Dynamic Graphs

Graphs model: knowledge, social networks, road networks, molecular structures, electrical circuits, the internet...
Any of these prone to change? Yes!
Motivates the study of *dynamic graphs*.

Dynamic graph algorithms:

- build a data structure for the graph
- update the data structure when edges appear or disappear
- use the data structure to answer questions about the graph.

Example: Connectivity.

- Can you compute the connected components of a graph? (and how fast?)
- If only edges appear (never leave), can you update connectivity info?
- Updating connectivity in general is highly non-trivial.
- Today: Updating connectivity in a forest.

Dynamic forest connectivity

Model: We have at all times a forest.
An edge may join or leave the forest.
Want to answer questions of the form: "is u in the same forest as v?"

Answer: Sleator & Tarjan's Link-Cut Trees.
Able to update and answer connectivity information in $O(\log n)$ **worst-case** time, where n is the number of vertices.
Bonus: With a little extra work, it can be used to maintain all sorts of extra information about the trees and about relations between vertices.

First:

- Why is *polylogarithmic time*, i.e. $O((\log n)^c)$ time, so attractive?
- Warm-up case: Forest of paths.

Polylogarithmic time. The P-class of dynamic graphs.

Dynamic graph on n vertices, we generally have the following operations:

- delete(e) deleting an edge e,
- insert(u, v) inserting an edge between u and v,
- relation(u, v) asking about some relation between u and v.

An example of a relation could be connected (are u and v connected? Yes/no), but it could also be lowest common ancestor or max flow or shortest path or ....

Input size. How many bits to encode an operation?
$O(1)$ to encode which operation it is, plus at most $O(\log n)$ to encode which vertex, vertex pair, or edge is involved. = total $O(\log n)$ bits.

So a polynomial-time algorithm would process these in $O((\log n)^c)$ time.
For some constant c. (We call this polylogarithmic time.)

Question: If a problem admits a polylogarithmic time solution for dynamic graphs, does this say anything about the time it takes to answer the problem for *static* (i.e. non-dynamic) graphs?
Bonus info: Not all near-linear-time solvable problems admit polylogarithmic update time dynamic solutions! Even "planar dijkstra" takes $\Omega(\sqrt{n})$ to update.
Research question: Which problems are polylog-updatable?

Warm-up case: connectivity in a forest of paths

Model We have at all times a forest of paths.
An edge may join or leave the forest of paths.
Query: connected(u, v).

Idea: Build a balanced binary tree over each path.
Connectivity: u and v are connected if they are in the same tree. $O(\log n)$ time.
Cut(u,v): Tree surgery. $O(\log n)$ time.
E.g. rotate u to be root, cut child containing v, rebalance.
Link(u,v): Two cases.

- linking a head and a leaf – concatenate paths – easy. $O(\log n)$.
- otherwise, we need to invert the direction of one path. Invert:
  - Set a lazy mirroring bit.
  - When visiting internal node x, dissolve lazy bit by switching left and right child, and pushing lazy bit to left and right child.

Exercise: Can you use/extend this data structure to answer connectivity queries (connected(u, v))? Distance queries (dist(u, v))? Minimum edge-label on path? Sum of edge-labels on path? (Solve exercise 2 and 1).

Model: We have at all times a forest of trees.

Idea: Heavy path decomposition.

- Vertex v has heavy child c if weight of subtree is large: $w(c) > \frac{1}{2}w(v)$

Connected: u and v can each find the root of their tree by following $\log n$ light edges, and $\log n$ head-of-heavy-path calls. $O(\log n)$ total time.
Link: If v is root of its tree, then linking u and v is easy. Simply add v as a child of u. Are we done? No, because now u is heavier. How many light edges may become heavy? At most $\log n$ on the path from u to its root. $O(\log n)$.
Now we need two things: make-root, and cut.
Cut: Simply deleting the edge between v and its parent u may cause the opposite problem to that above. Up to $\log n$ edges may become light.
Idea: Define for each heavy edge e an indicator lefttilt(e) of how far the heavy edge is from being light. Navigate the balanced binary tree to find the "least heavy" heavy edge. If its heaviness is smaller than $w(v)$, lighten it, recurse. We can store information about lightness lazily like flip.
Make-root(v): Pretend weight of v is ∞. Now root's heavy path: $r \rightarrow v$. Flip direction of that path. (Undo pretend ∞.)
Flipping direction: Need to maintain both lefttilt and righttilt! (Implicitly.)

Connectivity (et al.) in general trees. Result.

Given a dynamic graph that is at all times a forest,
we may perform link, cut, and answer connected queries
in worst-case $O((\log n)^2)$ time.
In fact, a closer analysis shows, worst-case $O(\log n)$ time.

Bonus: We may augment the path decomposition to maintain extra information.
The balanced trees over the heavy paths can be used to answer questions about segments of the paths.

Exercise: Can you use/extend the data structure to answer lowest common ancestor queries? Distance queries? Minimum edge-label on path?

Future work

Connectivity if edges only arrive – easy.
In a forest of paths – easy'ish.
In a forest of general trees – doable (today)
In general graphs – ... there are polylog-time algorithms that are either amortised, or in expectation, or both. Deterministic worst-case polylog?

One of the exercises was about colouring dynamic trees. Which k-colourable graphs may we efficiently, worst-case $k + O(\log n)$-colour? Open question!

In general, we may ask: Which near-linear time computable questions about graphs are updatable in polylogarithmic time? Which are not?

Dynamic graphs allows us to study fundamental graph problems – again – from a new perspective. (More dynamic graphs next week.)