Encoding top-k and range selection

Roberto Grossi[1] John Iacono[2] Gonzalo Navarro[3]
Rajeev Raman[4] S. Srinivasa Rao[5]

Università di Pisa, Italy.

Polytechnic Institute of New York University, United States.

Universidad de Chile.

University of Leicester, UK.

Seoul National University, S. Korea.

NII Shōnan seminar #29, 27 September 2013
Some results were presented at ESA 2013.

RMQ problem

Given an array $A[1..n]$, pre-process A to answer the query:

$$RMQ(l, r) = \arg\max_{l \le i \le r} A[i]$$

$A =$ | 10 | 8 | 3 | 1 | 6 | 2 | 9 | 5 | 4 | 7 | $RMQ(3, 6) = 5.$

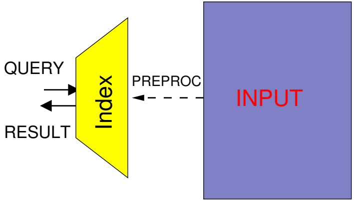
This is a *data structuring* problem.

- Preprocess input data to answer long series of queries.
- Want to minimize:
  1. Query time.
  2. Space usage of data structure.
  3. Time/space for pre-processing.

We do not consider updates to A.

Encoding Model

- Preprocess input to get *index*, delete input.
- Queries can *only* read index.
- Minimize index size and query time.



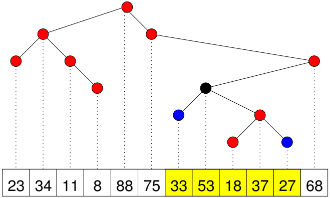
Motivations:

- Values in A can be intrinsically uninteresting (e.g. document scores).
- Encoding size may be smaller than size of A and can fit in "local" or "faster" memory.

Encoding RMQ

- Trivial RMQ encoding uses $\Theta(n \log n)$ bits: can we do better?

- Yes: encoding size is $2n - O(\log n)$ bits.
  - via *Cartesian tree* [Vuillemin, '80].
  - RMQ = LCA.
- Data structures:
  - $2n + o(n)$ bits, $O(1)$ query time.



[Fischer, Heun SICOMP'11],[Davoodi, R, Rao COCOON'12], building on [Harel, Tarjan, FOCS'83].

Encoding top-k and range selection

- Given $A[1..n]$ and k, encode A to answer the query:

  $\texttt{top-k-pos}(l, r)$: return positions of the k largest values in $A[l..r]$.

  - generalizes RMQ (case $k = 1$).
  - lower bounds on encoding size.
  - one-sided/prefix top-k queries $\texttt{top-k-pos}(r) = \texttt{top-k-pos}(1, r)$.
  - general two-sided queries.
  - other variants of problem.

  [First paper on encoding top-k-pos.]

- Given $A[1..n]$ and κ, encode A to answer the query:

  $\texttt{select}(k, l, r)$: return the position of the k-th largest value in $A[l..r]$, for any $k \leq \kappa$.

  - Related work by many authors including [Brodal and Jorgensen, ISAAC'09] [Jørgensen and Larsen, SODA'11], [Chan and Wilkinson, SODA'13].

Our results

1. One-sided/prefix variant ($\texttt{top-k-pos}(r)$ queries):
    - encoding size $\Omega(n \log k)$ bits.
    - $n \log k + o(n \log k)$ bits, $O(k)$ time or $O(k \log k)$ time sorted.
2. General two-sided range top-k queries:
    - $O(kn)$ bits, $O(k^2)$ time.
    - $O(n \log k)$ bits, $O(k)$ time.
3. Range selection queries:
    - $O(n \log k)$ bits and $O(\log k / \log \log n)$ time.
    - Matches time bound of [CW SODA'13] but uses less space. Time cannot be improved using $n(\log n)^{O(1)}$ space [JL SODA'11].
    - Lower bound for range selection [JL SODA'11]:
        - If you use B bits of space you need $\Omega(\log k / \log(B/n))$ time.
        - $O(n \log k)$ bits $\Rightarrow \Omega(\log k / \log \log k)$ time: we beat this.
        - Their lower bound is only for finding the k-th largest *value*.

Lower bound on encoding size

Lemma

Any encoding for *one-sided* top-k queries must take $\Omega(n \log k)$ bits.

Proof: The index can encode $(n/k) - 1$ independent permutations over k elements $\Rightarrow \Omega((n/k) \cdot k \log k)$ bits $= \Omega(n \log k)$ bits.

