Dynamic Graph Connectivity

Ian Chen

University of Illinois Urbana-Champaign

Preliminaries

Euler-Tour Trees

Dynamic Graph Connectivity

Preliminaries

Euler-Tour Trees

Dynamic Graph Connectivity

Problem Statement

- Queries:
  - `Connected`(u, v), return whether there is a path between u and v
  - `Connected`(G), return whether G is connected
- Updates:
  - `Insert`(u, v), add (undirected) edge uv to the graph
  - `Remove`(u, v), remove edge uv from the graph

Example

1. `Insert`$(0, 1)$
2. `Insert`$(2, 3)$
3. `Connected`$(0, 1) = \textsc{True}$
4. `Connected`$(0, 2) = \textsc{False}$
5. `Insert`$(1, 2)$
6. `Connected`$(1, 3) = \textsc{True}$
7. `Remove`$(2, 3)$
8. `Connected`$(2, 3) = \textsc{False}$

Naive Methods

- $O(1)$ update time, $O(m)$ query time?
- $O(m)$ update time, $O(1)$ query time?
- $O(\log^* m)$ insert time, $O(\log^* m)$ query time?
  - What about deletions?

Bounds

- $\max\{ t_{\text{update}}, t_{\text{query}} \} = \Omega(\log n)$ (Pătraşcu and Demaine 2004)
- $O(n^{1/2})$ update time, $O(1)$ queries (Eppstein et al. 1997)
- Polylog update and query time (Kapron, King, and Mountjoy 2013)

Preliminaries

Euler-Tour Trees

Dynamic Graph Connectivity

Spanning Trees and Forests

- Represent each connected component with a *spanning tree*
- A collection of spanning trees that contains every vertex is a *spanning forest*
- Answering queries is easy given a spanning forest supporting:
  - `Find`(u), find the root of the tree that u is in
  - `Size`(u), number of nodes in the tree containing u
- u, v connected iff `Find`$(u) =$ `Find`(v)
- G connected iff `Size`$(v) = n$.

Linking and Cutting Trees

- Change the spanning forest in a dynamic graph:
- `Link`(u, v), merge the trees `Find`(u) and `Find`(v)
- `Cut`(u, v), split the tree `Find`$(u) =$ `Find`(v) across uv

Euler Tours

- An Euler-Tour is a closed walk that visits every edge exactly once.
- An (undirected, connected) graph has an Euler-Tour if and only if every vertex has even degree
- Trees never have Euler Tours (why?)
- Doubling each edge, the tree has an Euler Tour

Euler Tours on Trees I

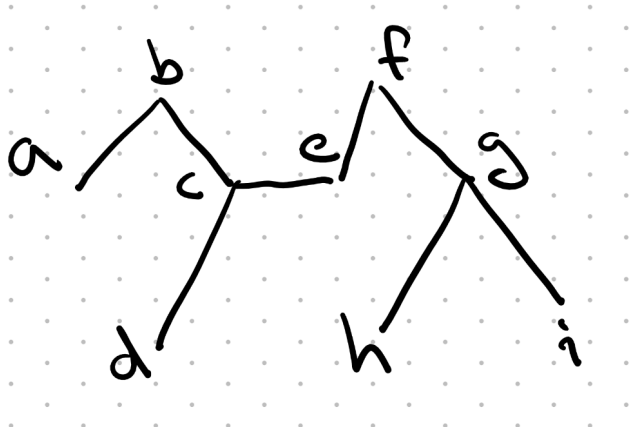

Figure: **Euler Tree on Trees:** (a) a spanning tree T on 9 nodes.

