Dynamic trees (Steator and Tarjan 83)

Operations that we do on the trees

maketree(v)

w = findroot(v)

(w,c) = mincost(v)   (can do maxcost(v) instead)

addcost(v,c)

link(v,w,c(v,w))

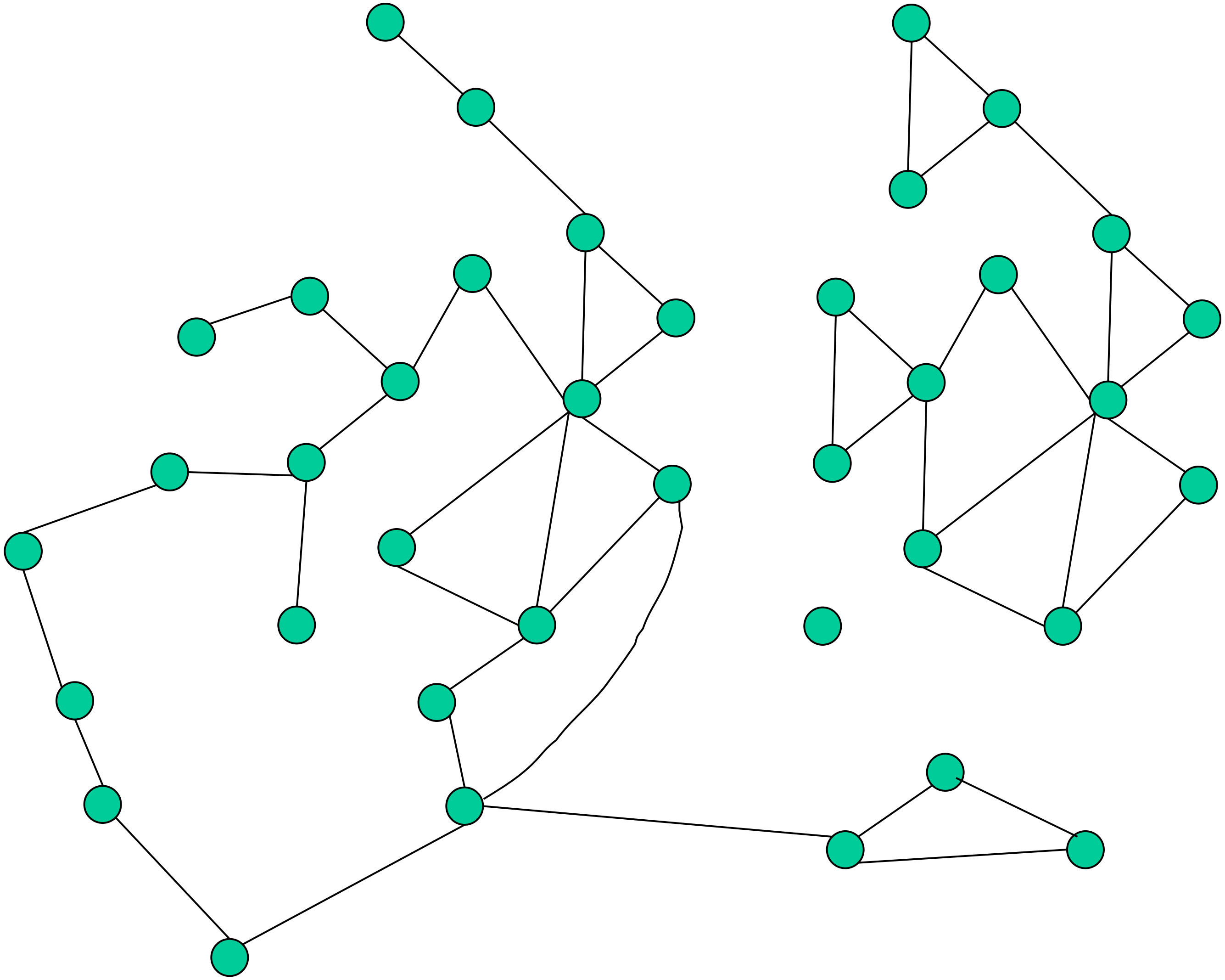
cut(v)

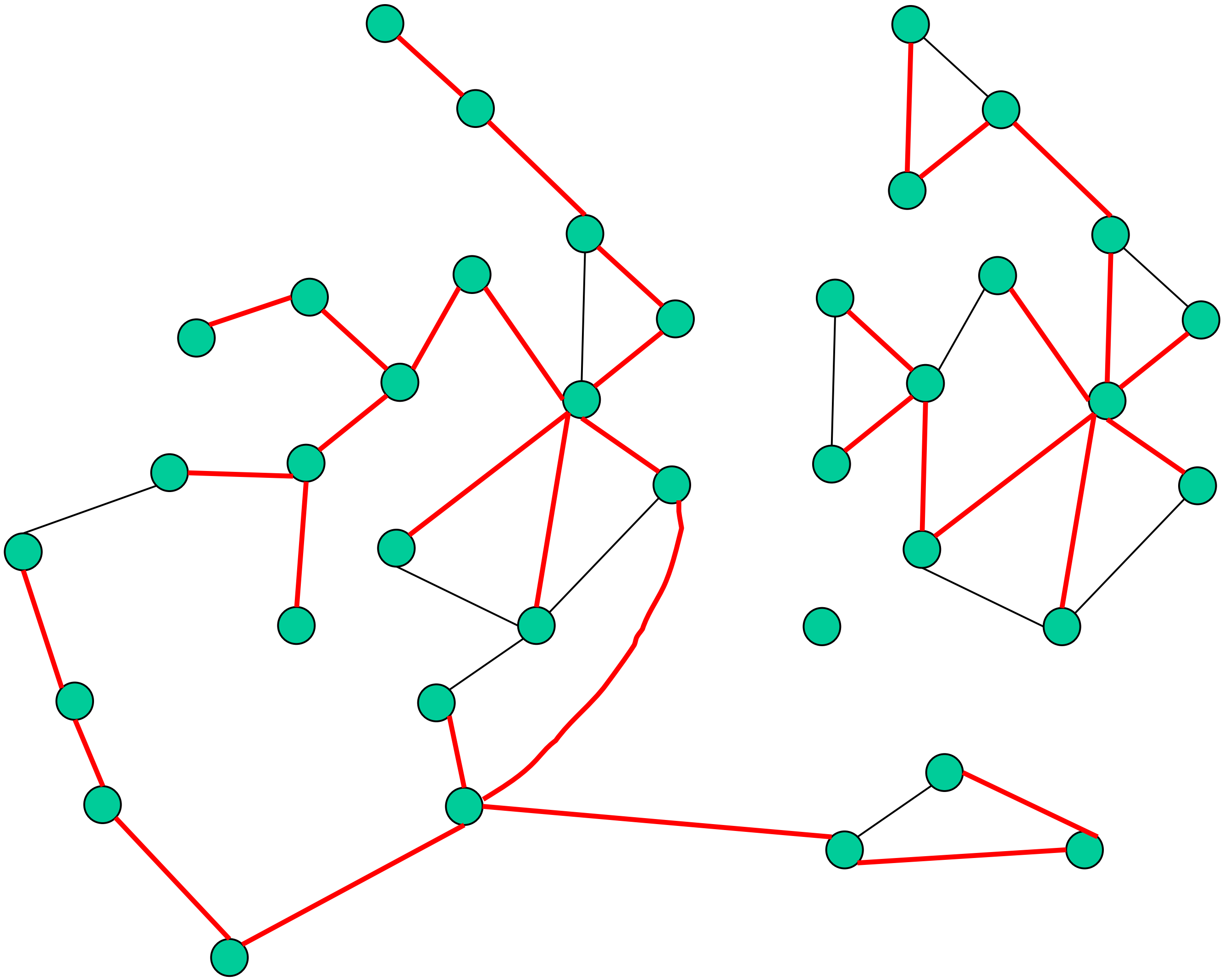
evert(v)

Applications

Incremental Minimum Spanning Forest

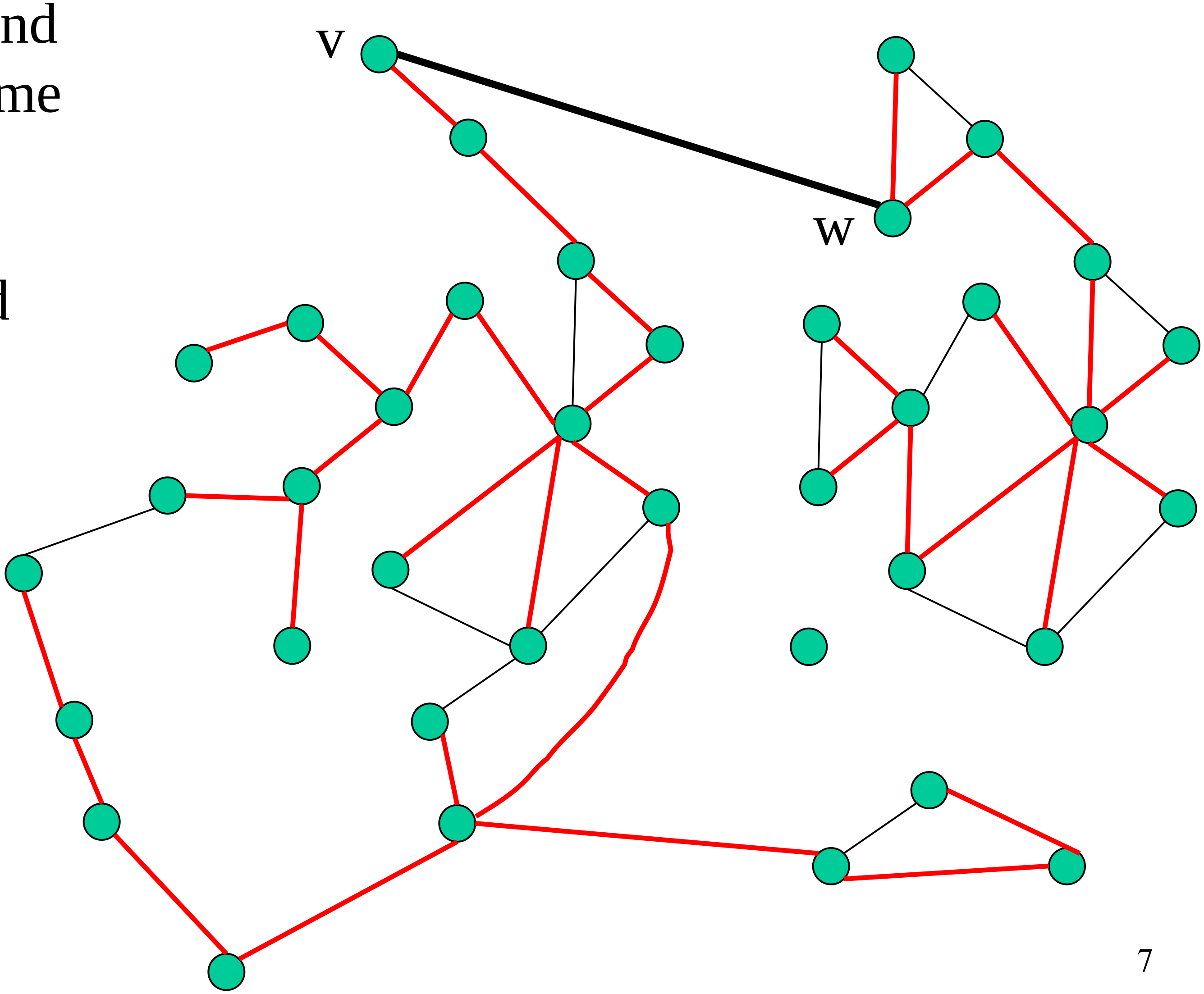
Maintain a minimum spanning forest of a graph to which we insert edges





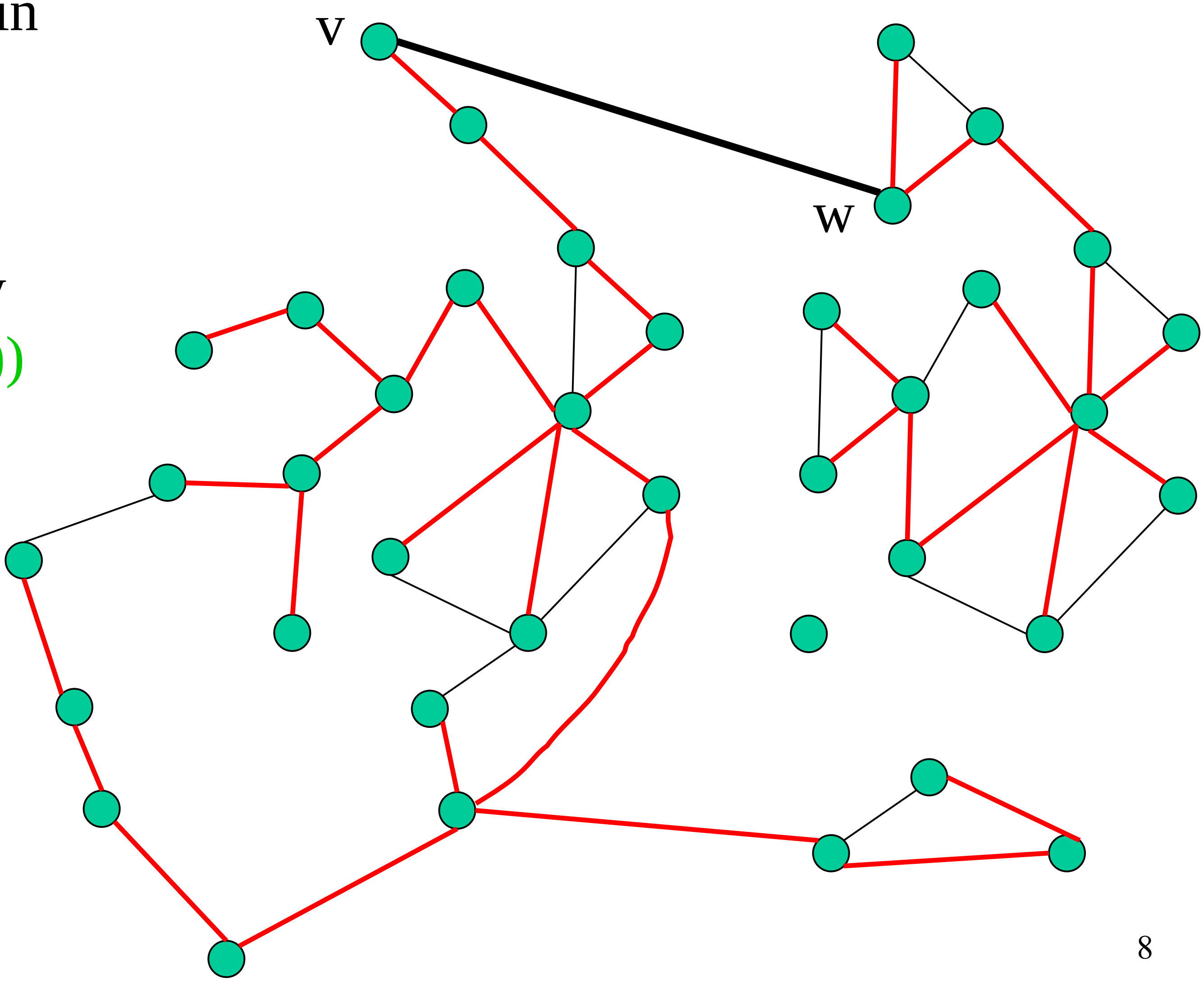
Add an edge (v,w)

Discover if v and w are in the same component by comparing findroot(v) and findroot(w)



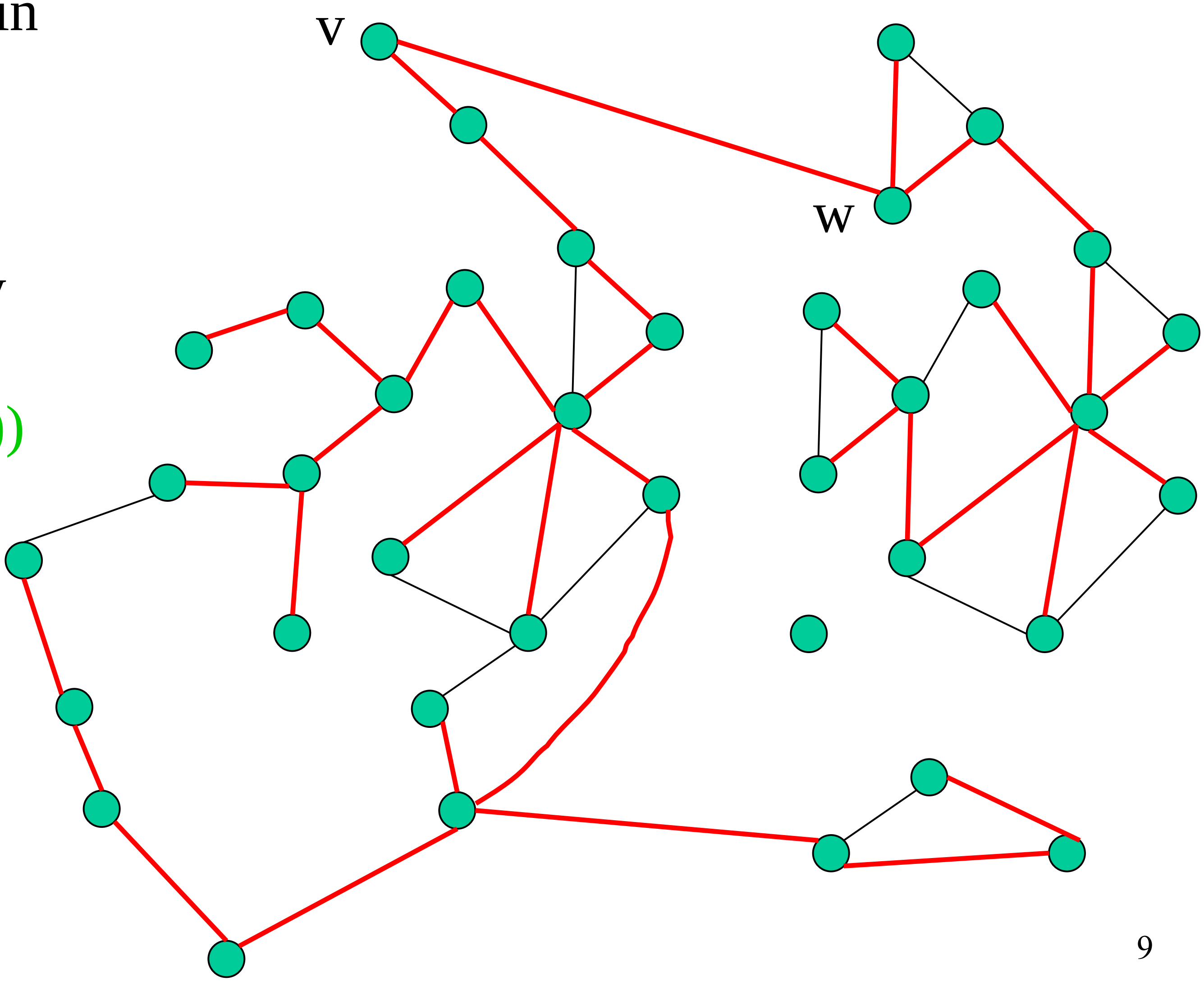
Add an edge (v,w)

If v and w are in different components then add (v,w) to the forest by link(v,w,c(v,w))



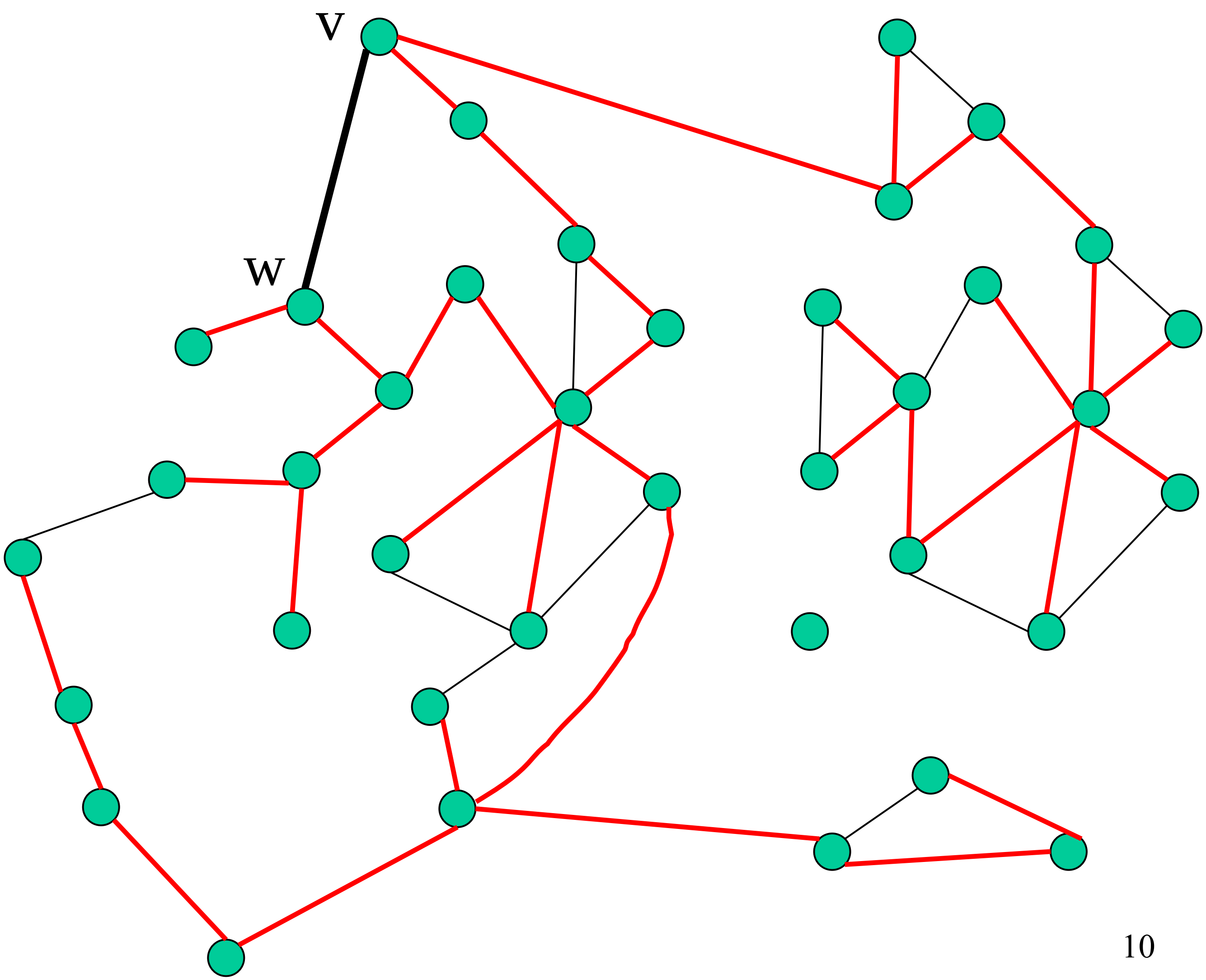
Add an edge (v,w)

If v and w are in different components then add (v,w) to the forest by evert(v), and link(v,w,c(v,w))



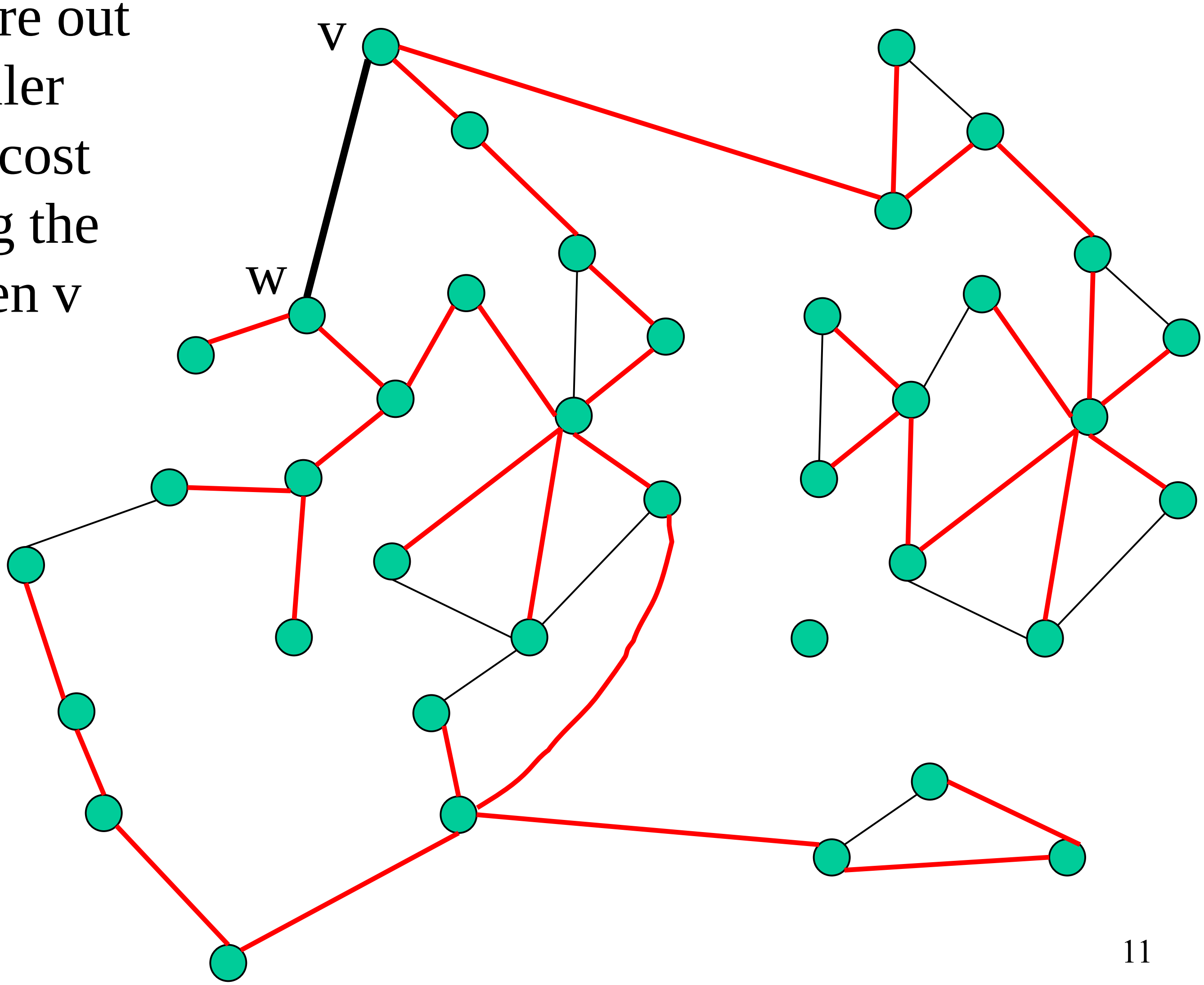
Add an edge (v,w)

What if v and w are in same component ?



