CS 224: Advanced Algorithms Fall 2014

Lecture 4 — Thursday Sep 11, 2014

Prof. Jelani Nelson *Scribe: Marco Gentili*

1 Overview

Today we're going to talk about:

1. linear probing (show with 5-wise independence)
2. approximate membership (bloom filters)
3. cuckoo hashing
4. bloomier filters (generalization of bloom filters)
5. power of 2 choices

2 Linear Probing

Recap: We have an array of size $m = 2n$. I is an interval of array locations, and $L(I) = |\{j \in S : h(j) \in I\}|$. I is defined to be $full$ if $L(I) \geq |I|$.

Lemma 1. *If inserting x takes k probes, then $h(x)$ is contained in at least k full intervals.*

Proof. (From last time) Let us define z_1 the position of the first empty cell on the left of $h(x)$ and z_2 the position of the first empty cell on the right of $h(x)$. It is easy to see that each of the intervals $[z_1, h(x)], [z_1, h(x)+1], \ldots, [z_1, z_2 - 1]$ are full. There are $z_2 - h(x)$ such intervals. But by the assumption of the lemma, $z_2 = h(x) + k$ which conclues the proof of the lemma. □

From last class, we wanted to show that $\mathbb{E}[\text{query time}] = O(1)$ for a 7-wise independent family of hash functions.

$$\begin{aligned}\mathbb{E}[\#\text{ of probes to } \texttt{insert(x)}] &\leq \mathbb{E}[\#\text{ of full intervals containing } h(x)] \\ &\leq \sum_{k\geq 1} \sum_{\substack{|I|=k \\ h(x)\in I}} \mathbb{P}(I \text{ full}) \leq \sum_{k\geq 1} k \max_{\substack{|I|=k \\ h(x)\in I}} \mathbb{P}(I \text{ full})\end{aligned}$$

If we can show that $\max \mathbb{P}(I \text{ is full})$ is $O\left(\frac{1}{k^3}\right)$, then the sum will converge, and then $\mathbb{E}[\text{query time}] = O(1)$.

$$\begin{aligned}
\mathbb{P}(I \text{ full}) &= \mathbb{P}(L(I) \geq |I|) \\
&= \mathbb{P}(L(I) \geq 2\,\mathbb{E}[L(I)]) \\
&= \mathbb{P}((L(I) - \mathbb{E}[L(I)]) \geq \mathbb{E}[L(I)]) \\
&\leq \mathbb{P}(|L(I) - \mathbb{E}[L(I)]| \geq \mathbb{E}[L(I)]) \\
&= \mathbb{P}(|L(I) - \mathbb{E}[L(I)]|^6 \geq \mathbb{E}[L(I)]^6) \\
&\leq \mathbb{E}[|L(I) - \mathbb{E}[L(I)]|^6] / \mathbb{E}[L(I)]^6 (\text{Markov's Inequality})
\end{aligned}$$

The second equality holds since $\mathbb{E}[L(I)] = \frac{|I|}{2}$.

Let us define $X_1, \ldots, X_n$ with $X_i = 1$ if $h(i) \in I$, 0 otherwise. Then $L(I) = \sum_{i \in S} X_i$. Substituting above, we want to bound $\mathbb{E}[(\sum_i X_i - \mathbb{E}[\sum_i X_i])^6]$. The trick for this situation is symmetrization. But first, two things:

Definition 2. (Minkowski Distance) Define for r.v. X, $||X||_p = \mathbb{E}[|X|^p]^{1/p}$. This is a norm for $p \geq 1$.

Theorem 3 (Jensen's Inequality)**.** *If f is convex, then $f(\mathbb{E}[X]) \leq \mathbb{E}[f(x)]$*

Proof Using Symmetrization
So with the above definition, we're looking at $||\sum_i X_i - \mathbb{E}[\sum_i X_i]||_6$. We define X_i' as a random variable that is independent of X_i but is identical otherwise. We use the fact that $\mathbb{E}\sum X_i = \mathbb{E}\sum X_i'$.

$$\begin{aligned}
||\sum_i X_i - \mathbb{E}[\sum_i X_i]||_6 &= ||\mathop{\mathbb{E}}_{X'} \sum_i (X_i - X_i')||_6 \\
&= (\mathop{\mathbb{E}}_{X}(\mathop{\mathbb{E}}_{X'}(\sum (X_i - X_i'))^6))^{1/6}) \\
&\leq ||\sum_i (X_i - X_i')||_6 \text{ (Jensen – expectation over } X_i \text{ and } X_i') \\
&= ||\sum_i \sigma_i (X_i - X_i')||_6 \text{ ($\sigma_i = \pm 1$ are independent)} \\
&\leq 2||\sum_i \sigma_i X_i||_6 \text{ (Minkowski)}
\end{aligned}$$