Euler Tours on Trees II

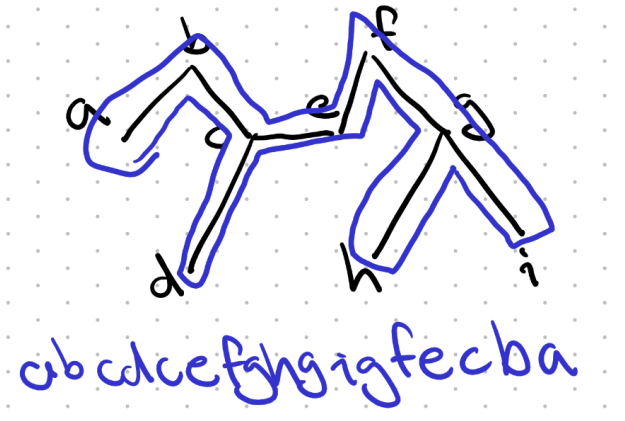

Figure: **Euler Tree on Trees:** (b) an Euler Tour on the tree T

Euler Tours on Trees III

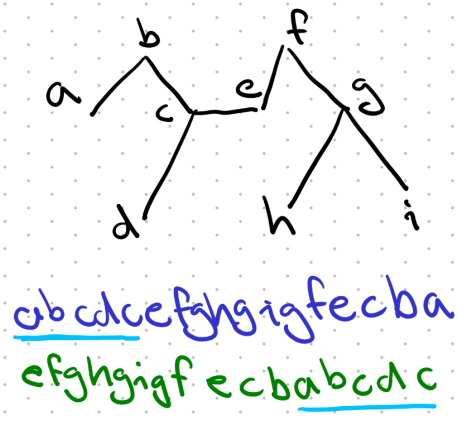

Figure: **Euler Tree on Trees:** (c) Rotations of Euler Tour changes roots.

Euler Tours on Trees IV

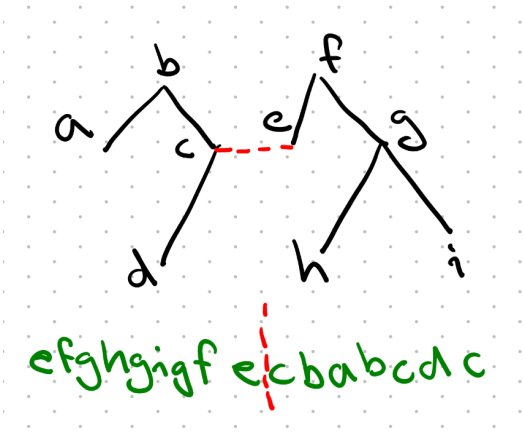

Figure: **Euler Tree on Trees:** (d) Subtrees are contiguous in Euler Tour.

Euler-Tour (ET) Trees

- Represent the sequence as a self-balancing tree
- `Find`(u)
  - $O(\log n)$ time; each node stores parent pointers
- `Size`(u)
  - $O(\log n)$ time; each node stores size of its subtree
- `Link`(u, v)
  - $O(\log n)$ time; merge two trees and update sizes
- `Cut`(u, v)
  - $O(\log n)$ time; remove edge from tree and update sizes

Euler-Tour (ET) Trees

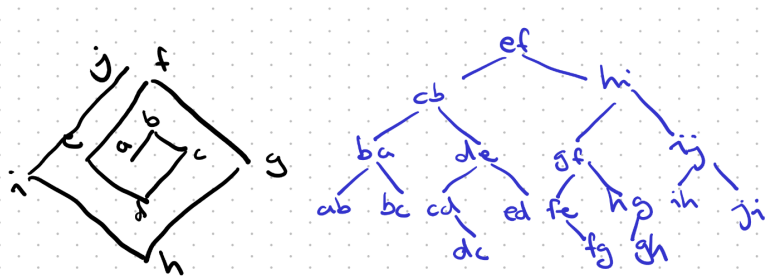

Figure: **ET Tree:** Left: a tree of depth 9. Right: balanced tree representation of Euler Tour of depth 4.

Inserting and Removing Edges

- Upon `Insert`(u, v), use the `Link`(u, v) operation
- Upon `Remove`(u, v), use the `Cut`(u, v) operation to form L and R subtrees
  - BUT, there may be another edge $u'v'$ connecting the L and R
  - We must also call `Link`(u', v')
  - How do we check existence and find this connecting edge?

