COMS-E6998: Information Theory in Computer Science **Fall 2017**

Lecture 4: Cell-Sampling, Dictionaries and Succinct Data Stuctures

Lecturer: Omri Weinstein *Scribes: scribe-name1,2,3*

In thi lecture...

4.1 The Cell-Sampling Method

Today we will learn a new and elegant technique for proving cell-probe lower bounds. The crux of this method is viewing an efficient data structure D as a proxy for *compressing the database*. The idea is that Alice, who holds the DB $\mathcal{D}$ can use D to describe to Bob the DB by.... What is crucial for this method to work (and often most difficult to show) is that *every* query answered correctly provides Bob "1 *independent* bit of information" about $\mathcal{D}$. If this is the case, then we have...

We next show a clean application of this method to the *polynomial evaluation* problem, which along the way, proves the highest known lower bound for any static data structure problem.

4.1.1 A Lower Bound for Polynomial Evaluation

4.2 The Dictionary Problem

In the Dictionary problem $\mathsf{Dict}_{n,k,\Sigma}$, we are given a set $|S| = k$ of keys in a (large) universe $[n] = 2^w$, where each element is (possibly) associated with some data $\in |\Sigma| \leq 2^w$. The goal is to preprocess $S \in \Sigma^n$ so that for any $i \in U$, the data structure quickly returns $S[i]$. An interesting special case is the (sparse) *Membership* problem, where $\Sigma = \{0, 1\}$ and the goal is simply to preprocess a small subset $S \subset [n]$ so that membership queries $i \in^? S$ can be answered efficiently. (Note that without the sparsity assumption the problem is trivial when Σ is a power of 2. we will mention this issue later in the lecture).

The most naiive solution for this problem is to store the indicator vector of S as is, which would result in ideal query time ($t = 1$), but the space is very wasteful $s = n \gg k$. The standard solution is to store S in a sorted array, using $s = k$ cells, and given a query $i \in U$, to perform binary search, resulting in $t = \lg k$ time (which for typical values of k is $\tilde{O}(\lg m)$).

We now show a dictionary data structure based on *hashing* that achieves near-optimal performance: $s = O(k)$ space and $t = O(1)$ time (!).

Theorem 4.2.1 (Fredman-Kolmos-Szemeredi)**.** *There is a (Las-Vegas) dictionary data structure in the cell-probe model with word-size $w = \Theta(\lg n)$, that uses $s = O(k)$ space in expectation and $t = O(1)$ time.*

A natural first attempt is to try and map the universe $[n]$ to $\approx k$ buckets using a hash function and store each $i \in S$ in location $h(i)$, so that given $i \in [n]$, we just need to look up $h(i)$ and check whether i is indeed written in this cell. Two immediate problems arise: First of all, the hash function should have a compact representation, as otherwise computing $h(i)$ in query time is too expensive. The second (and related) point is that $h(\cdot)$ better have no (or few) collisions *w.r.t elements in S* – indeed, if the bucket $h(i)$ contains many other elements $j \in S$, storing all of them will be costly in reading time. These two requirements conflict, but it turns out that using a *two-level smashing* idea, we can achieve something weaker that nevertheless suffices: A compactly-described random hash function whose collisions are well "spread out" :

Claim 4.2.2. *If $h \in_R \mathcal{H}$ where $\mathcal{H}$ is a 2-wise independent hash family mapping $[m]$ to $[n]$, and $B_i := \{j \in S : h(j) = h(i)\}$, then $\mathbf{E}_h[$total number of collisions$] = \mathbf{E}_h[\sum_i |B_i|^2] = O(k)$.*

Proof. Let $\mathbf{1}_{i,j}$ denote the indicator r.v of the event that for a randomly chosen $h \in_R \mathcal{H}$, $h(i) = h(j)$, for $i, j \in S$. Since the total number of collisions is precisely $\sum_{i\in[n]|B_i|^2}$, we have

$$\mathbf{E}_h\left[\sum_i |B_i|^2\right] = \mathbf{E}_h[\text{total number of collisions}] = n + \sum_{i,j\in S, i\neq j} \Pr_h[h(i) = h(j)] \leq k + \binom{k}{2}/k = O(k).$$

□

Since there are few collisions in each bucket in expectation, we can now afford to hash each bucket B_i to a universe of size $O(|B_i|^2)$ using *another independently chosen* 2-wise independent hash function $h_i : B_i \mapsto [8|B_i|^2]$. By the bday paradox, $\Pr_{h_i}[\exists \text{collision}] \leq \binom{|B_i|}{2} \cdot 1/8|B_i|^2 \leq 1/4$. Therefore, conditioned on a "good" h, we can find *perfect* hash functions $\{h_i\}_{i\in[n]}$ in the *second level.* The total space usage (in expectation) is $\mathbf{E}_h[\sum_i 8|B_i|^2] = O(k)$, and the query time is $O(1)$, since for any given $s \in U$m we only need to compute $h(x) := i$ and h_i, which takes $O(1)$ operations (with word size $w = O(\lg n)$), due to the succinct representation of 2-wise independent hash functions (= drawing a, b using $O(\lg n)$ bits each). We now simply check whether $h_i(x) =^? x$ and return (the pointer to the data associated with) x iff this is true.

Dynamic Dictionaries. Cuckoo Hashing and "Bloomier Filters"...

4.2.1 Succinct Dictionaries and Memberships

FKS's data structure seems as good as we can hope for (linear space, constant time). Alas, in some applications, it might not be useful or practical (not that we are talking about $O(n)$ *cells* $= O(n \lg n)$ bits in total, which for the Membership problem might still be a huge overhead in industrial scale). *Succinct data structures* are allowed to store only a small *additive* overhead beyond the information-theoretic minimum, which is $H := \lg \binom{n}{k}$ *bits* in case of the (sparse) Membership problem. Formally, we want to store $H + r$ *bits* (packed into cells of word size w as before), and recover $x \in^? S$ in t probes as before. (Btw, note that even for the non-sparse case $S \in \Sigma^n$ the problem is not trivial when $\Sigma \neq$ power of 2. Mention [DPT]). Hence we wish to understand the tradeoff between r and t.

We can achieve $s \leq H + 1$ bits of space using Huffman (or arithmetic coding), but of course the problem is that in order to read x_i we need to read the *entire* memory, hence $t \gtrsim H/w$ (which is bad for the natural case where $k \geq n/poly \lg n$). An intermediate solution is to use compression on *blocks*, by hoping the n coordinates into chunks of size n/t. This results in a *linear tradeoff* between time and redundancy $r \approx n/t$ (since Huffman can lose 1 bit per block, and we have n/t blocks). Can we do better? ...

Theorem 4.2.3 (Patrascu's "Succincter")**.** *...*

4.2.1.1 A lower bound using cell-sampling

Can we do better ? ...

4.2.2 Dictionaries with Priors: Locally Decodable Source Coding for Correlated Files

Consider the realistic scenario...

4.2.2.1 Research and implementation projects for LDSC