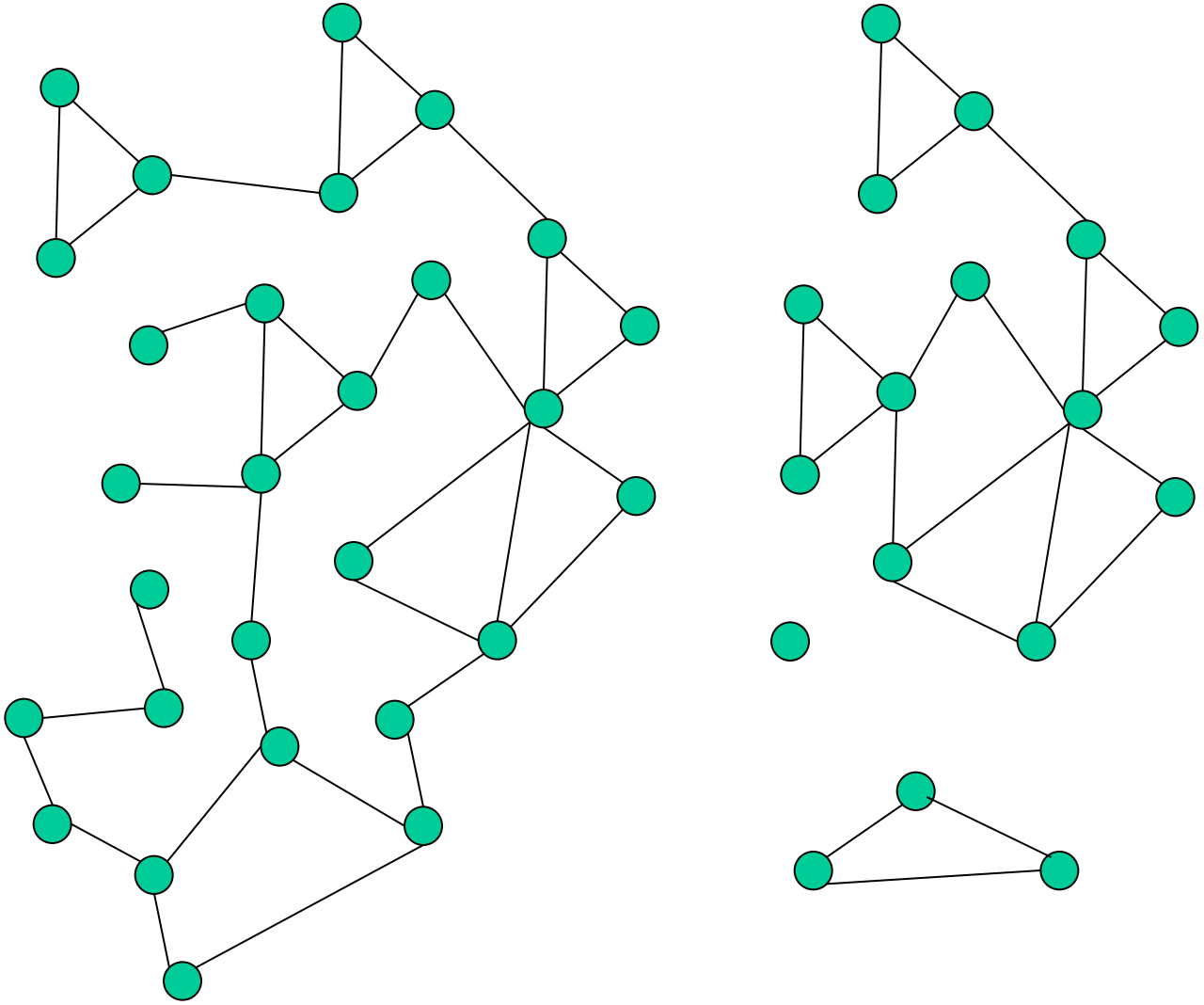
Dynamic graph connectivity

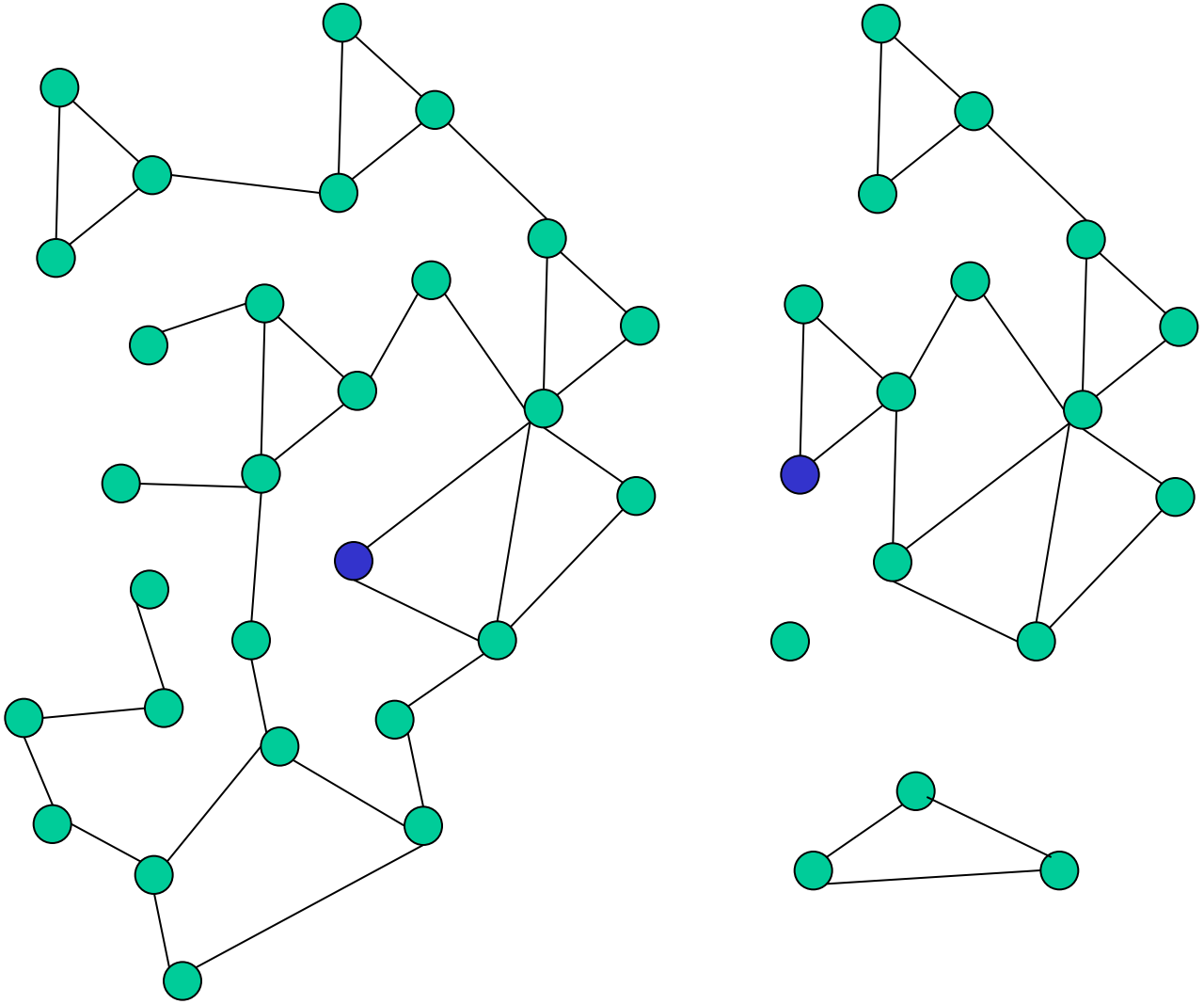
Holm, Lichtenberg, Thorup (98)

(based on Henzinger & King (95))

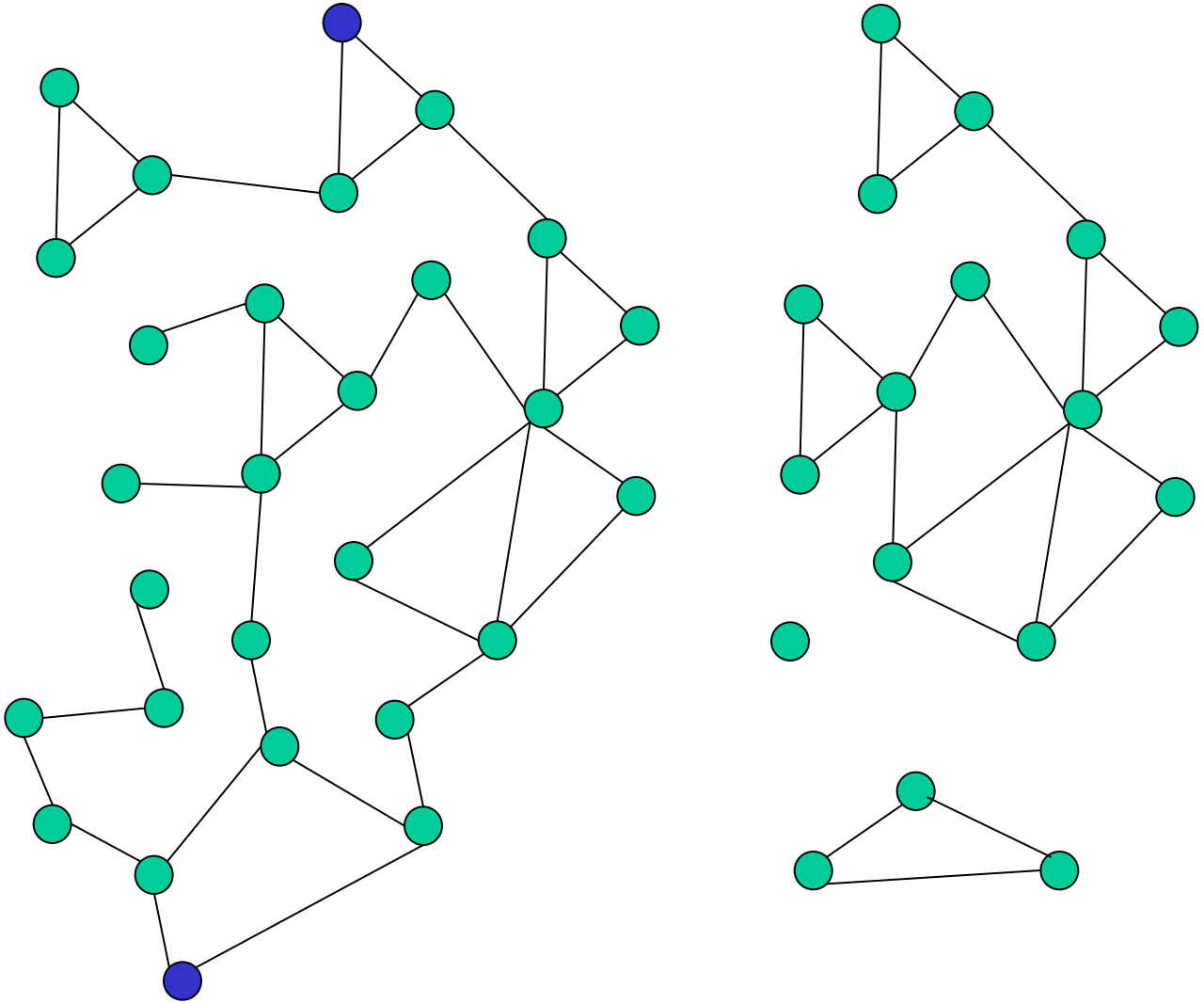
connected(v,w)



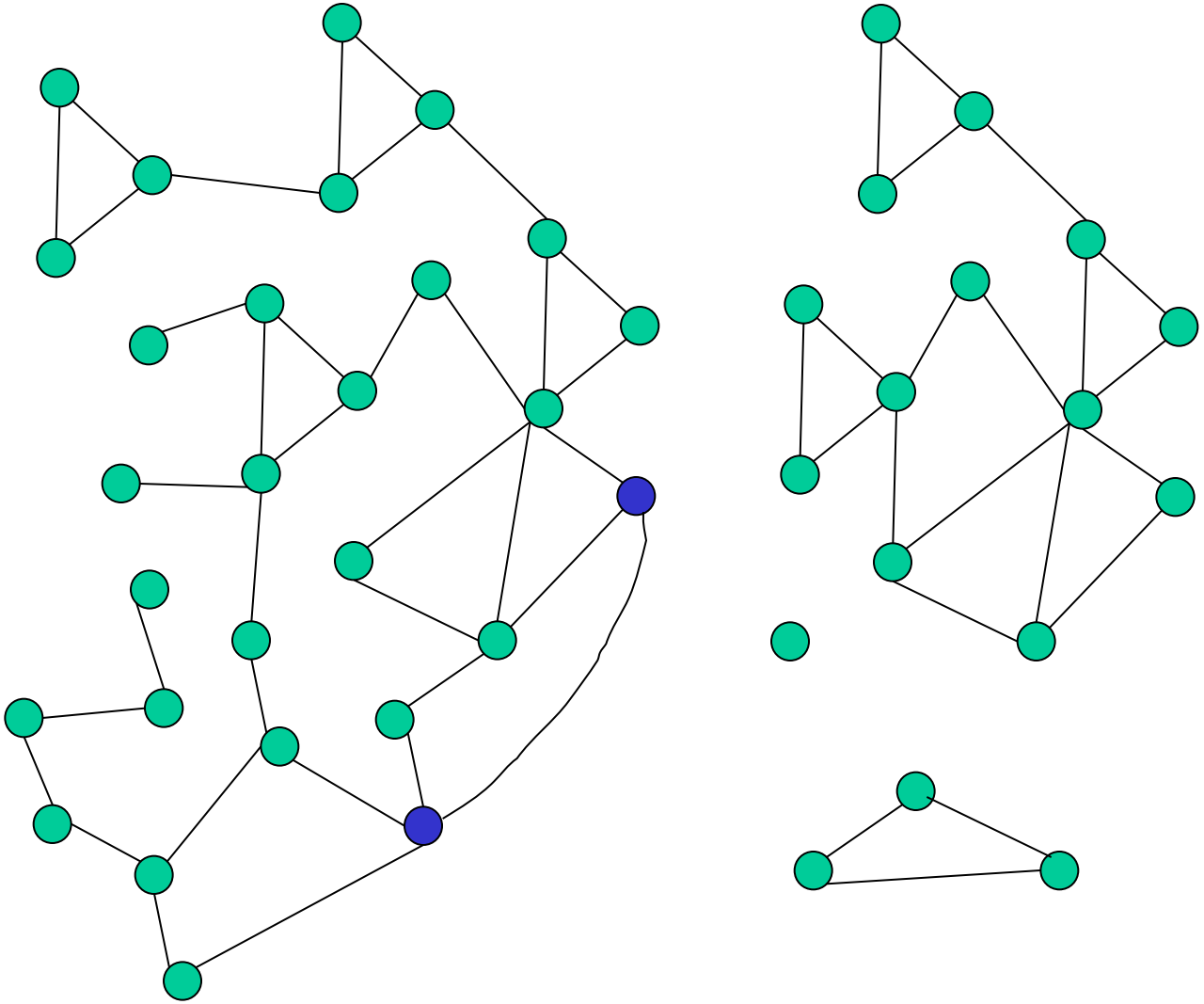
connected(v,w)



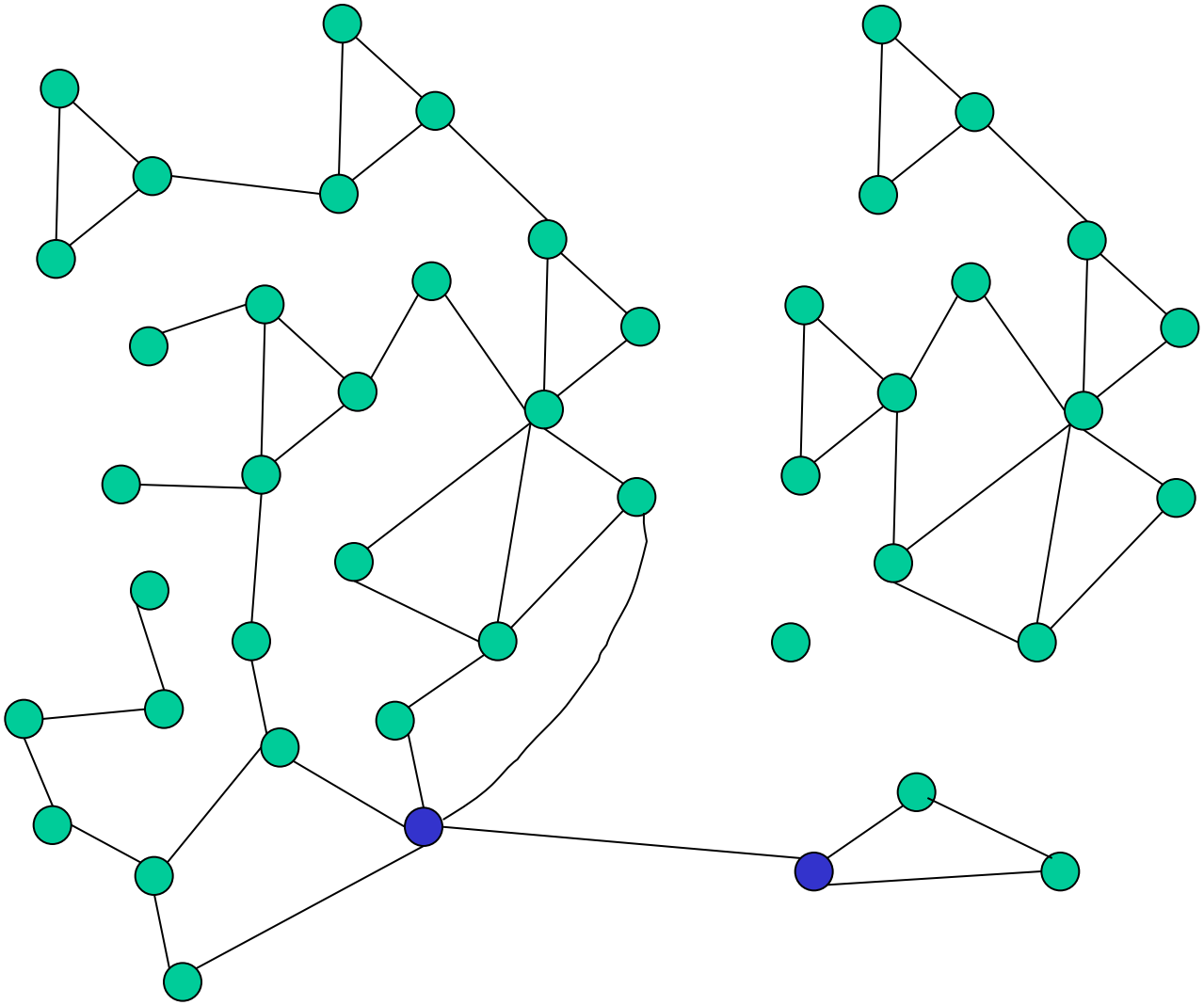
connected(v,w)



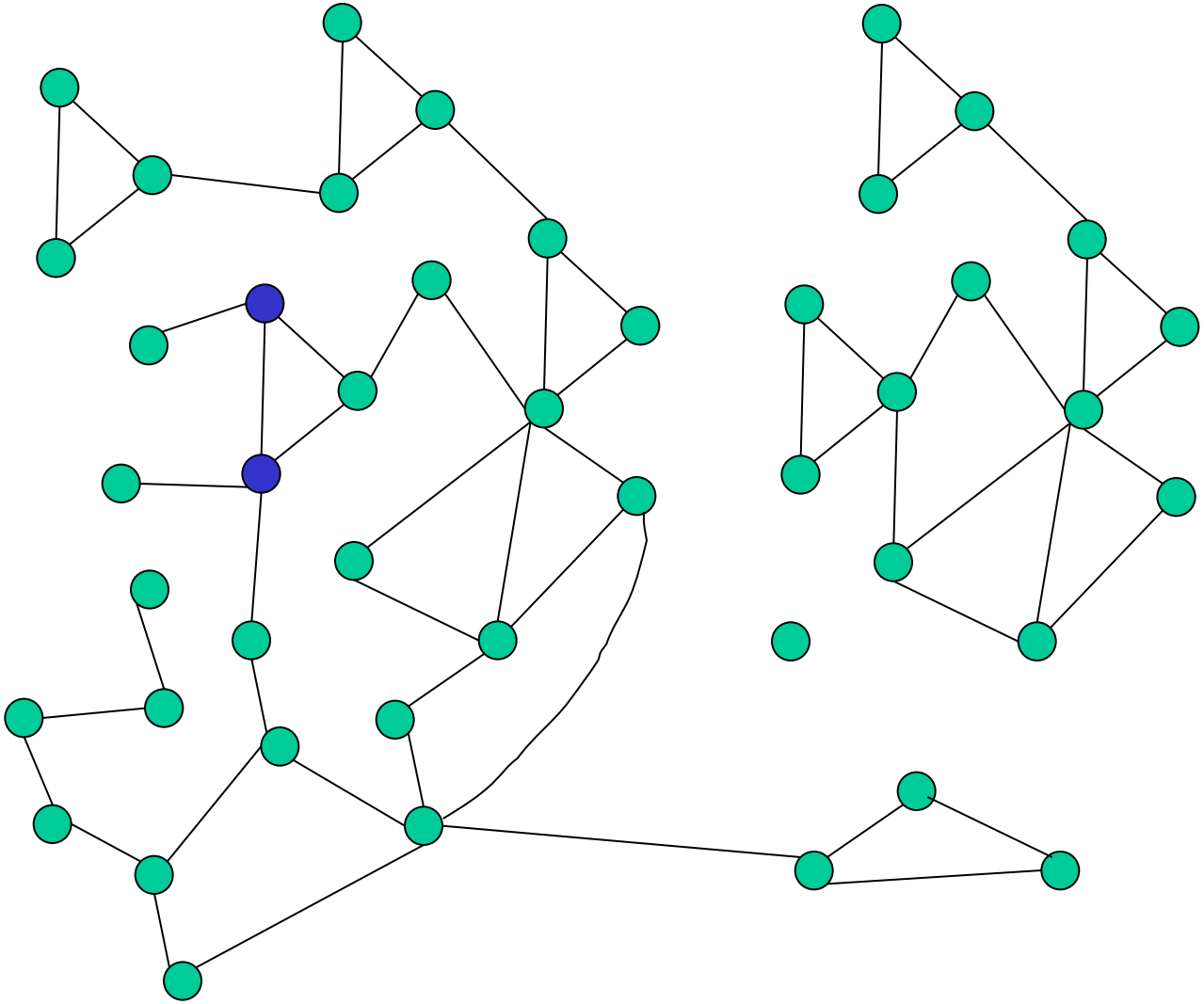
insert(v,w)



