deletes before shrinking

  Amortized **delete** cost O(m/2 / (m/2)) = O(1)

Summary

- Arbitrary sequence of insert/delete/find
- O(1) amortized time per operation

Welcome to:

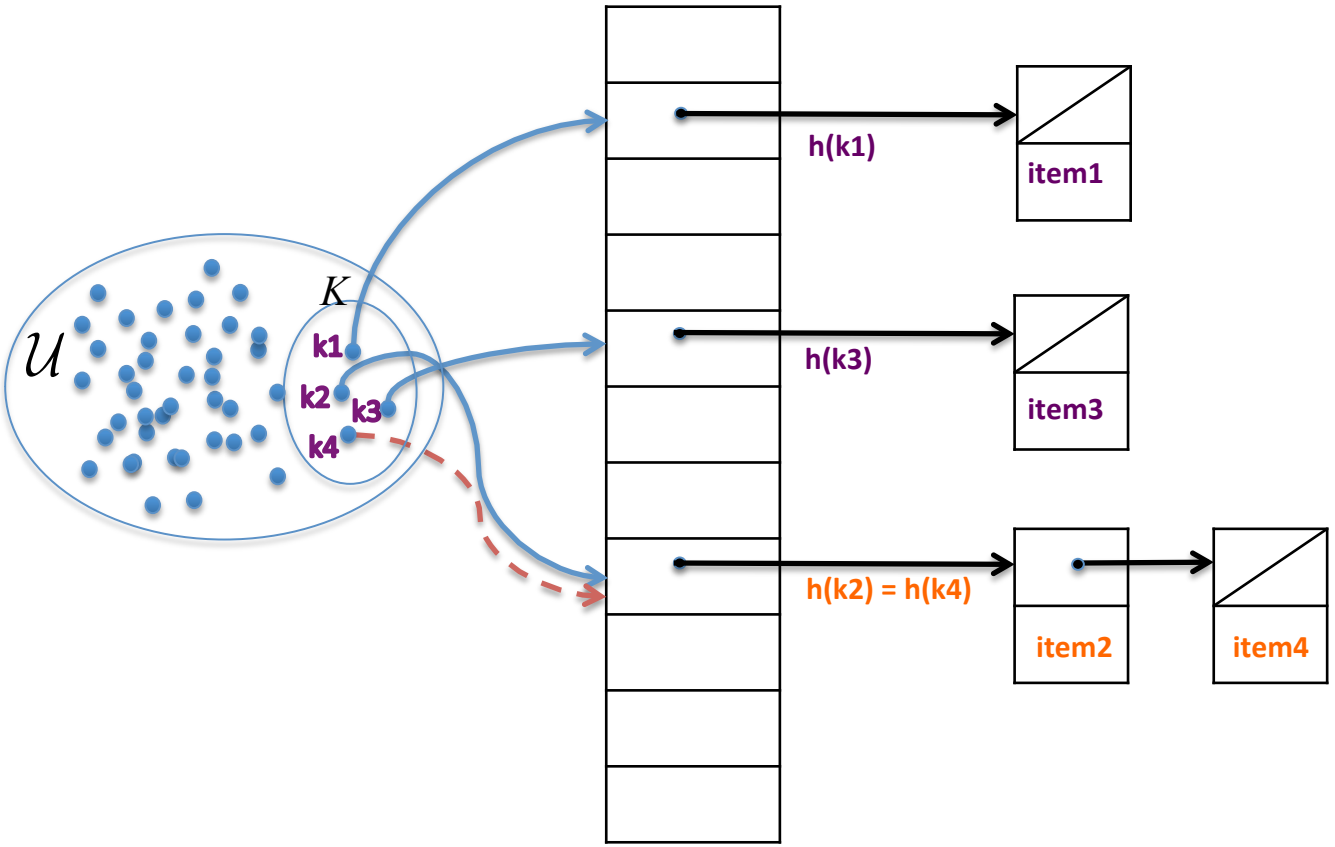
On-Line Algorithms!