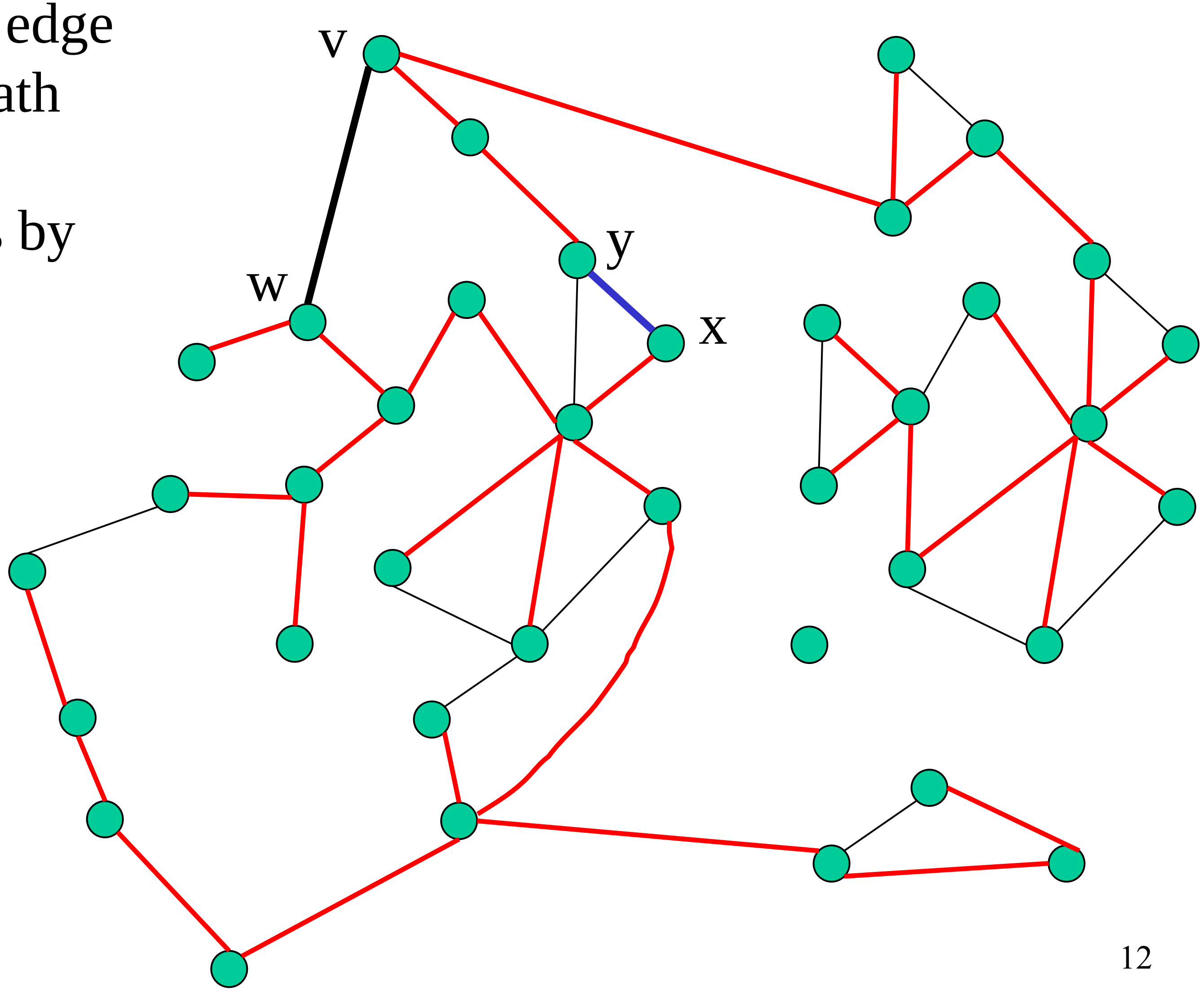
Add an edge (v,w)

We have to figure out if c(v,w) is smaller than the largest cost of an edge along the tree path between v and w.



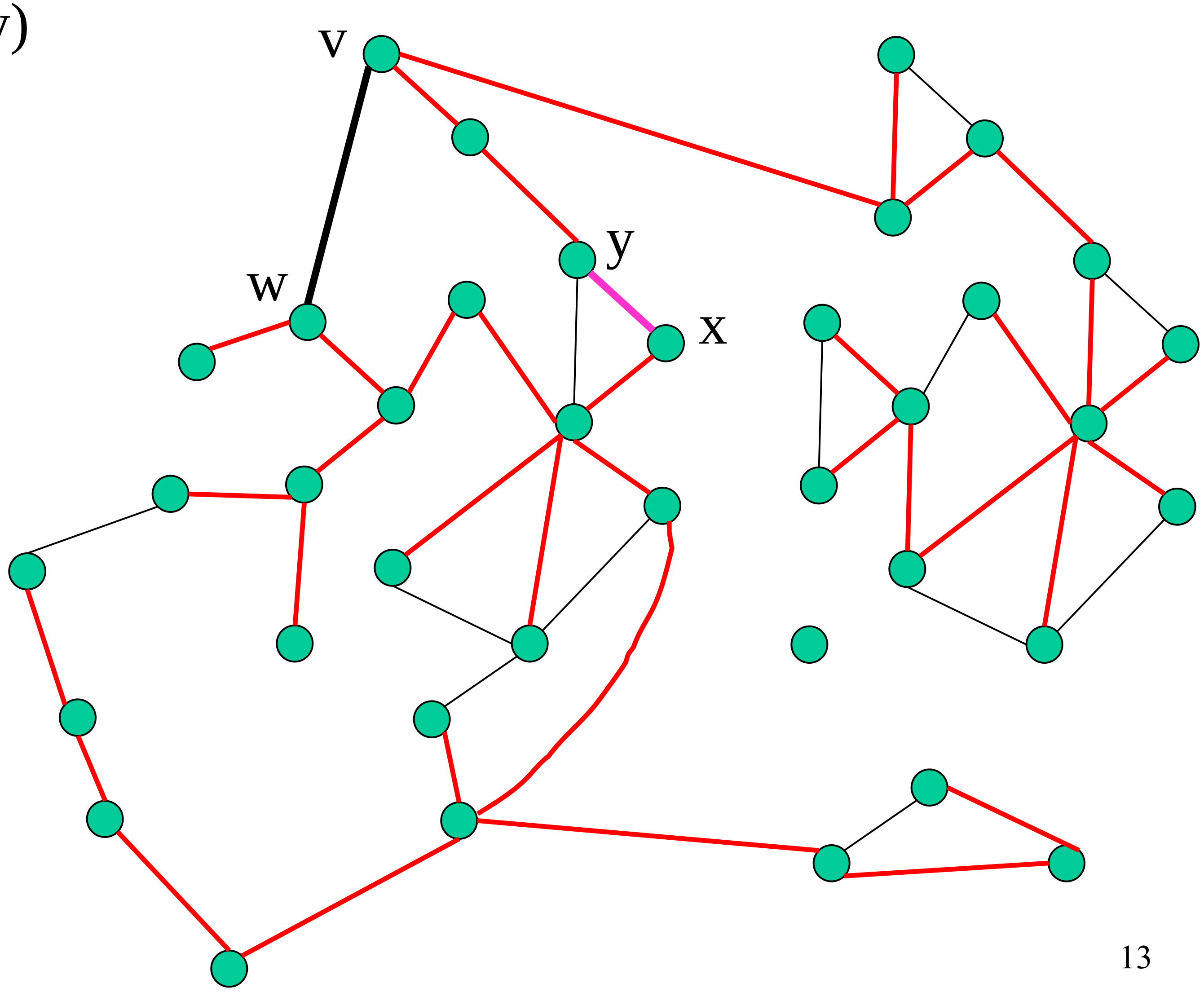
Add an edge (v,w)

Find the largest edge along the tree path from v to w by evert(v) follows by maxcost(w)



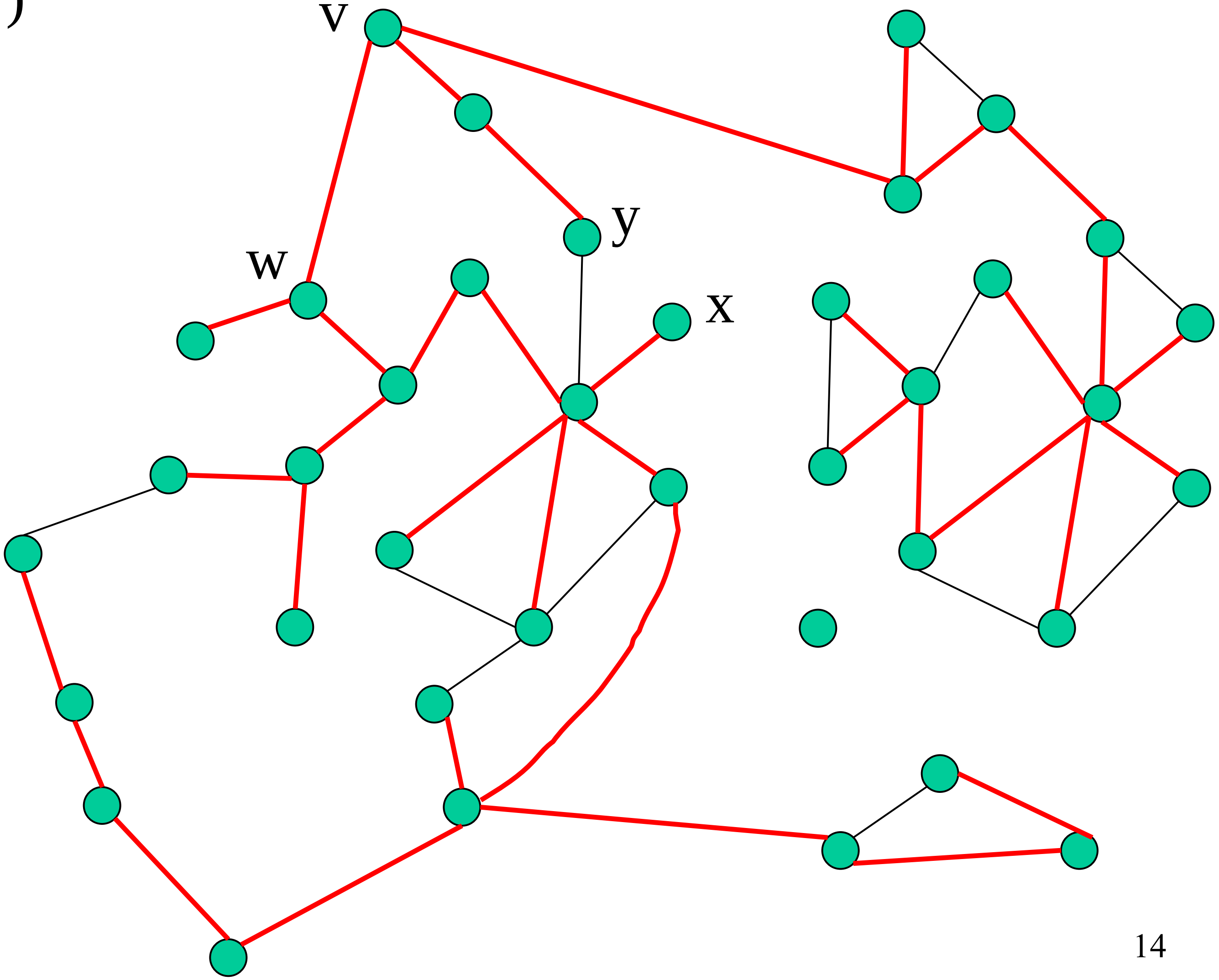
Add an edge (v,w)

If c(v,w) < c(x,y)
then cut(x) and
link(v,w,c(v,w))



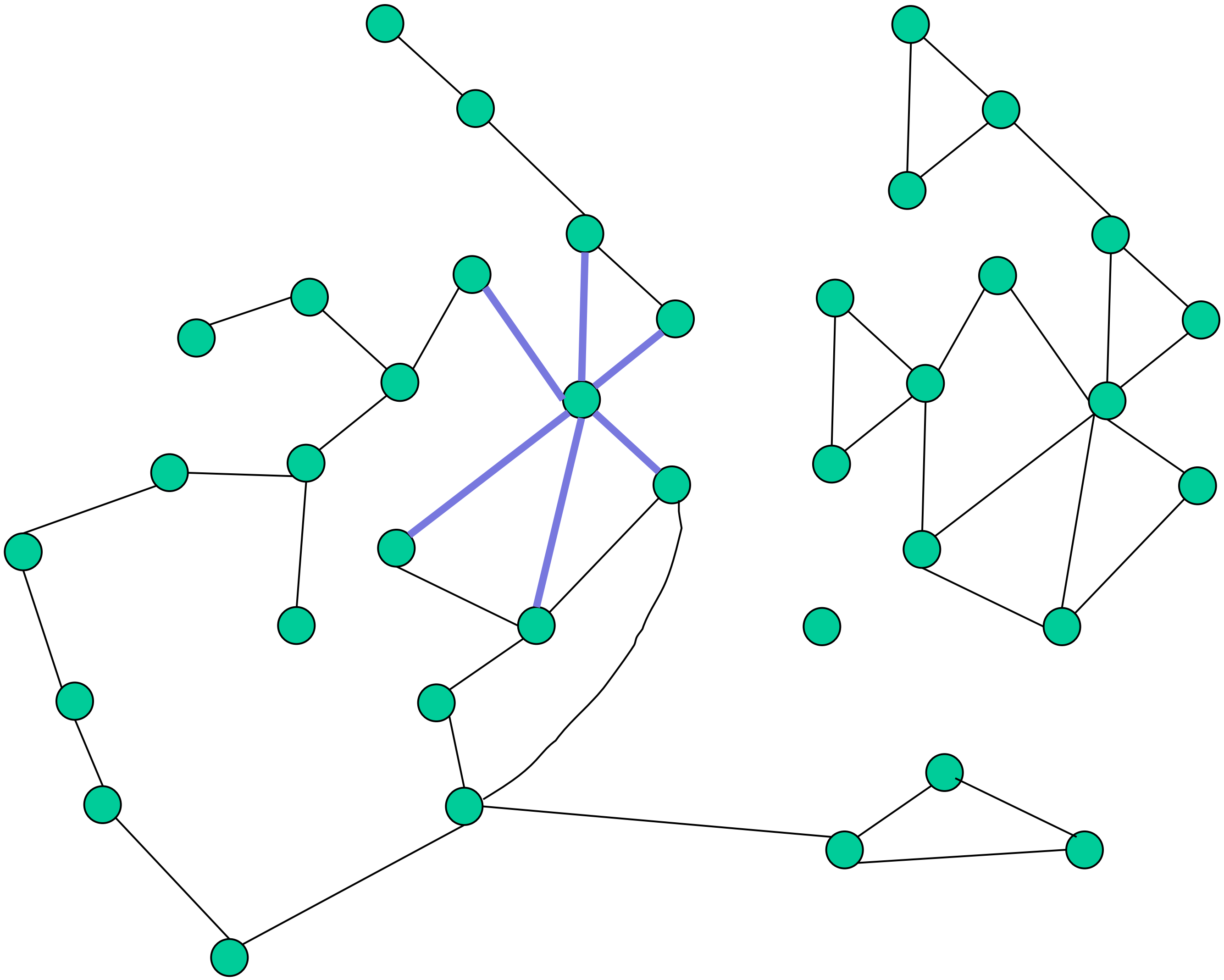
Add an edge (v,w)

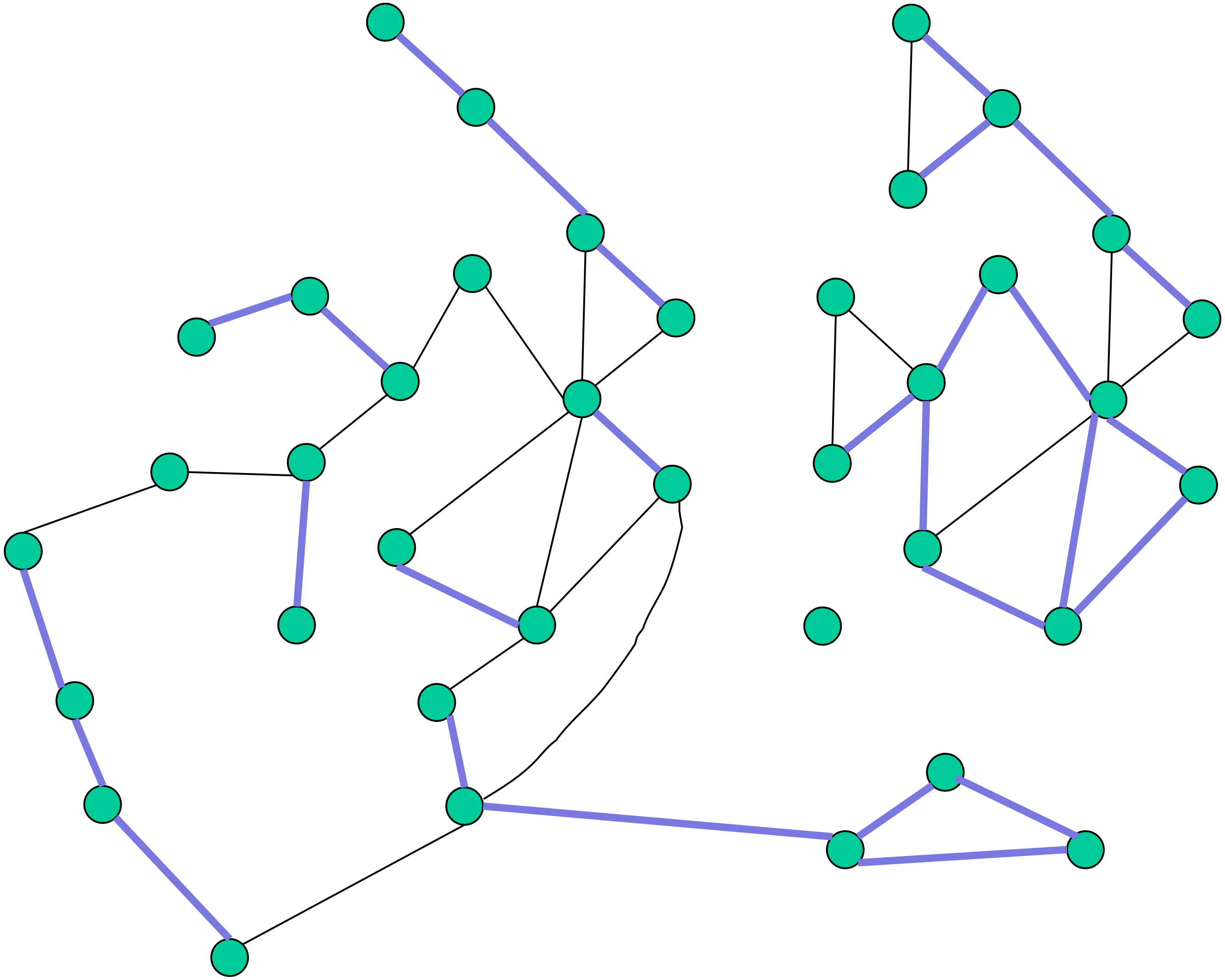
If c(v,w) < c(x,y)
then cut(x) and
link(v,w,c(v,w))



Application (2)

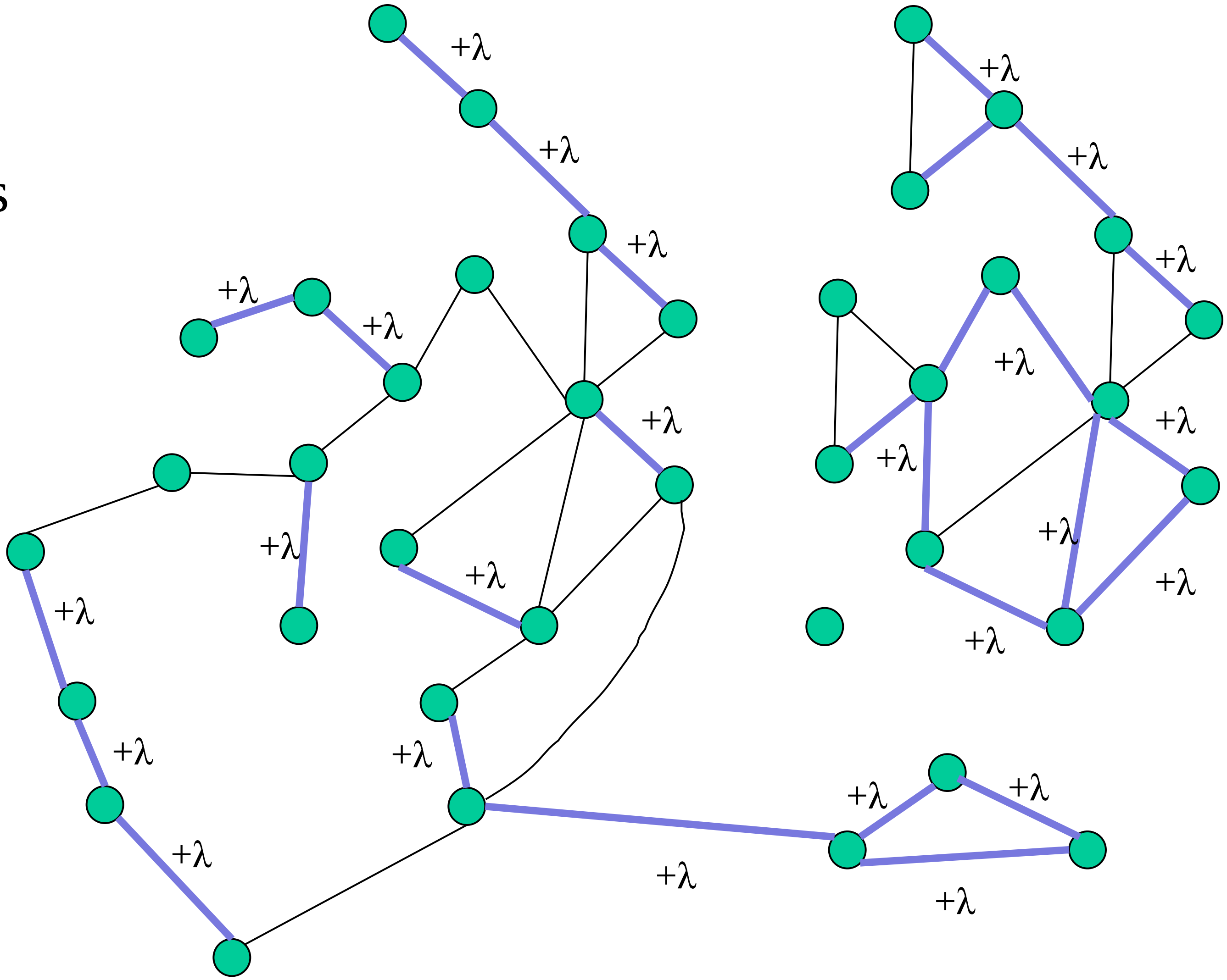
- Minimum spanning forest with a particular number of blue edges