$X_i = 1$ with probability $|I|/m = k/(2n)$ and 0 otherwise. Define p to be $k/(2n)$. Then we have $\mathbb{E}[(\sum_i \sigma_i X_i)^6] = \sum_{i_1,\ldots,i_6} \mathbb{E}[\sigma_{i_1} \ldots \sigma_{i_6}]\,\mathbb{E}[X_{i_1} \ldots X_{i_6}]$. The only monomials that matter are those with each index appearing an even number of times, because any odd power of σ_i has expectation 0.

The number of ways to select 1, 2, and 3 distinct indices in this way is n, $2n(n-1)$, and $n(n-1)(n-2)$, respectively. Thus, this is dominanted by the case of three distinct items appearing twice each, giving $O(n^3p^3) = O(k^3)$. Then

$$\mathbb{P}(I \text{ full}) \leq O(k^3)/(k/2)^6 = O(1/k^3)$$

Then going back to the expected number of probes, we have that this is $O(\sum_{k=1}^{\infty} k\frac{1}{k^3})$, which is constant.

In this proof we conditioned on x and then had products of 6 random variables which we expanded into the product of their individual probabilities. Thus, we needed $6 + 1 = 7$-wise independence. Now we will show that in fact 5 suffices.

5-wise independence

The main trick to show that 5-wise independence is sufficient is to reason more carefully so that we don't have the additional factor of k in our expression for $\mathbb{E}[\#$ of probes to `insert(x)`$]$. Pagh, Pagh, Ruzic [PPR07] showed that 5-wise works. Patrascu and Thorup [PT10] showed that 4-wise is not sufficient.

PPR Argument: We have an array of length $m = 2^k$ for some integer k. We build a perfect binary tree over the locations in the array. We imagine that every node in the binary tree corresponds to an interval of memory locations (namely everything in its subtree). We want to union-bound over a constant number of intervals rather than k.

Call a node *dangerous* if $\geq \frac{3}{4}2^h$ items hash there (height of node is h). We'd expect that a usual node would have $\frac{1}{2}2^h$ items hash there.

We adjust our lemma from before:

Lemma 4. *If we probe k slots in the array, that implies that there exists $\geq k$ full intervals containing $h(x)$ each of different length. Furthermore, the array cell immediately preceding each of these intervals is vacant.*

The proof is identical – just note that each of the k intervals constructed have different lengths because we constantly shift the right endpoint while leaving the left fixed. Furthermore the left endpoint of these intervals was chosen exactly as the point immediately after the rightmost empty cell left of $h(x)$.

Given $|I| = k$, look at the ≥ 4 and ≤ 9 nodes in the binary tree spanning I at level $h - 2$, where $k \in [2^h, 2^{h+1})$.

Look at the first 4 of these nodes. Call j^* the first array index in the interval. Cell j^* is in the first of the 4 nodes, and everything from that location onward amongst these 4 nodes must have been hashed to j^* or later (since the cell immediatley preceding j^* is vacant). Thus the first 4 nodes contain $\geq 3 \cdot 2^{h-2} + 1$ items that were hashed to j^* or later, which implies that one of these 4 has at least $> \frac{3}{4}2^{h-2}$ items hashed to it, meaning that it's *dangerous*.

All the intervals of length k containing $h(x)$ are contained in an interval of length $2k - 2$. So the total number of height $h-2$ nodes used to span these intervals is $O(1)$. If there exists a full interval of length k containing $h(x)$, then one of these $O(1)$ nodes of length $\geq k/8$ must be dangerous,

which implies that

$$\mathbb{P}(\exists \text{ full interval of length } k \text{ containing } h(x)) \leq O(1)\,\mathbb{P}(\text{length } k/8 \text{ interval is dangerous})$$

Define I' to be a length $k/8$ interval.

$$\begin{aligned}\mathbb{P}(\text{length } k/8 \text{ interval is dangerous}) &\leq \mathbb{P}(L(I)' - \mathbb{E}[L(I')] > 1/2\,\mathbb{E}[L(I')](|I'|/2)) \\ &\leq \frac{1}{(\frac{|I'|}{2})^4}\,\mathbb{E}[(L(I') - \mathbb{E}[L(I')])^4] \\ &= O(\frac{1}{k^2})\end{aligned}$$

Here we have conditioned on x, and the derivations follows the same symmetrization argument, except now we only need $4 + 1 = 5$-wise independence. Returning to the sum, we have that the expected number of probes is $O(\sum \frac{1}{k^2})$, which is constant. Thus 5-wise independence suffices.

