Dynamic Graph Algorithms

Giuseppe F. Italiano

University of Rome "Tor Vergata"

giuseppe.italiano@uniroma2.it

http://www.disp.uniroma2.it/users/italiano

(slides on my Web Page)

Outline

Dynamic Graph Problems – Quick Intro

Lecture 1. (Undirected Graphs)

Dynamic Connectivity

Lecture 2. (Undirected/Directed Graphs)

Dynamic Shortest Paths

Lecture 3. (Non-dynamic?)

2-Connectivity in Directed Graphs

Outline

Dynamic Graph Problems – Quick Intro

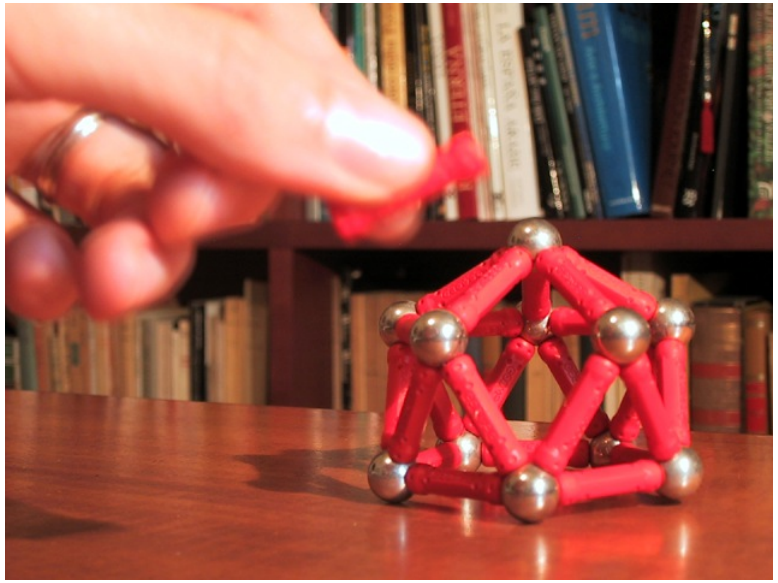
Lecture 1. (Undirected Graphs)
Dynamic Connectivity

Lecture 2. (Undirected/Directed Graphs)
Dynamic Shortest Paths

Lecture 3. (Non-dynamic?)
2-Connectivity in Directed Graphs

Dynamic Graphs

Graphs subject to update operations



Typical updates:

`Insert(u,v)`

`Delete(u,v)`

`ChangeWeight(u,v,w)`

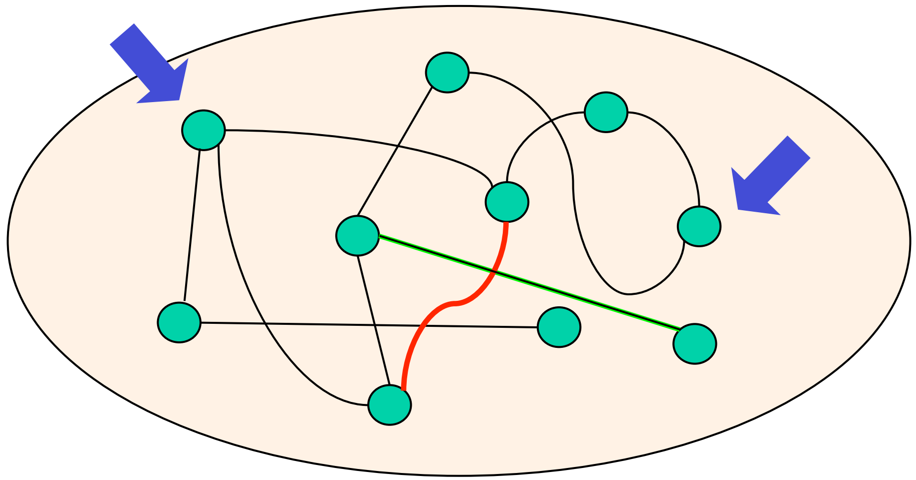
Dynamic Graphs

Initialize

Insert

Delete

Query



A graph

Dynamic Graph Algorithms

The goal of a dynamic graph algorithm is to support query and update operations as quickly as possible (usually much faster than recomputing from scratch).

Notation:

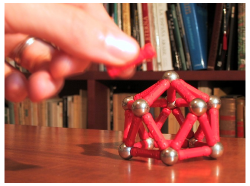
$G = (V,E)$

$n = |V|$

$m = |E|$

We will use also amortized analysis:

$$\frac{\text{Total worst-case time over sequence of ops}}{\text{\# operations}}$$



Dynamic Graphs

Partially Dynamic Problems

Graphs subject to insertions only, or deletions only, but not both.

Fully Dynamic Problems

Graphs subject to intermixed sequences of insertions and deletions.

Dynamic Graph Problems

Support queries about properties on a dynamic graph

- *Dynamic Connectivity (undirected graph G)*
  `Connected(): Connected(x,y):`
  Is G connected? Are x and y connected in G?

- *Dynamic Minimum Spanning Tree (undirected graph G)*
  Any property on a MST of G

Dynamic Graph Problems

- *Dynamic Transitive Closure (directed graph G)*
  `Reachable(x,y):`
  Is y reachable from x in G?

- *Dynamic All Pairs Shortest Paths*
  `Distance(x,y):`
  What is the distance from x to y in G?
  `ShortestPath(x,y):`
  What is the shortest path from x to y in G?

Dynamic Graph Problems

- *Dynamic Min Cut*

  `MinCut():` `Cut(x,y):`

  Min cut? Are x and y on the same side of a min cut of G?

- *Dynamic Planarity Testing*

  `planar():`

  Is G planar?

- *Dynamic k-connectivity*

  `k-connected():` `k-connected(x,y):`

  Is G k-connected? Are x and y k-connected?

Dynamic Graph Problems

- *Dynamic (Approximate) Maximum Matching*

  `Matching():`

  Maximum Matching?

  `ApproximateMatching():`

  Approximate Maximum Matching?

  `ValueofMatching():`

- *Dynamic (Approximate) Minimum Vertex Cover*

  `VertexCover():`

  Approximate Minimum Vertex Cover?

Outline

Dynamic Graph Problems – Quick Intro

Lecture 1. (Undirected Graphs)
Dynamic Connectivity

Lecture 2. (Undirected/Directed Graphs)
Dynamic Shortest Paths

Lecture 3. (Non-dynamic?)
2-Connectivity in Directed Graphs

Fully Dynamic Graph Connectivity

Maintain an undirected graph G under an intermixed sequence of operations of the following type:

- **insert(u,v) :** Add a new edge (u,v)
- **delete(u,v) :** Remove edge (u,v) from G (assumes (u,v) in G)
- **connected(u,v) :** Return yes if there is a path between u and v; return no otherwise

Subproblem (basic ingredient) in many other problems

Minimum spanning trees, 2-connectivity, …

Simple problem but lots of interesting ideas!

Applications in Computational Biology

Eyal, Halperin: Dynamic maintenance of molecular surfaces under conformational changes. Symposium on Computational Geometry 2005: 45-54

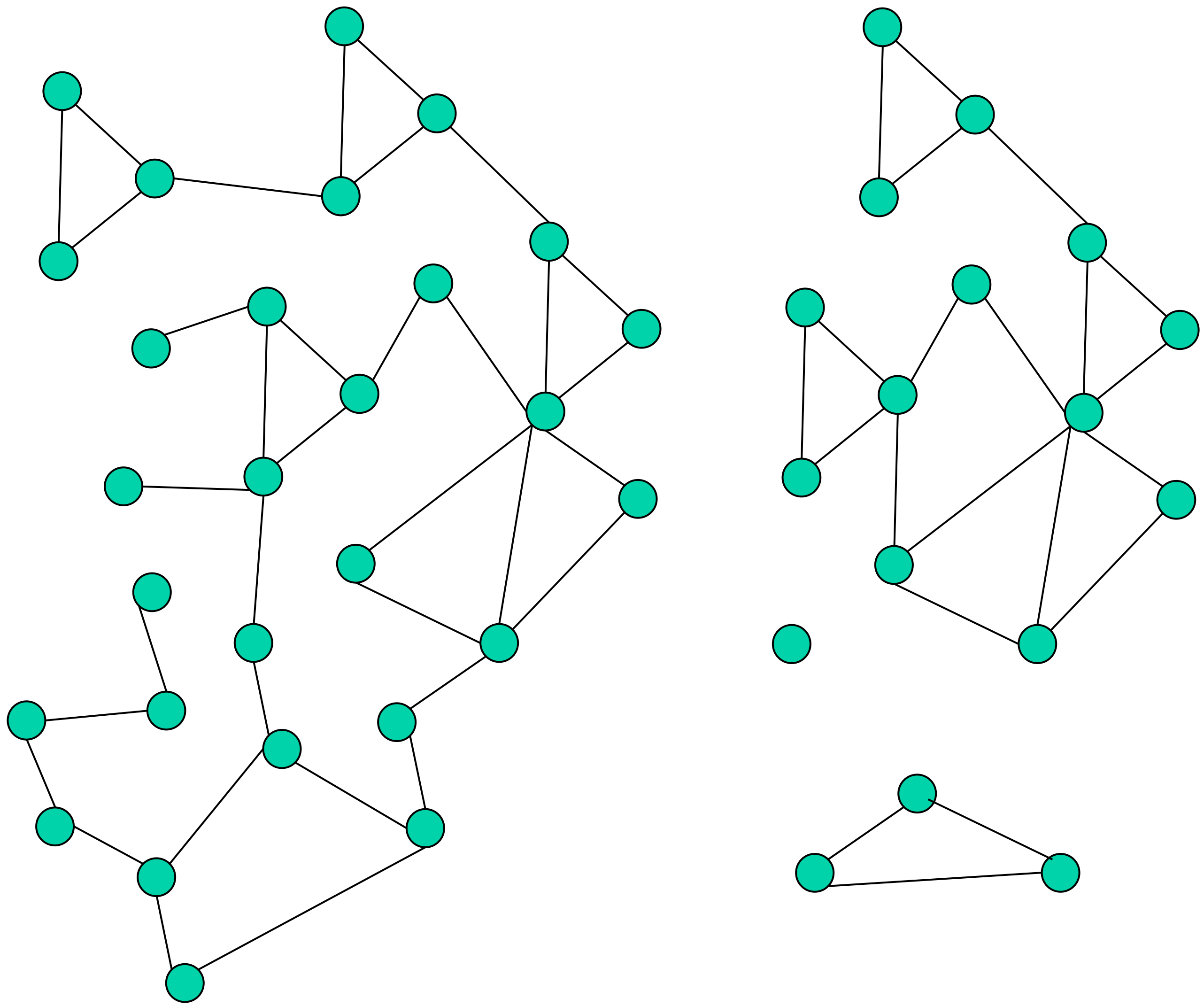
Eyal, Halperin: Improved Maintenance of Molecular Surfaces Using Dynamic Graph Connectivity. WABI 2005: 401-413

Bajaj, Chowdhury, Rasheed: A dynamic data structure for flexible molecular maintenance and informatics. SIAM/ACM Conference on Geometric and Physical Modeling 2009, 259-270, 2009

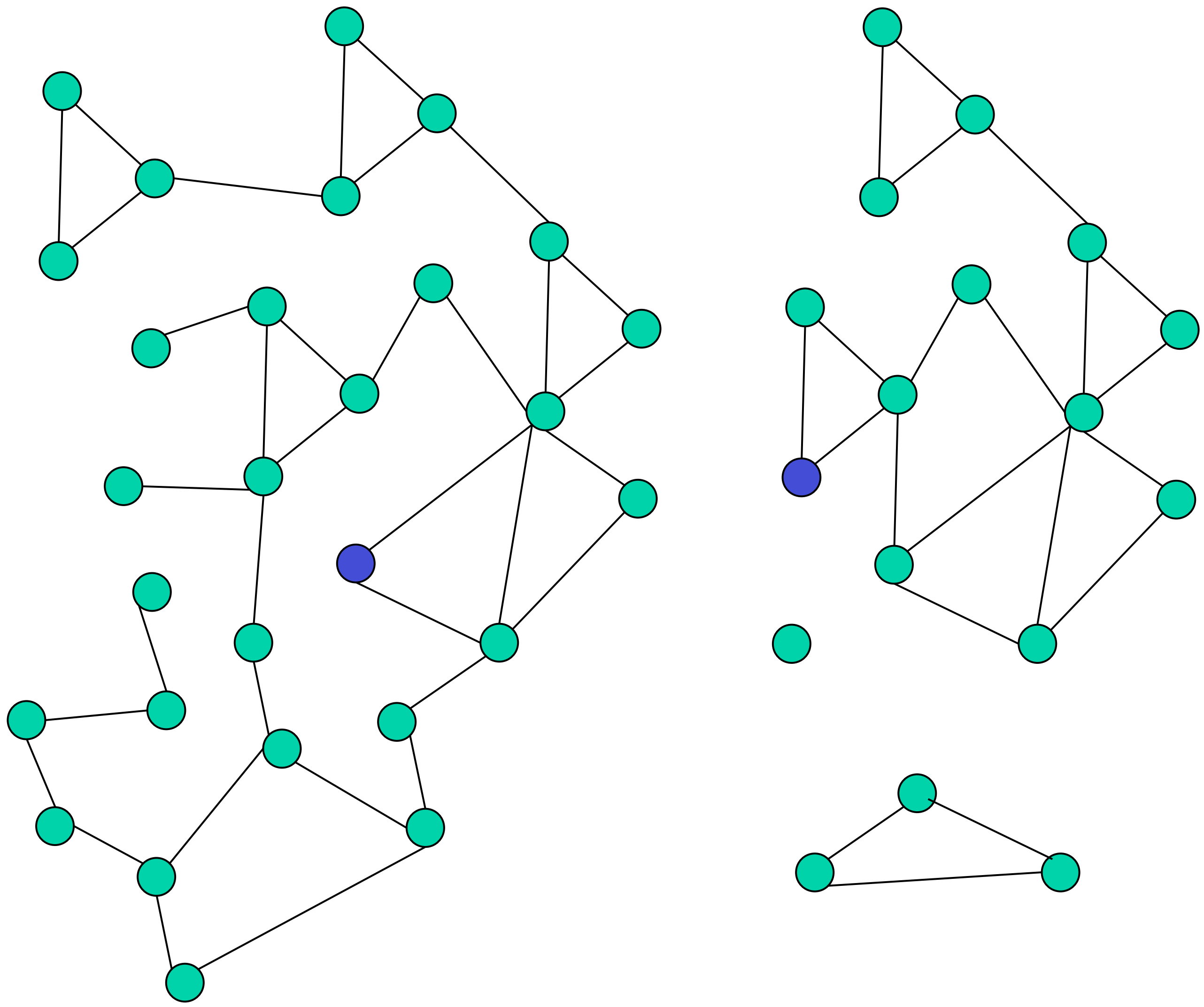
Ulitsky, Shamir: Identification of functional modules using network topology and high-throughput data. BMC Systems Biology 2007, 1:8

……

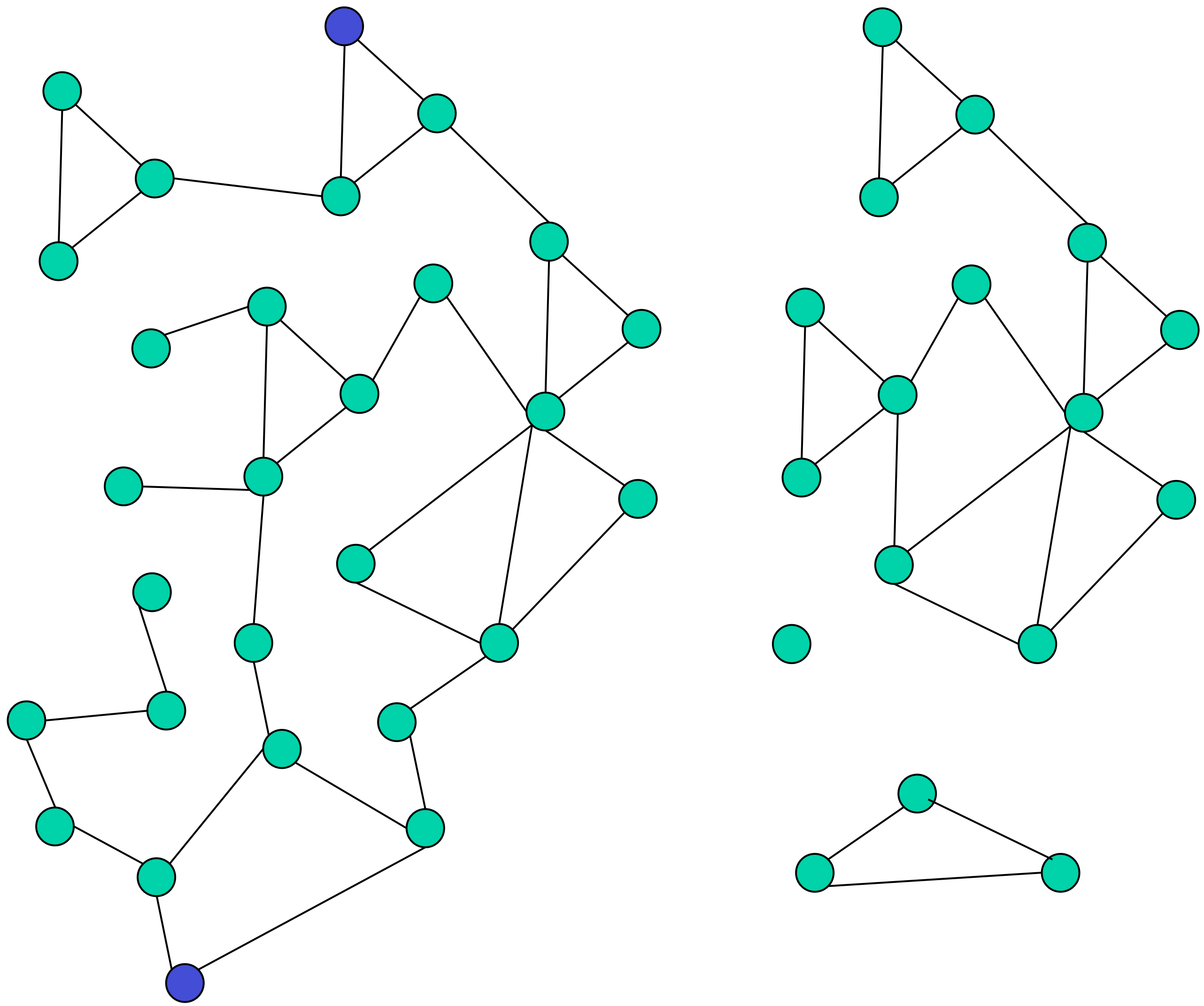
connected(v,w)



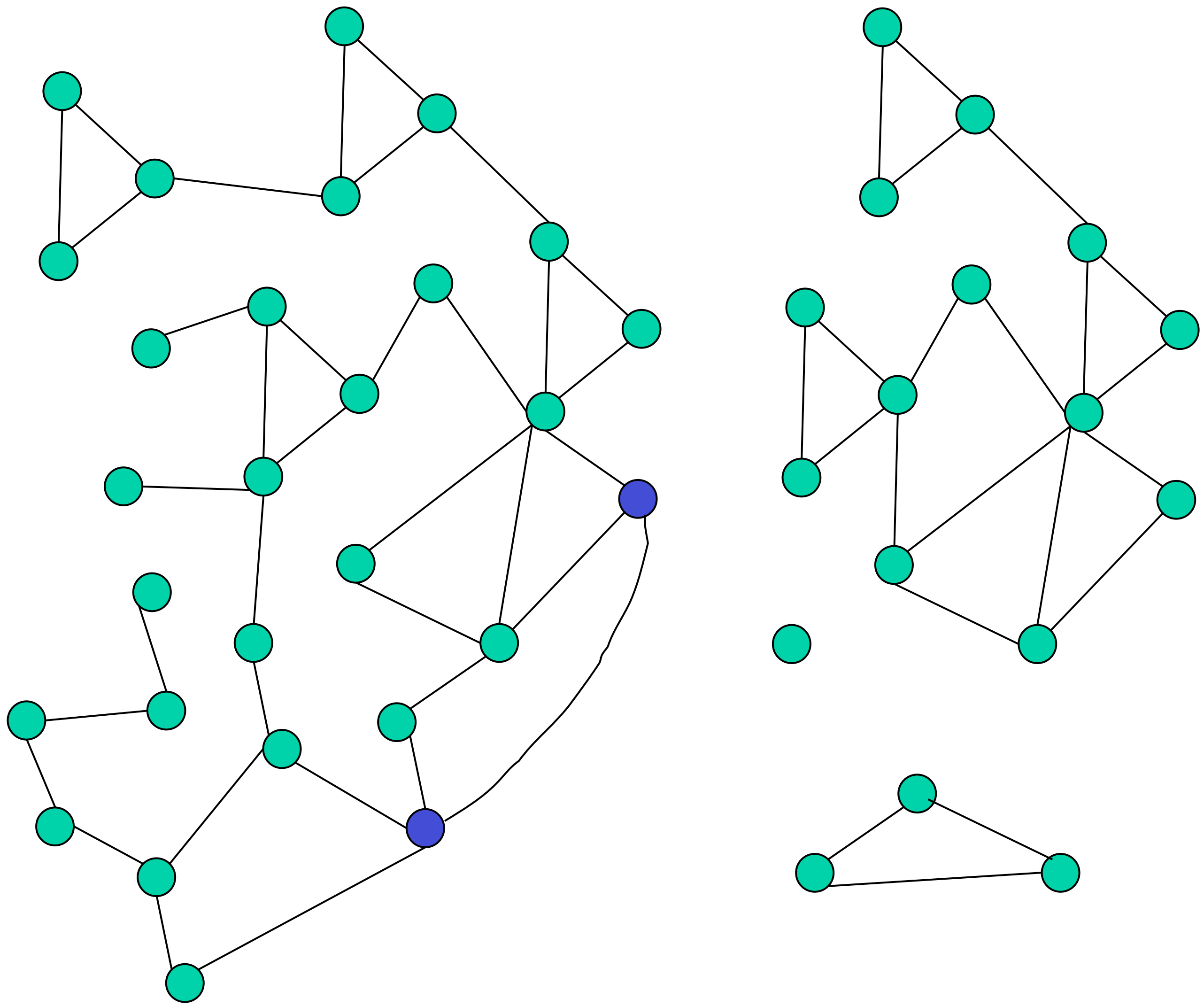
connected(v,w)