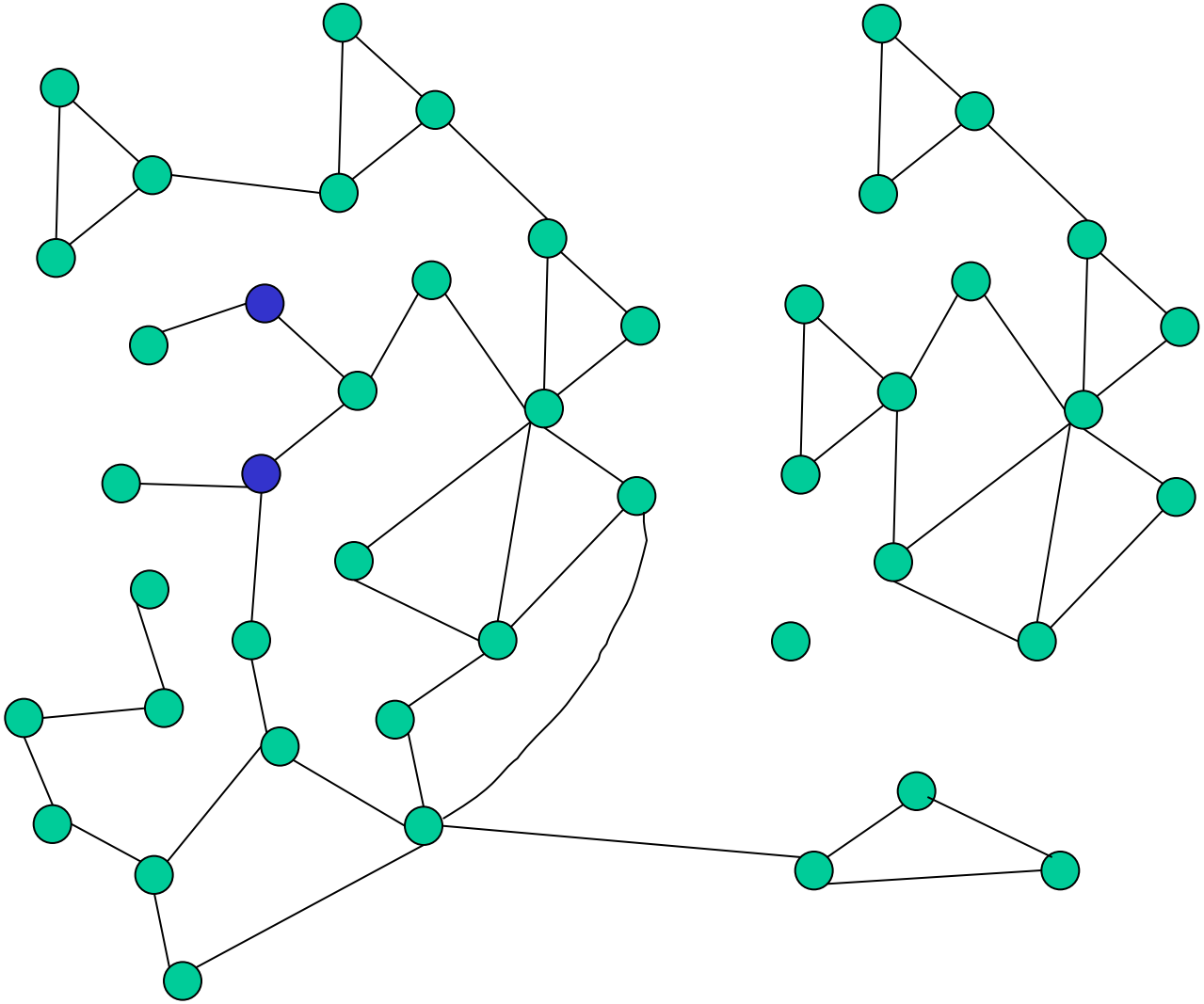
insert(v,w)



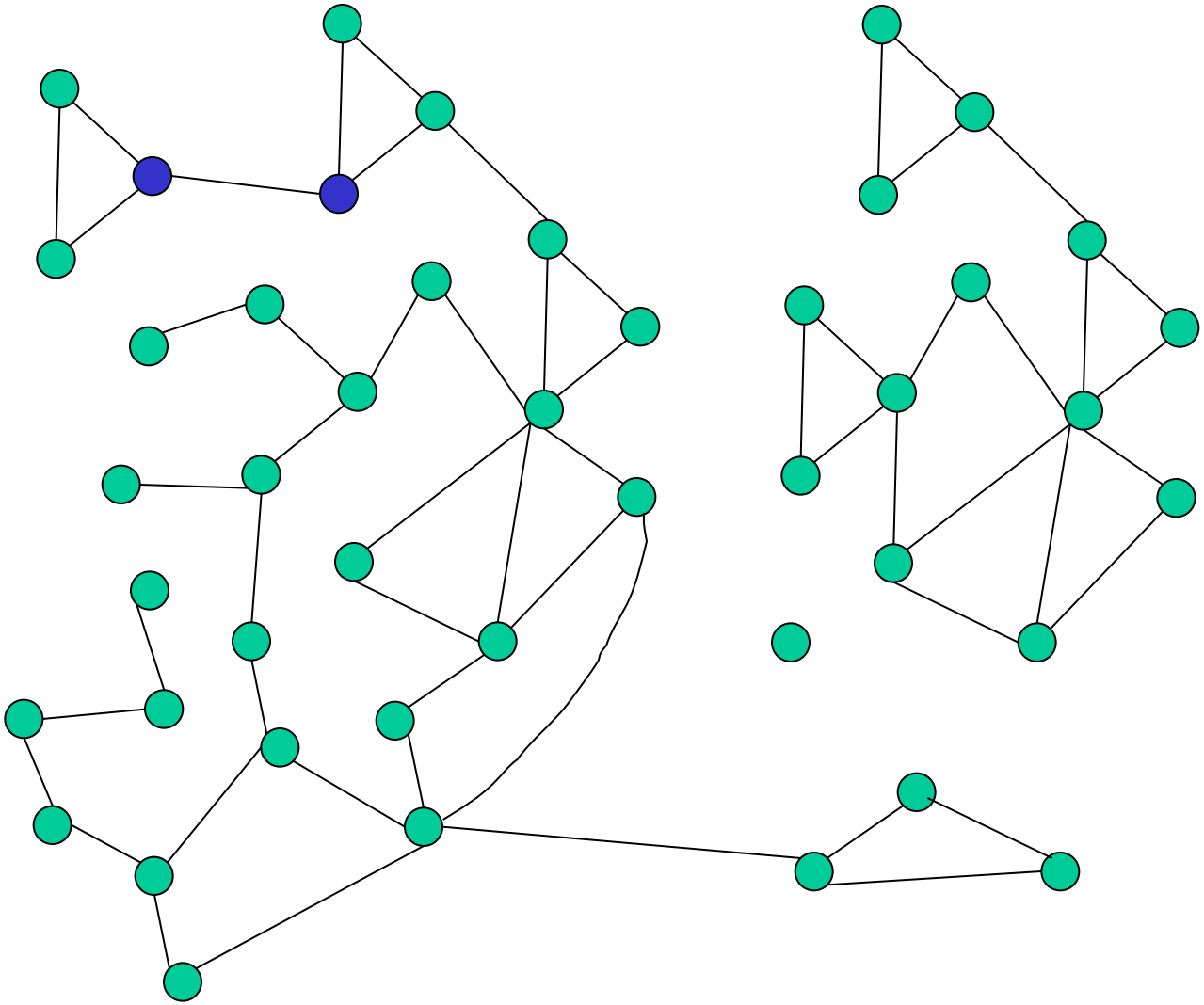
delete(v,w)



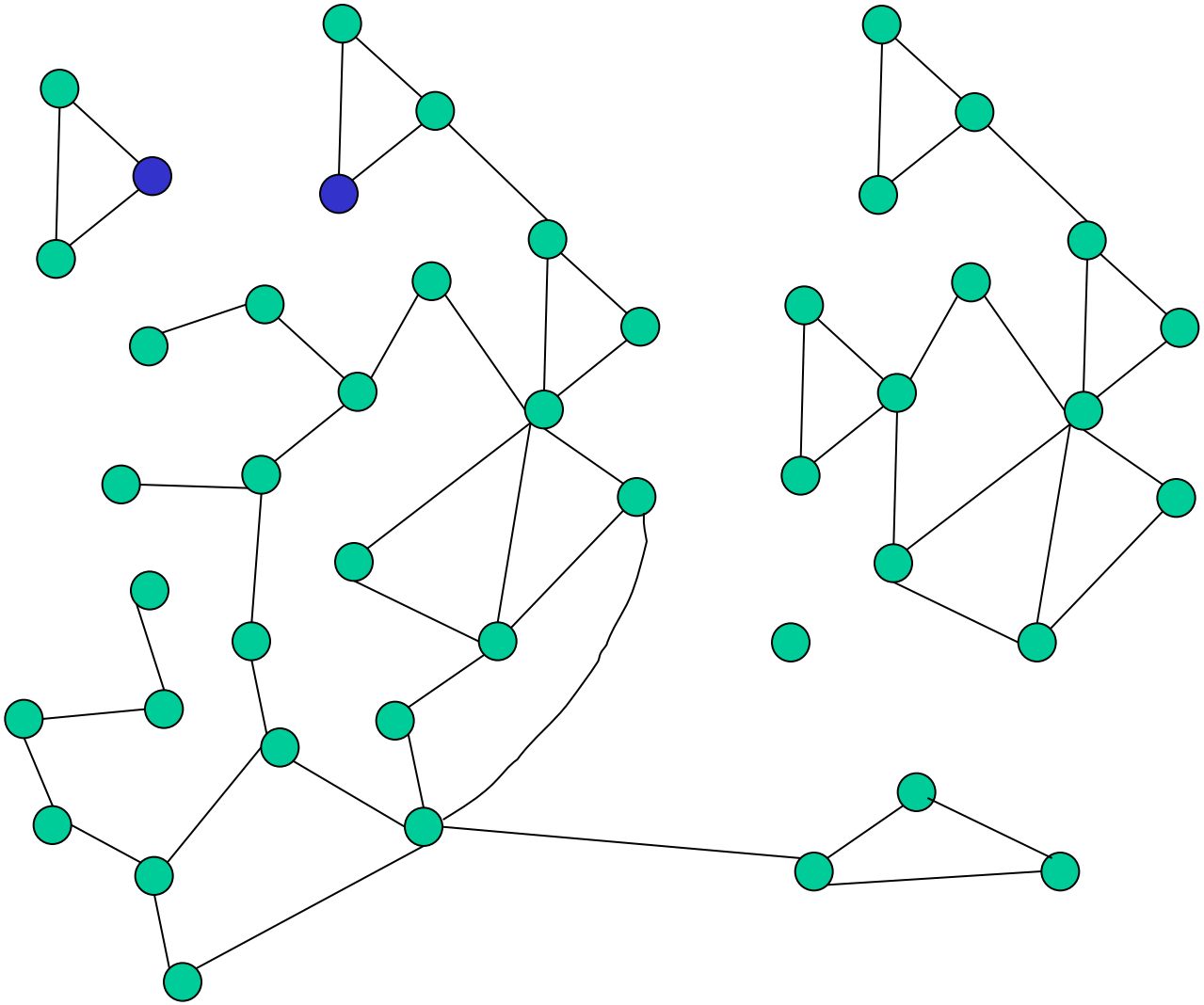
delete(v,w)



delete(v,w)



delete(v,w)

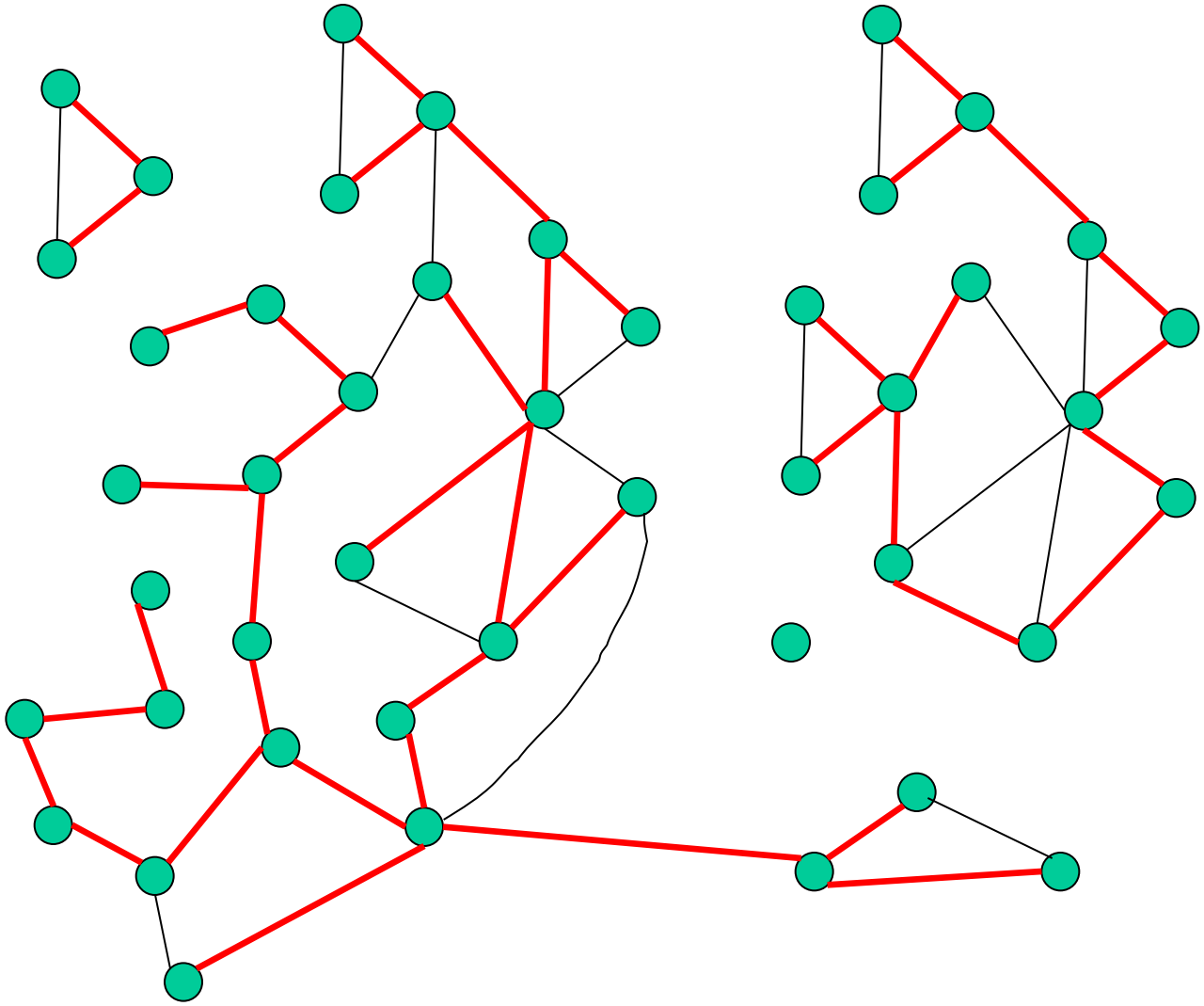


Observations

Without delete a Union-Find data structure would do

Lets reduce the problem to a problem on trees

We 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

Operations we need to do on the forest

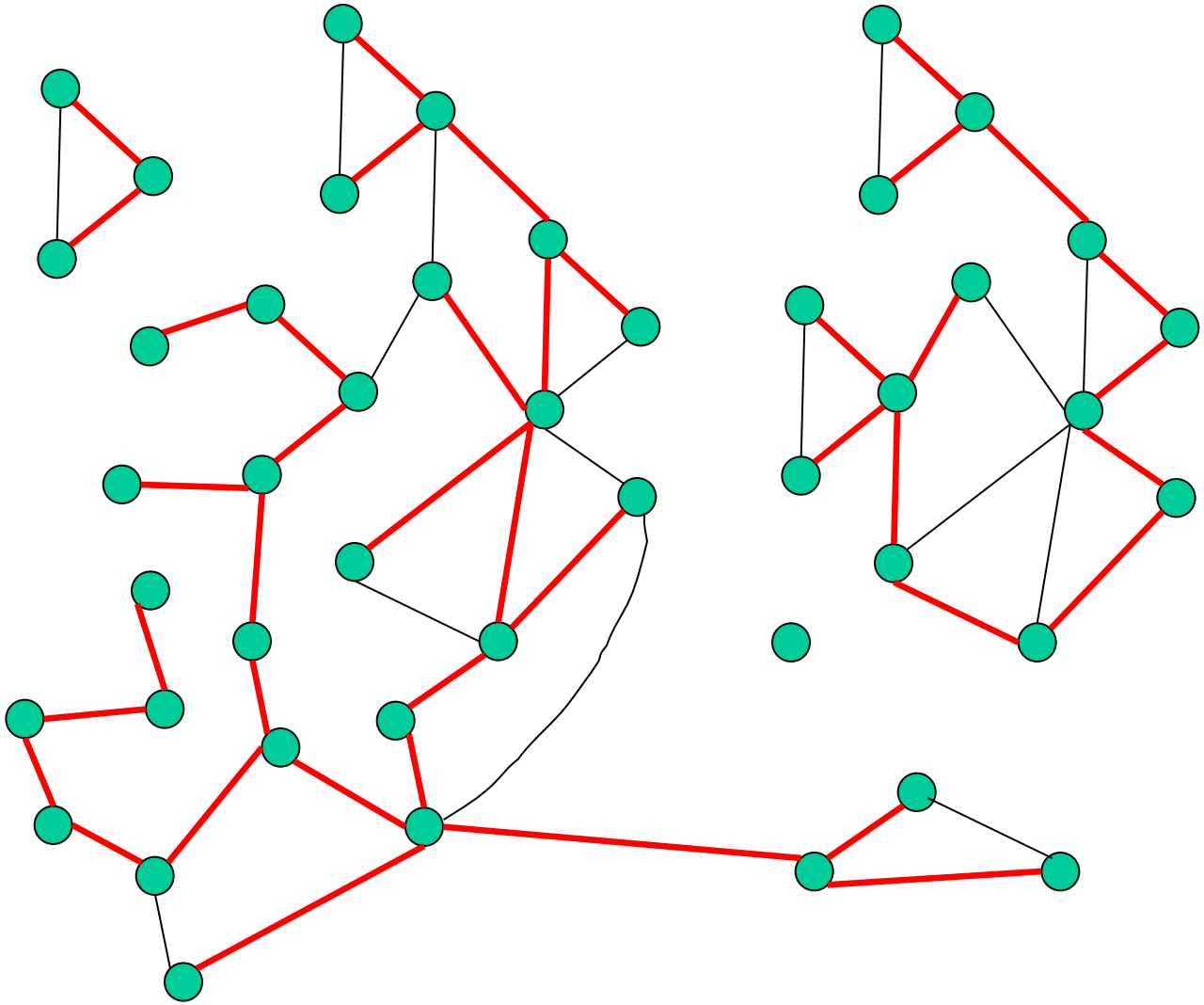
link(v,w) : assume v and w are in different trees

cut(v,w) : assume v and w are adjacent in a tree

findtree(v)

