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 $i$,
forming permutation $\pi_i$
- block operations:
  - **update** $(i, \pi)$: $\pi_i \leftarrow \pi$
    
    $\Rightarrow O(\sqrt{n})$ edge deletions & insertions
  - **verify-sum** $(i, \pi)$: $\sum_{j=1}^{i} \pi_j = \pi$?
    
    compose
    $\Rightarrow O(\sqrt{n})$ connectivity queries
- **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(\(i, \sum_{j=1}^{\frac{i}{2}} \pi_j\)) \(\Rightarrow\) answer = yes (but DS must check)
  - update \(i, \pi_{\text{random}}\) uniform random 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 \(v\) (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 \(\text{lca}(r, w)\)
  - linearity of expectation
  \(\Rightarrow\) \(\Omega(n \lg n)\) lower bound total (each leaf in \(\Theta(\lg n)\) 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 \([\text{information/Kolmogorov theory}]\)
- 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

\[
\begin{array}{ccccccc}
\pi_1 & \pi_2 & \pi_3 & \pi_4 & \pi_5 & \pi_6 & \pi_7 \\
\varnothing & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}
\]

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

- farther left \( \Rightarrow \) known
- not yet updated

Warmup: query is \( \sum(i) \rightarrow \sum_{j=1}^{i} \pi_j \) \( (\text{partial sums})\)
- 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 \([\text{assume poly. space}]\)
  \( \Rightarrow w = \Theta(\log n) \)

- 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

\( \Rightarrow |R \cap W| \cdot O(\log n) = \Omega(l \sqrt{n} \log n) \)
\( \Rightarrow |R \cap W| = \Omega(l \sqrt{n}) \)
\( \checkmark \)
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 \( R' \neq R \)
  - if read \( r \in R \setminus R' \), it must be incorrect
  - but can’t tell whether \( r \in W \setminus R \) or past \( (R \cap W) \)
  - can’t afford to encode \( R \) or \( W \)
- idea: encode separator \( S \)
  - for \( R \setminus W \) & \( W \setminus R \)
- when decoding, to read cell written in right subtree: easy
  - in \( R \cap W \): encoded explicitly
  - in \( S \): must be in past \( \Rightarrow \) known
  - not in \( S \): must not be in \( R \) \( \Rightarrow \) incorrect; ABORT
- only one simulation returns YES; rest NO or ABORT
\[ \Rightarrow \text{recover desired permutation} \]
\[ |\text{encoding}| = \Omega(\sqrt{n} \cdot \log n) \]
Separators:
- given universe \( U \) & number \( m \)
- separator family \( \mathcal{S} \) for size-\( m \) sets if
  \( \forall A, B \subseteq U \) with \( |A|, |B| \leq m \) & \( A \cap B = \emptyset \):
  \( \exists C \in \mathcal{S} \) such that \( A \subseteq C \) & \( B \subseteq U \setminus C \)
- claim: \( \exists \) separator family \( \mathcal{S} \) with \( |S| \leq 20(m + \log \log |U|) \)
- proof sketch:
  - perfect hash family \( \mathcal{H} \) with \( |\mathcal{H}| \leq 20(m + \log \log |U|) \)
    [Hagerup & Thorup - STACS 2001]
    gives mapping from \( A \) & \( B \) to \( O(n) \)-size table
  - store \( A \) or \( B \) bit in each table entry
  - \( 20m \) such vectors
  \( \Rightarrow 20m \cdot 20(m + \log \log |U|) = 20(m + \log \log |U|) \)

Encoding: \( R \cap W + \) separator of \( R \cap W \& W \setminus R \)
- size:
  \( |R \cap W| \cdot O(\log n) + O(|R| + |W| + \log \log n) \)
  \( = \Omega(\log n \cdot \log n) \)
  \( \Rightarrow |R \cap W| = \Omega(n \cdot \log n) \)
  or \( |R| + |W| = \Omega(\log n \cdot \log n) \)
  \( \Rightarrow \Omega(\log n) \) for op.
Update-query trade-off: (possible by same technique)
\[ t_q \cdot \log \frac{t_u}{t_q} = \Omega(\log n) \quad \& \quad t_u \cdot \log \frac{t_q}{t_u} = \Omega(\log n) \]

- for \( t_u = \Omega(t_q) \), trees can match (small mods. to link-cut trees)
- for \( t_u = \Omega(\log n \ (\log \log n)^3) \), can match

[Thorup-STOC 2000]