Lecture 6

Cell-probe model: (for lower bounds)

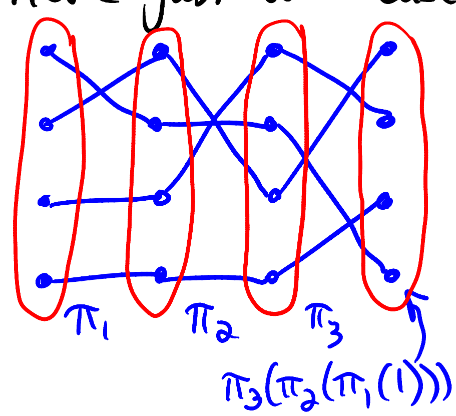
- memory (DS) consists of w-bit cells
- just count # reads & writes
- computation is free
- typically assume $w \geq \lg n$ or even $w = \Theta(\lg n)$

Dynamic connectivity lower bound [Pătraşcu & Demaine - STOC 2004 & SICOMP 2006]

$\Omega(\lg n)$ cell probes/op.

- holds even with amortization; here just worst case

Proof:



- consider $\sqrt{n} \times \sqrt{n}$ grid with perfect matching between consec. columns i & $i+1$ → permutation π_i
- block operations:
  - update(i, π): $\pi_i \leftarrow \pi$
    $= O(\sqrt{n})$ edge insertions/deletions
  - verify-sum(i, π): $\sum_{j=1}^{i} \pi_j = \pi$? ($\Sigma$ = compose)
    $= O(\sqrt{n})$ connectivity queries
- Claim: $\sqrt{n}$ updates + $\sqrt{n}$ verify-sums require $\Omega(\sqrt{n} \cdot \sqrt{n} \cdot \lg n)$ time (cell probes)

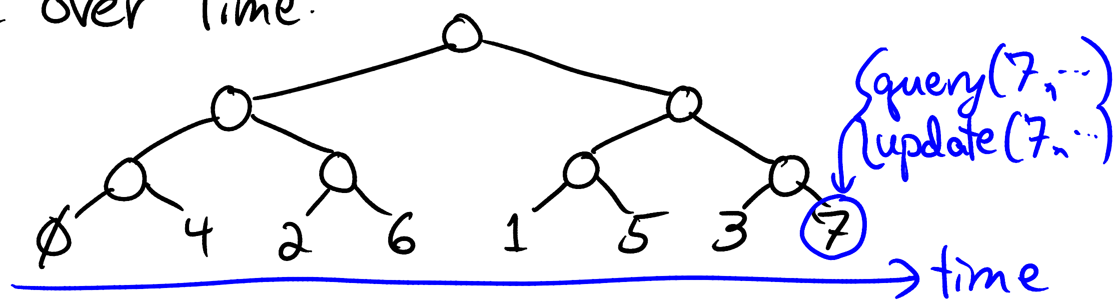
$\Rightarrow$ dynamic connectivity requires $\Omega(\lg n)$ time

Construction of bad access sequence:

- permutation π in each update(i, π) is chosen uniformly at random
- permutation π in each verify-sum(i, π) is the correct sum (but DS doesn't know) ← still must be correct
- i's follow <u>bit-reversal sequence</u>:

000	000	0
001 →	100 →	4
010	010	2
011	110	6
⋮	⋮	⋮

- pairs: verify-sum$(i, \sum_{j=1}^{i} \pi_j)$
  update-sum(i, π_{random})
- tree over time:



  - left & right subtrees of each node <u>interleave</u>

- <u>Claim</u>: for every node v in tree, say with l leaves in its subtree, during right subtree of v must do $\Omega(l\sqrt{n})$ cell probes in expectation that read cells last written during left subtree

- summing over all levels + linearity of expectation $\Rightarrow \Omega(n \lg n)$ lower bound overall (read r of write w is counted only at lca(r, w))

Proof of claim:

- left subtree has $\ell/2$ updates with $\ell/2$ rand. perms.
- any encoding of these permutations must use $\Omega(\ell\sqrt{n}\lg n)$ bits [info. theory/Kolmogorov arg.]
- if claim doesn't hold, we'll derive a smaller encoding $\Rightarrow$ contradiction.
- set up: know the "past" (before v's subtree)
- goal: encode (verified) sums in right subtree