3 Approximate Membership

3.1 Membership Problem

Set $S \subseteq [u]$, and we need to maintain S

- `update(x)`: $S \leftarrow S \cup \{x\}$
- `query(x)`: Is $x \in S$?

We want to use n bits of space for n words. We will allow false positives. More specifically, if $x \in S$, $\mathbb{P}(\texttt{query(x)} = 1) = 1$, and if $x \notin S, \mathbb{P}(\texttt{query(x)} = 1) \leq 1/2$.

Solution: Make a bit array of length $m = 2n$, where n is the number of keys (pretend this is a static problem for now). Let that array be A. Pick a hashfunction $h : [u] \to [m]$. Set $A[h(x)] = 1$ for $x \in S$.

If $x \notin S$, $\mathbb{P}(\texttt{query(x)} = 1) = \mathbb{P}(\exists y \in S \text{ s.t. } h(x) = h(y)) \leq \sum_{y \in S} \mathbb{P}(h(x) = h(y))$ by Union-Bound. $\mathbb{P}(h(x) = h(y)) = 1/m$, so $\mathbb{P}(\texttt{query(x)} = 1) \leq n\frac{1}{m} = 1/2$.

What if we want better bounds, say if $x \notin S$, then $\mathbb{P}(\texttt{query(x)} = 1) < \epsilon$. What we'll do is have many copies of bit arrays of length m. The number of copies is $\log(1/\epsilon)$, each with its own hashfunction. For insert, set the appropriate bit from each array to 1. For search, check if all the appropriate bits are 1. This requires space $O(n \log 1/\epsilon)$ bits and time $O(\log 1/\epsilon)$. This is called a Bloom filter [B70].

4 Cuckoo Hashing

A solution to the dynamic dictionary problem. Due to Pagh and Rodler [PR04]. The idea is to use two hash functions instead of just one.

Array A of length $m = c \cdot n$. We pick two hash functions g, h mapping $[u]$ to $[m]$. Imagine that they're perfect, totally random hash functions.

`insert(x)`:

- try put x in $A[g(x)]$ (call that location j)
- if some x' is there, then
- if $g(x') = j$, put x' in $h(x')$, else if $h(x') = j$, put x' in $g(x')$
- repeat this process until you either succeed or go on for $> 10 \log n$ steps, in which case rebuild the entire data structure

Goal: Show that the expected update time is $O(1)$.
Analysis: Main tool is cuckoo multigraph. m vertices and n edges. For each $x \in S$, draw edge between $g(x)$ and $h(x)$.

5 Bloomier Filters

What if we want to associate an r bit string with each key (retrieval problem)? Bloomier filters will be able to give you the r bit string associated with a key, but won't be able to tell you the keys from the Bloomier filter itself. Due to [CKRT04]. Solves r-bit retrieval problem.

To get Bloomier filter, create cuckoo hash table, and keep on recreating until we get a cuckoo graph with no cycles. Then the graph is a forest (bunch of trees). Each edge corresponds to an item. Say x *owns* its deeper node (root all the forests so that they have depth). At each node we'll store an r-bit string. Store 0 at the roots. Fill in r-bit strings in forest top down (every node gets a string).

`query(x)`: output $A[g(x)] \oplus A[h(x)]$
When filling in A top down, put whatever value in node v is necessary so that querying its owner is correct.

Thorup showed how to create k-wise independent families very quickly [T13]. Dietzfelbinger et al showed how to create 2-wise independent families using just bitshifting and multiplying.

References

[PPR07] Anna Pagh, Rasmus Pagh, and Milan Ruzic. Linear probing with constant independence. *STOC* , 39:318-327, 2007.

[PT10] Mihai Patrascu and Mikkel Thorup. On the k-independence required by linear probing and minwise independence. *ICALP*, pages 715–726, 2010.

[B70] Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. Commun. ACM 13(7): 422-426 (1970)

[PR04] Rasmus Pagh, Flemming Friche Rodler. Cuckoo hashing. J. Algorithms 51(2): 122-144 (2004)

[CKRT04] Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, Ayellet Tal. The Bloomier filter: an efficient data structure for static support lookup tables. SODA 2004: 30-39

[T13] Mikkel Thorup Simple Tabulation, Fast Expanders, Double Tabulation, and High Independence FOCS 2013: 90-99