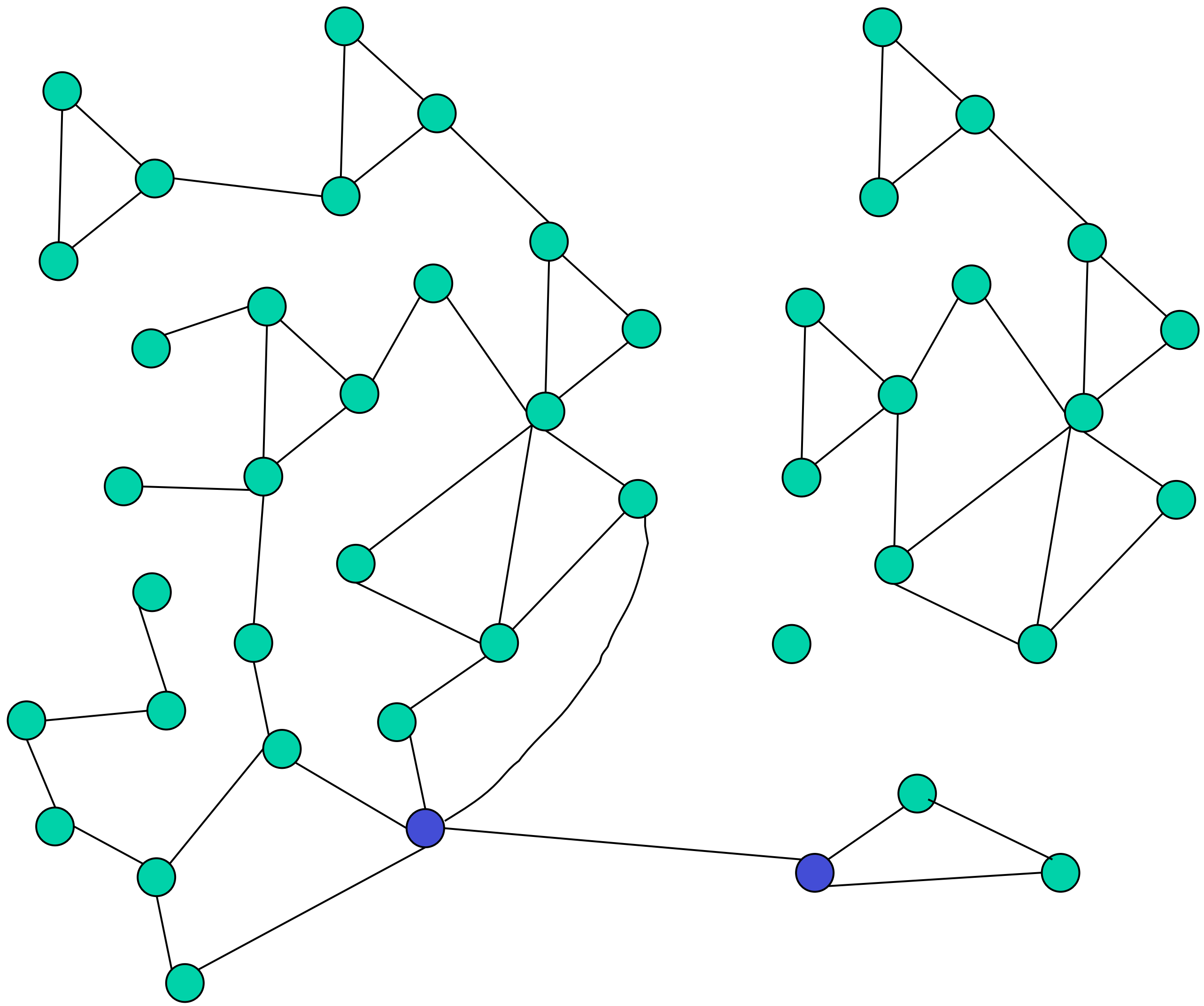
connected(v,w)



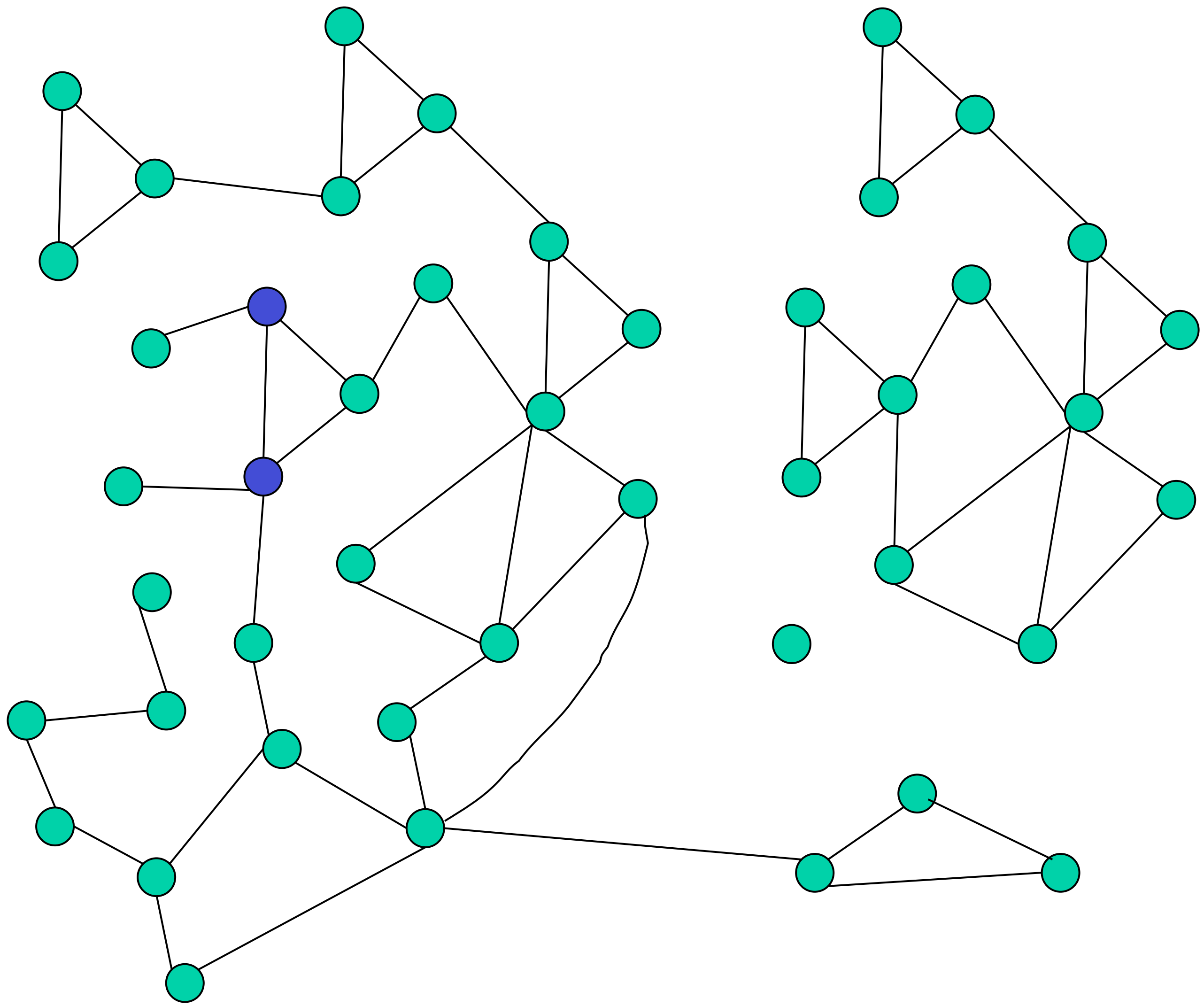
insert(v,w)



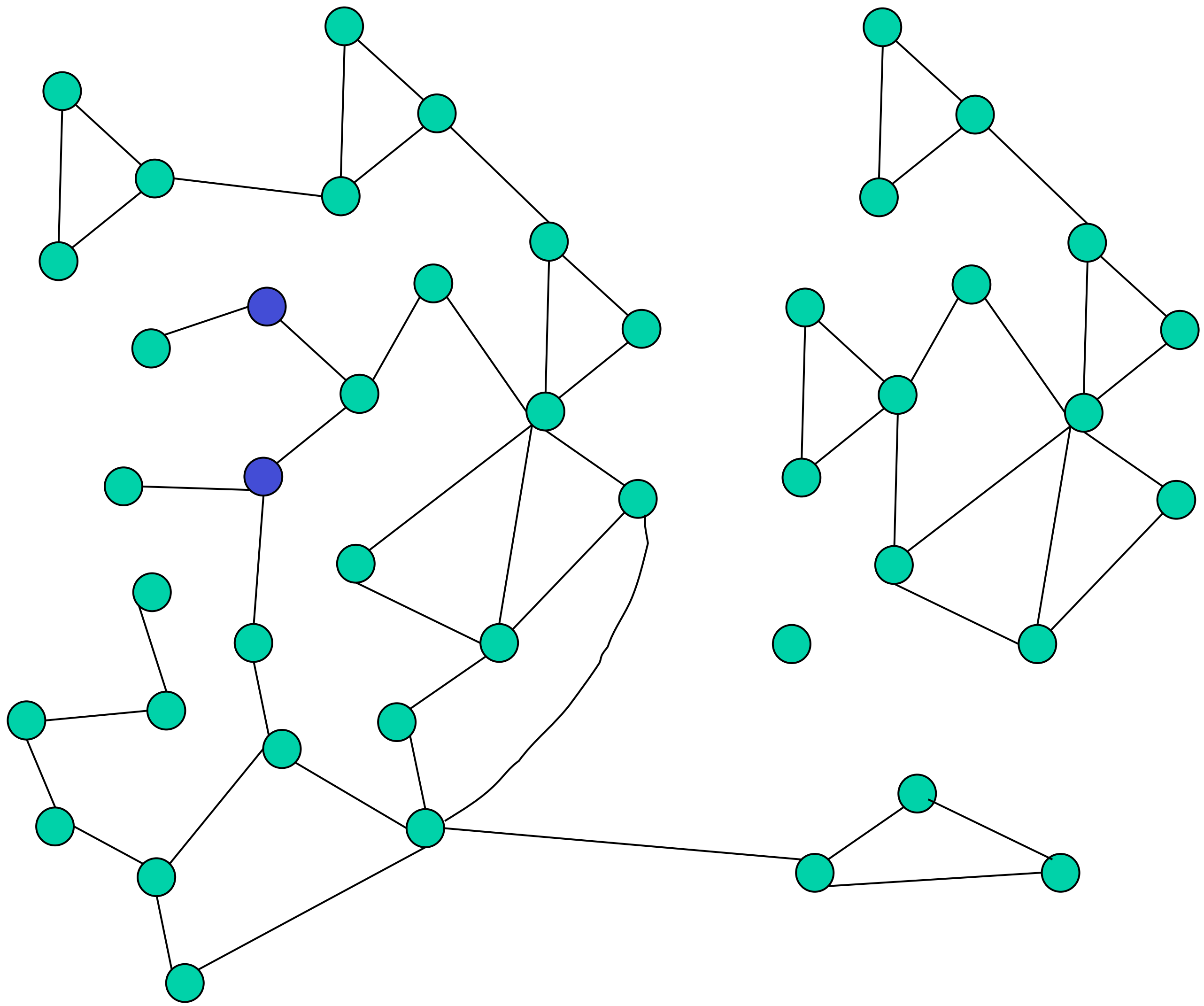
insert(v,w)



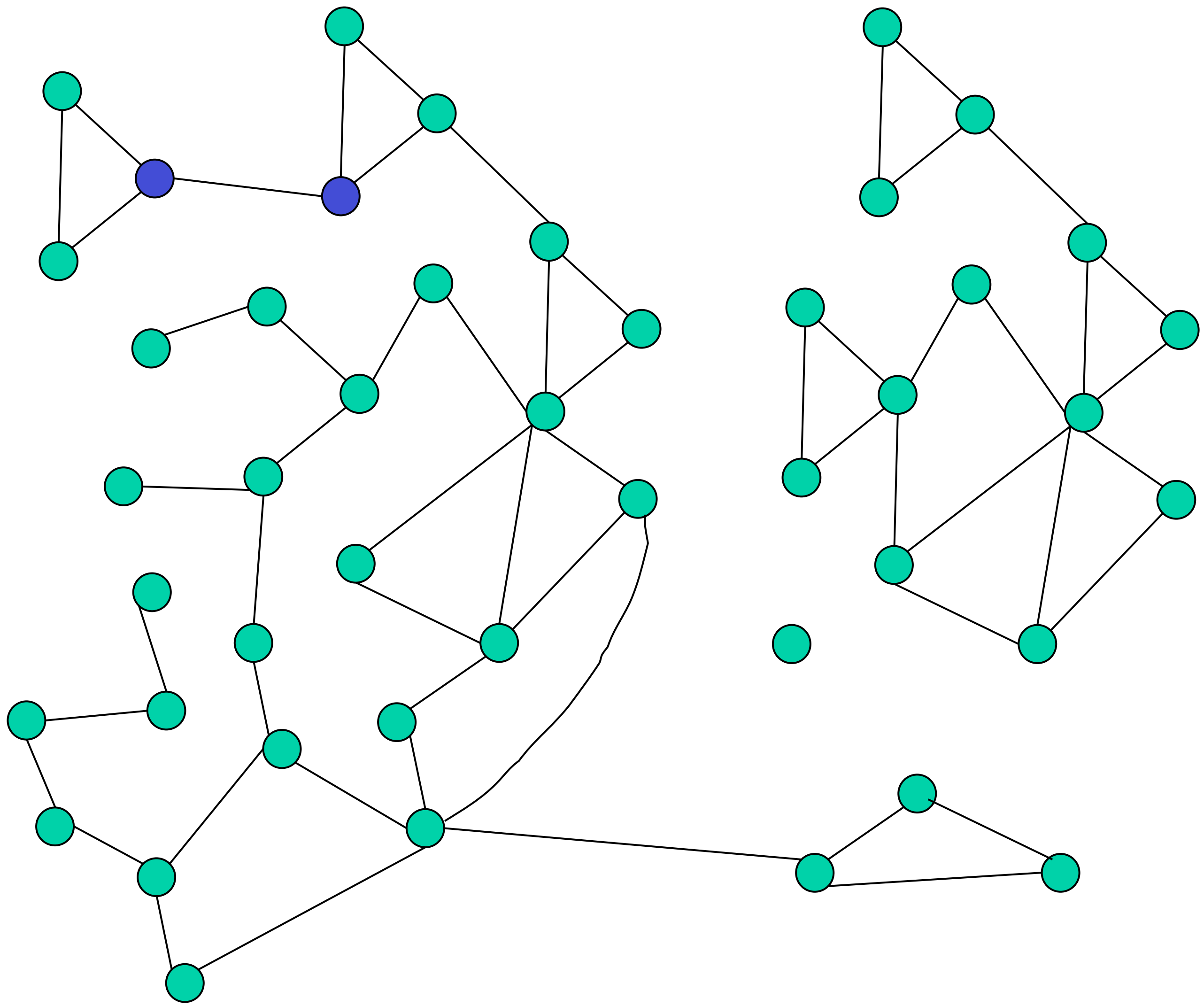
delete(v,w)



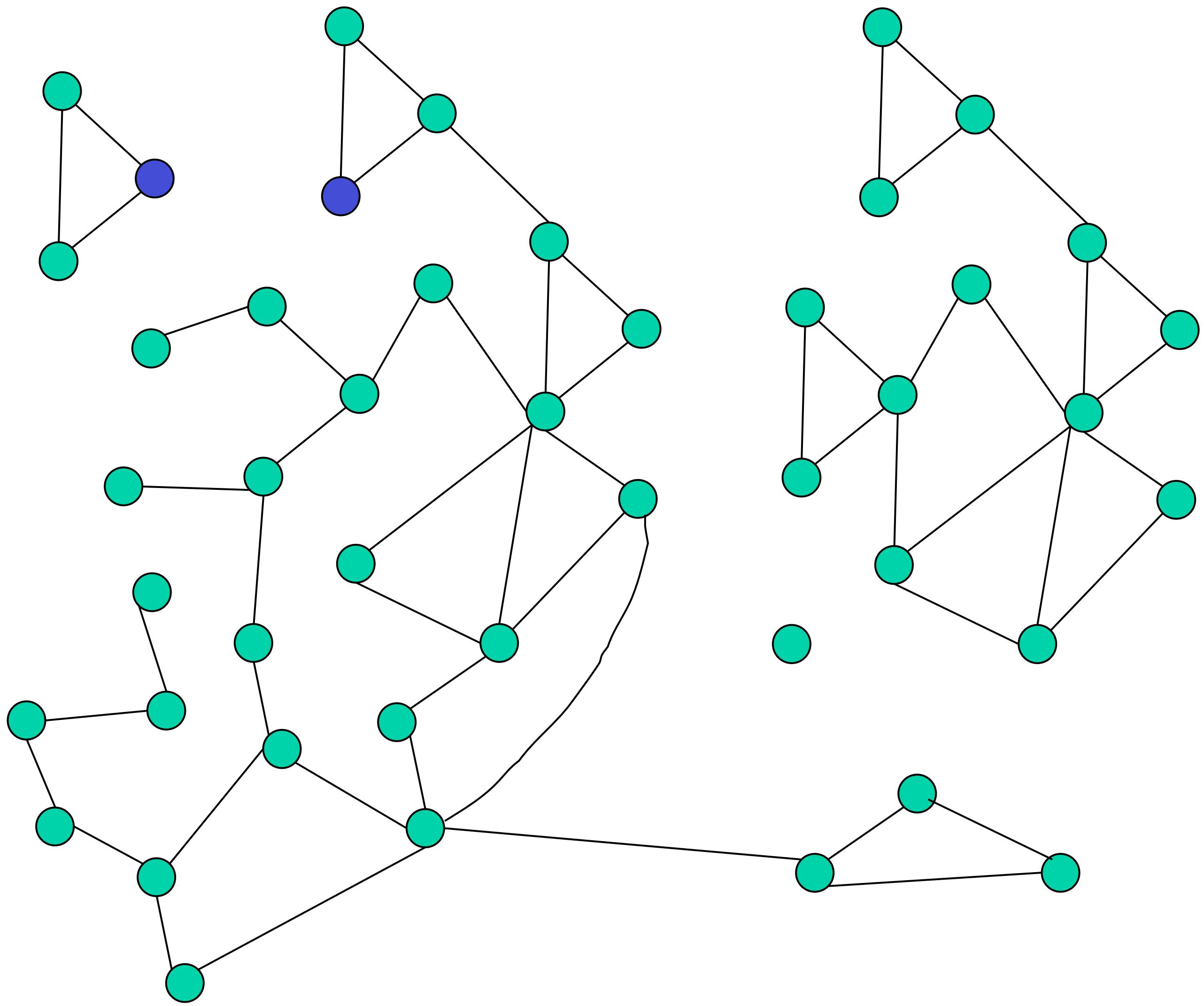
delete(v,w)



delete(v,w)



delete(v,w)



