Sketches

02807 Computational Tools for Data Science

Today

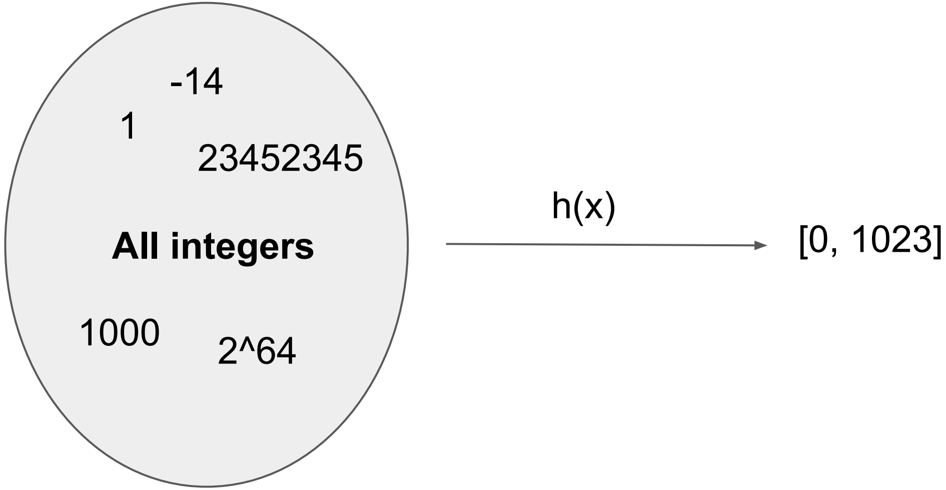
- Hash functions
- Sketches
  - The CountMin sketch
  - The HyperLogLog sketch

Hash functions

- A function h(x) that maps from some universe of values to a (smaller) domain of values

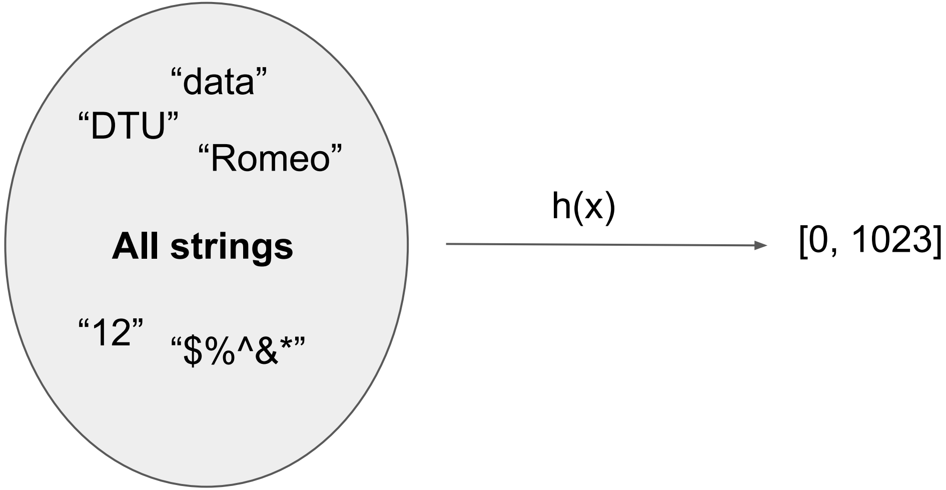
Hash functions

- A function h(x) that maps from some universe of keys to a (smaller) domain of values
- Example:



Hash functions

- A function h(x) that maps from some universe of keys to a (smaller) domain of values
- Example:

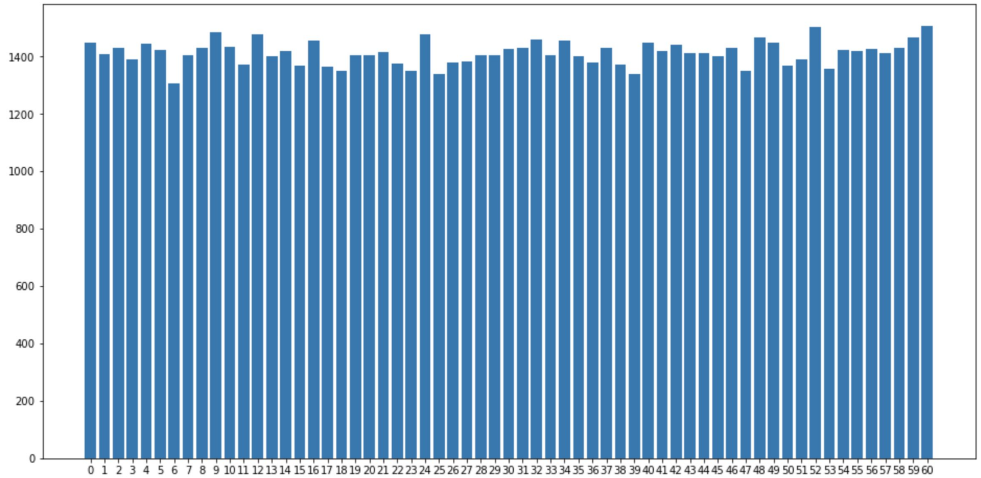


Hash functions: collisions

- Going from a big domain to a smaller domain means that collisions may occur
- A collision: h(x) == h(y) for x != y
- For our purpose, the hash function should distribute the keys uniformly across the values

