GREAT GALLOPING CUCKOOS

ALGORITHMS FASTER THAN $log(n)$

CODEMASH V2.0.1.6 – 8 JANUARY 2016

MATT WILLIAMS

matt@matthewkwilliams.com

@matt_k_williams

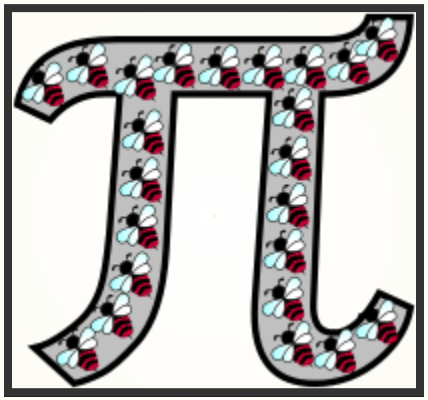
http://matthewkwilliams.com
http://20slides.com

http://github.com/aetherical

Note

The lovely picture is by my daughter Althea

BACKGROUND AKA WHY AM I UP HERE?



- Raspberry Pi Addict
- Search for "real" work for Pi's
- In the process I've discovered lots of new and nifty algorithms

DISCLAIMERS

1. I am not a Mathmetician
2. I am not a Chemist (this will make sense later)
3. I am not an expert
4. I have become passionate about this topic

GOOD ALGORITHMS MATTER

The real reason behind choosing good algorithms to solve a problem is that the good ones either break the problem into smaller problems or are myopic; focusing only on the steps needed to reach an answer. All else is superfluous.

NO SILVER BULLET

WHEN ALL YOU HAVE IS A HAMMER.....

REAL WORLD™ CONSIDERATIONS

Each algorithm has its "sweet spot", but the considerations tend to be among the following:

- Space
- Runtime
- Data Access

Know thy data – "Not Socrates"

Like sand through the hourglass, so is the data of our lives – Also "Not Socrates"

THE BIG O

- $O()$ refers to the complexity of an algorithm.
- You can also think of it in terms of the number of cycles/comparisons/steps to complete its task.

$$O(n)$$

LINEAR TIME

Size	Cycles
1	1
2	2
10	10
100	100
1000	1000

$$O(n^2)$$

QUADRATIC TIME

Size	Cycles
1	1
2	4
10	100
100	10000
1000	1000000

$$O(nlog(n))$$

Size	Cycles
1	0
2	1
10	23
100	460
1000	6907

$$O(log(n))$$

LOGARITHMIC TIME

Size	Cycles
1	0
2	1
10	3
100	7
1000	10

$$O(log(i))$$

LOGARITHMIC TIME

Size	Cycles
1	1
2	1
10	1
100	2
1000	3

$O(log(log(n)))$

"LOGALOG"

Size	Cycles
100000	2
1000000	2
10000000	2
100000000	2
1000000000	3

$$O(1)$$

CONSTANT TIME

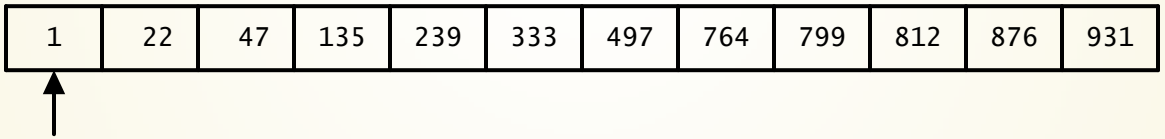
Size	Cycles
1	1
2	1
10	1
100	1
1000	1

SEARCHES

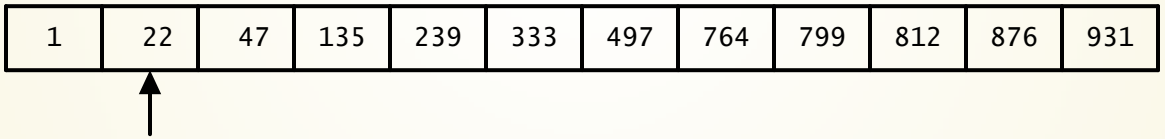
Looking for 812...