Preliminaries

Euler-Tour Trees

Dynamic Graph Connectivity

XOR Fingerprint

- Let $\ell(v)$ label the vertices from $0, 1, \ldots, n-1$ with $\log_2(n)$ bits
- Let $\ell(uv)$ (for $u < v$) be concatenation of label $[\ell(u), \ell(v)]$
- Let $\mathcal{F}(v) = \bigoplus_{e \in N(v)} \ell(e)$ be the fingerprint of vertex v
- For any $(S, \bar{S})$ cut of vertices:

$$\bigoplus_{v \in S} \mathcal{F}(v) = \bigoplus_{e \in E(S,\bar{S})} \ell(e)$$

- Computing $\bigoplus_{v \in S} \mathcal{F}(v)$ exactly the same as computing `Size`

XOR Fingerprint (Continued)

- If $E(L, R) = \emptyset$, then $\bigoplus_{v \in L} \mathcal{F}(v) = 0$
- If $E(L, R) = \{ e \}$, then $\bigoplus_{v \in L} \mathcal{F}(v) = \ell(e)$
- How can we find an edge in $E(L, R)$ in general?
- Find a *subset* of E with exactly one frontier edge

Edgeset Sampling

- Suppose $|E(L,R)| = \{ e_1, \ldots, e_k \}$, with $2^i \le k < 2^{i+1}$
- Let E' be sampling edges iid with probability 2^{-i}
- $\mathbb{P}(\ |E'(L,R)| = 1) \ge 1/e^2 = \Theta(1)$

$$\begin{aligned}
\mathbb{P}\big(\ \big| E'(L,R)| = 1 \big) &= k\,\mathbb{P}\big(e_1 \in E', e_2 \notin E', \ldots, e_k \notin E' \big) \\
&= (k)2^{-i}(1-2^{-i})^k \ge (2^i)2^{-i}(1-2^{-i})^{2^{i+1}} \\
&= \Big((1-2^{-i})^{2^i}\Big)^2 \ge 1/e^2
\end{aligned}$$

- Create $\log m$ independent sampling rates for each i
- Create $O(\log n)$ independent copies to get high probability

Kapron, King, Mountjoy Algorithm I

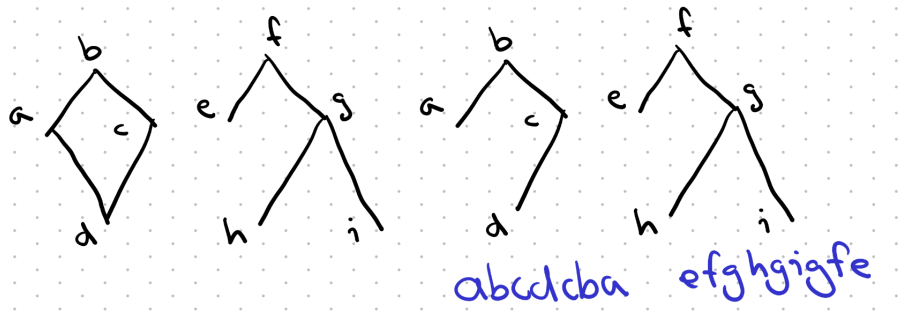

Figure: **KKM Algorithm:** (a) Left: graph G, Right: spanning forest F

Kapron, King, Mountjoy Algorithm II

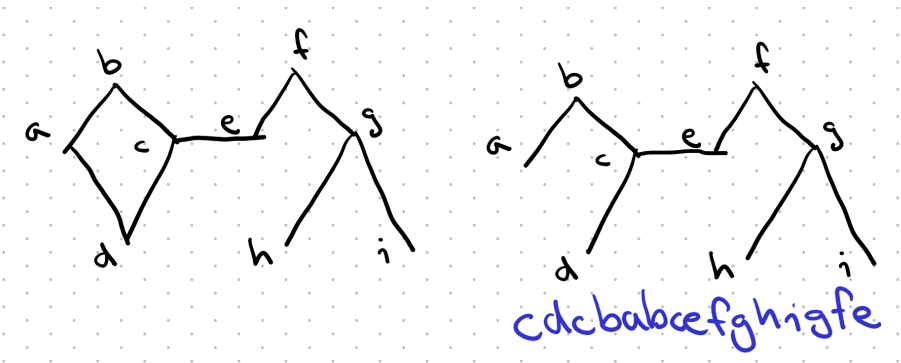

Figure: **KKM Algorithm:** (b) Insert edge ce into G. F becomes a single tree.

