COMS 4995-2: Advanced Algorithms (Spring '20) Jan 28, 2020

Lecture 3: Dictionary via Hashing, Perfect Hashing

Instructor: *Alex Andoni* Scribes: *Benjamin Most, Sian Lee Kitt*

1 Background

We have some fixed universe (a large set of numbers) $[U]=\{1, 2, ..., U\}$.

Problem: Given some subset $S \subset U$, with $|S| = m$, and some $x \in U$, report if $x \in S$.

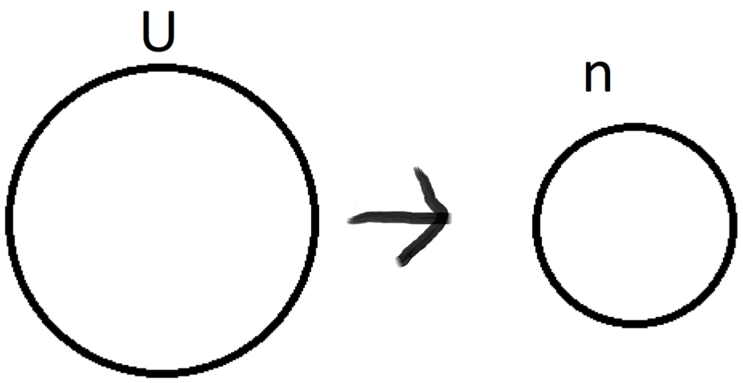
Last class, we discussed non-hashing solutions to this problem, including storing the entirety of S, using a Binary Search Tree, and using a bit array to store whether or not a value $x \in U$ is in S.

In this lecture, we discuss how to solve this problem using hashing.

Background terms/notation:

Collisions occur when our hash functions maps two elements into the same cell.

The expression $[k]$ for some integer k refers to the set $\{1, 2, ..., k\}$.



As displayed in the image above, we will consider hash functions mapping from $[U]$ to $[n]$. If $U > n \cdot m$ for some number m, then $\exists i \in [n]$ such that $|h^{-1}(i)| > m$, where h is our hash function.

2 Hashing

There are multiple ways we could solve the problem explained above using hashing.

2.1 Knuth's Solution

Knuth suggests the hash function

$$h(x) = \lfloor \{ \frac{\sqrt{5}-1}{2} x) \} n \rfloor$$

where the curly brackets denote taking the fractional part of the expression they contain.

Because it is deterministic, this hash function would perform better in some cases than in others, and in some scenarios it might lead to a high number of collisions.

2.2 Solution 1: Random Function

For every possible input x, we could randomly select the value of $h(x)$ and avoid all collisions. In this case, the average query time would be the expected value for the number of collisions we will get on computing the hash of x, which we denote as C_x. Therefore,

$$\begin{aligned} \mathbb{E}_n[C_x] &= \mathbb{E}_n[\# \text{ of } y \in S, y \neq x, h(y) = h(x)] \\ &= \mathbb{E}_n[\sum_{\substack{y \in S \\ y \neq x}} \mathbb{1}[h(y) = h(x)]] \\ &\leq m \cdot \frac{1}{n} \end{aligned} \tag{1}$$

So $\mathbb{E}[\text{query time}] = O(1)$ as long as $n \geq m$.

2.3 Solution 2: Less random

Definition: H family of $h : [U] \to [n]$ is universal if:

$$\forall x \neq y, \Pr_{h \in H}[h(x) = h(y)] = \frac{1}{n}.$$

Example: The family containing all hash functions is universal.

Fact: There exists a universal hash function family H such that $lg|H| = O(lg\ U)$.

Definition: H is α-almost-universal for $\alpha \geq 1$ if:

$$\Pr_{h \in H}[h(x) = h(y)] \leq \frac{\alpha}{n}$$

.

Example [Dietzfelbinger et al., '97]:

$\forall a \in [U]$, $h_a(x) = \lfloor (a \cdot x)\%U)/n \rfloor$.

$H = \{h_a, a \in U \text{ odd}\}$.

Fact: H is 2-almost-universal. This is equivalent to:

$$\forall x \neq y \in [U], \underset{a \in U \text{ odd}}{Pr}[h_a(x) = h_a(y)] \leq \frac{2}{n}$$

3 Perfect Hashing

The goal of perfect hashing is to have zero collisions.
We want O(1) query time. Define:

$$C \triangleq \sum_{x \in S} C_x \tag{2}$$

Zero collision count, C = 0, means that every bucket has at most one element.

3.1 Achieving a Zero Collision Count

We have that:

$$\mathbb{E}[C] = \mathbb{E}[\sum_{x \in S} C_x] \leq \frac{m^2}{n} \tag{3}$$

Recall: $\mathbb{E}[C_x] \leq m/n$.