1	22	47	135	239	333	497	764	799	812	876	931

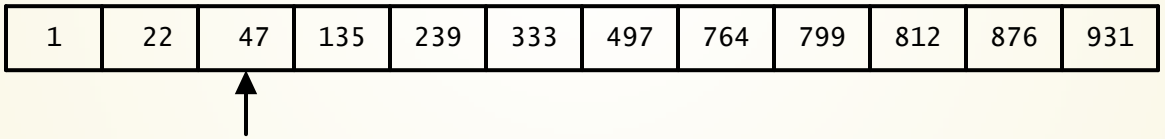
LINEAR



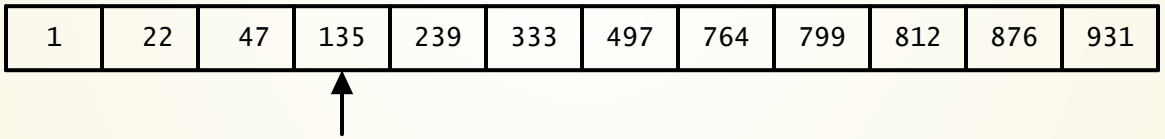
LINEAR



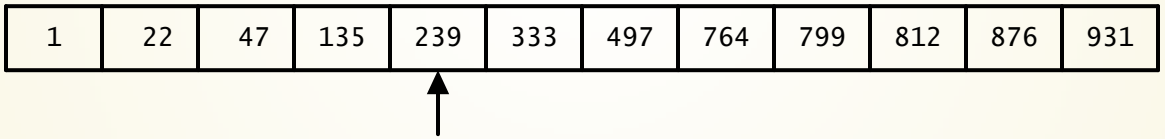
LINEAR



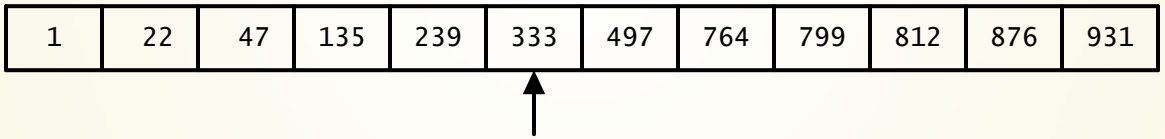
LINEAR



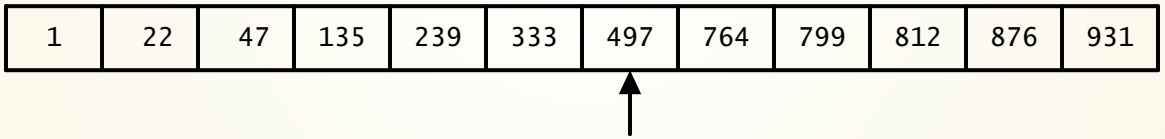
LINEAR



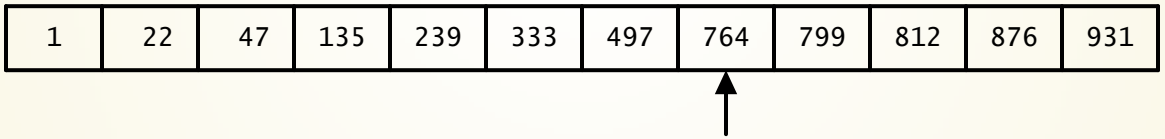
LINEAR



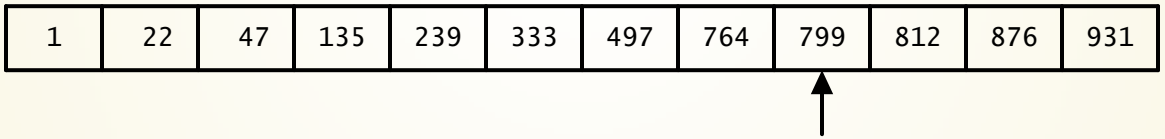
LINEAR



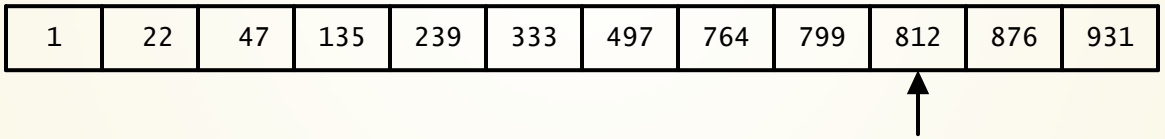
LINEAR



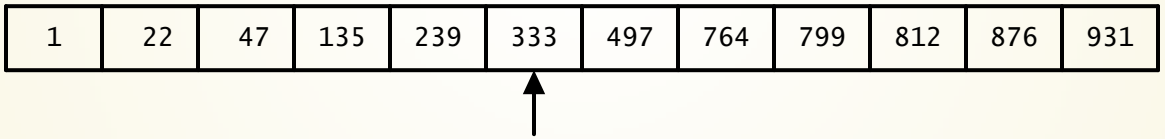
LINEAR



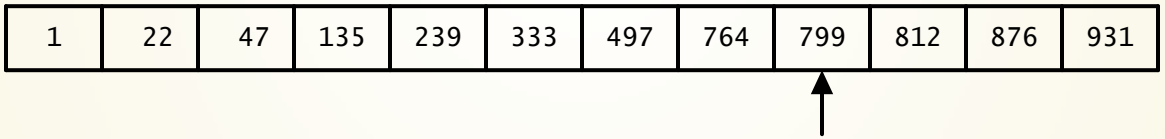
LINEAR



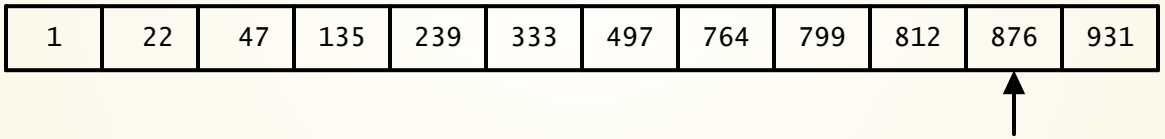
BINARY



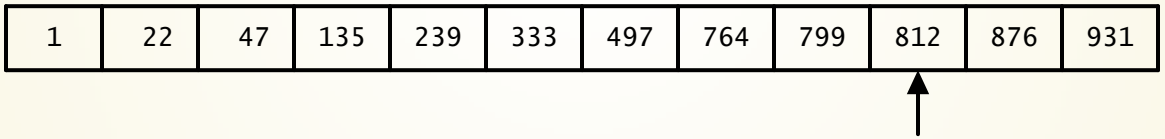
