CSE 100: HASH TABLES

Announcements

- Midterm 1 is graded
- HW2
  - Homework 2 covers Tries and Tree Search algorithms.
  - Deadline: 10/30 (Tuesday) @ 11:59PM
- PA2 is released
  - Checkpoint Deadline: 11:59pm on Tuesday, 11/6 (not slip day eligible)
  - Final Deadline: 11:59pm on Tuesday, 11/13 (slip day eligible)

Resolving Collisions: Double hashing

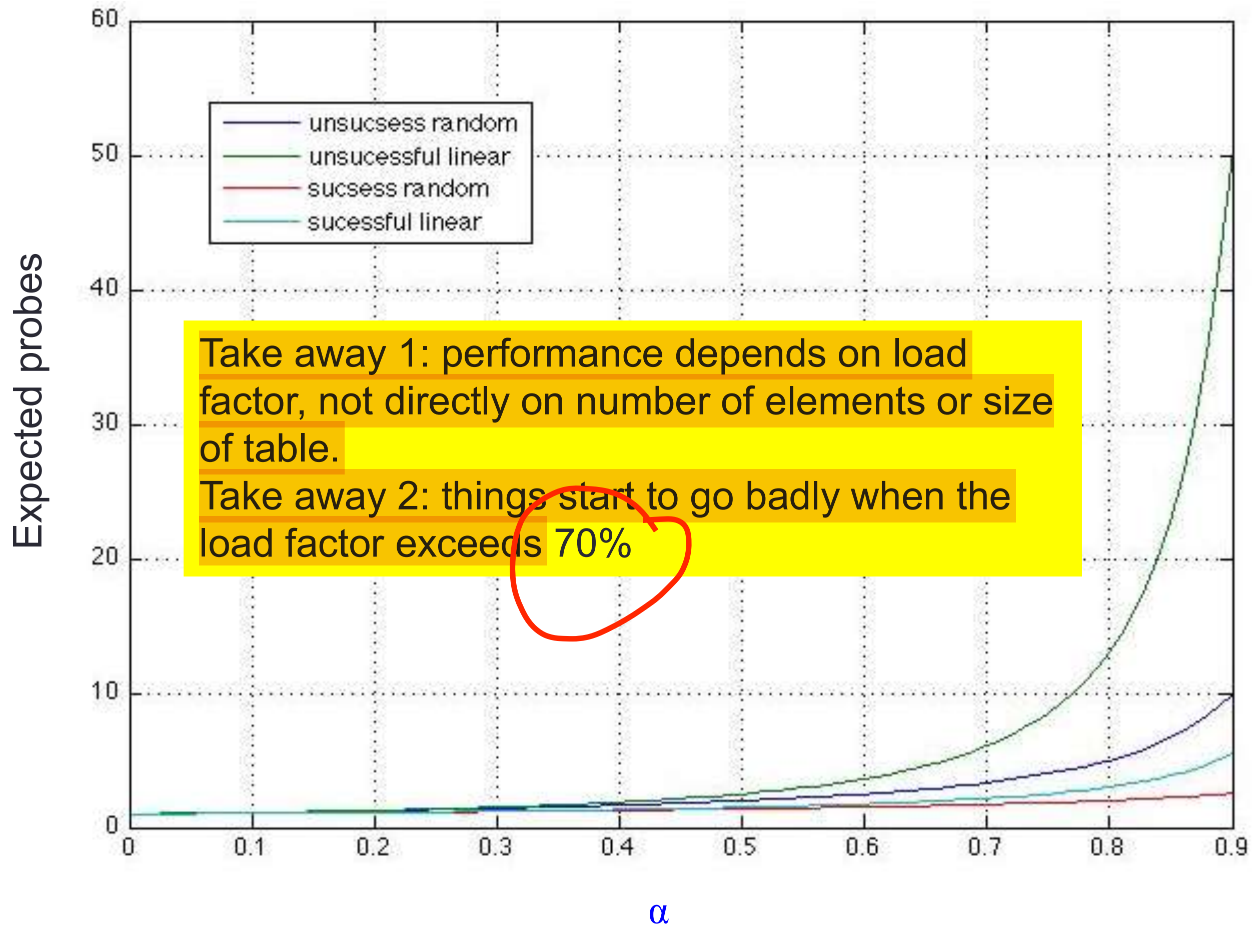
- A sequence of possible positions to insert an element are produced using two hash functions
- $h_1(x)$: to determine the position to insert in the array, $h_2(x)$: the offset from that position