Proof by example ($k = 3$).

$$A = \boxed{3 \mid 1 \mid 2 \mid 4 \mid 6 \mid 5 \mid 8 \mid 9 \mid 7 \mid \cdots}$$

Encode A. Now:

$\texttt{top-k-pos}(1, 4) = \{1, 3, 4\} \Rightarrow A[2] = 1.$
$\texttt{top-k-pos}(1, 5) = \{1, 4, 5\} \Rightarrow A[3] = 2.$
$\texttt{top-k-pos}(1, 6) = \{4, 5, 6\} \Rightarrow A[1] = 3.$

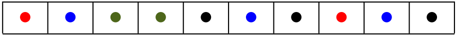
Encoding one-sided top-k queries

Use k colours $+1$ “null” ($=$ black) colour.

- First k elements assigned colours arbitrarily.
- Each new element gets colour of “ejected” element (“null” if none).

$A =$	6	4	2	10	3	7	5	8	9	1
$\Rightarrow$	●	●	●	●	●	●	●	●	●	●

Encoding one-sided top-k queries

$A \quad \Rightarrow$ 

To answer `top-k-pos`(j) queries, find the first occurrence before j of each colour. For example `top-k-pos`$(7) = \{6, 4, 1\}$.

Data structure for finding colours uses $\mathrm{succinct\ DS}^{\mathrm{TM}}$ technology, space used is $n \log k + o(n \log k)$ bits, time is $O(k)$.

Reports in unsorted order, but can compare colours, so can sort in $O(k \log k)$ time.

Open

$n \log k + o()$ space usage, $O(k)$ sorted reporting for 1-sided queries?

Encoding two-sided queries

Now we want the general problem: $\texttt{top-k-pos}(i, j)$.

- Basic approach: construct the Cartesian tree of top-k elements in $A[i], \ldots, A[j]$ in $O(k)$ time.
- Requires A to be available!
- It is enough if for each i, we store pointers to to k preceding and succeeding larger elements.

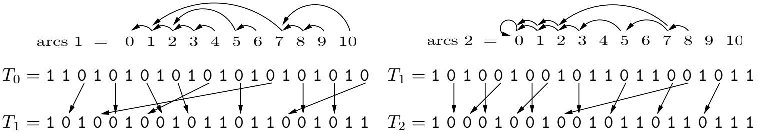
Specifically:

- Define arrays of pointers $P_0[1..n]$ to $P_k[1..n]$ as follows.
- $P_0[j] = j$ for all $j = 1, \ldots, n$.
- $P_{k+1}[j] = \max\left(\{i,\ \ i < P_k[j] \wedge a_i > a_j\} \cup \{0\}\right)$.

Naive representation of these arrays takes $O(kn \log n)$ bits.

Encoding pointers

A :	10	8	3	1	6	2	9	5	4	7
P_0 :	1	2	3	4	5	6	7	8	9	10
P_1 :	0	1	2	3	2	5	1	7	8	7
P_2 :	0	0	1	2	1	3	0	5	7	2



$B_1 = (\,)\,(\,(\,)\,(\,(\,)\,(\,)\,)\,(\,)\,)\,(\,(\,)\,(\,)\,)$

$B_2 = (\,)\,(\,(\,)\,)\,(\,(\,)\,)\,(\,(\,)\,(\,)\,(\,)\,)\,(\,)$

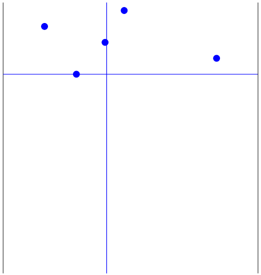
- Level i pointers are non-crossing.
- Can be encoded using $4n + o(n)$ bits $\rightarrow O(kn)$ bits overall.
- Can obtain $P_{i+1}[j]$ from $P_i[j]$ in $O(1)$ time.
- Find top-k in $O(k^2)$ time overall.

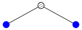
Optimal two-sided queries

- View A geometrically in 2D: $A[i] = y \Rightarrow (i, y)$.
- Use idea of *shallow cutting* for top-k [JL SODA'11].
- Take set of n given points and decompose into $O(n/k)$ *slabs* each containing $O(k)$ points such that:
  - For any 2-sided query `top-k-pos`(l, r) $\exists$ slab such that it and two other adjacent slabs contain the top-k elements in $A[l..r]$.
  - Gives a kind of encoding: store relative order among these $O(k)$ elements: $O(k \log k)$ bits/slab $= O(n \log k)$ bits, optimal!
  - But we need to represent the shallow cutting!