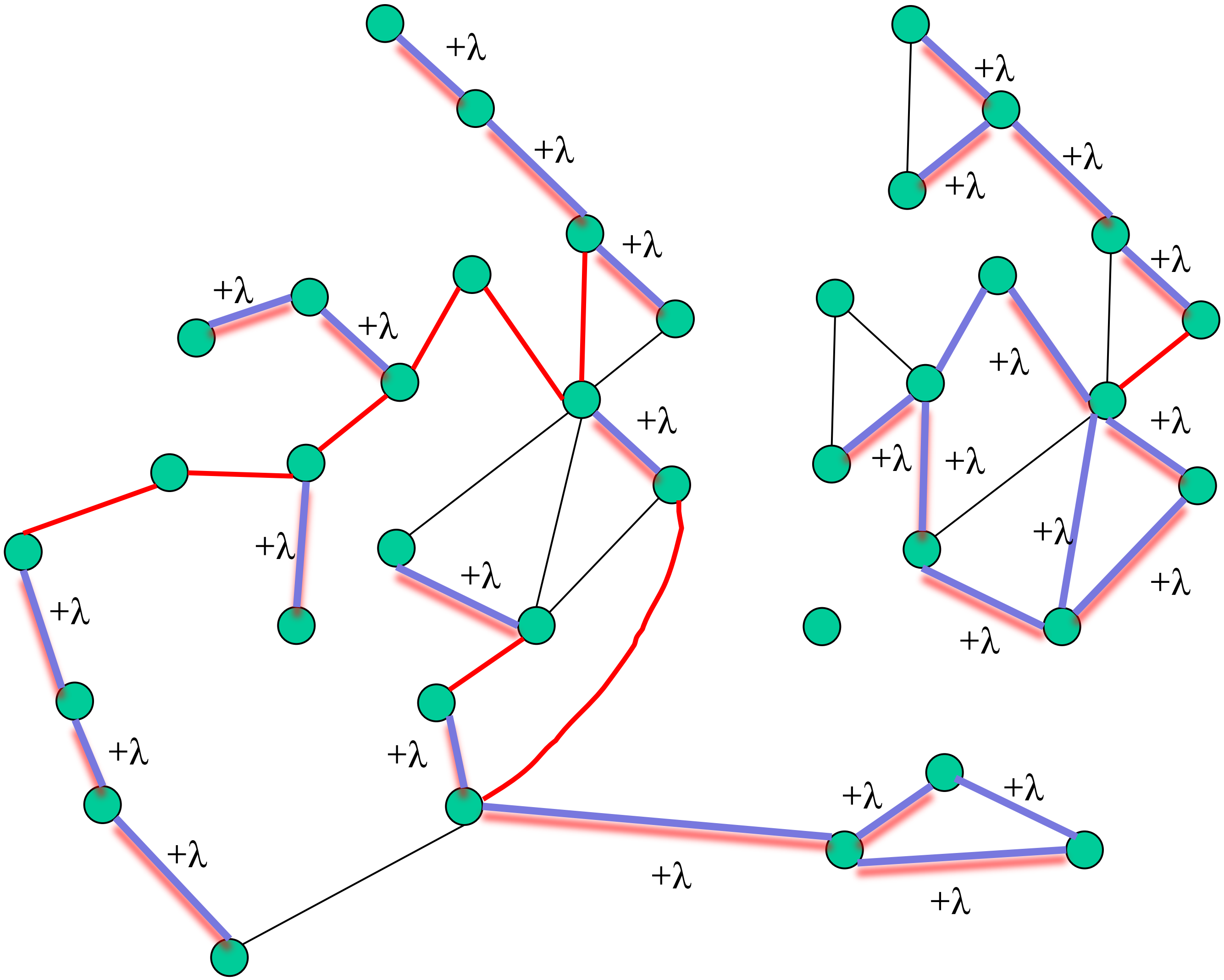




Suppose we add λ to the weights of the blue edges and compute the MSF

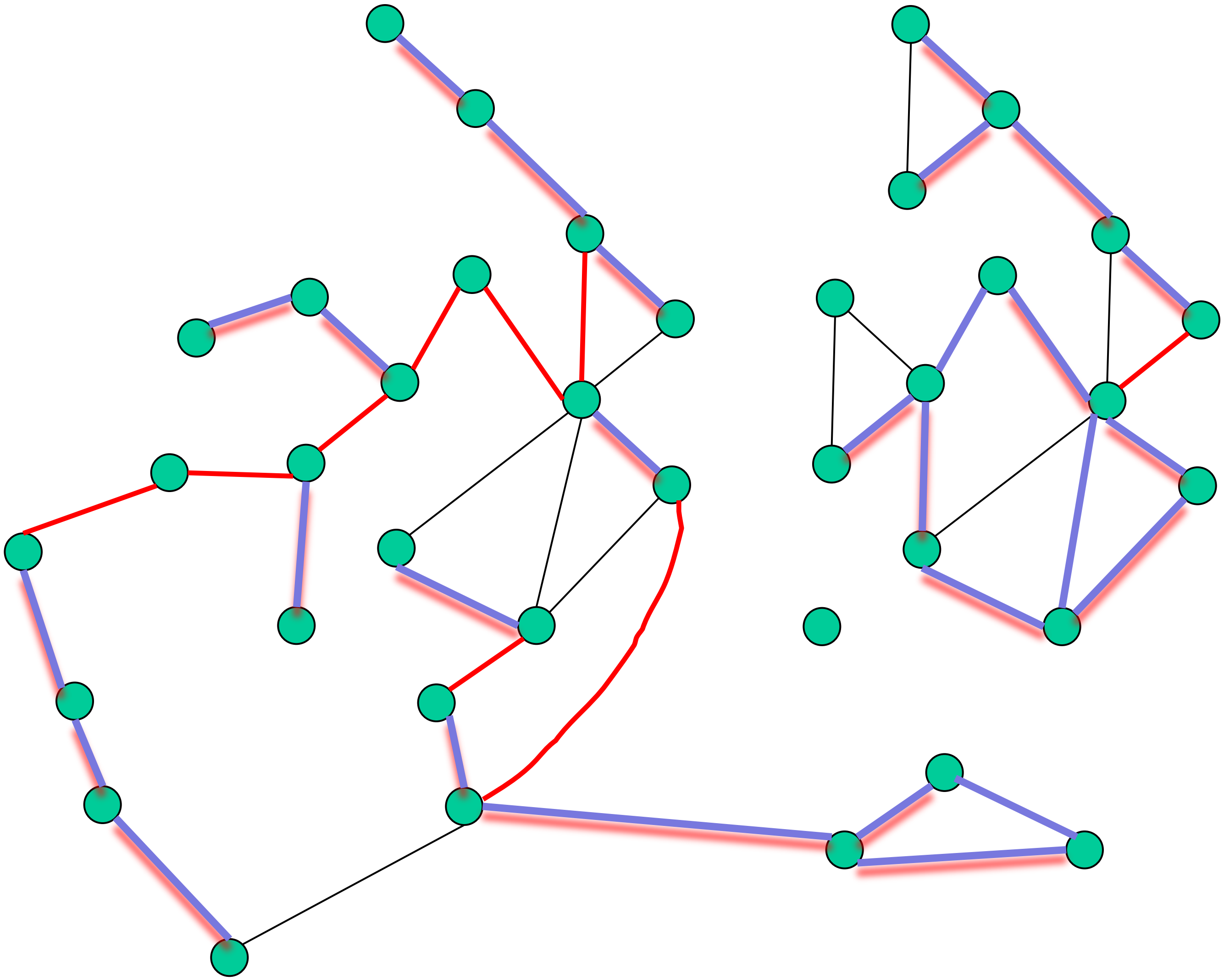
If $\lambda = -\infty$ we will get a MSF with as many blue edges as possible



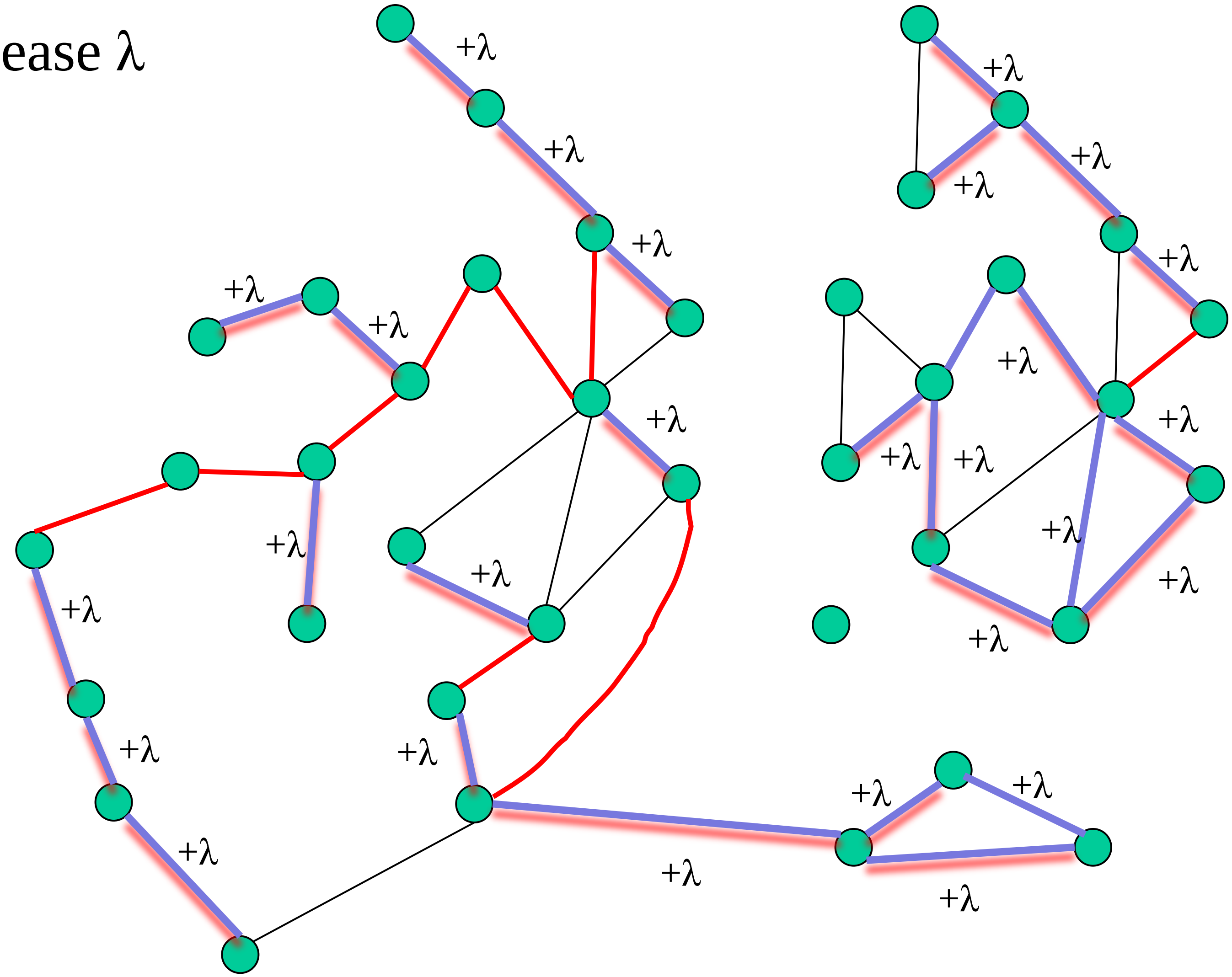

+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ
+λ

The blue edges excluded cannot be in any MSF. Black edges included will be in any MSF

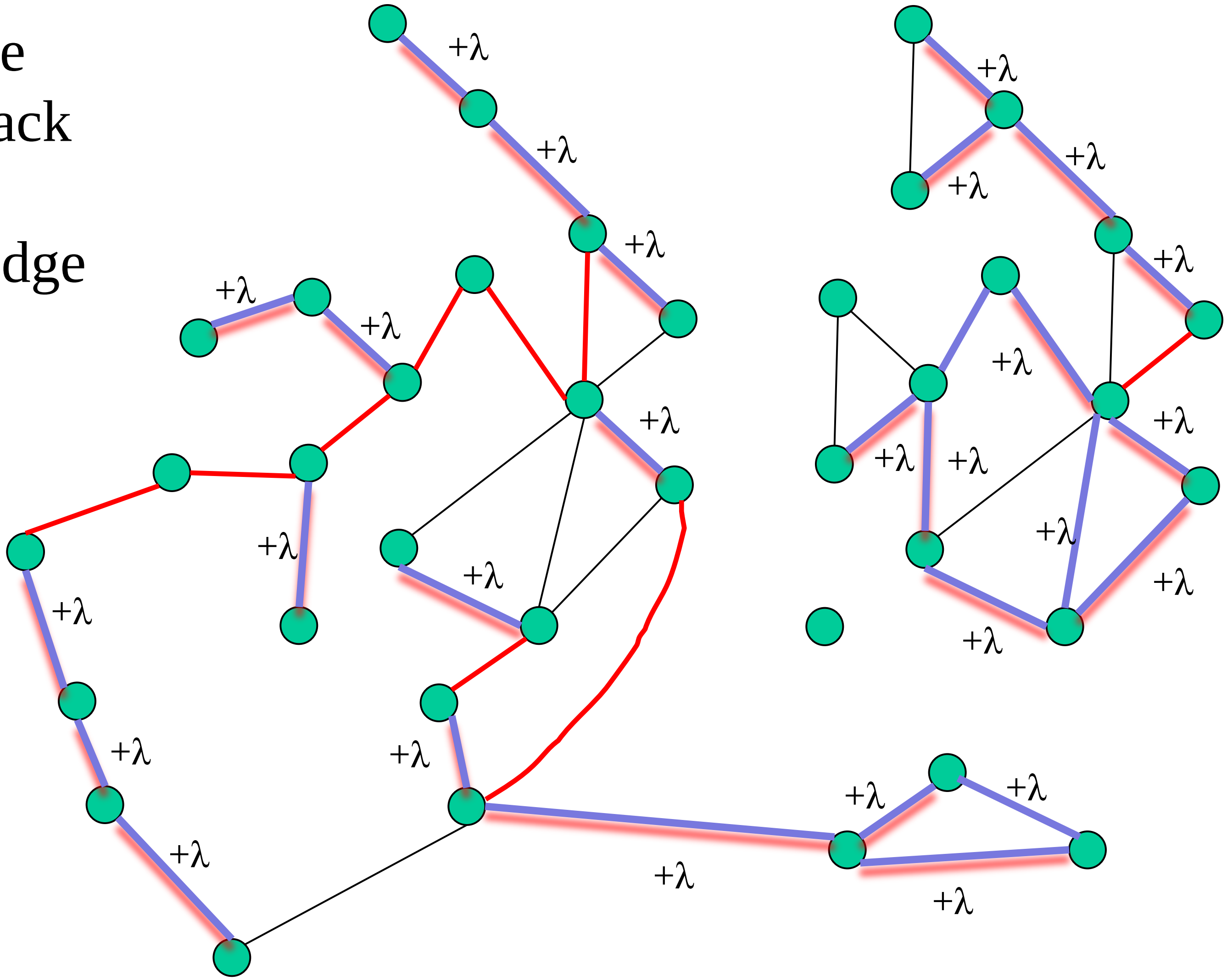
Let M be the maximum # of blue edges in a spanning forest



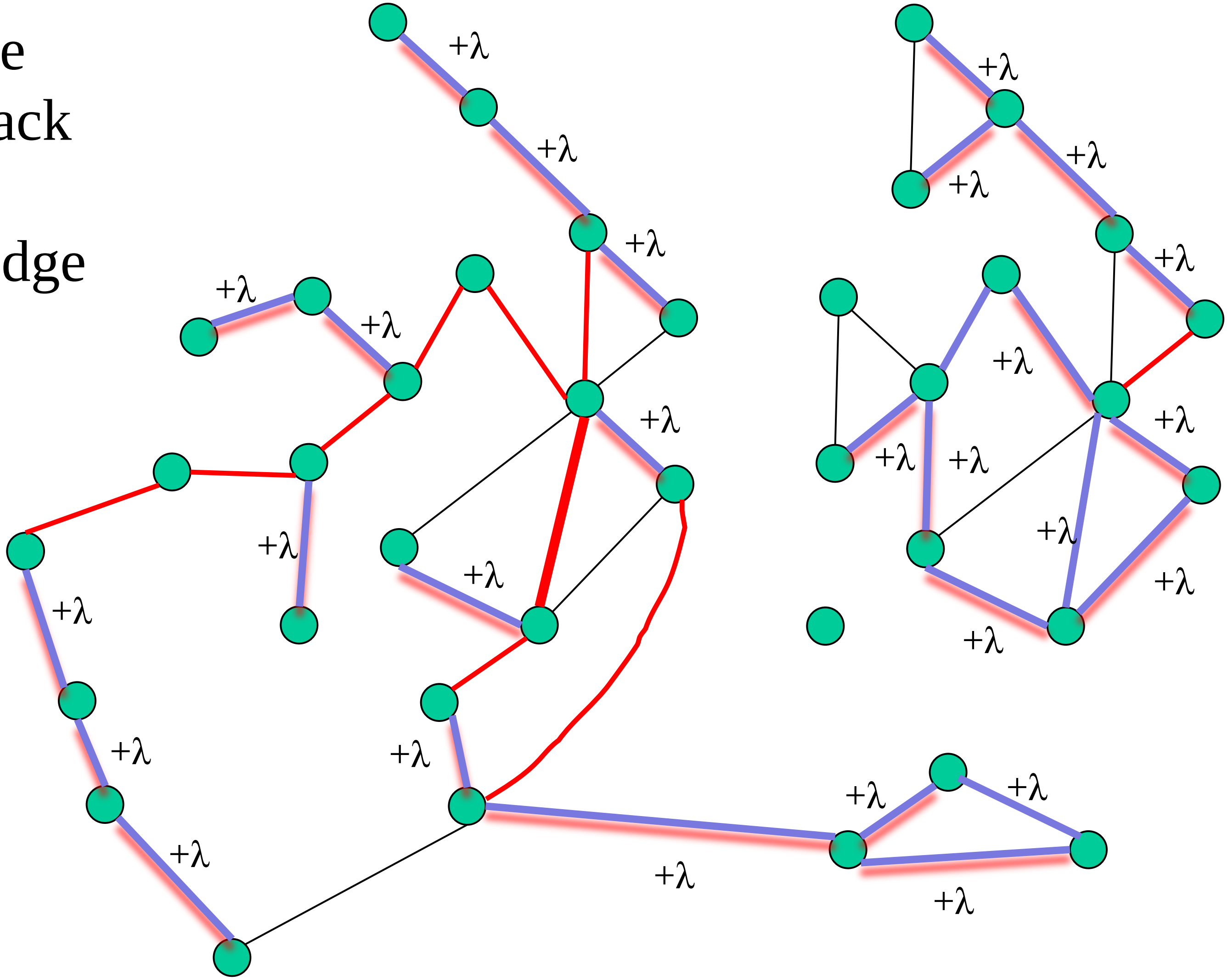
Imagine we increase λ



At some point we would trade a black edge for a more expensive blue edge

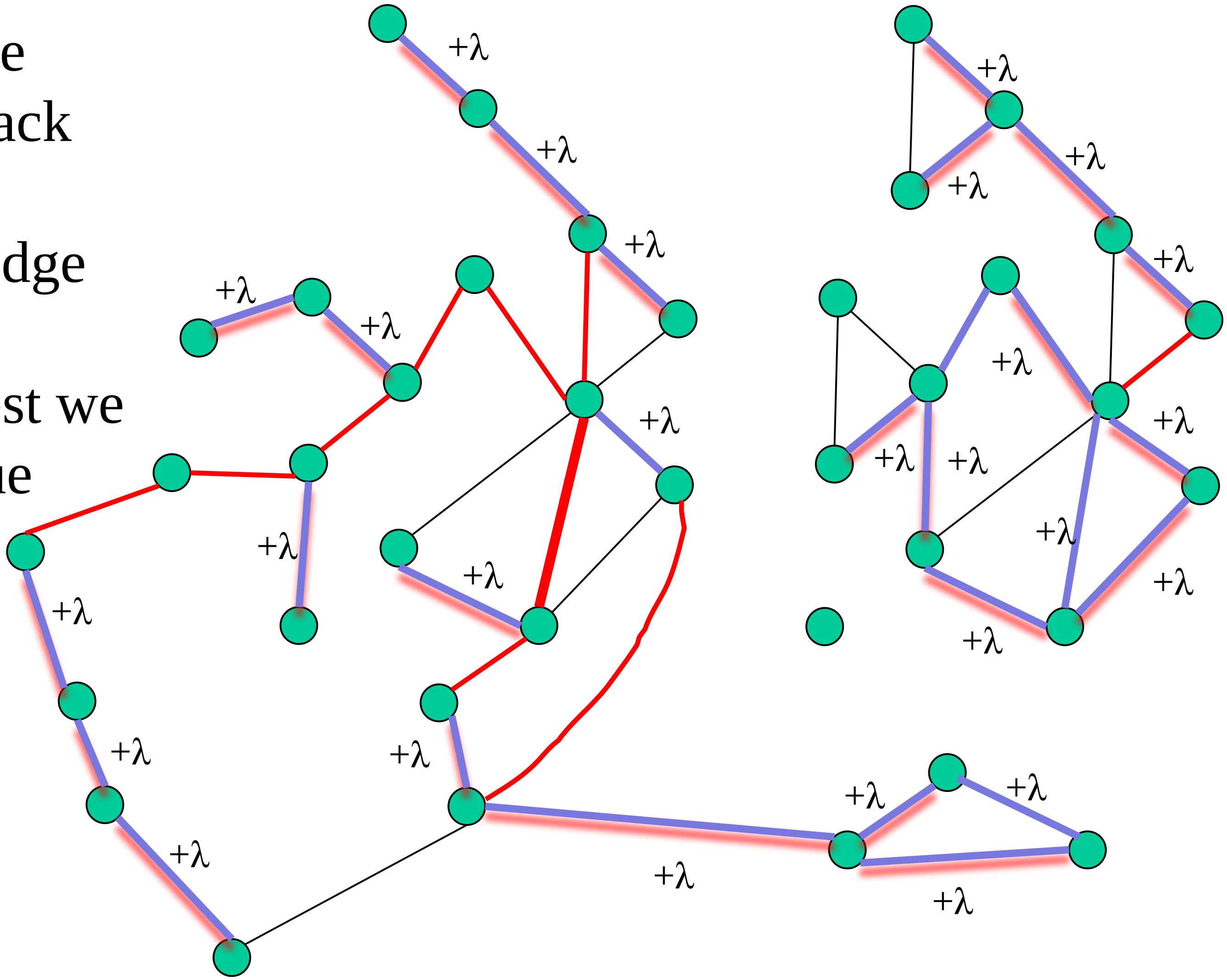


At some point we would trade a black edge for a more expensive blue edge



At some point we would trade a black edge for a more expensive blue edge

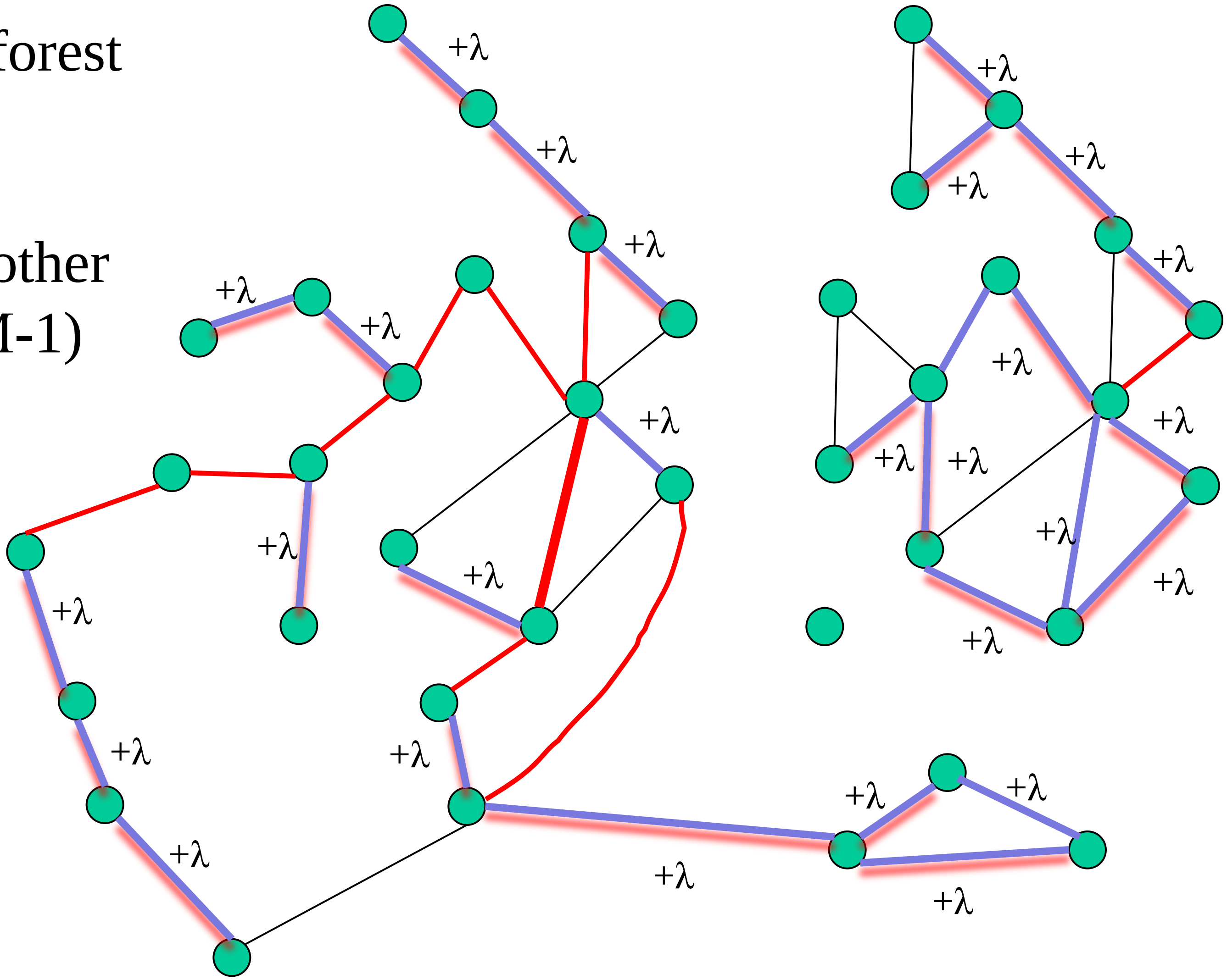
Let T be the forest we get with M-1 blue edges