Ignorance vs. Omniscience

Idea 2

OPEN ADDRESSING

Recall Chaining…



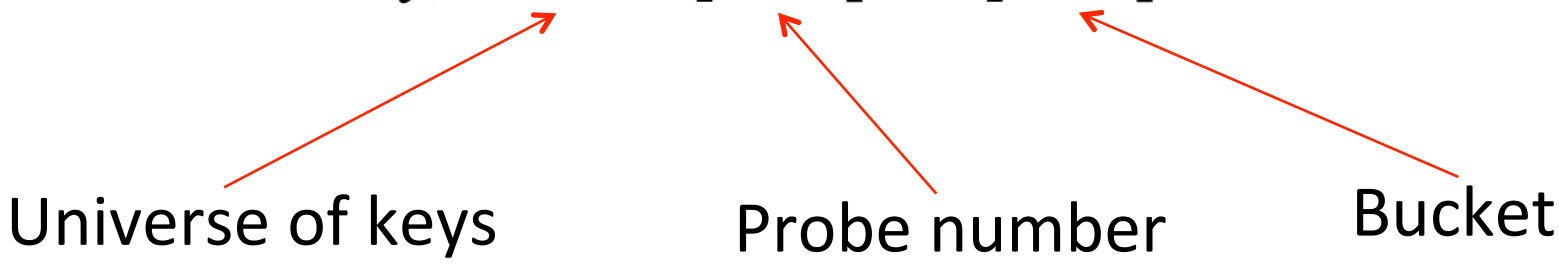
$\mathcal{U}$: *universe of all possible keys-huge set*

K : *actual keys-small set, but not known when designing data structure*

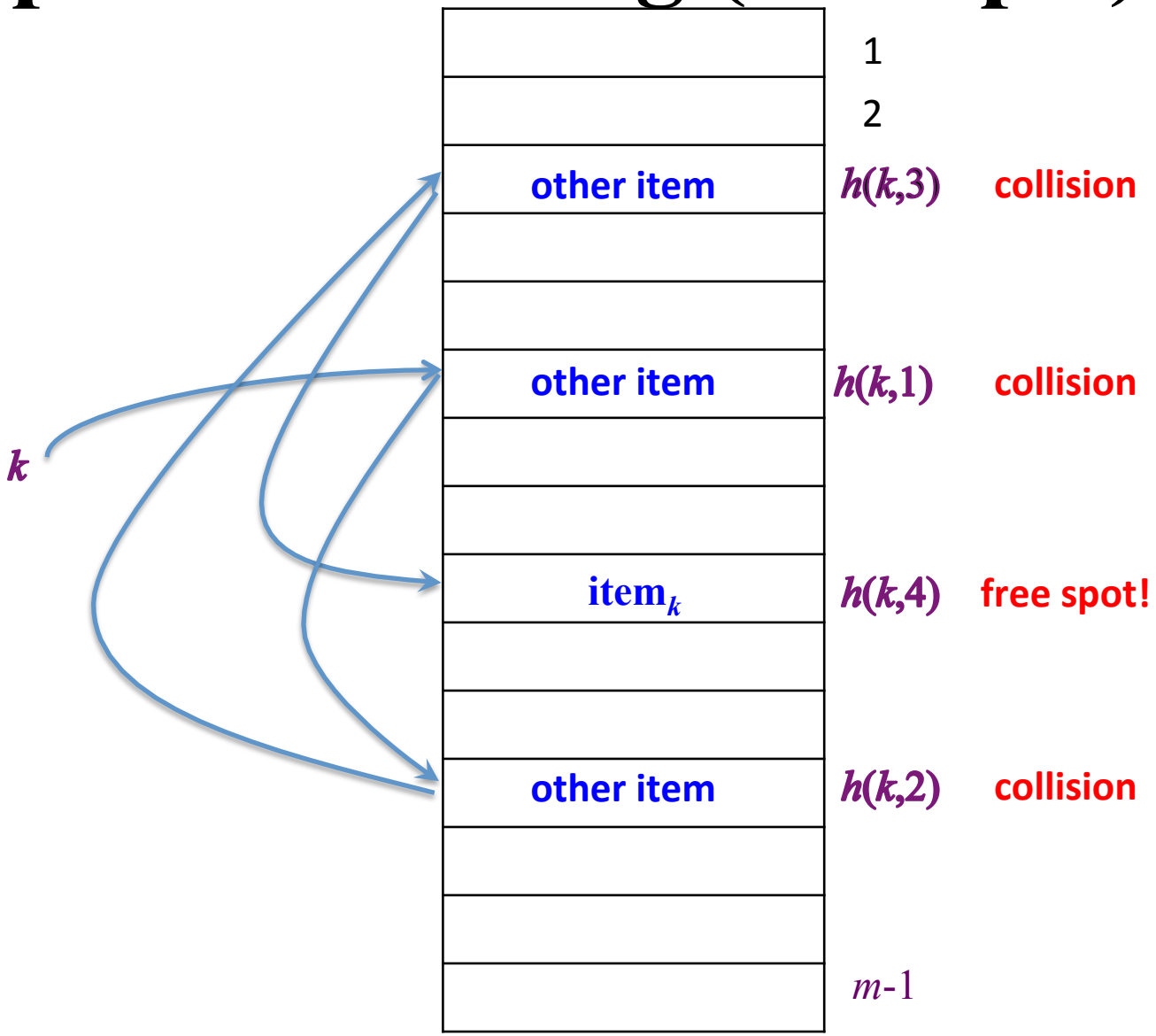
Different technique for dealing with collisions:

