Advanced Data Structures

Lecture 04: Predecessor and Range Minimum Query Data Structures

Florian Kurpicz
Recap

Succinct Planar Graphs

- using spanning tree of graph and
- special spanning tree of dual graph
- both represented succinctly
- represent planar graph succinctly
- store whether edge is in spanning tree or not
Predecessor and Successor

Setting
- Assume universe $\mathcal{U} = [0, u)$
- Let $u = 2^w$
- Sorted array of $n$ integers $A \subseteq \mathcal{U}$
- $\log n \leq w$ since $n \leq u$

Definition: Predecessor & Successor
Given an array $A$ of $n$ integers from an universe $\mathcal{U}$ and an integer $x \in \mathcal{U}$, the predecessor and successor of $x$ in $A$ are
- $\text{pred}(A, x) = \max \{y \in A : y \leq x\}$
- $\text{succ}(A, x) = \min \{y \in A : y \geq x\}$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>10</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

- $\text{pred}(3) = 2$
- $\text{pred}(10) = 10$
- $\text{succ}(23) = 32$

- In what time and space can we solve this using bit vectors? 📊 PINGO
Predecessor and Successor: Simple Solutions

- Binary search
  - $O(\log n)$ query time
  - No space overhead

- Using bit vector
  - $O(1)$ query time
  - $u + o(u)$ bits space

Predecessor of $x$ in Bit Vector
- $z = \text{rank}_1(x + 2)$
- Predecessor is $\text{select}_1(z)$

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>10</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>32</td>
</tr>
</tbody>
</table>

$\text{pred}(3) = 2$

1110100100100000000011100000001
- $\text{rank}_1(21) = 6$
- $\text{select}_1(6) = 10$
- $\text{pred}(19) = 10$
Elias-Fano Coding [Eli74; Fan71] (1/3)

- $n$ integers from universe $\mathcal{U} = [0, u)$
- split number in upper and lower halves
- upper half: $\lceil \log n \rceil$ most significant bits
- lower half: $\lceil \log u - \log n \rceil$ remaining bits

**Upper Half**
- monotonous sequence of $\lceil \log n \rceil$ bit integers
- not strictly monotonous
- let $p_0, \ldots, p_{n-1}$ be sequence
- use bit vector of length $2n + 1$ bits
- represent $p_i$ with a 1 at position $i + p_i$
- rank and select support requires $o(n)$ bits

**Lower Half**
- store lower half plain using $\lceil \log \frac{u}{n} \rceil$ bits
- $n \log \lceil \frac{u}{n} \rceil$ bits for lower half

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>10</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>32</td>
</tr>
<tr>
<td>0:</td>
<td>000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1:</td>
<td>000001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2:</td>
<td>000010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4:</td>
<td>000100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7:</td>
<td>000111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10:</td>
<td>001010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20:</td>
<td>010100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21:</td>
<td>010101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22:</td>
<td>010110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30:</td>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Elias-Fano Coding (2/3)

Access \( i \)-th Element
- upper: \( \text{select}_1(i) - i \)
- lower: corresponding bits from lower bit vector

Predecessor \( x \)
- let \( x' \) be \( \lceil \log n \rceil \) MSB of \( x \)
- \( p = \text{select}_0(x') \) \( \text{select}_0(0) \) returns 0
- scan corresponding values in lower till predecessor is found
- how many elements do we have to scan? PINGO
- scanning \( O(u/n) \) elements

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 10 & 20 & 21 & 22 & 32 \\
0 & 1 & 2 & 4 & 7 & 10 & 20 & 21 & 22 & 30 & & & \\
\hline
0: & 000000 & & & & 10: & 001010 & & & & & & \\
1: & 000001 & & & & 20: & 010100 & & & & & & \\
7: & 000111 & & & & 30: & 100000 & & & & & & \\
\end{array}
\]

upper: 11101101000111000100
lower: 00 01 10 00 11 10 00 01 10 00
Lemma: Elias-Fano Coding

Given an array containing $n$ distinct integers from a universe $\mathcal{U} = [0, u)$, the array can be represented using $n(2 + \log\lceil \frac{u}{n} \rceil)$ bits while allowing $O(1)$ access time and $O(\log \frac{u}{n})$ predecessor/successor time.
x-Fast Tries

- each number has $w$ bits
- build binary tree where leaves represent numbers
- edges are labeled 0 or 1
- labels on path from root to leaf are value represented in leaf

- store nodes in hash tables with bit prefix as key
- also store pointer to min and max in right and left subtree
- leaves are stored in doubly linked list
- using perfect hashing on each level requires $O(wn)$ space

- pointers to min and max are missing
- tree most likely not complete
x-Fast Tries: Queries

- traversing tree requires $O(w)$ time
- using binary search on levels requires $O(\log w)$ time
- if value not found go to min or max depending on query
- if value is found use doubly linked list to find predecessor or successor

- example on the board 📚
y-Fast Tries

- x-fast trie requires $O(wn)$ space
- group $w$ consecutive objects into one block $B_i$
- for each block $B_i$ choose maximum $m_i$ as representative
- build x-fast trie for representatives
- store blocks in balanced binary trees