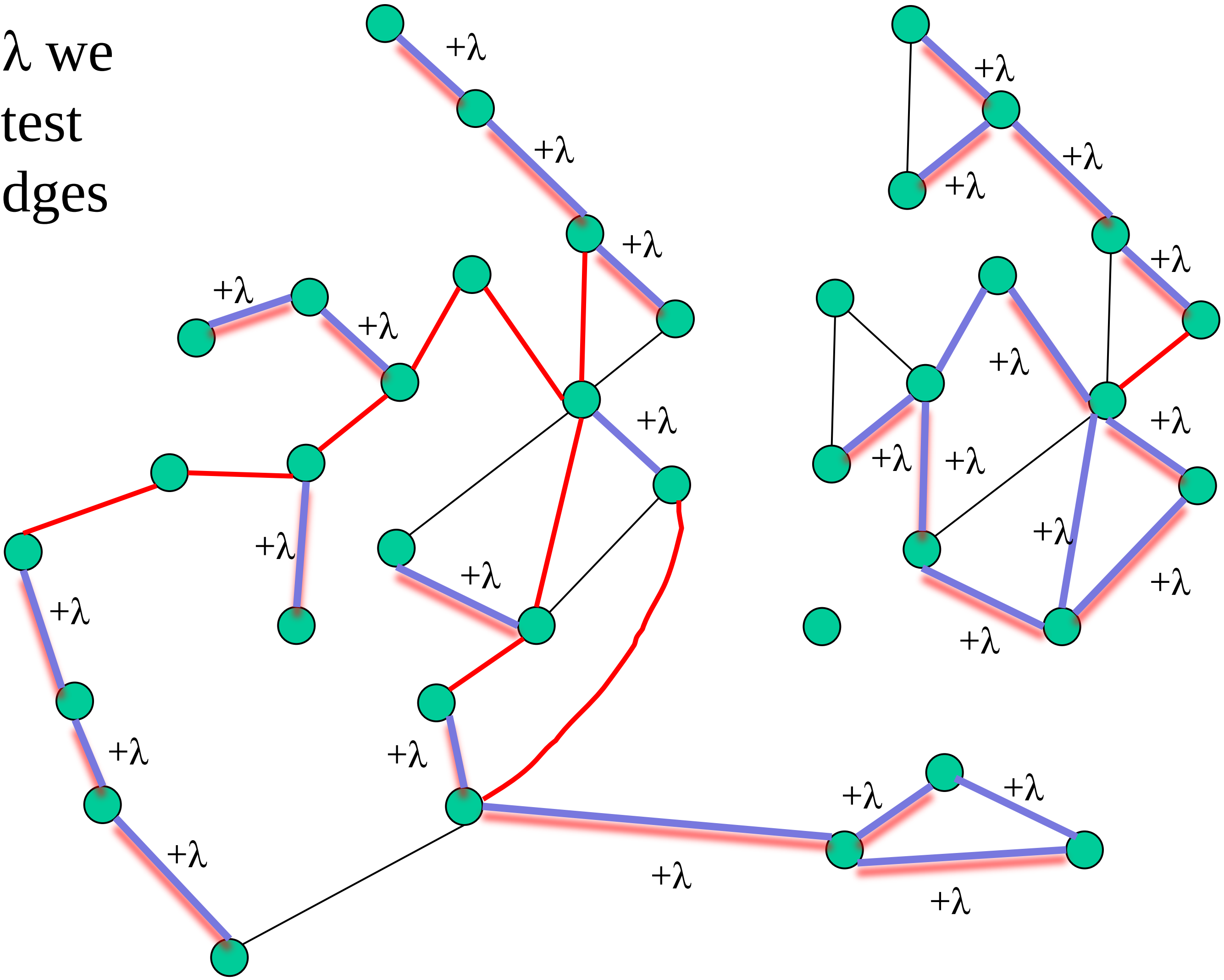
The cost of this forest is (M-1)λ + c(T)

The cost of any other forest T’ with (M-1) blue edges is (M-1)λ + c(T’)

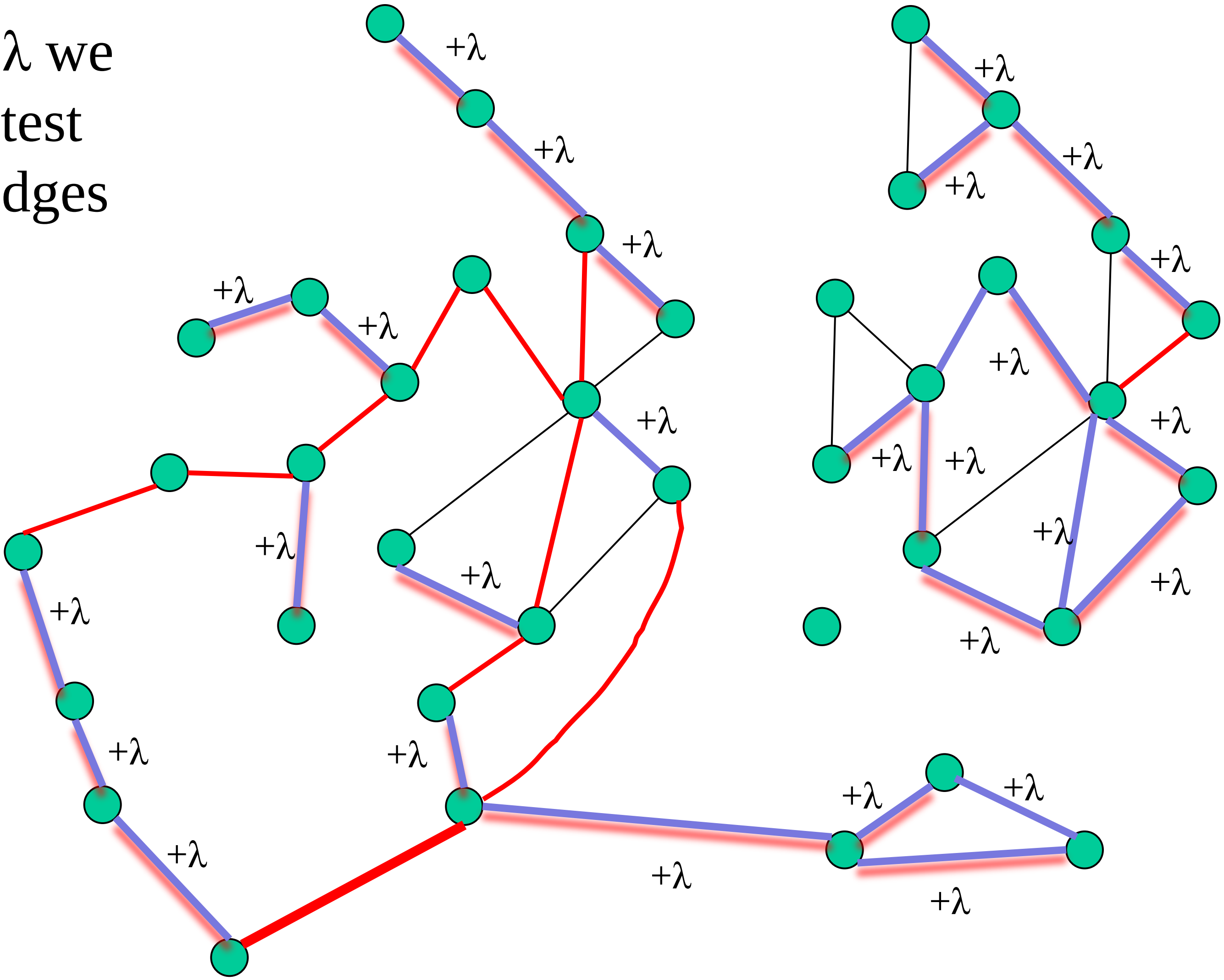
➔ c(T) < c(T’)



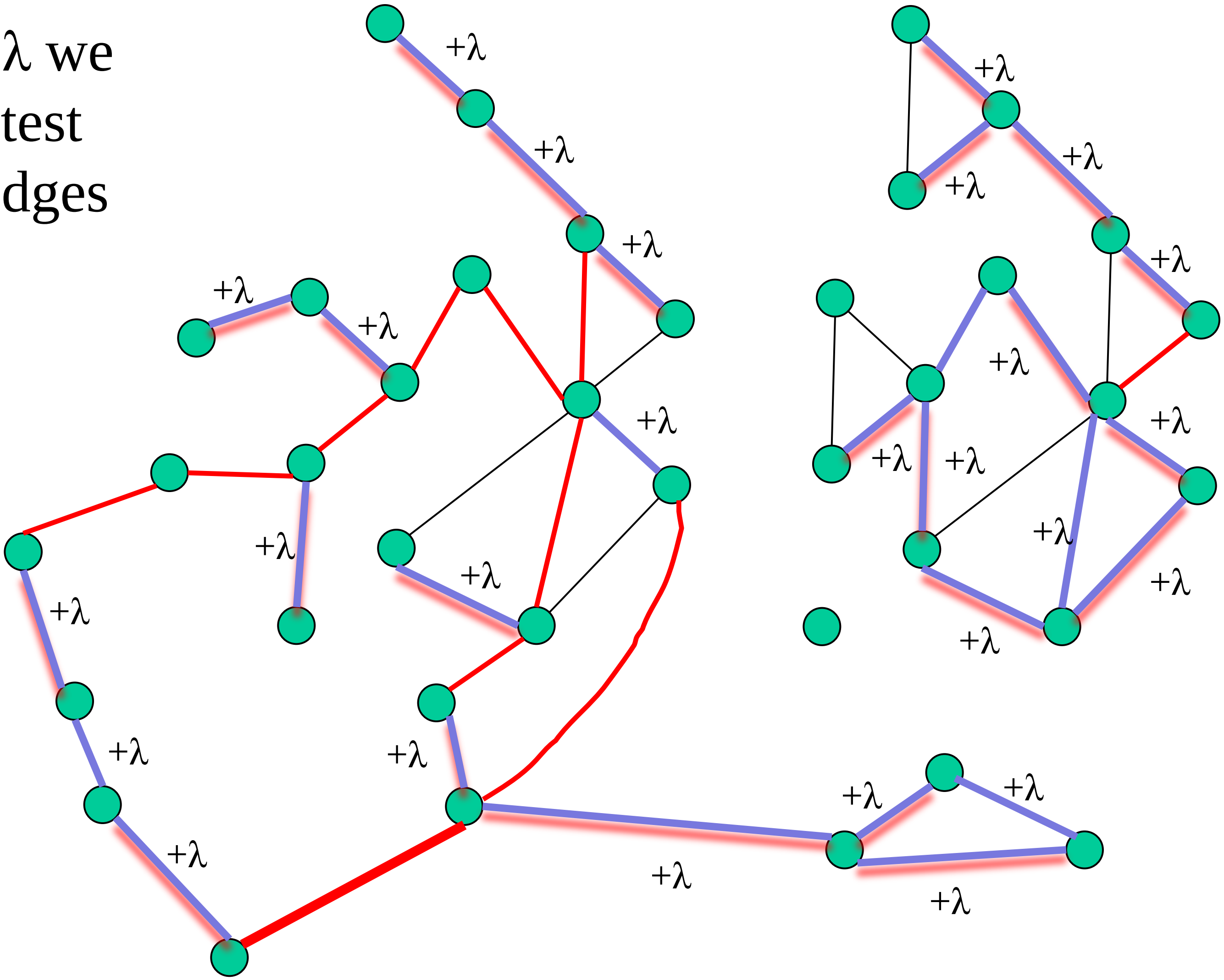
Keep increasing λ we will find the lightest MSF with M-2 edges and so on..



Keep increasing λ we will find the lightest MSF with M-2 edges and so on..



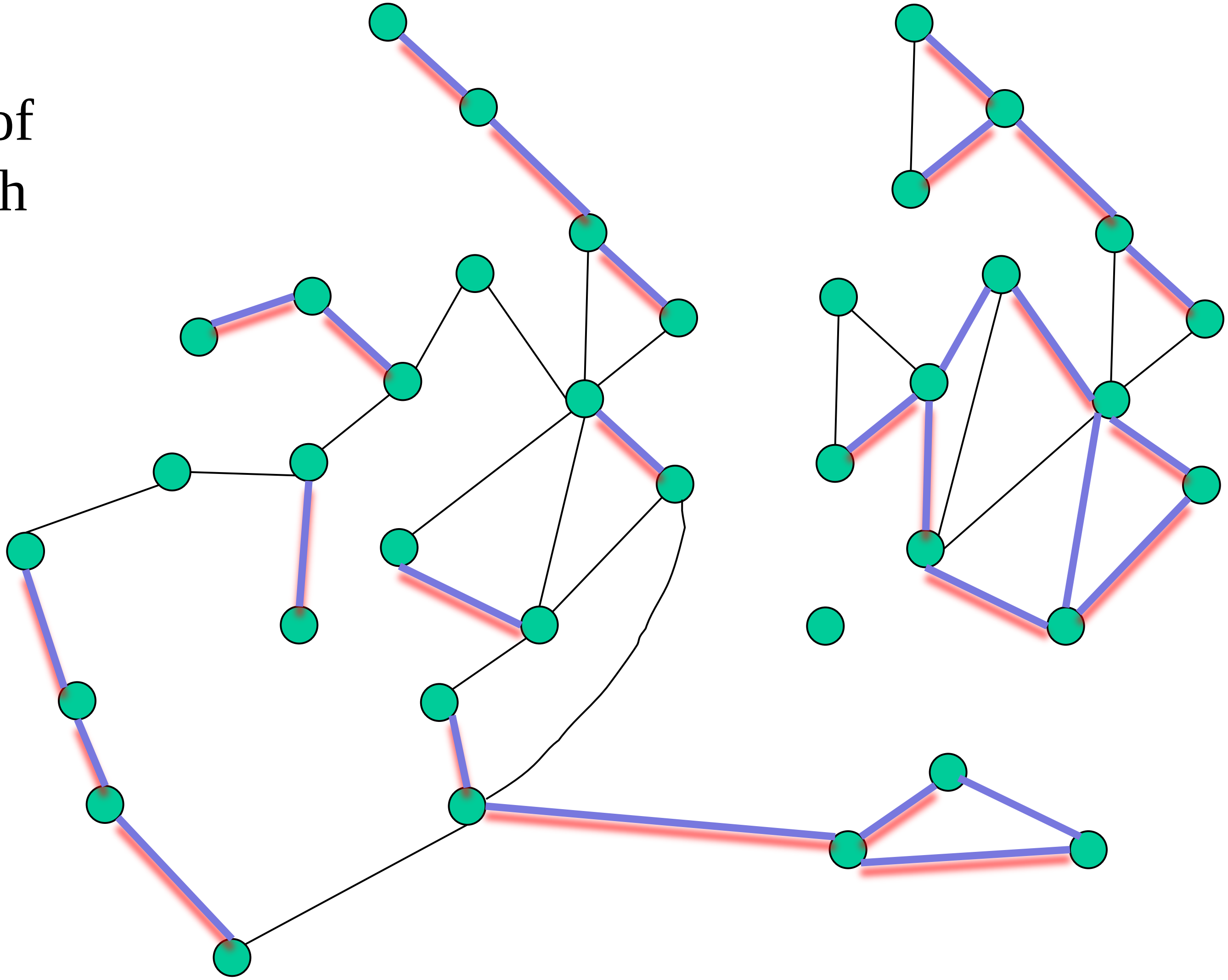
Keep increasing λ we will find the lightest MSF with M-2 edges and so on..



