COMS 4995-8: Advanced Algorithms (Spring'21) Jan 11, 2021

Lecture 4: Perfect Hashing, Intro to Sketching and Streaming

Instructor: *Alex Andoni* Scribes: *Max Helman, Elena Gribelyuk*

1 Perfect Hashing

1.1 Background

Recall that we have a dictionary that when given $S \subset U$ is designed to answer the query "is $x \in S$?". Furthermore, we define:

$$n \triangleq |S|$$

$$h : U \to [m]$$

$$C_x = \#y \neq x, y \in S \text{ s.t. } h(x) = h(y)$$

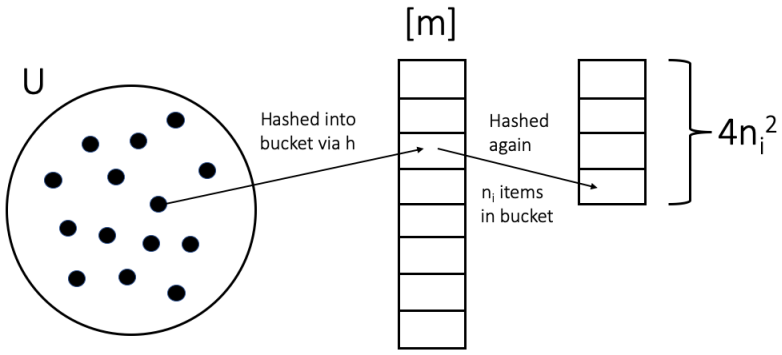
$$C = \sum_{x \in S} C_x$$

We claim that $\mathbb{E}[C] \leq \frac{n^2}{m}$, which allows us to construct a hash table of size $m = 4n^2$ where the size of each bucket ≤ 1.

1.2 Definition

Theorem 1. *Perfect hashing can always give us a data structure using $O(n)$ space, $O(1)$ query time, $O(n)$ construction time.*

Proof. To prove the properties of perfect hashing, we combine "standard" hashing with our $4n^2$ solution:



We are given $h : U \to [m]$ where $m \approx n$. A potential issue is that some buckets may be large ($>> 1$). We measure their size with:

$$n_i \triangleq \text{ size of bucket } i \ = \left|h^{-1}(i) \cap S\right|$$

From here, we add a second level to the hash table using the quadratic space solution. Our algorithm for perfect hashing is as follows:

Algorithm 1: Constructing a Table with Perfect Hashing

Result: a data structure using $O(n)$ space, $O(1)$ query time, $O(n)$ construction time
Pick some random $h : U \to [m]$ where $m = n$
for $i \in m$ **do**
  Build a second level hash function using $h_i : U \to \left[4n_i^2\right]$ such that there are no collisions in the second level ($\left[4n_i^2\right]$)
end
return the two-level hash table

Now, we must analyze this algorithm. Its correctness and consistent $O(1)$ query time are immediate by construction. The algorithm uses $O(n) + O\left(\sum_i n_i^2\right)$ space, corresponding to the first level table and second level tables respectively.

Claim 2. $\mathbb{E}\left[\sum_i n_i^2\right] = O\left(n\right)$

Proof.

$$\mathbb{E}\left[\sum_i n_i^2\right] \leq 2 \cdot \mathbb{E}\left[C\right]$$

$$= \mathbb{E}\left[n_1 + 2C_1 + n_2 + 2C_2 + ...\right]$$

$$= \mathbb{E}\left[C + \sum_i n_i\right] = n + \mathbb{E}\left[C\right]$$

$$= n + \frac{n^2}{m} = 2n$$

□

Note that in this proof, we add a factor of 2 because we have not defined whether we count a single collision as 1 collision (x collides with y) or 2 collisions (x collides with y and y collides with x). We also use the identity $n_1^2 = n_1(n_1 - 1) = n_1 + \binom{n_1}{2}$, where $\binom{n_1}{2}$ represents the number of collisions in bucket 1.

Observation 3. *Space is $O(n)$ in expectation only, but by Markov bounds, we know that it is $O(n)$ with probability $\geq 90\%$. Furthermore, we can repeat the construction $O(1)$ times until we achieve $O(n)$ space.*

Therefore, we have proved that perfect hashing will always give us a data structure using $O(n)$ space, $O(1)$ query time, $O(n)$ construction time. □

We remark that in this construction, we used expectation to provide analysis; a result of this is that it is enough to use universal hash functions to achieve these results.