Observation

Without delete, a union-find data structure would be just sufficient

(Main) History of the Problem

Update	Query	Type	Reference
$O(m^{1/2})$	$O(1)$	det/w-c	[Frederickson SICOMP'85]
$O(n^{1/2})$	$O(1)$	det/w-c	[Eppstein, Galil, I. & Nissenzweig JACM'97]
$O(\log^3 n)$	$O(\frac{\log n}{\log \log n})$	rand/amort	[Henzinger, King JACM'99]
$O(\log^2 n)$	$O(\frac{\log n}{\log \log n})$	rand/amort	[Henzinger, Thorup Rand. Struct. & Algs. '97]
$O(\log^2 n)$	$O(\frac{\log n}{\log \log n})$	det/amort	[Holm, de Lichtenberg & Thorup JACM'01]
$O(\log n (\log \log n))$	$O(\frac{\log n}{\log \log \log n})$	rand/amort	[Thorup STOC' 00]
$O(\frac{\log^2 n}{\log \log n})$	$O(\frac{\log n}{\log \log n})$	det/amort	[Wulff-Nilsen SODA'13]
$O(\log^5 n)$	$O(\frac{\log n}{\log \log n})$	rand/w-c	[Kapron, King & Mountjoy SODA'13]

Will see

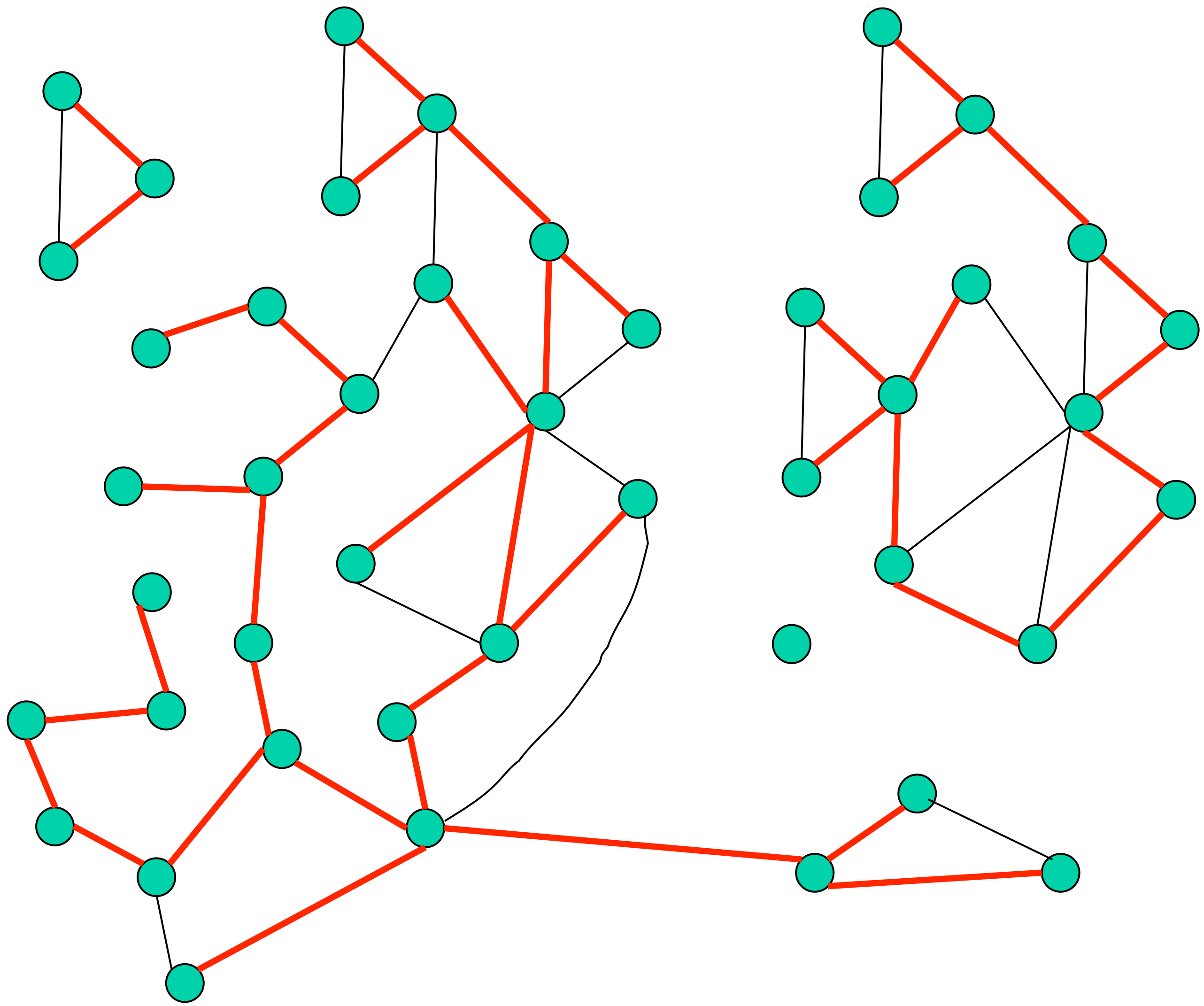
Update	Query	Type	Reference
$O(m^{1/2})$	$O(1)$	det/w-c	[Frederickson SICOMP'85]
$O(n^{1/2})$	$O(1)$	det/w-c	[Eppstein, Galil, I. & Nissenzweig JACM'97]
$O(\log^3 n)$	$O(\frac{\log n}{\log\log n})$	rand/amort	[Henzinger, King JACM'99]
$O(\log^2 n)$	$O(\frac{\log n}{\log\log n})$	rand/amort	[Henzinger, Thorup Rand. Struct. & Algs. '97]
$O(\log^2 n)$	**$O(\frac{\log n}{\log\log n})$**	**det/amort**	**[Holm, de Lichtenberg & Thorup JACM'01]**
$O(\log n (\log\log n))$	$O(\frac{\log n}{\log\log\log n})$	rand/amort	[Thorup STOC' 00]
$O(\frac{\log^2 n}{\log\log n})$	$O(\frac{\log n}{\log\log n})$	det/amort	[Wulff-Nilsen SODA'13]
$O(\log^5 n)$	$O(\frac{\log n}{\log\log n})$	rand/w-c	[Kapron, King & Mountjoy SODA'13]

Will actually see

Update	Query	Type	Reference
$O(m^{1/2})$	$O(1)$	det/w-c	[Frederickson SICOMP'85]
$O(n^{1/2})$	$O(1)$	det/w-c	[Eppstein, Galil, I. & Nissenzweig JACM'97]
$O(\log^3 n)$	$O(\frac{\log n}{\log\log n})$	rand/amort	[Henzinger, King JACM'99]
$O(\log^2 n)$	$O(\frac{\log n}{\log\log n})$	rand/amort	[Henzinger, Thorup Rand. Struct. & Algs. '97]
$O(\log^2 n)$	**$O(\log n)$**	**det/amort**	**[Holm, de Lichtenberg & Thorup JACM'01]**
$O(\log n (\log\log n))$	$O(\frac{\log n}{\log\log\log n})$	rand/amort	[Thorup STOC' 00]
$O(\frac{\log^2 n}{\log\log n})$	$O(\frac{\log n}{\log\log n})$	det/amort	[Wulff-Nilsen SODA'13]
$O(\log^5 n)$	$O(\frac{\log n}{\log\log n})$	rand/w-c	[Kapron, King & Mountjoy SODA'13]

Reduce the problem to a problem on trees (i.e., maintain a certificate for the property)

Maintain a spanning forest of the graph



We will have to link trees, cut trees, and determine whether two vertices are in the same tree in this forest

Operations we need to do on the forest

link(v,w) : Join two trees in the forest by inserting edge (v,w) (assume v and w are in different trees)

cut(v,w) : Split a tree by deleting edge (v,w) (assume v and w are adjacent in a tree)

findtree(v) : Return the tree containing vertex v in the forest

Operations we need to do on the forest

link(v,w) : Join two trees in the forest by inserting edge (v,w) (assume v and w are in different trees)

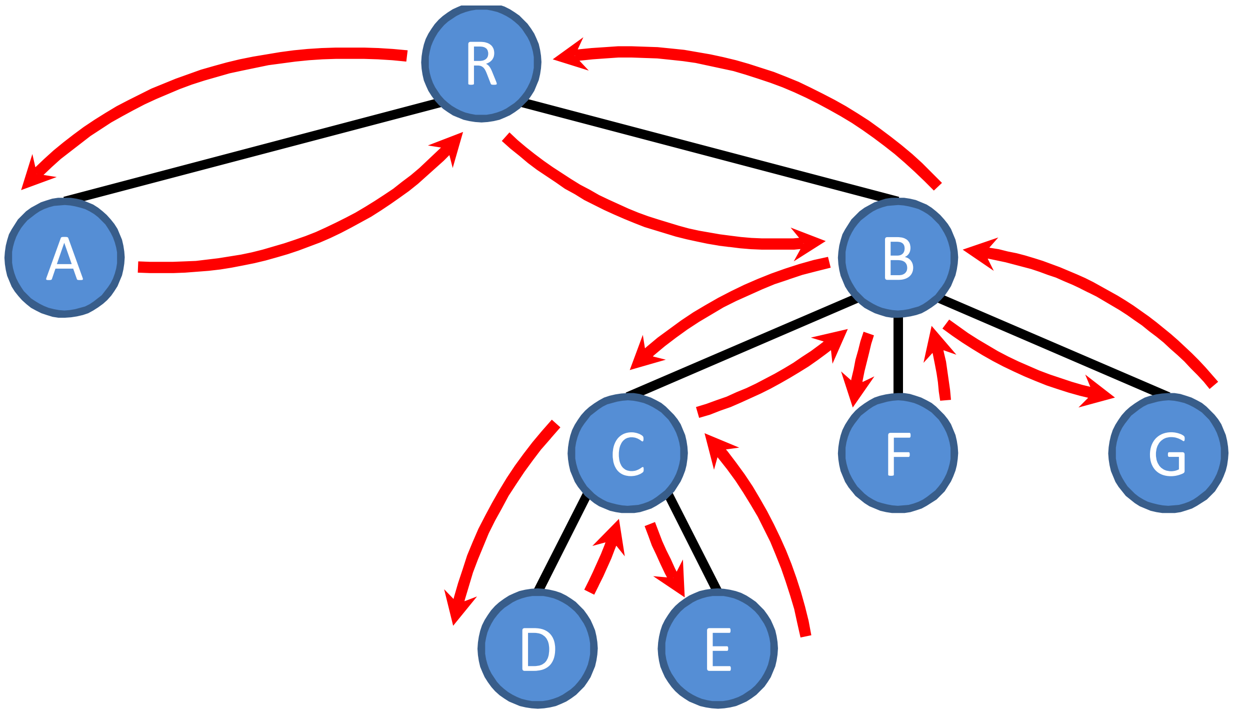
cut(v,w) : Split a tree by deleting edge (v,w) (assume v and w are adjacent in a tree)

findtree(v) : Return the tree containing vertex v in the forest

Can do this in O(log n) per operation with several data structures, e.g., ET-trees (Euler Tour trees)

We refer to those as dynamic tree data structures

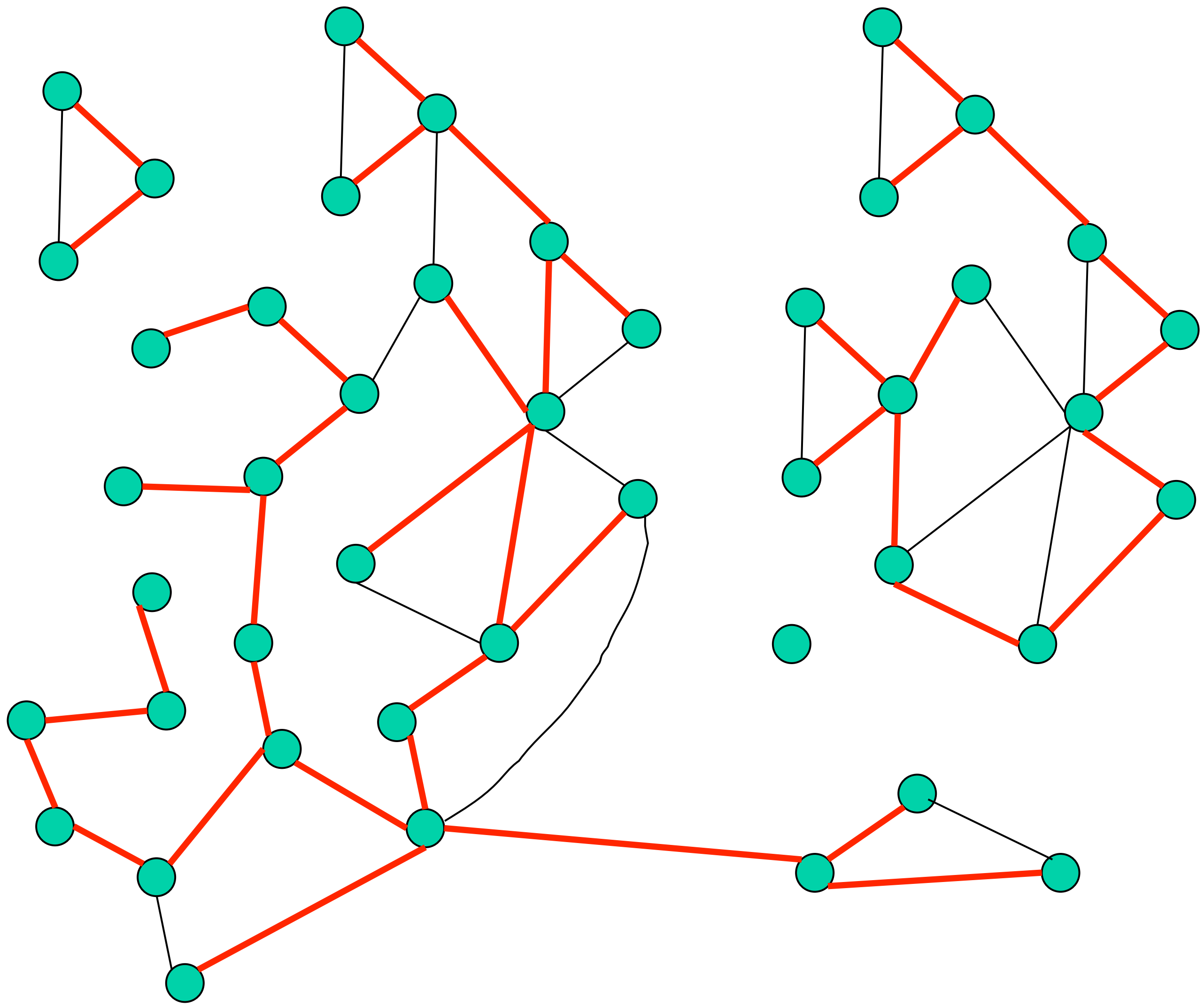
ET-trees

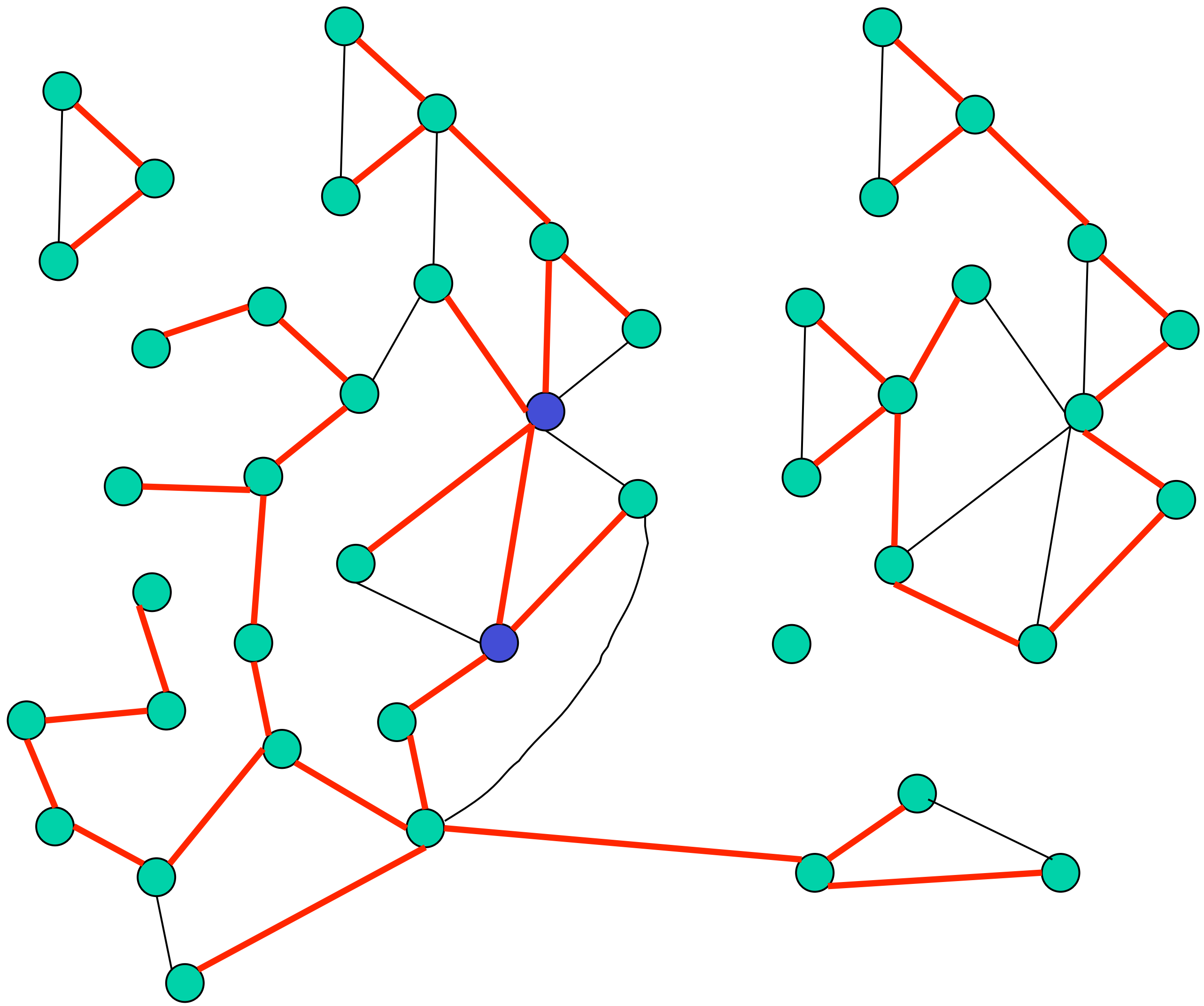


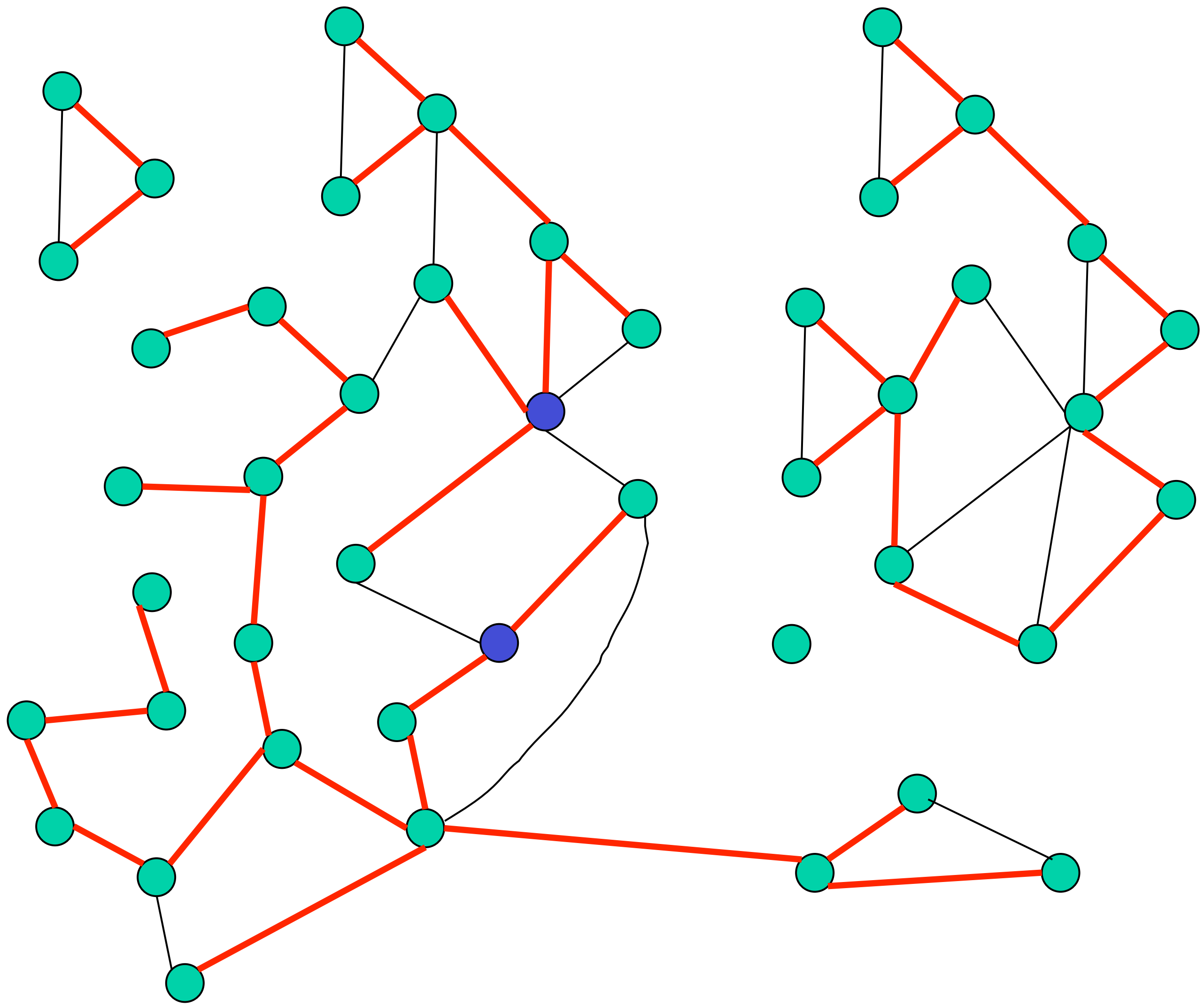
ET-tree is a balanced binary tree over the Euler tour of a tree.