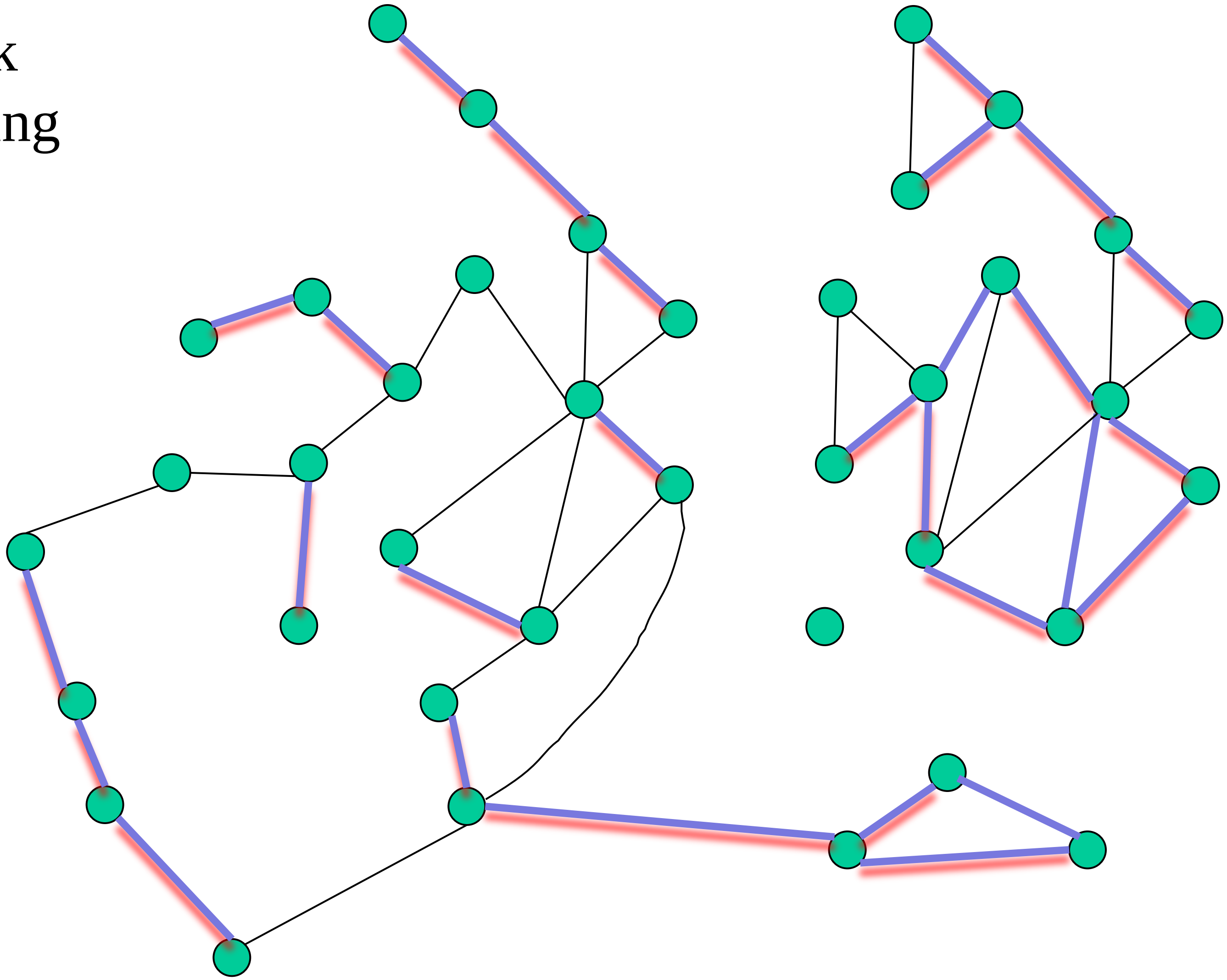
The key observation

- We can find the critical values of λ efficiently

Start with a spanning forest of the blue subgraph

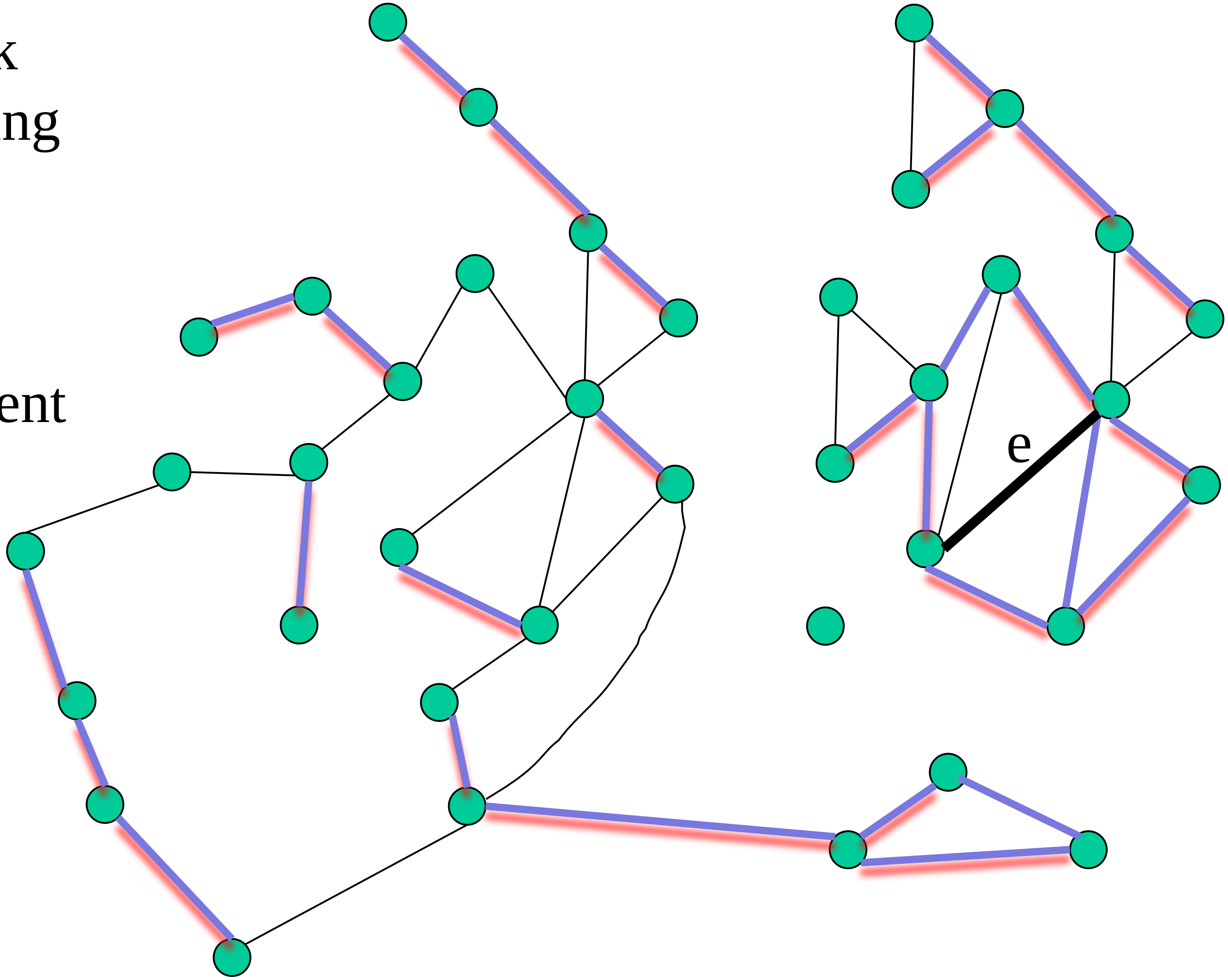


Process the black edges in increasing order of their weight



Process the black edges in increasing order of their weight

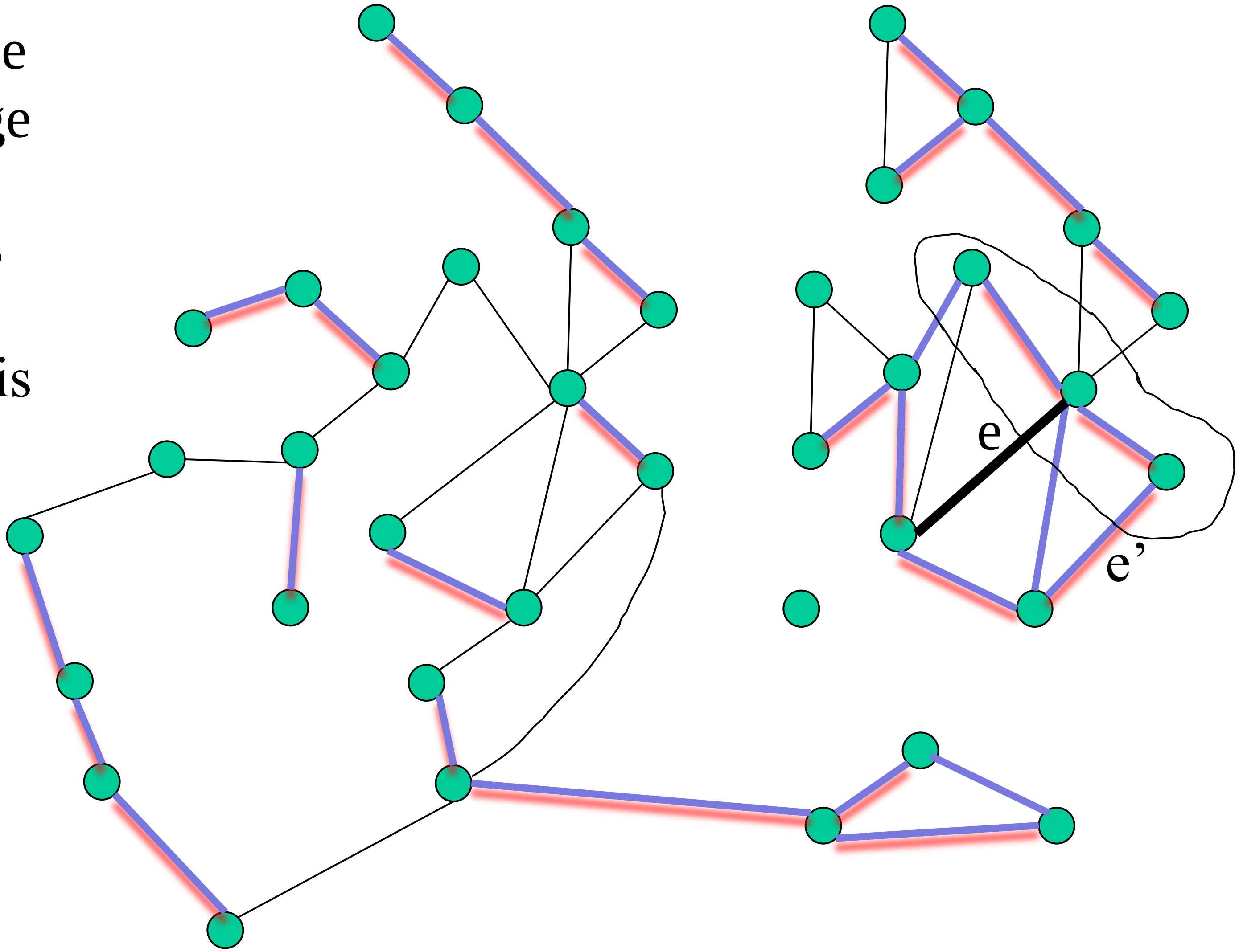
Let e be the current **black** edge



If e closes a cycle find the blue edge e’ of maximum cost on the cycle

$\lambda_e = c(e) - c(e')$ is a critical value

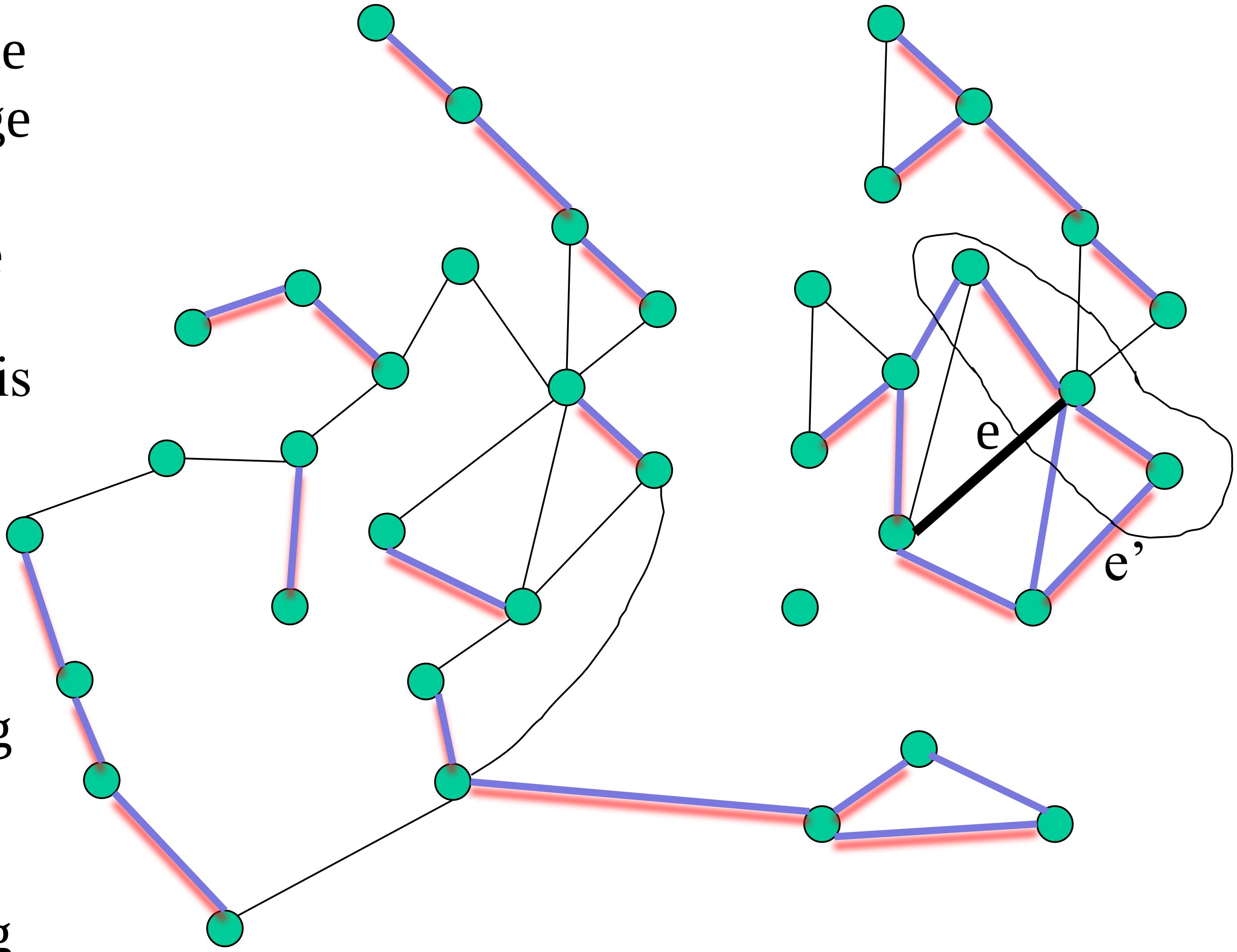
Consider the cut defined by removing e’ from the forest



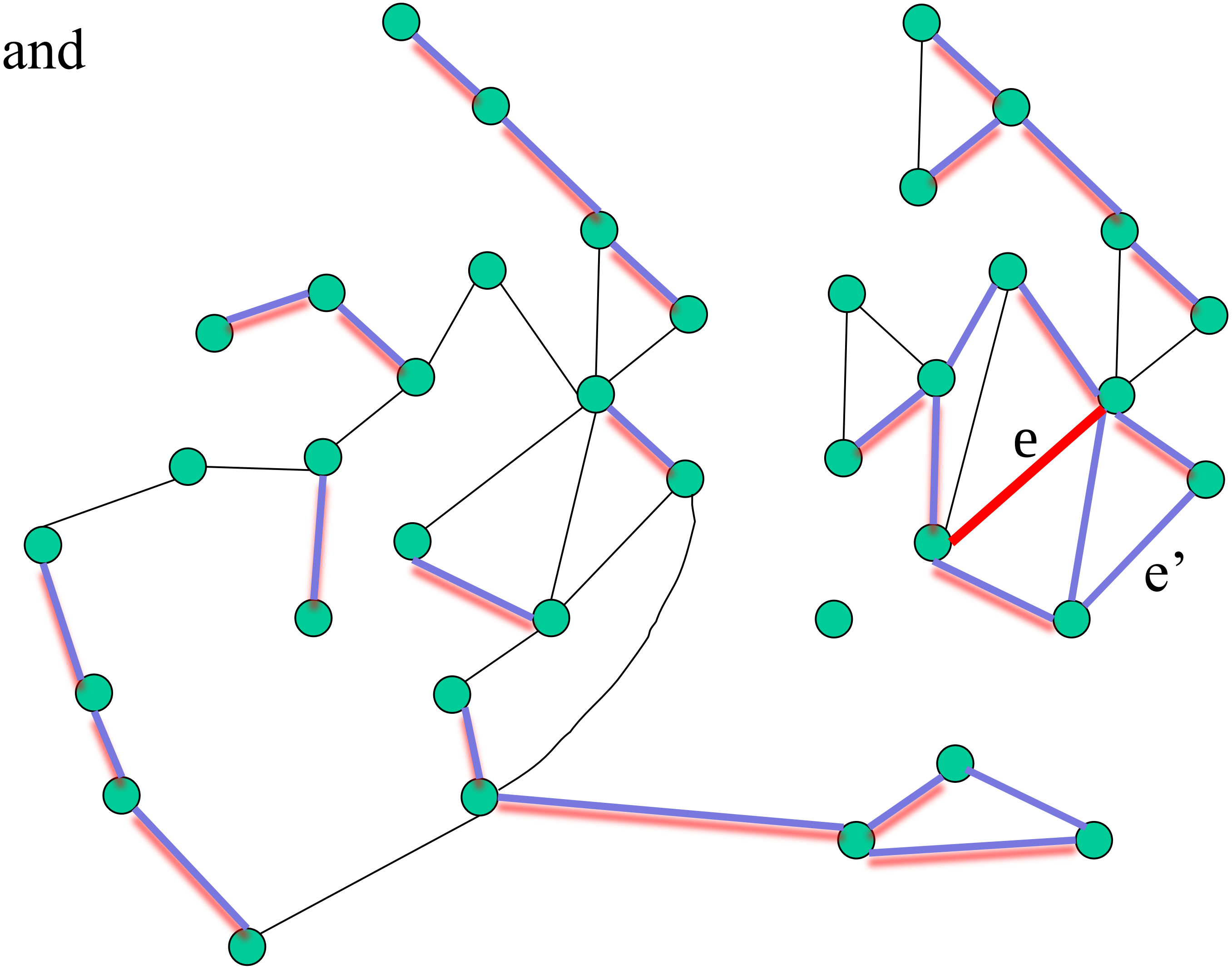
If e closes a cycle find the blue edge e’ of maximum cost on the cycle

$\lambda_e = c(e) - c(e')$ is a critical value

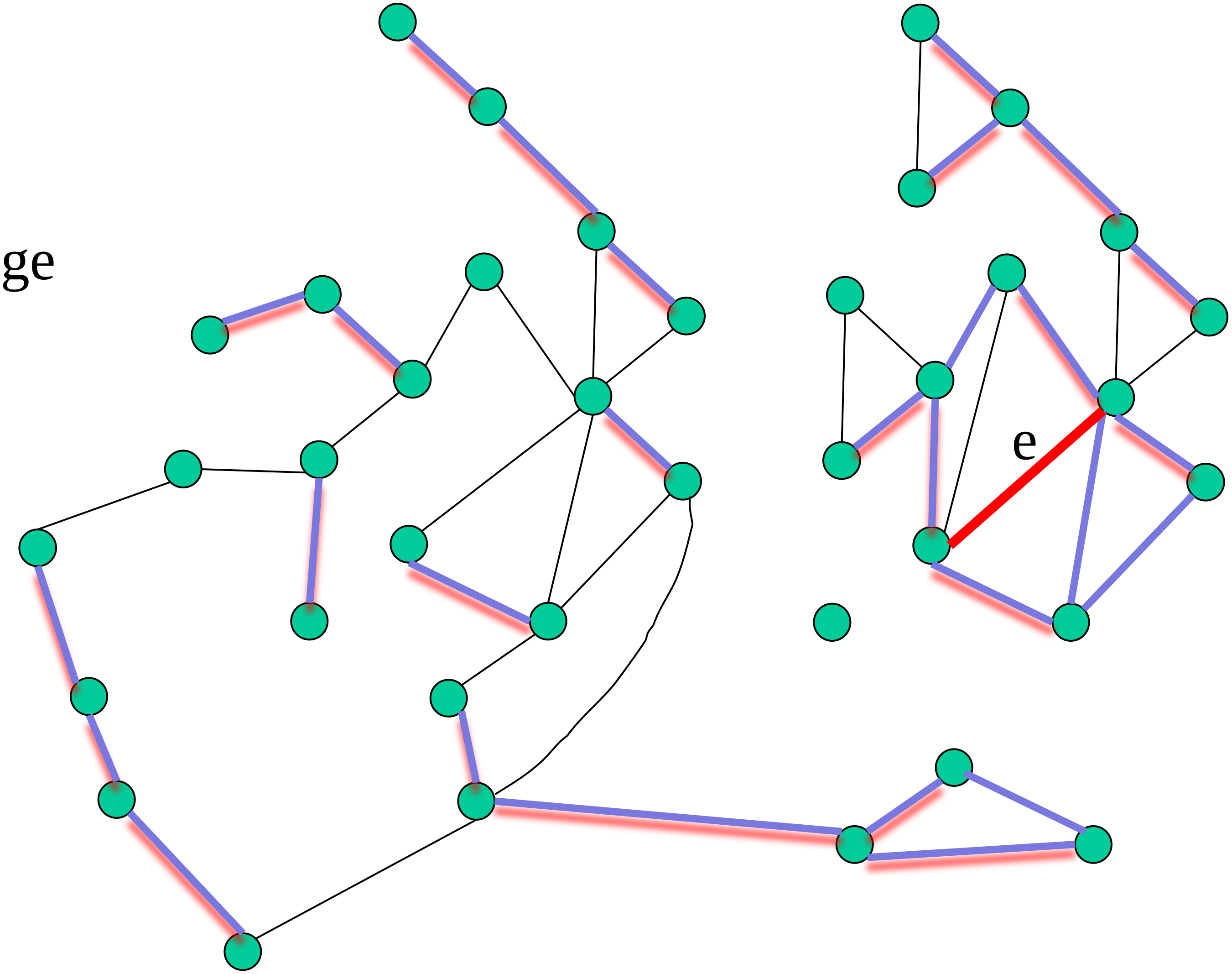
For $\lambda < \lambda_e$ the edge e’ is the smallest crossing it and for $\lambda > \lambda_e$ the edge e is the smallest crossing it



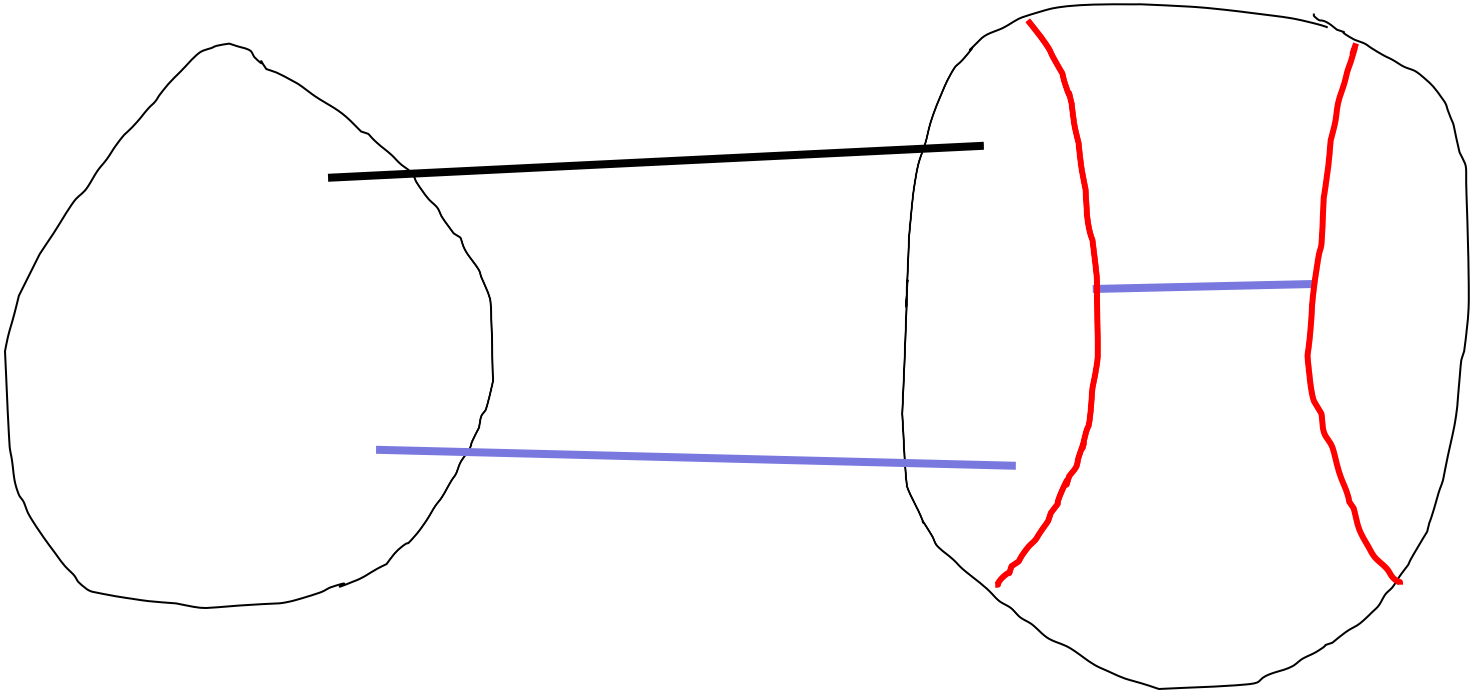
Replace e’ by e and continue



Invariant: each blue edge of the forest is the smallest blue edge crossing the cut that it defines

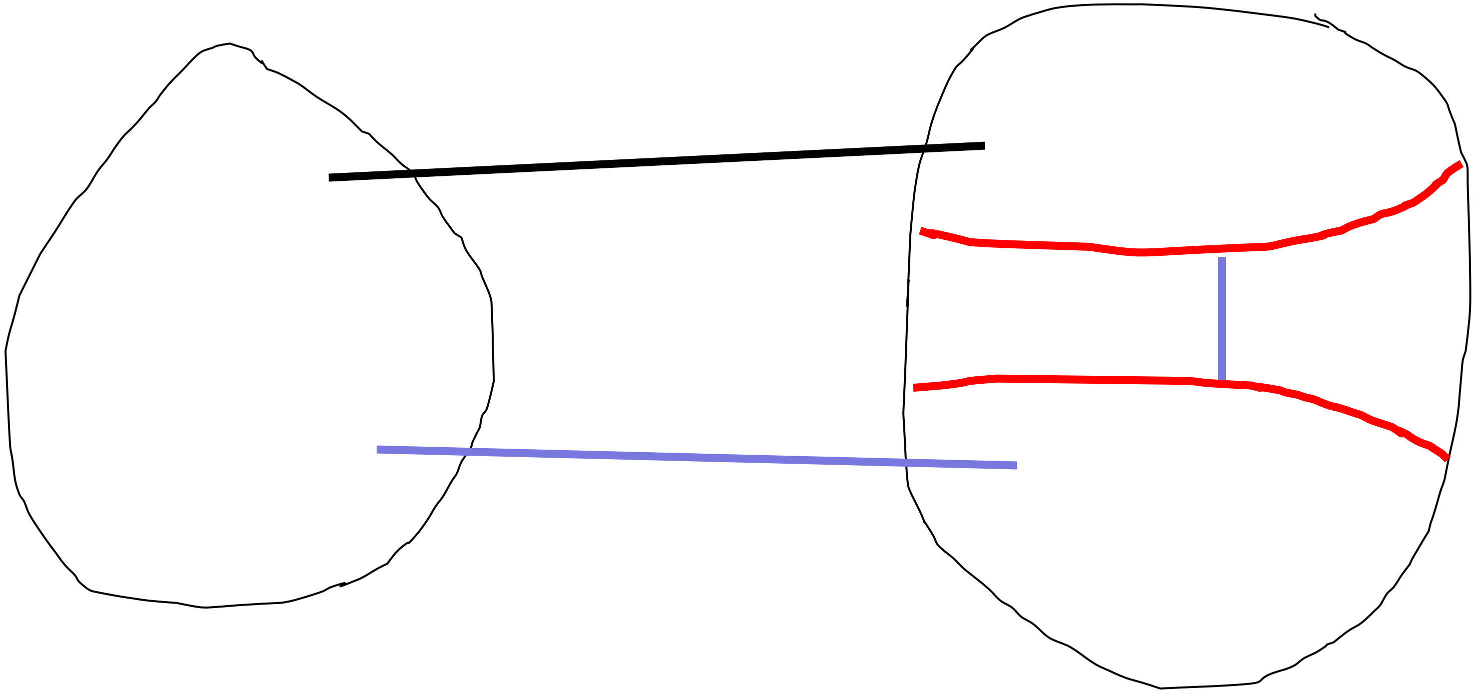


Why is this invariant true ?



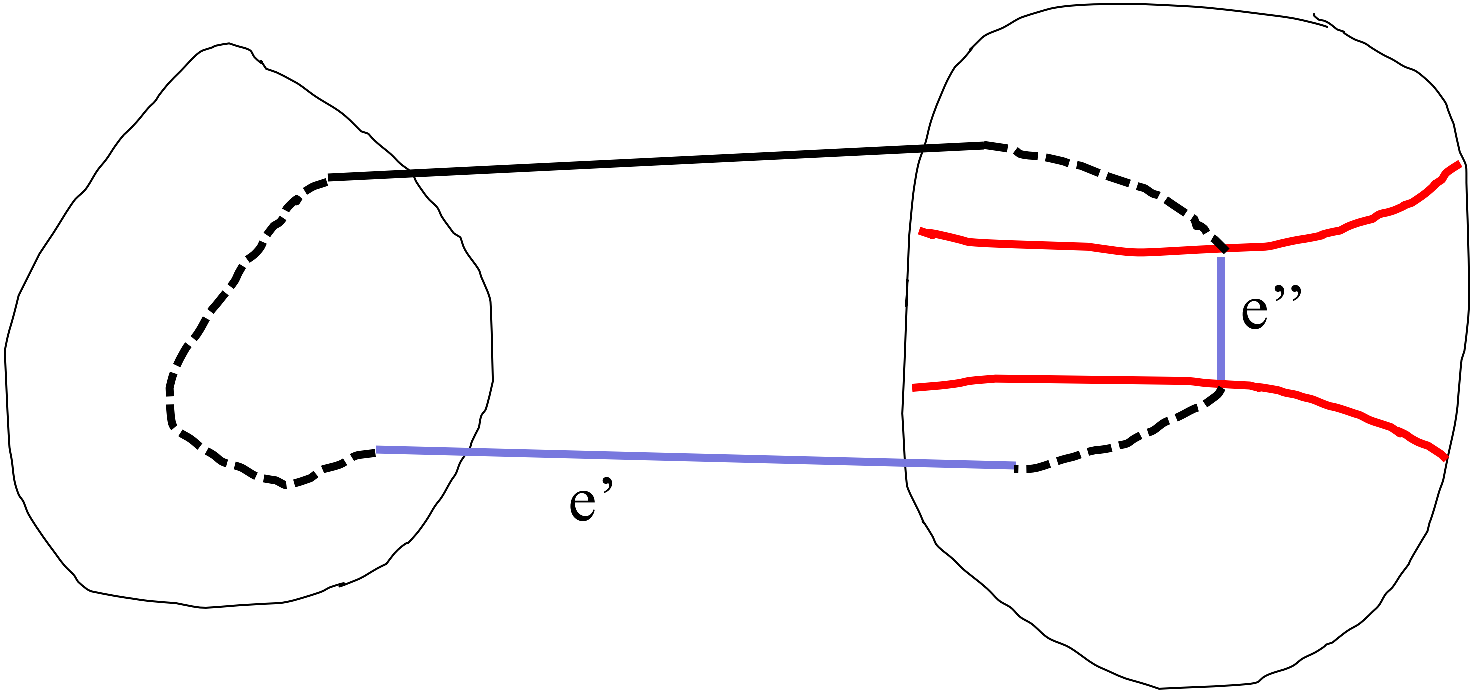
It is clear if the cut does not change

Why is this invariant true ?