If we fix $n = 4m^2$, then

$$\mathbb{E}[C] \leq \frac{1}{4}.$$

By Markov bound $Pr[C > \frac{3}{4}] \leq \frac{1}{3}$

With a probability $\geq \frac{2}{3}, C \leq \frac{3}{4}$

but since C $\in \mathbb{Z}$ then $C = 0$.

3.2 Solution 3

The algorithm is as follows:

```
At preprocessing, given input S, pick h randomly
If C = 0, done.
Else, try again.
```

Query time: $O(1)$ deterministically
Space: $O(m^2)$

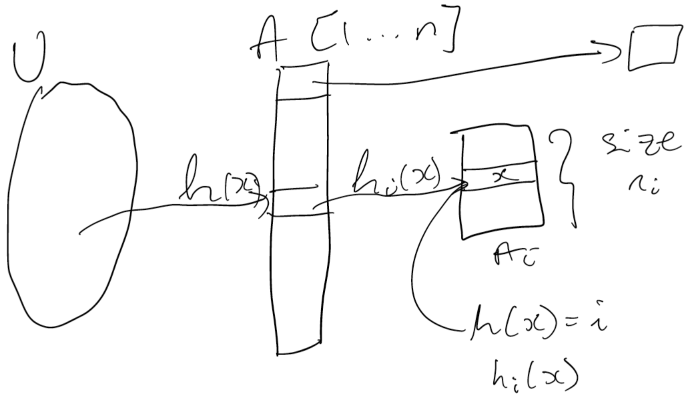
3.3 Solution 4: (Perfect Hashing)

We have two levels of hash functions: $h : [U] \rightarrow [n]$ (1st level)
$h_i : [U] \rightarrow [n_i]$ (2nd level) where $i \in [n]$ and $1 \leq n_i \leq m$

For Solution 4, we will use a two-level hashing system, with a goal of having no collisions in our 2nd-level arrays.

Our first-level hash function, h, will take elements of $[U]$ as input and return an output in $[n]$. We will have use an array A of length n, with n pointers to subarrays $A_1, A_2, ..., A_n$.

Our second-level hash functions, $h_1, h_2, ..., h_n$, will take an element of $[U]$ as input and return an element of $[n_i]$, where n_i is the length of subarray A_i. With this two-level hashing scheme, we can use h to find which subarray to access, then use the appropriate h_i to find the value we seek within that subarray, with no collisions in the subarrays giving us perfect hashing.



Pick h:[U] $\rightarrow$ [n]

Let s_i = number of elements x $\in$ S s.t. $h(x) = i$

$n_i \triangleq 4s_i^2$

At preprocessing for each $i \in [n]$,

if $\exists$ collisions in A_i, resample h_i until there are no collisions.

(With probability $\geq \dfrac{2}{3}$ there would be no collisions.)

Once there are no collisions for i, move on to $i + 1$

Query time: $O(1) + O(1) \implies O(1)$. $O(1)$ for first level hashing and $O(1)$ for second level hashing since there are no collisions. This is deterministic.

Space: $O(1) + O(n) + O(n) + O(\sum_{i\in[n]} n_i) = O(n) + O(\sum_{i\in[n]} s_i^2)$ where the hash function requires space $O(1)$, the first-level array requires space $O(n)$, and each h_i requires space $O(1)$ (so all h_i functions require space $O(n)$). The $O(\sum_{i\in[n]} s_i^2)$ term refers to the sum of the space required by each subarray.

Note that we approximate $n = m$.

Claim: $\mathbb{E}[\sum_i s_i^2] = O(m)$.

Proof.

$$\mathbb{E}[\sum_i s_i^2] = \mathbb{E}[\sum_{i=1}^{n} S_i(S_i - 1)] + \mathbb{E}[\sum_i S_i]$$

(where $s_i(s_i - 1)$ is the number of collisions in bucket i)

$= \mathbb{E}[C] + m$

$\leq \dfrac{m^2}{n} + m$

$\leq O(m)$ as long as $n = \Omega(m)$

□

So $\mathbb{E}[\mathit{space}] = O(m)$.

By Markov, space $\leq O(m)$ with probability $\geq \frac{1}{2}$.

4 Consistent Hashing

Consider a scenario in which we have m objects, n servers (similar to buckets) where n is much smaller than n (n $\ll$ m).

$\mathbb{E}[C_x] \leq \frac{m}{n}$. where C_x = objects per server.

$\mathbb{E}[x \in S(\text{Server}) \text{ s.t. } h(x) = i] \leq \frac{m}{n} \forall i \in [n]$.

But servers cannot last forever. Say we want to rebalance the network with new servers. We need to move a bunch of items to a new server.

For example: $h : [U] \to [n]$, $h' : [U] \to [n+1]$.

Items will completely reshuffle, and about $1 - \frac{1}{n}$ of the total items will be moved.

The goal is to have no collisions in the second level array. If size of second level using $4m^2$, then no collisions.