Can perform link, cut and findtree in O(log n)

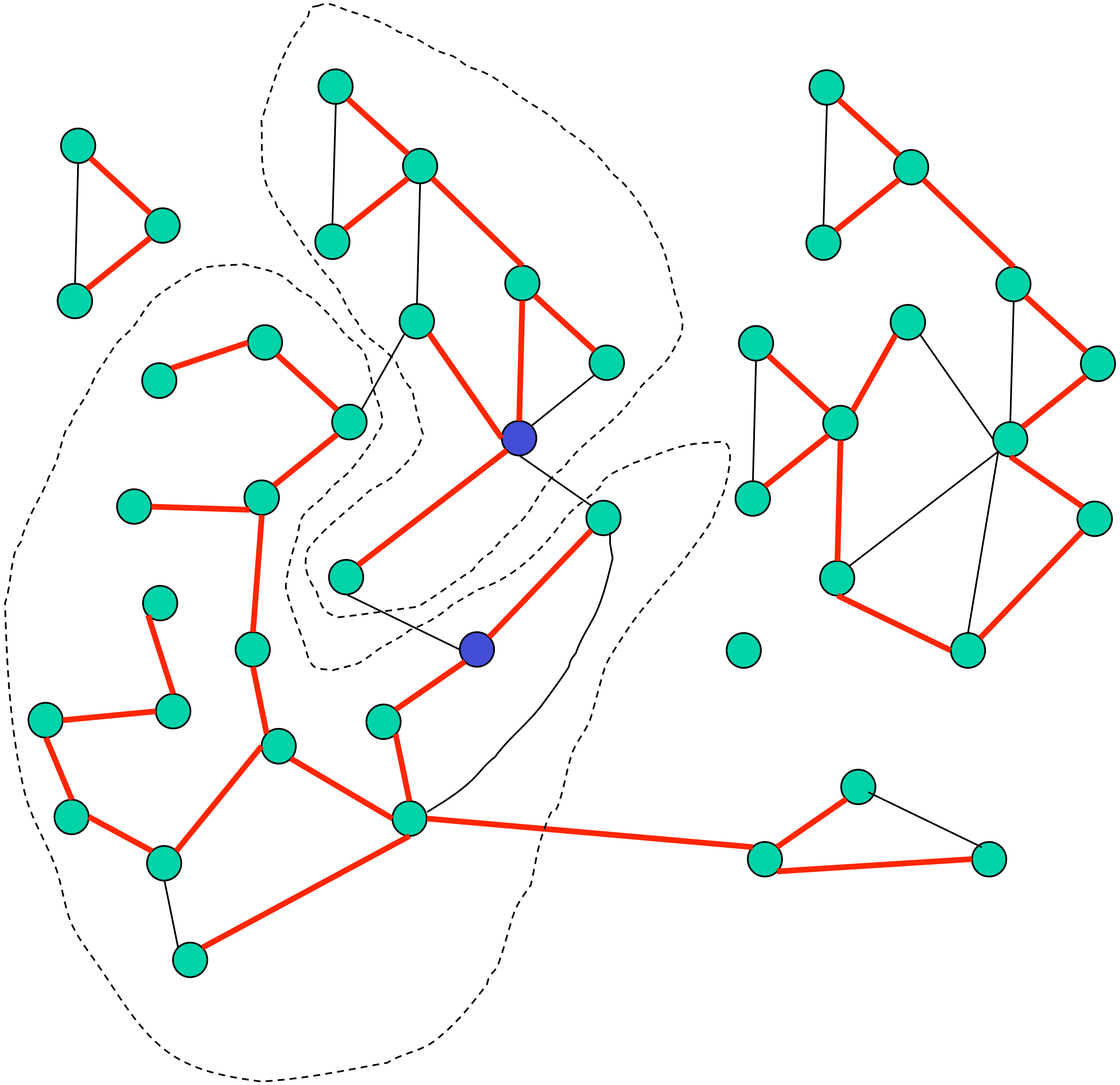
But dynamic tree data structures are not enough: we still have a problem with deleting a tree edge







How do we find out whether there is a "replacement" edge for the forest or it really got disconnected ?



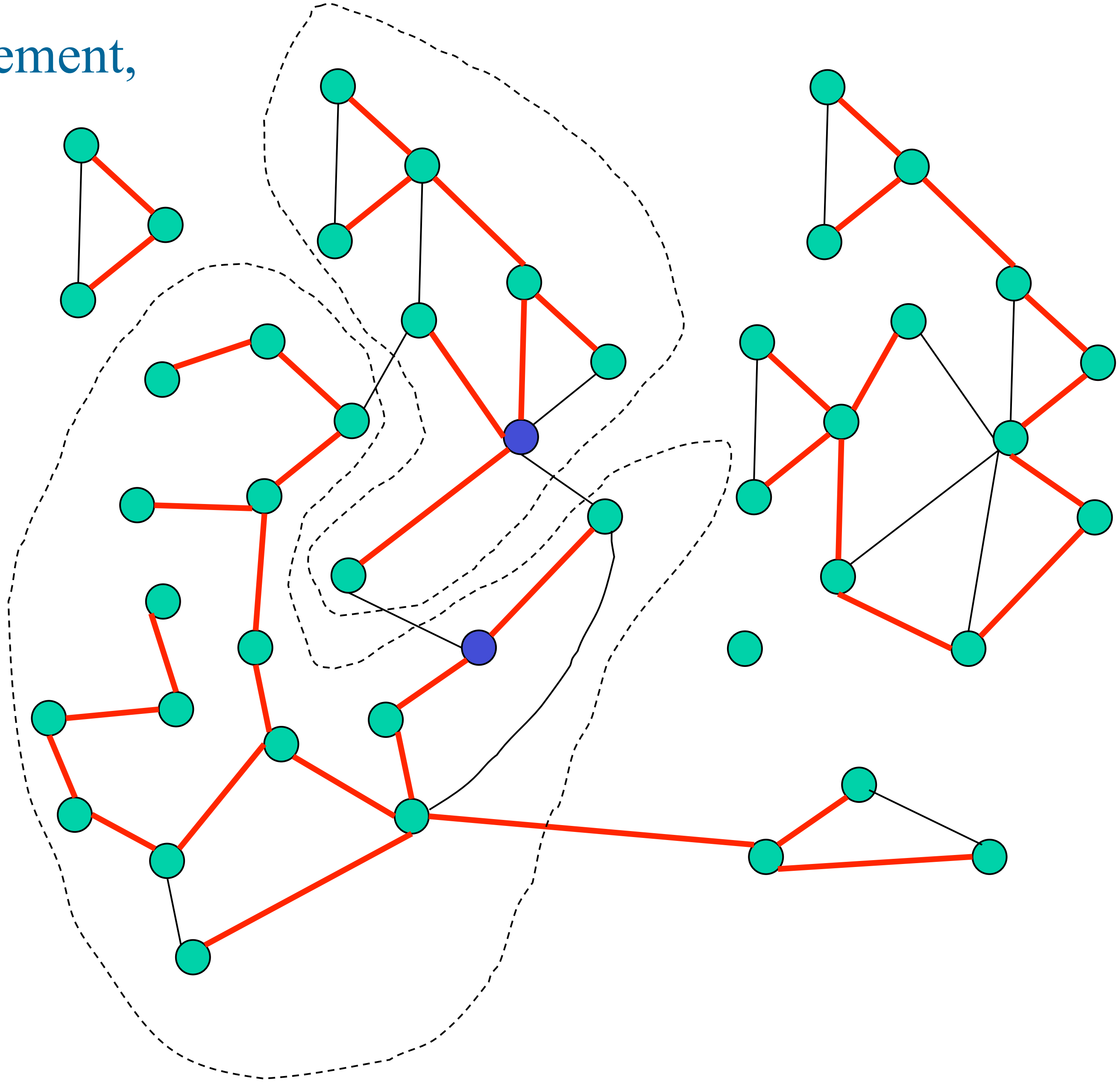
Summarising so far

	Tree Edge	**Non-Tree Edge**
Insert	Link	Easy
Delete	Cut, **Replacement?**	Easy

To find a replacement, need to traverse one of the trees, which can be quite expensive.

Randomization [Henzinger, King]: sample non-tree edges in smaller tree

If sampling fails, push "sparse cut" to upper level

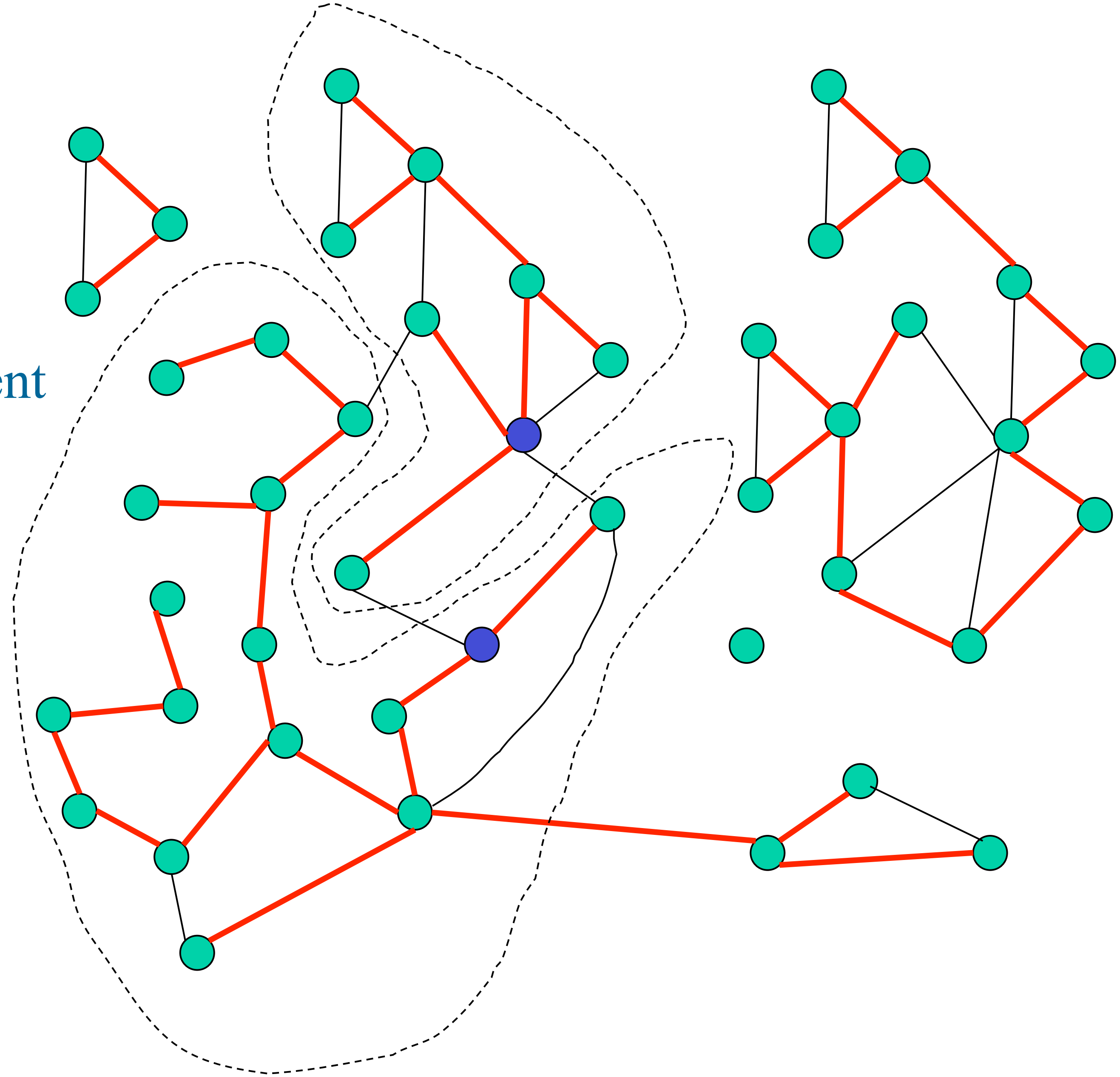


Can we do this deterministically?

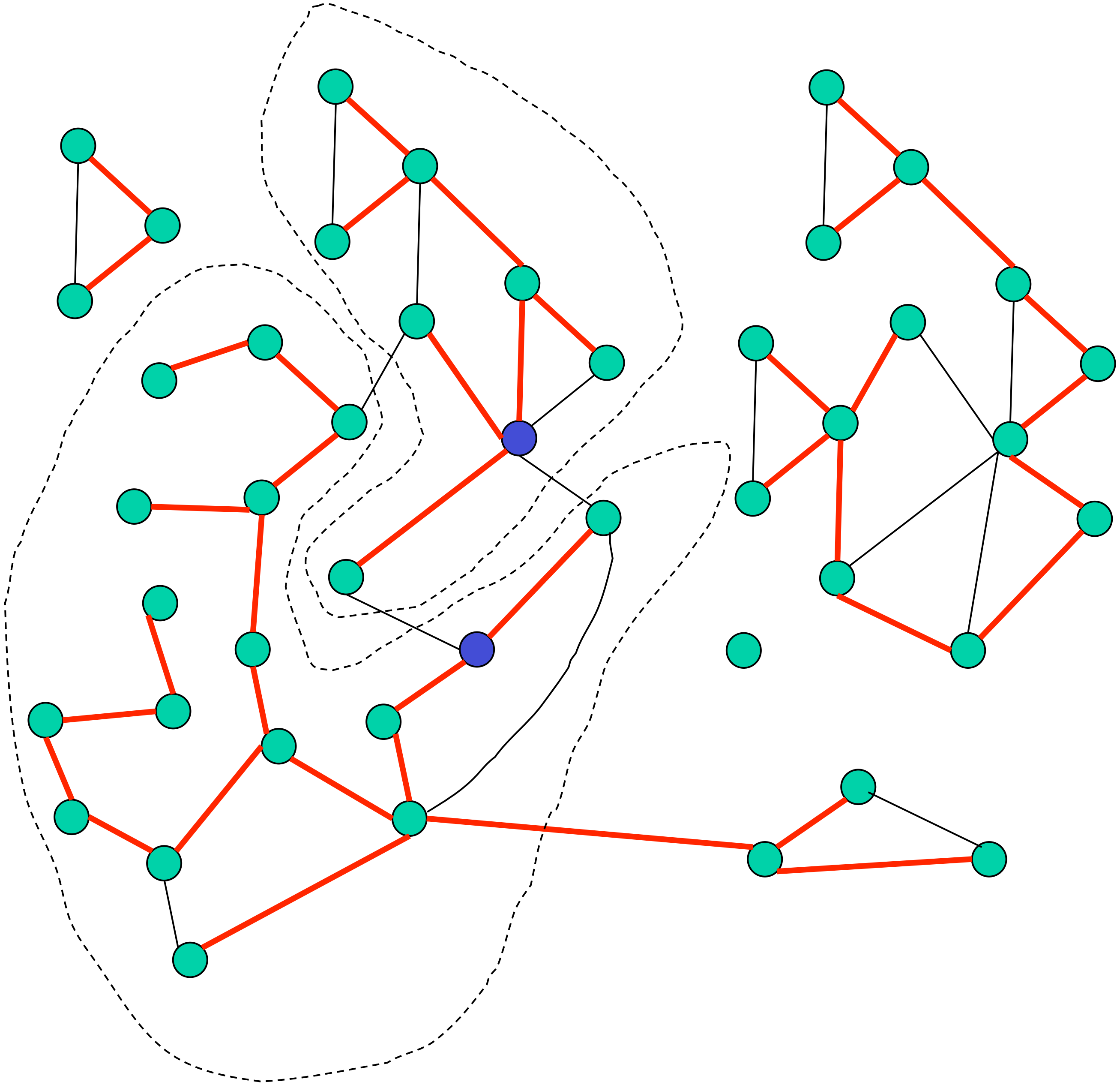
Look in the smaller tree:

- tree edge
- no replacement
- replacement

Wish to gain something (in amortized sense) by accumulating information as we do that