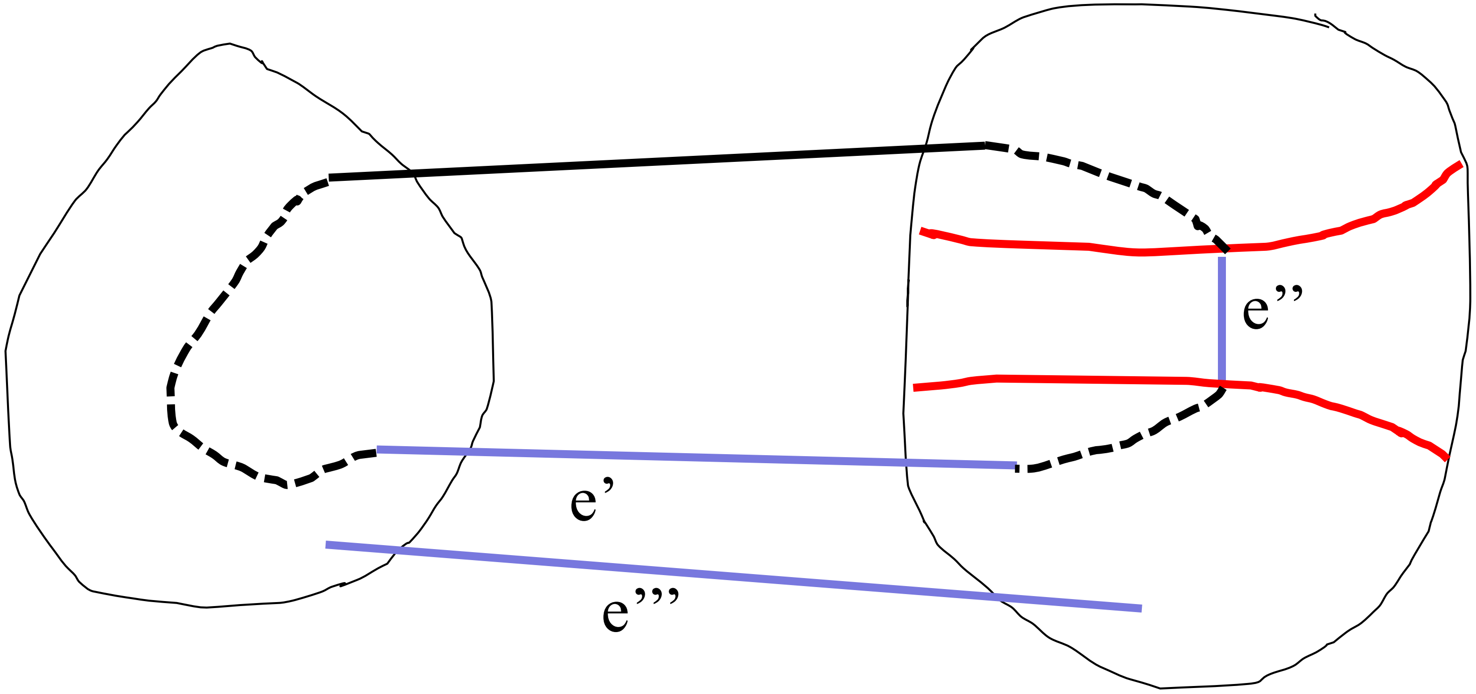
But the cut may change…

Why is this invariant true ?



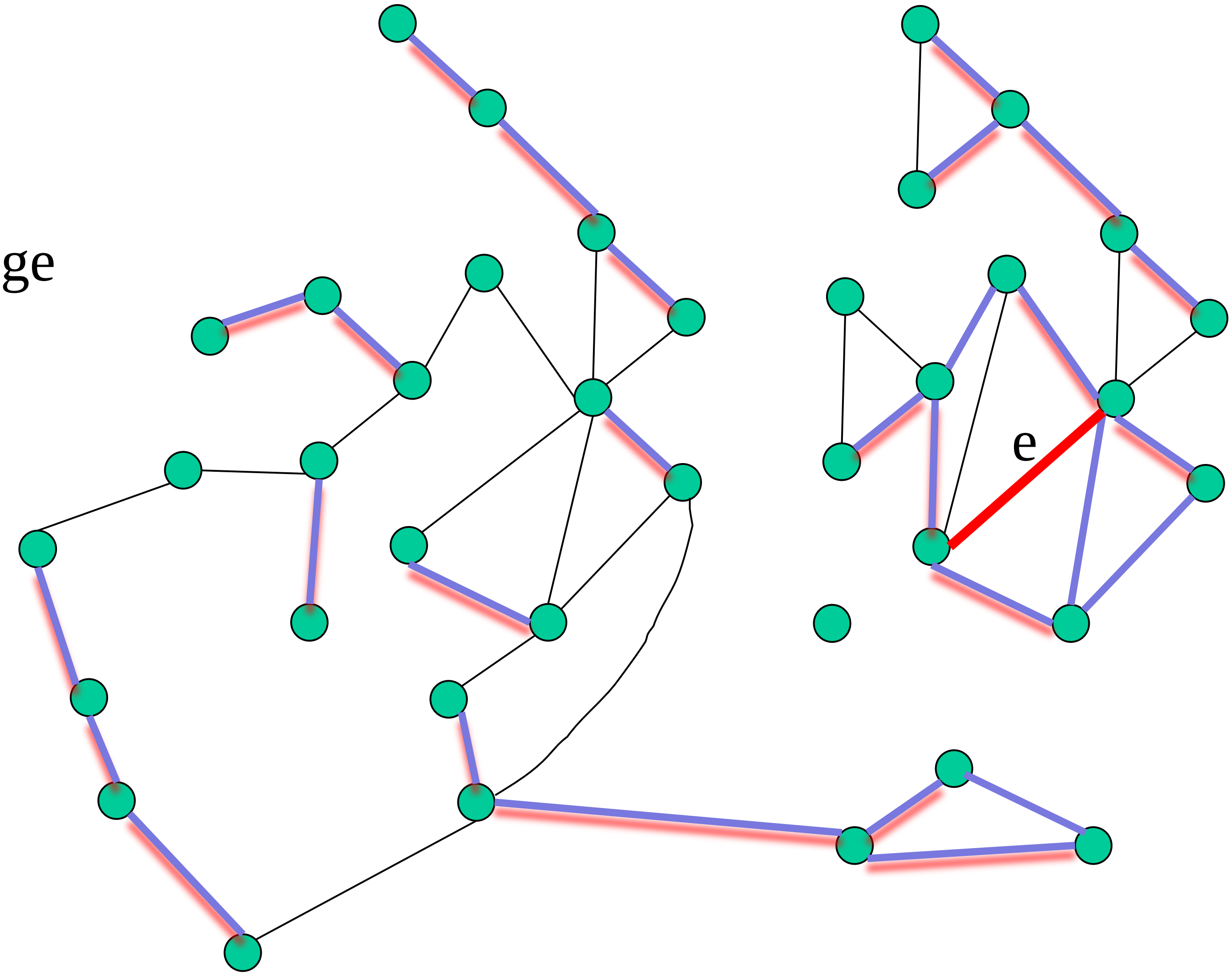
We have c(e'') ≤ c(e')

Why is this invariant true ?



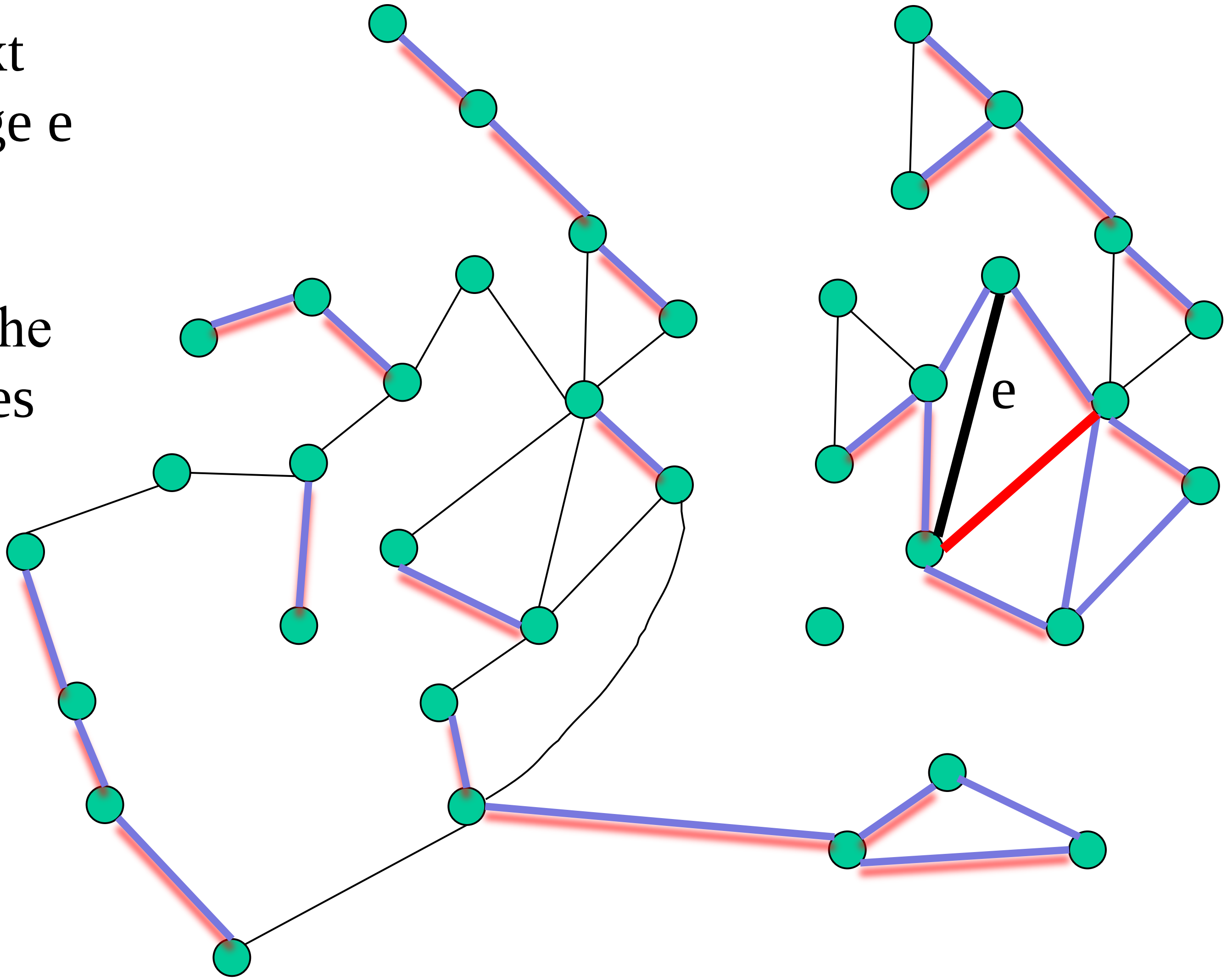
And c(e’’) ≤ c(e’) ≤ c(e’’’)

Invariant: each blue edge of the forest is the smallest blue edge crossing the cut that it defines



Consider the next largest black edge e

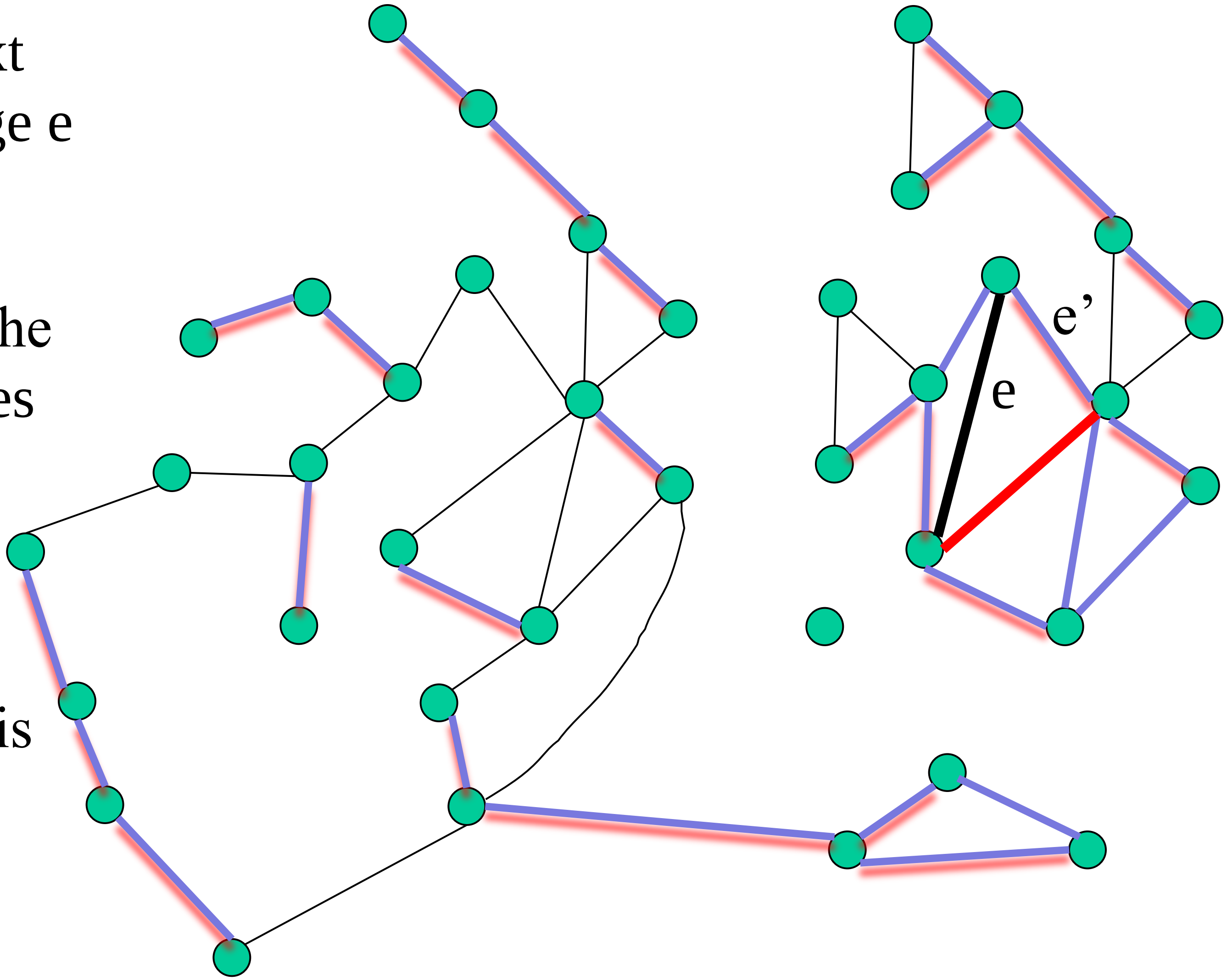
Find the largest blue edge e’ on the cycle that e closes with the current forest



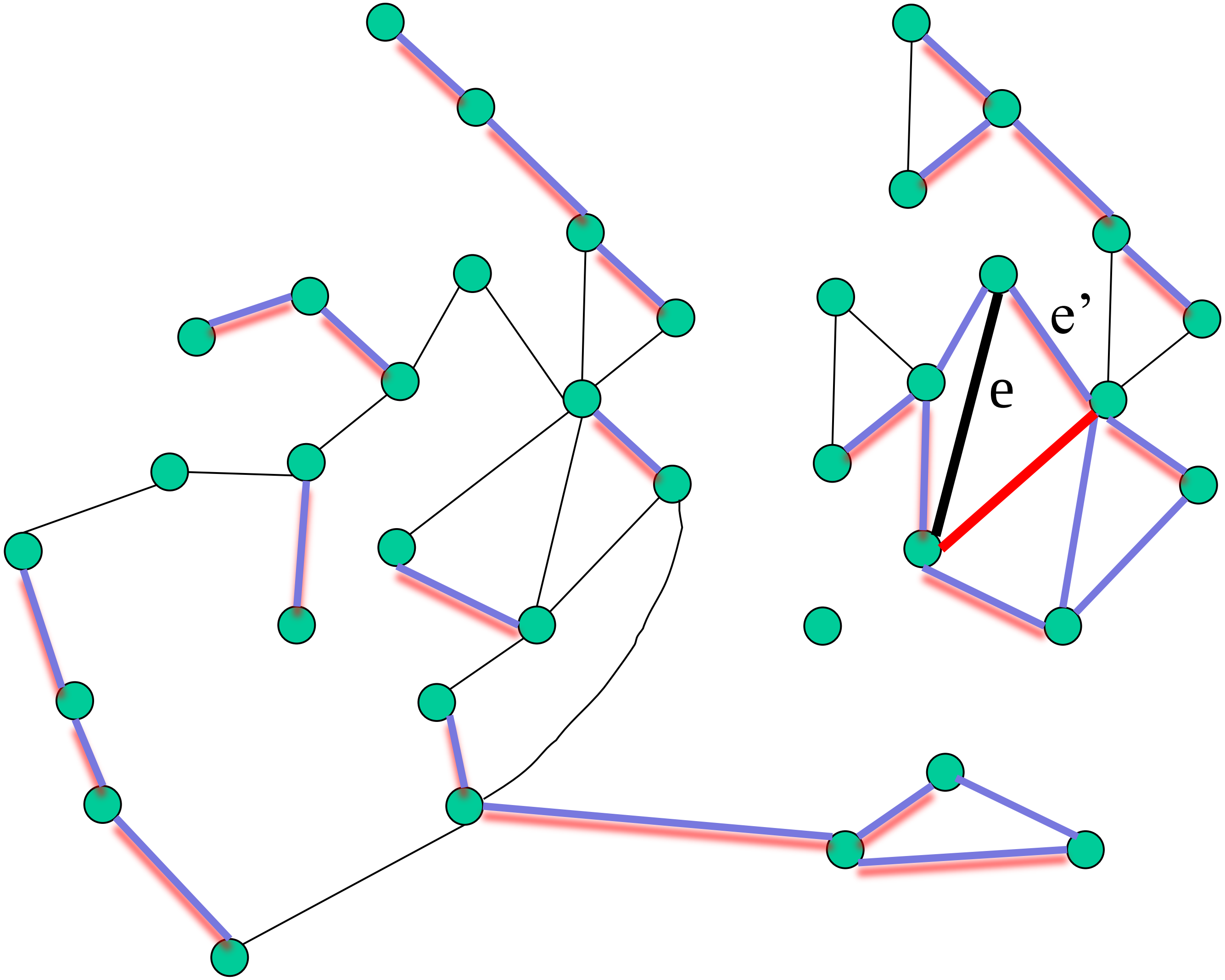
Consider the next largest black edge e

Find the largest blue edge e’ on the cycle that e closes with the current forest

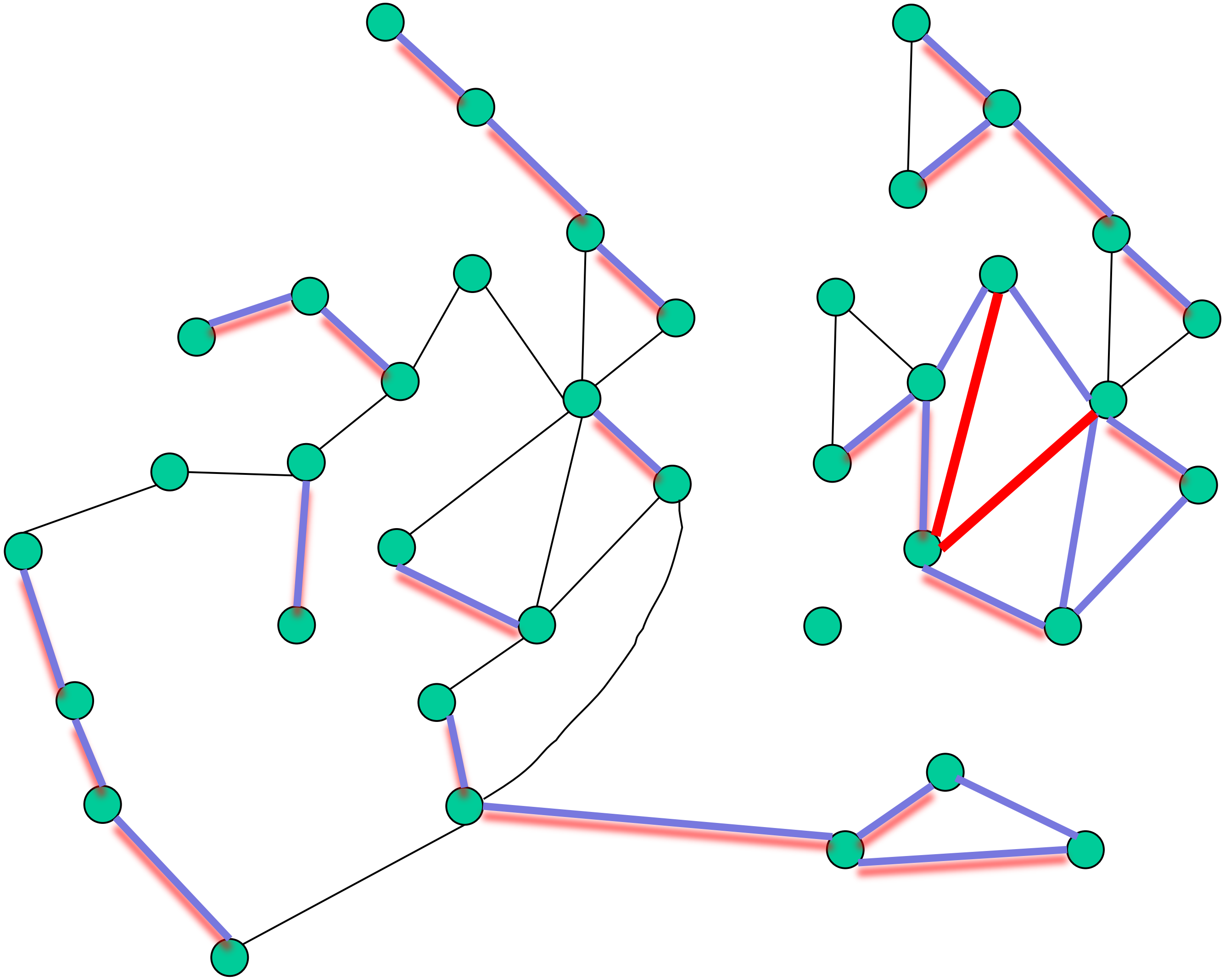
$\lambda_e = c(e) - c(e')$ is a critical value