- x-fast trie requires $O(n)$ space
- search in x-fast trie requires $O(\log w) = O(\log \log n)$ time \textit{For large $n$}
- search in balanced binary tree requires $O(\log w) = O(\log \log n)$ time

example on the board

Dynamic y-Fast Trie

- use cuckoo hashing
- representative does not have to be maximum
- any element separating groups suffices
- merge and split blocks that are too small/too big
- query time only expected
Range Minimum Queries

Setting

- array of $n$ integers
- not necessarily sorted

Definition: Range Minimum Queries

Given an array of $A$ of $n$ integers

$$rmq(A, s, e) = \arg \min_{s \leq i \leq e} A[i]$$

returns the position of minimum in $A[s, e]$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>8</strong></td>
<td><strong>2</strong></td>
<td><strong>5</strong></td>
<td><strong>1</strong></td>
<td><strong>9</strong></td>
<td><strong>11</strong></td>
<td><strong>10</strong></td>
<td><strong>20</strong></td>
<td><strong>22</strong></td>
<td><strong>4</strong></td>
<td></td>
</tr>
</tbody>
</table>

- $rmq(0, 9) = 3$
- $rmq(0, 2) = 1$
- $rmq(4, 8) = 4$

- naive in $O(1)$ time
- how much space does a naive $O(1)$-time solution need?

- using $O(n^2)$ space

$rmq(s, e) = M[s][e]$
instead of storing all solutions
store solutions for intervals of length $2^k$ for every $k$
$M[0..n][0..\lfloor \log n \rfloor)$

**Queries**
- query $rmq(A, s, e)$ is answered using two subqueries
- let $\ell = \lfloor \log (e - s - 1) \rfloor$
- $m_1 = rmq(A, s, s + 2^\ell - 1)$ and $m_2 = rmq(A, e - 2^\ell + 1, e)$
- $rmq(A, s, e) = \arg \min_{m \in \{m_1, m_2\}} A[m]$

**Construction**

$M[x][\ell] = rmq(A, x, x + 2^\ell - 1)$

$= \arg \min \{A[i] : i \in [x, x + 2^\ell)\}$

$= \arg \min \{A[i] : i \in \{rmq(A, x, x + 2^{\ell-1} - 1),$

$= rmq(A, x + 2^{\ell-1}, x + 2^\ell - 1)\}\}$

$= \arg \min \{A[i] : i \in \{M[x][\ell - 1],$

$= M[x + 2^{\ell-1}][\ell - 1]\}\}$

- how much time do we need to fill the table?
- dynamic programming in $O(n \log n)$ time
divide $A$ into blocks of size $s = \frac{\log n}{4}$
blocks $B_1, \ldots, B_m$ with $m = \lceil n/s \rceil$
query $rmq(A, s, e)$ is answered using at most three subqueries
one query spanning multiple block
at most two queries within a block each

Query Spanning Blocks

- use array $B$ containing minimum within each block
- $B$ has $m$ entries
- use $O(n \log n)$ data structure for $B$
- $O(m \log m) = O\left(\frac{n}{s} \log \frac{n}{s}\right) = O\left(\frac{n}{\log n} \log \frac{n}{\log n}\right) = O(n)$
- use additional array $B'$ storing position of minimum in each block
- for queries within block use Cartesian trees
Definition: Cartesian Tree
Given an array $A$ of length $n$, a Cartesian tree $C(A)$ of $a$ is a labeled binary tree with
- root $r$ is labeled with $x = \arg \min \{A[i] : i \in [0, n)\}$
- left and right children of $r$ are Cartesian trees $C(A[0, x))$ and $C(A[x + 1, n))$ if interval exists

Lemma: Cartesian Tree Construction
A Cartesian tree for an array of size $n$ can be computed in $O(n)$ time

Proof (Sketch)
- scan array from left to right
- insert each element by
  - following rightmost path from leaf to root till element can be inserted
  - everything below becomes left child of new node
- each node is removed at most once from the rightmost path
- moving subtree to left child in constant time gives $O(n)$ construction time

- example on the board
Lemma: Equality of Cartesian Trees

Given two arrays $A$ and $B$ of length $n$ with equal Cartesian trees, then

$$rmq(A, s, e) = rmq(B, s, e)$$

for all $0 \leq s < e < n$

Proof (Sketch)

- proof by induction over the size of the array
- if the array has size one, this is true
- assuming this is correct for arrays of size $n$, showing this for arrays of size $n + 1$ uses recursive definition of Cartesian trees
Query Within a Block

- consider every possible Cartesian tree for arrays of size \( s = \frac{\log n}{4} \)
- tree can be represented using \( 2s + 1 \) bits
- store bit representation of Cartesian tree for every block
- for every possible Cartesian tree and every start and end position store position of minimum
- \( O(2^{2s+1} \cdot s \cdot s \cdot \log s) = O(\sqrt{n} \log^2 n \cdot \log \log n) = O(n) \) space
Conclusion and Outlook

This Lecture
- successor and predecessor data structures
- range minimum query data structures

Advanced Data Structures

- Successor
  - static
  - BV
- RMQ
  - static
  - succ. trees
- Range min-max tree
- Succ. graphs
Bibliography I


[Fan71] Robert Mario Fano. On the Number of Bits Required to Implement an Associative Memory. 1971.