Smaller and Faster: Succinct Data Structures

Session 1: Intro, Bit Vectors

Jérémy Barbay
(some figures by Gonzalo Navarro)

Structure of the Course

1. Basic Concepts; Bit Vectors and Static Dictionaries;
2. Trees; Graphs;
3. Permutations; Functions;
4. Strings; Binary Relations;
5. Labeled Objects; Overview;

Talk From Sorting to Compression

Foreword

Warning

- Not a specialist.
- Not technically inclined.
- Five Sessions is very short.

Why do I give those lectures?

- Teaching = learning better.
- Did not find a good (English) tutorial.
- Succinct Data Structures get to be practical.

What you should get from it

- List of useful references;
- Selection of technical examples;
- Some general culture about techniques to apply to your problem.

Succinct Data Structures
Why?

(Old) Secondary Memory arguments

- Machines run faster,
- Memory gets larger,
- Access gets slower.

More specifically

- Lower bounds (e.g. Beame and Fich (1999))
- Provably optimal (e.g. Golynski (2007))

Succinct Data Structures
How?

- An object chosen among N is optimally encoded by $\lceil \lg N \rceil$ bits, but rarely usable as such (e.g. trees).
- We explore the relation between:
  - Space to code;
  - Space to represent;
  - Time to precompute the representation;
  - Time to navigate and search.
- What is the smallest amount of space and time needed to precompute a representation allowing to navigate and search such an object?

Succinct Data Structures
Example: ordinal trees

- $2n - \Theta(\lg n)$ bits to code an ordinal tree;
- Using pointers
  - $4n \lg n \approx 128 \times n$ bits represents
  - and navigates it in constant time,
  - with $\mathcal{O}(n)$ preprocessing time.
- Using techniques from [Sadakane (2008)]
  - $2n + n/\lg n \in 2n + o(n)$ bits ($< 3n$ in practice) represents
  - and navigates and searches it in constant time,
  - with $\mathcal{O}(n)$ preprocessing time.

(All results in a $\theta(\lg n)$ bits architecture.)

Succinct Data Structures
What?

- An object chosen among N is optimally coded by $\lceil \lg N \rceil$ bits, but rarely usable as such (e.g. trees).
- What is the smallest amount of space and time needed to represent, navigate and search such an object?
- Several type of answers:
  - A Succinct Encoding uses $\lceil \lg N \rceil + o(\lg N)$ bits for everything;
  - A Succinct Index uses $o(\lg N)$ bits and access to the unmodified data;
  - An Ultra-Succinct Encoding uses $H(I) + o(\lg N)$ bits to compress the data;
  - A Self-Index uses $\mathcal{O}(H(I))$ bits to compress and index the data;

Note that:

- Succinct Encoding $\Leftarrow$ Succinct Index $\Leftarrow$ Compressed Index;
- Succinct Encoding $\Leftarrow$ Compressed Self Index.

Dictionary ADT (from CS240, Waterloo)

- Container of key-element pairs
- Required operations:
  - `insert( k,e )`,
  - `remove( k )`,
  - `find( k )`,
  - `isEmpty()`.
- When an order is provided:
  - `closestKeyBefore( k )`,
  - `closestElemAfter( k )`,
  - `rank(k)`: nb. of values smaller than k in the set,
  - `select(r)`: r-th smallest number in the set.

Note: No duplicate keys

Specific Dictionary ADTs and their DS

Notes on Computation Model

- Dictionaries and Sets are implemented (almost) identically.
- Focus on static data structures and operators
  - `closestKeyBefore( k )`, `closestElemAfter( k )`
  - `rank(k)`, `select(r)`.
- Advanced implementations address the other operations.
- We will consider the $\Theta(\lg n)$-RAM model, where operations on $\Theta(\lg n)$ bits are supported in constant time.

Dictionary as Bit Vector

String Succinct Encodings support

- `bin_rank`(α, x): nb. of α-occurrences before pos. x;
- `bin_select`(α, r): position of r-th α-occurrence.

Example:

0	0	0	1	0	0	0	1	0	0

- `bin_rank`$(1, 6) = ?1$
- `bin_select`$(1, 2) = ?8$

General Technique

Summaries of Summaries

- Precompute (and code) the function for a small selection of values;
- For some in-between values, code the difference;
- For the smallest sequences of left in-between values, use a greedy approach.

Bender and Farach-Colton (2004) gave a very nice writing of the technique. Even though they were not aiming at succinct data structure but rather small precomputing time, they achieve the same goal, and explain it in a nice incremental way.

Specific Technique

Basics

- Binary Rank through recursive definition of the bit vector in superblocs and blocs;
- Binary Select through recursive definition of the parameter space in sparse and dense blocs.

Further Improvements

- Raman et al. (2002) showed how to compress to the binary entropy.
- Golynski (2006) reduced the space required through the common use of counting index.

Binary Entropy

if there are n_0 zeroes and n_1 ones in a bit vector B ($n_0 + n_1 = n = |B|$)

$$\begin{aligned}\mathcal{H}_0(B) &= \frac{n_0}{n}\lg\frac{n}{n_0} + \frac{n_1}{n}\lg\frac{n}{n_1} \\ &= \frac{1}{n}\lg\binom{n}{n_0} + O\left(\frac{\lg n}{n}\right)\end{aligned}$$

Homework

Balanced Parenthesis: (()())

- How does this relate to bit vectors?
- What kind of operators would you support on it?
- How do you support them?

Ordinal trees

- How does this relate to BP?
- What kind of operators would you support on it?
- How do you support them?

Bibliography

Beame, P. and Fich, F. E. (1999). Optimal bounds for the predecessor problem. In *Proceedings of the 31st annual ACM symposium on Theory of computing*, pages 295–304.

Bender, M. A. and Farach-Colton, M. (2004). The level ancestor problem simplified. *Theor. Comput. Sci.*, 321(1):5–12.

Golynski, A. (2006). Optimal lower bounds for rank and select indexes. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*.

Golynski, A. (2007). *Upper and Lower Bounds for Text Indexing Data Structures*. PhD thesis, University of Waterloo.

Raman, R., Raman, V., and Rao, S. S. (2002). Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete algorithms*, pages 233–242.