CSG399: Gems of Theoretical Computer Science. Lectures 23-24. Apr. 3-7, 2009
Instructor: Emanuele Viola Scribes: Dimitrios Kanoulas & Eric Miles

Succinct Data Structures

Consider the following problem: we are given n ternary elements $\langle t_1, t_2, \ldots, t_n \rangle \in \{0,1,2\}^n$ and we want to represent them using b bits so that (1) each element $t_i \in \{0,1,2\}$ can be recovered by reading as few bits as possible and (2) b is close to the information-theoretic minimum $(\log_2 3)n$: $b = (\log_2 3)n + r$ where r is some small *redundancy*.

- The simplest solution to this problem is to use 2 bits per t_i. With such an encoding we can retrieve each $t_i \in \{0,1,2\}$ by reading just 2 bits (which is optimal). The space used is $b = 2n$ and we have linear redundancy.
- Another solution to this problem is what is called *arithmetic coding*: we think of the concatenated elements as forming a ternary number between 0 and $3^n - 1$, and we write down its binary representation. To retrieve t_i we need to read all the $b = O(n)$ bits, but the space needed is only $b = \lceil n(\log_2 3) \rceil \leq n(\log_2 3) + 1$, i.e. we have redundancy 1 (which is optimal).

These solutions represent the two extremes for solving the problem. The first one optimizes the retrieval time but not the redundancy, and the second optimizes the redundancy but not the retrieval time.

A polynomial tradeoff. To trade between these two extremes, we can group the t_i's into blocks. If we take blocks of t ternary elements and encode each block with arithmetic coding, the retrieval time will be $O(t)$ bits and the needed space will be $(n/t)\lceil t(\log_2 3) \rceil \leq n(\log_2 3) + n/t$ (assuming t divides n). Thus block-wise arithmetic coding gives a *polynomial* trade-off between retrieval time and redundancy. (Using number-theoretic results on logarithmic forms, one can show that this last inequality is tight up to changing n/t into $n/t^{\Theta(1)}$, so indeed this approach gives a polynomial tradeoff.)

We now present a recent work by Pătraşcu, later refined with Thorup, giving an *exponential* trade-off: retrieval time $O(t)$ bits and redundancy $n/2^t + O(1)$. In particular, if we set $t = O(\log n)$, we get retrieval time $O(\log n)$ and redundancy $O(1)$. Moreover, the bits read are all consecutive, so this can be implemented in the RAM model with cells of size $O(\log n)$ to obtain retrieval time $O(1)$ cells and redundancy $O(1)$.

1 An exponential trade-off between retrieval time and redundancy

Definition 1 (Encoding and redundancy). *An encoding of a set A into a set B is a one-to-one (a.k.a. injective) map $f : A \to B$. The* redundancy *of the encoding f is* $\log_2 |B| - \log_2 |A|$.

The following lemma gives the building-block encoding we will use.

Lemma 2. *For all sets X and Y, there is an integer b, a set K and an encoding $f : (X \times Y) \rightarrow \left(\{0,1\}^b \times K\right)$ such that the following four properties hold:*

1. *f has redundancy $\leq c/\sqrt{|Y|}$,*
2. *$x \in X$ can be recovered just by reading the b bits in $f(x,y)$,*

where c is a constant independent of X, Y.

Note that Property (1) says that $b + \log|K| - \log|X| - \log|Y| \leq c/\sqrt{|Y|}$. For Property (2) to hold we must have $b \geq \log|X|$. Combining this with the previous expression we obtain $\log|K| - \log|Y| \leq c/\sqrt{|Y|}$. In particular we get that $|K| \leq 2^c \cdot |Y|$ (in fact it will be the case that $|K| \leq c \cdot \sqrt{|Y|}$, but the looser bound is sufficient).

The basic idea for proving the lemma is to break Y into $C \times K$ and then encode $X \times C$ by using b bits:

$$X \times Y \rightarrow X \times C \times K \rightarrow \{0,1\}^b \times K.$$

There is however a subtle point. If we insist on always having $|C|$ equal to, say, $\sqrt{|Y|}$ or some other quantity, then one can cook up sets that make us waste a lot (i.e., almost one bit) of space. The same of course happens in the more basic approach that just sets $Y = K$ and encodes all of X with b bits. The main idea will be to "reason backwards," i.e., we will first pick b and then try to stuff as much as possible inside $\{0,1\}^b$. Still, our choice of b will make $|C|$ about $\sqrt{|Y|}$.

Proof. Pick any two sets X and Y, where $|Y| > 1$ without loss of generality. Define $b := \left\lceil \log_2\left(|X| \cdot \sqrt{|Y|}\right)\right\rceil$, and let $B := \{0,1\}^b$. To simplify notation, define $d := 2^b/|X| = \Theta\left(\sqrt{Y}\right)$.

How much can we stuff into B? For a set C of size $|C| = \lfloor |B| / |X| \rfloor$, we can encode elements from $X \times C$ in B. The redundancy of such an encoding can be bounded as follows:

$$\begin{aligned}
&\log|B| - \log|X| - \log|C| = \log\frac{2^b}{|X|} - \log\left\lfloor\frac{2^b}{|X|}\right\rfloor \\
=\ & \log d - \log\lfloor d\rfloor \leq \log d - \log(d-1) = \log\left(1 + \frac{1}{d-1}\right) \\
\leq\ & O\left(\frac{1}{d-1}\right) \qquad \text{(because for every } x \in \mathbb{R},\ (1+x) \leq e^x\text{)} \\
\leq\ & O\left(\frac{1}{\sqrt{|Y|}-1}\right) \leq O\left(\frac{1}{\sqrt{|Y|}}\right).
\end{aligned}$$

