2018-10-24

Succinct Data Structures for Range Minimum Problems

Sebastian Wild

wild@uwaterloo.ca



UNIVERSITY OF WATERLOO
FACULTY OF MATHEMATICS
David R. Cheriton School
of Computer Science

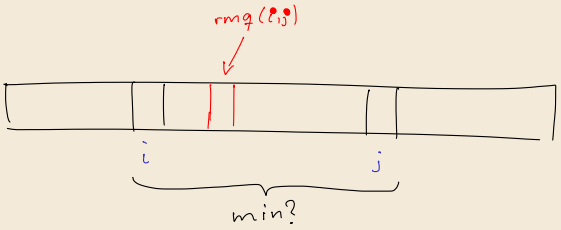
Part I : RMQ? Succinct?

Range Minimum Problem

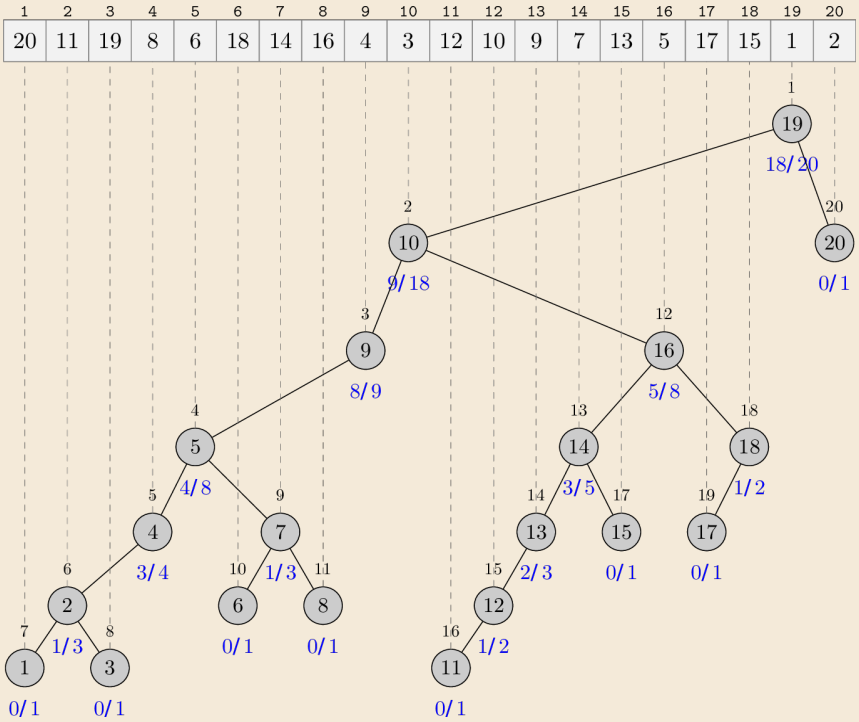
Given : array A of numbers

Task : preprocess A

answer rmq(i,j) = position of (a) min in A[i..j]



Cartesian tree

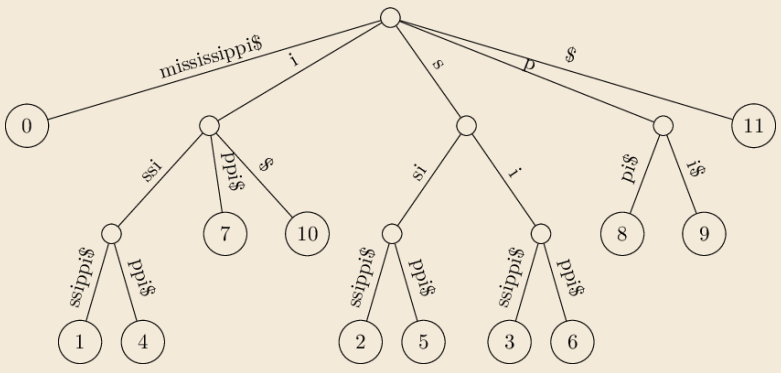


Applications of rmq

space efficient suffix trees

1 2 3 4 5 6 7 8 9 10 11 12

mississippi$



all suffixes

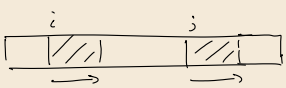
```
mississippi$
ississippi$
ssissippi$
sissippi$
issippi$
ssippi$
sippi$
ippi$
ppi$
pi$
i$
$
```

sort →

longest common prefix		suffix array
0	$	12
1	i$	11
1	ippi$	8
4	issippi$	5
0	ississippi$	2
0	mississippi$	1
1	pi$	10
0	ppi$	9
2	sippi$	7
1	sissippi$	4
3	ssissippi$	3
	ssippi$	6

longest common extension of two positions

$lce(i,j) = LCP[rmq_{LCP}(i+1, j)]$



LCP[i] = lcp (ith and (i-1)st suffix)