Hash functions: collisions

- Going from a big domain to a smaller domain means that collisions may occur
- A collision: h(x) == h(y) for x != y
- For our purpose, the hash function should distribute the keys uniformly across the values
- The words of Shakespeare:



Hash functions: example 1

- From integers to [0,..,m-1] (pick a and b at random and let p>m be a prime)

$$h(x) = (ax + b \bmod p) \bmod m$$

Hash functions: example 2

- Hashing strings: use the Murmur3 algorithm
- No theoretical guarantees, but values are uniformly distributed for many real world inputs and it's fast
- Python:

```
In [41]: import mmh3
         mmh3.hash('some string')

Out[41]: 307557468
```

```c
uint32_t murmur3_32(const uint8_t* key, size_t len, uint32_t seed)
{
    uint32_t h = seed;
    if (len > 3) {
        size_t i = len >> 2;
        do {
            uint32_t k;
            memcpy(&k, key, sizeof(uint32_t));
            key += sizeof(uint32_t);
            k *= 0xcc9e2d51;
            k = (k << 15) | (k >> 17);
            k *= 0x1b873593;
            h ^= k;
            h = (h << 13) | (h >> 19);
            h = h * 5 + 0xe6546b64;
        } while (--i);
    }
    if (len & 3) {
        size_t i = len & 3;
        uint32_t k = 0;
        do {
            k <<= 8;
            k |= key[i - 1];
        } while (--i);
        k *= 0xcc9e2d51;
        k = (k << 15) | (k >> 17);
        k *= 0x1b873593;
        h ^= k;
    }
    h ^= len;
    h ^= h >> 16;
    h *= 0x85ebca6b;
    h ^= h >> 13;
    h *= 0xc2b2ae35;
    h ^= h >> 16;
    return h;
}
```

Hash functions: families

- A "parametrized" hash function can be seen as a family (collection) of hash function
- For example,

$$h_{a,b}(x) = (ax + b \bmod p) \bmod m$$

Choosing other a's and b's gives other functions in the same family.

- In Murmur3, use seed:

```
In [14]: import mmh3
         mmh3.hash('some string', seed=42)

Out[14]: -110992965
```

Sketches

- A data structure that can be used to compute an estimate
- Compact sketch/summary of data
- Answers to queries are approximate
- Composable

Today

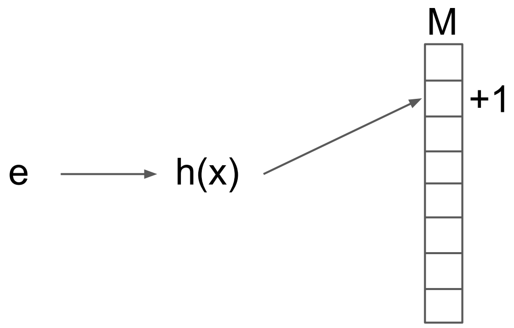
- CountMin
- HyperLogLog

CountMin Sketch

- Compute the frequency of elements in a stream without storing all elements
- Query: How many times did element *e* occur in the stream?
- Answer: $f_e \leq f'_e \leq f_e \cdot \epsilon N$ (with high probability, for the right choice of parameters)

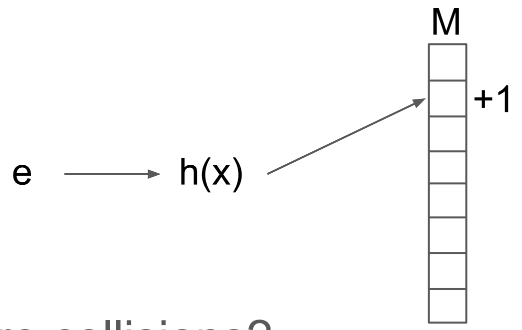
CountMin Sketch: simplified

- Choose a hash function h(x) -> [0,..,m-1]
- Initialize array M of m counters to 0
- When an element *e* arrives, increment M[h(e)] by 1
- To answer a query return M[h(e)]



