CS5314
Randomized Algorithms

Lecture 15: Balls, Bins, Random Graphs (Hashing)

Objectives

- Study various hashing schemes
- Apply balls-and-bins model to analyze their performances

Chain Hashing

- Suppose our library wants to maintain a book inventory system so that a user can search if a certain book is available

A Natural Method:

Keep a list of the names of the books

→ When a user asks for a certain book, we check if its name w is in the list

Chain Hashing (2)

- Assume each book name is of $O(1)$ length (say, 8 to 80 characters)
- Let m = # books in our library
- To speed up the checking process, we store the m book names in sorted order

➔ checking w takes: $O(\log m)$ time

Chain Hashing (3)

Another idea to speed up:

Create a hash function f that places the m book names into n bins

➔ Name x is placed in Bin f(x)

- When w arrives, we compare w with all the names in Bin f(w)

➔ Report found if w is in Bin f(w)

Chain Hashing (4)

- Usually, we can find a good hash function f, such that:

  For a random name x,

  1. $\Pr(f(x) = j) = 1/n$ for each j

     → f appears random

  2. Values of $f(x)$ are independent of each other → f appears independent

Chain Hashing (5)

- In addition, suppose further we can compute $f(x)$ in $O(1)$ time ...

- What will be the time for the checking?

  [Can you see it is exactly asking about the load in the Balls-and-Bins model?]

- Firstly,

$$E[\#\text{ names in a bin}] = m/n$$

Chain Hashing (6)

- If $n = m$,

➔ Expected # names = 1

- Also, maximum # names in a bin is:

  $\Theta(\ln m / \ln \ln m)$ w.h.p.

➔ Better than binary search !!!

➔ Drawback: wasted space

For instance, if we use m bins for m items, several bins will be empty ...

Approximate Membership

- Suppose we now have a similar problem : to maintain a password checker system so that a user can tell if a certain password is in the blacklist

➔ Before a user updates the password to w, we check if w is in the blacklist

- Let m = # bad passwords in the blacklist

Approximate Membership

Using the previous ideas, we can either

- Maintain sorted list:
  ➔ checking time: $O(\log m)$

- Find a good hash function:
  ➔ checking time: $O(\ln m/\ln \ln m)$ w.h.p.

Approximate Membership

Alternative scheme :

Target: To save space

Trade-off: Allow false positive errors

(meaning: we may say w is bad even if it is not in the blacklist)

However, we will never say w is good if it is in the blacklist

Approximate Membership (2)

Idea: To represent each of the m bad passwords with a short fingerprint

Then, when w arrives,

1. compute the fingerprint of w
2. If it matches fingerprints of any bad passwords, we say w is in the list
3. Else, we say w is not in the list

Thus, the shorter the fingerprint, the more likely that a false positive error occurs

Approximate Membership (3)

In general, our problem is as follows:

- Let $S = \{ s_1, s_2, \ldots, s_m \}$, with $s_i \in [1,U]$.
- Assume we have a good hash function so that each s_i can be mapped randomly to a short fingerprint of b bits long
- Suppose we also allow

$$\Pr(\text{false positive error}) \leq r$$

Question: What is min length of b?

Approximate Membership (4)

With the given hash function,

for an item s' not in S, an item s_j in S,

Pr(s' and s_j have different fingerprints)

$= 1 - 1/2^b$

➔ For an item s' not in S,

Pr(false positive error)

$= 1 - (1 - 1/2^b)^m \geq 1 - e^{-m/2^b}$

Approximate Membership (5)

Since we want the false positive error probability to be at most r, we need

$$r \geq 1 - e^{-m/2^b}$$

So, $e^{-m/2^b} \geq 1 - r$, or $-m/2^b \geq \ln(1 - r)$

➔ $2^b \geq -m / \ln(1 - r)$

➔ $b \geq \log_2(-m / \ln(1 - r))$

Thus, if r is a constant, $b = \Omega(\log m)$

Approximate Membership (6)

- What if we choose $b = 2 \log m$?
- In this case,

  Pr(false positve error)

  $= 1 - (1 - 1/2^b)^m$

  $= 1 - (1 - 1/m^2)^m$

  $< 1/m$

Bloom Filters

Can we get more tradeoff between space (b) and false positive error probability (r)?

A method, called Bloom Filter, is to prepare:

- an n-bit vector A[1..n] (initially all bits are 0)
- k independent good hash functions,

$$h_1, h_2, \ldots, h_k,$$

each can map an element to [1,n]

Bloom Filters (2)

Then, for each element s_j in S,

1. Compute k hash values $h_i(s_j)$
2. Mark corresponding bits $A[h_i(s_j)]$ to 1

Later, to test if a value s is in S,

1. Apply the k hash functions on s
2. Find the corresponding k bits in A
3. If all are 1, we conclude that s is in S
4. Else, we conclude that s is not in S

Bloom Filters (3)

Questions:

When can a Bloom filter make an error?

(1) Will it say s is in S when s is not in S?

(2) Will it say s is not in S when s is in S?

Answer. (1) Yes. (2) No.

➔ Only have false positive errors

Bloom Filters (4)

- The probability of false positive error can be calculated as follows:

  (recall: m = size of S, n = length of A)

- First, in the desired Bloom filter,

  Pr(a specific bit $A[x] == 0$)

  $= (1 - 1/n)^{km} \approx e^{-km/n} = p$

- Next, we assume exactly a fraction of p entries in A is 0

  → this assumption will be removed later

Bloom Filters (5)

Based on the assumption, we have

Pr(false positive error)

$= (1 - p)^k$

➔ We should minimize the value

$$f = (1 - p)^k = (1 - e^{-km/n})^k$$

Question:

Should we use a large k? Or a small k?

