2-INF-237 Vybrané partie z dátových štruktúr

2-INF-237 Selected Topics in Data Structures

- Instructor: Broňa Brejová
- E-mail: brejova@fmph.uniba.sk
- Office: M163

- Course webpage: http://compbio.fmph.uniba.sk/vyuka/vpds/

Introduction to hashing

- Universe $U = \{0, ..., u - 1\}$
- Table of size m
- Hash function $h : U \to \{0, ..., m - 1\}$
- Let X be the set of elements currently in the hash table, $n = |X|$
- We will assume $n = \Theta(m)$

Totally random hash function

(a.k.a uniform hashing model)

- select $h(x)$ for each $x \in U$ uniformy independently
- not practical - storing this function requires $u \lg m$ bits
- used for simplified analysis of hashing in ideal case

Universal family of hash functions

- set of hash functions H
- $\exists c$ such that for any distinct $x, y \in U$ we have
  $\Pr(h(x) = h(y)) \leq c/m$
- Example: choose prime $p \geq u$
  $H_p = \{h_a \mid h_a(x) = (ax \bmod p) \bmod m), 1 \leq a \leq p-1\}$

Hashing with chaining and universal family of h.f.

- Linked list (or vector) for each hash table slot
- Let c_i be the length of chain at position i,
  let $C_x = c_{h(x)}$ for $x \in U$
- We have $E_{h \in H}[C_x] \leq 1 + cn/m = O(1)$
- However $E_{h \in H}[\max_i c_i] = O(\sqrt{n})$
- For a totally random function this is $O(\log n / \log \log n)$

Perfect hashing

- Fredman, Komlos, Szemeredi 1984
- Top level: universal hash function to table of size $\Theta(n)$
- Second level: bucket i with c_i elements hashed to a table of size αc_i^2
- If any collision at second table, choose a new hash function
- If overall size too big, start from scratch
- Overall $O(1)$ deterministic search, $O(n)$ space, $O(n)$ expected preprocessing
- Dynamic version resizes similarly as vector, $O(1)$ amortized expected update, but still $O(1)$ deterministic search

Bloom filter (Bloom 1970)

Supports insert x, test if x is in the set

– may give false positives, e.g. claim that x is in the set when it is not

– false negatives do not occur

Algorithm

a bit vector $B[0, \ldots, m-1]$,

k hash functions $h_1, \ldots, h_k : U \to \{0, \ldots, m-1\}$

insert(x): set $B[h_1(x)], \ldots, B[h_k(x)]$ to 1

contains(x): check if $B[h_1(x)], \ldots B[h_k(x)]$ are all 1

– if yes, claim x is in the set, but possibility of error

– otherwise answer no, surely true

Analysis: If all h_i are totally random and independent, the probability of error is approximately $(1 - e^{-nk/m})^k$.

Bloom filter analysis

- If hash functions are totally random and independent, the probability of error is approximately $(1 - e^{-nk/m})^k$
- For $k = \ln(2)m/n$, get error $c^{-m/n}$, where $c = 2^{\ln(2)} \approx 1.62$
- To get error rate p for some n, we need $m = n \lg(1/p) \lg(e)$
- For 1% error, we need about $m = 10n$ bits of space and $k = 7$
- Memory and error rate are independent of the universe size
- Hash table needs at least to store data (e.g. in $n \lg u$ bits)
- If we used $k = 1$ (one hash function), we need $99.5n$ for 1% error

Bloom filter analysis

If all h_i are totally random and independent, the probability of error is approximately $(1 - e^{-nk/m})^k$.

- Insert n elements, each with k hash functions
- $\Pr(B[i] = 1) = 1 - (1 - 1/m)^{nk} \approx 1 - e^{-nk/m}$, denote p
- Expected number of 1's is pm, but individual $B[i]$ not independent
- False positive: k randomly chosen slots all 1
- If the hash table exactly pm 1's and $(1 - p)m$ 0's
  probability of false positive would be p^k as claimed
- But we do not know exact count of 1's
  therefore individual samples not independent
- With high prob. the count close to pm (modified Chernoff bound)

Why independence not true?

Insertion of n elements creates unknown number of 1's.

Each hash function in search tells us something about this count.

This would happen even if $B[i]$ independent.

Simplified toy example

Hash table with 2 slots,

each set to 1 or 0 uniformly independently.

Twice sample a slot (indep., uniformly),

results X_1, X_2

$\Pr(X_2 = 1) = 1/2$

$\Pr(X_2 = 1 \mid X_1 = 1) = 3/4$

B_1	B_2	X_1	X_2
0	1	B_2	$B_1 = 0$
0	1	B_2	$B_2 = 1$
1	0	B_1	$B_1 = 1$
1	0	B_1	$B_2 = 0$
1	1	B_1	$B_1 = 1$
1	1	B_1	$B_2 = 1$
1	1	B_2	$B_1 = 1$
1	1	B_2	$B_2 = 1$

Bloom filters in practice

- Approximate index of a larger data structure on disk / over network
- If Bloom filter says no, do not check on disk

  If Bloom filter says yes, check if it is indeed on disk

  False positives increase running time, but no effect on correctness
- Google BigTable [Chang et al 2008]

  Bloom filter for triples (row, column, timestamp)

  If Blom filter says yes, try to retrieve value from the table

Counting Bloom filters

- Support insert and delete
- Each $B[i]$ a counter with b bits
- Insert increases counter, delete decreases
- Assume no overflows, or reserve largest value as infinity

Bloom filters in theory

- Bloom filter uses about $1.44n \lg(1/p)$ bits to achieve error prob. p
- Lower bound $n \lg(1/p)$ bits [Carter et al 1978]
- Constant 1.44 can be improved to $1 + o(1)$ with more complex structures [Pagh et al 2005]

