Dynamic Graph Algorithms

Solve standard algorithmic problems in the setting where graphs change over time.

**Example:** dynamic connectivity
- Insert \((u,v)\) \(\Rightarrow\) insert only?
- Delete \((u,v)\) \(\Rightarrow\) delete only?
- Connected \((u,v)\)? connected \((G)\)?

Basic question, many results, but not completely resolved yet.

Clearly if arbitrary updates @ each operation \(\Rightarrow\) can run DFS each time.
So try minimizing update time vs query time.

- Fredericson SCMP 85
  \(O(m^{1/2})\) update \(O(1)\) query
  deterministic
- Eppstein et al.
  \(O(n^{1/2})\) update \(O(1)\) query
  HW.
- Holm de Lichtenberg Thorup '98
  \(O(\text{polylog})\) update \(O(\text{log}|G|)\) query
  random
- Kaplan Kiry Monemzay
  \(O(\text{polylog})\) update \(\Rightarrow\)

Let's see some ideas used in these papers today.

**Theorem:** Let \( \text{Domain} \) be the update/query tradeoff:

\[
\text{Update} \times \log \left(\frac{T_{\text{query}}}{T_{\text{update}}}\right) \leq \Omega(\log n).
\]

\[
\text{Ternary } \log \left(\frac{T_{\text{query}}}{T_{\text{update}}}\right)
\]

from randomized algo (Monte Carlo algo).
Frederickson (A weaker $m^{3/3}$ result).

Sp's insert-only. Then union-find does the job. Insert = $O(n)$, implicitly maintains a spanning forest.

Query = few finds.

But delete: hmm: suppose we maintain a spanning forest and delete an edge in tree.

How to find a replacement edge. (if delete off-tree edge, no worries).

Idea: Cluster the tree into pieces of size $2$. (same parameter).

Assumption 1: $G$ has $\circ$ degree $\leq 3$.

Replace high degree nodes by gadgets of cycles

Now if new edge draw

Insert (deletes $1$ edge in original graph)

$\Rightarrow$ insert delete of $O(1)$ edges in new graph.

New graph has $m = \Theta(n)$ and $g$ degree $\leq 3$. 
Fact 2: Given a tree $G$ with maximum degree $\leq 3$, can $G$ be marked so that non-marked edges form components of size $\in \{2,3\}$?

**Proof:** Direct each edge from smaller subtree to larger.

If there is exactly one vertex without degree 0, then deleting at least one of these in-edges, for some tree $T$, gives components of size $\geq 2$ on both sides. Recurse.

Actually, can do this in time linear in the tree. (How? Exercise.)

Call this a 2-bounded cluster of the tree. (If tree has < 2 nodes, all wts in the same cluster.)

We maintain: a spanning forest $F$ of $G$.

- a 2-bounded cluster of each tree.
- for each pair of clusters, 1 bit for whether some tree $E$ is in $G_i \cap G_j$ not in $F$.

Inserts are easy (exercise!) so look at deletes.

If off-tree edge deleted, just update the bit for the clusters $(G_i,G_j)$ this edge goes between. (Must belong to some tree in $F$, since $F$ spanning forest.)
So delete ee F. (say ee T).

Sps e within clustee (other case similar)

Try to find a replacement edge
between left & right clusters.

- At most $O\left(\frac{m}{2}\right)$ clusters $\Rightarrow$ for each look up bit $\Rightarrow O\left(\frac{m}{2}\right)$ time.

But careful! No record of $(C_e, C')$ edges
since $C_e \cup C_r = C$ was original clustee.

No prob! Only $O(\tau)$ edges hit $C_e \cup C_r = C$, so scan all of them in $O(\tau)$ time.

- Finally, either replacement edge or not. Either case $C_e, C'$ maybe too small.
  So merge with some neighbor cluster and re-partition if too big
  Again $O(\tau)$ time.

Total time: $O\left(\left(\frac{m}{2}\right)^2 + \tau\right)$. Balance to get $\tau = O(m^{2/3})$. 😊

Using more careful data structures, get $O(m^{2/3})$. [same paper].

Epstein et al gave clever technique (very general) to reduce to $O(m^{2/3})$.

[Homework]
Randomized Idea (Very high level sketch)

Suppose we have just one deletion. Want fast worst case time, only that error probability is $o(1)$.

