#18 – Finding Similar Items (4)

Hashing & Application Example

Hash functions are critical…

- Bloom filters use k (small) different functions
- Min-Hash uses hundreds
- LSH uses several

- In Movies application we apply to sets of numbers…
- In LSH we apply to fixed number of integers..
- When inserting strings in a Bloom Filter we apply to strings …

Desired properties

1. The keys are nicely spread out so that we do not have too many collisions
2. The function is fast to compute
   - shouldn't be too complicated, because that affects the overall runtime.

3. M = O(N):
   - in particular, we would like our scheme to achieve property (1) without needing the table size M to be much larger than the number of elements N

Universal Hashing

- Definition:

- A randomized algorithm H for constructing hash functions h : U → {1,...,M} is universal if for all pairs *x*, *y* in U, we have

$$\Pr_{h \leftarrow H}[h(x) = h(y)] \leq 1/M.$$

- U is the Universe

Universal Hashing

- We also say that a set H of hash functions is a **universal hash function family** if the procedure "choose h ∈ H at random" is universal.
  - Here we are identifying the set of functions with the uniform distribution over the set.

Constructing – Matrix method

- Assume keys are u-bits long
- We will pick h as a random b x u matrix with 0s and 1s
- and define h(x) = hx
  - using addition mod 2
- Example:
  - u = 4
  - b= 3

$$\begin{array}{ccc} h & x & h(x) \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = & \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \end{array}$$

What means multiply h by x ?

- Can be interpreted as adding some of the columns of h, indicated by where x has 1s

- In the example:

$$\begin{array}{cccc} h & x & & h(x) \\ \begin{bmatrix}1&0&0&0\\0&1&1&1\\1&1&1&0\end{bmatrix} & \begin{bmatrix}1\\0\\1\\0\end{bmatrix} & = & \begin{bmatrix}1\\1\\0\end{bmatrix}\end{array}$$

- 1st and 3rd column of h are added
  - 1 + 0 = 1
  - 0 + 1 = 1
  - 1 + 1 = 0

How / Why it works

- Assume M is power of 2: $M = 2^b$
- Claim:
  - for $x \neq y, P[h(x) = h(y)] = \frac{1}{M} = 1/2^b$

- "Proof":
- Take an arbitrary pair of keys x,y with $x \neq y$
  - They must differ someplace
  - Assume they differ in position i,
    - ex: $x_i = 0$ and $y_i = 1$

“Proof” (continuation)

- Imagine we first choose all of h but the ith column
- Over the remaining choices of ith column $h(x)$ is fixed

- However, each of the 2^b different settings of ith column gives a different value of $h(y)$

- So there is exactly a $\frac{1}{2^b}$ chance that $h(x) = h(y)$

Another method

- A more efficient method than matrix method
- Instead viewing the key as a vector of bits, we will view the key as a vector of integers
  - $[x_1, x_2, \ldots, x_k]$
  - With x_i in the range $\{0,1,\ldots,M-1\}$
  - And M a prime number
- Example: for strings of length k, x_i could be the ith character

- To select a hash function h we choose k random numbers $r_1, r_2, \ldots, r_k$ from $\{0,1,\ldots,M-1\}$
- and define:
  - $h(x) = r_1 x_1 + r_2 x_2 + \ldots + r_k\, x_k \;\; mod\ M$

Why it works?

- The proof follows the same lines as for the matrix method.

- See https://www.cs.cmu.edu/~avrim/451f11/lectures/lect1004.pdf

Algorithm

Step1 - ***condition***

Select m to be prime.