Nearest neighbor

- Preprocess set X
- Query(x):

  find $y \in X$ minimizing $d(x, y)$ for a given distance measure d

Approximate near neighbour (r, c)-NN

- Preprocess a set X for given $r > 0$, $c > 1$
- Query(x):

  if there is $y \in X$ such that $d(y, x) \leq r$,

  return $z \in X$ such that $d(z, x) \leq c \cdot r$
- Locality sensitive hashing returns such z with probability $\geq f$

Algorithm

For Hamming distance on strings of length d

- Hash each string using L hashing functions,
  each using k randomly chosen positions (random projections)
- Then rehash to obtain total space $O(nd + nL)$
- Given x, find collisions in each hash function,
  report if any at distance at most cr
- If checked more than $3L$ collisions, stop
- Query time $O(Lk + Ld)$

$k = O(\log n)$, $L = O(n^{1/c})$

Probability of failure less than some constant $f < 1$

Can be boosted by repeating independently

Approximate string matching

Hamming distance $d_H(S_1, S_2)$ between two strings of equal length:

the number of positions where they differ

Task: find approximate occurrences of P in T with Hamming distance $\leq k$

$\{i \mid d_H(P, T[i..i+m-1]) \leq k\}$

Trivial algorithm $O(nm)$

Algorithm with suffix trees and LCA: $O(nk)$ [Landau, Vishkin 1986]

More complex version: $O(n\sqrt{k \log k})$ [Amir, Lewenstein, Porat 2000]

Using fast Fourier transform: $O(n\sigma \log m)$ [Fischer and Paterson 1974]

A LSH family

A family of hash functions H is (r_1, r_2, p_1, p_2)-sensitive if for any $x, y \in U$ we have

If $d(x, y) \leq r_1$ then $\Pr_{h \in H}(h(x) = h(y)) \geq p_1$

If $d(x, y) \geq r_2$ then $\Pr_{h \in H}(h(x) = h(y)) \leq p_2$

We want $p_1 > p_2$, $r_1 = r$, $r_2 = cr$

Example:

Consider binary strings of length d under Hamming distance

Let $h_i(x)$ be the i-th bit of x

$(r, cr, 1 - \frac{r}{d}, 1 - \frac{cr}{d})$-sensitive family of size d

Amplification of (r_1, r_2, p_1, p_2)-sensitive family H

AND amplification

Randomly choose $h_1, \ldots, h_k \in H$

Let $g(x) = (h_1(x), \ldots, h_k(x))$

$g(x) = g(y)$ iff $h_i(x) = h_i(y)$ for **all** $i \in \{1, \ldots, k\}$

Such g form (r_1, r_2, p_1^k, p_2^k)-sensitive family

OR amplification

Randomly choose $h_1, \ldots, h_L \in H$

Create L hash tables, one for each h_i, take union of hits from all tables

Consider $g(x) = g(y)$ iff $h_i(x) = h_i(y)$ for **some** $i \in \{1, \ldots, L\}$

Similar to $(r_1, r_2, 1 - (1 - p_1)^L, 1 - (1 - p_2)^L)$-sensitive family

Combine to get $(r_1, r_2, 1 - (1 - p_1^k)^L, 1 - (1 - p_2^k)^L)$-sensitive family

Results for LSH

Consider $(r_1, r_2, 1-(1-p_1^k)^L, 1-(1-p_2^k)^L)$-sensitive family

$k = \lceil \log_{1/p_2} n \rceil$, $L = n^\rho / p_1$

$\rho = \log(1/p_1)/\log(1/p_2)$

Results

Probability of success ≥ 0.29

Space $O(nd + nL)$

Query time $O(Lk + Ld)$

Hamming distance

$p_1 = 1 - r/d$, $p_2 = 1 - cr/d$

$\rho \leq 1/c$

If needed, pad with zeroes (increase d) to make $p_1, p_2 = \Omega(1)$

Space $O(nd + n^{1+1/c})$, query time $O(n^{1/c}(d + \log n))$

Outline of analysis

Consider $(r_1, r_2, 1-(1-p_1^k)^L, 1-(1-p_2^k)^L)$-sensitive family

$k = \lceil \log_{1/p_2} n \rceil$, $L = n^\rho / p_1$

$\rho = \log(1/p_1)/\log(1/p_2)$

Probability of collision of x and $y \in X$ s.t. $d(x,y) \leq r_1$ is

$\geq 1-(1-p_1^k)^L \geq 1 - 1/e \approx 0.63$

Probability of collision x and $y \in X$ s.t. $d(x,y) \geq r_2$ in one hash table is

$\leq p_2^k = 1/n$

Expected number of collisions in all tables is $\leq L$

By Markov inequality $\geq 3L$ collisions with probability $\leq 1/3$

Prob. of getting a good collision and not too many collisions

$\geq 1 - 1/3 - 1/e \approx 0.298787$

Minimum hashing

Jaccard index

similarity of two sets $J(A, B) = |A \cap B|/|A \cup B|$,

distance d(A,B) = 1-J(A,B)

used e.g. for approximate duplicate document detection based on set of words in the document

Minimum hash of set A

$\text{minhash}_h(A) = \min_{x \in A} h(x)$

Assume totally random hash function h and no collisions in $A \cup B$

$\Pr(\text{minhash}_h(A) = \text{minhash}_h(B)) = J(A, B) = 1 - d(A, B)$

$(r_1, r_2, 1 - r_1, 1 - r_2)$-sensitive family

Again can we apply amplification

LSH notes

Real-valued vectors: map to bits using random projections $\text{sgn}(w \cdot x + b)$

Data-dependent hashing

Use PCA, optimization, kernel tricks, even neural networks to learn hash functions good for a particular set of points and distance measure.