Karlsruhe Institute of Technology

Advanced Data Structures

Lecture 12: Sparse Sets and Variable Bit-Length Arrays

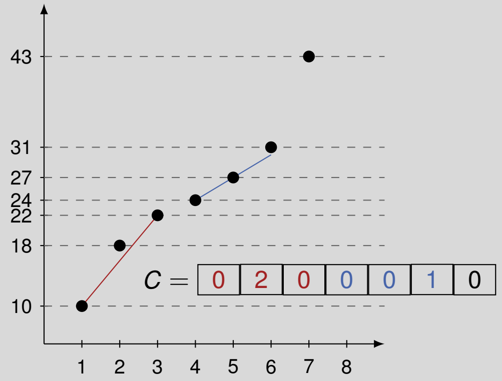
Florian Kurpicz

Recap: Learned Data Structures

- use piece-wise linear approximation
- store corrections
- compress everything

Open Questions

- are y-intersections monotonic increasing
- are $\log u + \log n$ bits enough to store slope



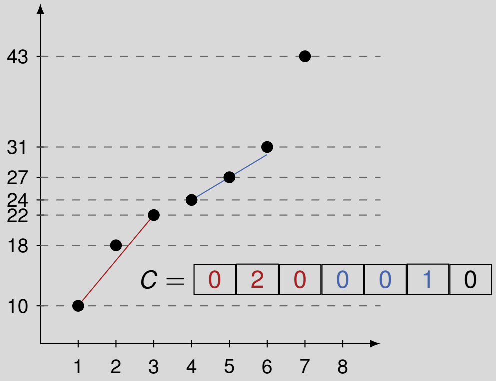
Recap: The PGM-Index [FV20]

- now S has to be stored
- how do we access elements in S
  - e.g., predecessor
- trick used before requires too much space

- store *key* instead position
- recurs on first *keys* of each segment

For Queries

- $\epsilon = \Theta(B)$
- load $2\epsilon + 1$ blocks per level



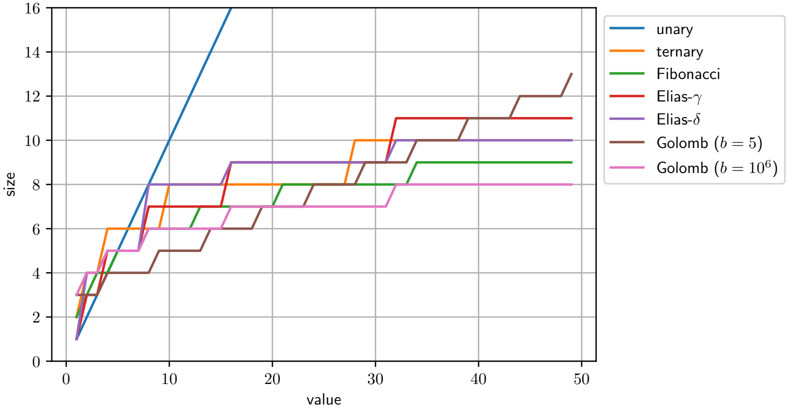
- $S = \langle 10, 18, 22, 24, 27, 31, 43 \rangle$

Variable Bit-Length Arrays

- not all elements require the same space
- arrays with *w* bits per element can waste space
- e.g., integers can be encoded with space proportional to their size

Definition: Variable Bit-Length Data

Let $a[1..n]$ be an array containing entries of size $|a[i]|$ bits for $i \in [1, n]$.



Sampling (1/2)

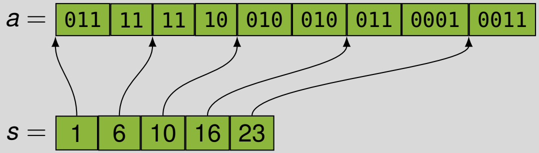
- encode a using close to $N = \sum_{i=1}^{n} |a[i]|$ bits

Definition: Sampling

Sample the starting position of every k-th element in array s.

Lemma: VLA with Sampling

Using sampling, storing a requires $N + O(n \log N/k)$ bits of space. Accessing a single element requires $O(k)$ time.



- space can be reduced using Elias-Fano coding
- access time depends on input size unless $k = O(1)$

Sampling (2/2)

Karlsruhe Institute of Technology

Definition: Two-Level Sampling

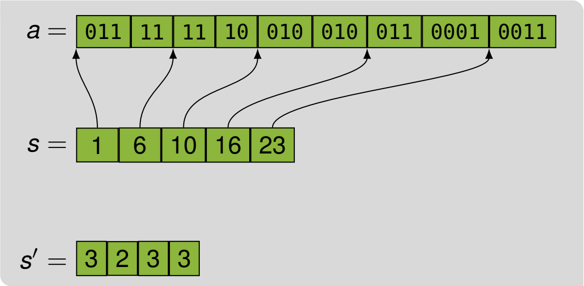
In addition to sampling every k-th element, also sample the offset to the closest preceding sampled element for each non-sampled element.