CountMin Sketch: simplified

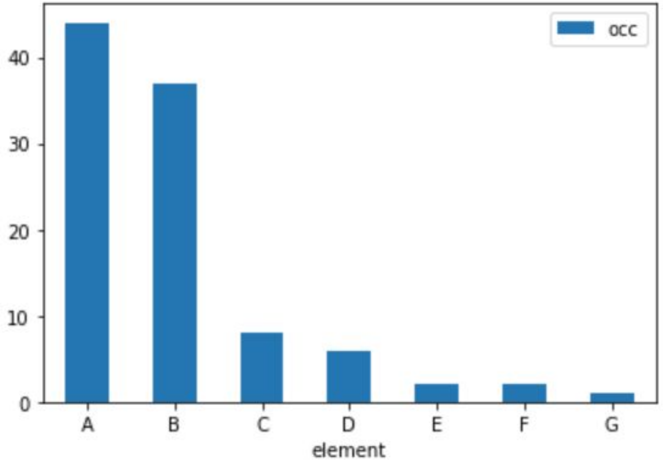
- Choose a hash function h(x) -> [0,..,m-1]
- Initialize array M of m counters to 0
- When an element *e* arrives, increment M[h(e)] by 1
- To answer a query return M[h(e)]



- What happens if there are collisions?

CountMin Sketch: simplified example

- Due to collisions, we always over count
    - If all elements are distinct (and the hash function is good), we overcount by ~N/m
- If the stream has elements occurring a lot more often than others, the error for frequent items is likely to be small



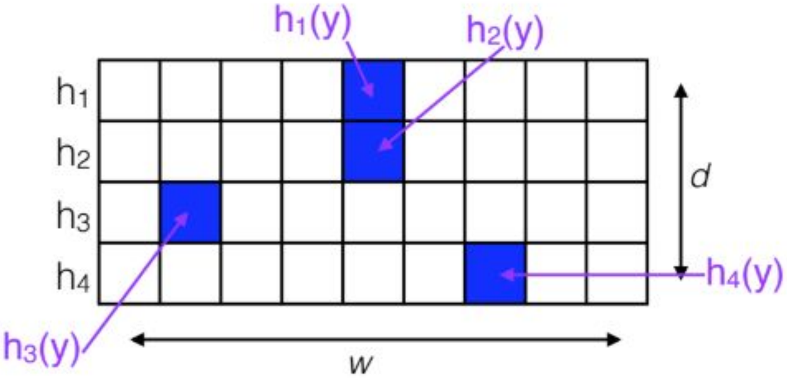
f'_A is close to the real value if h(A) != h(B)

but

f'_G is very far from the real value if e.g. h(A) = h(G)

CountMin Sketch: full algorithm

- Choose d hash functions $h_i(x)$ -> [0,..,w-1] (at random from same family)
- Initialize d times w array M to 0
- When an element *e* arrives, increment M[i, h(e)] by 1 (for i=0..d-1)
- To answer a query return $\min_{i=0..d-1}$(M[i, h(e)])



CountMin Sketch != Count Sketch

Warning: The literature contains a similar data structure called a count sketch. It is not the same as a CountMin sketch.

HyperLogLog Sketch

- Compute the number of distinct elements in a stream without storing all elements
- Query: How many distinct elements occured in the stream?
- Answer: $E^* = E \pm 1.04 \cdot E/\sqrt{m}$ (with high probability, for the right choice of parameters)

HyperLogLog Sketch: simplified

- Choose a hash function h(x) -> [0,..,m-1]
- Initialize one counter C = 0
- When an element *e* arrives:
  - Compute the position p of the leftmost 1-bit in h(e) (e.g. 000**1**010111001 yields 4)
  - Set M to max(C, p)
- To answer a query return 2^C

HyperLogLog Sketch: simplified example

- Suppose m = 16
- The probability that first 1-bit is in position 2 is 4/16 = ¼
- In expectation we need to see 4 distinct elements before C = 2
- When C = 2 then $2^C = 4$
- The expected value of C is E, but variance is big
    - First element hash to 0000 or 0001
    - First 8 distinct elements hash to 1XXX

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

HyperLogLog Sketch: full algorithm

- Choose a hash function h(x) -> [0,..,w-1] (w = 32 or 64)
- Initialize m counters M[1],...,M[m] to 0
- When an element *e* arrives:
  - Compute h(e)
  - Split the binary representation of h(e) into an upper and lower part (e.g. 0010100010100111l), so that the upper part has $\log_2(m)$ bits
  - Compute the position p of the leftmost 1-bit of the lower part
  - Let j be the integer representation of the upper part + 1
  - Set M[j] to max(M[j], p)
- To answer a query return m times the harmonic mean of the counters (adjusted by a constant to correct for some bias)

The full picture

- “An Improved Data Stream Summary: The Count-Min Sketch and its Applications”, G. Cormode & S. Muthukrishnan
- “HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm”, P. Flajolet, E. Fusy, O. Gandouet & F. Meunier
- Wikipedia and other sources are also good!