Shallow cutting (pre-processing)

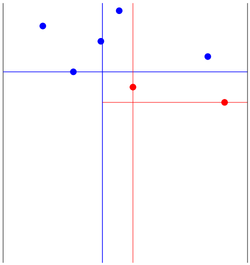
- Sweep a horizontal line down from $x = +\infty$.
- Initially just one slab. Place points as encountered into their slab.
- When slab has $2k - 1$ points, split and create boundaries as follows:
  - median x-coordinate as vertical boundary.
  - bottom y-coordinate as bottom boundary.
- Example: $k = 3$.

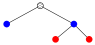




Shallow cutting (pre-processing)

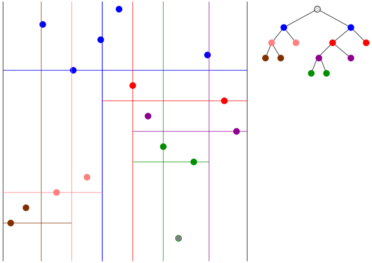
- Sweep a horizontal line down from $x = +\infty$.
- Initially just one slab. Place points as encountered into their slab.
- When slab has $2k - 1$ points, split and create boundaries as follows:
    - median x-coordinate as vertical boundary.
    - bottom y-coordinate as bottom boundary.
- Example: $k = 3$.





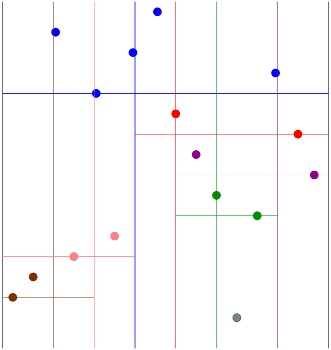
Shallow cutting (pre-processing)

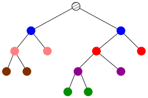
- Sweep a horizontal line down from $x = +\infty$.
- Initially just one slab. Place points as encountered into their slab.
- When slab has $2k - 1$ points, split and create boundaries as follows:
  - median x-coordinate as vertical boundary.
  - bottom y-coordinate as bottom boundary.
- Example: $k = 3$.



- At end: $O(n/k)$ slabs each with $\Theta(k)$ elements.
- Slabs naturally form full binary "tree of slabs" T_s.
- Naive encoding of x-coordinates requires $O(k \log n)$ bits/slab, or $O(n \log n)$ bits overall.

Encoding the slabs





- Retrieve resolving slab: LCA.
- Retrieve x-coordinates of slab boundaries: top-2 pointers, $O(n)$ bits. slab bottom: ?
- Retrieve x-coordinates of points + answer queries: perform RMQs using CT of A, guided by $O(k \log k)$ bits of ordering info.

Theorem

There is an encoding of size $O(n \log k)$ bits that supports `top-k-pos` queries in $O(k)$ time.

Encoding range selection

Problem

Given $A[1..n]$ and κ, encode A to answer $\texttt{select}(k, l, r)$ which returns the position of the k-th largest value in $A[l..r]$, for any $k \leq \kappa$.

Overall approach is similar:

- Create κ-shallow cutting.
- For $O(\kappa)$ points in each slab, store range selection data structure: $O(\kappa \log \kappa)$ bits.
- Find resolving slab for given query and use slab's range selection data structure to answer query.
- Convert answer back to "global" coordinates.

Encoding shallow cutting

Previous shallow cutting representation was space optimal but could only enumerate all $O(\kappa)$ x-coordinates in a slab in $O(\kappa)$ time. We want $O(\log k / \log\log n)$ query time.

▷ We need a more sophisticated representation of slabs which can:

- in $O(1)$ time, retrieve the i-th largest x-coordinate in the slab (access query).
- in $O(\log k / \log\log n)$ time, perform predecessor search for l and r among x coordinates in a slab.

Previous result by [CW SODA'13]

- $O(n \log \kappa + \underbrace{n \log\log n + (n \log n)/\kappa}_{\text{non-optimal terms}})$ bits of space.

Tree Partitioning and Marking

We partition the *tree of slabs* T_s. T_s has $n' = O(n/\kappa)$ nodes.