Trivial solutions for rmq

- scan $A[i..j]$, no preprocessing

  preprocessing time ↓
  $\langle O(1), O(n) \rangle$ solution
  ↑ query time (worst case)

- precompute all

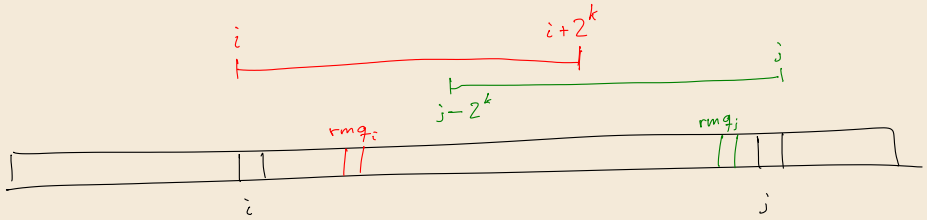
  $\langle O(n^2), O(1) \rangle$



Can we do better? Yes $\langle O(n \log n), O(1) \rangle$ possible

Sparse table rmq

Idea: only store solutions for regions with length 2^k



min is either rmq_i or rmq_j

⇒ $\lceil \lg n \rceil$ lengths, n left endpoints ⇒ $\sim n \lg n$ entries

can be filled bottom-up $O(1)$ time per entry

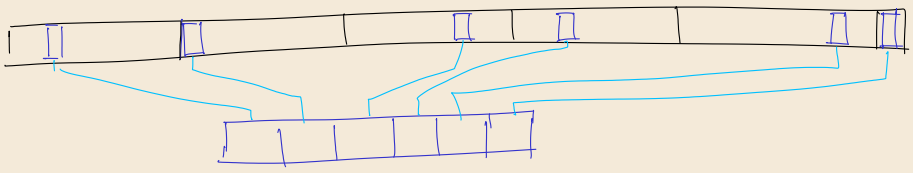
⇒ $\langle O(n \log n), O(1) \rangle$ solution

Can we do better?

Yes: $\langle O(n), O(1) \rangle$ possible

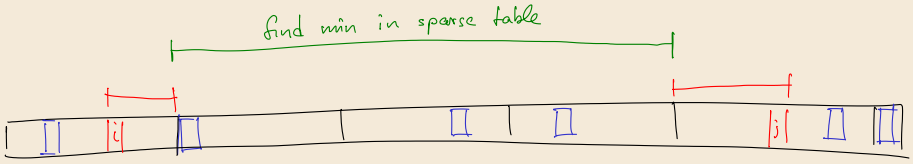
Linear time rmq via blocking

Divide A in blocks of $b = \left\lceil \frac{\lg n}{4} \right\rceil$ elements



Store:
- position of min in each block — $\frac{n}{b}$ entries
- sparse table solution for blocks (using mins as values for blocks) — entries: $\frac{n}{b} \cdot \lg\left(\frac{n}{b}\right) \leq \frac{n}{b} \lg n = 4n$ entries

Queries:



Could scan two blocks
→ $O(\log n)$ time

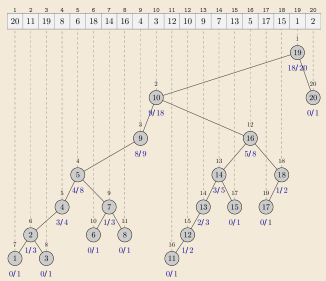
Constant time: Precompute all intra-block queries.

naively: $\frac{n}{b} \cdot b^2 = \Theta(n \log n)$ (#queries: b^2, #blocks: $\frac{n}{b}$)



"Four-Russians" Trick

Recall:
rmq ≡ Cartesian tree

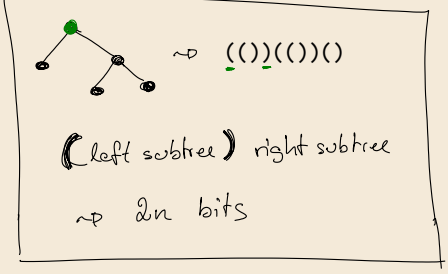


only $C_b = \binom{2b}{b}/(b+1) \leq 4^b = \sqrt{n}$ different Cartesian trees

$4^b = 2^{2 \cdot \log n/4} = n^{1/2}$

Store:
- solution to block-local rmq for all — $\sqrt{n} \cdot b^2$ entries
- encoding of Cartesian tree for each block