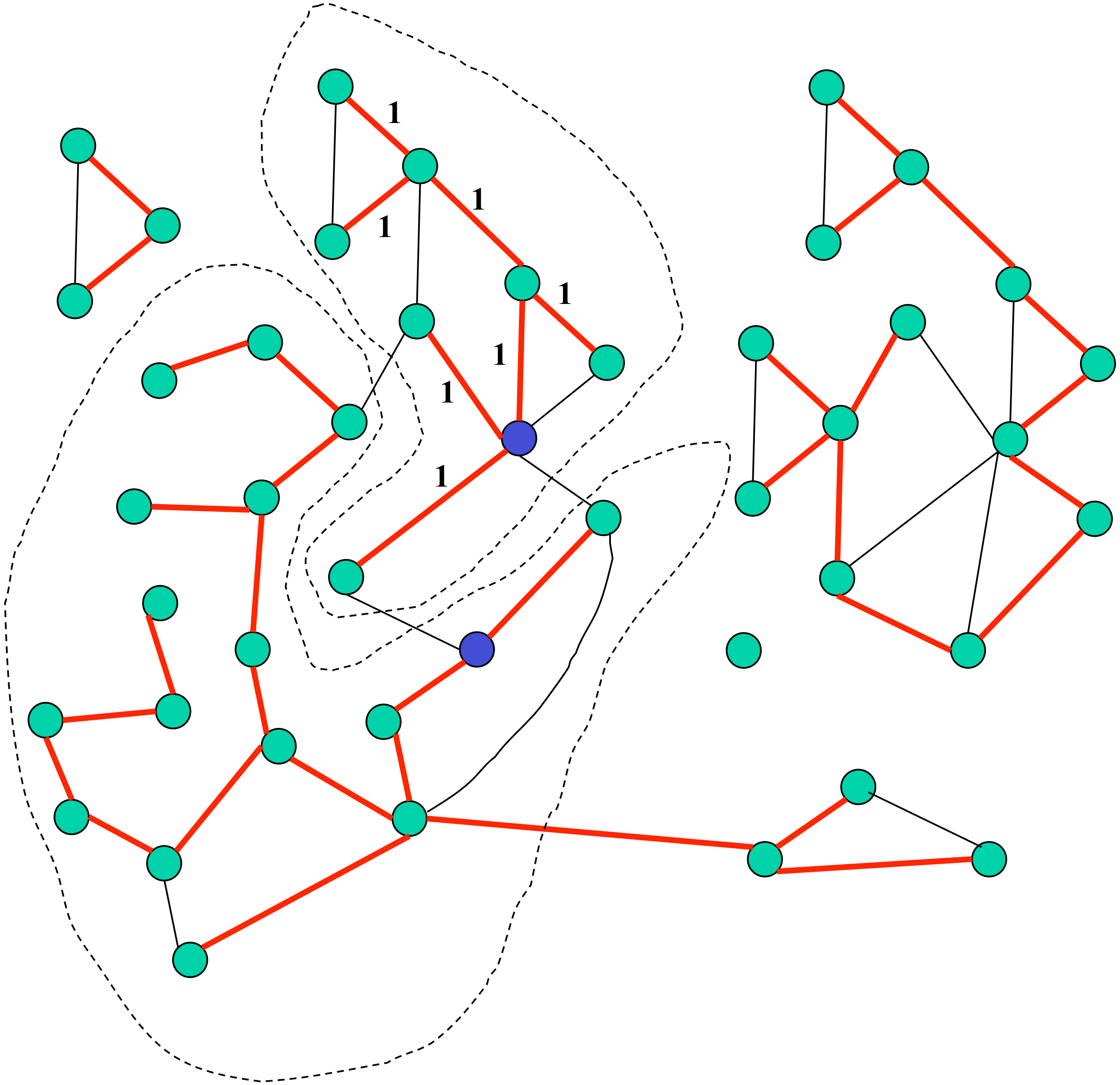


Each edge has
a level



Each edge has a level

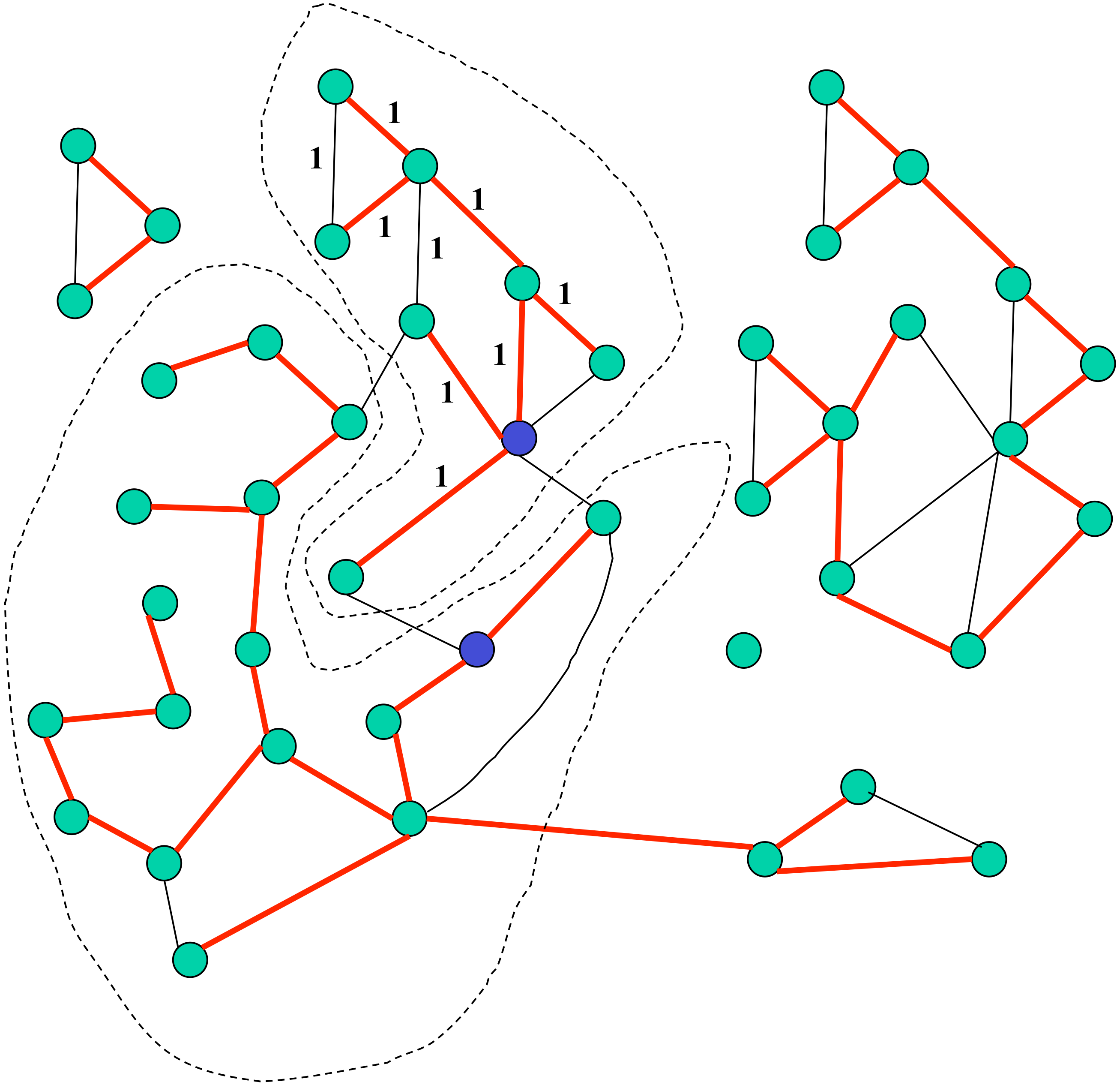
Increase the level of the edges in the smaller tree…



Each edge has
a level

Increase the
level of the
edges in the
smaller tree…

… and of any
edge discovered
not to be a
“replacement”

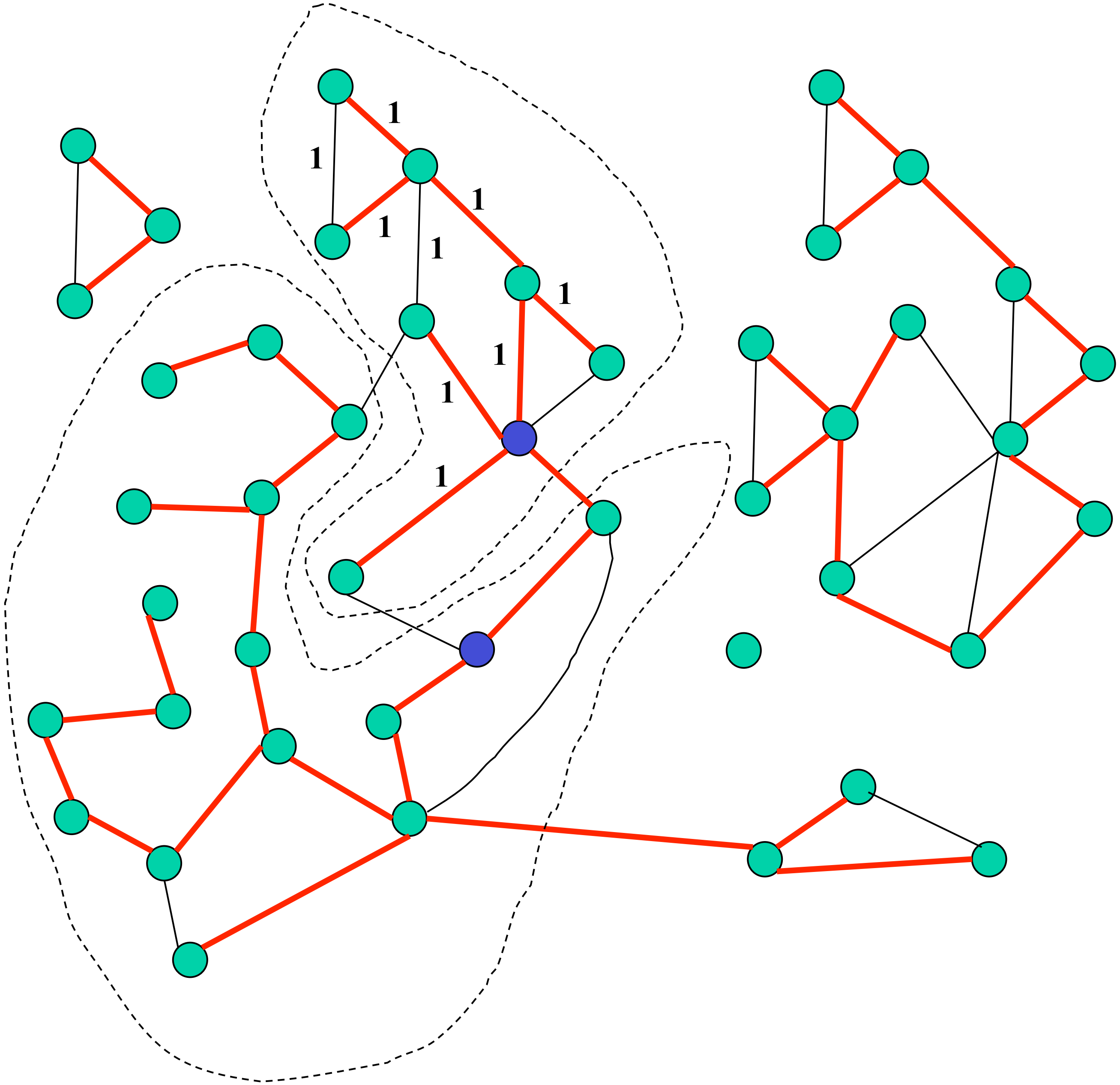


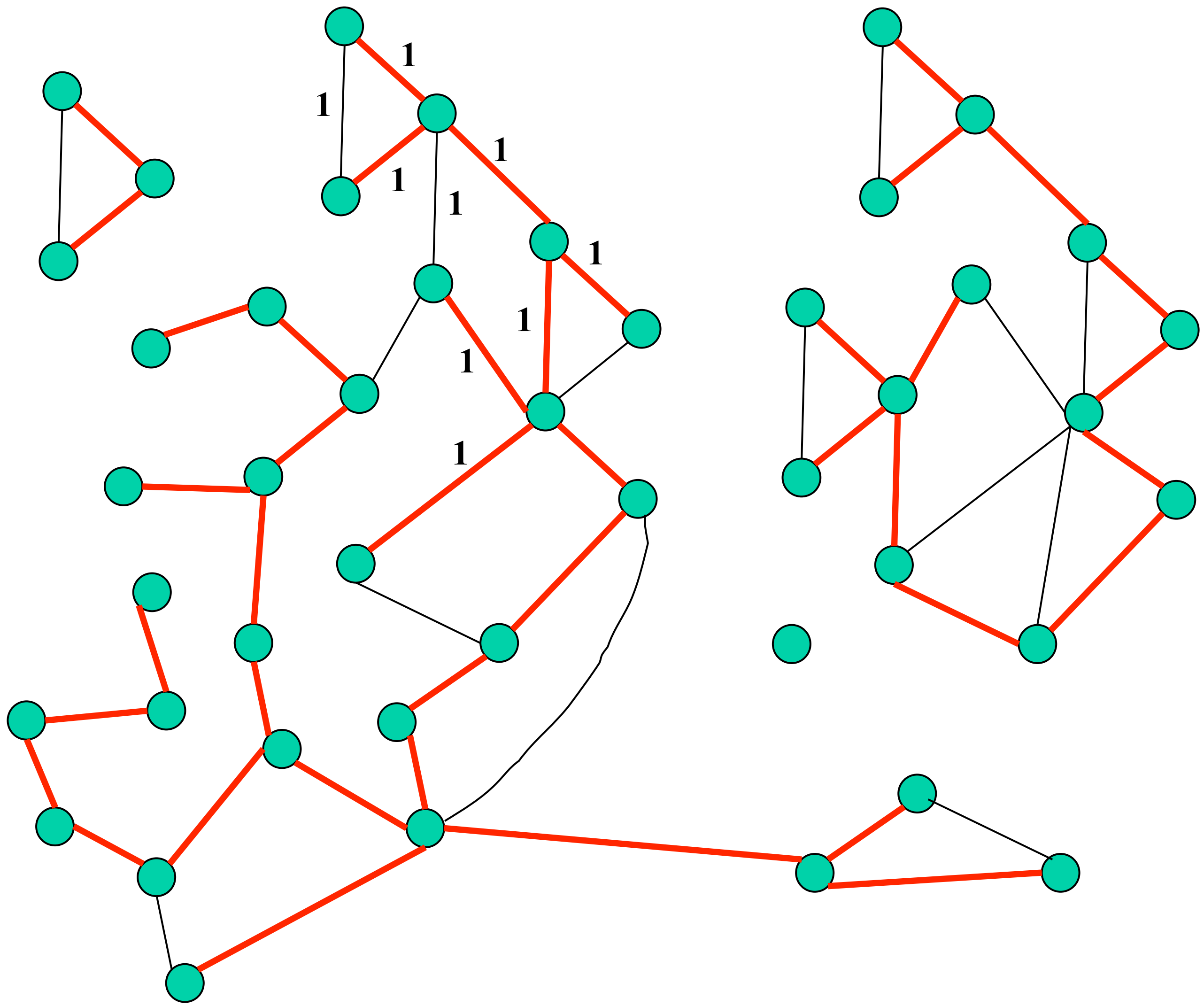
Each edge has a level

Increase the level of the edges in the smaller tree…

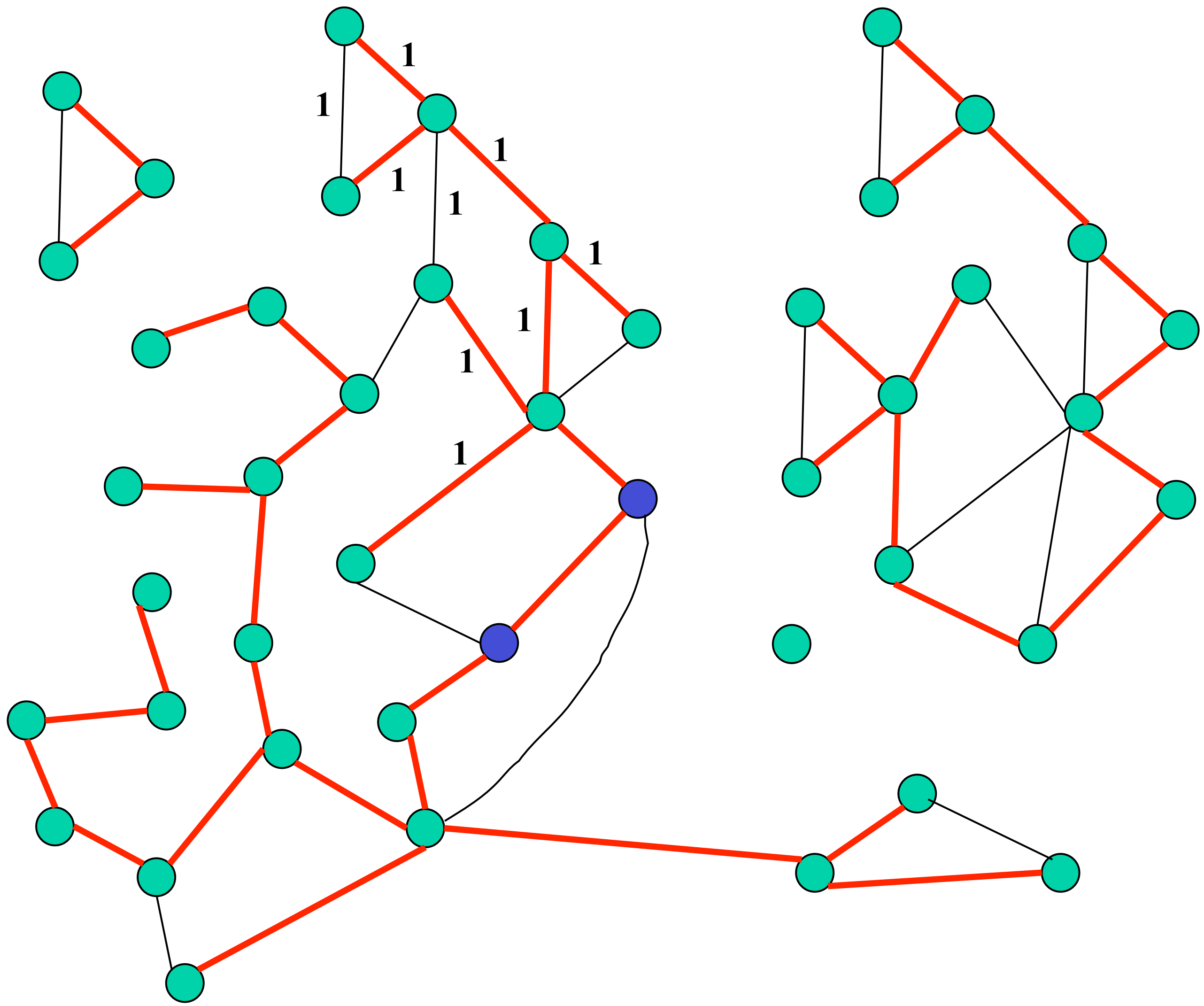
… and of any edge discovered not to be a "replacement"

until you find a "replacement"



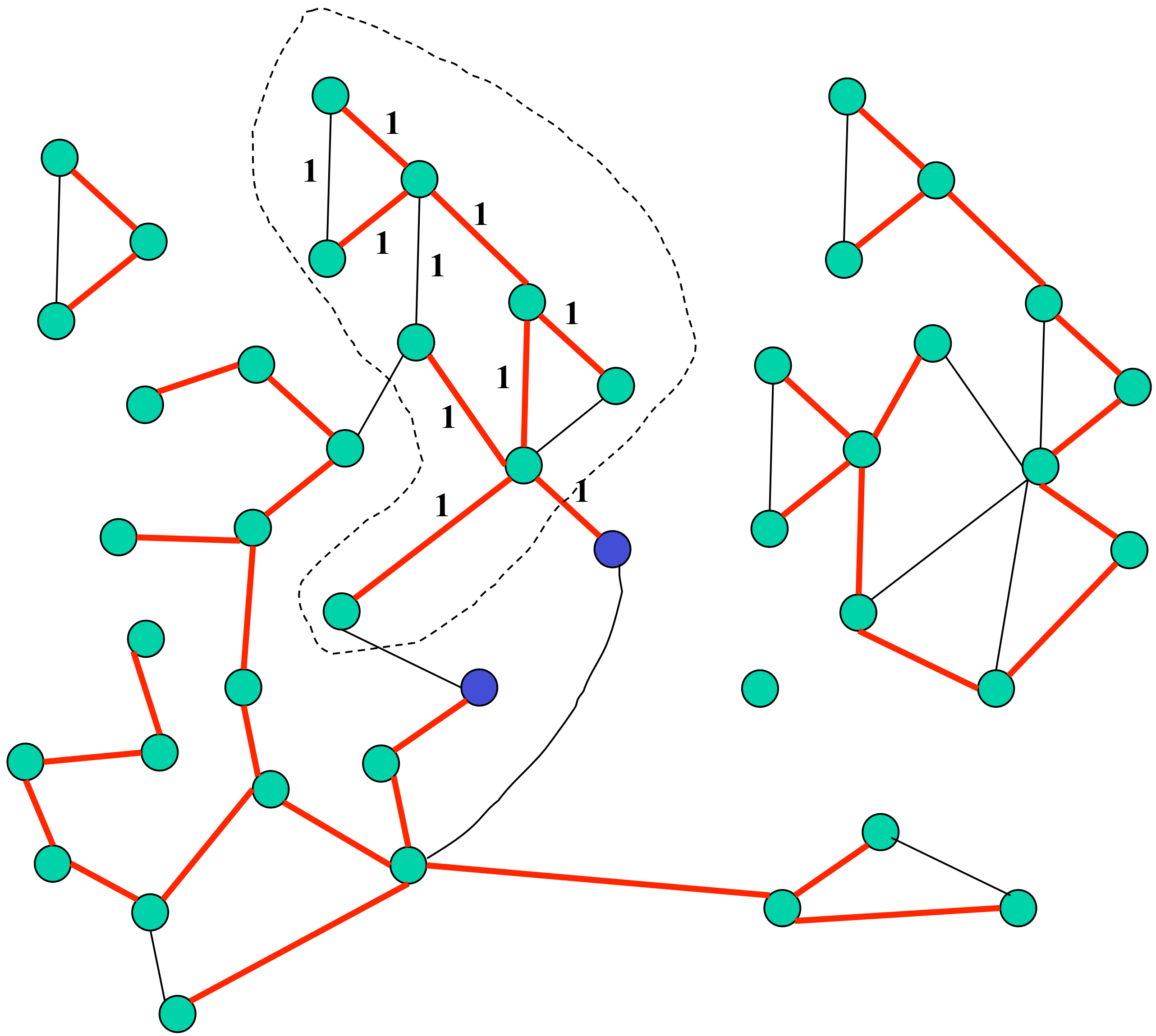

1
1
1
1
1
1
1
1
1

Intuition:
Next time you
have to look
again for a
replacement…



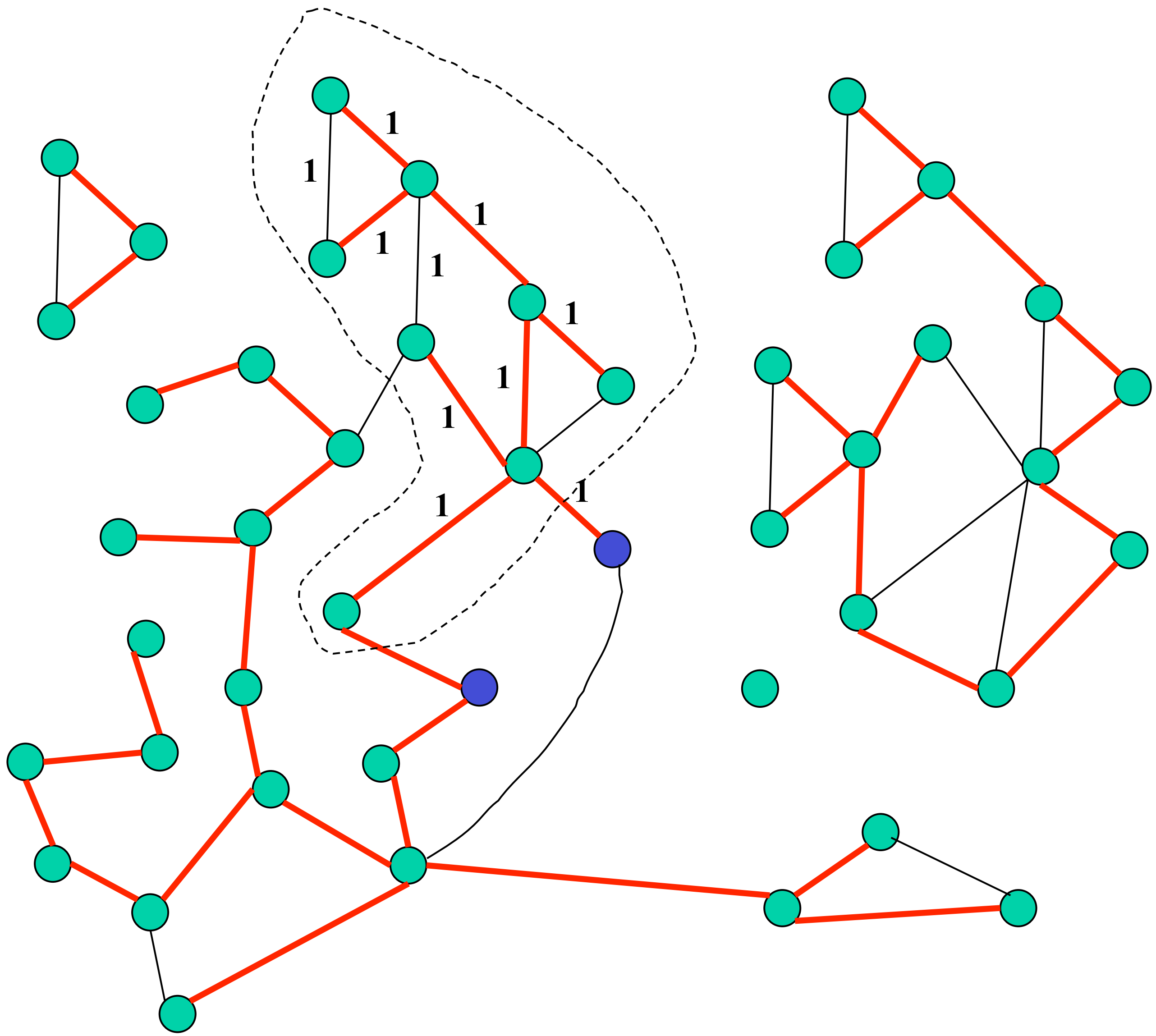
Intuition:
Next time you have to look again for a replacement…