701 (1,4), 145 (5,5), 218 (1,2), 12 (5,3), 750 (1,5)

in this table:

		701				145	
index:	0	1	2	3	4	5	6

Dependence of average performance on load



Take away 1: performance depends on load factor, not directly on number of elements or size of table.
Take away 2: things start to go badly when the load factor exceeds 70%

Average case costs with separate chaining

What you need to know:

- Separate chaining performance also depends only on load factor, and *not the number of elements* or size of table
- In practice it performs extremely well, even with relatively high loads

How big should a hash table be?

- The size of a hash table, M, is the number of buckets in the hash table. Let's say you want to store up to 100 keys in your hash table. Which of the following is the best choice of hash table size?

A. 100
B. 101
[✓ C. 151]
D. 200

Cuckoo Hashing: General Idea & Demo

- Use two (unrelated!) hash functions to hash into two different tables
- If you have a collision, the *new* item gets to stay and then other is kicked out (like Cuckoo bird chicks!)

http://www.lkozma.net/cuckoo_hashing_visualization/

Cuckoo Hashing: Running time

- What is the worst case for successful find in a hash table that uses Cuckoo hashing?

[✓ A. O(1)]
B. O(logN)
C. O(N)
D. More than O(N)

Is the bound the same for unsuccessful find?

Applications of Hashing - Bloom Filters

0	1	0	1	0	0	0	1	0	1	0	1

Provides one-sided guarantees:

- If an item is in the set and you query for membership, it will be correct.
- If an item is not in the set and you query for membership, it will usually be correct but could be wrong.

How does it work? Hashing.

Applications of Hashing - Bloom Filters

0	1	0	1	0	0	0	1	0	1	0	1

What if you want to do spell check, but have VERY limited space?
What if you want to transmit a set of addresses visited with very limited space?

In each case, there’s a key 1-sided assumption.

Bloom Filters

0	1	0	1	0	0	0	1	0	1	0	1

How does it work?

- Insert based on multiple hashes – mark all as 1.
- Search based on multiple hashes – if all are 1, found, else, not found.

Bloom Filter Example

Example:
H1(k) = k mod 7
H2(k) = k*3 % 7

Add the following Members in the set: 5(5, 1), 11(4, 5)

Find another element which would be considered part of the set which isn't. 12

0	0	0	0	0	0	0

Applications of Hashing – The Heavy Hitters Problem

1	2	1	1	1	2	4	1	4	2	4	1

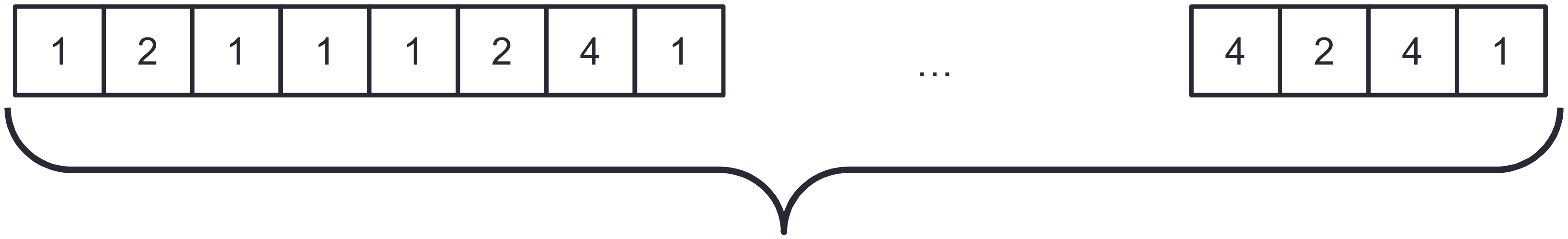
Given a set of elements of size *n,* and some (much) smaller value k determine which elements occur at least n/k times.