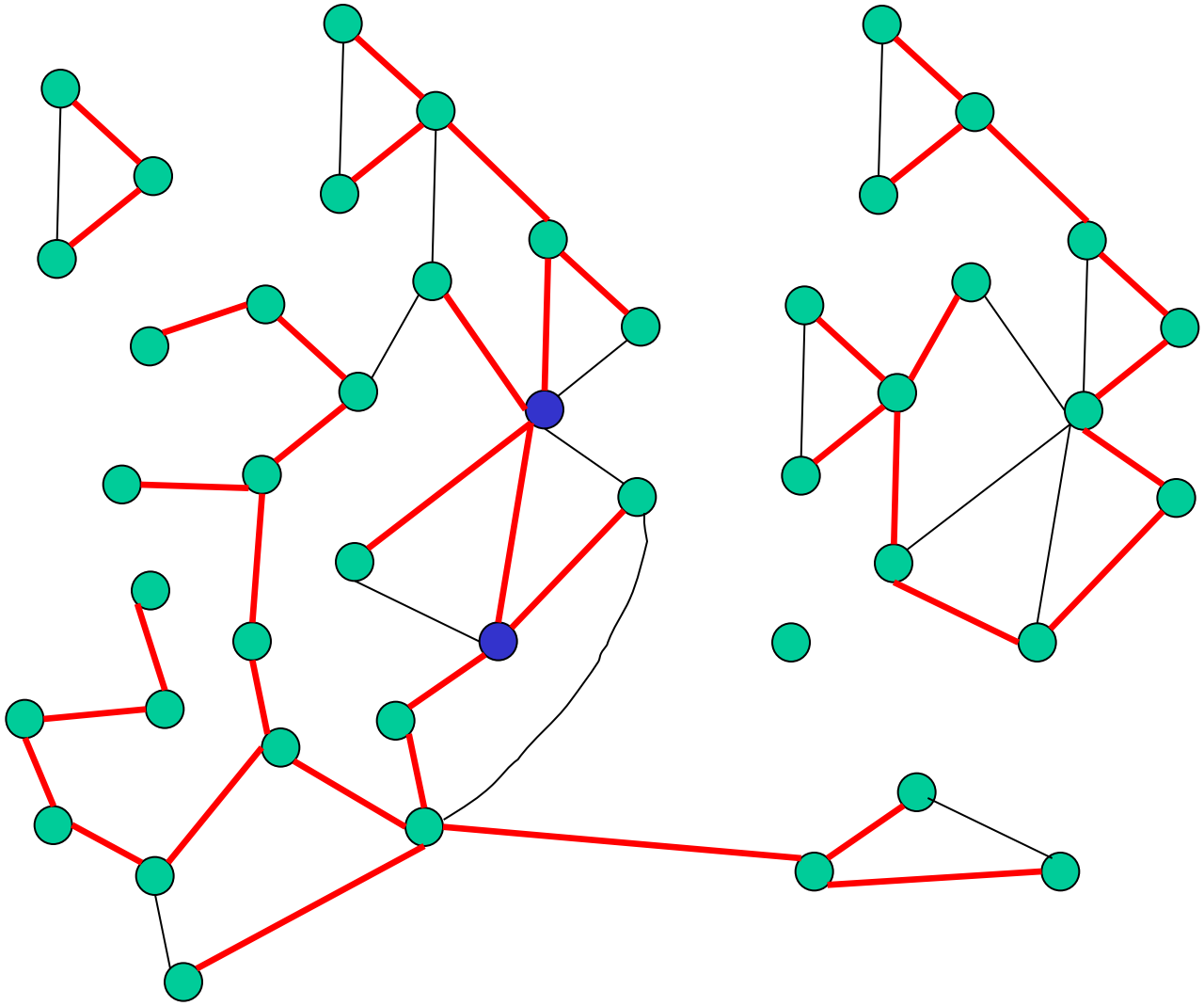
But even if
we know how
to do that

we still have
a problem
when deleting
a tree edge



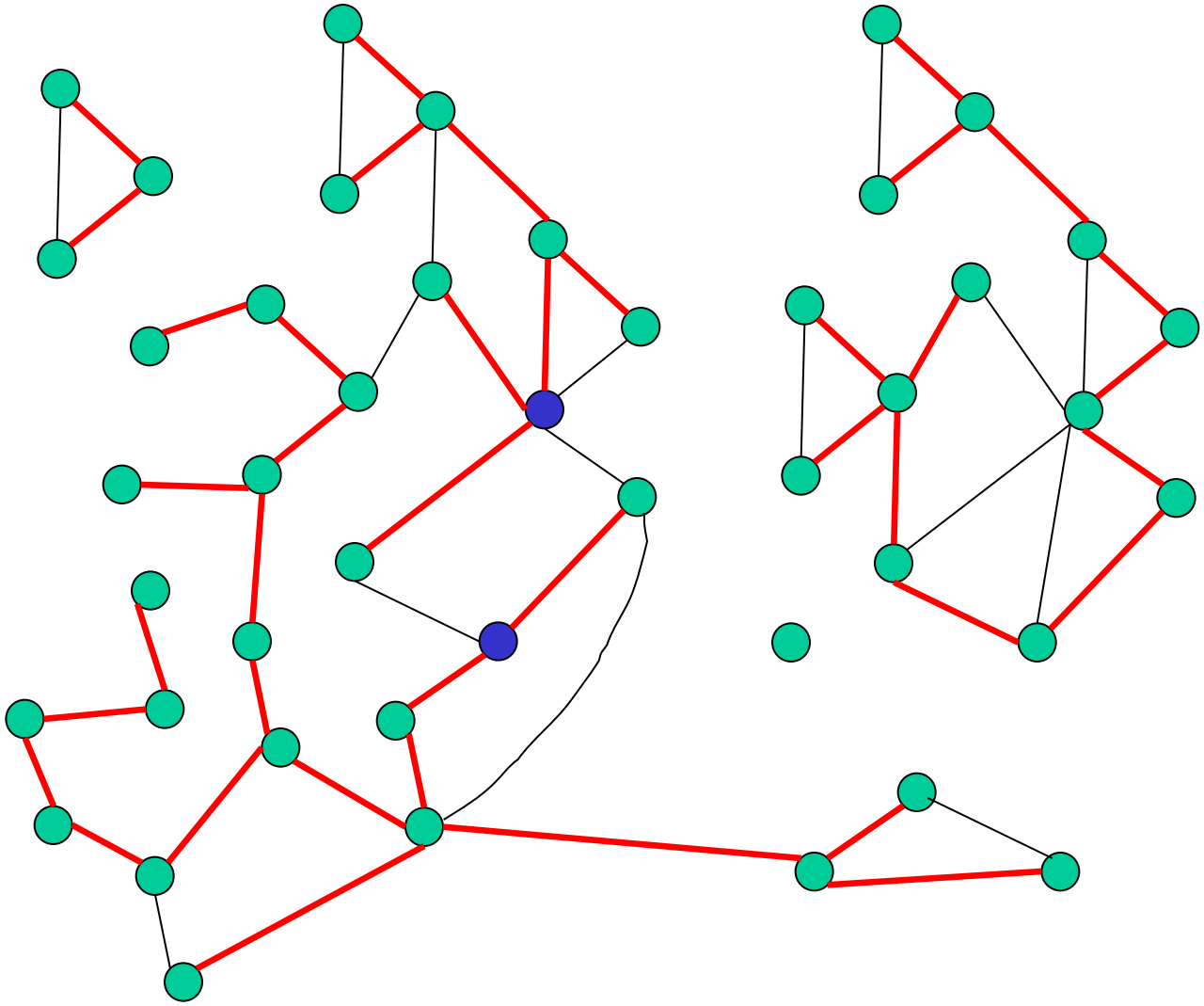
But even if
we know how
to do that

we still have
a problem
when deleting
a tree edge

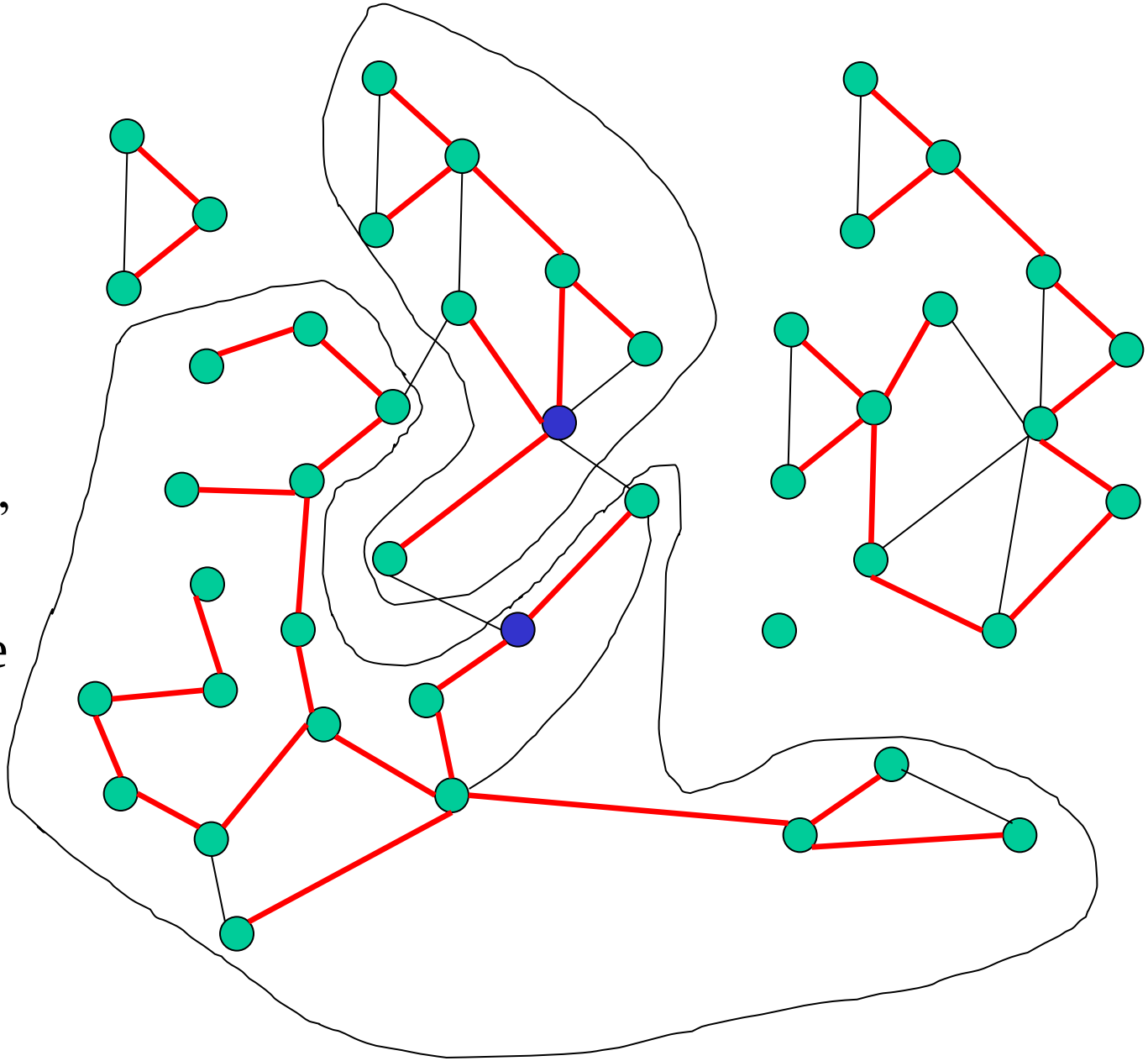


But even if
we know how
to do that

we still have
a problem
when deleting
a tree edge

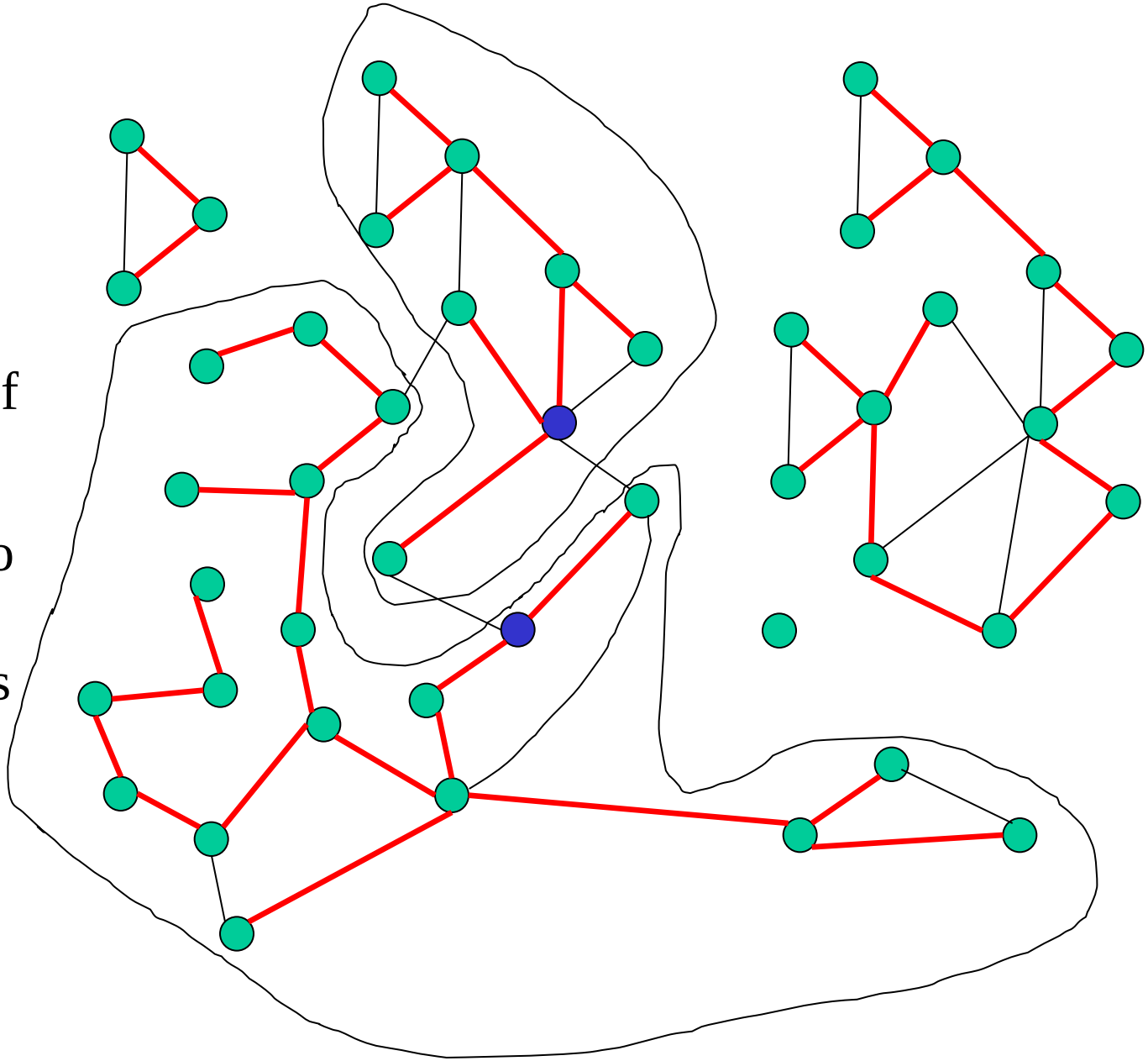


How do we find out if there is a “replacement” edge that reconnects the tree or not ?



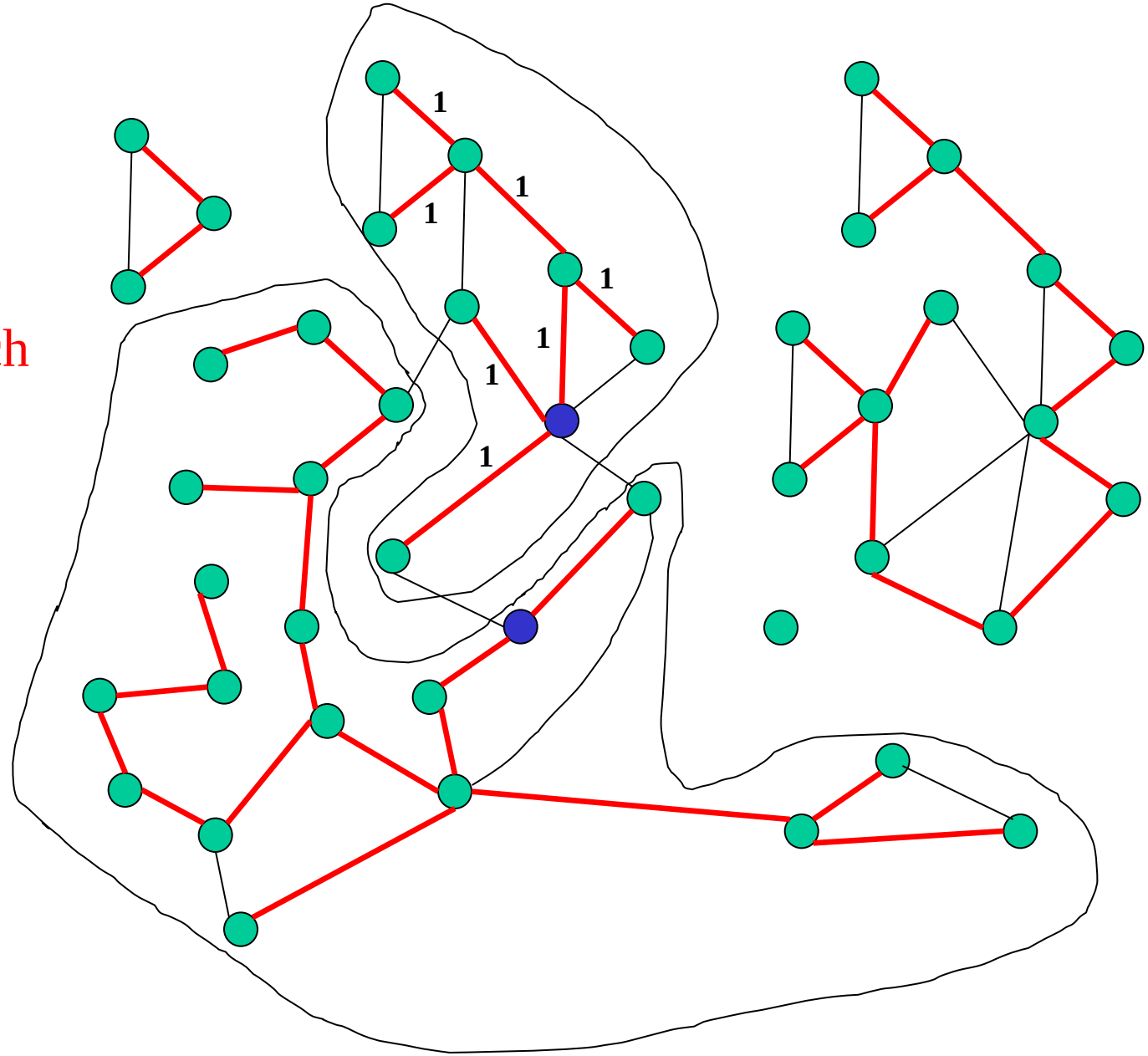
We will traverse one of the trees

but we need to accumulate information as we do that



Associate a level with each edge

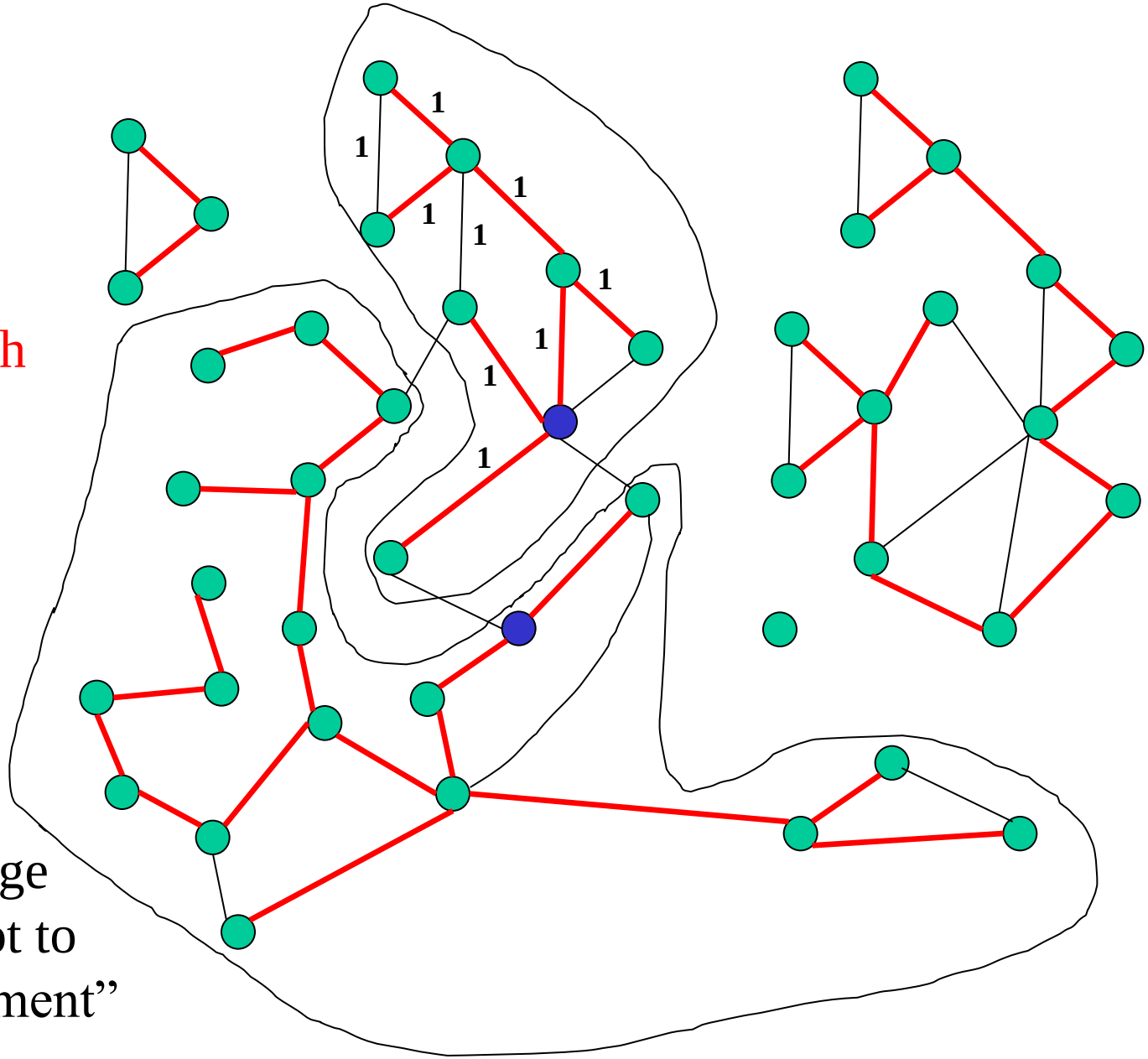
Increase the level of the edges of the smaller tree