$\frac{n}{b} \cdot 2b = 2n$ bits



How about space?

solutions above uses $O(n)$ words of space
(words: e.g. indices in array)

↳ that is actually $\Theta(n \log n)$ bits

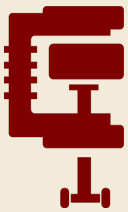
Information content in rmq answers = information content in Cartesian tree

→ can represent latter in $\lg C_n \sim 2n$ bits

(but not clear how to find rmq answer efficiently)

Goal in succinct data structures: $2n + o(n)$ bits of space & efficient operations

optimal space (up to l.o.t.) for the worst case



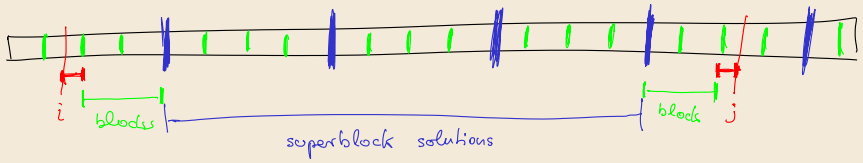
Succinct data structures for rmq

Option 1: above idea

but use bigger blocks $B = \lg^3 n$ ⇒ sparse table $o(n)$ bits

⇒ need 2 levels of sparse tables — for superblocks
\ for blocks within 1 superblock

answer is one of 5 candidates:



⚠ still need original array A

Option 2: use a succinct tree data structure to store the Cartesian tree

need operations to
- find a node given its inorder index
- find lowest common ancestor of 2 nodes

can be done.

(⚠ some of these ds use rmq as subroutine ...)

Part II Compressed Trees

If we do not have access to A at query-time $\sim 2n$ bits are needed in the worst case.

What if A is random? $A[i]$ iid reals
or random permutation

Can we reduce the space usage in the average case?

shape distribution of Cartesian trees = random binary search tree

position of min $\hat{=}$ rank of root $\quad u[1..n]$

entropy of : $\sim 1.736n$ bits

Structural Complexity
of Random Binary Trees
J. C. Kieffer, E-H. Yang, and W. Szpankowski

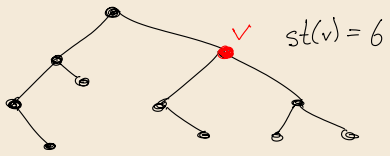
Subtree-size entropy

Notation: t binary tree (shape)

v node in t

$st(v)$ size of v's subtree

T random BST (shape)



$$\Pr[T = t] = \prod_{v \in t} \frac{1}{st(v)}$$

← prob that among all values in v's subtree, v is inserted first

entropy of T's distribution $= \sum_t \Pr[T=t] \cdot \lg \frac{1}{\Pr[T=t]}$

$\sim 1.736n$

$= \quad$ ——"—— $\lg \left(\prod_{v \in t} st(v) \right)$

$= \quad$ ——"—— $\boxed{\sum_{v \in t} \lg(st(v))} =: H_{st}(t)$ "subtree-size entropy"

$= \mathbb{E}\left[H_{st}(t) \right]$

$\Rightarrow$ If we can encode trees using $H_{st}(t) + l.o.t.$ bits, we are optimal on average!

Encoding in $H_{st}+2$ bits

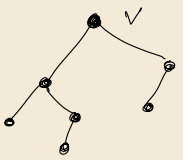
⇝ use arithmetic coding

idea : store the sizes of all left subtrees

for node v : $ls(v) \in [0..\, st(v)]$

↖ uniformly in random BSTs !?

↳ local entropy $\lg(st(v))$



$st(v) = 7$

$ls(v) = 4 \in [0..\, st(v)-1]$

Invariant : $st(v)$ already encoded / known ← encode nodes in preorder traversal

$\Rightarrow$ Can encode all $ls(v)$ with arithmetic encoding

optimal (up to +2 bits)

compression scheme



How to access specific nodes ?

How to compute lca ?

Polylog-query compressed data structure

break Cartesian tree into blocks

store blocks using above encoding ⇝

+ index data structures

How to define blocks ?