In the example above for:
n=12
k=3

Which elements should be returned?

A. {1}
B. {1, 4}
[✓ C. {1, 4, 2}]
D. {}

The Heavy Hitters Problem

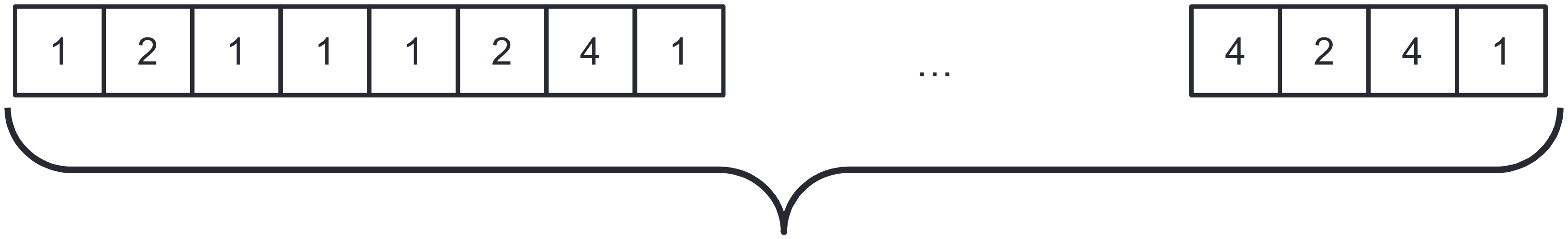


n is generally VERY LARGE (millions or billions)
You can't store it in memory, or even perhaps on disk!

Given a set of elements of size *n,* and some (much) smaller value k determine which elements occur at least n/k times.

What are the real-world applications of this problem?

The Heavy Hitters Problem: A sad fact



n is generally VERY LARGE (millions or billions)
You can't store it in memory, or even perhaps on disk!

Given a set of elements of size *n*, and some (much) smaller value k determine which elements occur at least n/k times.

There is no single pass algorithm that can solve this problem in a sublinear amount of auxiliary space!

Building Toward Count-Min Sketches: Sacrificing Exact Solutions for Large Savings!

Given a set of elements of size *n,* and some (much) smaller value k determine which elements occur at least n/k times.

1	35	10	1	1	10	1	35	1	35	16	1
1	3	2	1	1	2	1	3	1	3	0	1

Example:
n=12
k=2

Idea: Use a small Hash Map (of size b << n) to store counts

Example: b = 4, H(k) = k mod 4

1	6	2	3

What's the problem here?

Building Toward Count-Min Sketches: Sacrificing Exact Solutions for Large Savings!

Given a set of elements of size *n,* and some (much) smaller value k determine which elements occur at least n/k times.

	1	35	10	17	9	14	1	35	17	35	16	9
H(k) = k mod 4:	1	3	2	1	1	2	1	3	1	3	0	1

Example:
n=12
k=2

Idea: Use a small Hash Map (of size b << n) to store counts

Example: b = 4, H(k) = k mod 4

1	6	2	3

Any ideas on how to improve this?

Count-Min Sketch:
Sacrificing Exact Solutions for Large Savings!

Given a set of elements of size *n,* and some (much) smaller value k determine which elements occur at least n/k times.

1	35	10	17	9	14	1	35	17	35	16	9

Example:
n=12
k=2

Idea: Use ℓ small Hash Maps (of size b << n) to store counts

Example: ℓ=3, b = 4

	0	1	2	3
$H_0(k) = (k\%5)\ \%\ 4$				
$H_1(k) = (k\%7)\ \%\ 4$				
$H_2(k) = (k\%13)\ \%\ 4$				

Count in each hash function, advance slides to get to the example

Count-Min Sketch: Sacrificing Exact Solutions for Large Savings!

Idea: Use ℓ small Hash Maps (of size b much less than n) to store counts

Example:
n=12
k=2

Given a set of elements of size *n*, and some (much) smaller value k determine which elements occur at least n/k times.