2 Intro to Sketching and Streaming

2.1 Motivation

Next, we shift to the second topic of this course: sketching and streaming. Suppose we are given a stream of data $x_1, ..., x_m$ as to an algorithm. Our goal will be to store a "sketch" that maintains several key properties of the data such that the size of the sketch is smaller than or equal to the size of the original input data.

2.2 The Count-Distinct Problem

Consider the problem of counting distinct elements in a stream of size m; in particular, given $x_1, x_2, ..., x_m \in [n] = \{1, 2, ..., n\}$, we want to find the number of values of $[n]$ that appear at least once, i.e. the number of different values x_i present in the stream. This is a well-known problem with numerous applications– for instance, the elements might represent IP addresses of packets processed by a router or visitors to a website.

Note that we will analyze the space used by the proposed algorithm to be able to answer the desired question (i.e. the count-distinct question) at the end of the stream. This will be our sole performance measure, as we are currently discounting the time complexity of updating the sketch.

To solve this problem, we can immediately come up with the following relatively-naive, deterministic solutions:

- Use a hash table H. If $x_i \notin H$, hash x_i to insert it into the table. The space utilized by this solution is $O(d) \leq O(n)$, where $d =$ number of distinct items in the stream.
- Alternatively, store every x_i in the stream. This will take $O(m \log n)$ space.

So, we are led to ask: is it possible to do better? If we aim for an exact, deterministic solution, the answer is no. But, if we allow approximation and randomization, we can achieve a much better space complexity.

2.3 The Flajolet-Martin Algorithm

Theorem 4. *For $\epsilon < \frac{1}{2}$, $d > \Omega(\frac{1}{\epsilon^2})$. Furthermore, we can achieve $O(\frac{\log n}{\epsilon^2})$ space complexity.*

We will use the concept of a hash function to reduce the space complexity of our solution to the distinct element counting problem. However, in contrast to previously-used hash functions, we now define a fully random hash function $h : [n] \to [0, 1]$. We now introduce the Flajolet-Martin algorithm.

Algorithm 2: Flajolet-Martin

Result: number of distinct elements appearing in the stream
$z = 1$;
for $i \leftarrow 1$ ***to*** m **do**
  $z \leftarrow min\{h(i), z\}$
end
return $\frac{1}{z} - 1$

To demonstrate that this algorithm provides the correct result with $\geq 90\%$ probability, we will assert and prove a series of claims.

Claim 5. $\mathbb{E}[z] = \frac{1}{d+1}$.

Proof. To see this, we note that z is defined to be the minimum value of d random variables over $[0, 1]$. Pick some $a \in [0, 1]$ at random. Then, since h maps values in $[n]$ to $[0, 1]$ uniformly, it must be that the probability $Pr[a < z] = z$ precisely. Alternatively, we note that the quantity $Pr[a < z]$ also represents the probability that $h(x_m) = min\{h(x_1), h(x_2), ..., h(x_m)\}$. In other words, this is equal to the probability that when we choose $d+1$ random variables uniformly in $[0, 1]$, the last chosen value $h(x_m)$ is the smallest. Thus, $Pr[a < z] = \frac{1}{d+1}$. As a result, we conclude that $z = \frac{1}{d+1} \implies \mathbb{E}[z] = \frac{1}{d+1}$, as desired. □

Observation 6. *We observe that* $\mathbb{E}[\frac{1}{z}] \neq \frac{1}{\mathbb{E}[z]}$ *in general, suggesting that our estimator* $\frac{1}{z} - 1$ *is biased. So, in order to find upper and lower bounds for our estimate, we will now proceed to assert and prove a concentration bound for* z.

Claim 7. *For a fixed completely-random hash function* $h : [n] \to [0, 1]$, $var_h[z] \leq \frac{2}{d^2}$.