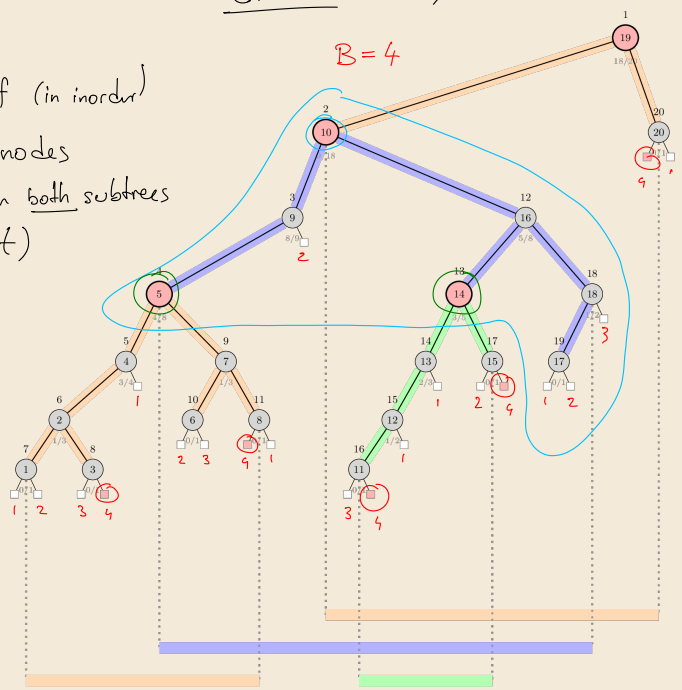
extended binary tree

B = 4

① mark every Bth leaf (in inorder)

② mark all internal nodes with marked leaf in both subtrees (+ always mark root)

③ marked nodes = roots of blocks



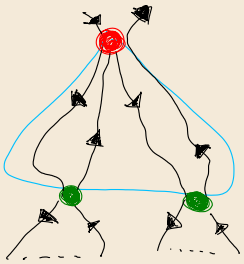
Properties :

(a) $\approx \frac{n}{B}$ blocks

(b) between 2 and 3B nodes in a block

(c) blocks are complete subtree except for two portal nodes

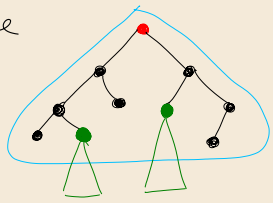
(d) traversals visit each block ≤ 3 times

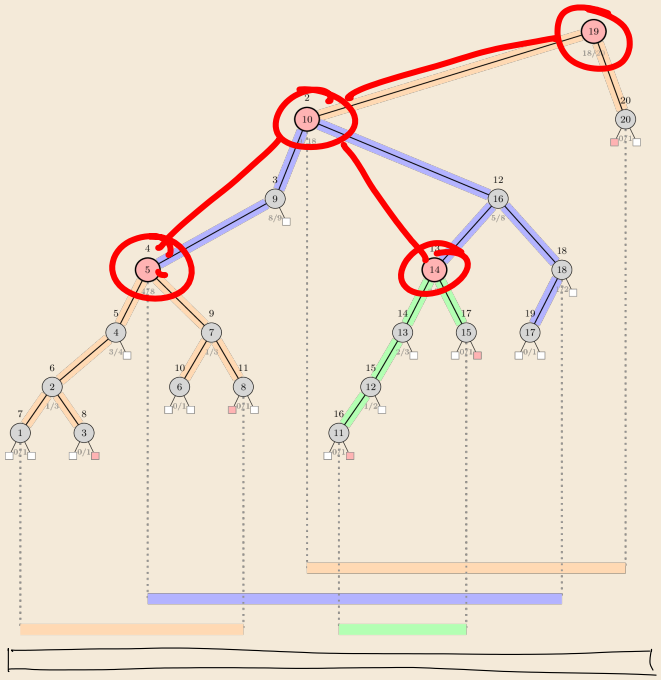


data structures stored

$$B = \lg^2 n$$

- lca info for marked nodes
  (again a binary tree!)
- for each block — local to block
  - left-subtree sizes in arithmetic code
  - the portals
  - info about root:
    global preorder index
    global inorder index





- array blockId[1..n] for all nodes
  ↖ given by preorder index
- blockIdByInorder[1..n]

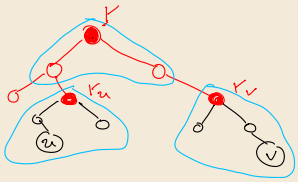
queries (time O(B))

$$rmq(i,j) = \text{inorder}\big(\text{lca}(\text{nodes with inorder } i \text{ and } j)\big)$$