Step 2 - pre-processing

Decompose key k into r+1 digits: k = <$k_0, k_1, \dots, k_r$> where $k_i \in \{0, 1, \dots, m\text{-}1\}$
(equivalent to writing key k in base m)

Step 3 - ***the randomness***

Pick a = <$a_0, a_1, \dots, a_r$> at random. Each $a_i \in \{0, 1, \dots, m\text{-}1\}$

Step 4 - ***the hash function***

h_a (k) = $(\sum_{i=0}^{r} a_i\, k_i)$ mod m

Example of use

- IP addresses

Universe $\mathcal{U}$ is IP addresses. Each IP address is a 32-bit 4-tuple $<x_1, x_2, x_3, x_4>$ where $x_i \in \{0, ..., 255\}$.

Let m be a prime number (m=997 if we need to store 500 IPs for example).

Define h_a for the 4-tuple a = $<a_1, a_2, a_3, a_4>$ where $a_i \in \{0, ..., \text{m-1}\}$.

h_a : IP address → Slot

Q.) How do I evaluate which slot $<x_1, x_2, x_3, x_4>$ IP address hashes to using the above formulation of $\mathcal{H}$?

A.) Using the following equation which requires constant space and time:

$$h_a(x_1, x_2, x_3, x_4) = a_1 x_1 + a_2 x_2 + a_3 x_3 + a_4 x_4 \pmod{m}$$

cryptographic hash functions.

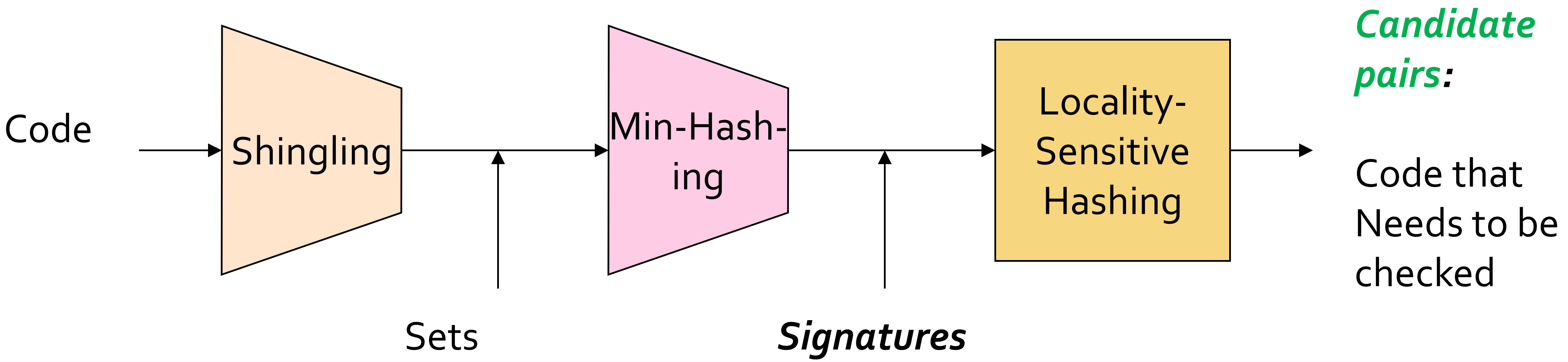
- For cryptographic hash functions M is as an exponentially large number, like 2^256.
- The table would be bigger than the # electrons in the universe!
- But we don’t have the table. Instead, h(x) is a "fingerprint" of x.

- Note that even for M very big (like 2^256) the indices h(x) are fairly small
  - e.g., only 256 bits (32 bytes)

- The main property you want for a cryptographic hash function is that given x, it should be computationally intractable for anyone to find $y \neq x$ such that $h(y) = h(x)$

Sources

- https://www.cs.cmu.edu/~avrim/451f11/lectures/lect1004.pdf
- https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/video-lectures/lecture-8-universal-hashing-perfect-hashing/lec8.pdf
- http://cs-www.bu.edu/faculty/homer/537/talks/SarahAdelBargal_UniversalHashingnotes.pdf



Example of Application

Detection of copies ….

Pipeline

- Get list of documents
  - How ?
- Shingles for each document
  - How ?
- MinHash for Sets of Shingles
- Candidate pair by LSH

demoPlágio