BINARY



BINARY



BINARY

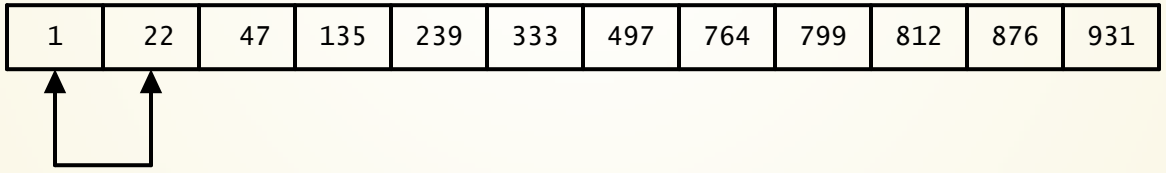


GALLOPING SEARCH

- First published in 1976!!!!
- Sliding Window
- Good for cases where
  - The target is at beginning or end
  - The bounds of the search space are not closed (like log file)
- Approaches $O(log(i))$

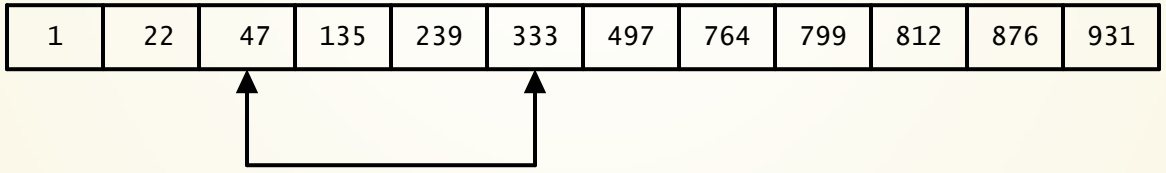
GALLOPING SEARCH

First window is 2^1 long.



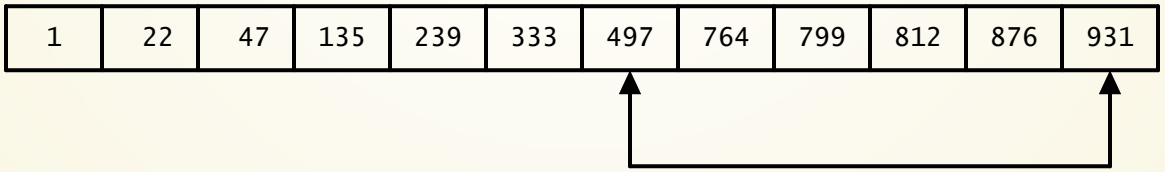
GALLOPING SEARCH

Second window is 2^2 long.



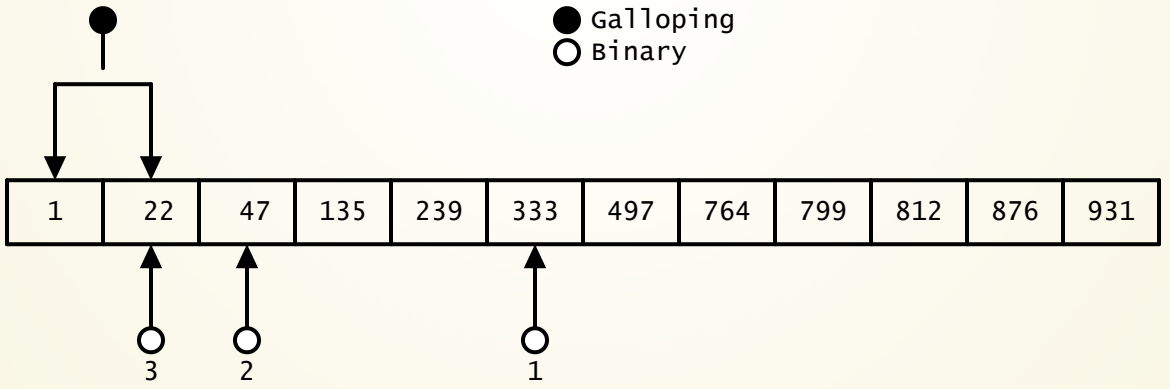
GALLOPING SEARCH

Third window is 2^3 (or to the end of the array) wide. When the proper window is found, then do a binary search on the window.



