15-853:Algorithms in the Real World

Announcements:

- HW2 due tomorrow noon.
  - Small correction made in the BWT question.

- Naama's office hour cancelled. Francisco holding additional office hours instead.

15-853:Algorithms in the Real World

Announcements:

- Plan for the coming week:
  - I am away at ACM SOSP 2019
  - Graph compression guest lecture on Oct 29 by Laxman Dhulipala
  - Cryptography-1 guest lecture on Oct 31 by Francisco Maturana
- There will be a homework on Hashing + Cryptography modules by the end of first week of November

15-853:Algorithms in the Real World

Announcements:

Course project:

- 2-3 people teams
- 3 types of projects
  - Survey of a topic: At least 2 papers per team member (state-of-the-art papers; can include surveys)
  - Read papers (at least 3) + light weight “research-y” stuff (potentially implementation and comparison etc.)
  - Full fledged research: typically based on one paper and addressing a research question

15-853:Algorithms in the Real World

Announcements:

Course project:

- By Friday Nov 8 team and project plan (which papers, what question etc.) should be finalized
- Share through one Google doc per team
- Use the class email list:
  - 15853f19-students@lists.andrew.cmu.edu
  - with subject beginning “project-team-finding” to ping your classmates to form teams

Ideas for project topics

ECC:

- **Coding for distributed storage systems** (at least 2 potential project topics here)
  - Several additional metrics become important such as “reconstruction locality”, “reconstruction bandwidth”
  - Several new classes of codes have been proposed as alternatives to Reed-Solomon codes, e.g.,
    - Local reconstruction codes
    - Regenerating codes
    - Piggyback codes
  - Some employed in Microsoft Azure cloud storage, some in Apache Hadoop Distributed File System, some in Ceph, etc.

Ideas for project topics

ECC (cont.)

- Coding for **latency sensitive streaming communication** (at least 1 potential project topic here)
    - Sequential encoding and decoding
    - Strict latency constraints
    - A new class of codes called “streaming codes”

Ideas for project topics

Compression:

- Quantization in neural networks
- DNA compression
- Latest compression algorithm Zstd developed by Facebook

Ideas for project topics

Hashing:

- Several network applications
  - Used for network monitoring
  - Sketching using hashing

15-853:Algorithms in the Real World

Hashing:

- Concentration bounds
- Load balancing: balls and bins
- Hash functions (cont.)

Recall: Hashing

Concrete running application for this module: **dictionary**.

Setting:

- A large universe of keys (e.g., set of all strings of certain length): denoted by **U**
- The actual dictionary **S** (subset of U)
- Let |S| = N (typically N << |U|)

Operations:

- add(x): add a key x
- query(q): is key q there?
- delete(x): remove the key x

Recall: Hashing

"....**with high probability** there are not too many collisions among elements of S"

On what is this probability calculated over?

Two approaches:

1. Input is random
2. Input is arbitrary, but the hash function is random

Input being random is typically not valid for many applications.

So we will use 2.

- We will assume a family of hash functions H.
- **When it is time to hash S, we choose a random function h $\in$H**

Recall: Hashing: Desired properties

Let [M] = {0, 1, ..., M-1}

We design a hash function h: U -> [M]

1. Small probability of distinct keys colliding:
   1. If $x \neq y \in S$, $P[h(x) = h(y)]$ is "small"
2. Small range, i.e., small M so that the hash table is small
3. Small number of bits to store h
4. h is easy to compute

Recall: Ideal Hash Function

Perfectly random hash function:

For each $x \in S$, $h(x)$ =a uniformly random location in $[M]$

Properties:

- Low collision probability: $P[h(x) = h(y)] = 1/M$ for any $x \neq y$
- Even conditioned on hashed values for any other subset A of S, for any element $x \in S$, $h(x)$ is still uniformly random over $[M]$

Recall: Universal Hash functions

Captures the basic property of non-collision.