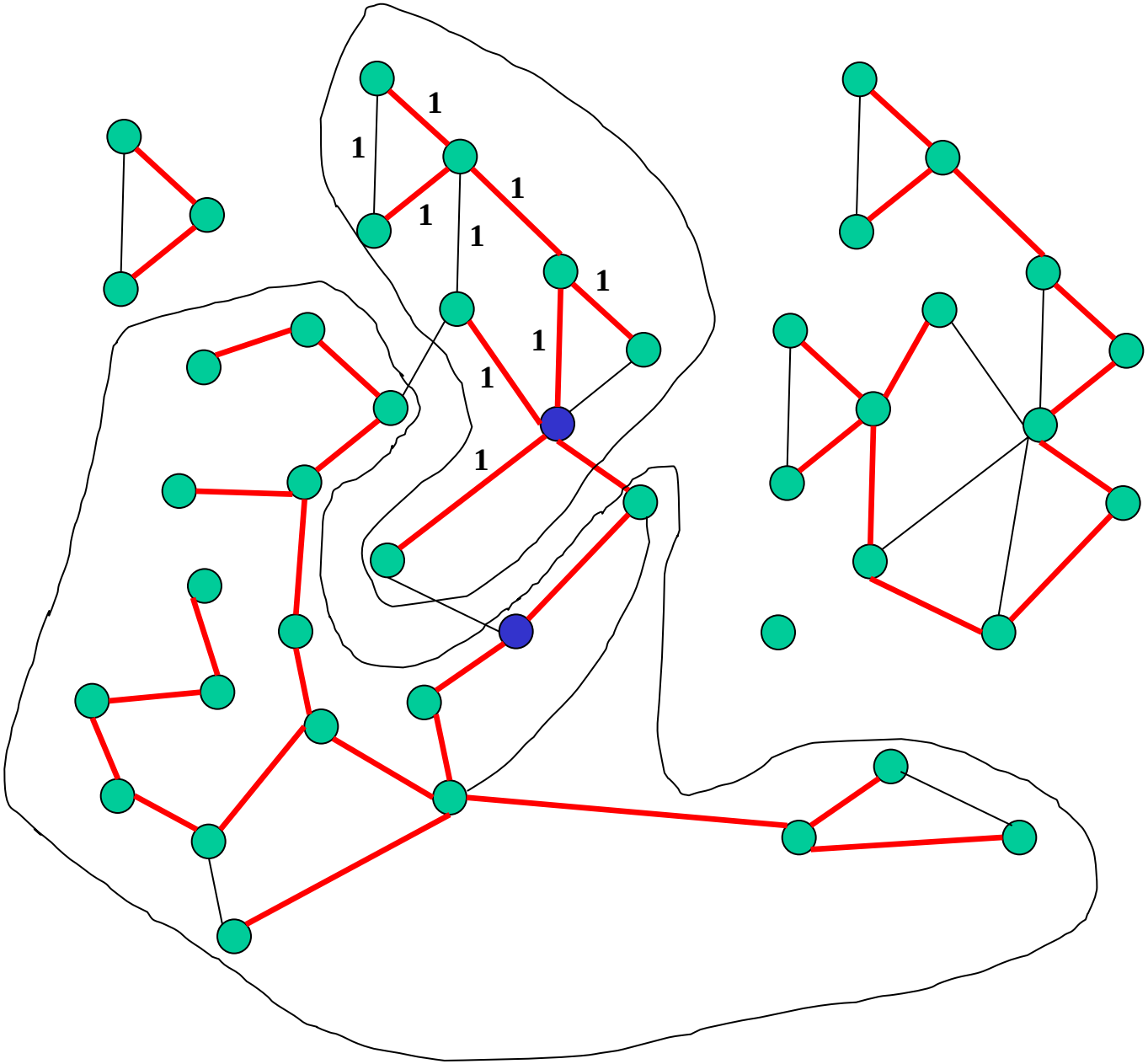


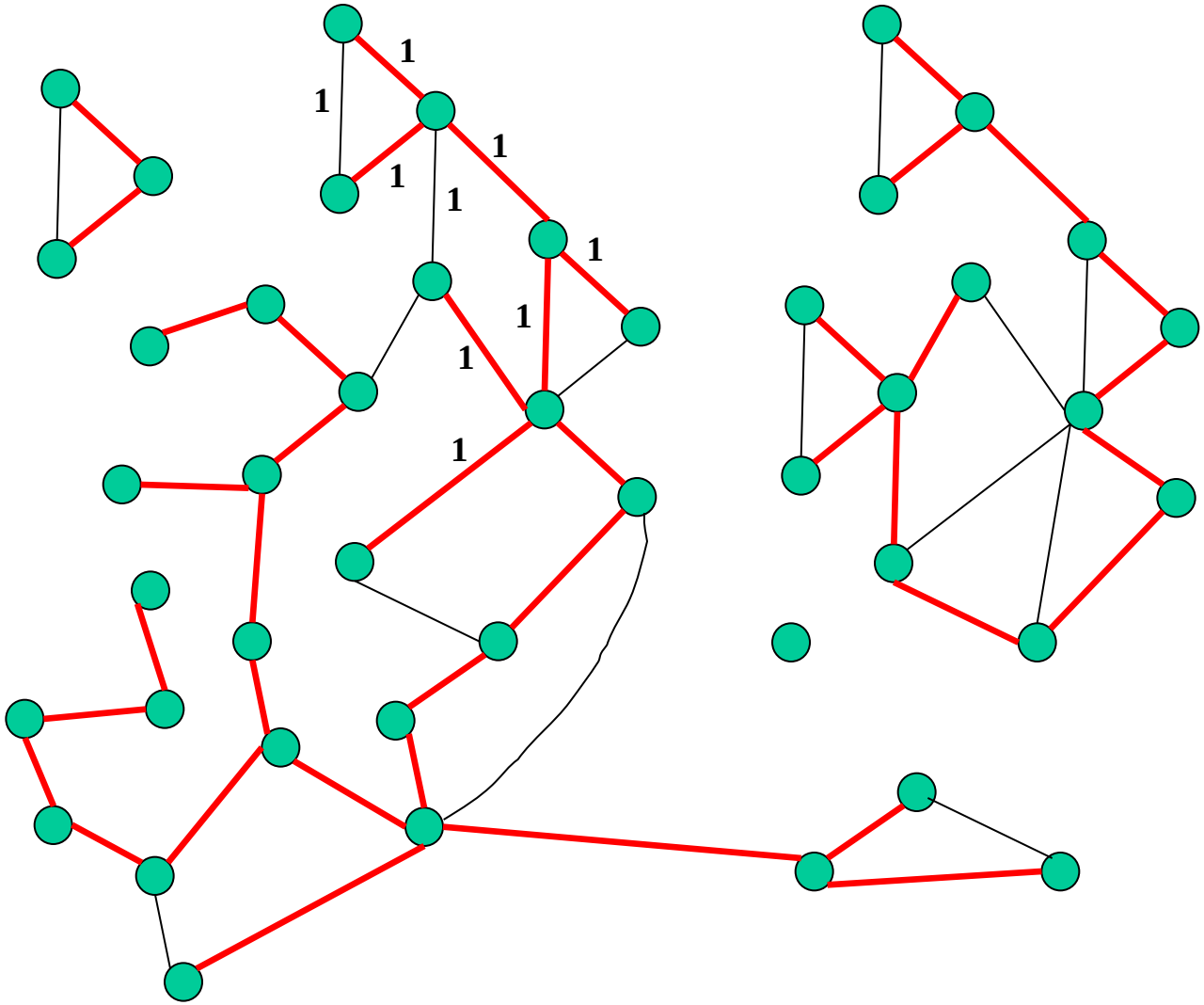
Associate a level with each edge

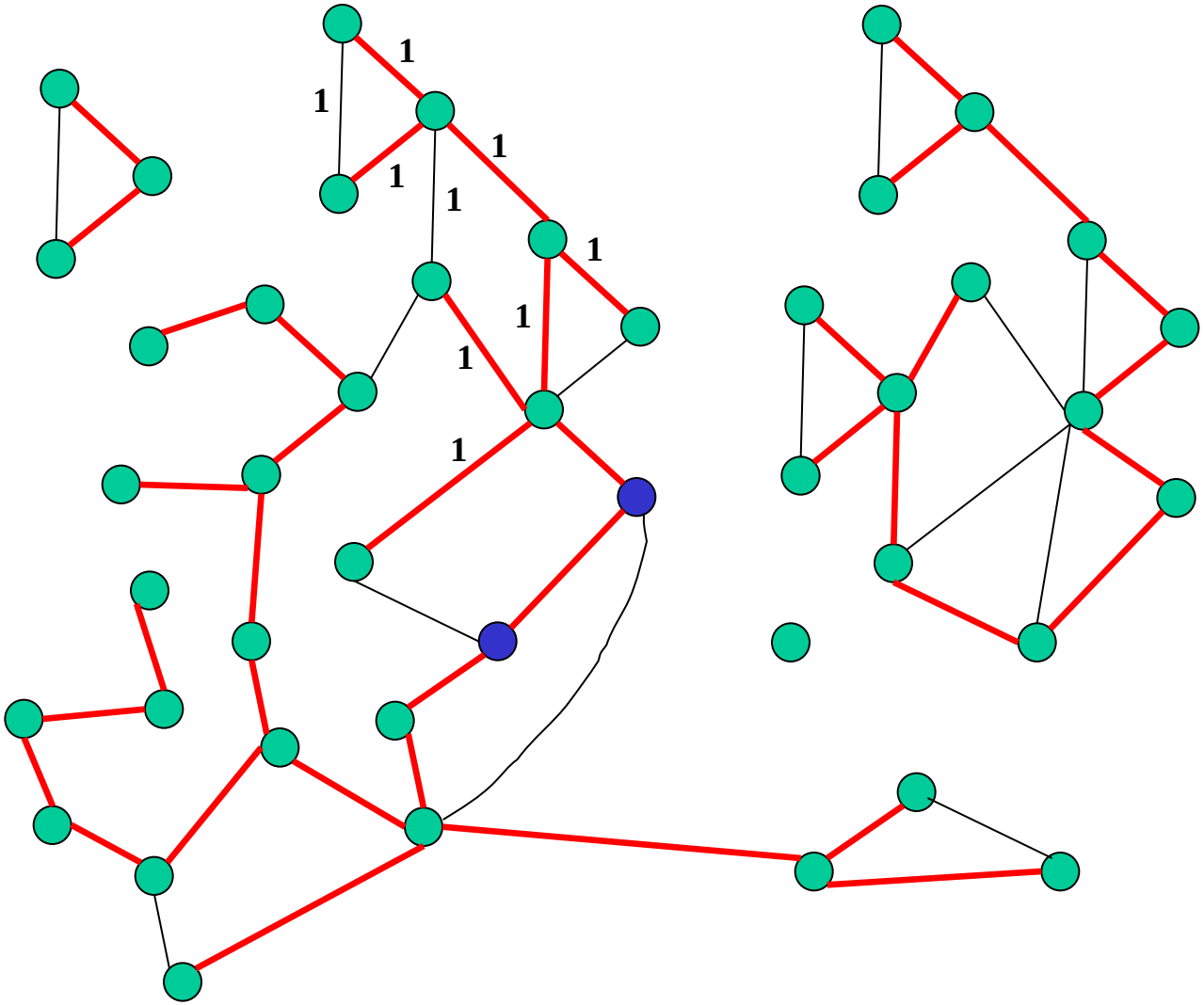
Increase the level of the edges of the smaller tree

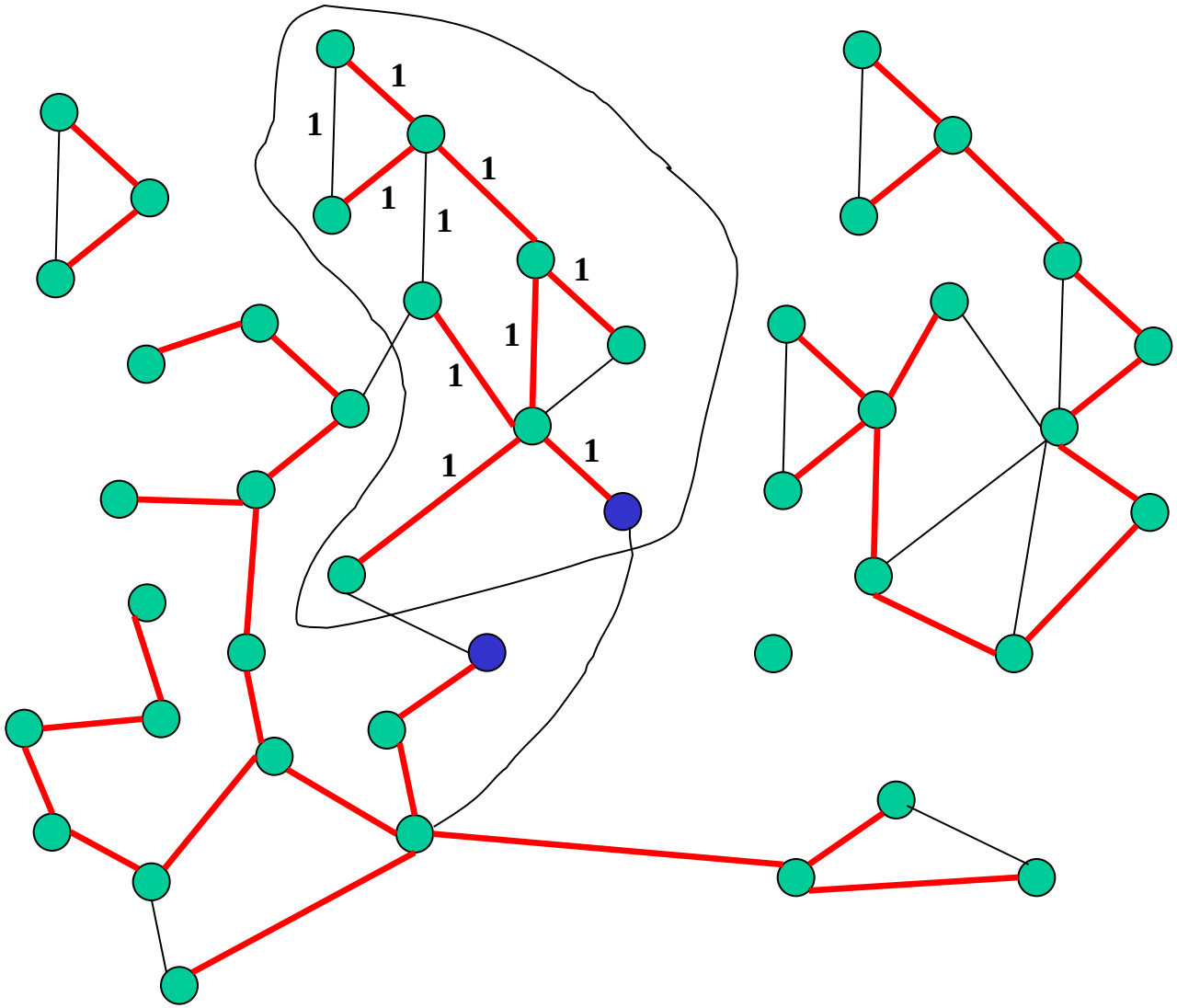
and of any edge discovered not to be a “replacement”