No linked lists: if bucket occupied, find other bucket (need $m \geq n$)

- For insert: probe a sequence of buckets until find empty one!
- *h* specifies probe sequence for key *x*
  - Ideally, h(x) sequence "visits all buckets"
  - Technically, $h: U \times [1..m] \rightarrow [1..m]$



Open Addressing (example)



Operations

Insert:

- Probe till find empty bucket, put item there

Search:

- Probe till find item (return with success)
- Or find empty bucket (return with failure)
  - Because if item inserted, would use that empty bucket

Delete:

- Probe till find item
- Remove, ~~leaving empty~~ bucket

Problem with Deletion

Consider the following sequence:

- Insert x
- Insert y
  - suppose probe sequence for y passes x bucket
  - store y elsewhere
- Delete x (leaving hole)
- Search for y
  - Probe sequence hits x bucket
  - Bucket now empty
  - Conclude y not in table (else y would be there)

Solution for deletion

- When delete x
  - Leave it in bucket, but mark it deleted

- Future search for x sees x is deleted
  - Returns "x not found"

- "Insert z" probes may hit x bucket
  - Since x is deleted, overwrite with z

  (So keeping deleted items doesn't waste space)

What probe sequence?

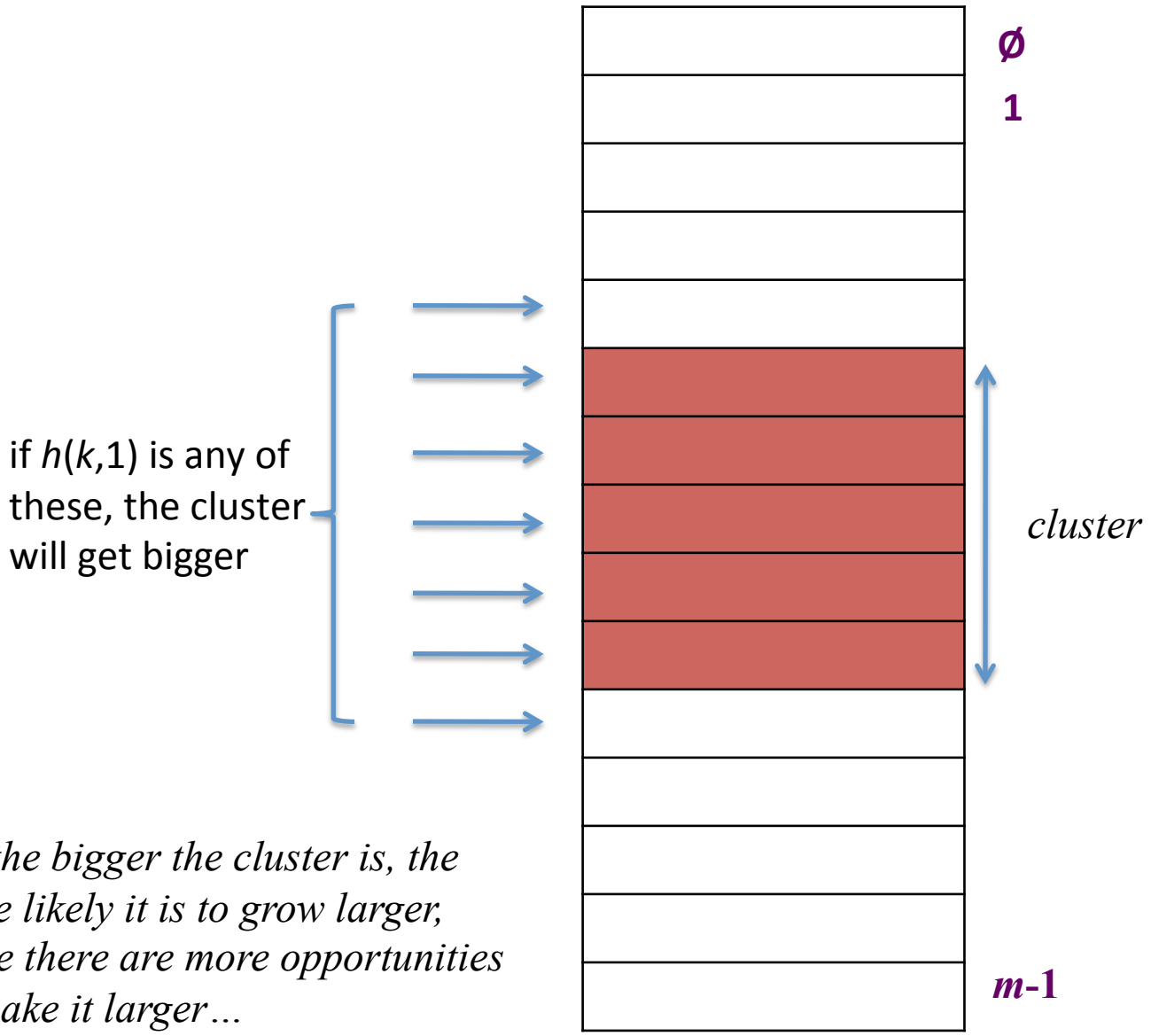
Linear probing

- $h(k,i) \triangleq h'(k) + i$ for ordinary hash h'

Problem: creates "clusters",

i.e. sequences of full buckets

  - exactly like parking
  - Big clusters are hit by lots of new items
  - They get put at end of cluster
  - Big cluster gets bigger: "rich get richer" phenomenon