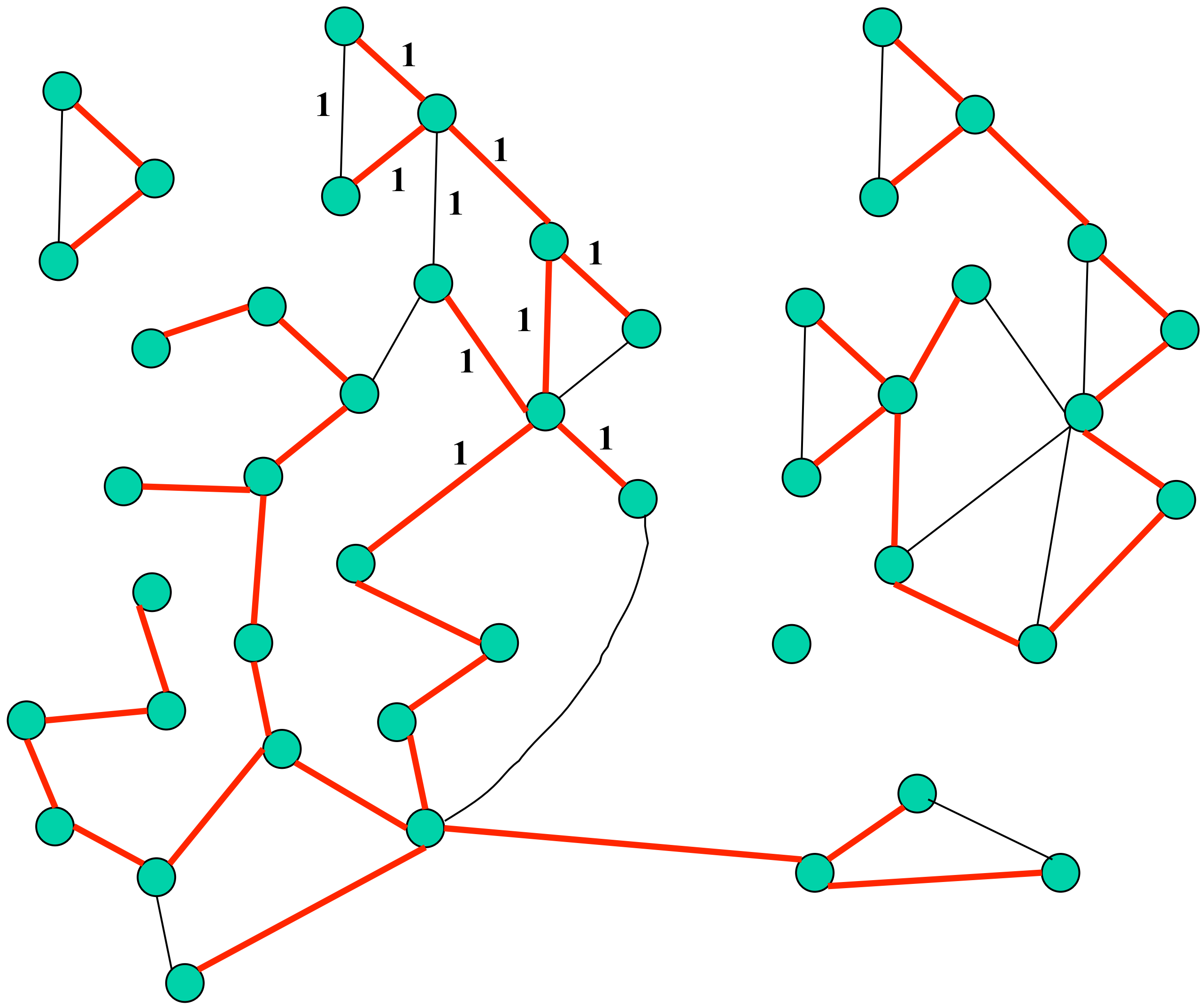
… no need to look at non-tree edges with label 1!



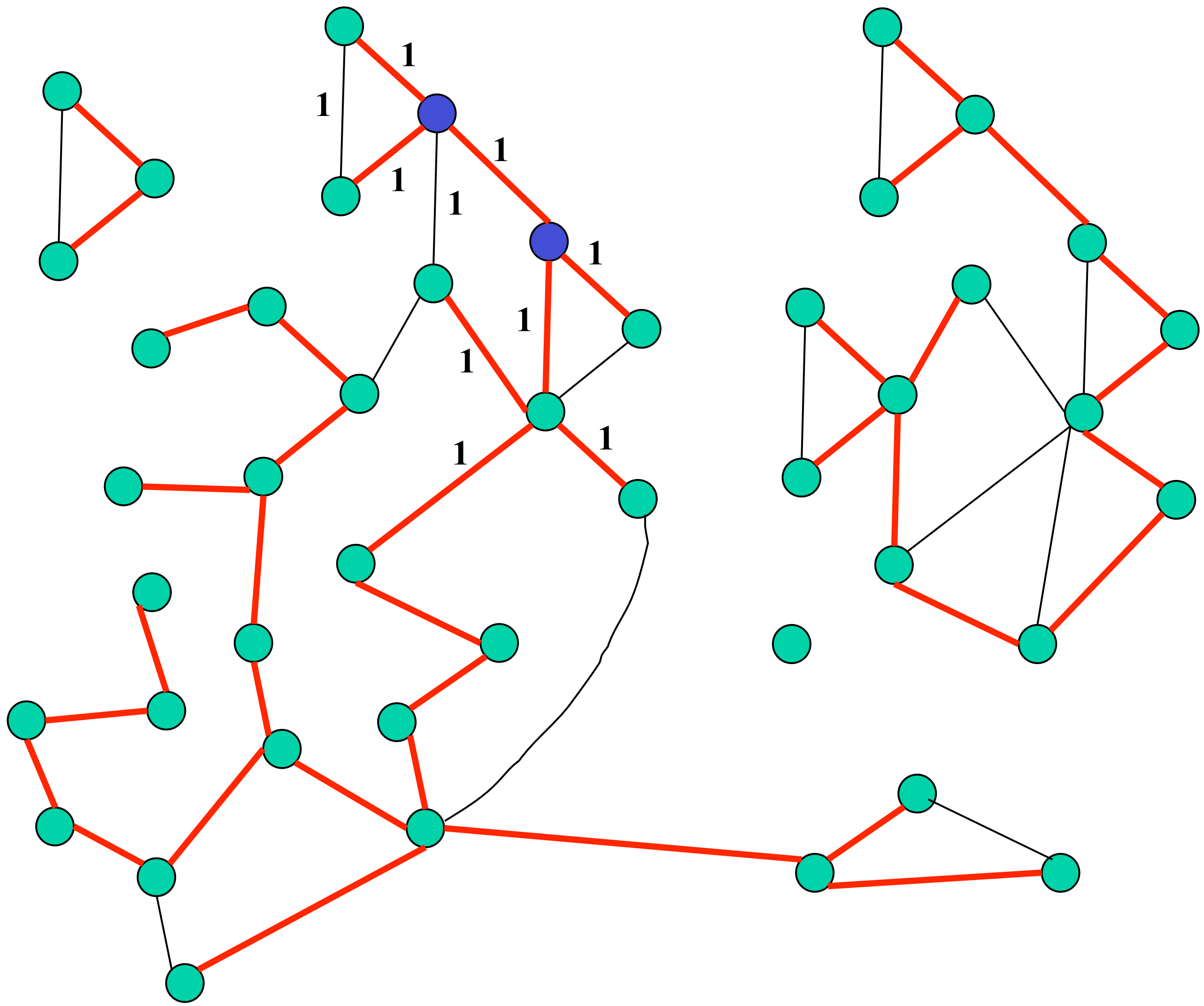
Intuition:
Next time you have to look again for a replacement…

… no need to look at non-tree edges with label 1!

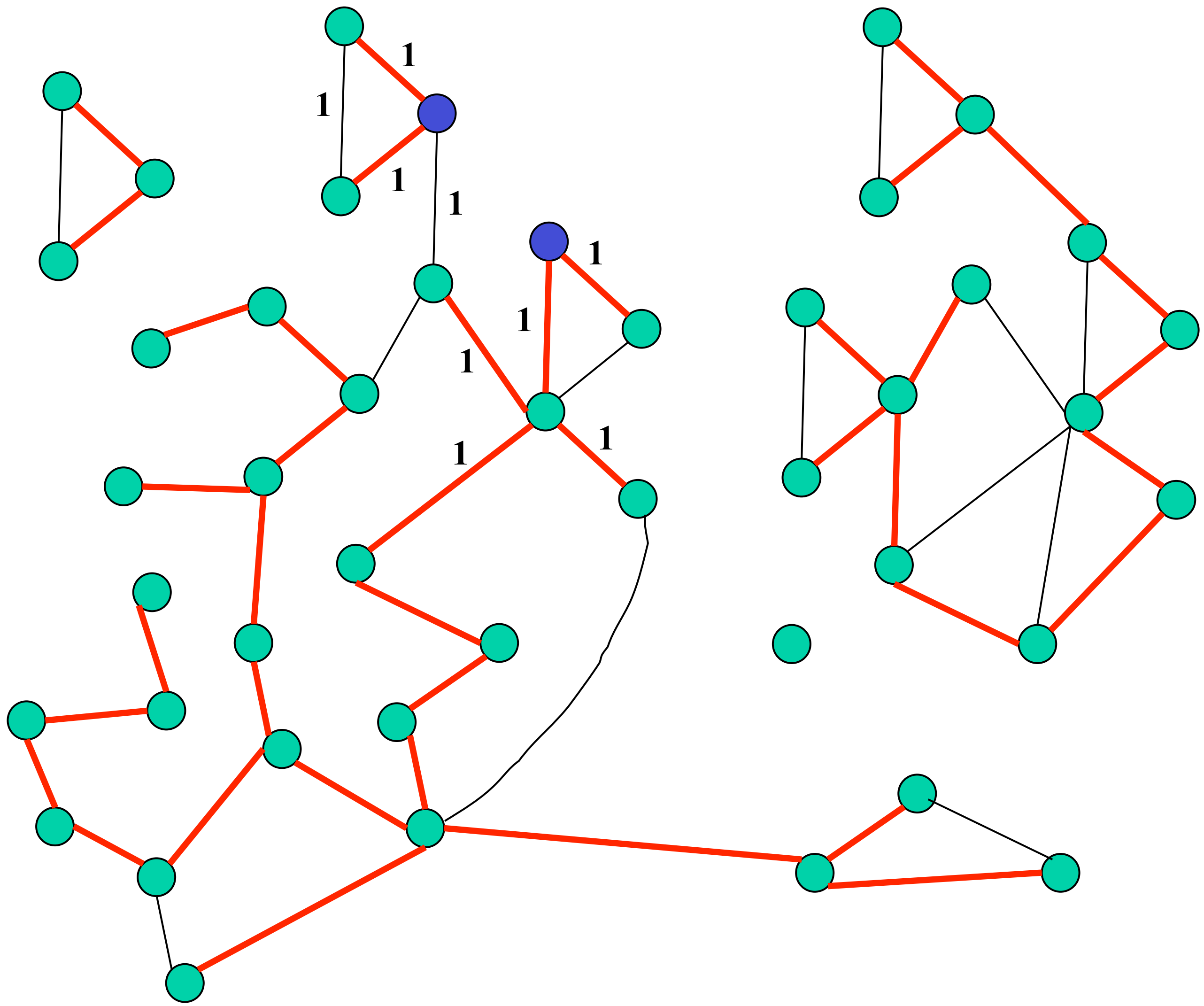



1
1
1
1
1
1
1
1
1
1

Keep on doing that upon edge deletions

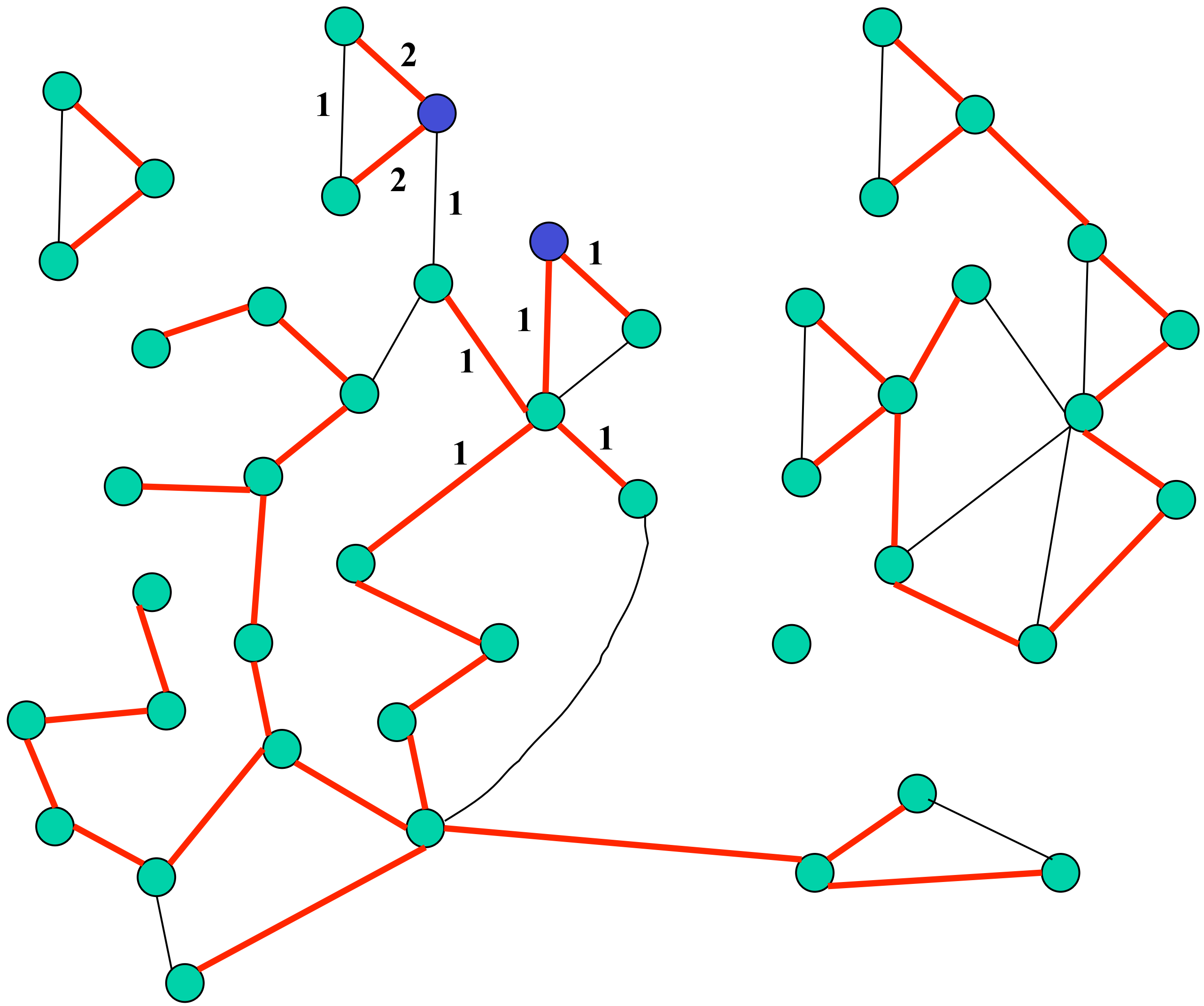


Keep on doing that upon edge deletions



Keep on doing that upon edge deletions

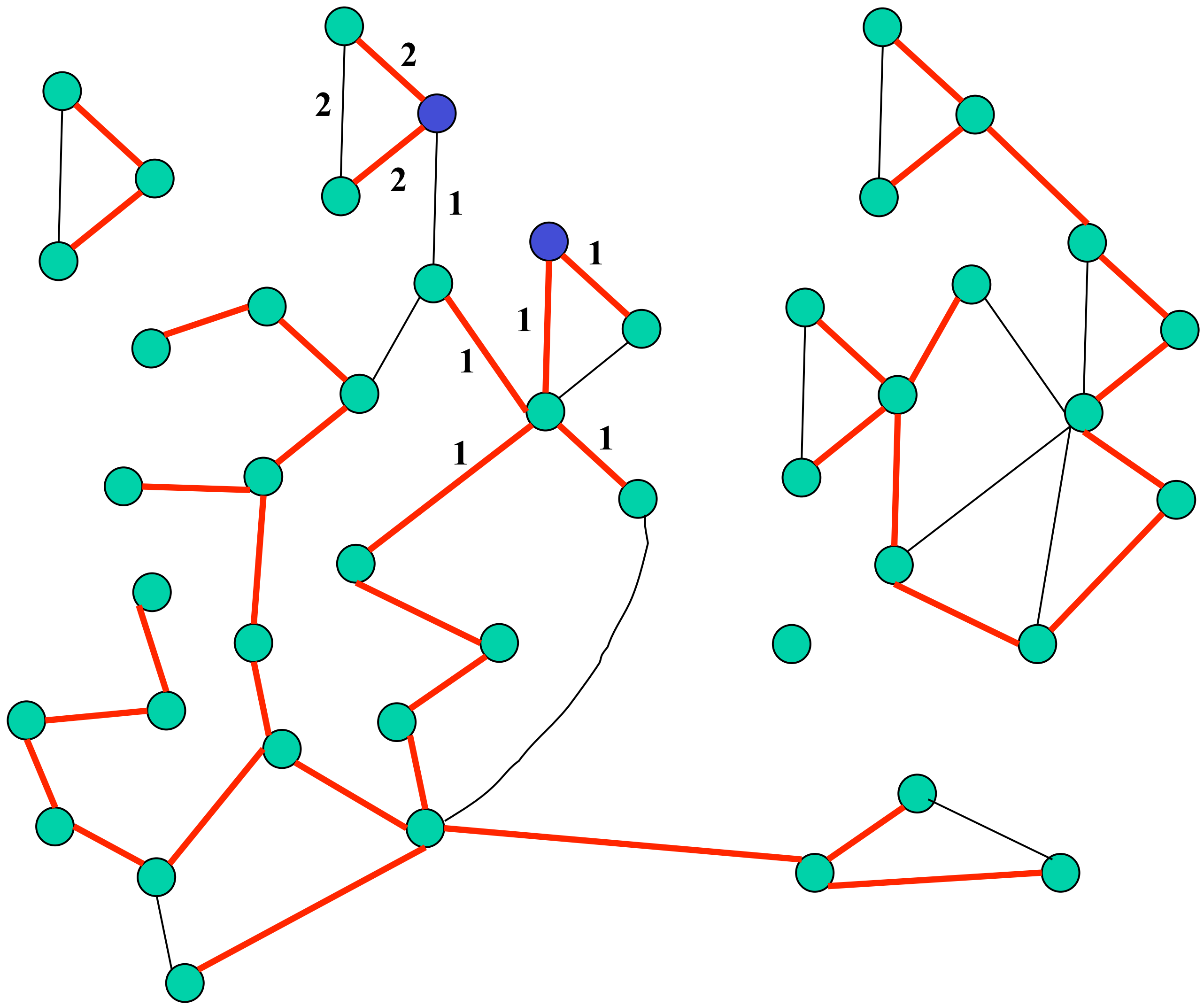
Again, increase the level of the edges in the smaller tree…