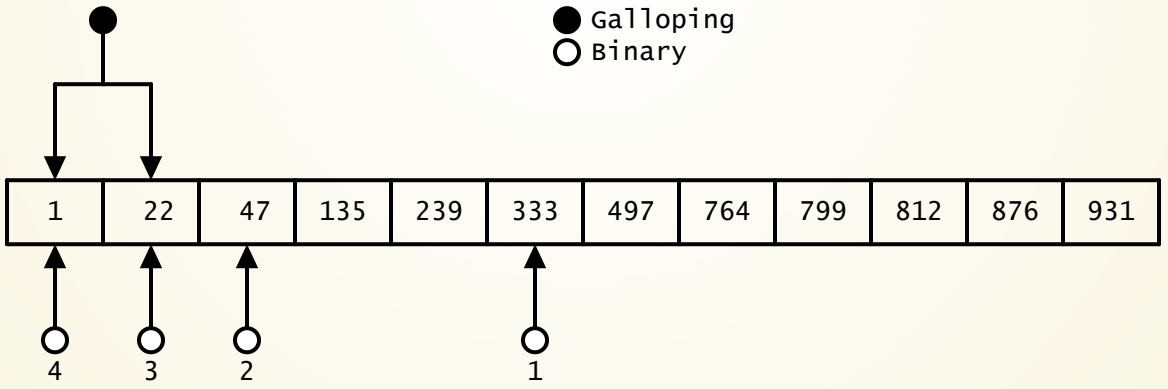
WHERE IT BEATS BINARY SEARCH

- If 22 is the target, binary search takes 3 checks. Galloping search finds it on a window boundary.



WHERE IT BEATS BINARY SEARCH

- If looking for 13, then binary search takes 4 checks to find it isn't a match. Galloping finds it isn't in the array immediately.



VARIANTS ON GALLOPING SEARCH

- Start at other end
- Trot – step grows linearly (1,2,3,...) so that the indexes are (1,3,6,10,15,...). Once the value is overshot, then hone in with binary search. $O(\sqrt{n})$

INTERPOLATION SEARCH

INTERPOLATION SEARCH

- Similar to how we look up a word in a dictionary.
  - Instead of splitting in $1/2$ each time, make a guess where it should be in range of values
- Assumes an uniform distribution of points
- Approaches $O(log(log(n)))$

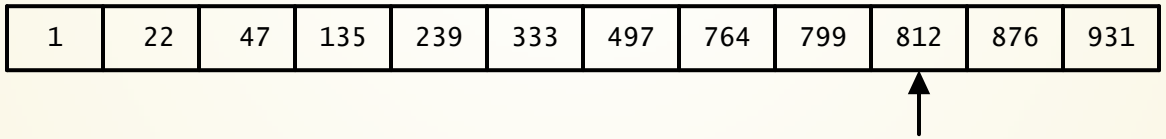
INTERPOLATION SEARCH

PREDICTED LOCATION

$$(seeking/max)/size$$

INTERPOLATION SEARCH

First check point: $(812/931) * 12 = 10$



SEARCH SUMMARY

Searching for 812

Algorithm	# of Iterations
Linear	10
Binary	4
Galloping	4
Interpolation	1

"What the hell is this monstrosity? And why the hell did he name it corned_beef?"

"Jerry says the name is probably some kind of rotten pun. What does it do?"

"Basically it takes the value of the characters of a demon's name, multiplies them by a number, adds another number and then divides the result by 65,353. Then it uses that result as a subscript in some kind of an array." She shook her head again. *"Why 65,353? Jesus! You know, if this guy doesn't come back we may never understand some of this stuff."* – *The Wizardry Compiled by Rick Cook*

Hash

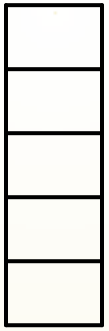
plural hashes

1. Food, especially meat and potatoes, chopped and mixed together.
2. A confused mess.
3. The # symbol (octothorpe, pound).
4. (computing) The result generated by a hash function.

HASH

- Used for sparsely populated universes
- Quick lookup of values
- Often implemented with a space the size of a (largeish) prime. 65353 is a large prime near the maximum size of a 16 bit unsigned Integer.

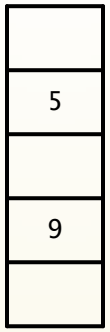
HASH



HASH

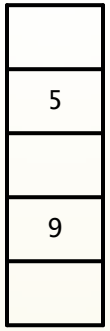
Hashing function maps entity to limited space in hash