Bloom Filters (6)

Suppose m and n are given. Observe that:

(1) False positive error occurs only if all the corresponding k bits are 1

→ if k is large, more difficult to occur

→ Better to have large k

(2) If k is very large, the bit-vector A in will be nearly all 1's !

→ easy to have false positive error …

Bloom Filters (7)

First, to minimize $f \Leftrightarrow$ minimize $\ln f$

- Let us find the optimal k by calculus:
- Let $g(k) = \ln f = k \ln (1 - e^{-km/n})$
- Differentiating g, we have

$$g' = \ln (1 - e^{-km/n}) + ke^{-km/n}(m/n)/(1 - e^{-km/n})$$

➔ $g' = 0$ when $k = (\ln 2)\,(n/m)$

which corresponds to a global minimum

Bloom Filters (8)

When we choose the best $k = (\ln 2)\,(n/m)$,

$$f = (1 - e^{-km/n})^k$$
$$= (1/2)^k$$
$$= (0.6185)^{n/m}$$

Remark 1: In practice, k must an integer, so we cannot achieve the global min

➔ Actual f will be slightly higher

Remark 2: If $k = 1$, it is exactly the previous fingerprint scheme

Bloom Filters (9)

Question: What is space usage per item?

- The space of the k hash functions should be negligible
- A Bloom filter uses n bits, and we have m items → n/m bits per item

Is Bloom Filter better than the previous fingerprint scheme?

Bloom Filters (10)

For fingerprint scheme,

constant false positive error probability requires $\Omega($ log m $)$ bits per item …

For Bloom filter,

already very effective if we have constant bits per item

E.g., when $n/m = 8$, k is around 5 or 6

➔ Pr(false positive error) $\approx$ 0.021

Bloom Filters (11)

- We now remove the assumption that exactly a fraction of p entries in A is 0
- In the actual case, the fraction of 0 is equivalent to the fraction of empty bins after km balls are thrown into n bins

(1) What is E[#entries with 0 balls] ?

(2) How to bound the actual fraction of 0 is very close to p?

Bloom Filters (12)

Answer:

(1) The expected number of entries with 0 balls = $n(1 - 1/n)^{km}$

(2) Let us use Poisson Approximation

Let $p' = (1 - 1/n)^{km}$

Let X = number of 0-entries

$r = km$ = number of balls

Bloom Filters (13)

Also, define indicator

$X_j = 1$ if j^{th} entry has 0 balls

$X_j = 0$ otherwise

→ $X = X_1 + X_2 + \ldots + X_n$

→ $\Pr(|X - np'| \geq \varepsilon n$ in exact case$)$

$\leq e r^{1/2} \Pr(|X - np'| \geq \varepsilon n$ in Poisson case$)$

$= e r^{1/2} \Pr(|\Sigma_j X_j - np'| \geq \varepsilon n$ in Poisson case$)$

Bloom Filters (14)

- In Poisson case, X_j's are independent, and each of them has probability p' to be 1
- In other words, in Poisson case,

X = sum of n independent Bernoulli trials each with probability p' of success

= $Bin(n, p')$

Bloom Filters (15)

- Thus, we can apply Chernoff bound for Bin(n, p′) and obtain:

$\Pr(\ |X - np'| \geq \varepsilon n$ in exact case)

$\leq e r^{1/2}\ \Pr(\ |X - np'| \geq \varepsilon n$ in Poisson case)

$=\ e r^{1/2}\ \Pr(\ |\mathrm{Bin}(n, p') - np'| \geq \varepsilon n\ )$

$\leq e r^{1/2}\ 2e^{-n\varepsilon^2/3p'} \leq 0.00001$ when n is large

Bloom Filters (15)

- Thus, when n is large, the actual fraction of 0, X/n, is very close to p', w.h.p.

- Also, recall: $p' = (1 - 1/n)^{km}$

  and $p = e^{-km/n}$

  so that $p' \approx p$

  ➔ actual fraction of 0 is very close to p

  ➔ previous assumption is true w.h.p.

Breaking Symmetry

- Suppose n users run their programs on a server and want to get the running times
- In order to measure the time accurately, they agree to use the server sequentially, one program at a time
- Of course, each user wants to be scheduled as early as possible …

Question: How can we decide a permutation of the users quickly and fairly?

Breaking Symmetry (2)

We can use hashing to help !

1. Create a hash function f that maps each user to one of the 2^b bins (i.e., hash a user into a number between $[1,2^b]$)
2. Sort users based on their hash values

For this scheme to work, we do not want two users to have the same hash value

➔ this should happen w.h.p. when b is large

Breaking Symmetry (3)

Assume that the hash function is good
(which appears random and independent)

→ Probability that a particular user receive a hash value same as some other user is:

$$1 - (1 - 1/2^b)^{n-1} \le (n-1)/2^b$$

Thus, by union bound,

Pr(all users has distinct hash value)

$\ge 1 - n(n-1)/2^b$

$\ge 1 - 1/n$ ... when $b = 3 \log n$

Breaking Symmetry (4)

Advantage: Extremely flexible !

New user can join at any time, as long as they do not have the same hash value as the existing users

Related problem:

Selecting a leader from n people

➔ If we have a good hash function, we can hash each user and select one with smallest value to be the leader

In this case, what should b be? (Ex. 5.25)