i.e. the bigger the cluster is, the more likely it is to grow larger, since there are more opportunities to make it larger…

- E.g., $0.1 < \alpha < 0.99$, cluster size $\Theta(\log n)$
- Wrecks our constant-time operations

Double Hashing

- Two ordinary hash functions f(k), g(k)
- Probe sequence $h(k,i) \triangleq f(k) + i \cdot g(k) \bmod m$
- If g(k) always relatively prime to m, E.g., $m=2^r$ g(k) odd
  Then probe sequence for k can hit all buckets

Proof: The same bucket is hit twice if for some i,j:

$f(k) + i \cdot g(k) = f(k) + j \cdot g(k) \bmod m$

$\rightarrow i \cdot g(k) = j \cdot g(k) \pmod m$

$\rightarrow (i-j) \cdot g(k) = 0 \pmod m$

→ m and g(k) not relatively prime

(otherwise m should divide i-j, which is not possible for i, j<m)

Performance of Open Addressing

- Operation time is length of probe sequence
- How long is it?
- In general, hard to answer.
- If h(k,i) **as before**, then we "can" make the

  Uniform Hashing Assumption (UHA):

  - Probe sequence= h(k,1) h(k,2) … h(k,m) is a uniform random permutation of [1..m]

Note: this is different to the simple uniform hashing assumption (SUHA))

Analysis under UHA

Suppose:

- a size-m table contains n items
- we are using open addressing
- we are about to insert new item

Q: Probability **first** prob successful?

$$Prob\left(\frac{empty\ buckets}{all\ buckets}\right) = \frac{m - n}{m} \triangleq p$$

Why? From UHA, probe sequence random permutation
Hence, first position probed randomly
m-n out of the m slots are unoccupied

Analysis (II)

Q: If first probe unsuccessful, probability second prob successful?

$$\frac{m-n}{m-1} \geq \frac{m-n}{m} = p$$

Why?