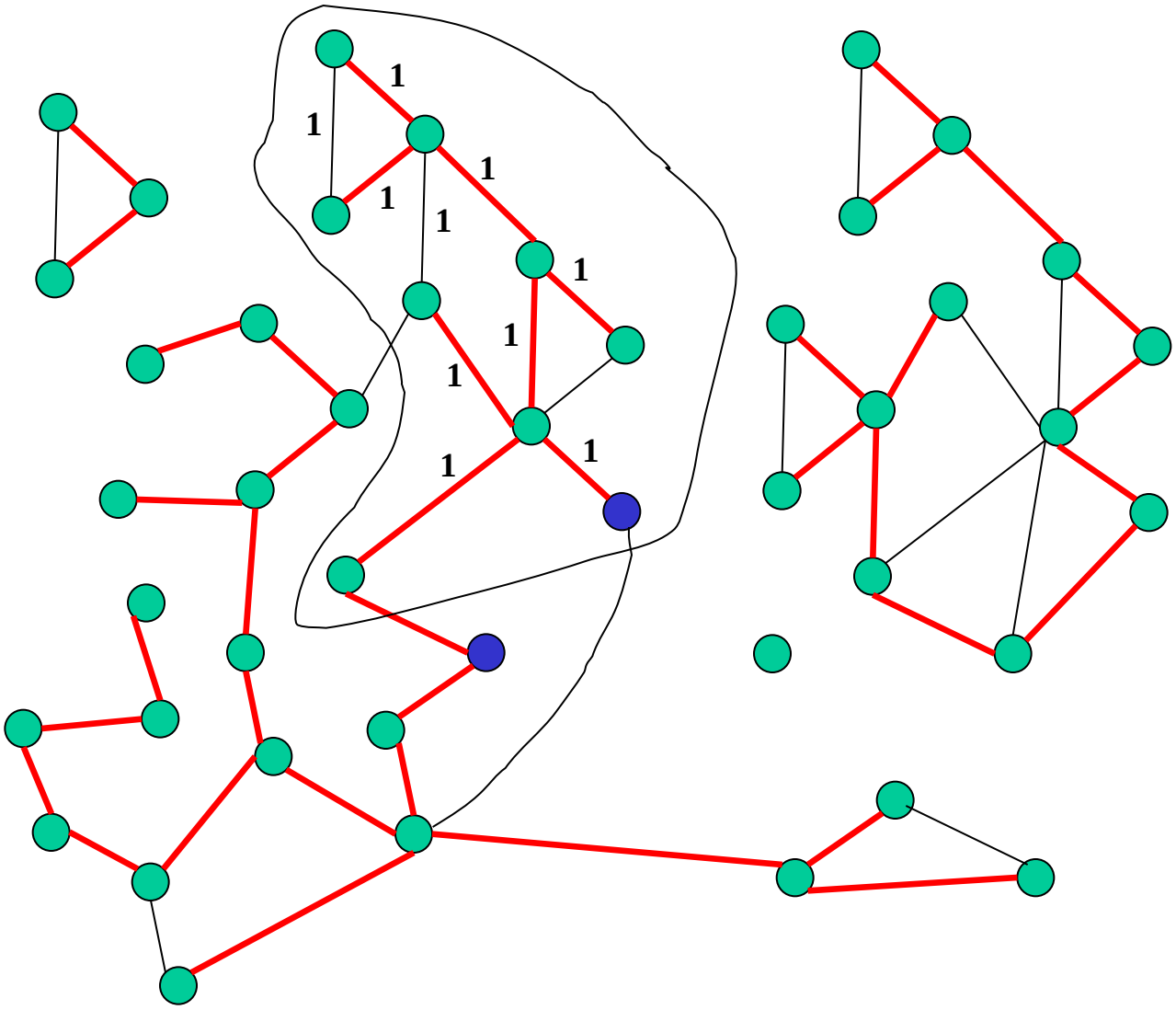

1
1
1
1
1
1
1
1
1
1

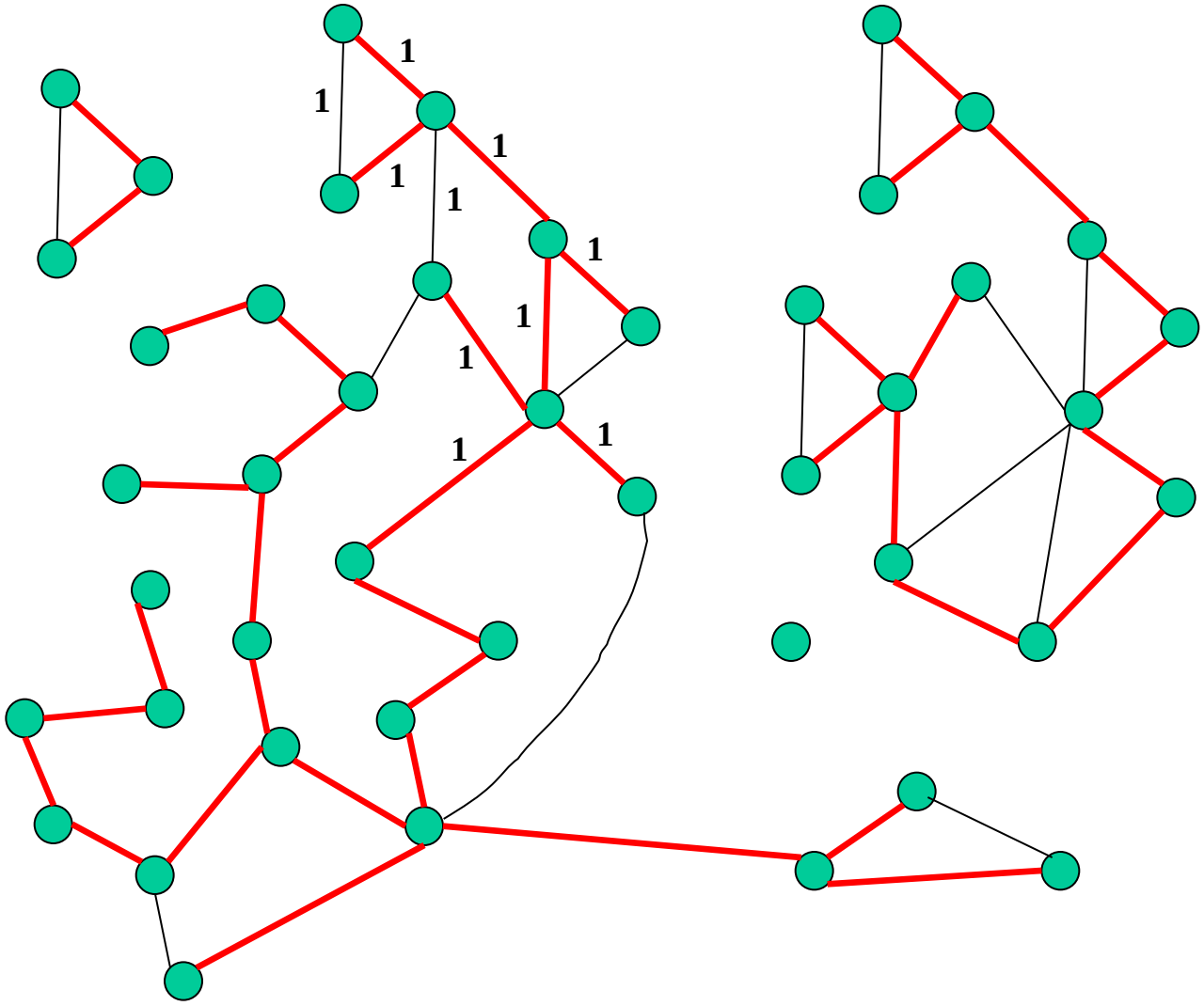

1
1
1
1
1
1
1
1
1
1

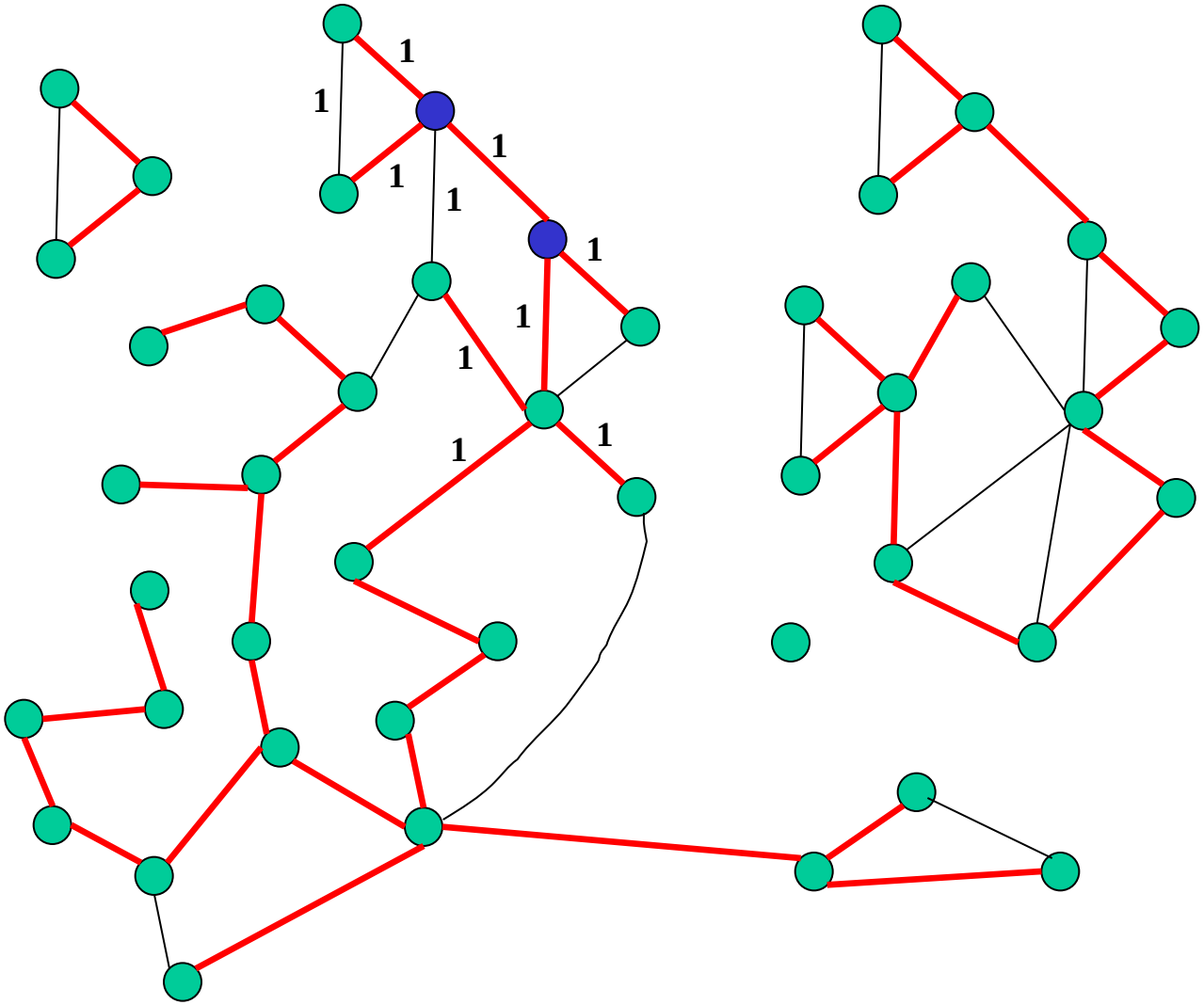

1
1
1
1
1
1
1
1
1
1

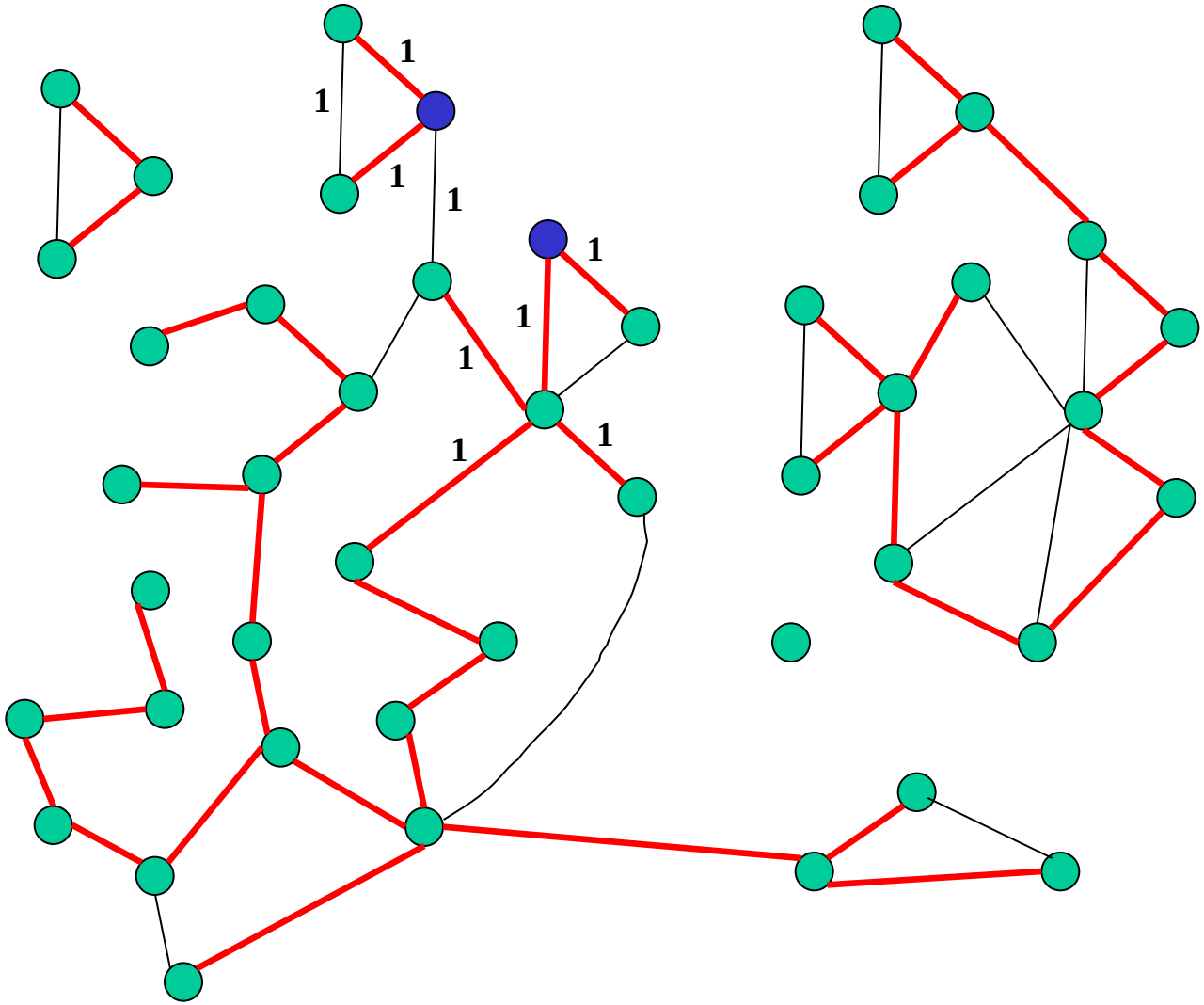


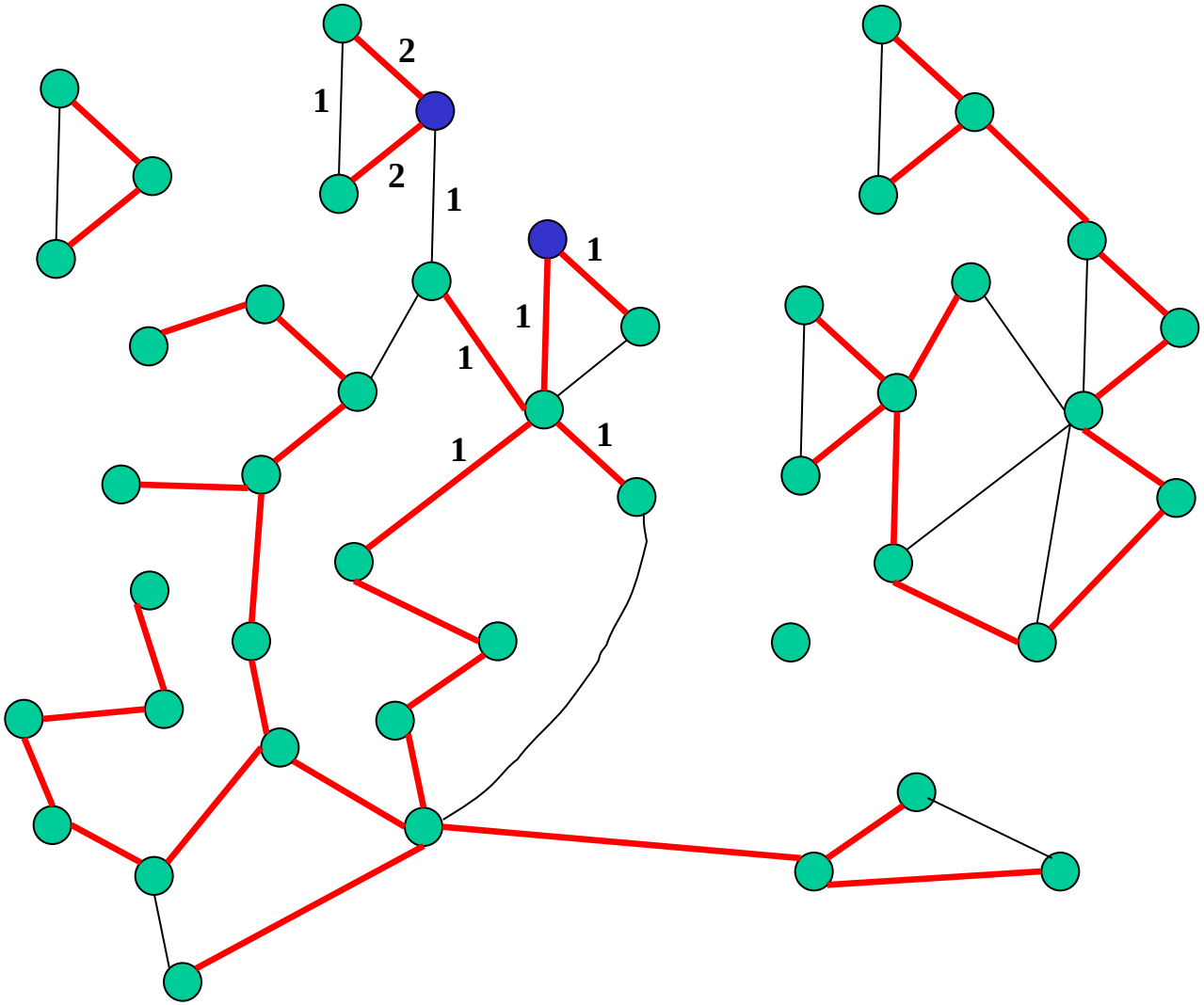
No need to look at non tree edges with label 1

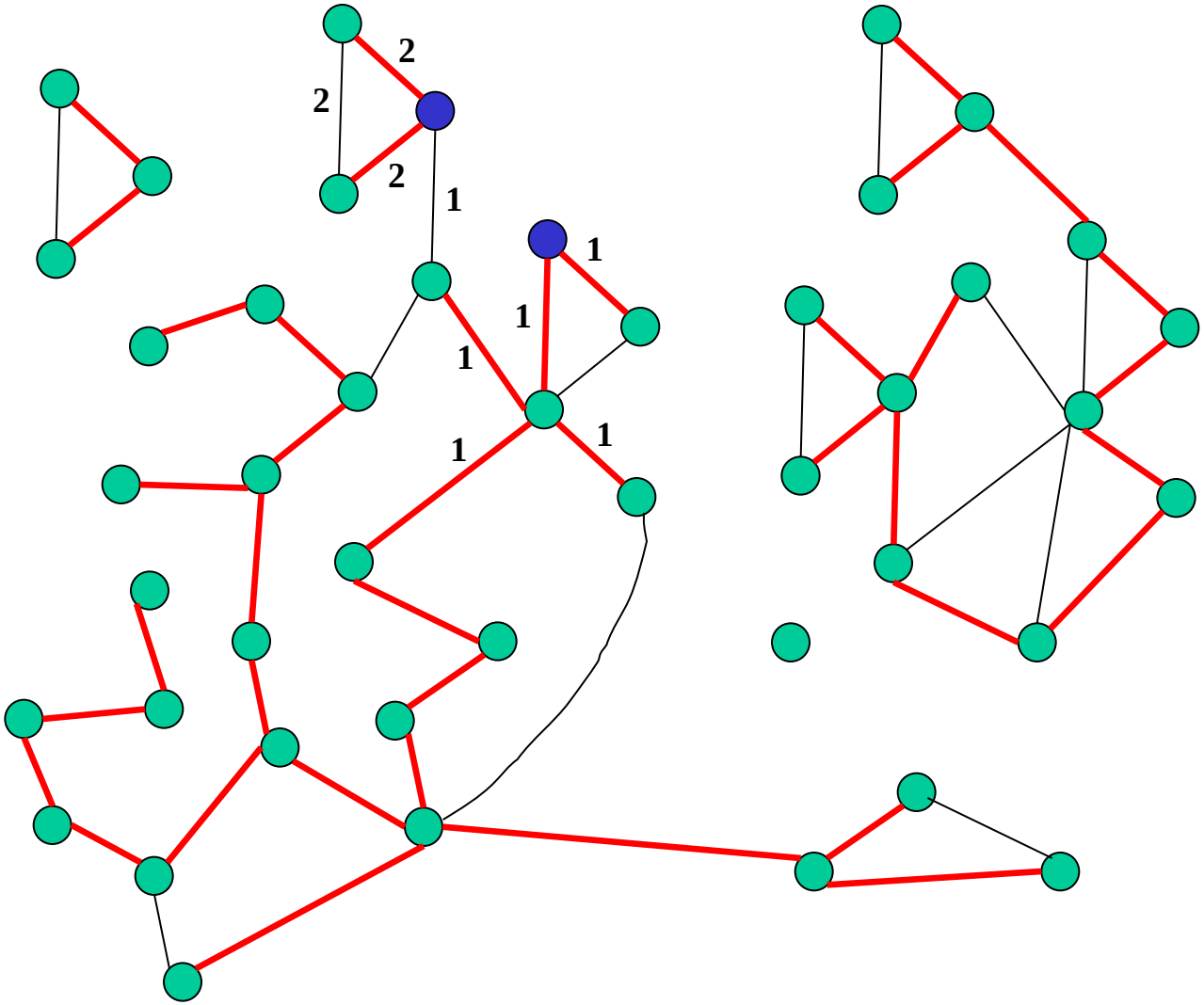

1
1
1
1
1
1
1
1
1
1
1

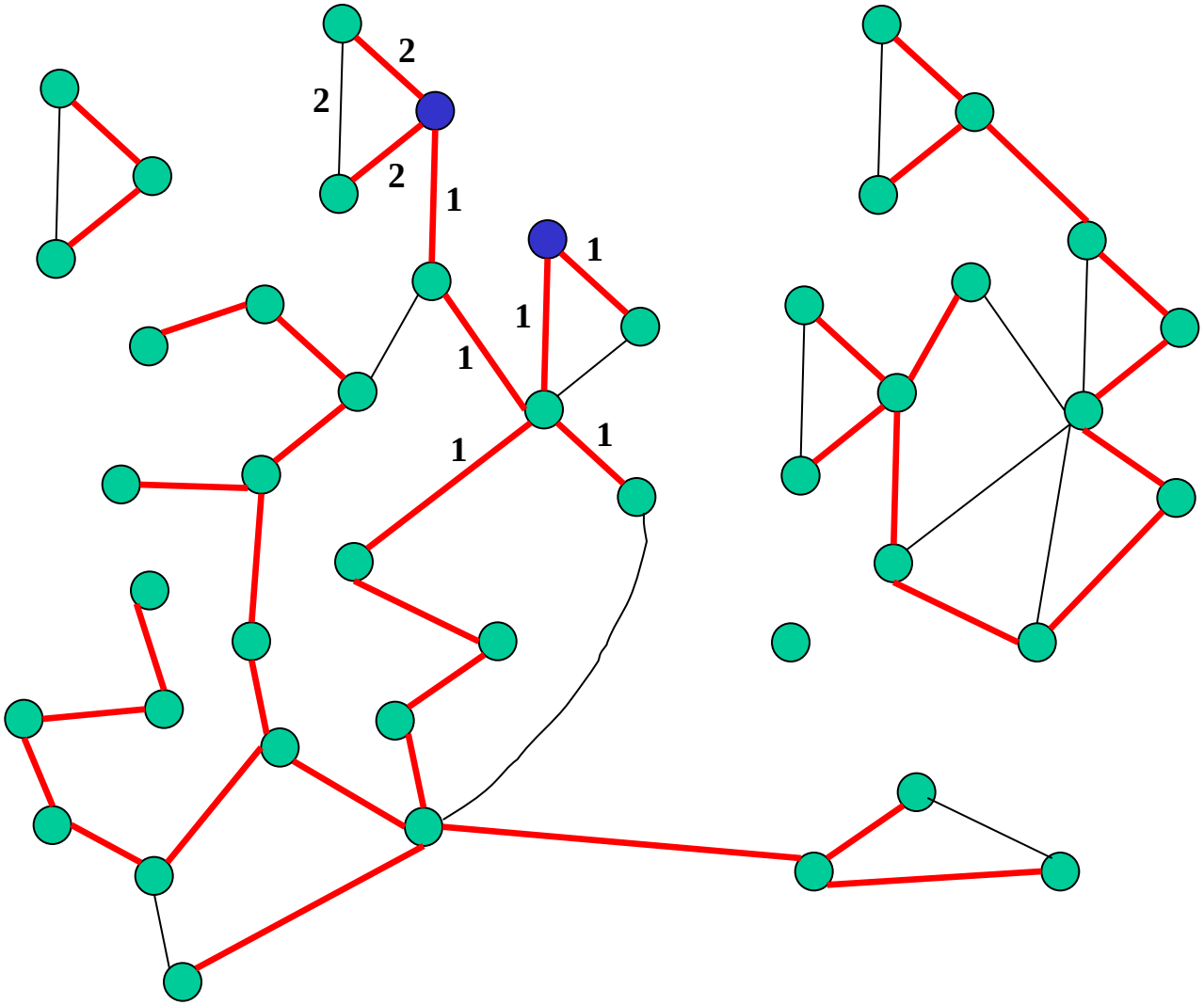

1
1
1
1
1
1
1
1
1
1
1

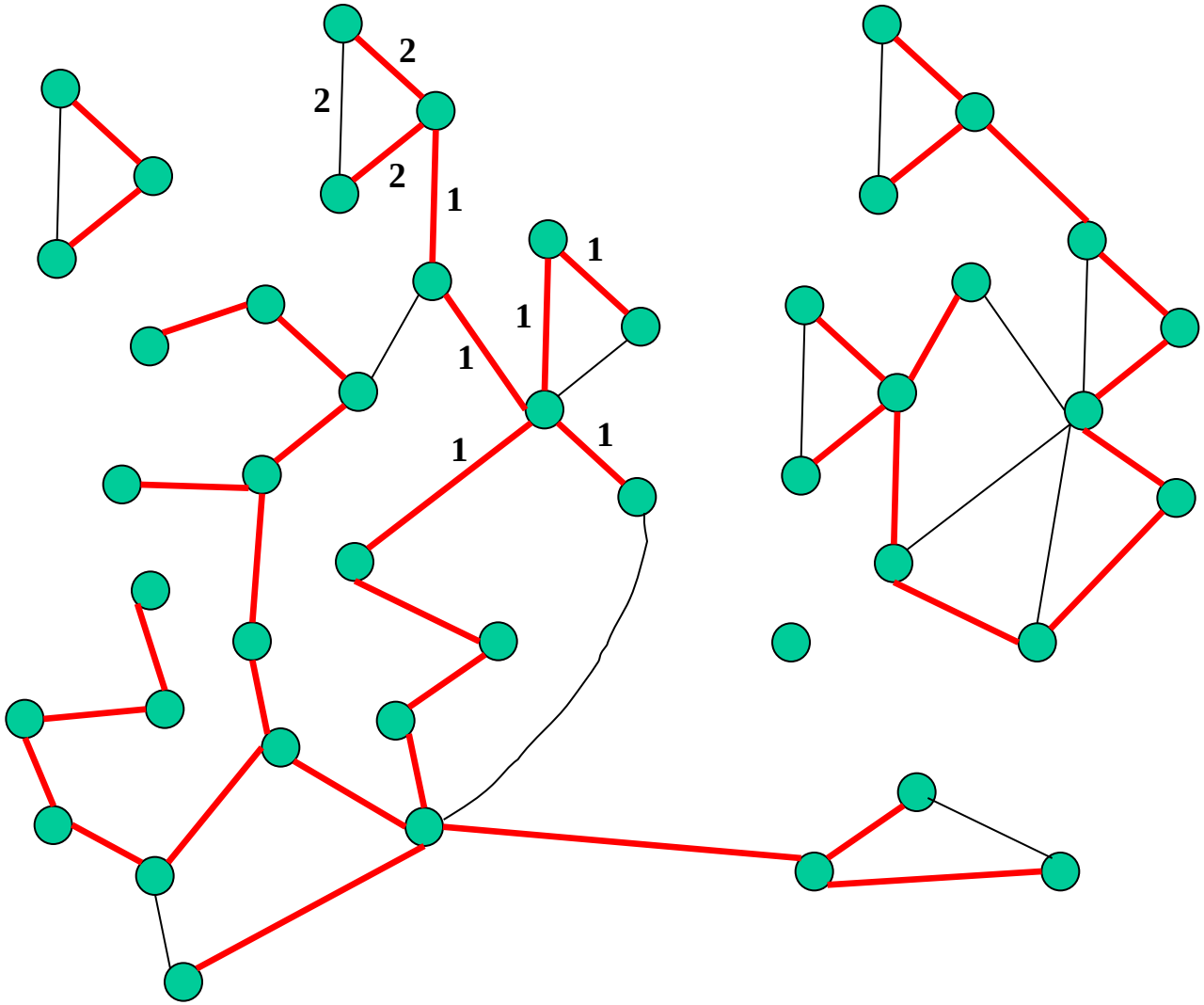

1
1
1
1
1
1
1
1
1
1
1


1
1
1
1
1
1
1
1
1


2
1
2
1
1
1
1
1
1


2
2
2
1
1
1
1
1
1


2
2
2
1
1
1
1
1
1
1


2
2
2
1
1
1
1
1
1

What is going on exactly ?

Invariants

- The forest is a maximum spanning forest with respect to the levels of the edges

➔ If we delete an edge at level ℓ then a replacement edge is of level $\leq \ell$

- Let F_i be the subforest of edges of level $\geq i$, then each tree in F_i is of size no larger than $n/2^i$

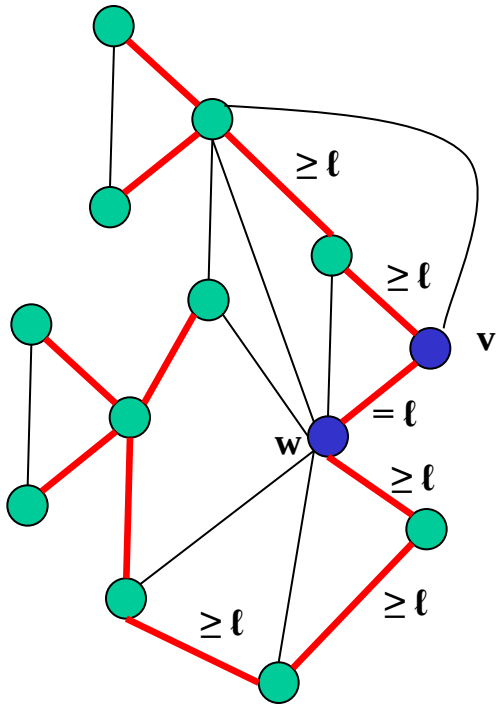
➔ No more than log n levels

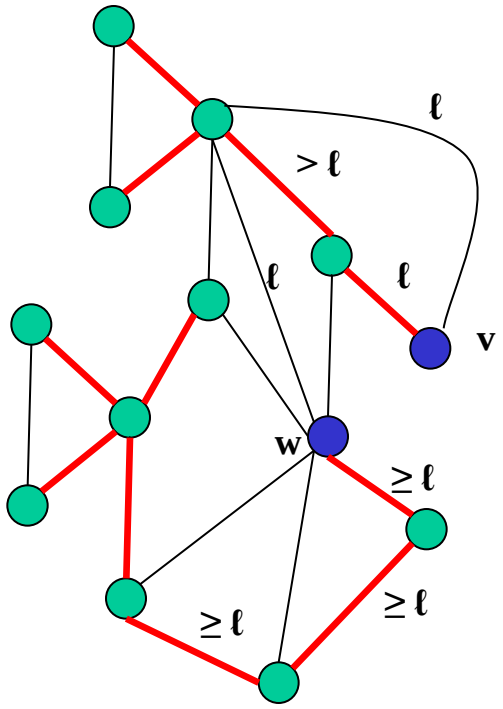
Suppose we delete a tree edge of level ℓ. Then it belongs to some tree T of F_ℓ

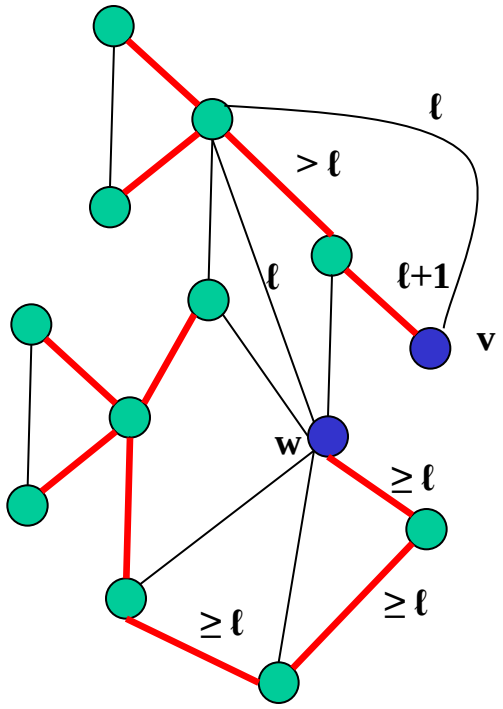
If there is a replacement at level ℓ then it is incident to one of the pieces of T

Let T_v and T_w be the pieces of T in F_ℓ containing v and w respectively after deleting (v,w)

Assume $|T_v| \leq |T_w|$ then we increase the level of edges of level ℓ in T_v to be $\ell+1$

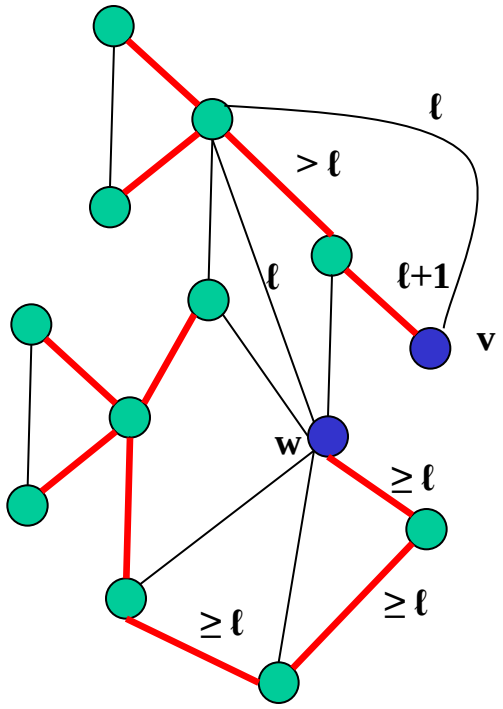



ℓ
> ℓ
ℓ
ℓ
v
w
≥ ℓ
≥ ℓ
≥ ℓ


ℓ
> ℓ
ℓ
ℓ+1
v
w
≥ ℓ
≥ ℓ
≥ ℓ

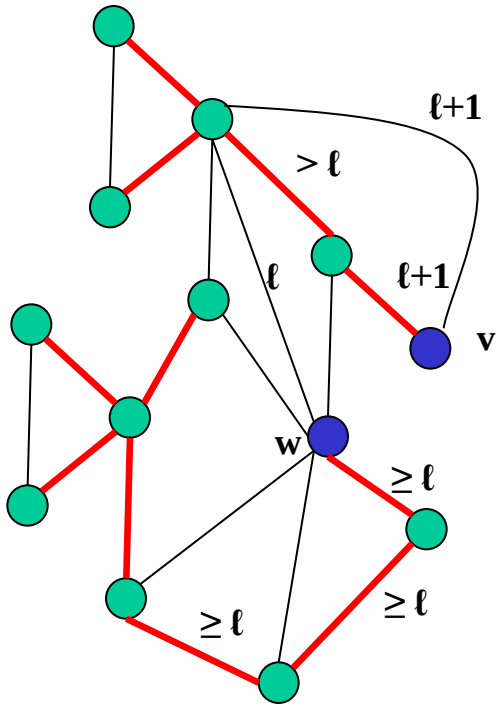
Then we traverse all level ℓ non-tree edges incident to T_v to find a level-ℓ replacement edge.