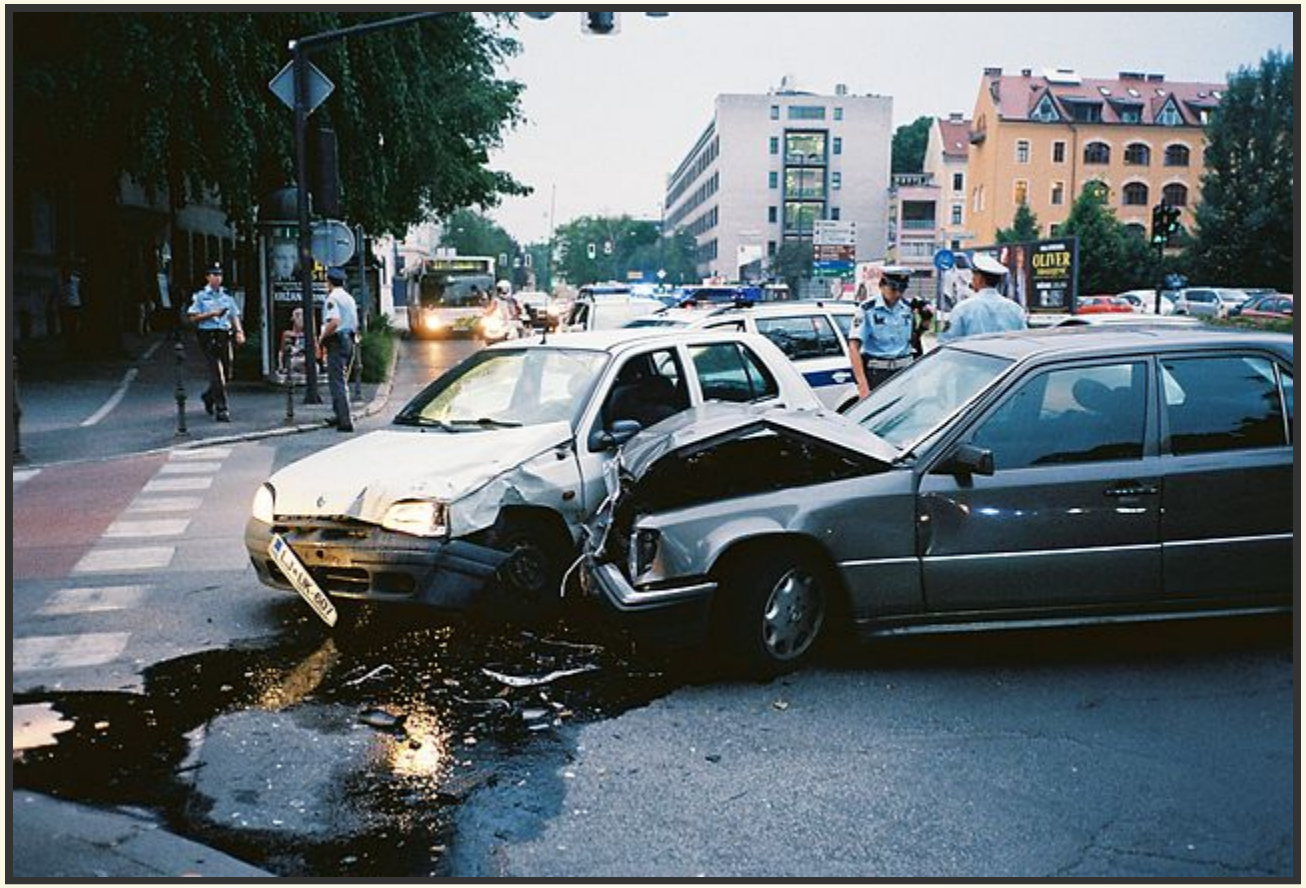
$$(CRC(entity) modulus ARRAY LENGTH)$$



HASH COLLISION

Try to insert 23; but it collides with 5.

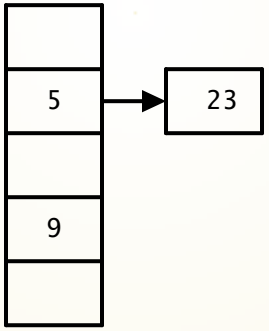




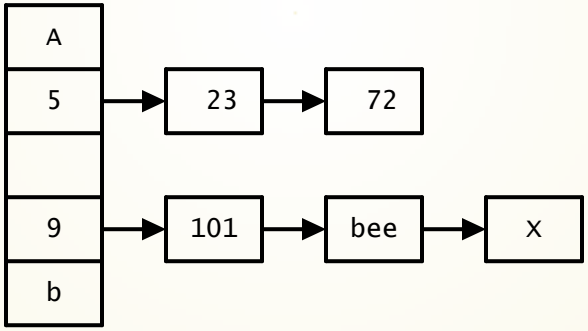
HANDLING COLLISIONS

- Rehash with larger hash table
- Buckets

HASH: BUCKET HASH



HASH: BUCKET HASH




MY BUKKIT IS FULL!
memegenerator.net

CUCKOO HASH

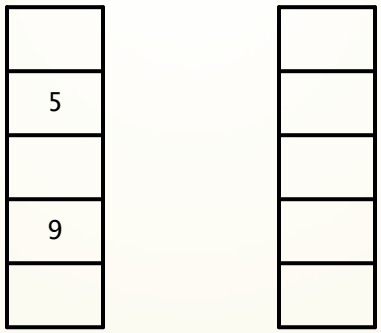
A common cuckoo chick pushes one of its host's eggs out of the nest. Detail of figure 1 from Anderson et al. (2009)

CUCKOO HASH

A common cuckoo chick pushes one of its host's eggs out of the nest. Detail of figure 1 from Anderson et al. (2009)