Lemma: VLA with Two-Level Sampling

Using two-level sampling, storing a requires $N + O(n \log N/k)$ bits of space for the first level and additional

$$n \cdot \max_{i \in \{0,k,2k,\dots\}} \left\lceil \log \sum_{j=1}^{k-1} |a[i+k]| \right\rceil$$

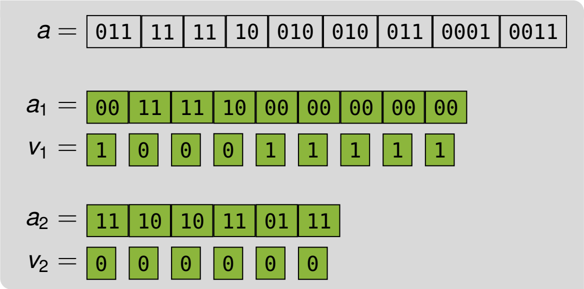
bits of space for the second level.



- for elements of polylogarithmic size, this means $O(n \log \log n)$ additional bits of space
- constant access time
- example on the board

Directly Addressable Codes (1/2) [BLN09]

- now, encode problem instead of indexing
- partition variable bit-length elements
- mark if not last partition
- similar to VByte encoding [WZ99]



Definition: Directly Addressable Codes

Each element is partitioned into length-ℓ slices. Every elements k-th (fixed-length) slice is stored in a_k. Use bit vector v_k to mark elements that continue in a_{k+1}.

Directly Addressable Codes (2/2)

Karlsruhe Institute of Technology

Lemma: VLA with Directly Addressable Codes

Using Directly Addressable codes, storing a requires at most $\ell n + N/\ell$ bits of space.

Proof (Sketch)

- at most $\ell - 1$ bits wasted in first slice
- one bit needed to mark each slice

- can be made more space-efficient
- choose different partition size for each level

$a =$ `011 | 11 | 11 | 10 | 010 | 010 | 011 | 0001 | 0011`

$a_1 =$ `00 | 11 | 11 | 10 | 00 | 00 | 00 | 00 | 00`

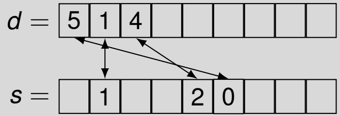
$bv_1 =$ `1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1`

$a_2 =$ `11 | 10 | 10 | 11 | 01 | 11`

$bv_2 =$ `0 | 0 | 0 | 0 | 0 | 0`

An Efficient Representation of a Sparse Set [BT93]

- represent a sparse (dynamic) set $S \subseteq [1, u]$
- using bit vector
  - u bits of space
  - iterating, clearing, comparing requires $|S|$ select queries
  - inserting requires rebuilding select support
  - without select support $O(u)$ time operations
- use custom representation



- double the space for efficient operations

Definition: Sparse Set Representation

The sparse set consists of a *dense* set d and a *sparse* set s. Let S contain n elements. To insert $i \notin S$ in S, set $d[n] = i$ and $s[i] = n$.

Operations on the Sparse Set

Karlsruhe Institute of Technology

insert(i)

- $d[n] = i$
- $s[i] = n$
- n++

is_in_set(i)

- return $s[i] < n$ and $d[s[i]] == i$

iterate

- for i in $1..n$
  - yield $d[i]$

clear

- $n = 0$

remove_from_set(i)

- if not is_in_set(i)
  - return
- $tmp = d[n - 1]$
- $d[s[i]] = tmp$
- $s[tmp] = s[i]$
- n–

Recap: Advanced Data Structures

- bit vectors with rank and select support
- succinct trees ℹ LOUDS, BP, DUFUDS
- succinct planar graphs
- predecessor data structures ℹ Elias-Fano, y-fast trie
- range minimum queries ℹ three solutions
- persistent data structures ℹ partial and full persistence
- orthogonal range search ℹ kd-trees, range trees, layered range trees
- binary space partition ℹ BSP-tree
- PaCHash
- compressed suffix array ℹ Elias-Fano with quotenting and recursive
- String B-trees
- retroactive data structures ℹ decomposable search problems, partial retroactive PQs
- minimal perfect hashing ℹ BDZ, CHD, RecSplit
- learned data structures ℹ encoding and indexing
- sparse sets and variable bit-length arrays

Preparation Oral Exam

everybody can choose first topic

Now, some examples

Bibliography I

Karlsruhe Institute of Technology

[BLN09] Nieves R. Brisaboa, Susana Ladra, and Gonzalo Navarro. "Directly Addressable Variable-Length Codes". In: *SPIRE*. Volume 5721. Lecture Notes in Computer Science. Springer, 2009, pages 122–130. DOI: 10.1007/978-3-642-03784-9_12.

[BT93] Preston Briggs and Linda Torczon. "An Efficient Representation for Sparse Sets". In: *LOPLAS* 2.1-4 (1993), pages 59–69. DOI: 10.1145/176454.176484.

[FV20] Paolo Ferragina and Giorgio Vinciguerra. "The PGM-index: A Fully-Dynamic Compressed Learned Index with Provable Worst-Case Bounds". In: *Proc. VLDB Endow.* 13.8 (2020), pages 1162–1175. DOI: 10.14778/3389133.3389135.

[WZ99] Hugh E. Williams and Justin Zobel. "Compressing Integers for Fast File Access". In: *Comput. J.* 42.3 (1999), pages 193–201. DOI: 10.1093/COMJNL/42.3.193.