- From UHA, probe sequence random permutation
- Hence, first probed slot is random; the second probed slot is random among the remaining slots, etc.
- Since first probe unsuccessful, it probed an occupied slot
- Hence, the second probe is choosing uniformly from m-1 slots, among which m-n are still clean

Analysis (III)

- If first two probes unsuccessful, probability third prob successful?

$$\frac{m-n}{m-2} \geq \frac{m-n}{m} = p$$

- …

➔ every trial succeeds with probability ≥p

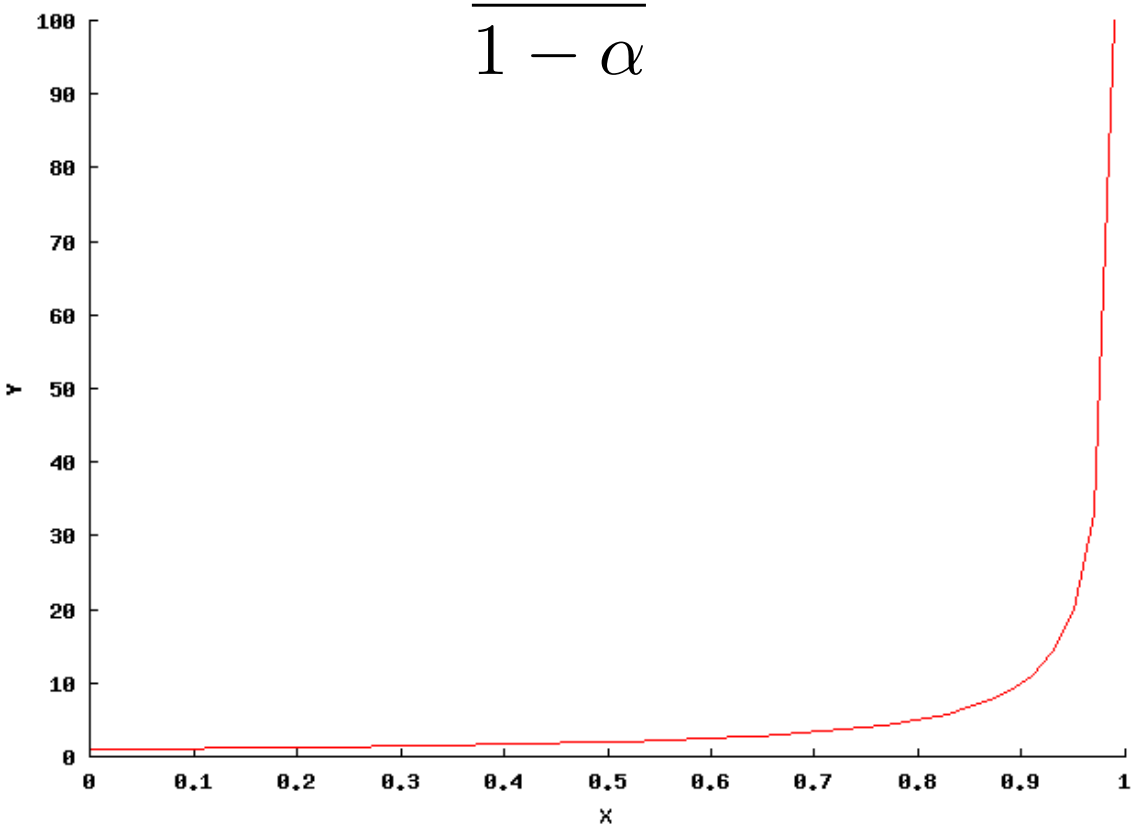
expected number of probes till success? $\leq \frac{1}{p} = \frac{1}{1-\alpha}$

e.g. if α=90%, expected number of probes is at most 10

Open Addressing vs. Chaining

- Open addressing skips linked lists
  - Saves space (of list pointers)
  - Better locality of reference
    - Array concentrated in m space
    - So fewer main-memory accesses bring it to cache
    - Linked list can wander all over memory
- Open addressing sensitive to load α
  - As $\alpha \rightarrow 1$, access time shoots up

$$\frac{1}{1-\alpha}$$



What IF?

ADVANCED HASHING ?

covered in recitation (for those who care)

Idea 3

VIA UNIVERSAL HASHING

Goal

- **Get rid of simple uniform hashing assumption**

- Create a family of hash functions H
- When you start, pick at random $h \in H$
- Unless you are unlucky, few collisions
  ~Adversary doesn't know what hash you will use
  So cannot pick keys that collide too much

DEF: Universal Hash Family

- *..is a family (set) of hash functions such that, for any keys x and y, if you choose a random h from the family,*

$$Pr[h(x) = h(y)] = 1/m$$

Thm: UHF produces few expected collisions

Proof:

$$\begin{aligned} E[collisions\ with\ x] &= E[number\ of\ y\ s.t.\ h(x) = h(y)] \\ &= E\left[\sum_y 1_{h(x)=h(y)}\right] \\ &= \sum_y E\left[1_{h(x)=h(y)}\right] (linearity\ of\ E) \\ &= \sum_y \Pr[\ h(x) = h(y)] \\ &= n/m \end{aligned}$$

THM: ∃Universal Hashing Families!