Keep on doing that upon edge deletions

Again, increase the level of the edges in the smaller tree…

… and of any edge discovered not to be a "replacement"

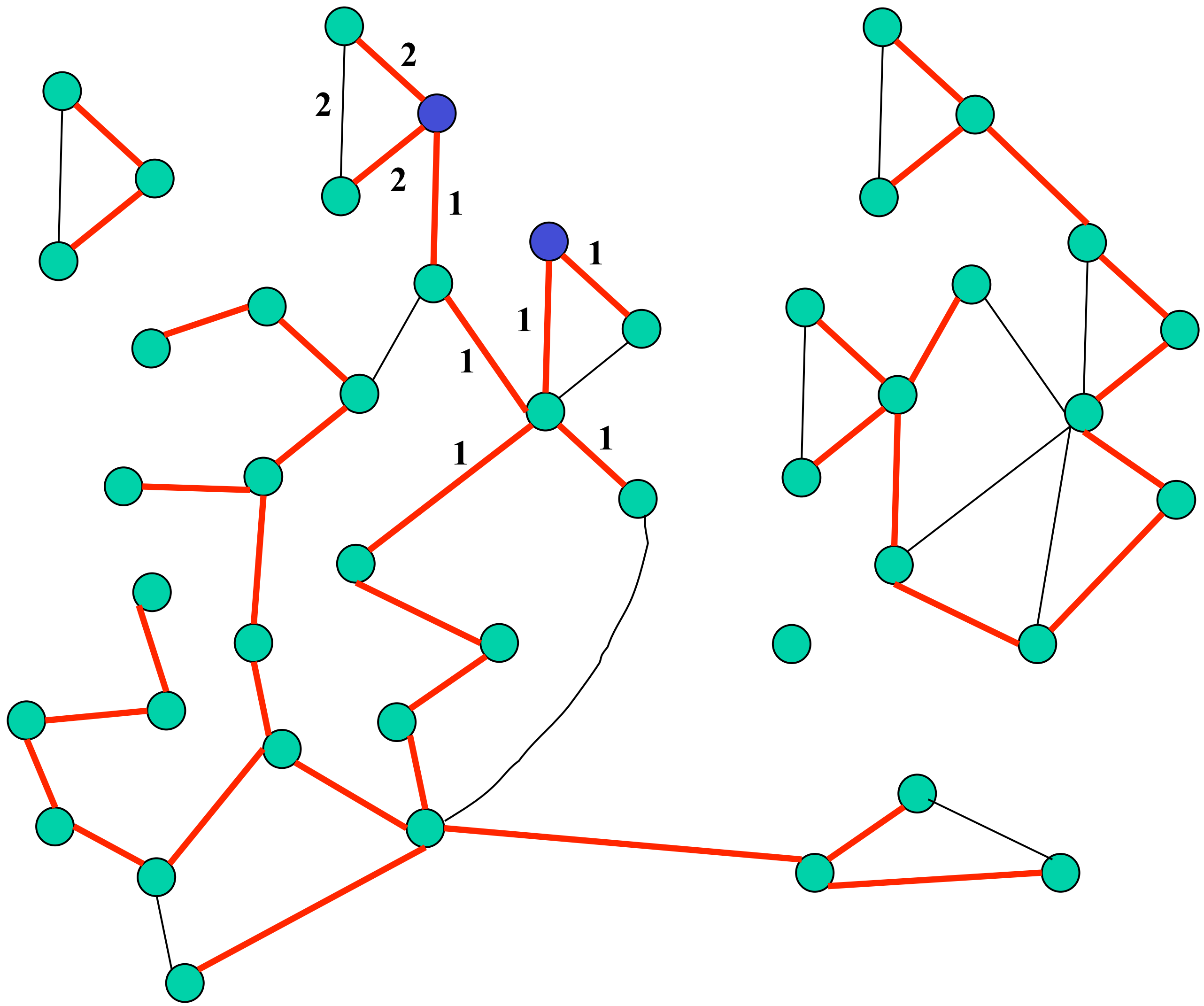


Keep on doing that upon edge deletions

Again, increase the level of the edges in the smaller tree…

… and of any edge discovered not to be a “replacement”

until you find a “replacement”

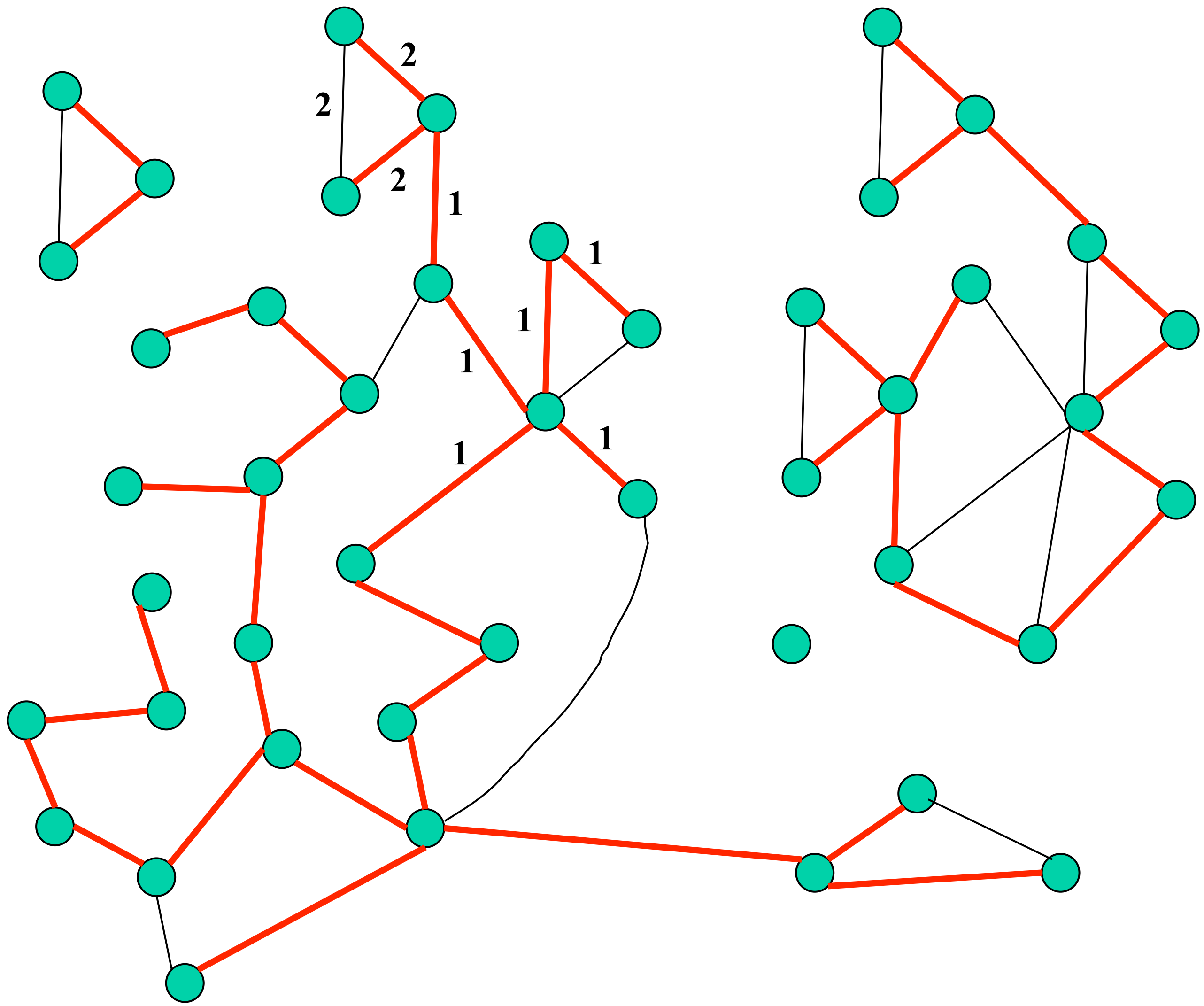


Keep on doing that upon edge deletions

Again, increase the level of the edges in the smaller tree…

… and of any edge discovered not to be a "replacement"

until you find a "replacement"



Terminology

G is the dynamic graph. **F** is a spanning forest of G.

An edge is either a tree edge or a non-tree edge.

Each edge has a level ℓ.

$\mathbf{G_\ell}$ is subgraph of G induced by edges of level $\geq \ell$.

$$G_{max} \subseteq \ldots \subseteq G_\ell \subseteq \ldots \subseteq G_2 \subseteq G_1 \subseteq G_0 = G$$

$\mathbf{F_\ell}$ is subforest of F induced by edges of level $\geq \ell$.

$$F_{max} \subseteq \ldots \subseteq F_\ell \subseteq \ldots \subseteq F_2 \subseteq F_1 \subseteq F_0 = F$$

Invariants

Recall: F_ℓ subforest of F induced by edges of level $\geq \ell$.

Will keep the following two invariants:

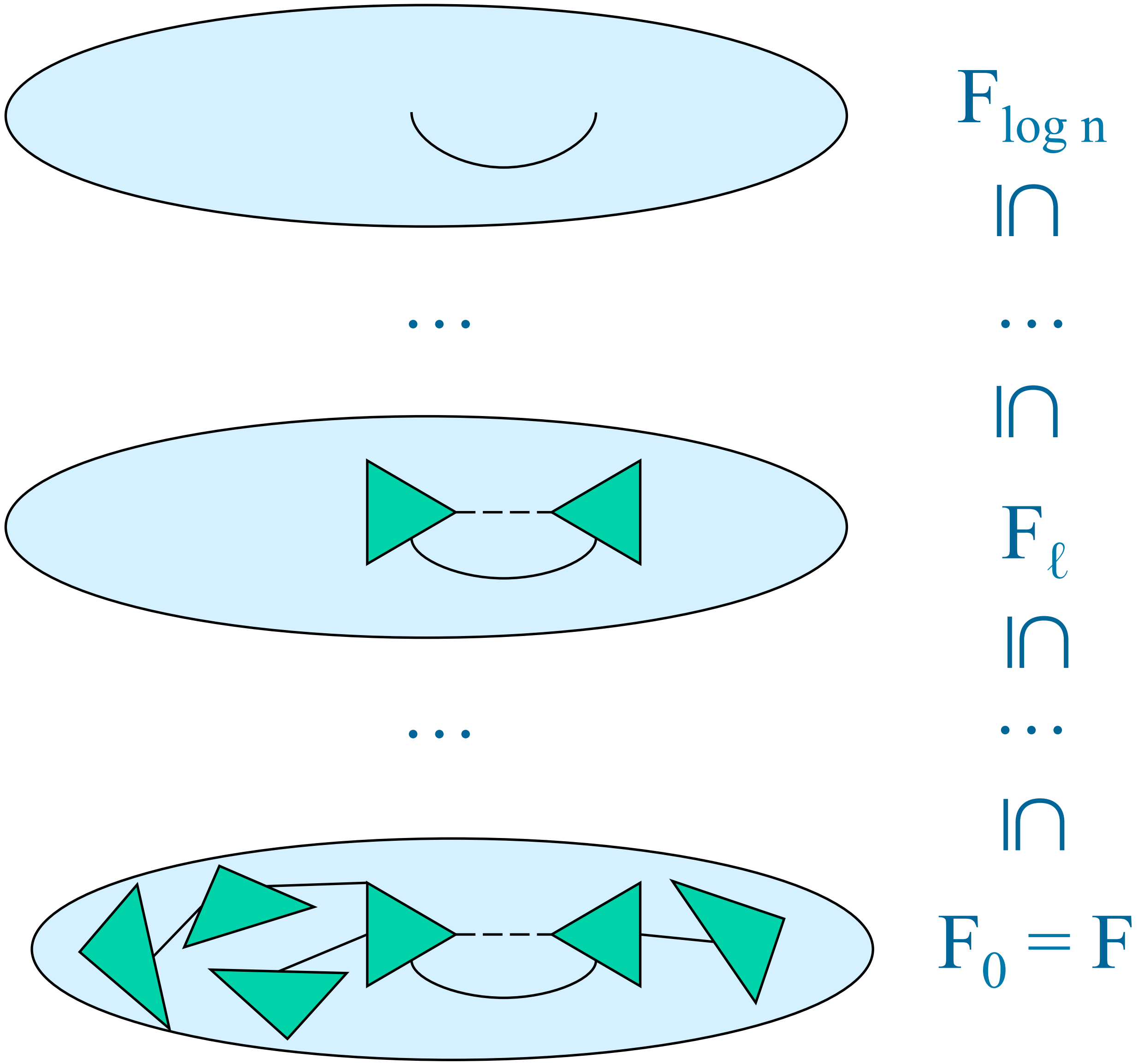
(Invariant 1) Each tree in F_ℓ (i.e., connected component in G_ℓ) has at most $n/2^\ell$ vertices

➔ At most (log n) levels

(Invariant 2) The forest F is a maximum spanning forest with respect to the levels of the edges

➔ If (v, w) is a non-tree edge of level ℓ, then v and w are connected (i.e., in the same tree) in F_ℓ

➔ If a tree edge at level ℓ is deleted, then a replacement edge (if there is one) must be of level $\leq \ell$


$F_{\log n}$
$\subseteq$
$\cdots$
$\subseteq$
F_ℓ
$\subseteq$
$\cdots$
$\subseteq$
$F_0 = F$

Observations

Initially all edges at level 0 (both invariants satisfied)

Amortization argument: Levels of an edge can only increase, so we can have ≤ log n increases per edge

Intuition: When level of non-tree edge increased, it is because we discovered that its endpoints are close enough in F to fit in a smaller tree (higher level)

Increasing the level of a tree edge is always safe for Invariant 2 (F is a maximum spanning forest) but it may violate Invariant 1

Invariant 1

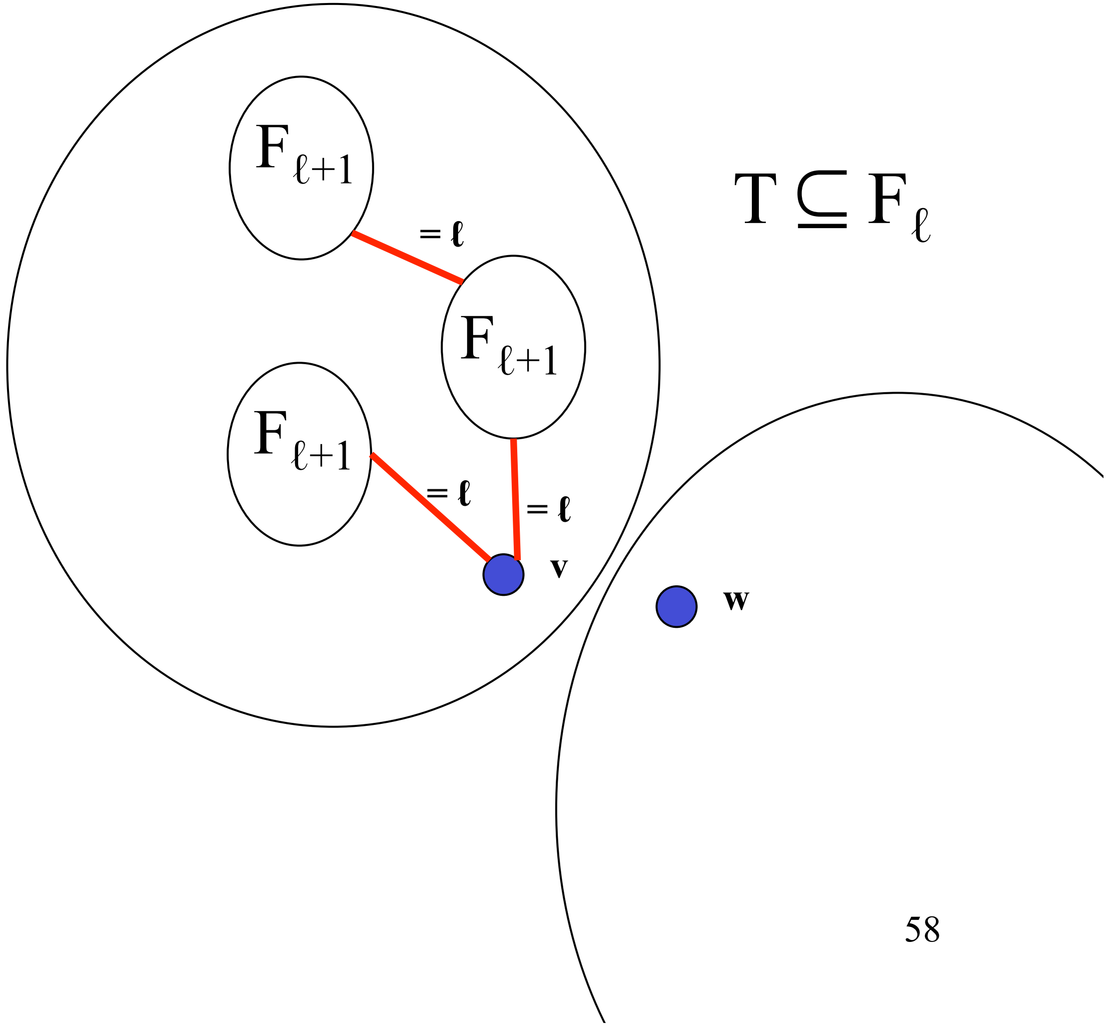
$|T| \leq n/2^{\ell}$

$|T_v| \leq |T_w|$

➔ $|T_v| \leq n/2^{\ell+1}$

We can afford to push all edges of T_v from level ℓ up to level $\ell + 1$ (while still preserving Invariant 1).

The replacement edge stays at level ℓ



$T \subseteq F_{\ell}$

Implementation

For each level ℓ:

- Maintain F_ℓ in a dynamic tree data structure.

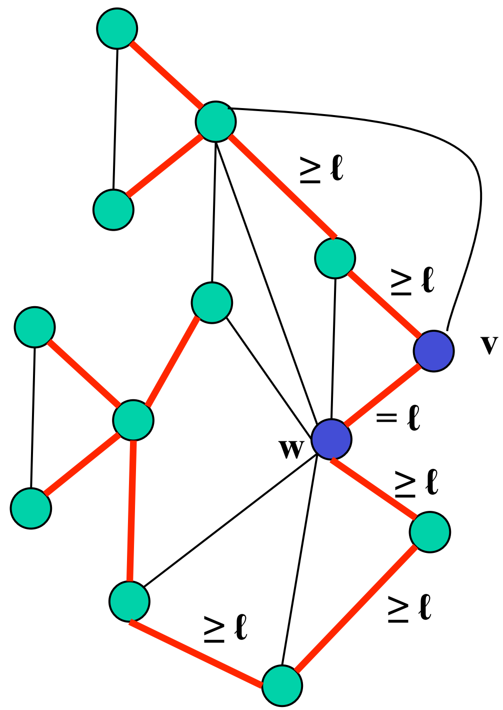
For each vertex v and each level ℓ:

- Maintain a list of incident tree edges and a list of incident non-tree edges at that level.
  (So each vertex has 2 lists per level, i.e., a total of 2 log n lists.)