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



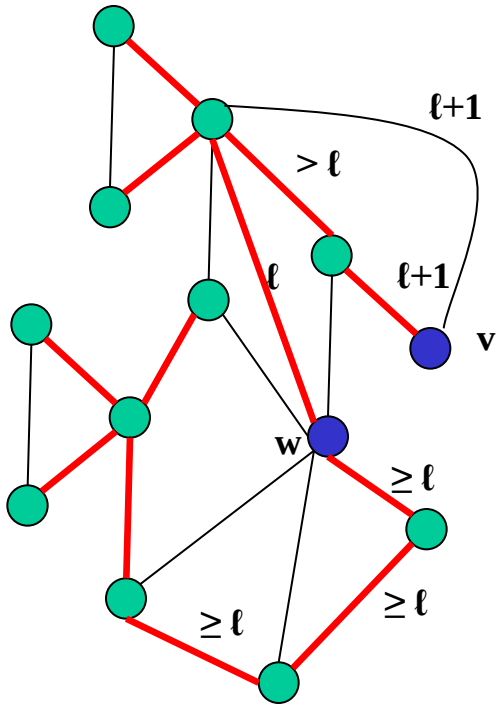
Then we traverse all level ℓ non-tree edges incident to T_v to find a level-ℓ replacement edge.

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



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

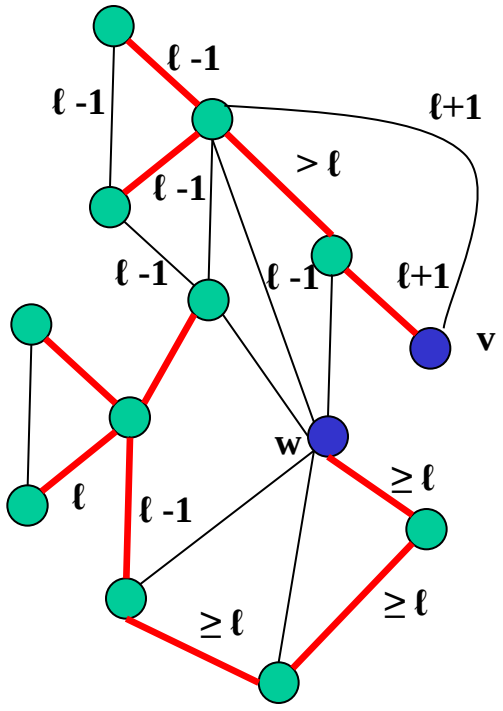
If a traversed edge is not a replacement we increase its level to $\ell+1$



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 none exists

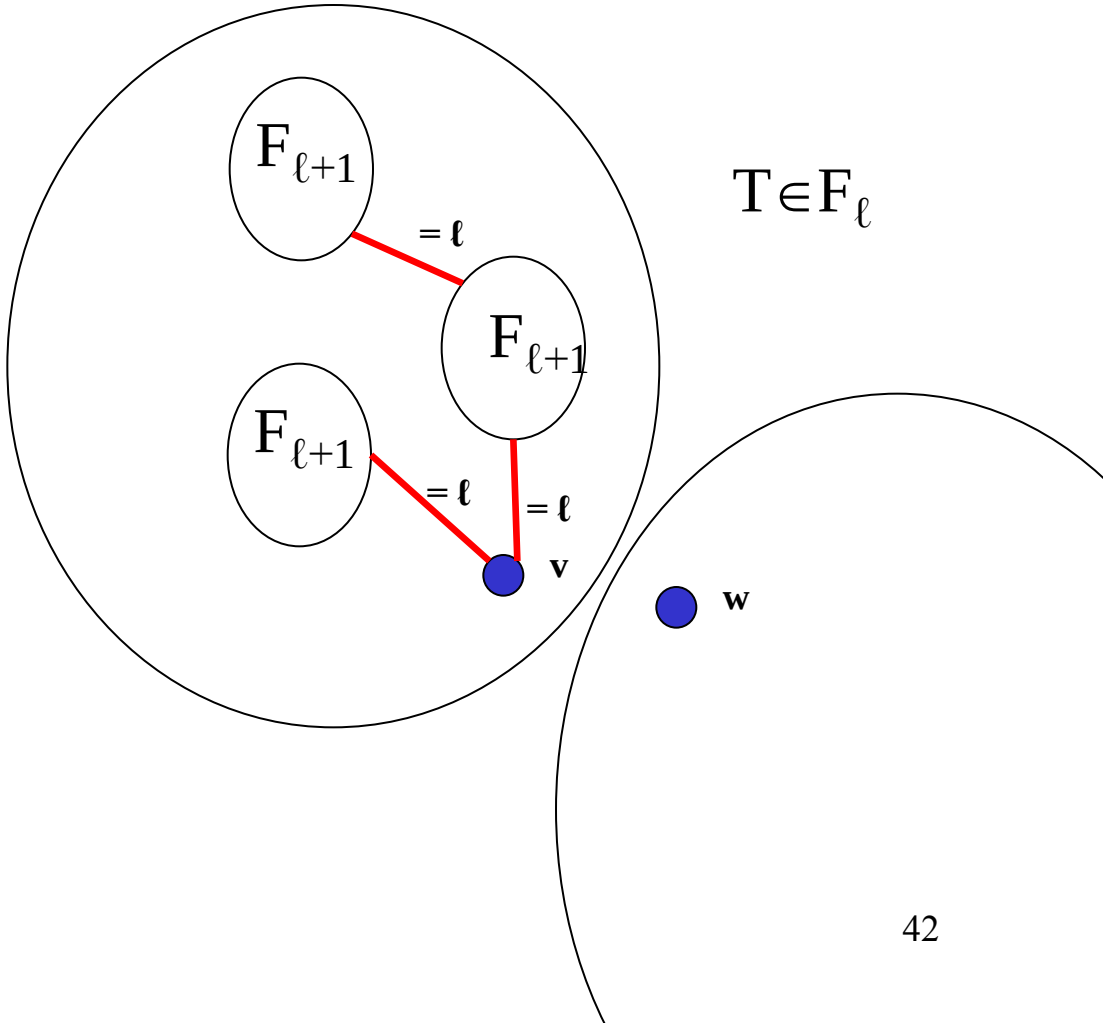
- As we go we keep our invariants

Why do we keep the second invariant ?

$|T| \leq n/2^{\ell}$ ➔

$|T_v| \leq n/2^{\ell+1}$

The replacement edge stays at level ℓ



Precise implementation

- We keep each forest $F_0 \supseteq F_1 \supseteq F_2 \supseteq$ separately
- The non-tree edges of level ℓ are kept with the nodes of F_ℓ

Complete definition of the operations

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

connected(v,w) : Check if v and w are in the same tree of F_0

Delete(v,w): Let ℓ be the level of (v,w).

• If (v,w) is a non-tree edge of level ℓ then simply delete it from v and w in F_ℓ.

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

Operations we need to do on the forests

link(v,w) : assume v and w are in different trees

cut(v,w) : assume v and w are adjacent in a tree

findtree(v)

Find an edge of level ℓ in $T \in F_\ell$

Find a non-tree edge of level ℓ incident with $T \in F_\ell$

Add/delete a non-tree edge of level ℓ incident with some $v \in T \in F_\ell$

Suppose we can do it in O(log n) time per operation

Analysis

- Insert takes O(log n) time + the time to increase the level of the edge. Each increase costs O(log n) so it is $O(\log^2 n)$ total.
- Query takes O(log n)
- Delete takes $O(\log^2 n)$

How do we do these operations in O(log(n)) time ?

link(v,w) : assume v and w are in different trees

cut(v,w) : assume v and w are adjacent in a tree

findtree(v)

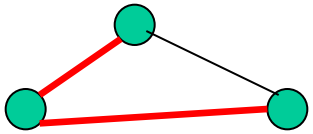
Find an edge of level ℓ in $T \in F_\ell$

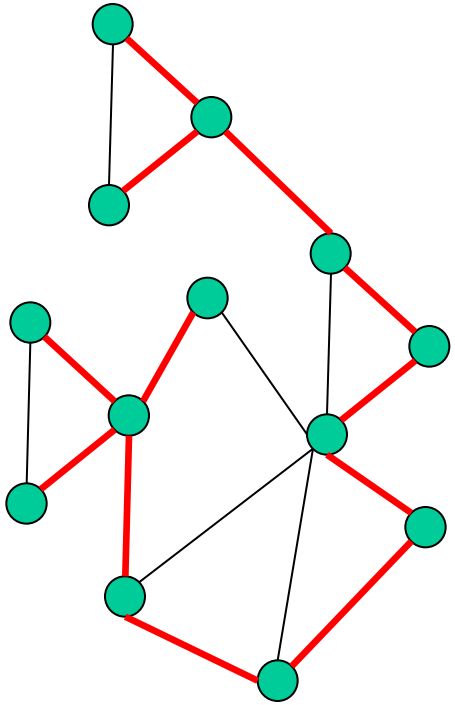
Find a non-tree edge of level ℓ incident with $T \in F_\ell$

Add/delete a non-tree edge of level ℓ incident with some $v \in T \in F_\ell$

Can modify dynamic trees to support these, but there is a simpler way:

Euler tour trees





Maintain a forest under the operations:

link(v w)

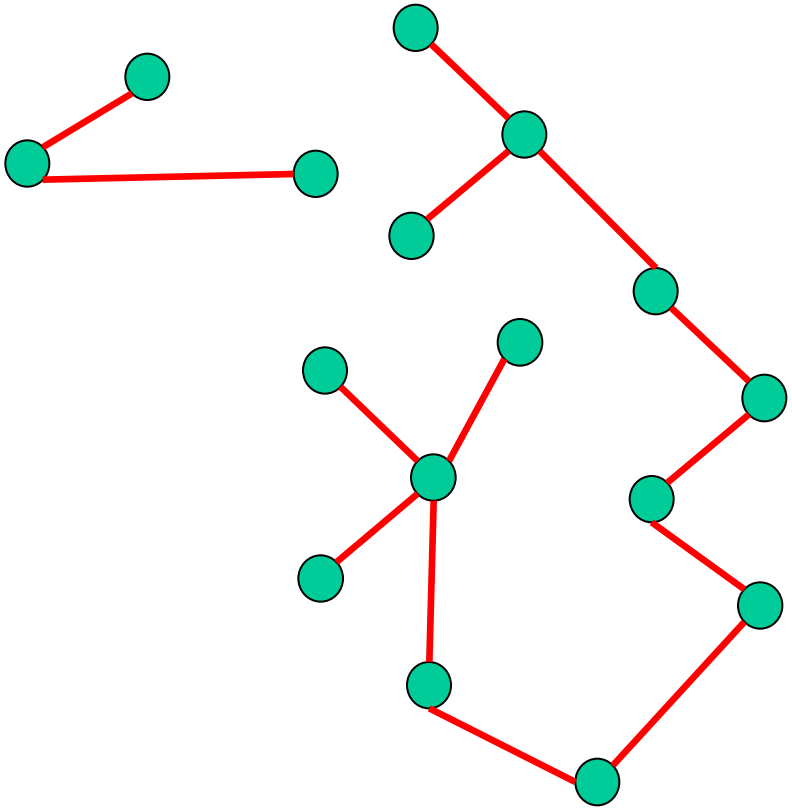
cut(v,w)

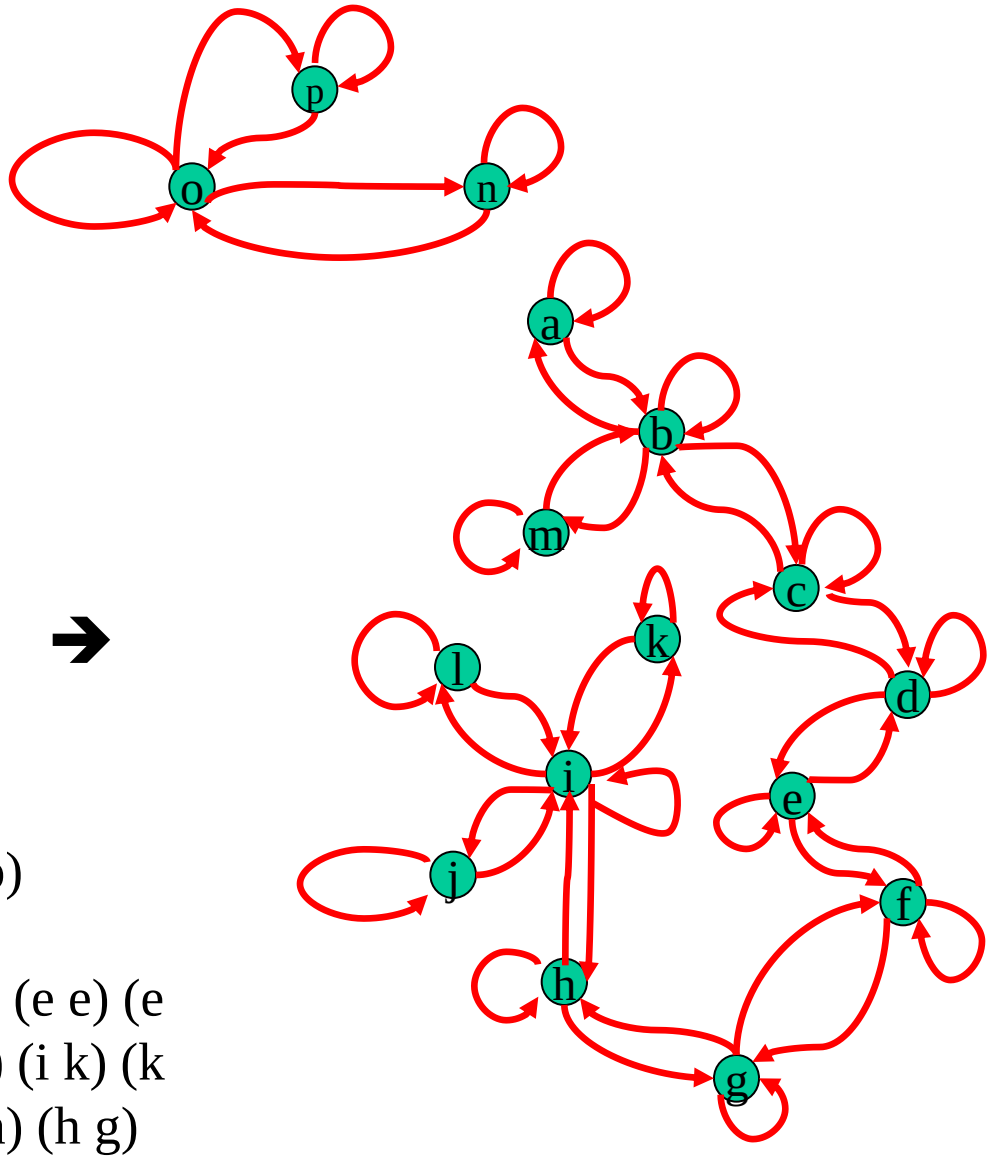
find-tree(v)

find-min-val(T)

change-val(v,x)

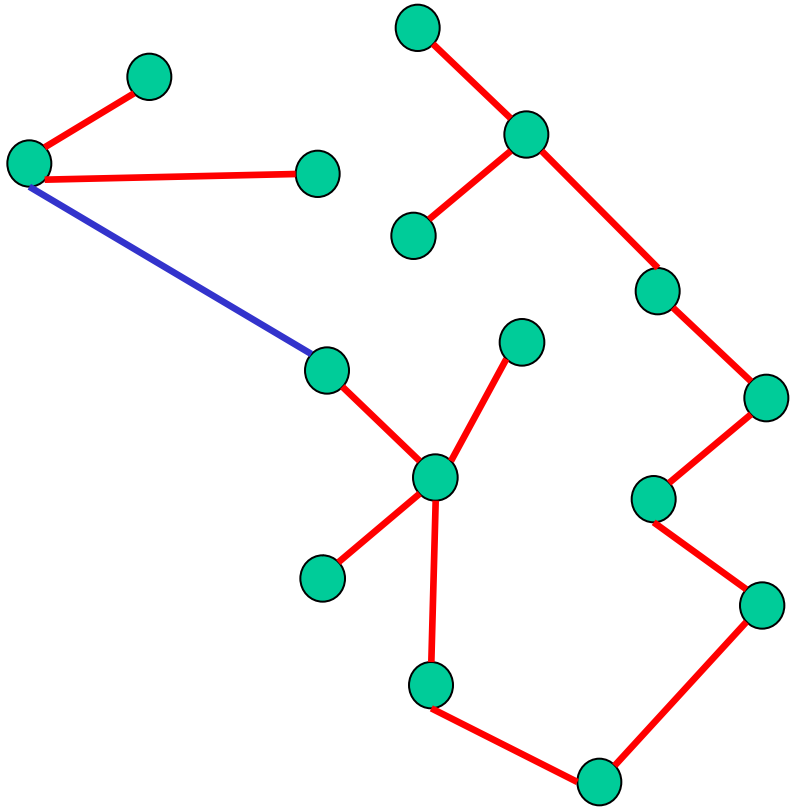
add-val(T,x)





(p o) (o n) (n n) (n o) (o o) (o p) (p p)

(a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h i) (i i) (i k) (k k) (k i) (i l) (l l) (l i) (i j) (j j) (j i) (i h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a)



(p o) (o n) (n n) (n o) (o o) (o p) (p p)

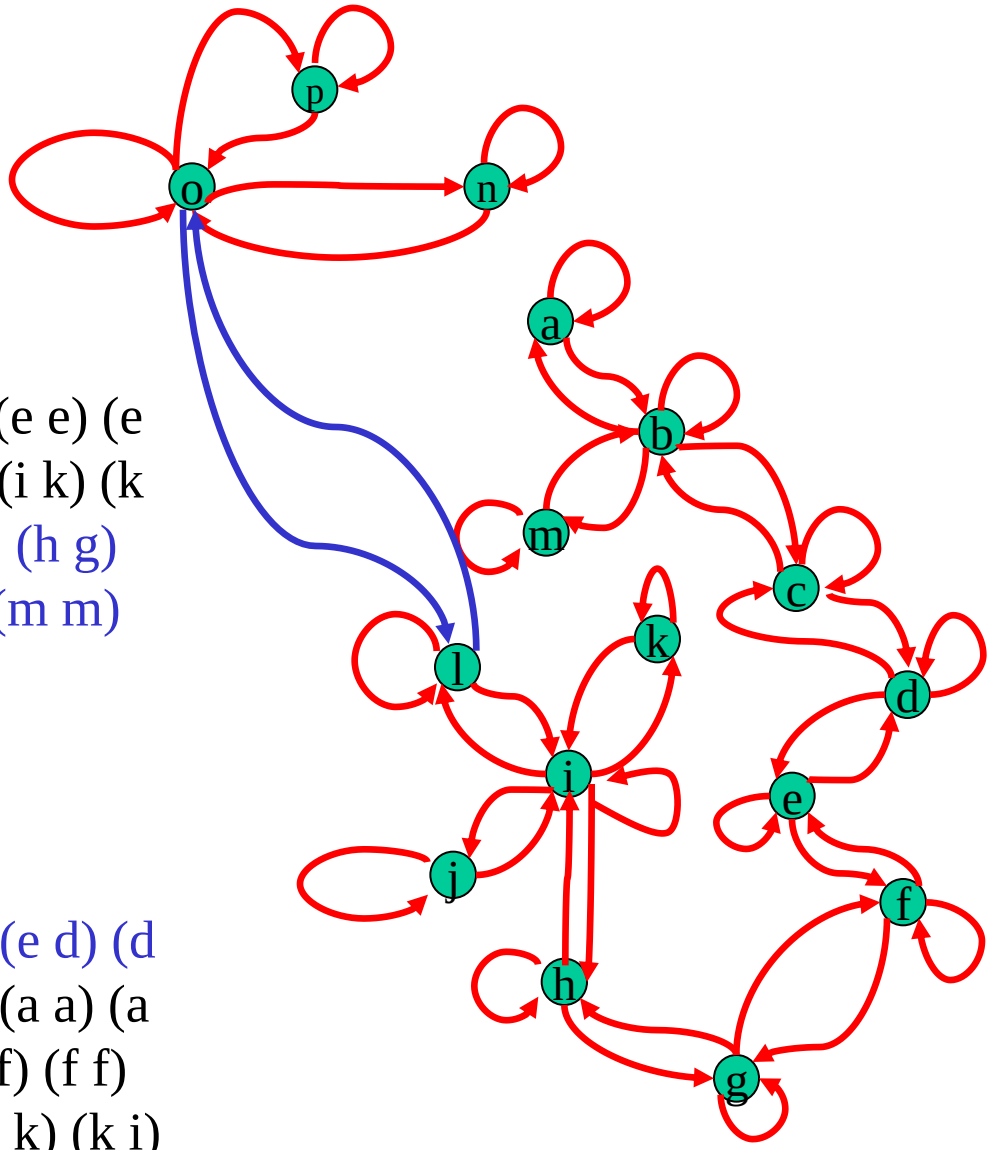
(a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h i) (i i) (i k) (k k) (k i) (i l) (l l) (l i) (i j) (j j) (j i) (i h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a)

(p o) (o n) (n n) (n o) (o o) (o p) (p p)

(a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h i) (i i) (i k) (k k) (k i) (i l) (l l) (l i) (i j) (j j) (j i) (i h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a)

(o p) (p p) (p o) (o n) (n n) (n o) (o o)