Replace e’ by e
and continue



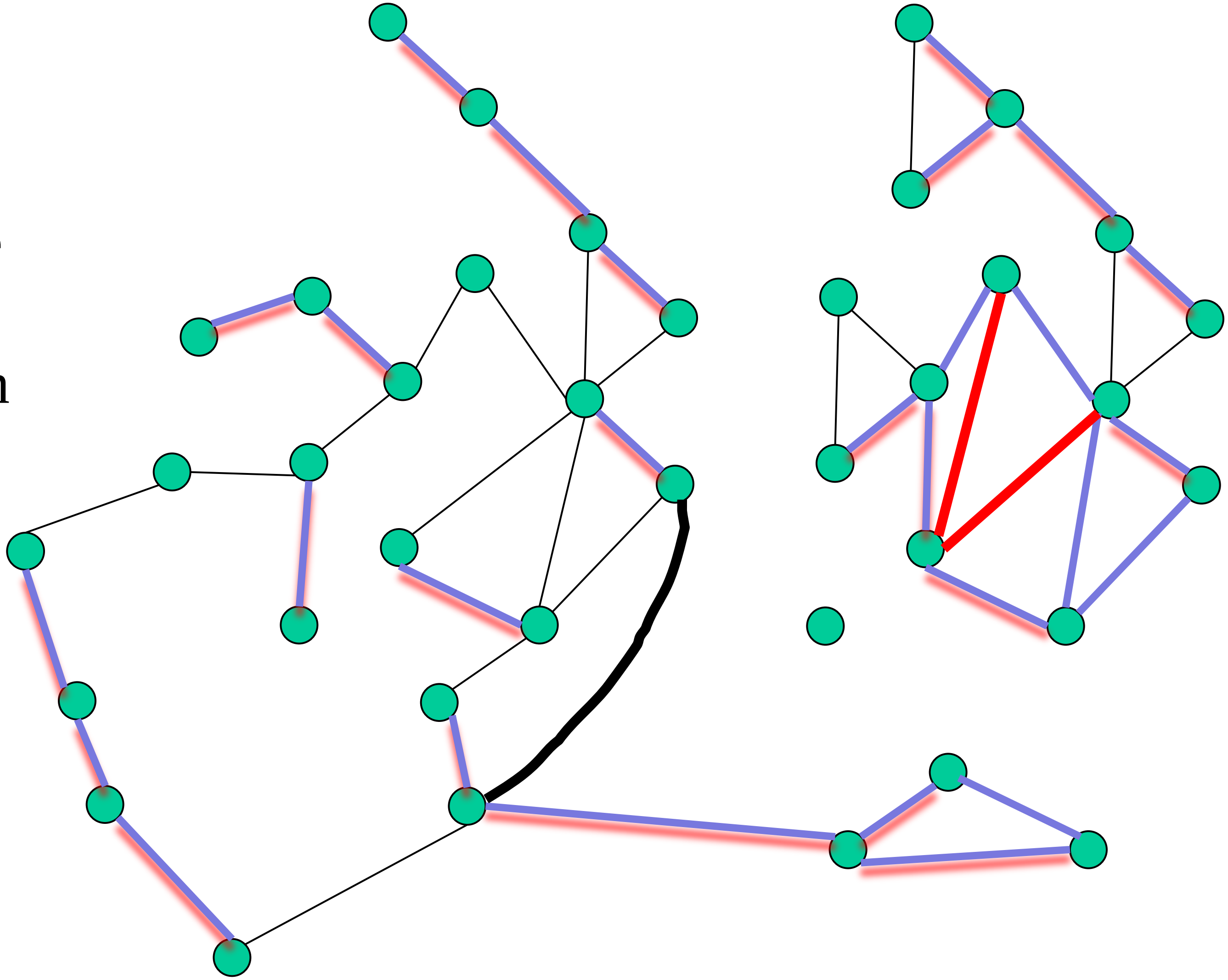
Replace e’ by e and continue



Replace e’ by e and continue

If the black edge connects two components then it appears in any spanning forest

Add it and continue



- If a black edge closes a cycle of black edges just discard it

Summary

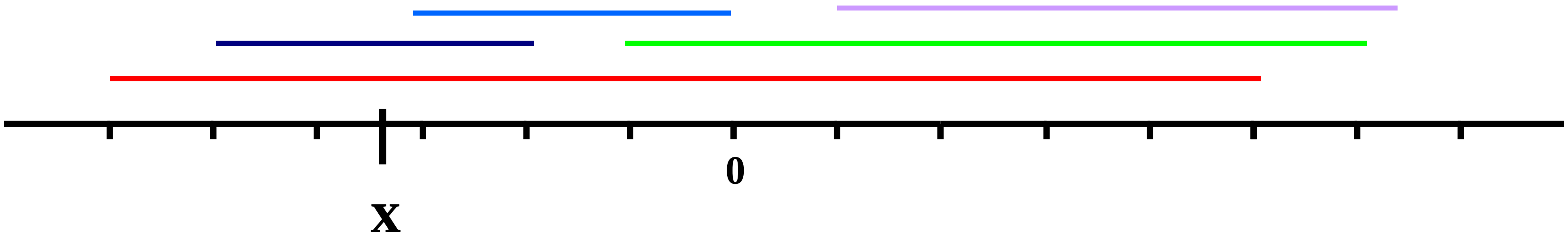
- We identify a set of blue edges that are never in the tree
- We identify a set of black edges always in the tree (say b many)
- Other edges are partitioned into black-blue pairs each with an associated critical λ
- Sort the pairs by λ
- If you want b + z black edges then take the black edges of the first z pairs and the blue edges of the rest
- $O(m\log(n))$ total time

Application (3)

1D Range reporting

Given a set of intervals S on the line, preprocess them to build a structure that allows efficient queries of the from:

Given a point x find all intervals containing it.

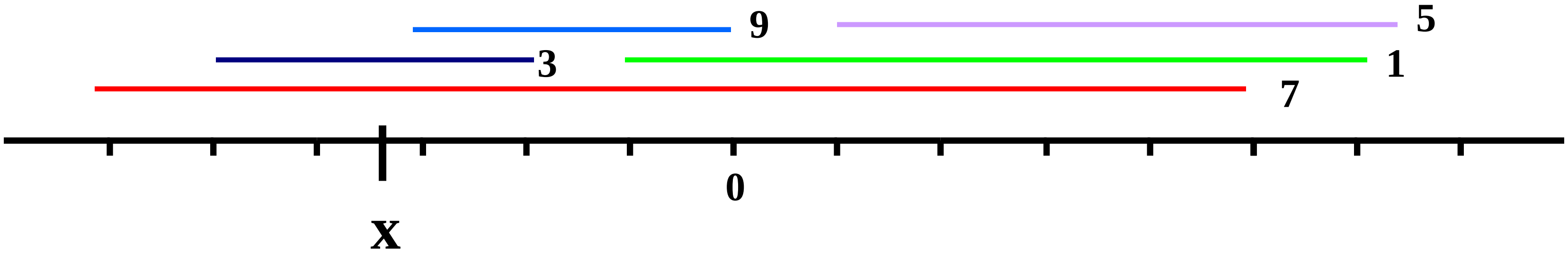


Dynamic range reporting + priorities

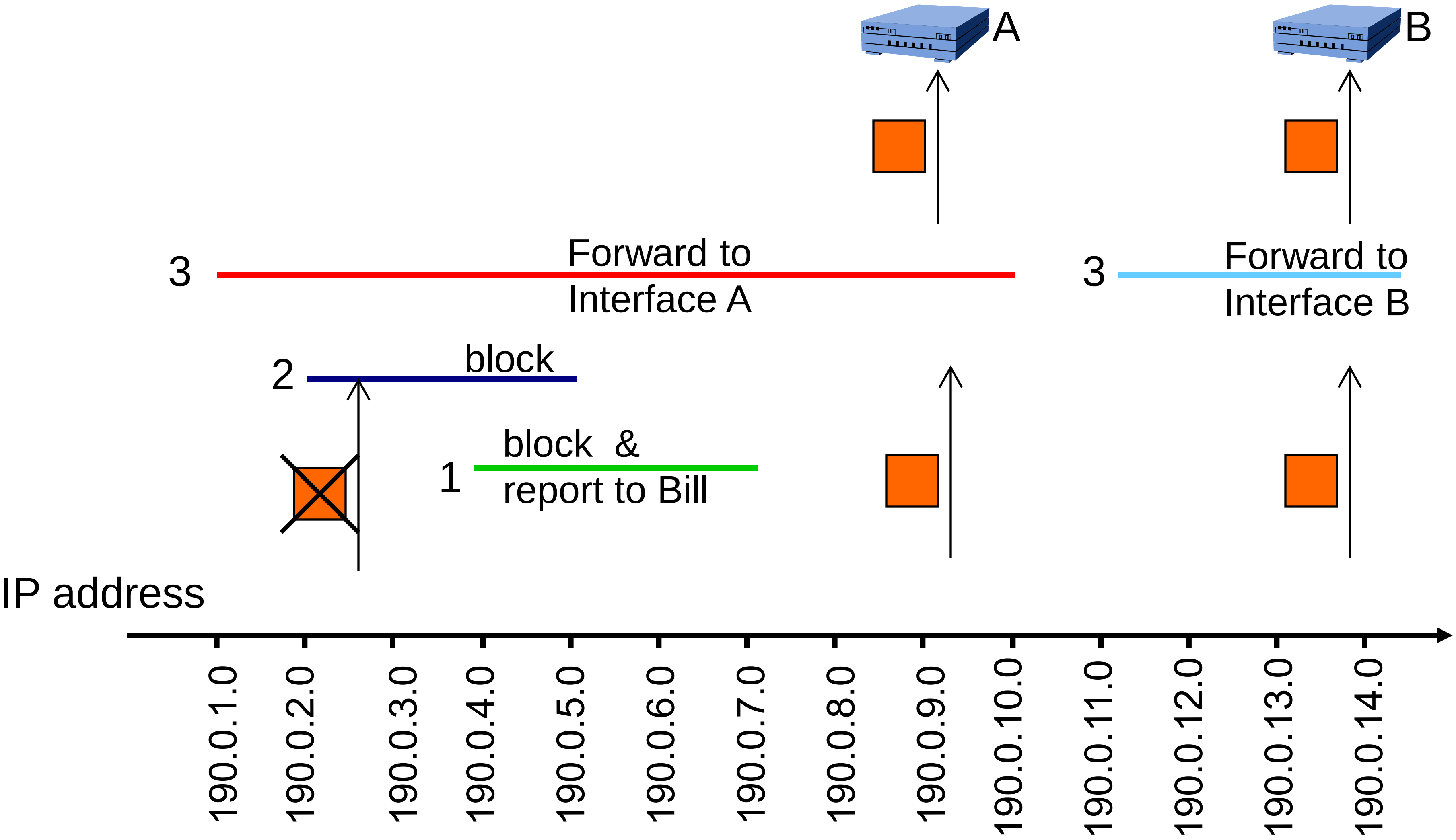
Given a set of intervals S on the line, each with priority assigned to it, build a structure that allows efficient queries of the from:

Given a point x find interval with minimum priority containing it.

Updates – insert or delete an interval



Motivation – Packet classification



A

B

3 Forward to Interface A

3 Forward to Interface B

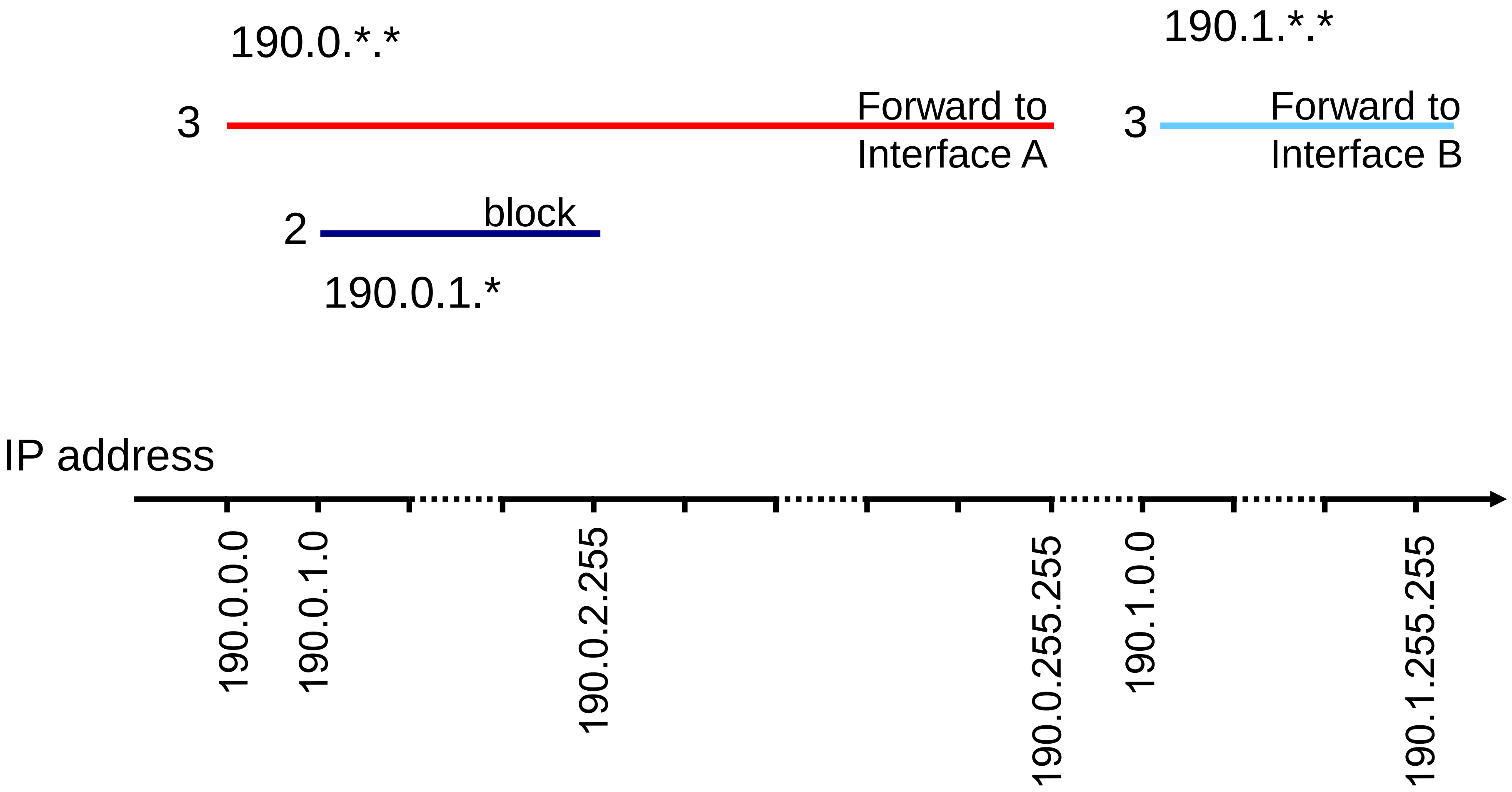
2 block

1 block & report to Bill

IP address

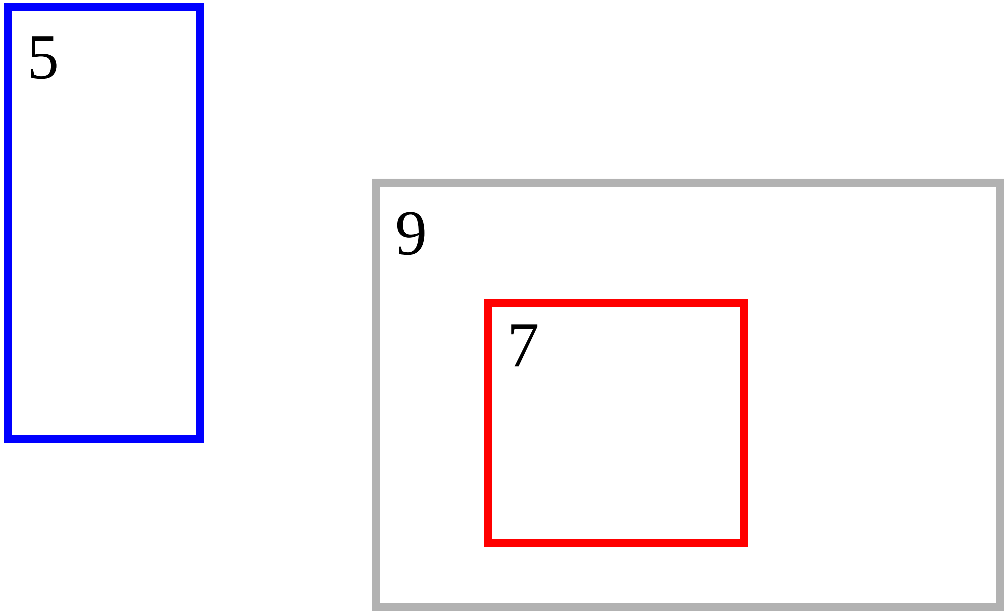
190.0.1.0
190.0.2.0
190.0.3.0
190.0.4.0
190.0.5.0
190.0.6.0
190.0.7.0
190.0.8.0
190.0.9.0
190.0.10.0
190.0.11.0
190.0.12.0
190.0.13.0
190.0.14.0

Nested intervals, IP prefixes

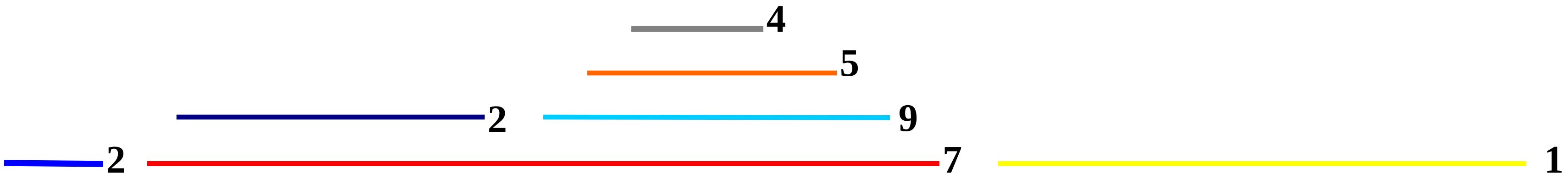


Extension to 2D

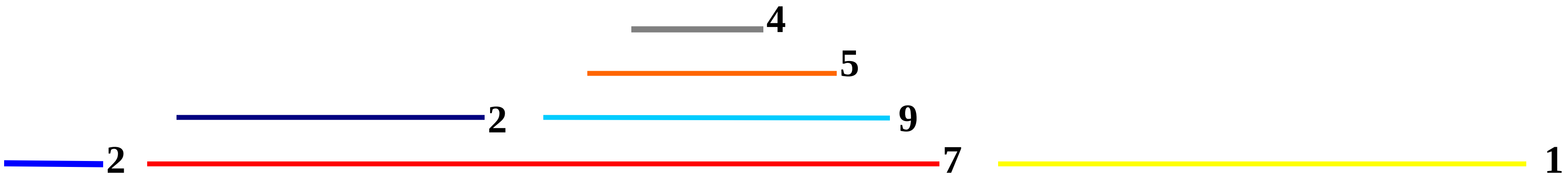
- Query = point in R^2
  - (Sender IP, receiver IP)
- interval = rectangle with priority



One dimensional data structure for nested intervals

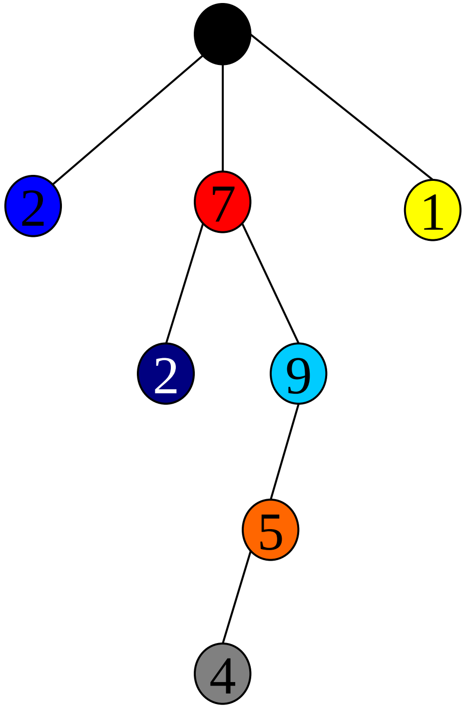


Nested Intervals

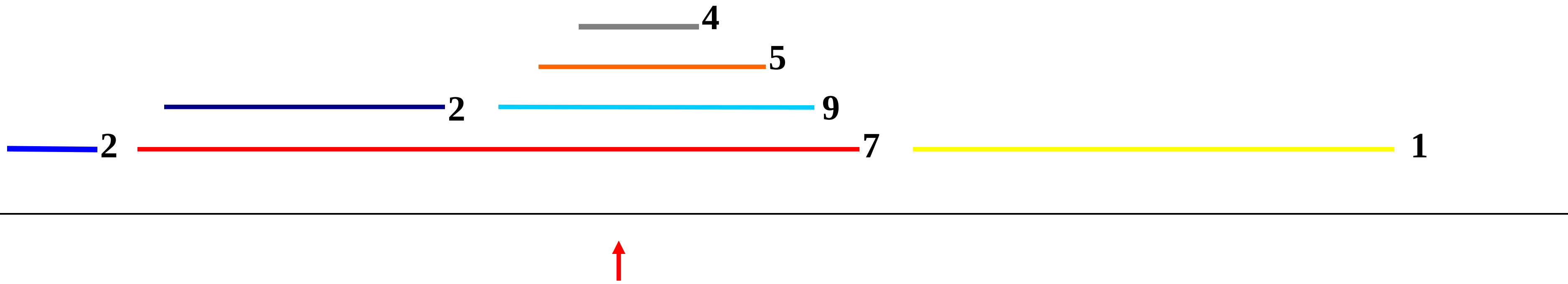


Containment tree:

The parent of interval *v* is the smallest interval containing *v*



Nested Intervals

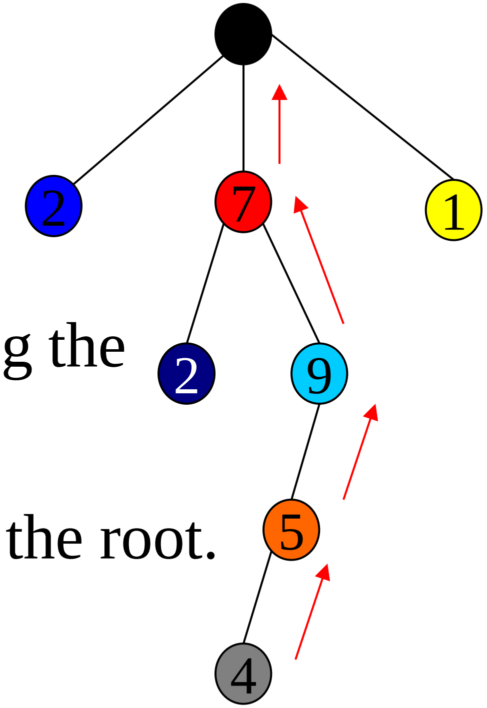


<u>Query:</u>

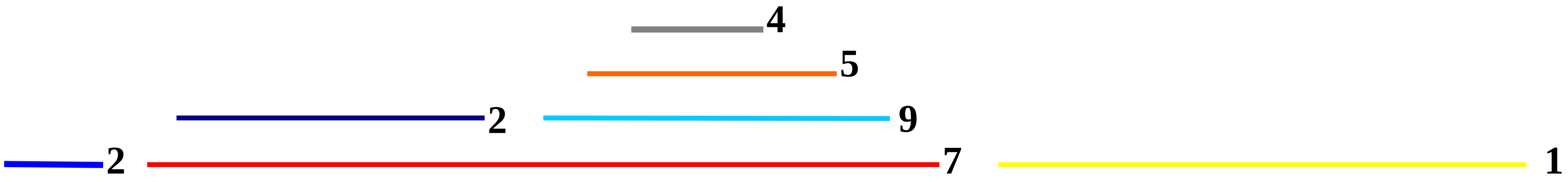
Starting node *s* = smallest interval containing the query point

Relevant priorities are on the path from *s* to the root.