Each vertex replicated in at most log n levels

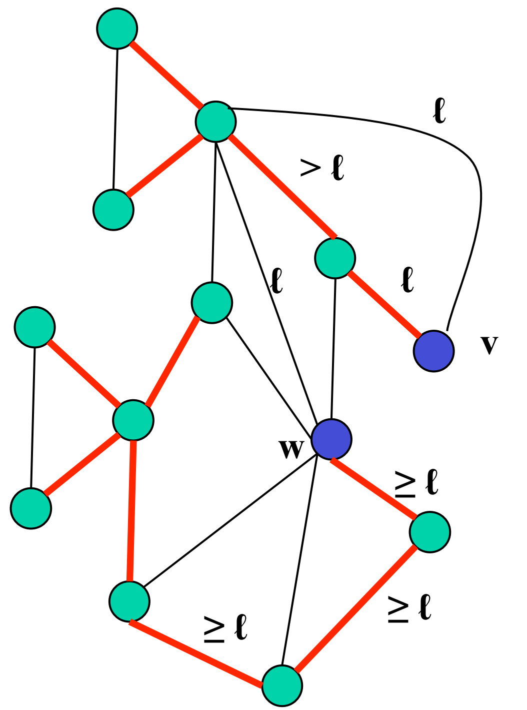
Thus, space usage will be $O(m + n \log n)$

Suppose a tree edge of level ℓ, say (v,w), is deleted. Then (v,w) belongs to some tree T of F_{ℓ}



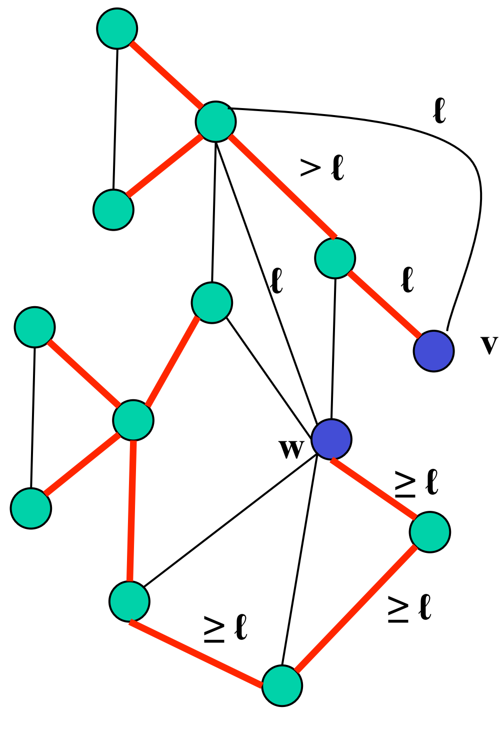
Suppose a tree edge of level ℓ, say (v,w), is deleted. Then (v,w) belongs to some tree T of F_ℓ

If there is a replacement at level ℓ then it must be incident to one of the pieces of T



Suppose a tree edge of level ℓ, say (v,w), is deleted. Then (v,w) belongs to some tree T of F_ℓ

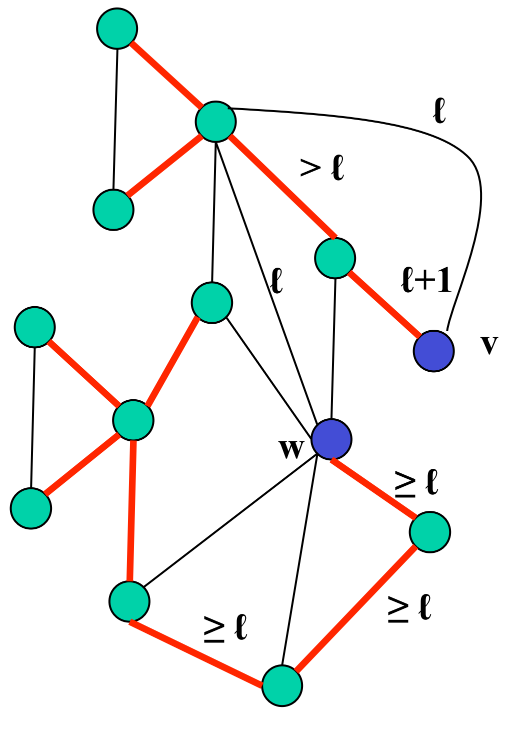
Let T_v and T_w be the pieces of T in F_ℓ containing respectively v and w after deleting edge (v,w). W.l.o.g. assume $|T_v| \leq |T_w|$.



Suppose a tree edge of level ℓ, say (v,w), is deleted. Then (v,w) belongs to some tree T of F_ℓ

Let T_v and T_w be the pieces of T in F_ℓ containing respectively v and w after deleting edge (v,w). W.l.o.g. assume $|T_v| \leq |T_w|$.

We increase to $\ell+1$ the edges of level ℓ in T_v

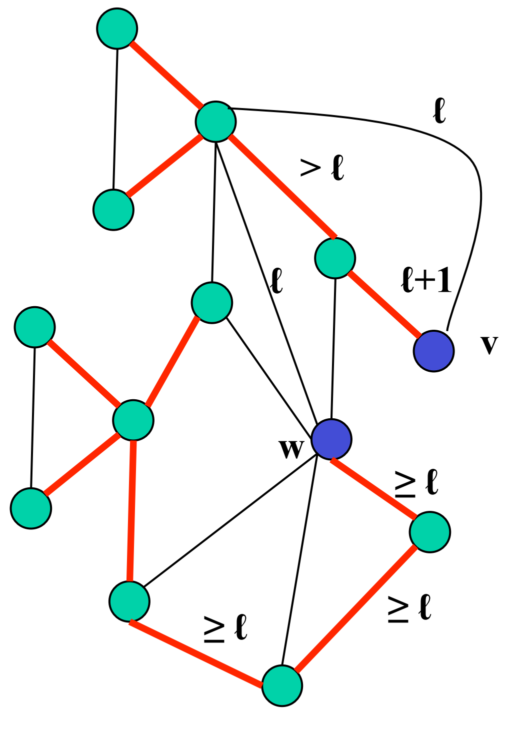


Suppose a tree edge of level ℓ, say (v,w), is deleted. Then (v,w) belongs to some tree T of F_ℓ

Let T_v and T_w be the pieces of T in F_ℓ containing respectively v and w after deleting edge (v,w). W.l.o.g. assume $|T_v| \leq |T_w|$.

We increase to $\ell+1$ the edges of level ℓ in T_v

Next, we traverse all level ℓ non-tree edges incident to T_v to find a level-ℓ replacement edge.



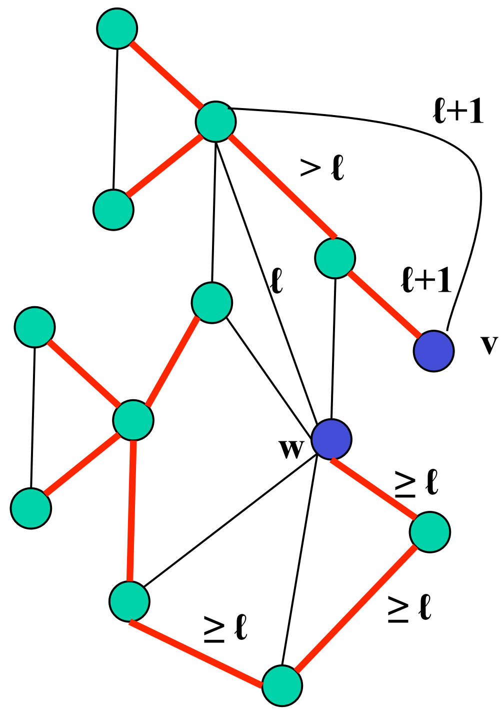
Suppose a tree edge of level ℓ, say (v,w), is deleted. Then (v,w) belongs to some tree T of F_ℓ

Let T_v and T_w be the pieces of T in F_ℓ containing respectively v and w after deleting edge (v,w). W.l.o.g. assume $|T_v| \leq |T_w|$.

We increase to $\ell+1$ the edges of level ℓ in T_v

Next, we traverse all level ℓ non-tree edges incident to T_v to find a level-ℓ replacement edge.

If a traversed edge is not a replacement we increase its level to $\ell+1$



Suppose a tree edge of level ℓ, say (v,w), is deleted. Then (v,w) belongs to some tree T of F_ℓ

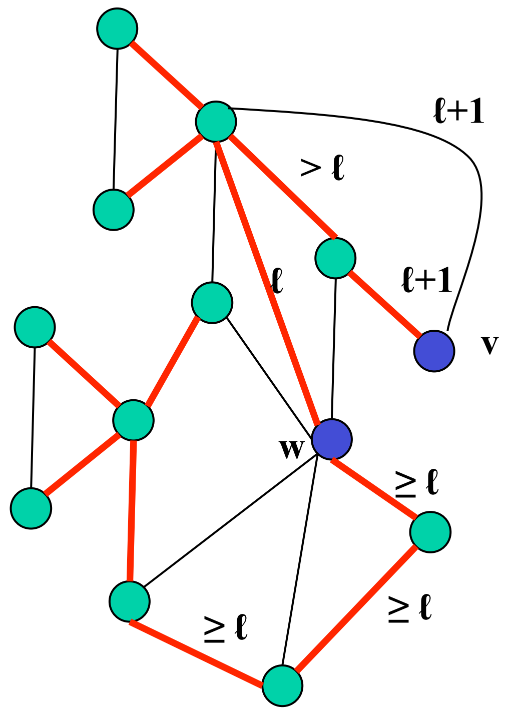
Let T_v and T_w be the pieces of T in F_ℓ containing respectively v and w after deleting edge (v,w). W.l.o.g. assume $|T_v| \leq |T_w|$.

We increase to ℓ+1 the edges of level ℓ in T_v

Next, we traverse all level ℓ non-tree edges incident to T_v to find a level-ℓ replacement edge.

If a traversed edge is not a replacement we increase its level to ℓ+1

If there is a replacement edge at level ℓ, then we are done



Suppose a tree edge of level ℓ, say (v,w), is deleted. Then (v,w) belongs to some tree T of F_ℓ

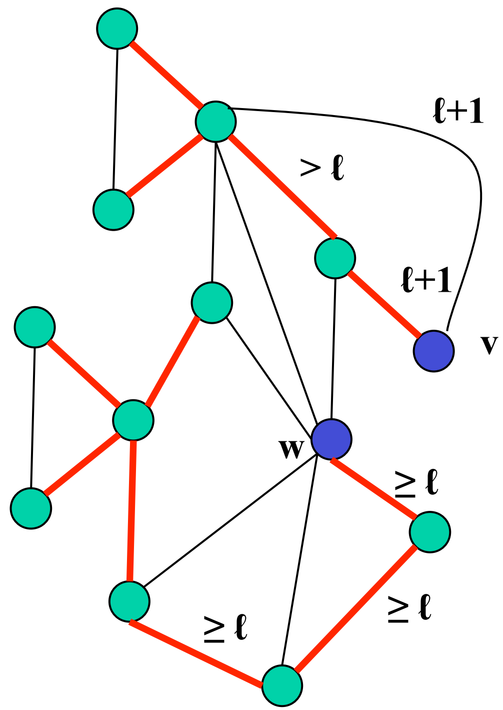
Let T_v and T_w be the pieces of T in F_ℓ containing respectively v and w after deleting edge (v,w). W.l.o.g. assume $|T_v| \leq |T_w|$.

We increase to $\ell+1$ the edges of level ℓ in T_v

Next, we traverse all level ℓ non-tree edges incident to T_v to find a level-ℓ replacement edge.

If a traversed edge is not a replacement we increase its level to $\ell+1$

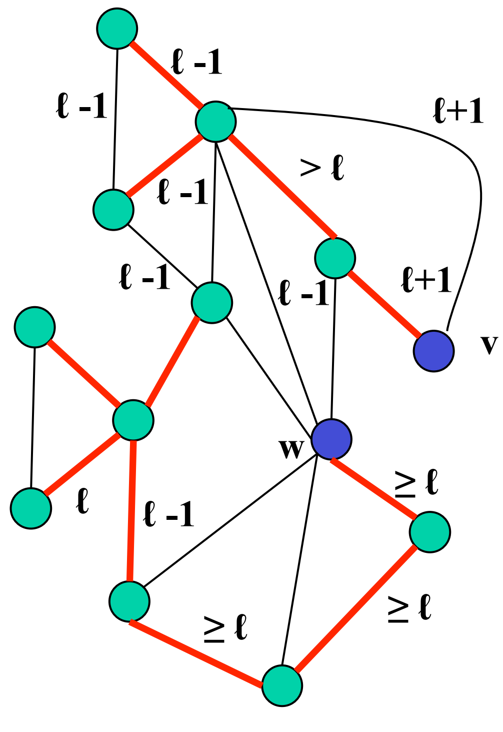
What if there is a no replacement edge at level ℓ?



If there is no replacement edge of level ℓ we look for replacement edges of level ℓ - 1

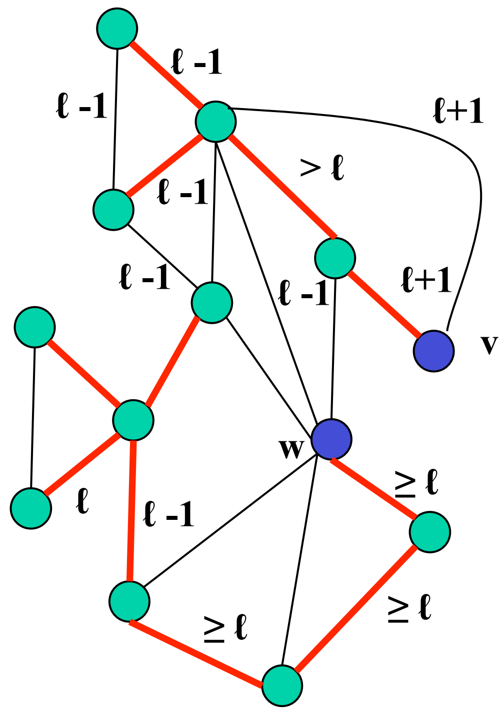
Let T_v and T_w be the trees in $F_{\ell-1}$ after deleting (v,w) containing v and w respectively

Assume $| T_v | \leq | T_w |$: then we increase the level of edges of level ℓ-1 in T_v to be ℓ and we start traversing the non-tree edges of level ℓ-1 incident to T_v



We keep going down like that level by level and either we find a replacement edge or we conclude that no replacement edge exists

As we go, we keep our invariants



Implementation

- We keep each forest $F_0 \subseteq F_1 \subseteq \ldots \subseteq F_{\log n}$ separately
- The non-tree edges of level ℓ are kept with the nodes of F_ℓ

Implementing the operations

connected(v,w) :

Check whether v and w are in the same tree of F_0

insert(v,w) :

If v and w are in different trees of F_0 add the edge to F_0 (i.e., at level 0). Otherwise, just add a non-tree edge of level 0 to v and w.

Both invariants are still satisfied.

Implementing the operations

delete(v,w):

Let ℓ be the level of edge (v,w).

• If (v,w) is a non-tree edge of level ℓ then simply delete it from v and w in $F_{\ell.}$

• Otherwise, delete (v,w) from the trees containing it in F_ℓ , $F_{\ell-1}$, … , F_0 and find a replacement edge as described before (at the highest possible level). If a replacement edge (x,y) is found at level $k \leq \ell$, then add (x,y) to F_k, F_{k-1}, … , F_0

Operations we need to do on the forests

For each ℓ, wish to maintain the forest F_ℓ together with all non-tree edges on level ℓ.

For any vertex v, wish to find the tree T_v in F_ℓ containing it

Want to be able to compute the size of T_v

Want to be able to find an edge of T_v on level ℓ, if one exists.

Want to be able to find a level ℓ non-tree edge incident to T_v, if any.

Operations we need to do on the forests

Trees in F_ℓ may be cut (when an edge is deleted) and linked (when a replacement edge is found, an edge is inserted or the level of a tree edge is increased).

Moreover, non-tree edges may be introduced and any edge may disappear on level ℓ (when the level of an edge is increased or when non-tree edges are inserted or deleted).

All this can be done in $O(\log n)$ time (by suitably augmenting ET-trees)

Analysis

- Query takes O(log n)
- Insert takes O(log n) time + charge the time to increase the level of the edge. Each level increase costs O(log n) so it $O(\log^2 n)$ total.
- Delete cuts and links O(log n) forests + level increases (charged to insert). Overall it takes $O(\log^2 n)$

(Main) History of the Problem

Update	Query	Type	Reference
$O(m^{1/2})$	$O(1)$	det/w-c	[Frederickson SICOMP'85]
$O(n^{1/2})$	$O(1)$	det/w-c	[Eppstein, Galil, I. & Nissenzweig JACM'97]
$O(\log^3 n)$	$O(\frac{\log n}{\log\log n})$	rand/amort	[Henzinger, King JACM'99]
$O(\log^2 n)$	$O(\frac{\log n}{\log\log n})$	rand/amort	[Henzinger, Thorup Rand. Struct. & Algs. '97]
$O(\log^2 n)$	$O(\frac{\log n}{\log\log n})$	det/amort	[Holm, de Lichtenberg & Thorup JACM'01]
$O(\log n\,(\log\log n))$	$O(\frac{\log n}{\log\log\log n})$	rand/amort	[Thorup STOC' 00]
$O(\frac{\log^2 n}{\log\log n})$	$O(\frac{\log n}{\log\log n})$	det/amort	[Wulff-Nilsen SODA'13]
$O(\log^5 n)$	$O(\frac{\log n}{\log\log n})$	rand/w-c	[Kapron, King & Mountjoy SODA'13]

Best Bounds

Update	Query	Type	Reference
$O(m^{1/2})$	$O(1)$	det/w-c	[Frederickson SICOMP'85]
$O(n^{1/2})$	$O(1)$	det/w-c	[Eppstein, Galil, I. & Nissenzweig JACM'97]
$O(\log^3 n)$	$O(\frac{\log n}{\log \log n})$	rand/amort	[Henzinger, King JACM'99]
$O(\log^2 n)$	$O(\frac{\log n}{\log \log n})$	rand/amort	[Henzinger, Thorup Rand. Struct. & Algs. '97]
$O(\log^2 n)$	$O(\frac{\log n}{\log \log n})$	det/amort	[Holm, de Lichtenberg & Thorup JACM'01]
$O(\log n (\log \log n))$	$O(\frac{\log n}{\log \log \log n})$	rand/amort	[Thorup STOC' 00]
$O(\frac{\log^2 n}{\log \log n})$	$O(\frac{\log n}{\log \log n})$	det/amort	[Wulff-Nilsen SODA'13]
$O(\log^5 n)$	$O(\frac{\log n}{\log \log n})$	rand/w-c	[Kapron, King & Mountjoy SODA'13]

Best Bounds

Update	Query	Type	Reference
$O(n^{1/2})$	$O(1)$	det/w-c	[Eppstein, Galil, I. & Nissenzweig JACM'97]
$O(\log n\,(\log\log n))$	$O(\frac{\log n}{\log\log\log n})$	rand/amort	[Thorup STOC' 00]
$O(\frac{\log^2 n}{\log\log n})$	$O(\frac{\log n}{\log\log n})$	det/amort	[Wulff-Nilsen SODA'13]
$O(\log^5 n)$	$O(\frac{\log n}{\log\log n})$	rand/w-c	[Kapron, King & Mountjoy SODA'13]

Lower Bounds

Update	Query	Type	Reference
$O(n^{1/2})$	$O(1)$	det/w-c	[Eppstein, Galil, I. & Nissenzweig JACM'97]
$O(\log n\,(\log\log n))$	$O(\frac{\log n}{\log\log\log n})$	rand/amort	[Thorup STOC' 00]
$O(\frac{\log^2 n}{\log\log n})$	$O(\frac{\log n}{\log\log n})$	det/amort	[Wulff-Nilsen SODA'13]
$O(\log^5 n)$	$O(\frac{\log n}{\log\log n})$	rand/w-c	[Kapron, King & Mountjoy SODA'13]

$$O(x \log n) \Rightarrow \Omega(\frac{\log n}{\log x})$$

$$\Omega(\frac{\log n}{\log x}) \Leftarrow O(x \log n)$$

[Patrascu, Demaine SICOMP'06]

Open: Close the Gaps

Update	Query	Type	Reference
$O(n^{1/2})$	$O(1)$	det/w-c	[Eppstein, Galil, I. & Nissenzweig JACM'97]
$O(\log n\,(\log\log n))$	$O(\frac{\log n}{\log\log\log n})$	rand/amort	[Thorup STOC' 00]
$O(\frac{\log^2 n}{\log\log n})$	$O(\frac{\log n}{\log\log n})$	det/amort	[Wulff-Nilsen SODA'13]
$O(\log^5 n)$	$O(\frac{\log n}{\log\log n})$	rand/w-c	[Kapron, King & Mountjoy SODA'13]

$$O(x \log n) \Rightarrow \Omega(\frac{\log n}{\log x})$$

$$\Omega(\frac{\log n}{\log x}) \Leftarrow O(x \log n)$$

[Patrascu, Demaine SICOMP'06]

Open Problems

- Deterministic algorithm with O(polylog n) update and query in the worst case?
- Deterministic / randomized algorithm with O(log n) update and query?
- Deterministic / randomized algorithm with o(log n) update and O(polylog n) query?

References

D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. J. ACM, 44(5):669–696, 1997. See also FOCS'92.

G. N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. SIAM J. Comput., 14(4):781–798, 1985. See also STOC'83.

M. R. Henzinger and V. King. Randomized dynamic graph algorithms with polylogarithmic time per operation. Proc. 27th ACM Symposium on Theory of Computing (STOC), 1995, pp. 519–527.

M. R. Henzinger and M. Thorup. Sampling to provide or to bound: With applications to fully dynamic graph algorithms. Random Structures and Algorithms, 11(4):369–379, 1997. See also ICALP'96.

References

J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J. ACM, 48 (4): 723–760, 2001. See also STOC'98.

B. M. Kapron, V. King, and B. Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. 24th ACM-SIAM Symposium on Discrete Algorithms (SODA) 2013: 1131-1142.

M. Patrascu and E. Demaine. Logarithmic Lower Bounds in the Cell-Probe Model. SIAM J. Comput., 35(4): 2006. See also STOC 2004.

References

M. Thorup. Near-optimal fully-dynamic graph connectivity. Proc. 32nd ACM Symposium on Theory of Computing (STOC), 2000, pp. 343–350.

C. Wulff-Nilsen: Faster Deterministic Fully-Dynamic Graph Connectivity. 24th ACM-SIAM Symposium on Discrete Algorithms (SODA) 2013: 1757-1769