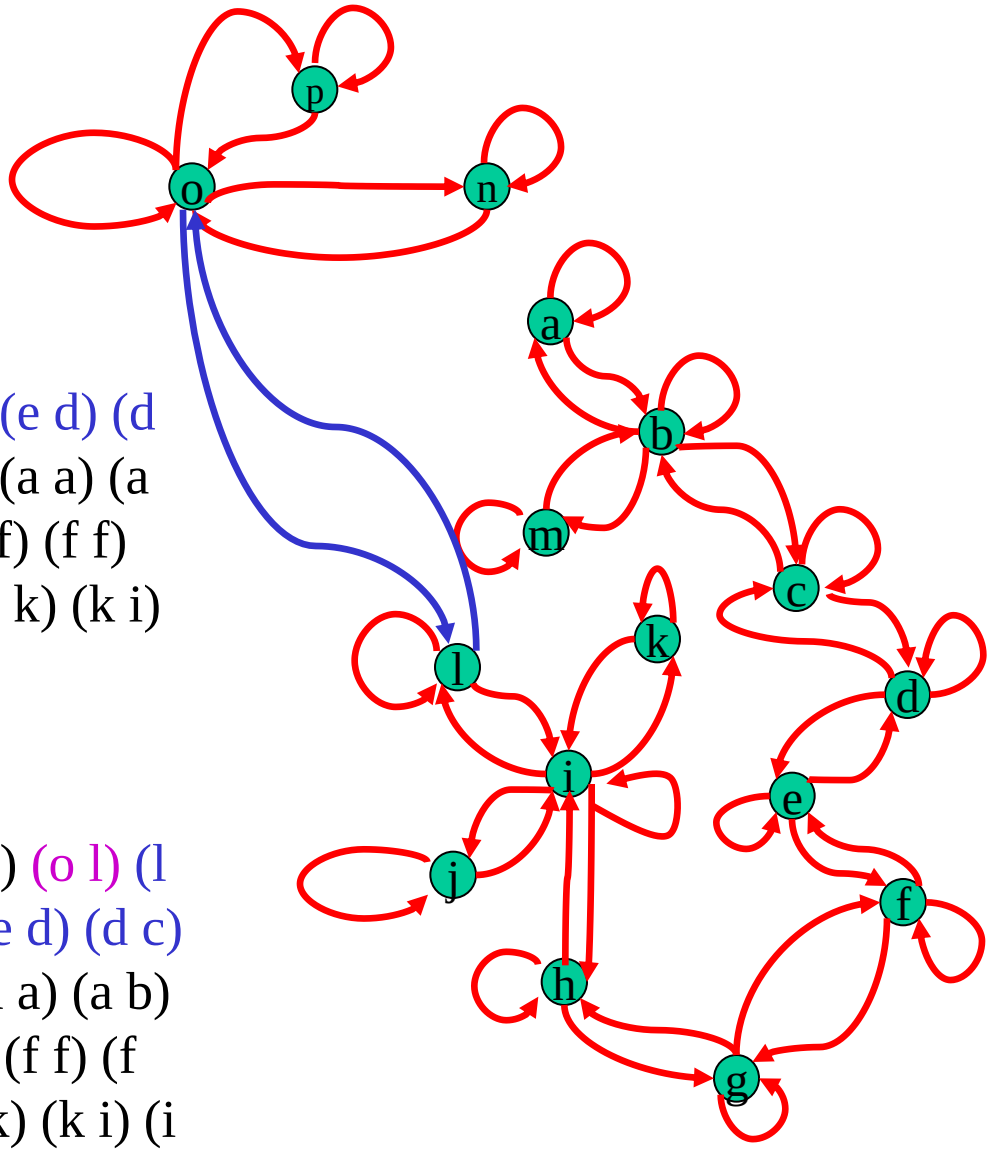
(l i) (i j) (j j) (j i) (i h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a) (a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h i) (i i) (i k) (k k) (k i) (i l) (l l)

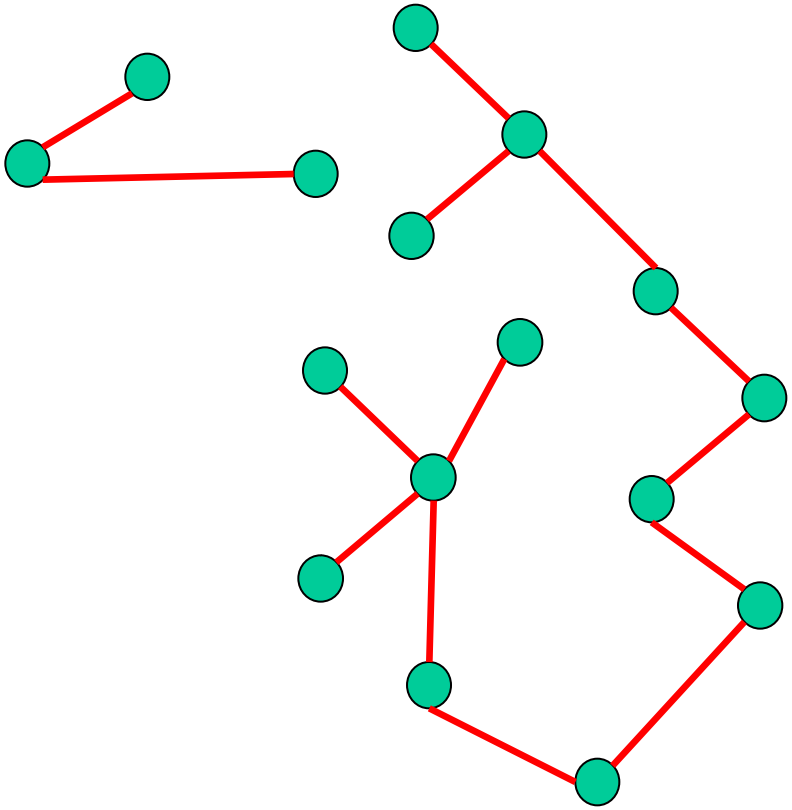


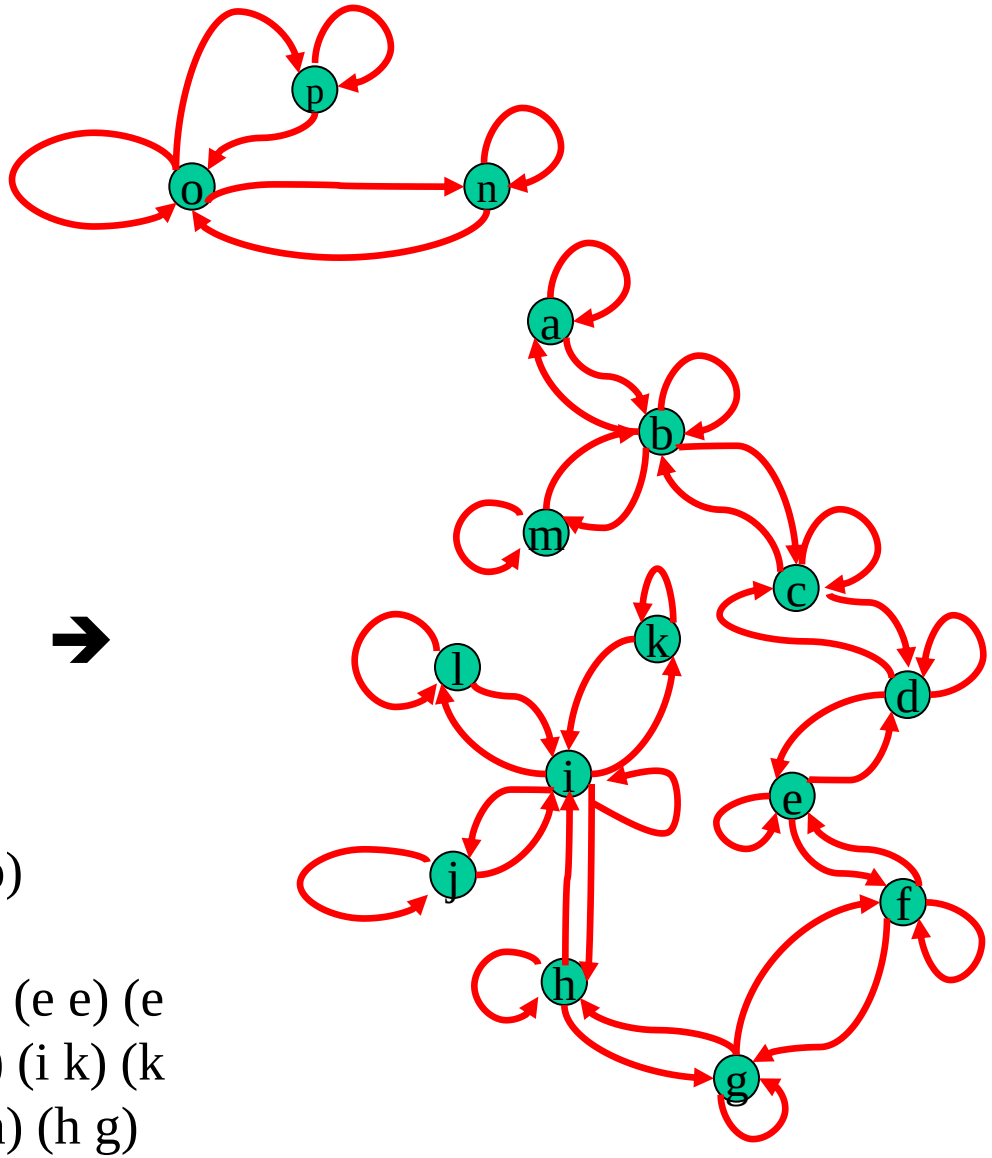
(o p) (p p) (p o) (o n) (n n) (n o) (o o)

(l i) (i j) (j j) (j i) (i h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a) (a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h i) (i i) (i k) (k k) (k i) (i l) (l l)

(o p) (p p) (p o) (o n) (n n) (n o) (o o) (o l) (l i) (i j) (j j) (j i) (i h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a) (a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h i) (i i) (i k) (k k) (k i) (i l) (l l) (l o)

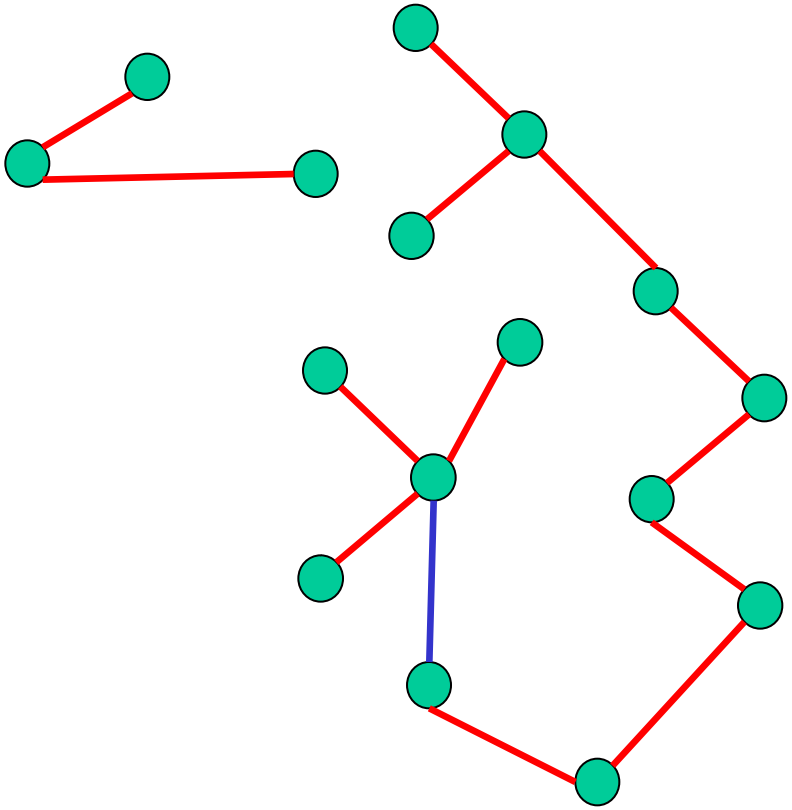


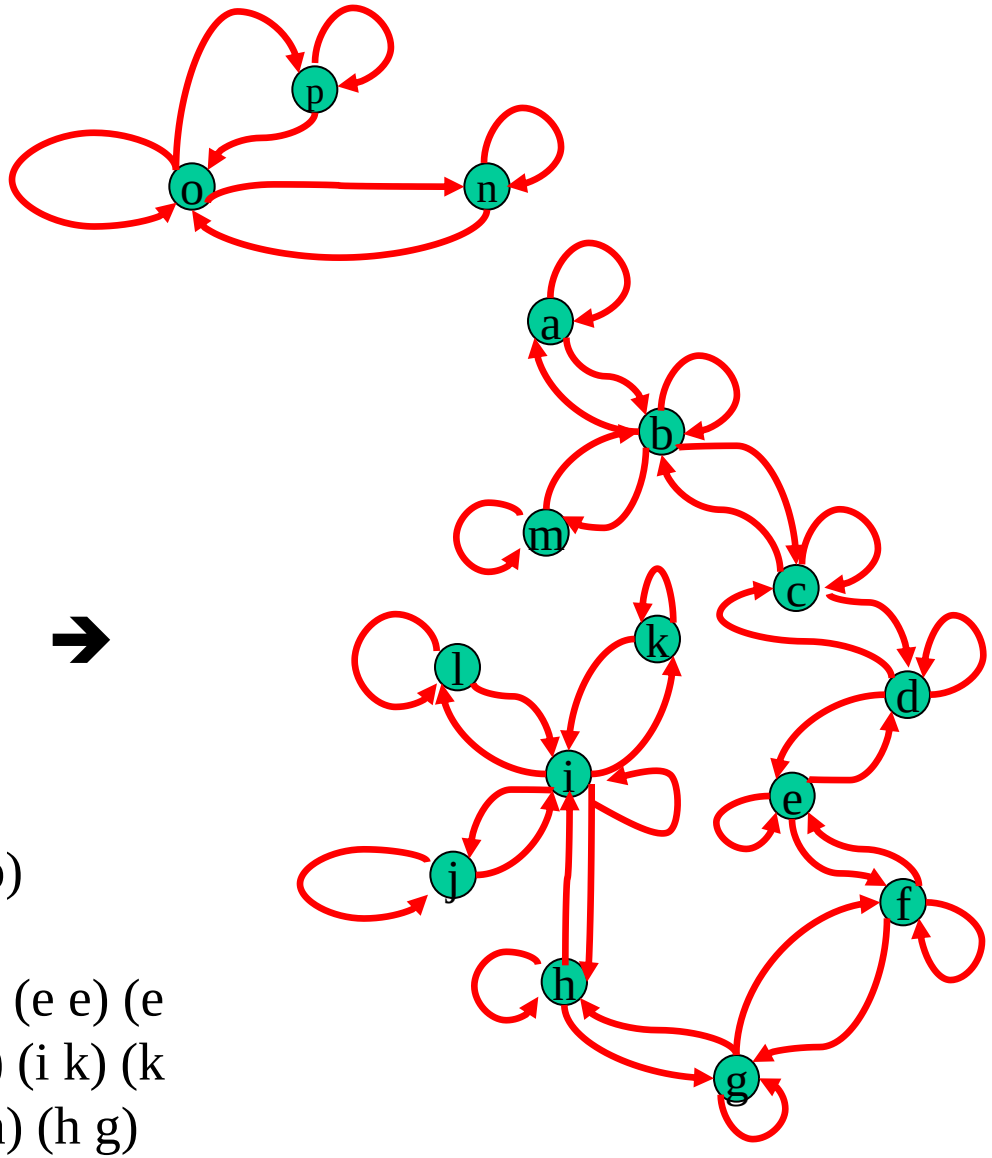




(p o) (o n) (n n) (n o) (o o) (o p) (p p)

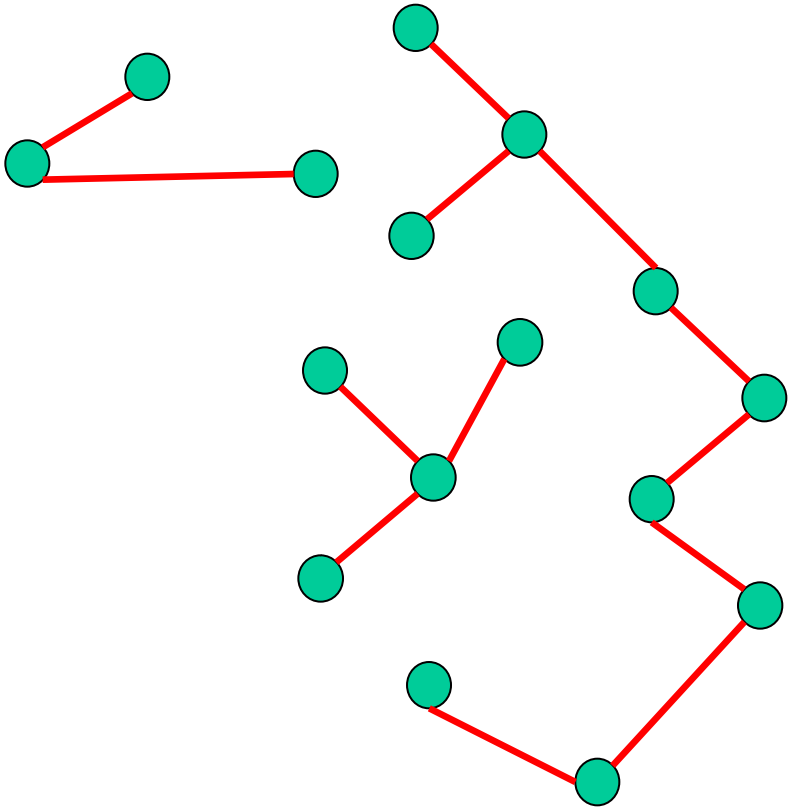
(a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h i) (i i) (i k) (k k) (k i) (i l) (l l) (l i) (i j) (j j) (j i) (i h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a)

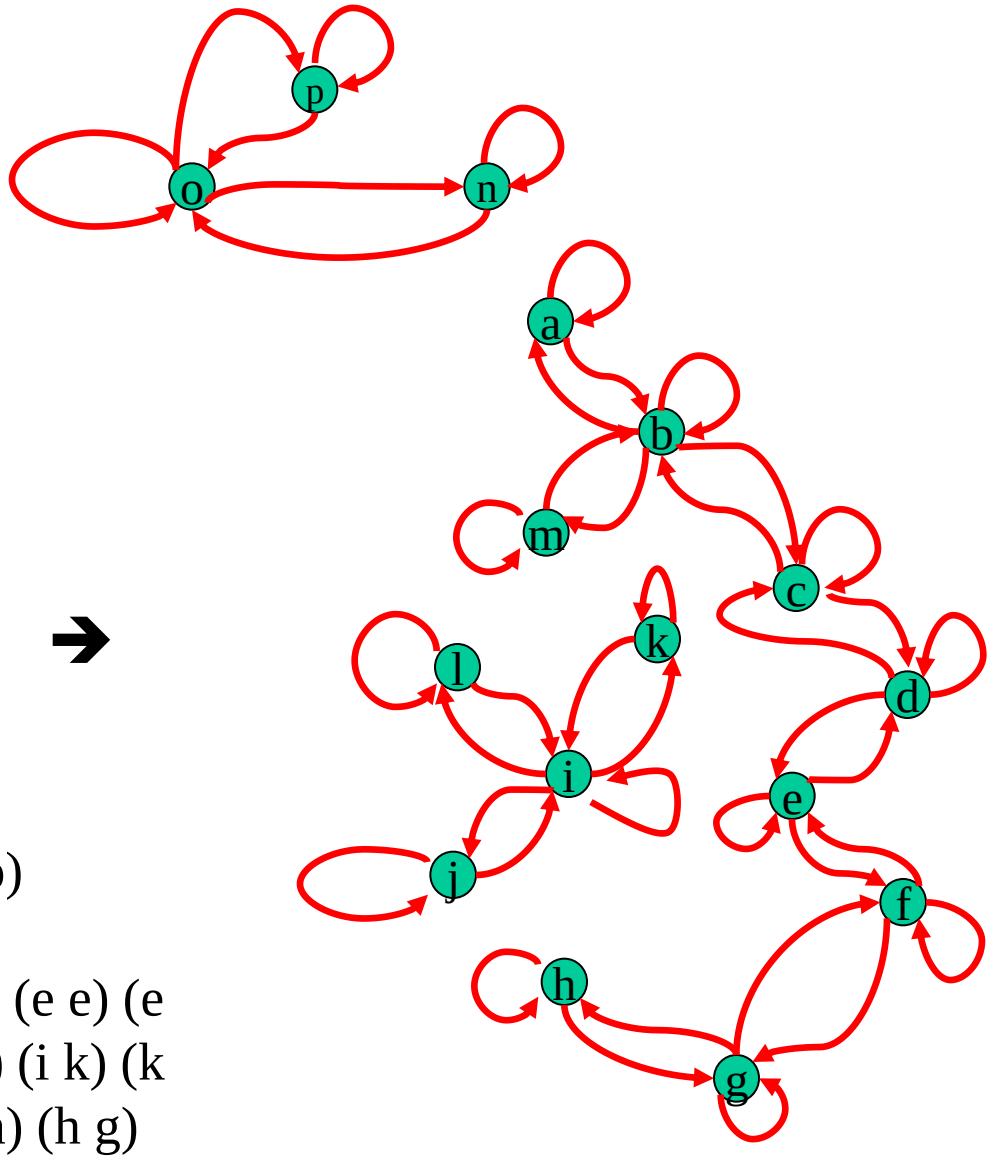




(p o) (o n) (n n) (n o) (o o) (o p) (p p)

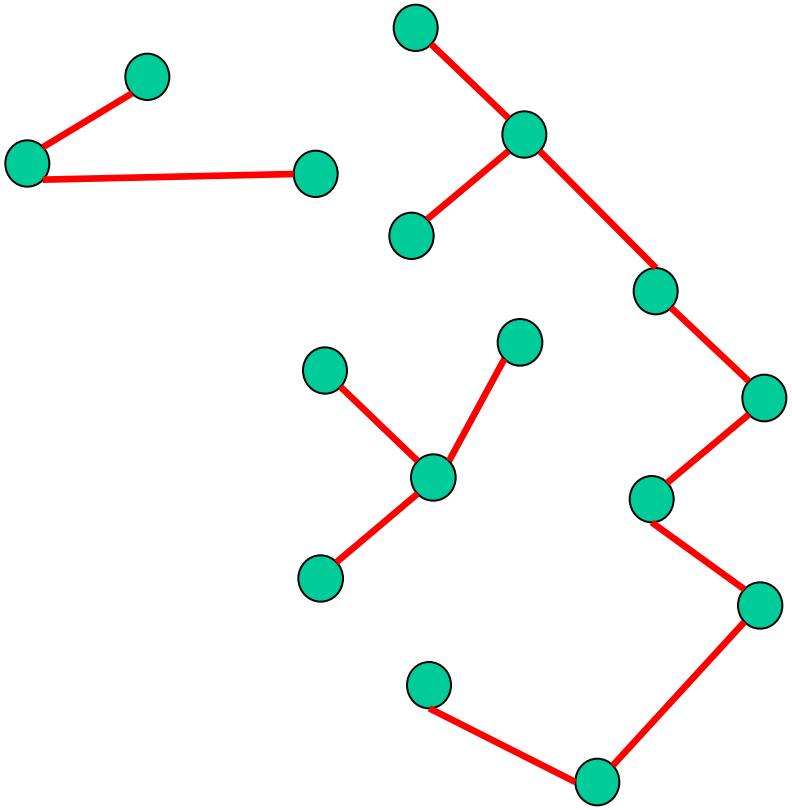
(a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h i) (i i) (i k) (k k) (k i) (i l) (l l) (l i) (i j) (j j) (j i) (i h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a)