Due to Carter and Wegman (1979)

Definition: A family H of hash functions mapping U to [M] is universal if for any $x \neq y \in U$,

$$P[h(x) = h(y)] \leq 1/M$$

Note: Must hold for every pair of distinct x and $y \in U$.

Recall: Universal Hash functions

A simple construction of universal hashing:

Assume $|U| = 2^u$ and $|M| = 2^m$

Let A be a m x u matrix with random binary entries.

For any x $\in$U, view it as a u-bit binary vector, and define

$$h(x) := Ax$$

where the arithmetic is modulo 2.

Theorem. The family of hash functions defined above is universal.

Recall: Addressing collisions in hash table

One of the main applications of hash functions is in hash tables (for dictionary data structures)

Handling collisions:

Closed addressing

Each location maintains some other data structure

One approach: “**separate chaining**”

Each location in the table stores a **linked list** with all the elements mapped to that location.

Look up time = length of the linked list

To understand lookup time, we need to study the number of many collisions.

Recall: Addressing collisions in hash table

Let us study the number of many collisions:

Let C(x) be the number of other elements mapped to the value where x is mapped to.

Q: What is E[C(x)] ?

E[C(x)] = (N-1)/M

Hence if we use M = N = |S|,

lookups take **constant time in expectation**.

Item deletion is also easy.

Let C = total number of collisions

Q: What is E[C] ?

$$\binom{N}{2} 1/M$$

Recall: Addressing collisions in hash table

Can we design a collision free hash table?

Suppose we choose $M >= N^2$

Q: P[there exists a collision] = ?

½

$\Rightarrow$ Can easily find a collision free hash table!

$\Rightarrow$ Constant lookup time **for all** elements! (worst-case guarantee)

But this is large a space requirement.

(Space measured in terms of number of keys)

Can we do better? O(N)? (while providing worst-case guarantee?)

Application: Perfect hashing

Handling collisions via "**two-level hashing**"

First level hash table has size O(N)

Each location in the hash table performs a collision-free hashing

Let C(i) = number of elements mapped to location i in the first level table

Q: For the second level table, what should the table size at location i?

C(i)^2 (We know that for this size, we can find a collision-free hash function)

Application: Perfect hashing

Q: What is the total table space used in the second level?

$$\sum_{i=1}^{M} C(i)^2$$

We know $E(C) = \binom{N}{2}\frac{1}{M} \Rightarrow E\left[\sum_{i=1}^{M}\binom{C(i)}{2}\right] = \binom{N}{2}\frac{1}{M}$

$E\left[\sum_{i=1}^{M} C(i)^2 - \sum_{i=1}^{M} C(i)\right] = O(N)$ since $M = O(N)$

$\Rightarrow E\left[\sum_{i=1}^{M} C(i)^2\right] = O(N)$ since $E\left[\sum_{i=1}^{M} C(i)\right] = O(N)$ as shown earlier

Q: What is the total table space?

O(N)

Collision-free and O(N) table space!

k-wise independent hash functions

In addition to universality, certain independence properties of hash functions are useful in analysis of algorithms

Definition. A family H of hash functions mapping U to [M] is called k-wise-independent if for any k distinct keys

$x_1, x_2, \ldots x_k$ and any k distinct values $\alpha_1, \alpha_2, \ldots, \alpha_k$

we have

$$P\left(h(x_1) = \alpha_1 \cap h(x_2) = \alpha_2 \cap \ldots \cap h(x_k) = \alpha_k\right) \le \frac{1}{m^k}$$

Case for k=2 is called “pairwise independent.

k-wise independent hash functions

Properties:

Suppose H is a k-wise independent family for k>=2. Then

1. H is also (k-1)-wise indepdent.
2. For any $x \in U$ and $a \in [M]$ $P[h(x) = a] = 1/M$.
3. H is universal.

Q: Which is stronger: pairwise independent or universal?

Pairwise independent is stronger.

E.g.?