Here's idea: each vertex name in $O(\lg n)$ bin: $l(x) \leftarrow \text{"label of } x\text{"}.$

edge $e = (u, v)$ represented by $\langle l(u), l(v) \rangle \overset{def}{=} l(e)$
(assume $u < v$ according to some total order on vertex names).

Maintain spanning forest on $G$. Now get delete($e=(u,v)$) operation.
(assume tree $T$)

Need replacement edge between $L \cup R$.

Suppose: Only one edge in $\text{Cut}(L \cup R) \setminus \{e\}$. Unique replacement edge.

Then: $\forall x \in L \cup R \left( l(e') = l(x) = \bigoplus_{e' \in \text{incidents of } x} l(e') \right)$.

What if there are many edges crossing from $L \cup R$ in $G \setminus e$?

Sample! For each $j$, sample $\frac{1}{2} l(e)$ and maintain the $\bigoplus l(e)$. 

Maintain zero crossing.
for each such sample. (let $\oplus (e)$ be the "signature" of $x$.)

This will work if $T_a$ unique edge from $L \to R$ in the sample. So if $F$ edges across
the cut, $P_r[unique\ edge\ survives] \leq \sum_{e \in E_{\delta}(x)} \frac{1}{2^e} \cdot (1 - \frac{1}{2^e})^{|e - 1|} \leq \Theta(1)$.

Now if repeat $O(\log n)$ times, one sample succeeds up to $1 - \frac{1}{\log n}$!!

---

Does not quite work immediately for multiple deletions. If the tree after the
first delete depends on the sample, and now subsequent calculations cannot use
independence the same way.

But Kapron King Moniweg [MMAA12] find a way around it.

Don't do this in class.

---

The Holm et al. idea for amortized polylog (deterministic) bound.

Again, how to find replacement edges? Can scan all edges of $L \cup R$ but may
not do this repeatedly! Need some way to measure progress.

Idea: Each edge is kept at some "level" $\in [0, \log_2 n]$. "Chop" scanning edges to
raising the level of some edge.

Insert: edge added at level 0.

Delete: if $e$ at level $k < \log_2 n$ just level + 1.

lo at tree $T < e$, splits into

$L \cup R$. Say $|L| \leq |R|$

raise edges of $L$ to level $k + 1$
(those edges of level $e$).

II Maintains invariant II

Invariants: $E_0 \leq E_1 \leq E_2 \leq E_3

\begin{enumerate}
\item $F_i$ is spanning forest of $E_i$

$F_{i - 1} \subseteq F_i$

ie. if $xy \in E_i$ connected $\Rightarrow$ $x, y$ connected in $F_i$

\item each component in $F_i$ has $\leq n^c_i$ nodes
\end{enumerate}
Now start scanning edges incident to $V(L)$. Those of level $l$.

If it goes within $L$, raise to level $l+1$.

If it goes to $R$, done. Stop, and add to $F_2, F_3, \ldots F_d$.

Start from level $l$ edges, then go to level $l-1$.

Now $F_{l-1}$ has tree $T_{l-1}$ containing $e$ that has split into $L_{l-1}$ & $R_{l-1}$.

Again if $|L_{l-1}| \leq |R_{l-1}|$, raise edges of level $l-1$ in $L_{l-1}$ to level $l$,

and start scanning level $l-1$ edges incident to $V(L_{l-1})$.

---

Note we maintain invariants.

(Raising levels of edges does not harm Inv I, may mess with Inv II, so that is why
we raise levels carefully ensuring that size of any subtree when levels raise)

---

Time analysis:

We were kinda vague about how we maintain the forests. Basically each $F_i$ is
maintained by Sleator/Tarjan Link/Cut trees. When edge added, use Link.

When delete, use Cut. And then for each edge during the scan, use membership
tests of LC trees to test if edge goes across (and use Link if it does).

Fact: Amortized cost of these operations $O(\lg n)$.

Edge edge $O(\lg n)$ times, $\Rightarrow O(n^2 \lg n)$ per update.

Query time: check if $(x,y)$ in same component, $O(\lg n)$ time.

---

Takeaway ideas: "charge to smaller side". For Holm et al also.

"power of XOR" and "power of sampling" for KKM.

"clustering and tree separator for Frederikson."