TODAY: Dynamic graphs III (of 3)
- dynamic connectivity lower bound:
  - block operations
  - bit-reversal bad access sequence
  - tree over time
- sum lower bound
- connectivity lower bound
Dynamic connectivity lower bound:

[Patrascu & Demaine - SICOMP 2006]

Inserting/deleting edges & connectivity queries require \( \Omega(\lg n) \) cell probes/op.

Even if connected components are paths even amortized (but here prove for worst case)

\( \Rightarrow \) link-cut & Euler-tour trees are optimal

Proof:

- Consider \( \sqrt{n} \times \sqrt{n} \) grid with perfect matching between columns \( i \) & \( i+1 \) for each \( i \).

  - Block operations:
    - Update \((i, \pi)\): \( \pi_i \mapsto \pi \)
      \(= O(\sqrt{n}) \) edge deletions & insertions
    - Verify-Sum \((i, \pi)\): \( \sum_{j=1}^{i} \pi_j = \pi \) ?

- Claim: \( \sqrt{n} \) updates + \( \sqrt{n} \) verify sums require \( \Omega(\sqrt{n} \cdot \sqrt{n} \cdot \lg n) \) cell probes

  \( \Rightarrow \) \( \Omega(\lg n) \)/op.
Bad access sequence:
- for i in bit-reversal sequence:
  - verify-sum(i, \sum_{j=1}^{i} \pi_j) \Rightarrow answer=\text{yes (but DS must check)}
- update(i, \pi_{\text{random}}) \text{ uniform random permutation}
- build tree over time:

- left & right subtrees of each node interleave

- Claim: for every node v in tree, say with l leaves in its subtree, during right subtree of v \text{ (time interval)} must do \Omega(l \sqrt{n}) expected cell probes reading cells last written during left subtree

- sum lower bound over all nodes:
  - read r of write w only counted at lca(r,w)
  - linearity of expectation

\Rightarrow \Omega(n \log n) \text{ lower bound total (each leaf in } \Theta(lg n) \text{ subtrees)}
Proof of claim:
- left subtree has \( l/2 \) updates with \( l/2 \) rand. perms.
- any encoding of these permutations must use \( \Omega(l\sqrt{n} \log n) \) bits [information/Kolmogorov theory]
- if claim fails, find smaller encoding \( \Rightarrow \) contradict.
- setup: know the past (before \( v \)'s subtree)
- goal: encode (verified) sums in right subtree
  \( \Rightarrow \) can recover (updated) perms. in left subtree

\[ \begin{array}{ccccccc}
\phi & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array} \]

\[ \pi_i = \pi_{i-1}^{-1} \circ \cdots \circ \pi_1^{-1} \circ \pi_j \circ \pi_{i+1}^{-1} \]

- farther left \( \Rightarrow \) known \( \frac{\text{right}}{q} \) not yet updated

Warmup: query is \( \text{sum}(i) \Rightarrow \frac{\sum_{j=1}^{i}}{\pi_j} \) (partial sums)
- let \( R = \{ \text{cells read during right subtree} \} \)
  \( W = \{ \text{cells written during left subtree} \} \)
- encode \( R \cap W \) (address & contents of each cell)
  \( \Rightarrow |R \cap W| \cdot O(\log n) \) bits \( \Rightarrow w = \Theta(\log n) \)
- decoding alg. for sums in right subtree:
  - simulate sum queries in right subtree
  - to read cell written in right subtree: easy
    in left subtree: \( R \cap W \) in past: known

\[ \Rightarrow |R \cap W| \cdot O(\log n) = \Omega(l\sqrt{n} \log n) \]
\[ \Rightarrow |R \cap W| = \Omega(l\sqrt{n}) \]
Verify-sum instead of sum:
- permutations π given to verify-sum encode the information we want
- setup:
  - know (fixed) past
  - don’t know updates in left subtree
  - don’t know queries in right subtree
  - but know that queries return YES
- decoding idea:
  - simulate all possible input permutations for each query in right subtree
  - know one returns YES, all others NO
- trouble: incorrect query simulation
  - reads cells R’ ≠ R
  - if read r ∈ R’ \ R, it must be incorrect
  - but can’t tell whether r ∈ W \ R or past \ (R \ W)
  - can’t afford to encode R or W
- idea: encode separator S for R \ W & W \ R
- when decoding, to read cell written in right subtree: easy
  - in \ W: encoded explicitly in S: must be in past ⇒ known
  - not in S: must not be in R ⇒ incorrect; ABORT
  - only one simulation returns YES; rest NO or ABORT
⇒ recover desired permutation
⇒ |encoding| = \Omega(l\sqrt{n} \log n)
Separators:
- Given universe $U$ & number $m$
- Separator family $\mathcal{S}$ for size-$m$ sets if
  \[ \forall A, B \subseteq U \text{ with } |A|, |B| \leq m \text{ & } A \cap B = \emptyset : \]
  \[ \exists C \in \mathcal{S} \text{ such that } A \subseteq C \text{ & } B \subseteq U \cap C \]
- Claim: \[ \exists \text{ separator family } \mathcal{S} \]
  \[ \text{with } |\mathcal{S}| \leq 2^{O(m+\log \log U)} \]
- Proof sketch:
  - Perfect hash family $\mathcal{H}$ with $|\mathcal{H}| \leq 2^{O(m+\log \log U)}$
    \[ \text{[Hagerup & Tholey - STACS 2001]} \]
  gives mapping from $A$ & $B$ to $O(n)$-size table
  - Store $A$-or-$B$ bit in each table entry
  - $2^{O(m)}$ such vectors
  \[ \Rightarrow 2^{O(m)} \cdot 2^{O(m+\log \log U)} = 2^{O(m+\log \log U)} \]

Encoding: $R \cap W$ + separator of $R \cap W \cup W \setminus R$
- Size: $|R \cap W| \cdot O(\log n) + O(|R| + |W| + \log \log n)$
  \[ = \Omega(\sqrt{n} \log n) \]
  \[ \Rightarrow |R \cap W| = \Omega(\sqrt{n}) \]
  \[ \text{or } |R| + |W| = \Omega(\sqrt{n} \log n) \Rightarrow \Omega(\log n) \text{ for opt.} \]
Update-query trade-off: (possible by same technique)

\[ t_q \lg \frac{t_u}{t_q} = \Omega(\lg n) \quad \& \quad t_u \lg \frac{t_q}{t_u} = \Omega(\lg n) \]

- for \( t_u = \Omega(t_q) \), trees can match (small mods. to link-cut trees)
- for \( t_u = \Omega(\lg n \ (\lg \lg n)^3) \), can match [Thorup-STOC 2000]