$\Rightarrow$ can recover (updated) perms. in left subtree

Warmup: query is sum(i) $\rightarrow \sum_{j=1}^{i} \pi_j$

- let $R = \{$cells read during right subtree$\}$
  $W = \{$cells written during left subtree$\}$
- encode $R \cap W$ (address & contents of each cell)

$\Rightarrow |R \cap W| \cdot O(\lg n)$ bits [assume poly. space, $w = \Theta(\lg n)$]

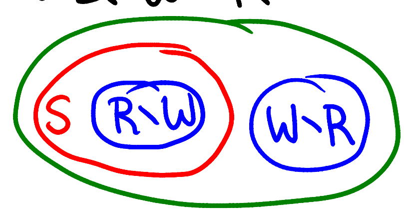
- decoding strategy for sums in right subtree:
  - simulate sum queries in right subtree
  - to read cell written in right subtree $\rightarrow$ easy
    in left subtree $\rightarrow R \cap W$
    in past $\rightarrow$ known

$\Rightarrow |R \cap W| \cdot O(\cancel{\lg n}) = \Omega(\ell\sqrt{n}\cancel{\lg n})$

$\Rightarrow |R \cap W| = \Omega(\ell\sqrt{n})$ ✓

Verify-sum instead of sum:

- permutations π given to verify-sum encode the information we want
- set up: - know (fixed) past
  - don't know updates in left subtree
  - don't know queries in right subtree
  - but know queries returned YES
- decoding idea:
  - simulate all possible input permutations for each query in right subtree
  - know one returns YES, all others return NO
- trouble: incorrect query simulation reads cells $R' \neq R$
  - if read $r \in R' \setminus R$, it must be incorrect
  - but can't tell whether $r \in W \setminus R$ or past $\setminus (R \cap W)$
  - can't afford to encode R or W
- idea: encode separator S for $R \setminus W$ & $W \setminus R$
- when decoding, to read a cell
  written in right subtree $\to$ easy
  in $R \cap W \to$ encoded explicitly
  in $S \Rightarrow$ must be in past $\to$ known
  not in $S \Rightarrow$ must not be in $R \Rightarrow$ wrong guess $\to$ ABORT

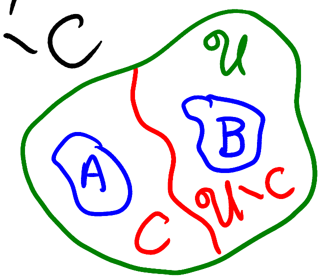


- only one simulation will return YES; rest will return NO or ABORT.

$\Rightarrow |\text{encoding}| = \Omega(\ell \sqrt{n} \lg n)$

Separators

- given universe $\mathcal{U}$ & a number m
- separator family $\mathcal{S}$ for size-m sets if $\forall A, B \subseteq \mathcal{U}$ with $|A|, |B| \le m$ & $A \cap B = \emptyset$: $\exists C \in \mathcal{S}$ such that $A \subseteq C$ & $B \subseteq \mathcal{U} \setminus C$



- claim: there is a separator family $\mathcal{S}$ for size-m sets with $|\mathcal{S}| \le 2^{O(m + \lg\lg|\mathcal{U}|)}$

  proof sketch: - perfect hash family $\mathcal{H}$ with $|\mathcal{H}| \le 2^{O(\lg m + \lg\lg|\mathcal{U}|)}$

  gives mapping from A & B to $O(m)$-size table

  - store bit in each table entry: A vs. B
  - $2^{O(m)}$ such bit vectors

  $\Rightarrow 2^{O(m)} \cdot 2^{O(\lg m + \lg\lg|\mathcal{U}|)} = 2^{O(m + \lg\lg|\mathcal{U}|)}$ $\square$

Encoding: $R \cap W$ + separator of $R \setminus W$ & $W \setminus R$

size $= |R \cap W| \cdot O(\lg n) + O(|R| + |W| + \lg\lg n)$

$= \Omega(\ell\sqrt{n}\lg n)$

$\Rightarrow |R \cap W| = \Omega(\ell\sqrt{n})$ or $|R| + |W| = \Omega(\ell\sqrt{n}\lg n)$

$\Rightarrow$ claim $\qquad\qquad \Rightarrow \Omega(\lg n)$ directly

Formally: if all ops. $= o(\lg n) \Rightarrow |R| + |W| = o(\lg n)$ then claim holds