Optimal Encodings for Range Majority Queries

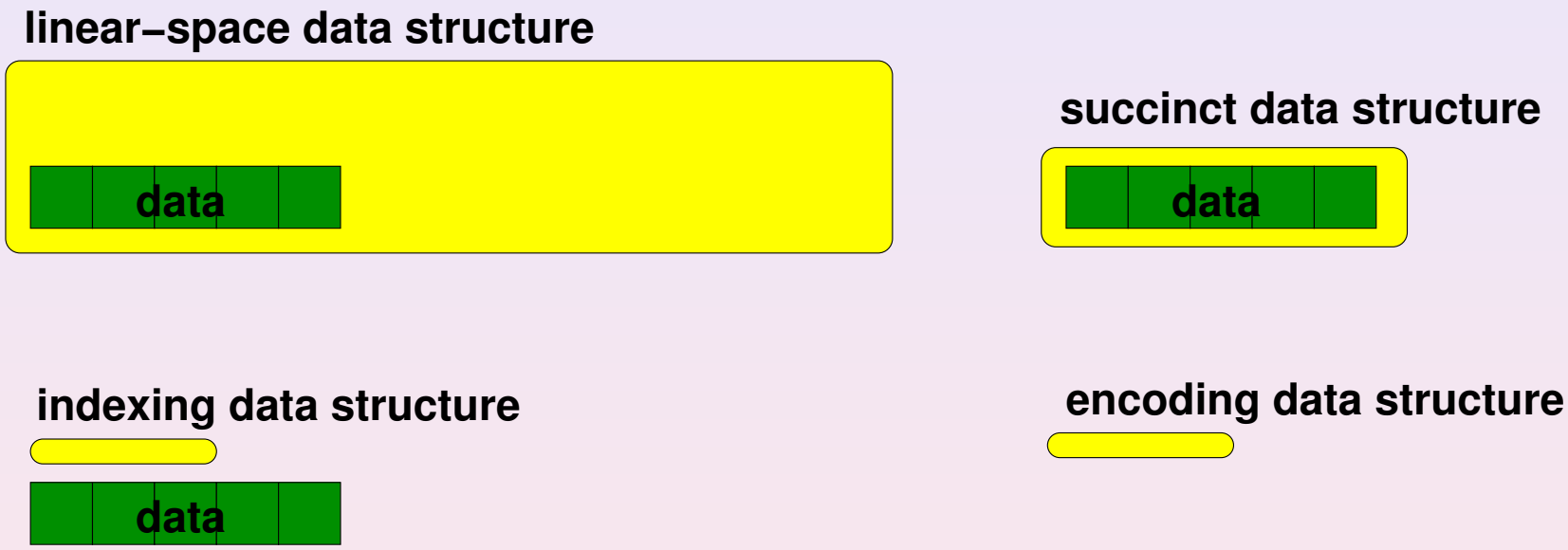
Gonzalo Navarro Sharma V. Thankachan

Gonzalo Navarro
Department of Computer Science
University of Chile

What are encoding data structures?

- Let $|data|$ be the worst-case entropy of the data, in bits.
- A linear-space data structure uses $O(|data|)$ bits (and even words).
- A succinct data structure uses $|data| + o(|data|)$ bits.
- An index data structure uses $o(|data)|$ bits, but needs access to the raw data to operate.
- An encoding data structure uses $o(|data|)$ bits, and answers queries without accessing the data.

What are encoding data structures?



Why are they interesting?

- The data can be deleted or stored in slower storage for sporadic use.
- An encoding can use less space than the entropy of the data! $o(|data|)$
- The structure can work at faster levels of the memory hierarchy, avoid I/Os and communication.
- Fewer computers, communication, and energy use in distributed deployments.

Best-known example: RMQs on arrays

[J. Fischer, Optimal succinctness for range minimum queries, LATIN 2010]

What new interesting questions they pose?

- In general, reduced-space data structures contain all the data, so the interesting questions on them is: how much time is needed to answer queries using how much space?
- Encoding data structures pose, in addition, another interesting question: how much space is needed to answer the desired queries, regardless of the time used?
- This is called the effective entropy of the encoding data structure, and is the minimum space it can use.
- It can be seen as the entropy of an ADT.

[M.J. Golin, J. Iacono, D. Krizanc, R. Raman, and S.S. Rao, Encoding 2D range maximum queries, ISAAC 2011]

Majority Queries

- Let $A[1, n]$ be an array of arbitrary elements and $0 < \tau < 1$.
- A τ-majority in a range $A[i, j]$ is an element that occurs more than $\tau(j - i + 1)$ times in $A[i, j]$.
- Problem:
  - Preprocess $A[1, n]$.
  - At query time, an interval $[i, j]$ and a value τ are given.
  - Return one position of each τ-majority in $A[i, j]$, or say there are none.