To calculate the total redundancy, we still need to examine the encoding from Y to $C \times K$. Choose K of size $|K| = \lceil |Y| \, / \, |C| \rceil$, so that this encoding is possible. With a calculation similar to the previous one, we see that the redundancy is:

$$\begin{aligned}
\log|C| + \log|K| - \log|Y| &= \log\left(\left\lceil\frac{|Y|}{|C|}\right\rceil\right) - \log\left(\frac{|Y|}{|C|}\right)\\
&\leq \log\left(1+\frac{|C|}{|Y|}\right) \leq O\left(\frac{|C|}{|Y|}\right) \leq O\left(\frac{\left\lfloor\frac{2^b}{|X|}\right\rfloor}{|Y|}\right) \leq O\left(\frac{2^b}{|X|\cdot|Y|}\right)\\
&\leq O\left(2^{(\log|X|\cdot\sqrt{|Y|})+1}/(|X|\cdot|Y|)\right) \leq O\left(\frac{\sqrt{|Y|}}{|Y|}\right) = O\left(\frac{1}{\sqrt{|Y|}}\right).
\end{aligned}$$

The total redundancy is $O\left(1/\sqrt{|Y|}\right) + O\left(1/\sqrt{|Y|}\right) = O\left(1/\sqrt{|Y|}\right)$, which gives Property 1.

For Property 2, it is clear from the construction that any $x \in X$ can be recovered from the element of B only. □

From this building block, we can now construct the encoding.

Theorem 3. *Any n ternary elements $\langle t_1, t_2, \ldots, t_n\rangle \in \{0,1,2\}^n$ can be encoded by $n(\log_2 3) + n/2^t + O(1)$ bits in such a way that any t_i can be retrieved by reading $O(t)$ bits. This holds for any t; in particular, we can encode the ternary elements using $n(\log_2 3) + O(1)$ bits with retrieval time $O(\log n)$.*

Proof. Break the ternary elements into blocks of size t: $\langle t'_1, t'_2, \ldots, t'_{n/t}\rangle \in T_1 \times T_2 \times \ldots \times T_{n/t}$, where $T_i = \{0,1,2\}^t$ for all i. The encoding, illustrated in Figure 1, is constructed as follows, where we use f_L to refer to the encoding guaranteed by Lemma 2.

1. Compute $f_L(t'_1, t'_2) = (b_1, k_1) \in B_1 \times K_1$.

2. For $i = 2, \ldots, n/t - 1$ compute $f_L(k_{i-1}, t'_{i+1}) := (b_i, k_i) \in B_i \times K_i$.

3. Encode $k_{n/t-1}$ in binary as $b_{n/t}$ using arithmetic coding.

The final encoding is $(b_1, b_2, \ldots, b_{n/t})$. We now compute the redundancy and retrieval time.

Redundancy: From Property 1 of Lemma 2, the first $n/t - 1$ encodings have redundancy $O\left(3^{-t/2}\right) = 1/2^{\Omega(t)}$. For the last (arithmetic) encoding, the redundancy is at most 1. So the total redundancy is at most $\left(\frac{n}{t} - 1\right) \cdot \frac{1}{2^{\Omega(t)}} + 1 = \frac{n}{2^{\Omega(t)}} + O(1)$. One can visualize this as a "hybrid argument" transforming a product of blocks of ternary elements into a product of blocks of binary elements, one block at the time.

Retrieval Time: Say that we want to recover some t_j which is in block t'_i. To recover block t'_i, Lemma 2 guarantees that we only need to look at b_{i-1} and b_i. This is because k_{i-1} can be recovered by reading only b_i, and t'_i can be recovered by reading k_{i-1} and b_{i-1}. Thus to complete the proof it suffices to show that each b_i has length $O(t)$.

This is not completely obvious because one might have thought that the K_i become larger and larger, and so we apply the lemma to larger and larger inputs and the B_i get large too. However, recall that each $|K_i| \ll O(|T_i|) = O(3^t)$ from the comment after the statement of Lemma 2. Hence, every time we apply the lemma on an input of size at most $s \leq 3^{O(t)}$. Since the lemma wastes little entropy (Property 1 in Lemma 2), none of its outputs can be much larger than its input, and so $|B_i| = 2^{O(t)}$.

□

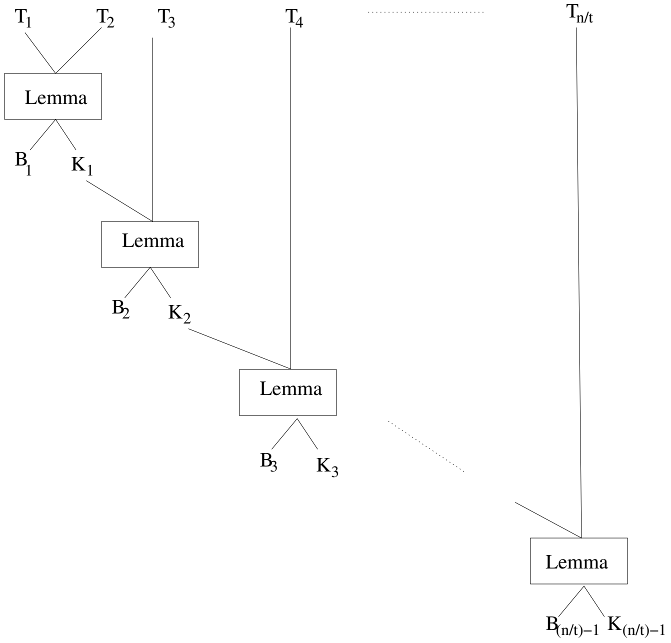

Figure 1: Succinct Encoding

Notes revised on October 6, 2009.
Minor revision on August 10, 2015.