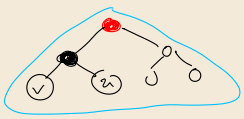
① inorder → node
- use blockIdByInorder to find block
- decode block encoded left-tree sizes, find node

② lca(u,v)
- find lca r of block roots r_u, r_v
  (lca data structure for marked nodes)

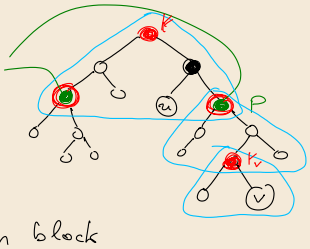
Case 1: $r_u \neq r \neq r_v$
⇒ return block lca



Case 2: $r_u = r = r_v$
⇒ u and v in same block
decode block and find lca within block



Case 3: $r_u = r \neq r_v$
⇒ answer is lca of u and one portal
- find correct portal p by checking
  if lca(p,v) = p.
- decode block and find lca(p,u) within block



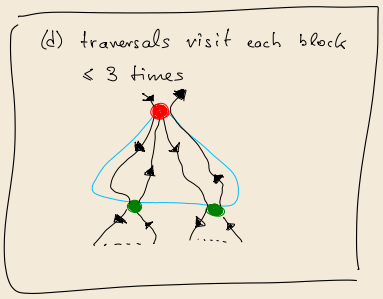
③ node → inorder
when decoding blocks use info on root & portals
to find inorder of nodes in block



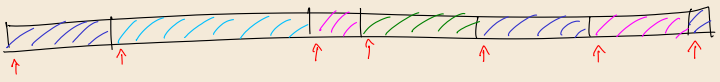
arrays blockId / blockIdByInorder need too much space

$n \cdot \lg\left(\frac{n}{B}\right) \sim n \lg n$

Piecewise-constant arrays

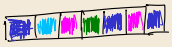


$\Rightarrow$ blockId changes at most $3 \cdot \frac{n}{B}$ times, constant in between



$\Rightarrow$ we can compress blockId arrays!



store:
- indices of changes
- array of values 

how to access blockId[v]?

use classic succinct data structures result

compressed bit vectors with rank

rank(i) = # 1-bits in A[1..i] in O(1) time

bitvector B[1..n] with m 1-bits can be stored with

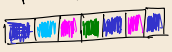
$\lg\binom{n}{m} + O\left(n \frac{\lg\lg n}{\lg n}\right)$ bits of space

information content

Succinct Indexable Dictionaries with Applications to Encoding *k*-ary Trees, Prefix Sums and Multisets

RAJEEV RAMAN
University of Leicester
VENKATESH RAMAN
Institute of Mathematical Sciences
AND
SRINIVASA RAO SATTI
University of Waterloo

$\Rightarrow$ Can compress piecewise constant arrays to

space for values  $+ \quad m \lg \frac{n}{m} \quad + O(m)$

$$\lg\binom{n}{m} \leq m \lg \frac{n}{m} + O(m)$$

↳ for blockId $m = 3\frac{n}{B}$

$\leq 3\frac{n}{B} \cdot \lg n + 3\frac{n}{B} \lg n = O\left(\frac{n}{\lg n}\right) = o(n)$

Compressed bitvectors with rank support

(rough) idea:

- break bitvector into blocks of b bits
- for block with c 1-bits, store pair (c, o) (c: class, o: offset)

  where $o \in [0..\binom{b}{c}]$

  ↖ few 1-bits ⇒ less space

Succinct Indexable Dictionaries with Applications to Encoding k-ary Trees, Prefix Sums and Multisets

RAJEEV RAMAN
University of Leicester
VENKATESH RAMAN
Institute of Mathematical Sciences
AND
SRINIVASA RAO SATTI
University of Waterloo

(they do more operations that we do not need here)

⇝
- use Four-Russians-trick to precompute block answers
- use 2-level index to precompute rank queries

⚠ storing offsets for blocks requires array with variable-size cells
can be done, again using 2-level index

Part III Outlook & Conclusion

We have seen

- elementary solutions for rmq
- refinements with less space
- even less space on average

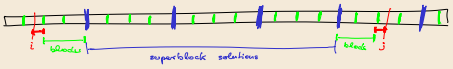


compression

Techniques



- blocking
- Four-Russians
- 2-level index



Mighty black boxes

- compressed bit vectors with rank & select
- variable-cell array

not: compressed data structure with constant-time queries

What else?

- Interesting compression schemes for further compressed data structures?
  - random graphs?
  - text indices for random text?