Example: Data mining

- Suppose you have a log of activities (Web page accesses, words in tweets, search keywords, etc.).
- You want to determine if there are trendy pages or words in a given week, or day, or time period.
- What you want is to find out whether there is some page or word that is mentioned more than a fraction τ of the times in a range of the log file.
- It may be that usually there is no trendy page or term; only when there is one, you want to examine the data more closely, retrieving it from a slower storage.

The non-encoding scenario

- Belazzougui et al. solved the problem in linear space and $O(1/\tau)$ time, for τ fixed at indexing time.
- They also achieved succinct space and $O((1/\tau)\log\log n)$ time, for variable τ.

[D. Belazzougui, T. Gagie, and G. Navarro, Better space bounds for parameterized range majority and minority, WADS 2013]

This paper: The encoding scenario

- One cannot give the majorities, but only their positions.
- We show that $\Omega(n)$ bits are required to list an arbitrary position of each τ-majority.
- We show that $\Omega(n \log(1/\tau))$ bits are required to list a fixed position or all the positions of each τ-majority.
- Therefore, the problem is interesting for $\log(1/\tau) = o(\log n)$.
- No interesting encoding can work for arbitrary τ values.

[CPM 2014 and arXiv 1404.2677]

This paper: The encoding scenario

- We work in the RAM model with word size $w = \Theta(\log n)$ bits.
- We give an encoding that uses the optimal space $O(n \log(1/\tau))$ bits for τ fixed at indexing time.
- It finds a fixed position of each τ-majority in time $O((1/\tau) \log \log_w(1/\tau) \log n)$.
- For $1/\tau = O(\text{polylog}\, n)$, we reach the optimal time $O(1/\tau)$.
- The structure can be build in time $O(n \log n)$.
- It also solves queries for any $\tau' \geq \tau$, but in the same time.

[CPM 2014 and arXiv 1404.2677]

The lower bound

- Consider encoding a permutation $x_1, \ldots, x_k$.
- Set $\tau = 1/(2k)$ and

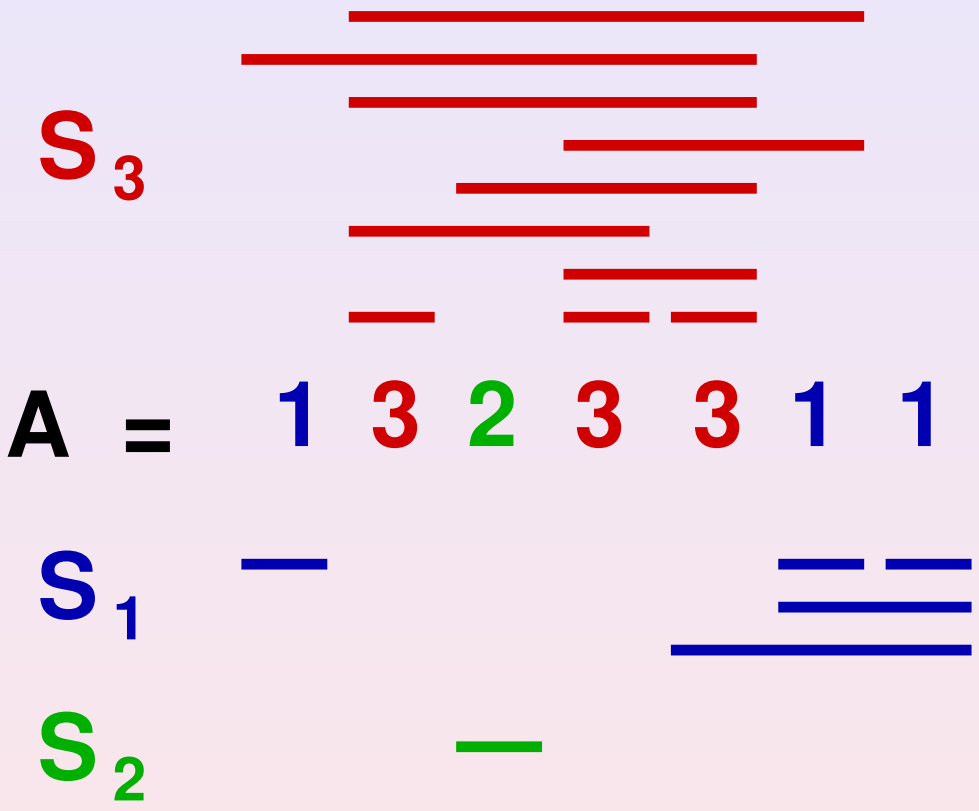
$$A[1, 3k] = -1, -2, \ldots, -k, \quad 1, 2, \ldots, k, \quad x_1, x_2, \ldots, x_k$$

- Then
  - $A[1, 2k]$ has no τ-majorities.
  - $A[2, 2k+1]$ has one τ-majority at position $k + x_1$.
  - $A[3, 2k+2]$ has two τ-majorities at $k + x_1$ and $k + x_2$, etc.
- Then m permutations, requiring $m \lg k!$ bits, are recovered with $\tau = \frac{1}{2k}$-majorities on an array $A[1, n = 3mk]$.
- Since $m \lg k! = \Omega(mk \log k)$, the encoding needs $\Omega(n \log(1/\tau))$ bits.