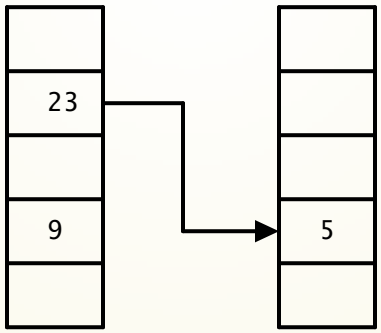
CUCKOO HASH

Trying to insert 23 again, it collides with 5

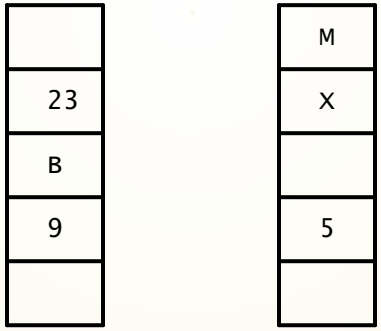


EVICT 5

Each hash has a separate hashing function

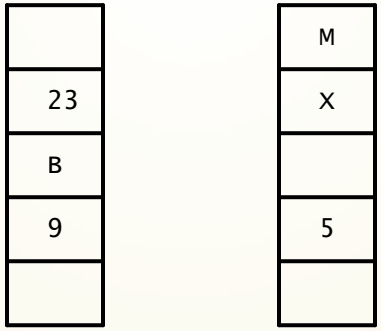


TIME PASSES

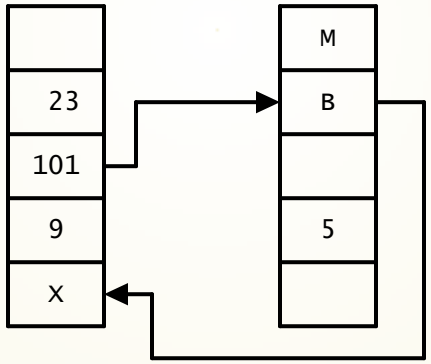


MULTIPLE EVICTIONS

Adding 101, collides with B & B collides with X



MULTIPLE EVICTIONS



ADVANTAGES OF CUCKOO HASH

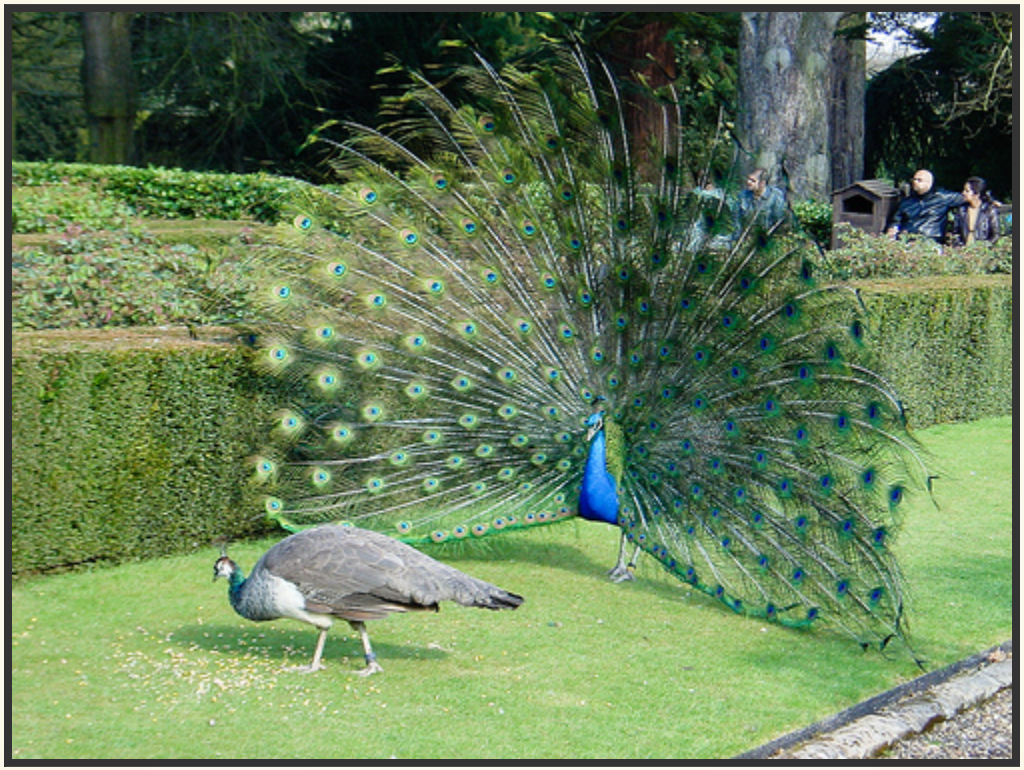
- Approaches $O(1)$ for inserts
- $O(1)$ for lookups

LIMITATIONS OF CUCKOO HASH

- Needs a certain amount of empty space; if not an eviction cycle can occur
- May need to resize & rehash to keep evictions from looping

Obi-Wan: "That Hash is our only hope."

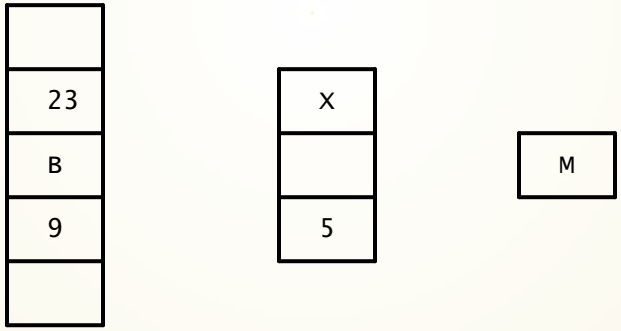
Yoda: "No.... there is another."