$h(x) = Ax$ construction since $P[h(0) = 0] = 1$

Some constructions: 2-wise independent

Construction 1 (variant of random matrix multiplication):

Let A be a m x u matrix with uniformly random binary entries.

Let b be a m-bit vector with uniformly random binary entries.

$$h(x) := Ax + b$$

where the arithmetic is modulo 2.

Claim. This family of hash functions is 2-wise independent.

Q: How many hash functions are in this family?

$2^{(u+1)m}$

Q: Number of bits to store?

O(um)

Can we do with fewer bits?

Some constructions: 2-wise independent

Construction 2 (Using fewer bits):

Let A be a m x u matrix.

- Fill the first row and column with uniformly random binary entries.
- Set $A_{i,j} = A_{i-1,j-1}$

Let b be a m-bit vector with uniformly random binary entries.

$$h(x) := Ax + b$$

where the arithmetic is modulo 2.

Claim. This family of hash functions is 2-wise independent. (HW)

Some constructions: 2-wise independent

Construction 3 (Using finite fields)

Consider GF(2^u)

Pick two random numbers a, b ∈ GF(2^u). For any x ∈ U, define h(x) := ax + b

where the calculations are done over the field GF(2u).

Q: What is the domain and range of this mapping?

[U] to [U]

Q: Is it 2-wise independent?

Yes (write as a matrix and invert) <board>

Some constructions: 2-wise independent

Construction 3 (Using finite fields)

Consider $GF(2^u)$.

Pick two random numbers a, b ∈ $GF(2^u)$. For any x ∈ U, define
h(x) := ax + b

where the calculations are done over the field GF(2u).

Q: What is the domain and range of this mapping?

[U] to [U]

Q: Is it 2-wise independent?

Yes

Q: How change the range to [M]?

Truncate last u=m bits. Still is 2-wise independent.

Some constructions: k-wise independent

Construction 4 (k-wise independence using finite fields):

Q: Any ideas based on the previous construction?

Hint: Going to higher degree polynomial instead of linear.

Consider GF(2^u).

Pick k random numbers $a_0, a_1, \ldots, a_{k-1} \in GF(2^u)$

$$h(x) = a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}$$

where the calculations are done over the field GF(2u).

Similar proof as before.

Other hashing schemes with good properties

Simple Tabulation Hashing:

Consider $U = [k]^u$

Initialize a 2-dimensional u x k array T with each of the u*k entries having a random m-bit string.

For the key $x = x_1 x_2 \ldots x_u$, define its hash as

$$h(x) := T[1, x_1] \oplus T[2, x_2] \oplus \ldots \oplus T[u, x_u].$$

Other hashing schemes with good properties

Simple Tabulation Hashing:

Consider $U = [k]^u$. Initialize a 2-dimensional u x k array T with each of the u*k entries having a random m-bit string.

For the key $x = x_1 x_2 \ldots x_u$, define its hash as

$$h(x) := T[1, x_1] \oplus T[2, x_2] \oplus \ldots \oplus T[u, x_u].$$

Q: How many random bits?

ukm

Q: Size of the hash family?

2^{ukm}

Theorem. Tabulation hashing is 3-wise independent but not 4-wise independent.

(We will not prove this)

Other approaches to collision handling

Open addressing:

No separate structures

All keys stored in a single array

Linear probing:

When inserting x and h(x) is occupied, look for the smallest index i such that (h(x) + 1) mod M is free, and store h(x) there.

When querying for q, look at h(q) and scan linearly until you find q or an empty space.

Other approaches to collision handling

Linear probing (cont.):

- Deletions are not quite as simple any more.
- It is known that linear probing can also be done in expected constant time, but universal hashing does not suffice to prove this bound: 5-wise independent hashing is necessary [PT10]and sufficient [PPR11].

Other probe sequences:

Using a step-size

Quadratic probing

[Mihai Patrascu and Mikkel Thorup, 2010]

[Anna Pagh, Rasmus Pagh, and Milan Ruzic, 2011]