Data structure

- Let S_x be the set of segments in $[1, n]$ where element x is a τ-majority.
- Let $A_x[1, n]$ be a bitmap so that $A_x[i] = 1$ iff i is in some segment of S_x.
- Let M_x be a bitmap aligned to the 1s in A_x, marking the positions equal to x in A.
- That is, if $A_x[i] = 1$, then $M_x[rank(A_x, i)] = 1$ iff $A[i] = x$.
- Then x is a τ-majority in $A[i, j]$ if $A_x[i, \ldots, j] = 1$ and 1 is a τ-majority in $M_x[rank(A_x, i), rank(A_x, j)]$.
- Therefore, bitmaps A_x and M_x are sufficient to answer any range τ-majority query in constant time.
- The problem is that they add up to $O(n^2)$ bits.

Example for $\tau = 1/2$



Example for $\tau = 1/2$

$$A_3 \quad 1\ 1\ 1\ 1\ 1\ 1\ 0$$

$$A = \quad 1\ 3\ 2\ 3\ 3\ 1\ 1$$

$$A_1 \quad 1\ 0\ 0\ 0\ 1\ 1\ 1$$

$$A_2 \quad 0\ 0\ 1\ 0\ 0\ 0\ 0$$

Example for $\tau = 1/2$

A_3	1	1	1	1	1	1	0
M_3	0	1	0	1	1	0	0
A =	1	3	2	3	3	1	1
A_1	1	0	0	0	1	1	1
M_1	1	0	0	0	0	1	1
A_2	0	0	1	0	0	0	0
M_2	0	0	1	0	0	0	0

Towards efficiency

- The key property is that the sum of 1s in all the A_x is $O(n/\tau)$.
- We prove it by showing that any new element can induce $O(1/\tau)$ new positions in segments.
- We can coalesce ranges of bitmaps A_x and A_y that do not clash in their areas of 1s (and have at least one 0 in between).
- We cannot be confused be querying the coalesced bitmaps (only that we don't know who is the element — which is good).

Example for $\tau = 1/2$

C_3	1	1	1	1	1	1	0
M_3	0	1	0	1	1	0	0
$A =$	1	3	2	3	3	1	1
C_2	1	0	1	0	1	1	1
M_2	1	0	1	0	0	1	1

Towards efficiency

- We prove that $O((1/\tau)\log n)$ bitmaps are sufficient to coalesce all the A_x for all x, since only $O((1/\tau)\log n)$ segments can contain a given point.
- We query the coalesced bitmaps one by one. If some gives us a majority, we can return (all) the positions.
- We don't know who is x, but we know it is a majority.
- Then the time is $O((1/\tau)\log n)$.
- The coalesced bitmaps can be represented in compressed form using $O((n/\tau)\log\log n)$ bits.

The final touch

- We separate the runs of 1s in coalesced bitmaps by their rough lengths.
- This yields a more efficient encoding in $O(n \log(1/\tau))$ bits.
- The coalesced bitmaps M' are smaller but sparser
  - n 1s out of $O(n/\tau)$
  - thus *rank* queries require $O(\log \log_w(1/\tau))$ time.
- Then the time is $O((1/\tau) \log \log_w(1/\tau) \log n)$.

And a final twist

- By adding $O(n \log \log n)$ bits we slash the $O(\log n)$ factor from the time.
- When $1/\tau = O(\text{polylog}\, n)$, this yields optimal space and time, $O(1/\tau)$.
- Key idea: store which coalesced bitmaps to query, instead of having to scan them all.
- We will manage to query only $O(1/\tau)$ coalesced bitmaps.

And a final twist

For short query ranges ($O(1/\tau)$)

- For each $A[i] = x$, we store in which coalesced bitmap the segment of A_x covering i has been merged.
- Thus, we can test the positions in the range one by one.

And a final twist

For long query ranges

- We cover A with overlapped intervals of all lengths powers of 2 and larger than $1/\tau$.
- Each interval stores the coalesced bitmap of its $(\tau/4)$-majorities.
- We find the smallest interval that contains the query range.
- Any τ-majority in the range is a $(\tau/4)$-majority in the interval.
- Thus we only have to check for the $O(1/\tau)$ candidates.

Open questions

- Can we reach optimal time in all cases, or show it is not possible?
- Can we reach time according to $1/\tau'$ for any $\tau' > \tau$?
- Is the lower bound good enough when reporting arbitrary positions?
- Or, can we use less space to report arbitrary positions?
- What about reporting some occurrence of some τ-majority?
- Are there other interesting types of range queries for encodings?

Thanks for your attention!