PEACOCK HASH

- Handles collisions by adding extra smaller hashes
- Originally devised for network routers to store IP/Mac Addresses
- Realtime, constant time

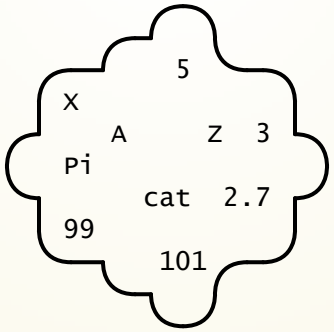
PEACOCK HASH



SETS

UNORDERED SET

- Can be implemented as a list or as a hash
- Inserts are cheap; retrievals may not be



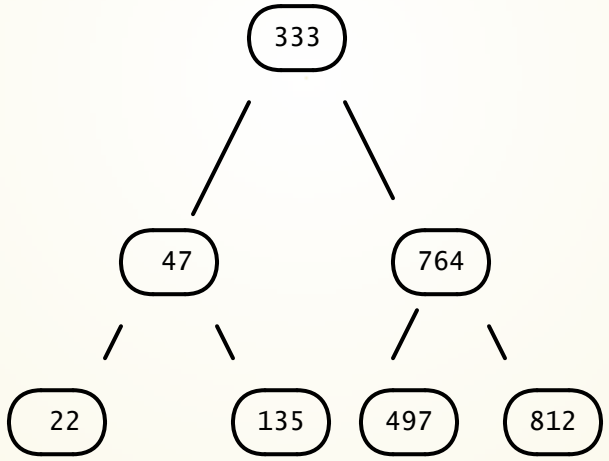

THERE CAN BE
ONLY ONE
PICTUREQUOTES.com

ORDERED SETS

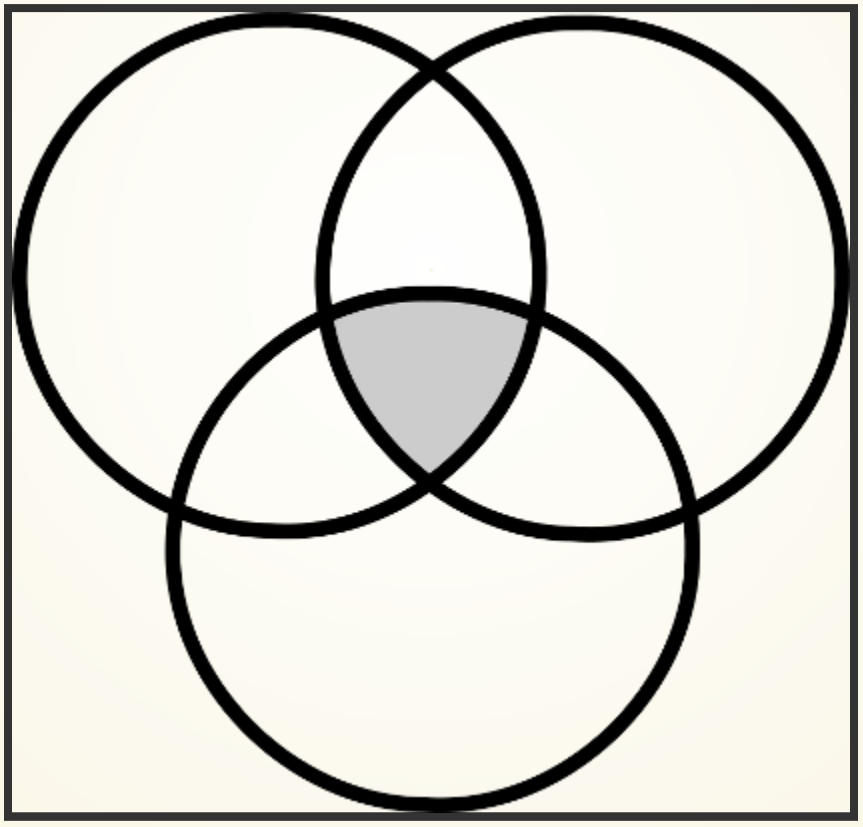
- Elements stored sorted; inserts can be expensive

1	22	47	135	239	333	497	764	799	812	876	931

STORED AS TREE



SET INTERSECTION



SET INTERSECTION

- Surveyed implementations are $O(n)$ for intersection
  - Ruby
  - Java
  - Go


THINK!
is there a better
way to do it ?
Plant Efficiency starts with your ability

GALLOPING INTERSECTION

COMPARISONS OF SET INTERSECTIONS

Excerpted from Faster Adaptive Set Intersections for Text Searching

Algorithm	# of comparisons
Sequential	119479075
Adaptive	83326341
Small Adaptive	68706234
Interpolation Sequential	55275738
Interpolation Adaptive	58558408

