Tree decompositions:
- preferred paths
- heavy-light decomp.
- separator decomp.
- ART decomp. / leaf trimming

- Tango trees \([L2]\)
- & link-cut trees \([L16]\)
- & many other apps
- tree search \([TODAY]\)
- level ancestors \([TODAY]\)

Separator theorem on trees: [Jordan 1864 - "Sur les assemblages de lignes"—J. Reine Angew. Math.]

Any tree on \(n\) vertices has a vertex whose removal disconnects the tree into components of size \(\leq n/2\)

Proof:
- pick any vertex \(v\)
- if not done, (exactly) one component of \(T-v\) has size \(> n/2\)
- walk one step into that component
- new component containing \(v\) has size \(< n/2\)

\(\Rightarrow\) never go back to \(v\) \(\Rightarrow\) terminate \(\square\)
Separator decomposition:
- apply separator theorem $\Rightarrow$ root of new tree
- recurse on components $\Rightarrow$ children subtrees
$\Rightarrow$ depth of new tree $= O(lg n)$

Tree search: [Ben-Asher, Farchi, Newman - SICOMP 1999]
- looking for a cat (node) in a tree
- given an edge $(v, w)$, oracle tells you which subtree (of $v$ or $w$) has cat
- separator decomp. lets you find cat in $O(\Delta lg n)$ oracle calls for max. degree $\Delta$
- $O(\Delta \frac{lg n}{lg \Delta})$ possible [Laber & Nogueira - ENDM 2001]
- best tree can be found in $O(n)$ time [Moses, Onak, Weimann - SODA 2008]
- applications:
  - file system synchronization
  - bug detection in tree of modules
  - search in a dynamic Voronoi diagram of points in convex position [Aronov, Bose, Demaine, Gudmundsson, Iacono, Langerman, Smid - LATIN 2006]
ART decomposition: [Alstrup, Husfeldt, Rauhe-FOCS 1997]

- define bottom tree rooted at each maximally high node with \( \leq \lg n \) leaves below
  \( \Rightarrow \) bottom trees are disjoint & each has compressed size \( O(\lg n) \)
  (similar trick in LCAs [Lecture 8/9])

- top tree on remaining nodes
  \( \Rightarrow \) has \( \leq n/\lg n \) leaves
  (charge a top leaf to \( >\lg n \) leaves in bottom trees below)

- recurse in top tree
  \( \Rightarrow \lg n/\lg \lg n \) recursive levels
Marked ancestor problem: [ART 1997]
- given rooted tree ~ here, static
- each node can be marked or unmarked
- updates: mark(v) & unmark(v)
- query: lowest marked ancestor of v

Bounds:
- \(O\left(\frac{\log n}{\log \log n}\right)\) query, \(O(\log \log n)\) update [TODAY]
- \(\Theta\left(\frac{\log n}{\log \log n}\right)\) query assuming \(\log \Theta(1) n\) update
- "chronogram technique" [Spring '05, L17]
- \(\Omega(\log \log n)\) update, even in a path: colored predecessor problem with \(u = n\)

Application: dynamic method dispatching in OOP
- tree = inheritance among classes
- mark = class implements method \(x\)
- query = call to \(x\)

OPEN: DAGs, for multiple inheritance
Marked ancestor upper bound:
recursive ART decomposition

**Query:** each bottom tree:
- has \( \leq \log n \leq w \) leaves \( \Rightarrow \) \(<w \) branches
- maintain bit vector of which
  compressed edges (nonbranching paths)
  have a marked node
- partially ordered by ancestry (e.g. pre-order)
- each node stores bitmask of its
  ancestor compressed edges
\( \Rightarrow \) mask & least significant 1 bit
  EITHER locates desired compressed edge
  OR says must be in top tree in \( O(1) \) time
- maintain van Emde Boas DS \( (u=n) \)
  on each compressed edge
\( \Rightarrow \) in bottom case, \( O(\log \log n) \) to finish query
- in top case, recurse on top tree
  with parent of root of bottom tree
\( \Rightarrow O(\frac{\log n}{\log \log n} + \log \log n) \) query

**Update:**
- each node stores which bottom tree it's in
- predecessor update for compressed edge
+ bit vector update for bottom tree
\( \Rightarrow O(\log \log n) \)
Decremental connectivity in a tree:
updates = edge deletions
query = is there a path v \rightarrow w? or what's root of v's component?
  = marked ancestor problem with mark only
  - assume all n-1 edges eventually deleted

① O(lg n) amortized update, O(1) query
  (also possible with link-cut/Euler tour trees [L16])
  - each node stores connected component i.d.
  - delete(v, w):
    - run DFS from both v & w in parallel
    - stop when one DFS stops \Rightarrow smaller comp.
    - update all nodes in smaller component to new i.d.
  - component containing an updated node shrinks by 2x
  \Rightarrow O(lg n) updates to any node
  (similar trick in union-find DS)
  - can also store i.d. \rightarrow root of component mapping
② O(1) amortized for a path
- split into chunks of length \( \log n \)
- store each chunk as a bit vector
- use (1) to store which chunks have a cut
  \( \Rightarrow O(\sqrt{n/\log n}) \) updates cost \( O(\log n) \)
- query: find right chunk in \( O(1) \) via (1) shift & least sig. 1 bit within a chunk

③ O(1) amortized for top tree
- use (1) on \( O(\sqrt{n/\log n}) \) compressed paths
- use (2) on each nonbranching path

④ O(1) amortized for bottom trees
- maintain bit vector of which \( \log n \) compressed paths have a cut
  \( \leq \log n \)
- again partially ordered by depth
- preprocess mask for ancestors of each node
- query within compressed path = mask, LSB
- use (2) on each nonbranching path

⑤ nonrecursive ART decomposition with (3) & (4)