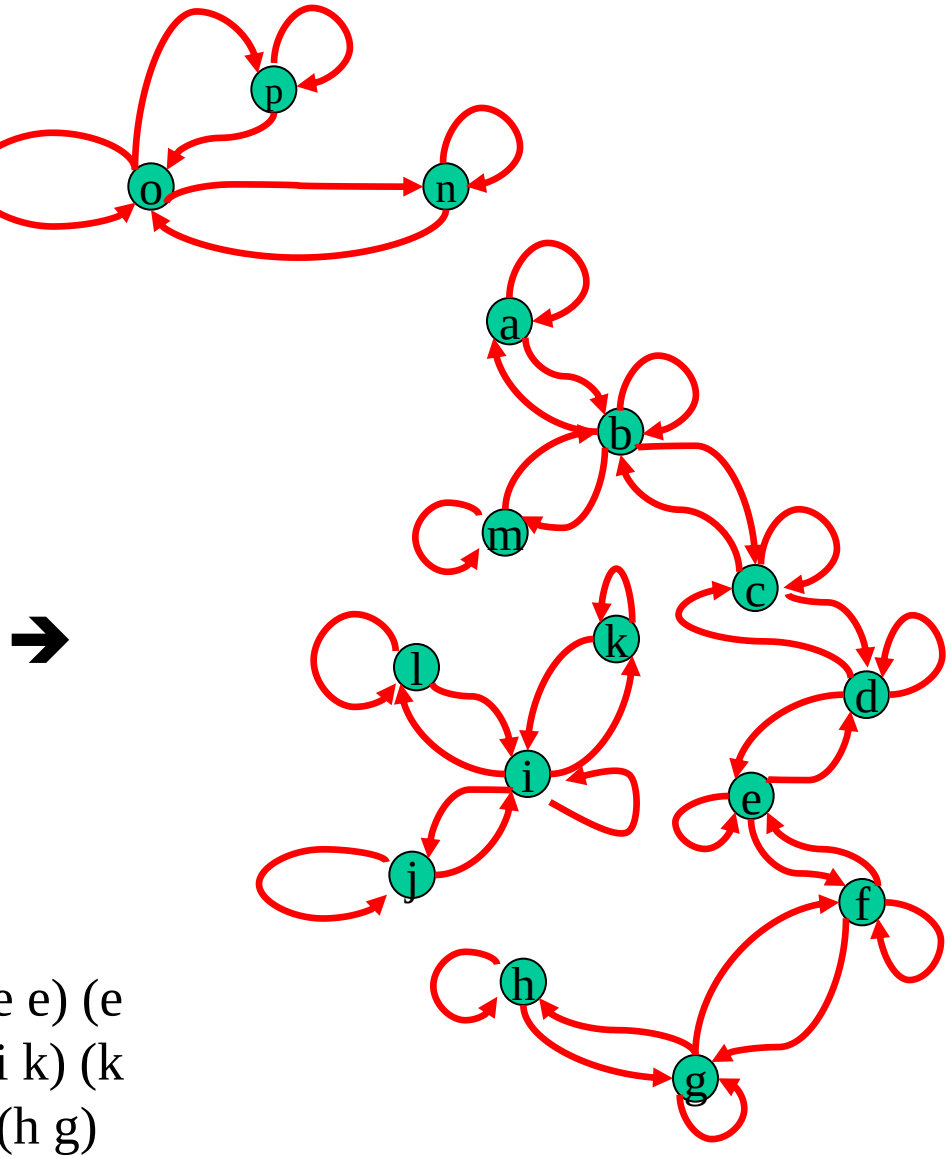




(p o) (o n) (n n) (n o) (o o) (o p) (p p)

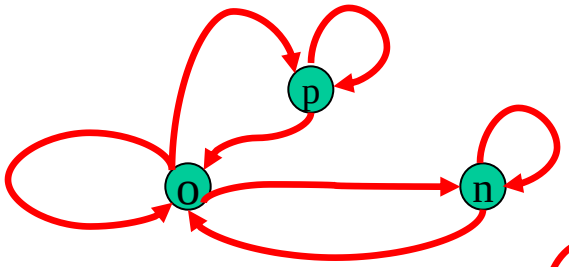
(a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h i) (i i) (i k) (k k) (k i) (i l) (l l) (l i) (i j) (j j) (j i) (i h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a)





(p o) (o n) (n n) (n o) (o o) (o p) (p p)

(a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h i) (i i) (i k) (k k) (k i) (i l) (l l) (l i) (i j) (j j) (j i) (i h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a)

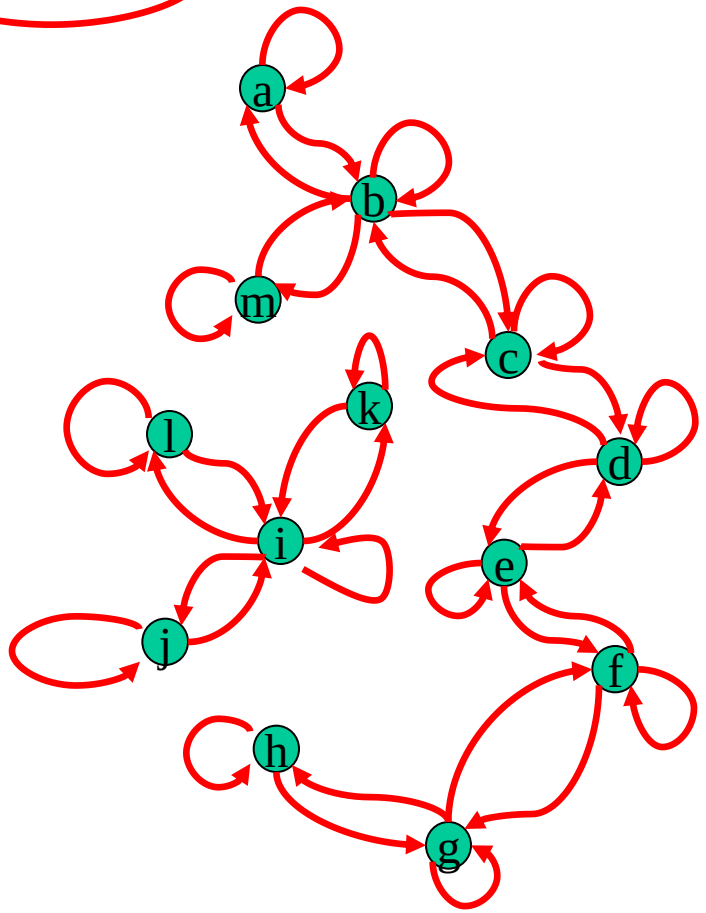


(p o) (o n) (n n) (n o) (o o) (o p) (p p)

(a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h i) (i i) (i k) (k k) (k i) (i l) (l l) (l i) (i j) (j j) (j i) (i h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a)

(a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a)

(i i) (i k) (k k) (k i) (i l) (l l) (l i) (i j) (j j) (j i)



Link + cut

- Linked lists would do.

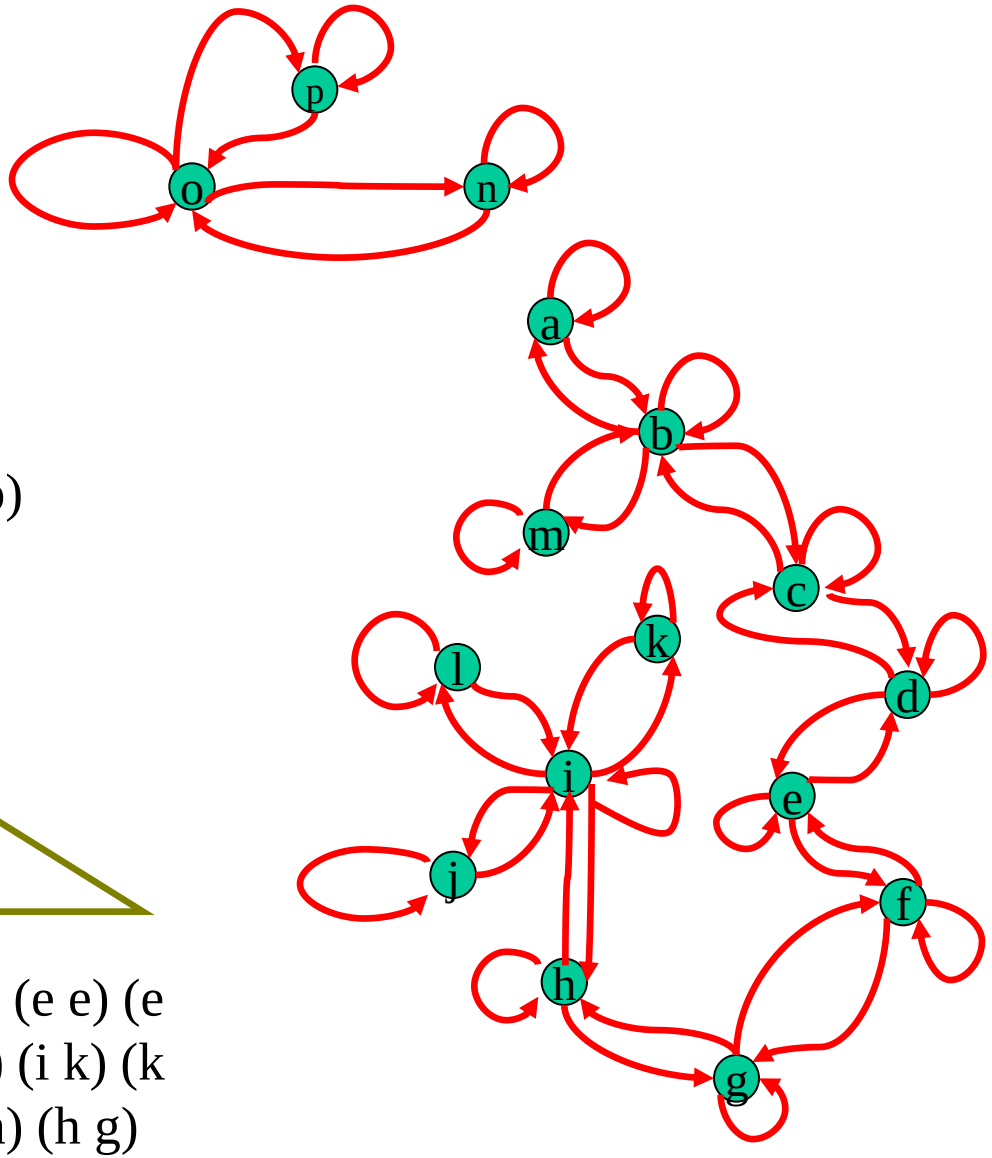
- What about finding the list containing a vertex ?
- What about vertex values ?

Search trees

- Represent the lists as search trees.

(p o) (o n) (n n) (n o) (o o) (o p) (p p)

(a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h i) (i i) (i k) (k k) (k i) (i l) (l l) (l i) (i j) (j j) (j i) (i h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a)



So we can easily do

link(v,w) : assume v and w are in different trees

cut(v,w) : assume v and w are adjacent in a tree

findtree(v)

In logarithmic time

What about vertex values ?

Store with each node the minimum value in its subtree

Maintain MSF under edge deletions

- The initial forest should be an MSF

- Traverse the edges incident to T_v looking for a replacement in order of increasing weights

This maintains an additional invariant

- Every cycle C has a non-tree edge which is of maximum weight and of lowest level

Proof:

- Deletions of edges keep this
-

This maintains an additional invariant

- Every cycle C has a non-tree edge which is of maximum weight and of lowest level

Proof:

- Deletions of edges keep this
- What happens when we increase the level of an edge e or insert e into the tree ?

This maintains an additional invariant

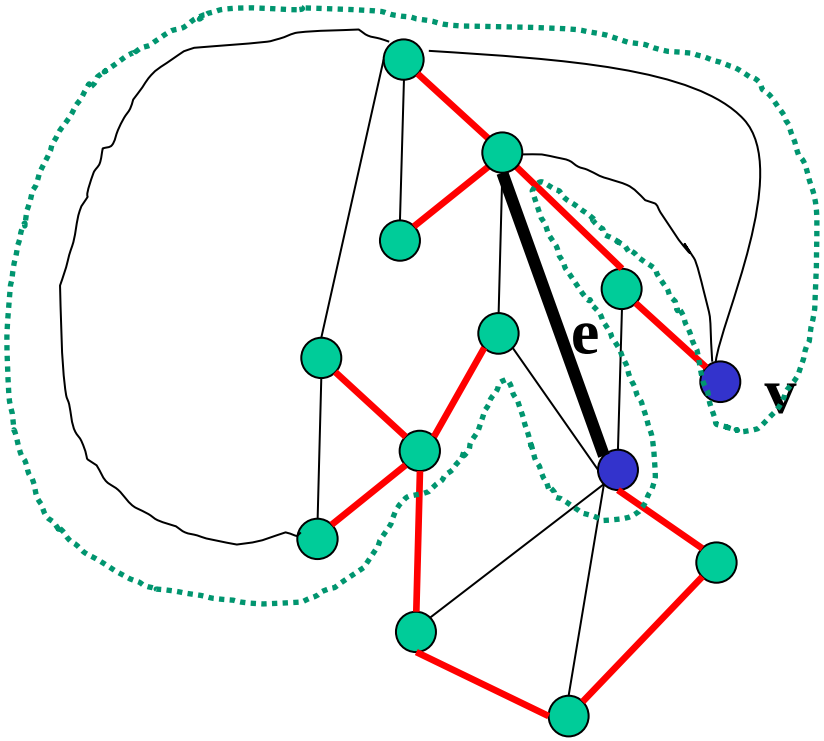
- Every cycle C has a non-tree edge which is of maximum weight and of lowest level

Proof:

- Deletions of edges keep this
- What happens when we increase the level of an edge e or insert e into the tree ?

Assume e is the unique lowest heaviest nontree edge on some cycle C. Let i be the level of e.

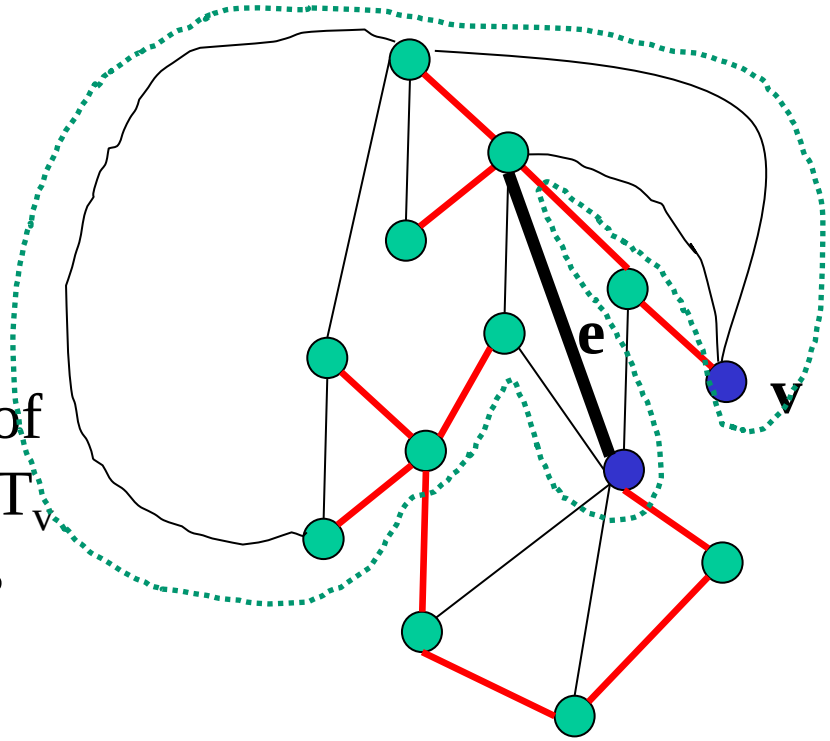
The lowest heaviest nontree edge invariant



Since we inspect lighter edges first then edges of C lighter than e incident to T_v are of level > i

e is unique lowest-heaviest so edges on C as heavy as e are of level > i

The lowest heaviest nontree edge invariant

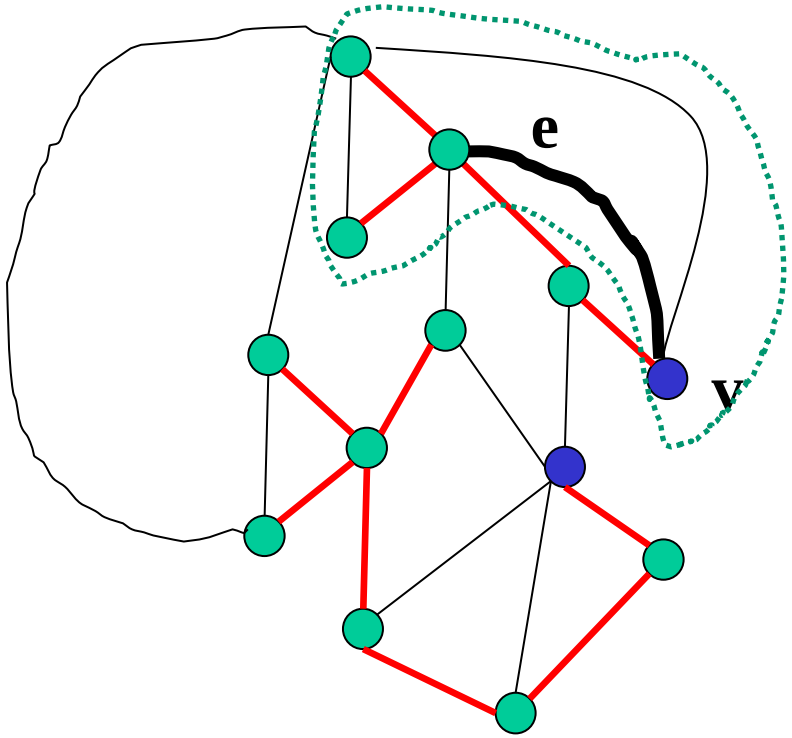


➔ All edges of C incident to T_v (except e) has level > i

➔ C cannot leave T_v

If it does, then there is another edge $f \neq e$ of C leaving T_v, f is of level > i so it must have been a replacement edge

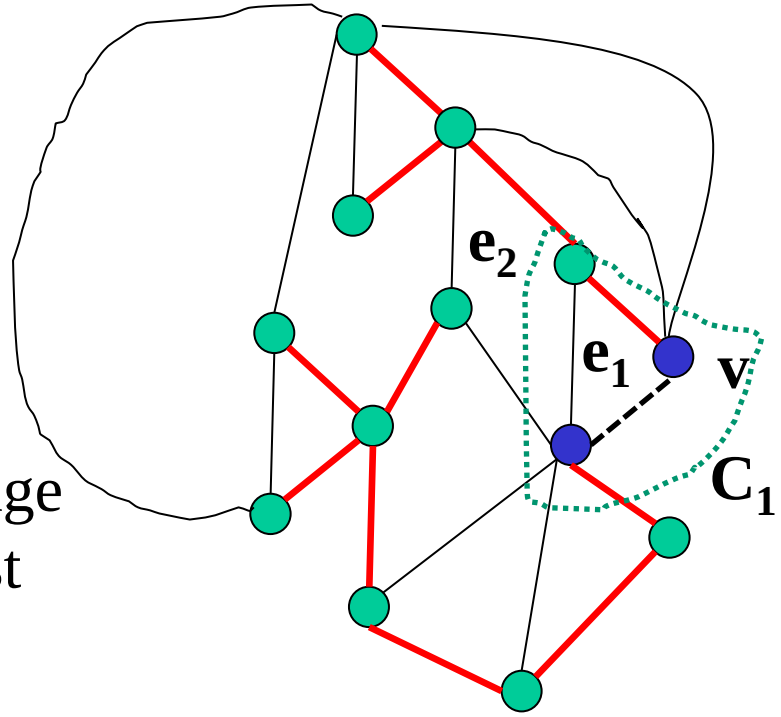
The lowest heaviest nontree edge invariant



So e is not a replacement edge and when we increase its level it stays the lowest as all other edges of C are at level > i at this point.

These invariants guarantee that we maintain an MSF

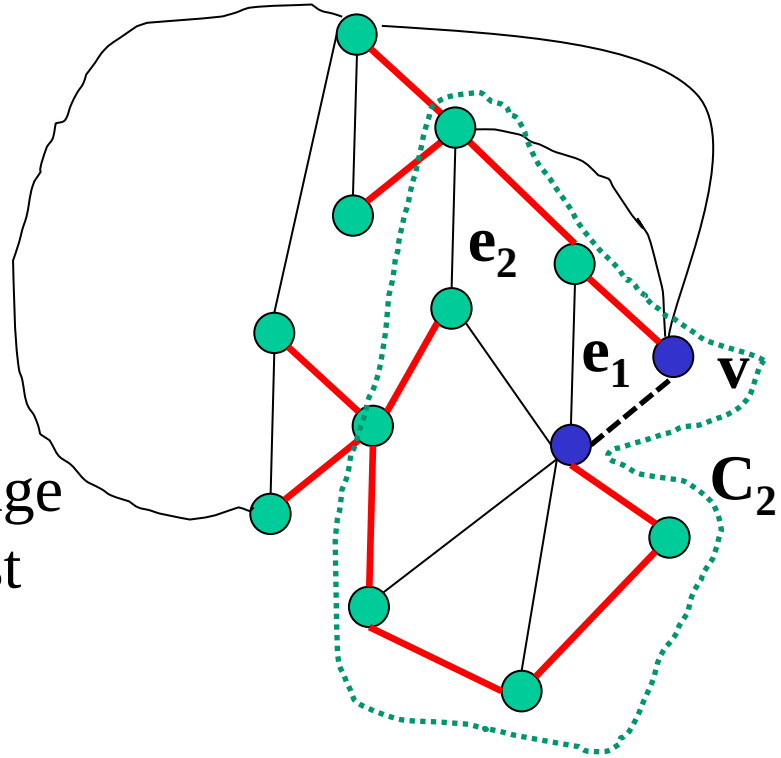
Lemma: The lightest replacement edge is of the highest level



Proof: Suppose e_1 is lighter than e_2. Consider the cycles C_1 and C_2

These invariants guarantee that we maintain an MSF

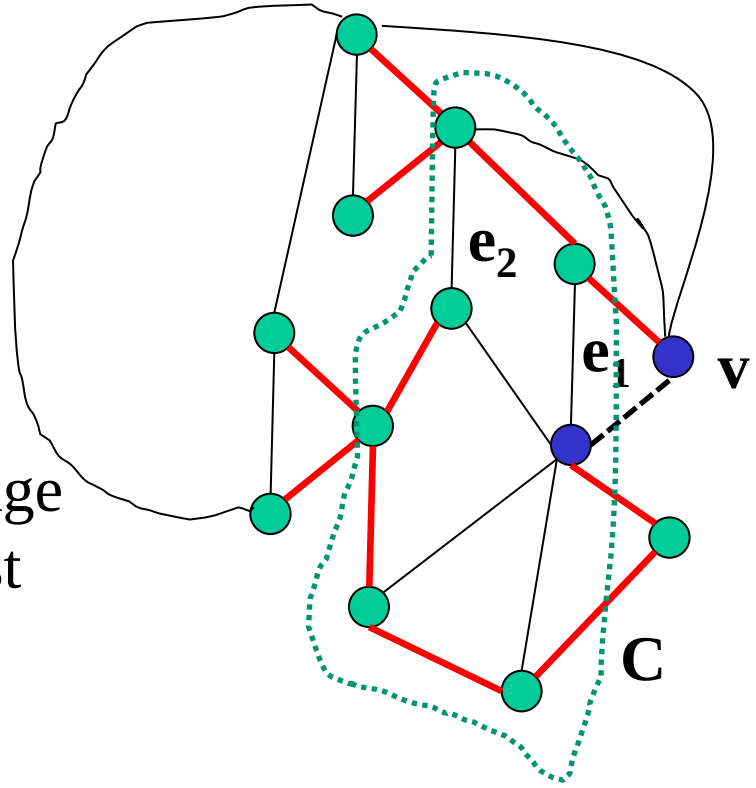
Lemma: The lightest replacement edge is of the highest level



Proof: Suppose e_1 is lighter than e_2. Consider the cycles C_1 and C_2

These invariants guarantee that we maintain an MSF

Lemma: The lightest replacement edge is of the highest level



Proof: Suppose e_1 is lighter than e_2. Consider the cycles C_1 and C_2. Let $C=C_1 \Delta C_2$. The edge e_2 is the heaviest non-tree edge on this cycle so $\ell(e_2) \leq \ell(e_1)$ by the new invariant