Improving Space by Bit Packing  (Chazelle'88)

for 2D Counting
idea - modify range tree
idea - replace list of y-coords by list of bits

\[ \text{left} = 0 \quad \text{right} = 1 \]

Standard Assumption  RAM model with fixed word size \( w \)
where \( w \geq \log n \) bits

Fact: can store array \( A \) of \( n \) bits using \( O(n/w) \) space
(in words)

Sit: can compute \( \text{rank}_0(i) = \# \text{Os in } A[1..i] \)
\( \text{rank}_1(i) = \# \text{1s in } A[1..i] \)

Pf.: divide array into \( n/w \) words

Precompute \( c_j = \# \text{Os in first } j \text{ words} \)

Count \# Os in a word by a nonstandard word op.
(if \( w = \log n \), can build table with all answers
in \( O(n) \) space)

Rank: can further reduce space to \( n/w + o(n/w) \)
by succinct DS
Space: $S(n) = 2 S(n/2) + O(\frac{n}{w})$
\[ \Rightarrow O\left(\frac{n}{w} \log n\right) = O(n) \] by setting $w = \log n$.

Query Alg, for dominance (2-sided) counting:
given position $i$ of q's $y$-coord:
- If $q$ left of median $x$:
  - recurse on left with position rank($i$)
- else add rank($i$) to count
  - recurse on right with position rank($i$)

\[ \Rightarrow Q_2(n) = Q_2(\frac{n}{2}) + O(1) = O(\log n) \]

for general (4-sided) counting:
reduce to dominance by subtraction/addition
\[ \Rightarrow O(\log n) \]

(Rank - related to wavelet trees
- reporting is costlier \( O(\log n + k \log n) \))

Improving query time, by bit tricks?

Assumption: input values are integers in \([1, U]\) word RAM model where each word has fixed size $w$ with $w \geq \log n$ (index/ptr fits in a word)
with $w \geq \log U$ (input value """)
Warm-Up: 1D Pred/Succ Search

Method 1: van Emde Boas (vEB) tree

- divide \([U]\) into \(V_0\) chunks of size \(V_0\)
- recurse inside each chunk
- let \(D = \{ \text{(indices of)} \}
  \text{ all nonempty chunks}\)
- recurse in \(D\)
- store min & max of each nonempty chunk

Query alg'm, given pt \(q\):

\( i = \text{index of chunk containing } q \) \( \in O(1) \) time
by arithmetic

- If \( i \in D\), recurse inside chunk
- If \( i \notin D\), recurse in \( D\)

\( O(1) \) time by hashing

\[ Q(U) = Q(\sqrt{U}) + O(1) \]

let \( U = 2^l \) \( \Rightarrow \) \( Q'(l) = Q'(l/2) + O(1) \)

\( (\sqrt{U} = 2^{l/2}) \) \( \Rightarrow \) \( Q'(l) = O(\log l) \)

\( \Rightarrow \) \( Q(U) = O(\log \log U) \)

Space: each element stored in \( \leq 2 \) recursive substructures

\( \text{s(U)} = 2 \text{s(\sqrt{U})} + O(1) \)

space per element

total space \( \text{O(n} \ 2^{\log \log U} = O(n \log U) \)
Space reduction trick:

- Select subset containing every $5^\text{th}$ element.
- Build uEB for subset.
- Space $O\left(\frac{n}{5} \log U \cdot n\right) = O(n)$
- Query time $O\left(\log \log U + \log b\right) = O(\log \log U)$

Finish by binary search.

Method 2: Fusion Tree (Fredman, Willard '90)

- $O(n)$ space, $O(\log \omega n)$ time.
- Rough idea: $b$-ary search tree with $b = \sqrt{\omega}$.

Need to search among $b$ numbers in $O(1)$ time.

How? Compress $b$ numbers into a single word.

Nonstandard approach: use a word of

e.g. $\{0100, 0110, 1110, 1111\}$

Trie

$O(bw)$ nodes: bad!

$O(b)$ nodes
compressed trie encodable in $O(\log w) \leq w$ bits

to: query $q$
  follow path to get element $z$

$z = 0110$

may be wrong

find first bit where $q$ & $z$ differ

(avoiding nonstandard ops is complicated!)

Method 3: Combine!

$$Q(n) = O\left(\min\{\log \log U, \log n^2\}\right)$$

$$\leq O\left(\min\{\log w, \log \log n^2\}\right)$$

$$\leq O\left(V \log n\right)$$

beats $\log n$ for any $U$! (near-optimal)

Back to 2D . . .

Advanced Method 1

dominance (2-sided) emptiness

reduce to

pred search in $x$

"staircase"

\[\Rightarrow O(n) \text{ space, } O(\log \log U) \text{ time}\]