Lecture 9 notes

Hash based data structures
Irene Tang
2/9/2021

Some definitions

<u>Hash code:</u> maps a data item to a finite codespace (e.g. a set of integers)
<u>Hash map:</u> you have an array of buckets

x -> h(x) -> figure out which bucket x should be stored in
O(1) lookup, O(1) insertion

Review question 1

Suppose you have a hash function h : x -> {0, …, N}
But N is too large for the present purpose, so you want to construct a new codespace N
How do you do this?

h'(x) = h(x) mod N'
this new function is no longer uniform because the first few buckets will be more full, due to the mod

Review question 2

Under what circumstances is mod N' okay?

When N is a multiple of N'
It is hard to construct N' this way because N is usually a large prime number

When N' << N (much smaller than), then you can restrict the codespace well because there are many foldbacks over the codespace so the different basket weights will not be so pronounced

Review question 3

You have a list of strings [a, a, b, c, d, b]
You hash each of the strings h : s -> {0, 1, 2, 3}
What is the probability that a is in the first bucket?

¼ , since the codespace is 4 possible buckets regardless of the data distribution

String similarity lookup

Your database contains a collection of strings, perhaps corresponding to a bunch of real movie titles. "The quick brown fox" is a real string in this collection. If you are given the similar input string "the quick", can you do a rough similarity lookup with the same O(1) speed as a hash map? You want to find all the matches in the database similar to your query string, namely "The quick brown fox".

Method #1 – Jaccard lookup

First, here is how to define similarity.
Jaccard similarity: measures the (# of words in common) ÷ (# total distinct words)

Naïve Jaccard lookup algorithm – O(N) lookup, O(1) insertion

1. You have a string q to look up
2. Go through each of the strings s in your HashMap
3. Calculate the Jaccard similarity J(s,q)
4. Gather the ones for which the Jaccard similarity is greater than some threshold t

Method #2 – MinHash function

Because we want to avoid going through every string, for efficiency

Calculate the hash of each individual word

    h(the)
    h(quick)
    h(brown)

Take the one with the smallest hash (aka the min value). Each has equal probability of being the minimum.

Suppose you have two strings s1, s2
What is the probability that MinHash(s1) = MinHash(s2)?

    Two possibilities for this to happen:
    Either they both have the same MinHash word
    Or there is a collusion

When the global min is contained in both sides
This is the Jaccard similarity between the two – Jacc(s1,s2) = int/union

1. Generate k independent hash functions. (K << N)
2. Do the MinHash x times for each of the hash functions
3. Given a query string q, we get a vector of MinHashes
4. Expected value is the Jaccard similarity, especially as k approaches infinity