Kapron, King, Mountjoy Algorithm III

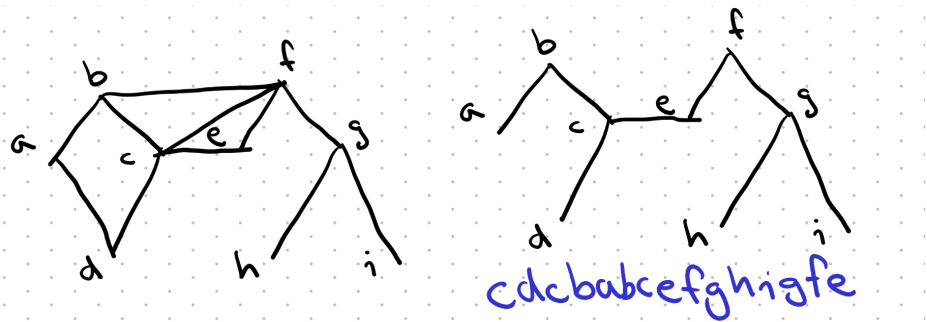

Figure: **KKM Algorithm:** (c) Insert edges cf and bf. F does not change.

Kapron, King, Mountjoy Algorithm IV

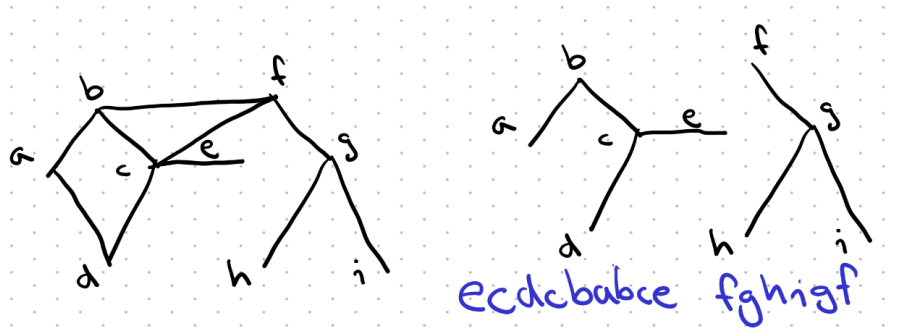

Figure: **KKM Algorithm:** (d) Remove edge ef. F cut into $abcde$ and $fghi$.

Kapron, King, Mountjoy Algorithm V

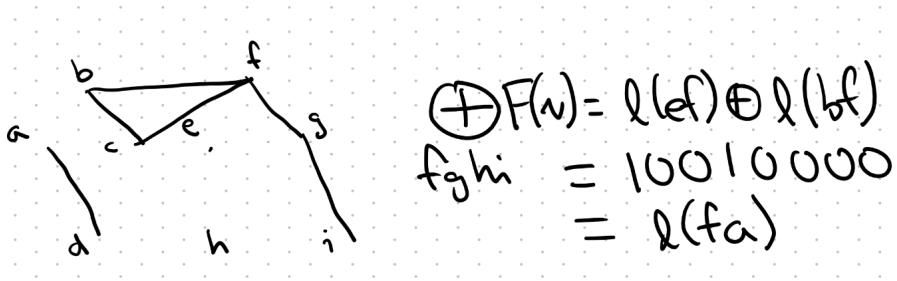

Figure: **KKM Algorithm:** (e) Fingerprints on subsampled edge set E_1. Fingerprint finds to find replacement edge fa not in E, so FAIL.

Kapron, King, Mountjoy Algorithm VI

Figure: **KKM Algorithm:** (f) Fingerprints on subsampled edge set E_2. Fingerprint finds to find replacement edge cf in E, so SUCCESS. Links forest back into one tree along cf.

Bibliography I

- Eppstein, David et al. (Sept. 1997). “Sparsification—a technique for speeding up dynamic graph algorithms”. In: *Journal of the ACM* 44.5, pp. 669–696. ISSN: 1557-735X. DOI: 10.1145/265910.265914. URL: http://dx.doi.org/10.1145/265910.265914.
- Kapron, Bruce M., Valerie King, and Ben Mountjoy (Jan. 2013). “Dynamic graph connectivity in polylogarithmic worst case time”. In: *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, pp. 1131–1142. DOI: 10.1137/1.9781611973105.81.
- Pătraşcu, Mihai and Erik D. Demaine (June 2004). “Lower bounds for dynamic connectivity”. In: *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. STOC04. ACM, pp. 546–553. DOI: 10.1145/1007352.1007435.