	1	35	10	17	9	14	1	35	17	35	16	9
$H_0(k) = (k\%5)\ \%\ 4$												
$H_1(k) = (k\%7)\ \%\ 4$												
$H_2(k) = (k\%13)\ \%\ 4$												

Have to compute hash for each number…

Example: ℓ=3, b = 4

	0	1	2	3
$H_0(k) = (k\%5)\ \%\ 4$				
$H_1(k) = (k\%7)\ \%\ 4$				
$H_2(k) = (k\%13)\ \%\ 4$				

Count-Min Sketch: Sacrificing Exact Solutions for Large Savings!

Idea: Use ℓ small Hash Maps (of size b much less than n) to store counts

Given a set of elements of size *n,* and some (much) smaller value k determine which elements occur at least n/k times.

Example:
n=12
k=2

	1	35	10	17	9	14	1	35	17	35	16	9
$H_0(k) = (k\%5)\ \%\ 4$	1	0	0	2	0	0	1	0	2	0	1	0
$H_1(k) = (k\%7)\ \%\ 4$	1	0	3	3	2	0	1	0	3	0	2	2
$H_2(k) = (k\%13)\ \%\ 4$	1	1	2	0	1	1	1	1	0	1	3	1

Start tallying and stop ~1/2 done

Example: ℓ=3, b = 4

$H_0(k) = (k\%5)\ \%\ 4$				
$H_1(k) = (k\%7)\ \%\ 4$				
$H_2(k) = (k\%13)\ \%\ 4$				
	0	1	2	3

Count-Min Sketch: Sacrificing Exact Solutions for Large Savings!

Idea: Use ℓ small Hash Maps (of size b much less than n) to store counts

Given a set of elements of size *n*, and some (much) smaller value k determine which elements occur at least n/k times.

Example:
n=12
k=2

	1	35	10	17	9	14	1	35	17	35	16	9
$H_0(k) = (k\%5)\ \%\ 4$	1	0	0	2	0	0	1	0	2	0	1	0
$H_1(k) = (k\%7)\ \%\ 4$	1	0	3	3	2	0	1	0	3	0	2	2
$H_2(k) = (k\%13)\ \%\ 4$	1	1	2	0	1	1	1	1	0	1	3	1

Example: ℓ=3, b = 4

$H_0(k) = (k\%5)\ \%\ 4$	7	3	2	0
$H_1(k) = (k\%7)\ \%\ 4$	4	2	3	3
$H_2(k) = (k\%13)\ \%\ 4$	2	8	1	1
	0	1	2	3

What is the estimate for 1?

A. 1
B. 2
[✓ C. 3]
D. 4
E. Other

Count-Min Sketch: Sacrificing Exact Solutions for Large Savings!

Idea: Use ℓ small Hash Maps (of size b much less than n) to store counts

Given a set of elements of size *n*, and some (much) smaller value k determine which elements occur at least n/k times.

Example:
n=12
k=2

	1	35	10	17	9	14	1	35	17	35	16	9
$H_0(k) = (k\%5)\ \%\ 4$	1	0	0	2	0	0	1	0	2	0	1	0
$H_1(k) = (k\%7)\ \%\ 4$	1	0	3	3	2	0	1	0	3	0	2	2
$H_2(k) = (k\%13)\ \%\ 4$	1	1	2	0	1	1	1	1	0	1	3	1

Example: ℓ=3, b = 4

$H_0(k) = (k\%5)\ \%\ 4$	7	3	2	0
$H_1(k) = (k\%7)\ \%\ 4$	4	2	3	3
$H_2(k) = (k\%13)\ \%\ 4$	2	8	1	1
	0	1	2	3

Which of the following values are overestimated?

A. 10
B. 17
C. 9
D. 16
E. More than one of these

Which value is overestimated by the most?

Count-Min Sketch

- ℓ (number of hash functions) and b (size of hash tables) can be much smaller than n. Size of storage depends on k not n!
- Updating in the table is fast: O(1), assuming hash functions are fast
- There is a very high probability that the overestimate of an number will be "small"
- Count-Min Sketches can solve an approximation of the Heavy Hitters problem (think about how)