Proof:

- Suppose table size= p prime
- Define $h_{ab}(x) = a \cdot x + b \ (mod\ p)$
- If a and b are random elements in $\{0, \ldots, p-1\}$, then $h_{ab}(x)$ is a UHF
- $mod\ p$ is a field, so you can divide/substract as well
- Pick two keys x and y. What is the probability (over the choice of a, b) that the hashes of x and y collide?
- Must be $a \cdot x + b = q \ (mod\ p)$ and $a \cdot y + b = q \ (mod\ p)$, for some q in $\{0, \ldots, p-1\}$
- For fixed q, this is a linear system in a, b
- Two variables, two equations, Unique solution: that is,

  unique h_{ab} makes this happen
- Probability of choosing this h_{ab} is $1/p^2$
- Collision if $h_{ab}(x) = h_{ab}(y) = q$ for some q
- There are p possible values for q, hence overall probability of collision= $p/p^2 = 1/p = 1/m$

Welcome to Probabilism!

Crucial because:

1. The Adversary wants to harm you

2. To harm you he must know what you'll be doing

3. He cannot know if **you yourself** do not know!

And

4. **SM**'s Law: All sufficient complex systems are adversarial!

Cryptography

Secret writing → *Adversarial Computation*

You pick h in a hash family H (but not which h you picked)

Adversary knows H (but not which $h \in H$ you picked!)

Adversary picks the sequence of keys you must hash

Adversary learns when he has caused a **collision**

Adversary learns the **values** $h(k_1)$, $h(k_2)$, … , $h(k_i)$

Adversary can choose $h(k_{i+1})$ **adaptively!**

And yet…

"Cryptographers never sleep"

SM

Happy 6:006 ⇒ Happy 6.875!

Credits

Teenagegirlsvslife.blogspot.com

Goldenstateofmind.com

SMgraphics.home

Vote!

Next Week: Sorting

Better? Perfect Hashing!

- Hash table with zero collisions
- So don’t need linked lists
- Can’t guarantee for arbitrary keys
- But if you know keys in advance, can quickly find a hash function that works
  - E.g. for a fixed dictionary

Summary

- Hashing maps a large universe to a small range
- But avoids collisions
- Result:
  - Fast dictionary data structure
  - Fingerprints to save comparison time

- Next week: sorting

NOT COVERED IN CLASS

Fingerprinting

- File backup service
  - Major cost in time and money: bandwidth
- How decide whether a file has changed?
  - And thus needs new backup
- Send whole file?
  - Too expensive
- Send hash of file (treating file as big number)
  - Only send file if hash differs
  - Might make a mistake, if hash same

What signature?

- File x and backup y, length n bits
- Treat as n-bit numbers
- Pick random prime number p in [2..n]
- Hash/compare x (mod p) vs. y (mod p)
  - Send log n bits
- False negative if
  - x and y different
  - but x (mod p) = y (mod p)
  - i.e. (x-y) (mod p) = 0
  - i.e. p is a factor of x-y

What are the odds?

- How many prime factors does x-y have?
  - It's an n-bit number
  - It's the produce of its factors p_1 .. p_k
  - Each $p_i \geq 2$
  - So $(x-y) = p_1 p_2 .. p_k \geq 2^k$
  - So $k \leq \log_2 n$ prime factors
- How many primes in range [1..n] ?
  - Prime number theorem says about n/ln n
  - So, Pr[pick wrong factor] = (log n)/(n/ log n)
  - For better safety, pick bigger prime

Randomized Algorithms

- Hashing/Fingerprinting make random choices
- Then you prove they probably work
- Prevent adversary from giving you a bad input
- Lot of applications in algorithms design
  - Take 6.856 some day

Another Approach

- Algorithm
  - Keep m a power of 2 (for faster computation)
  - Grow (double m) when $n \geq m$
  - Shrink (halve m) when $n \leq m/4$
- Analysis
  - Just after rebuild: $n=m/2$
  - Next rebuild a grow → at least m/2 more inserts
    - Amortized cost $O(2m / (m/2)) = O(1)$
  - Next rebuild a shrink → at least m/4 more deletes
    - Amortized cost $O(m/2 / (m/4)) = O(1)$