Lecture 7: Dynamic Graphs I

Updates: edge insertion / deletion
Queries: connectivity (u, v)
all on some undirected graph G

Variants:
- fully dynamic
- partially dynamic:
  - incremental (no delete)
  - decremental (no insert)

Known results

On trees/forests:
- $O(\log n)$ worst case per operation → TODAY
- decremental: $O(1)$ amortized (Alstrup et al.)

On planar graphs:
- $O(\log n)$ (Eppstein, Galil, Italiano, Spencer 1996)

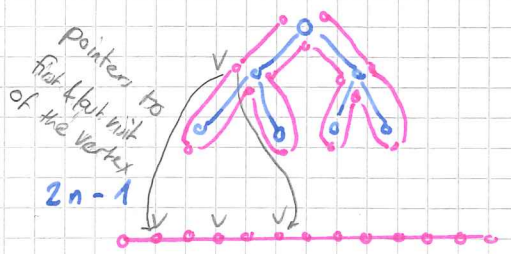
On general graphs:
- incremental: union-find $O(\alpha(m, n))$ per op. (Tarjan, 1975)
- decremental: $O(m \log n + \text{polylog}(n) n + \#\text{queries})$ for all op. (Thorup 1997)
- fully-dynamic:
  - dream/ideally: $O(\log n)$ per operation → OPEN
  - $O(\log^3 n)$ per update, $O(\log n / \log\log n)$ per query (amortized) (Holm et al. → TODAY
  - $O(\sqrt{n})$ per update, $O(1)$ per query (Eppstein et al. 1997)
  - $O(n^{0.49})$ per update (Wulff-Nilsen 2017)
- lower bounds:
  - $\Omega(\log n)$ per query (Patrascu & Demaine 2004/2007)

Dynamic Connectivity on Trees

two known solutions in $O(\log n)$/op.:
- link-cut tree (Sleator & Tarjan) — not discuss, useful for path aggregates
- Euler tour tree (Henzinger & King 1995) → what we will discuss, useful for subtree aggregates

operations:
- make_tree(v)
- link(u, v)
- cut(v): delete (v, parent(v)) edge
- find_root(v)

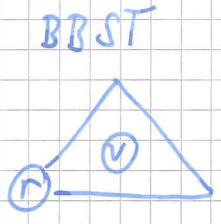
Euler tour around tree



→ keep balanced binary search tree on this Euler tour, ordered by order in tour

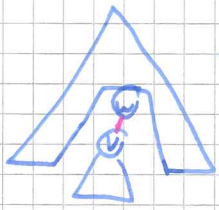
Implementing the operations of Euler tour trees:

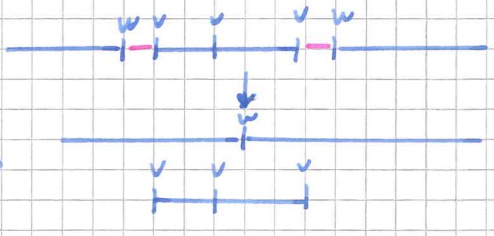
- find_root(v):
  - start in first visit of v
  - walk up to root of BST
  - walk down the left spine



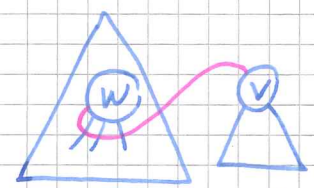
$O(\log n)$ time

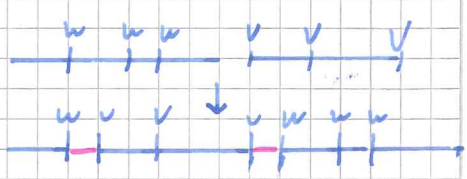
- cut(v)
  - split of BST in $O(\log n)$ time
  - remove 1x (v,v)
  - join the two trees for v





- link(u, v)
  - split after first occurrence of v
  - add one occurrence of v
  - join with u's tree





(if v is not the root of its tree, we cyclically shift its Euler tour to make it the root (and carefully update all the first/last pointers))

- connectivity(u,v)
  find_root(u) = find_root(v) ?

- to aggregate vertex values over subtrees by sum/min/max, just use the BBST and change any occurrence of a vertex in the tour

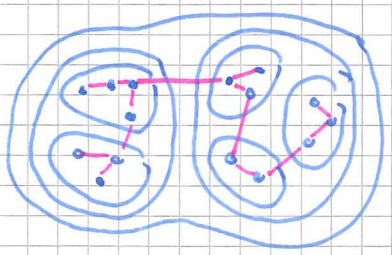
Dynamic Connectivity on General Graphs in $O(\log^2 n)$ amortized time (Holm et al.)

ideas:
- maintain spanning forests
- use amortization to pay for search of replacement edges

Hierarchically divide connected components of G
- each edge is on some „level", there are $\log n$ levels → partition of E
- G_i = subgraph of G of edges with level $\leq i$

$$G_0 \subseteq G_1 \subseteq G_2 \subseteq \dots \subseteq G_{\log n} = G$$

- Invariant 1: every connected component of G_i has $\leq 2^i$ vertices
- F_i = spanning forest of G_i s.t.
- Invariant 2: $F_i = F_{\log n} \cap G_i$

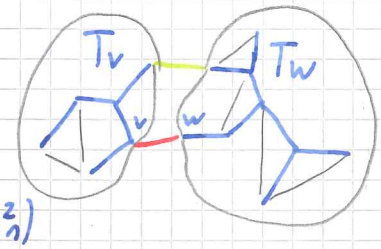


Operations:

insert($e=(v,w)$)
- add e to incidence lists of v, w
- $e.level = \log n$
- if v, w are disconnected in $F_{\log n}$, add e to $F_{\log n}$

query(u,v):
- Euler tour tree query on $F_{\log n}$

delete($e=(v,w)$)
- remove e from incidence lists
- if $e \notin F_{\log n}$ → done
- if $e \in F_{e.level}$
  - delete e from $F_{e.level}, \dots, F_{\log n}$ → costs $O(\log^2 n)$



// if there is a replacement edge at level x, it is good on $F_x, F_{x+1}, \dots, F_{\log n}$

for $i = e.level, \dots, \log n$
- T_v, T_w be the trees of F_i containing v & w, w.l.o.g. $|T_v| \le |T_w|$ *
- as $|T_v| + |T_w| \le 2^i$, $|T_v| \le 2^{i-1}$, we can afford to push i-level edges within $G[T_v]$ down to $i-1$
- for each edge $e'=(x,y)$ with $x \in T_v$ and $level=i$ **
  - if $y \in T_w$: add e' into $F_i, F_{i+1}, \dots, F_{\log n}$ & stop
  - else: $e'.level = i-1$ (since $y \in T_v$)
    add e' to F_{i-1} if it was part of T_v

* checking this requires subtree aggregates in the Euler tour tree of F_i

** we also need different subtree aggregates for this: namely, number of i-edges incident to vertices within each subtree of the Euler tour of T_v. This way, we can skip empty subtrees (subtrees where all edges are of level $< i$ and thus can not be pushed down further) quickly, allowing for $O(\log n)$ per edge e'.

Cost: $O(\log^2 n)$ + $O(\log n)$ * #level decreases → $O(\log^2 n)$ amortized
- $O(\log^2 n)$: cuts in all F_i
- $O(\log n)$: single F_i operation
- #level decreases $\le$ #insertions $\cdot \log n$