- Let $s(v)$ be the number of descendants of v in T_s.
- Let $t_0 = n'$ and $t_{i+1} = \lceil \log_2 t_i \rceil$, stopping when $t_z = 1$.
- A node v is *level* i if $t_i^2 \leq s(v) < t_{i-1}^2$.
  - Node levels decrease from leaf to root.
  - ▷ x-coordinates in a level i node take $O(\log t_{i-1}^2) = O(t_i)$ bits.

Mark an internal node in T_s if:

1. it is level i and both its children are level $> i$.
2. it is level i and both its children are level i.
3. it is level i and its parent is level $< i$.

Lemma

The number of marked level i nodes is $O(n'/t_i^2) = O(n/(\kappa t_i^2))$.

For each marked node we store all its x-coordinates explicitly. Sum over all level i nodes is $O((n/(\kappa t_i^2)) \cdot \kappa t_i) = O(n/t_i)$ bits $\Rightarrow O(n)$ bits overall.

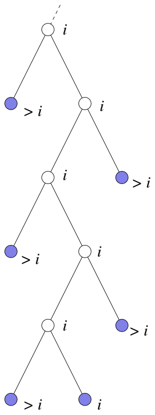
Tree Partitioning and Marking

Marking Rule

v is marked if:

1. it is level i and both children are level $> i$.
2. it is level i and both children are level i.
3. it is level i and parent is level $< i$.

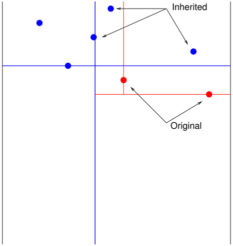
- Each unmarked level i node has:
  - one *marked* child at level $< i$.
  - one child at level i.
- Unmarked level i nodes form *paths* fringed by marked nodes.

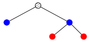


access queries

▷ Need to store the x-coordinates of points in an unmarked node v.

- Points in v are *original* or *inherited*.

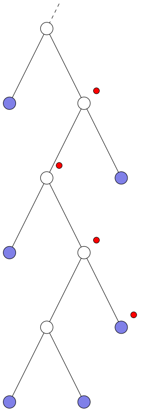




access queries

▷ Need to store the x-coordinates of points in an unmarked node v.

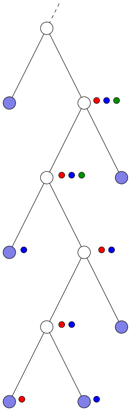
- Points in v are *original* or *inherited*.
- Each original point in v is stored explicitly in a marked node fringing the unmarked path.



access queries

▷ Need to store the x-coordinates of points in an unmarked node v.

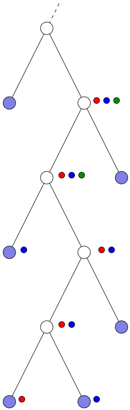
- Points in v are *original* or *inherited*.
- Each original point in v is stored explicitly in a marked node fringing the unmarked path.
- Pointers to the marked nodes where v's original points lie cost $O(n)$ bits summed over all unmarked nodes.



access queries

▷ Need to store the x-coordinates of points in an unmarked node v.

- Points in v are *original* or *inherited*.
- Each original point in v is stored explicitly in a marked node fringing the unmarked path.
- Pointers to the marked nodes where v's original points lie cost $O(n)$ bits summed over all unmarked nodes.
- For inherited points p, use $O(\kappa)$ colors (cf. 1-sided top-k) to find the ancestor where p is original: $O(n \log \kappa)$ bits.



access queries

Modulo many details (succinct DS$^{\text{TM}}$ technology):

Lemma

We can encode the cells of the shallow cutting to support access queries in $O(1)$ time.

Implies:

- Encoding for range selection using $O(n \log \kappa)$ bits in $O(\log k)$ time.
- Can return top-k, for any $k \leq \kappa$ in $O(k)$ time.

No details given:

Theorem

There is an encoding for range selection that takes $O(n \log \kappa)$ bits and supports range selection in $O(\log k / \log \log n)$ bits.

Conclusions and Open Problems

Conclusions:

- Gave optimal,non-trivial, encodings for range selection and range top-k.
- Improved prevous bounds, “broke” lower bound.

Open problems:

- Sorted reporting in one-sided case.
- Exact constant factors (progress for $k = 2$).
- Can we extend this to partially ordered A? (N. Yasuda)
- What about average-case encoding complexity? (S.-I. Minato)
- Obvious pre-processing times are $O(n \log k)$ for the one-sided case and $O(n \log n)$ for the 2-sided case. Can this be improved? (N. Yasuda)