COMPARISONS OF SET INTERSECTIONS

Excerpted from Faster Adaptive Set Intersections for Text Searching

Algorithm	# of comparisons
Sequential	119479075
Interpolation Small Adaptive	44525318
Extrapolation Small Adaptive	50018852
Extrapolate Many Small Adaptive	44087712
Extrapolate Ahead Small Adaptive	43930174

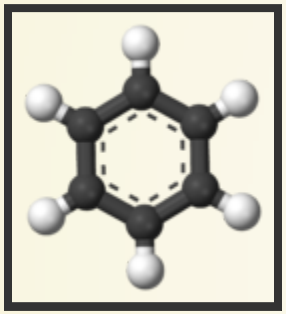
CASE STUDY

- Search for Real Work™ for my Pi Cluster
- Pi 2B
  - 4 Cores / 900 Mhz
  - 1G Ram
  - I/O on USB 2 Bus (480 MHz)

CASE STUDY

- Substructure Search
- PubChem from NIH
  - 150GB compressed download
- Uncompressed, 25K substances is 120MB; whole ~3TB
  - ~69 Million substances

SMILES



- Text Description of Chemical Structures
  - C1CCCCC1 is a benzene ring
- Typically Hydrogen is left out
- Can be more than one way to describe the same structure

NAIVE SOLUTION

SMILES is text. What do we use to search text?

Image by EpicBusinessman; from I drew this GREP thing and kinda forgot about it.

NAIVE SOLUTION

- 5 Pi 2B, searching a subset of data
  - ID, Smiles, Name
- Docker Swarm to distribute workload

NAIVE SOLUTION

Structure	Results	Time
C1CCCCC1=O	~2500	5s
C1CCCCC1O	~17500	15s

FINGERPRINTS

- Pubchem has a defined bitmap to describe a molecule
  - 880 bits wide
  - Has >2 Carbon, >4 Carbon, Boron, A ring

LIGHTBULB

- SMILES can be converted to Fingerprint with Chemistry Dev Kit
- If I create sets of substances which have a particular bit set, I can take the intersections of those sets to find the substances which match the fingerprint.

SETS

- 810 sets, one for each bit in the bitmap
- One file per bit in bitmap containing a list of substances which have the bit set.
- Using unsigned 32bit ints, the sets are anywhere from 4 bytes to 225MB in size.
  - Median 9.3M (2326026 substances)
  - Average 40M
- 33GB total

OPTIMIZATIONS

Memoization FTW!

If I've solved it once, why solve it again?

OPTIMIZATIONS

- Start from the smallest set and compare it with other sets
- In cases where the majority of the substances have this property, consider inverting the set – checking against the substances which don't have the property

IF IT'S A BITMAP... WHY NOT USE BITMAP TOOLS

- In order to fit the universe, 10000 x 10000 1 bit PNG
- 12K to 6.8M
  - Median 1.7M
  - Average 2.4M

BITMAPS

- ImageMagick on a 3.5 GHz box requires 10s/image intersectio
  core.
  - `convert FILE1 FILE2 -compose Darken -compos`
    - Using 5 cores can do 30/minute
    - Pre-Calculating all of the paths will take ~20 years.

DEMO

CREDITS

- Presentation created with Reveal.js
- Markdown and Diagram rendering via Markdeep

RESOURCES: GENERAL TOPICS

- Big O notation
- SIMD

RESOURCES: BITMAPS

- Better bitmap performance with Roaring bitmaps
- A General SIMD-based Approach to Accelerating Compression Algorithms
- Roaring Bitmap

RESOURCES: CUCKOOS

- Cuckoo Hashing Visualization
- Cuckoo Hashing (the original paper)
- cuckoo - GoDoc (go library)
- garethm/cuckoo (ruby)

RESOURCES: SEARCHES

- Binary search
- Linear search
- Prune and search

RESOURCES: GALLOPING SEARCH

- Beating the binary search algorithm – interpolation search, galloping search
- AN ALMOST OPTIMAL ALGORTTHM FOR UNBOUNDED SEARCHING

RESOURCES: INTERPOLATION

- Interpolation search

RESOURCES: PEACOCK HASHES

- Approximately-Perfect Hashing: Improving Network Throughput through Efficient Off-chip Routing Table Lookup
- Peacock Hashing: Deterministic and Updatable Hashing for High Performance Networking

RESOURCES: SETS

- A Fast Set Intersection Algorithm for Sorted Sequences
- Experimental Analysis of a Fast Intersection Algorithm for Sorted Sequences
- Experimental Comparison of Set Intersection Algorithms for Inverted Indexing
- Fast Set Intersection in Memory
- Faster Adaptive Set Intersections for Text Searching
- Faster Set Intersection with SIMD instructions by Reducing Branch Mispredictions
- SIMD Compression and the Intersection of Sorted Integers

RESOURCES: CHEMINFORMATICS

- The PubChem Project
- SDF Toolkit (perl)
- Chemistry Development Kit (java)
- Simplified molecular-input line-entry system (SMILES)

RESOURCES: RASPBERRY PI

- http://raspberrypi.org
- http://blog.hypriot.org
- http://matthewkwilliams.com

THANK YOU