Proof. To compute $var_h[z]$, note that

$$var_h[z] = \mathbb{E}[z^2] - \mathbb{E}[z]^2$$

By the result of Claim 5, we know that $\mathbb{E}[z]^2 = \frac{1}{(d+1)^2} > 0$, so

$$var[z] = \mathbb{E}[z^2] - \mathbb{E}[z]^2 \leq \mathbb{E}[z^2]$$

Now, fix an $x \in [0, 1]$. Since h is fully random and h takes independent values in the range $[0, 1]$, we see that

$$Pr[x < z^2] = Pr[\sqrt{x} < z] = \prod_{i=1}^{d} Pr[\sqrt{x} < h(x_i)] = (1 - \sqrt{y})^d$$

Let $f_a(y)$ be the probability density function for z over $[0, 1]$, and note that $f_a(y) = 1 \ \forall y \in [0, 1]$ since h takes values of $[0, 1]$ uniformly. Then, it follows that

$$\mathbb{E}[z^2] = Pr[a < z^2] = \int_0^1 Pr[y < z^2] \cdot f_a(y) dy = \int_0^1 (1 - \sqrt{y})^d dy =$$

$$-\frac{2(1 - \sqrt{y})^{d+1}((d+1)\sqrt{y} + 1)}{(d+1)(d+2)} (1 - 2t) \Big|_0^1 = \frac{2}{(d+1)(d+2)} \leq \frac{2}{d^2}$$

□

But, we ask if this a good enough bound for us? Unfortunately, this result does not rule out the possibility that $z = 0$, which is problematic. However, it is possible to get a $1 \pm \epsilon$ factor approximation by running $k = O(\frac{1}{\epsilon^2})$ Flajolet-Martin algorithms for independently and identically distributed random variables. To achieve a $1 \pm \epsilon$ approximation, we present the Bottom-k algorithm in the following section.

2.4 The Bottom-k Algorithm

The Bottom-k Algorithm yields a $1 \pm \epsilon$ approximation for the number of distinct elements seen.

Algorithm 3: Bottom-k

Result: number of distinct elements appearing in the stream

$z_1, ..., z_k = 1$;

for *item i in stream* **do**

  Compute $h(i)$;

  Maintain $z_1 < z_2 < ... < z_k$ smallest k hash values seen in stream

end

return $\frac{k}{z_k}$

We now want to prove $\hat{d} = \frac{k}{z_k}$ is within $1 \pm \epsilon$ of d with $\geq 90\%$ probability.

Claim 8. *$Pr\left[\hat{d} > d \cdot (1+\epsilon)\right] \leq 0.05$ for $k = O\left(\frac{1}{\epsilon^2}\right)$. We can prove the other direction similarly.*

Proof. $\hat{\delta} > \delta \cdot (1+\epsilon) \iff \frac{k}{z_k} < \delta \cdot (1+\epsilon)$

$\iff z_k < \frac{k}{\delta \cdot (1+\epsilon)}$

$\iff \exists$ at least k items in stream whose hash function values $< \frac{k}{\delta \cdot (1+\epsilon)}$

Now, we define $\pi = Pr\left[\text{choose } d \text{ random variables } \in [0,1] \text{ such that at least } k \text{ of them } < \frac{k}{\delta \cdot (1+\epsilon)}\right]$. We define x_i as the indicator random variable of the ith distinct item meeting the condition $h(i) < \frac{k}{\delta \cdot (1+\epsilon)}$. Now, we see:

$$\mathbb{E}[x_i] = Pr\left[h(i) < \frac{k}{\delta \cdot (1+\epsilon)}\right]$$

$$Var(x_i) \leq \mathbb{E}\left[x_i^2\right] \leq \frac{k}{\delta \cdot (1+\epsilon)} < \frac{k}{\delta}$$

$$\mathbb{E}\left[\sum_i x_i\right] = \delta \cdot \frac{k}{\delta \cdot (1+\epsilon)} = \frac{k}{1+\epsilon}$$

Of particular interest is the expectation of the sums, which leads to the result:

$$Var\left(\sum_i x_i\right) = \delta \cdot var(x_i) = k$$

This allows us to use Chebyshev's inequality to prove our bound:

$$Pr\left[\sum_i x_i - \frac{k}{1+\epsilon} > 20k\right] \leq \frac{1}{20} = 0.05$$

$$\sum_i x_i > \frac{k}{1+\epsilon} + \sqrt{20k}$$

We observe that $\frac{k}{1+\epsilon} + \sqrt{20k}$ for $k = c \cdot \frac{1}{\epsilon^2}$ and large enough c.

$$\iff \pi\left[\sum_i x_i > k\right] \leq 0.05$$

□

3 Acknowledgements

In producing this document, we referenced the lecture notes, as well as the scribe notes from lecture 4 of the 2017 Advanced Algorithms course, as provided on the current course website.