Problem: path may be long…

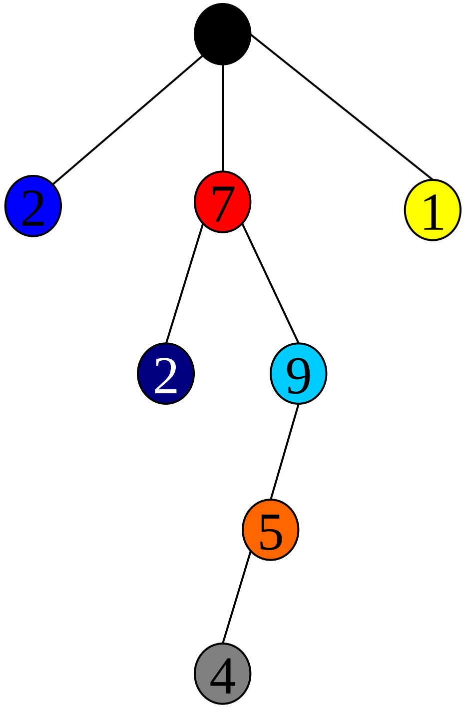


Hey, dynamic trees know how to do that

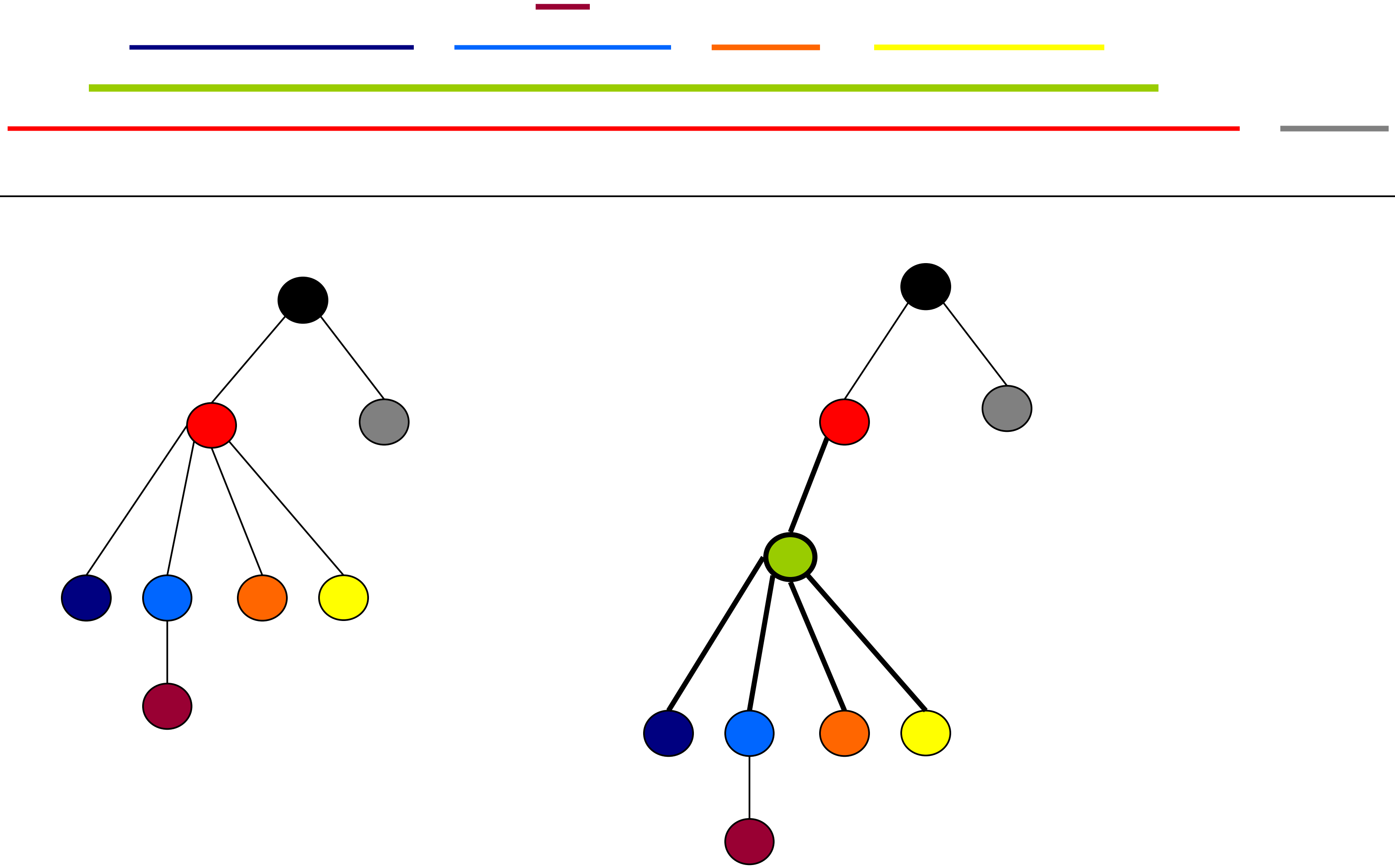


We can use a dynamic tree to represent the containment tree.

Query ➔ mincost()

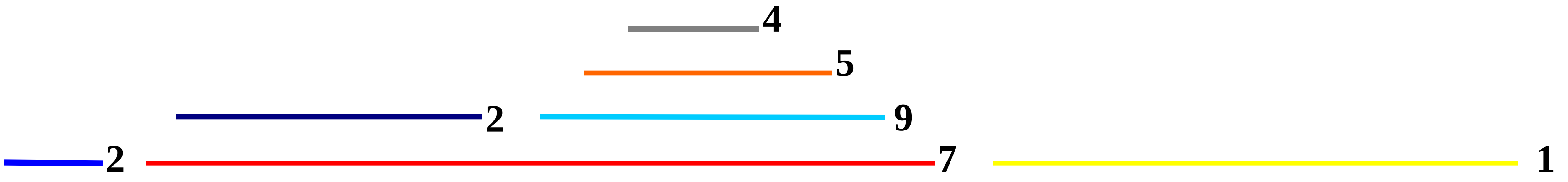


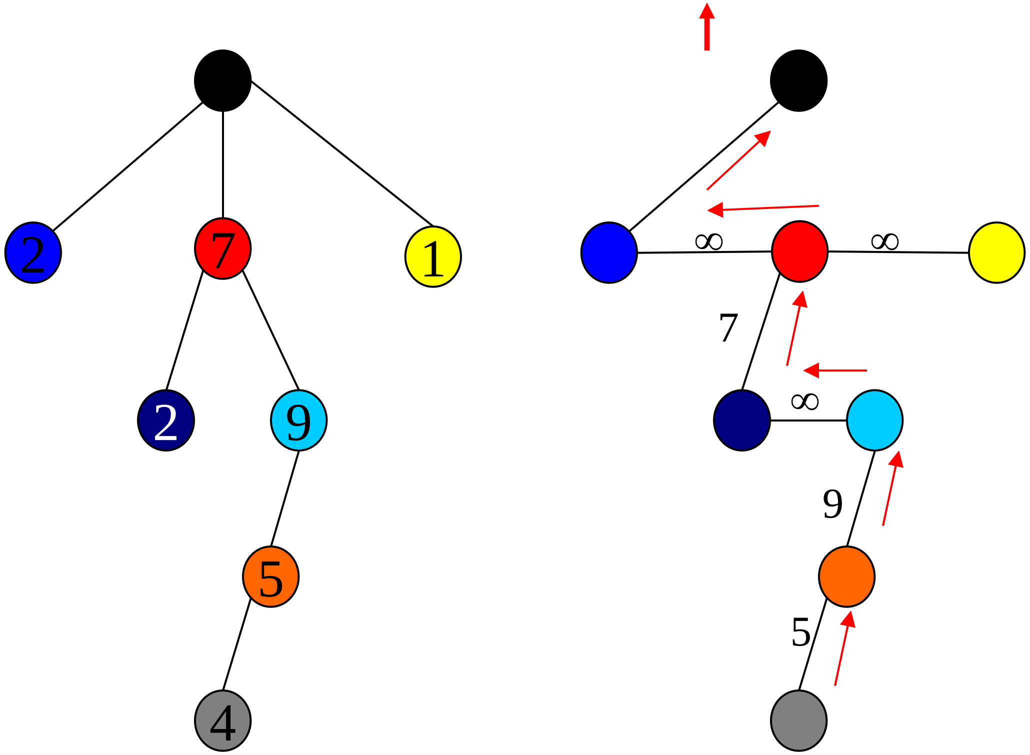
Insert



Problem: Updates => Many cuts & links

Binarization





Node v => node v

Leftmost child of v => Left child of v

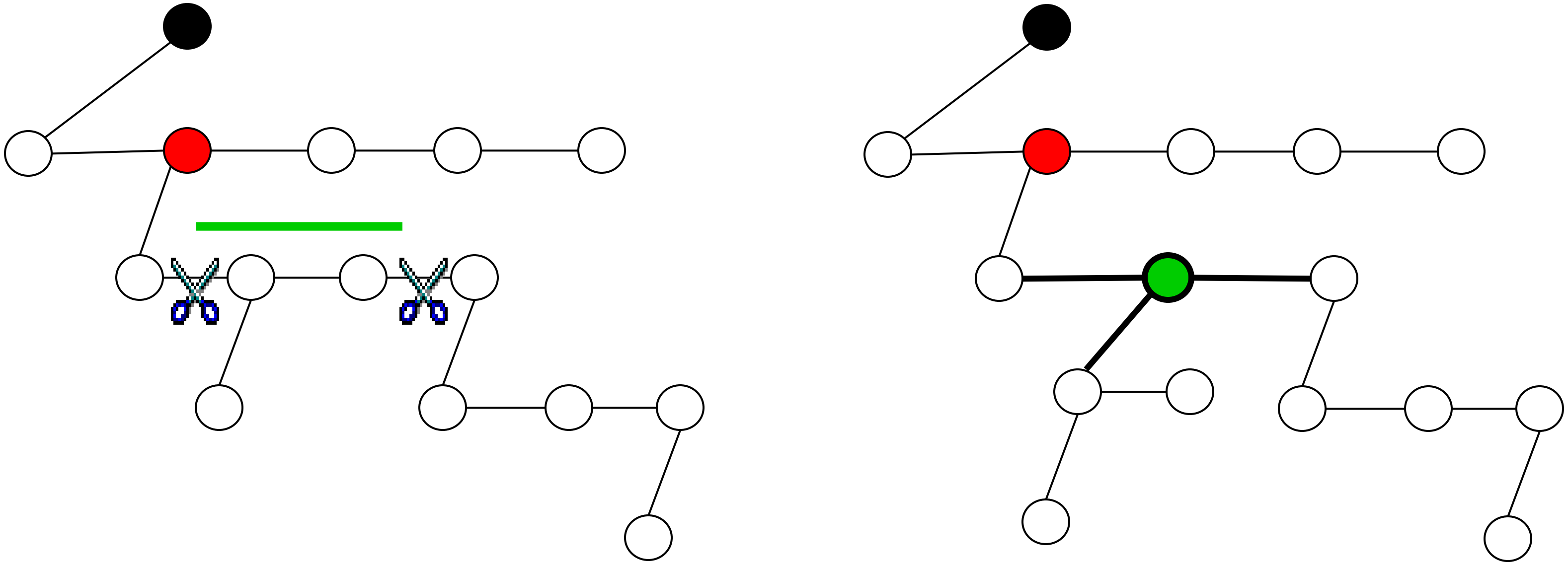
Any other child of v => right child of its left sibling

Adjust costs:

Left edge => priority of parent

Right edge => ∞

Insert (Cont.)



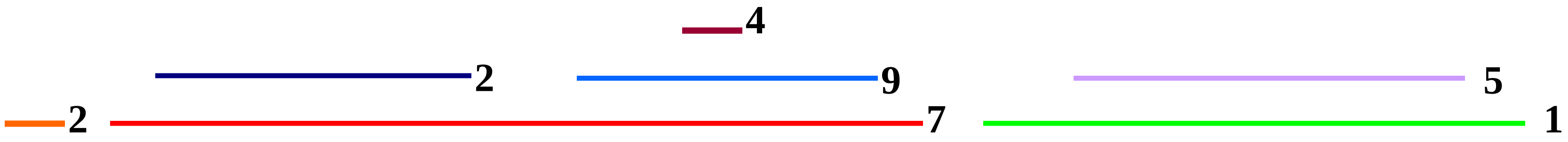
Constant number of links and cuts

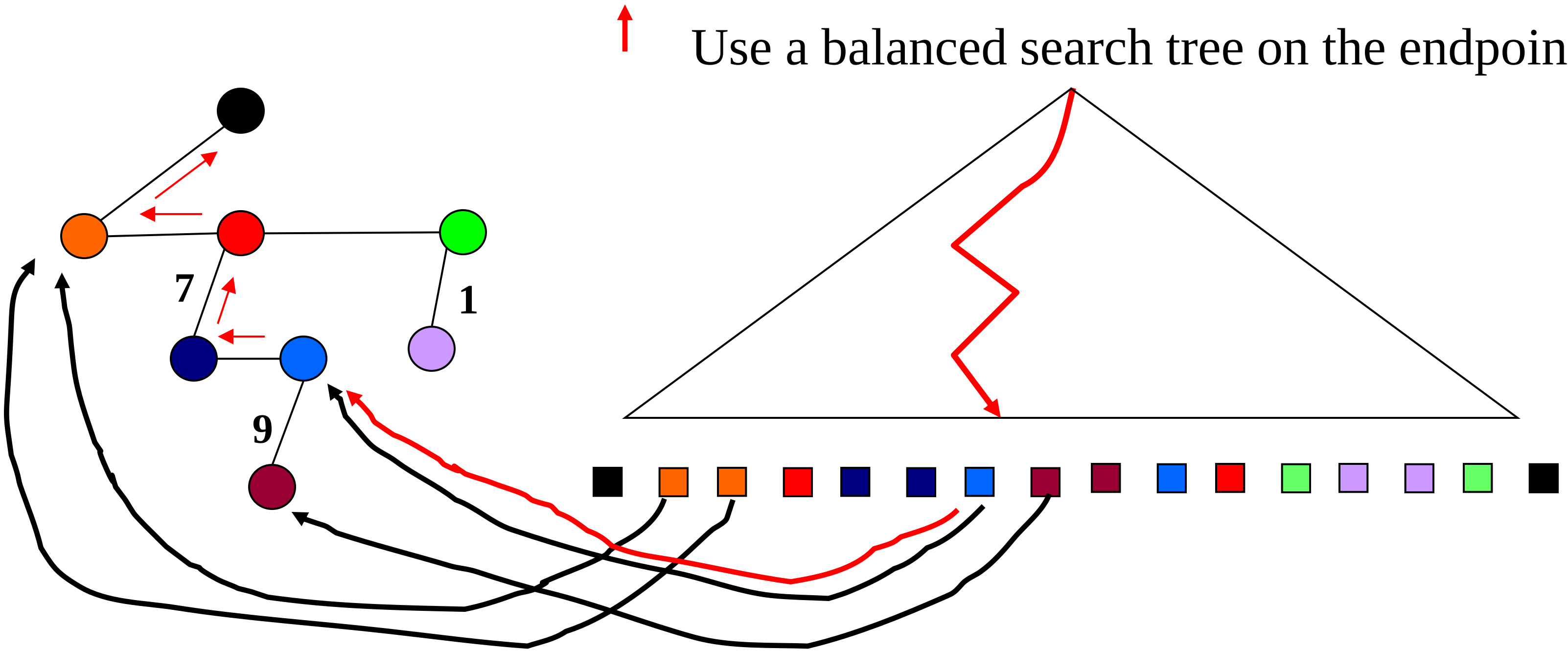
Summary

- Containment tree C

  Query = min cost on path from starting point to – root
- Represent C by binarized version B
- Represent B by dynamic tree D
- How do you find the point to start the query ?
- How do you find the edges to cut ?

How do you start the query ?

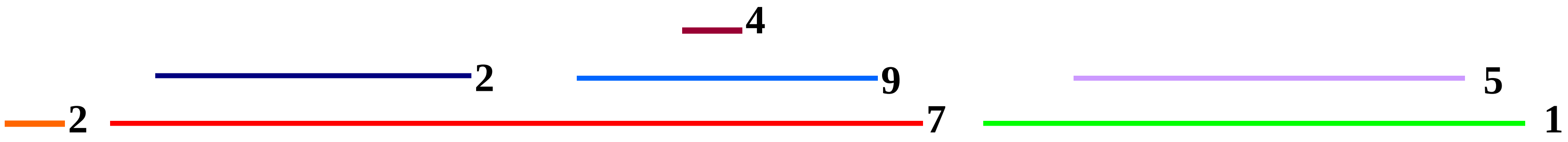


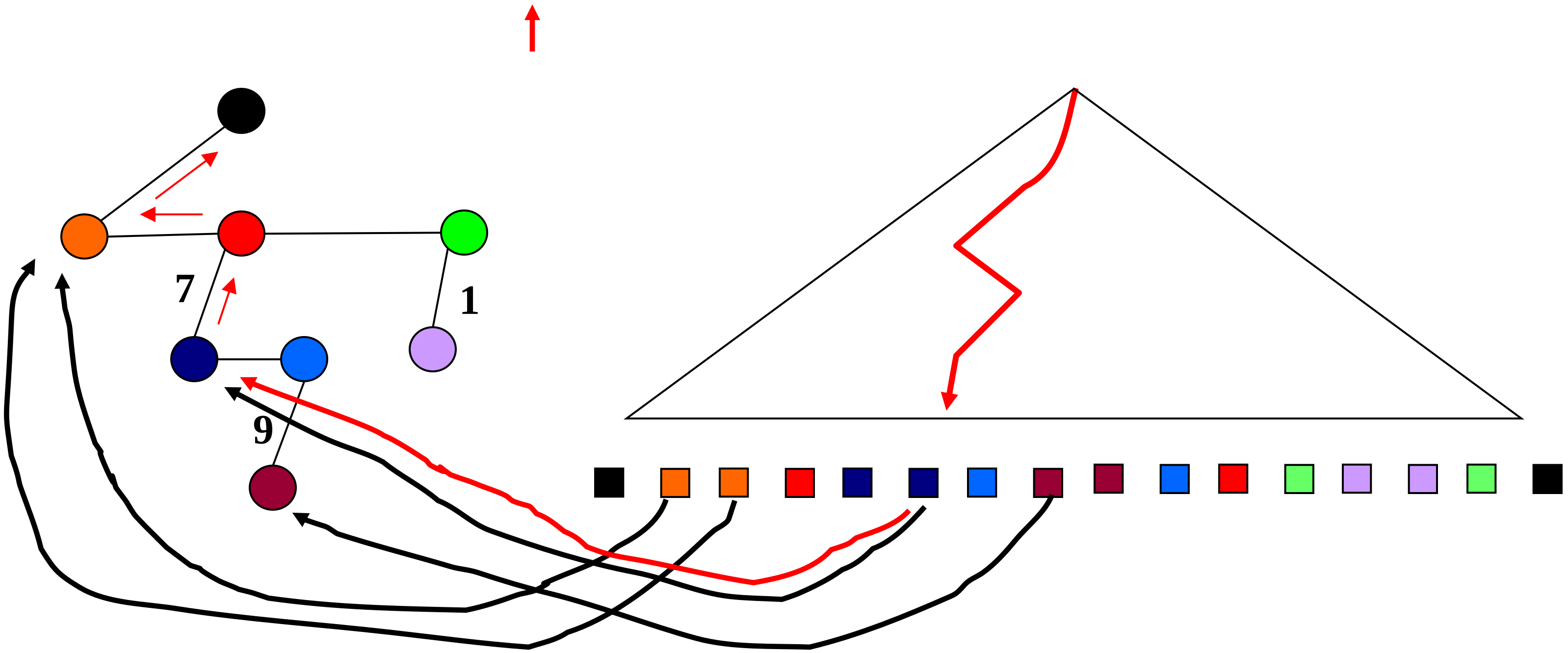


Use a balanced search tree on the endpoints

Min(Mincost(●),pri(●))

query (cont)





Mincost(●)

Lets implement this data type

maketree(v)

w = findroot(v)

(w,c) = mincost(v)

addcost(v,c)

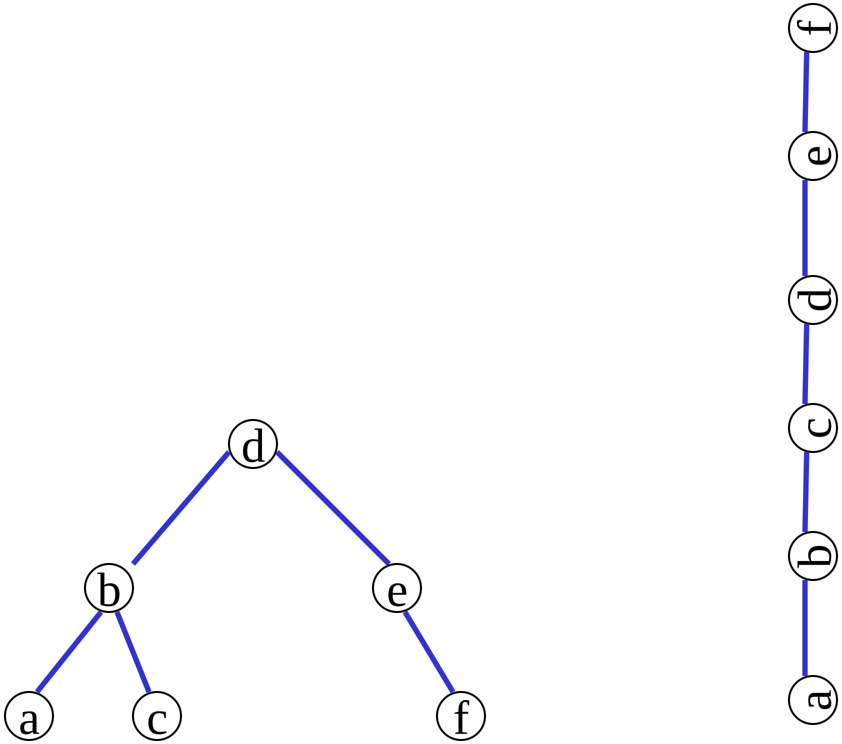
link(v,w,c)

cut(v)

evert(v)

Simple case -- paths

Assume for a moment that each tree T in the forest is a path.
We represent it by a virtual tree which is a simple splay tree.



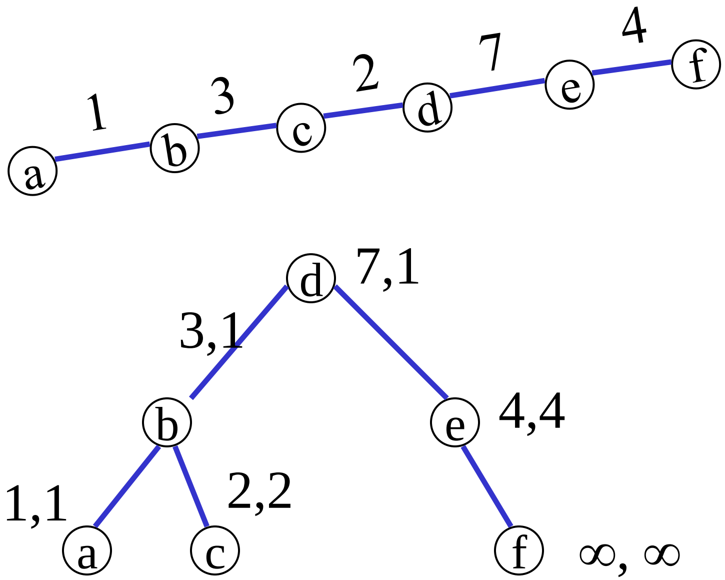
Findroot(v)

Splay at v, then follow right pointers until you reach the last vertex w on the right path. Return w and splay at w.

Mincost(v)

With every vertex x we record cost(x) = the cost of the edge (x,p(x))

We also record with each vertex x mincost(x) = minimum of cost(y) over all descendants y of x.



Mincost(v)

Splay at v and use mincost values to search for the minimum

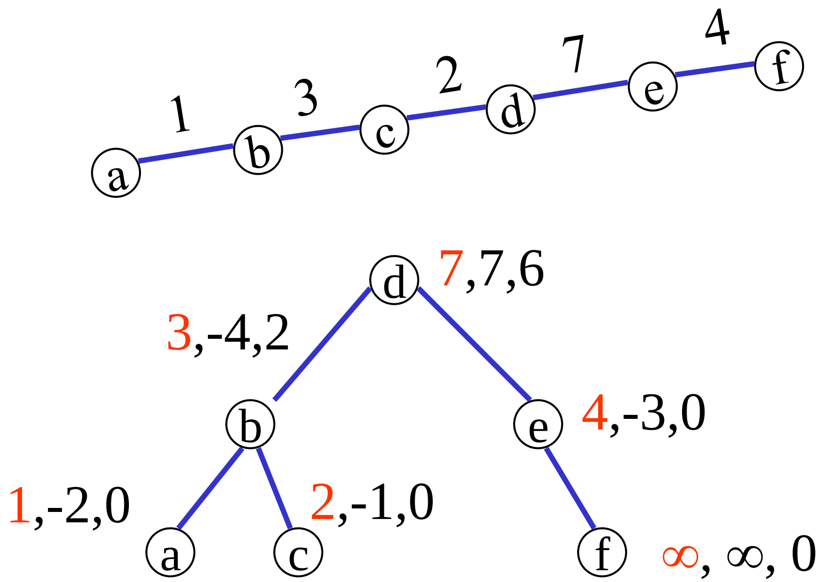
Notice: we need to update mincost values as we do rotations.

Addcost(v,c)

Rather than storing cost(x) and mincost(x) we will store

Δcost(x) = cost(x) - cost(p(x))

Δmin(x) = cost(x) - mincost(x)



Addcost(v,c) :
Splay at v,
Δcost(v) += c
Δcost(left(v)) -= c

similarly update
Δmin

Addcost(v,c) (cont)

Notice that now we have to update Δcost(x) and Δmin(x) through rotations

$\Delta cost'(v) = \Delta cost(v) + \Delta cost(w)$

$\Delta cost'(w) = -\Delta cost(v)$

$\Delta cost'(b) = \Delta cost(v) + \Delta cost(b)$

Addcost(v,c) (cont)

Update Δmin:

$\Delta\text{min}'(w) = \max\{0, \Delta\text{min}(b) - \Delta\text{cost}'(b), \Delta\text{min}(c) - \Delta\text{cost}(c)\}$

$\Delta\text{min}'(v) = \max\{0, \Delta\text{min}(a) - \Delta\text{cost}(a), \Delta\text{min}'(w) - \Delta\text{cost}'(w)\}$

Link(v,w,c), cut(v)

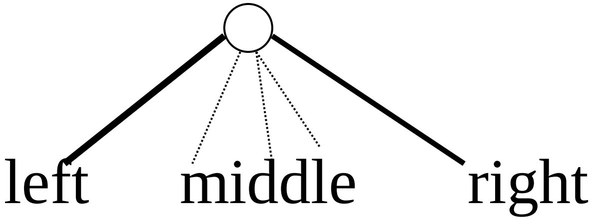
Translate directly into catenation and split of splay trees if we talk about paths.

Lets do the general case now.

The virtual tree

- We represent each tree T by a virtual tree V.

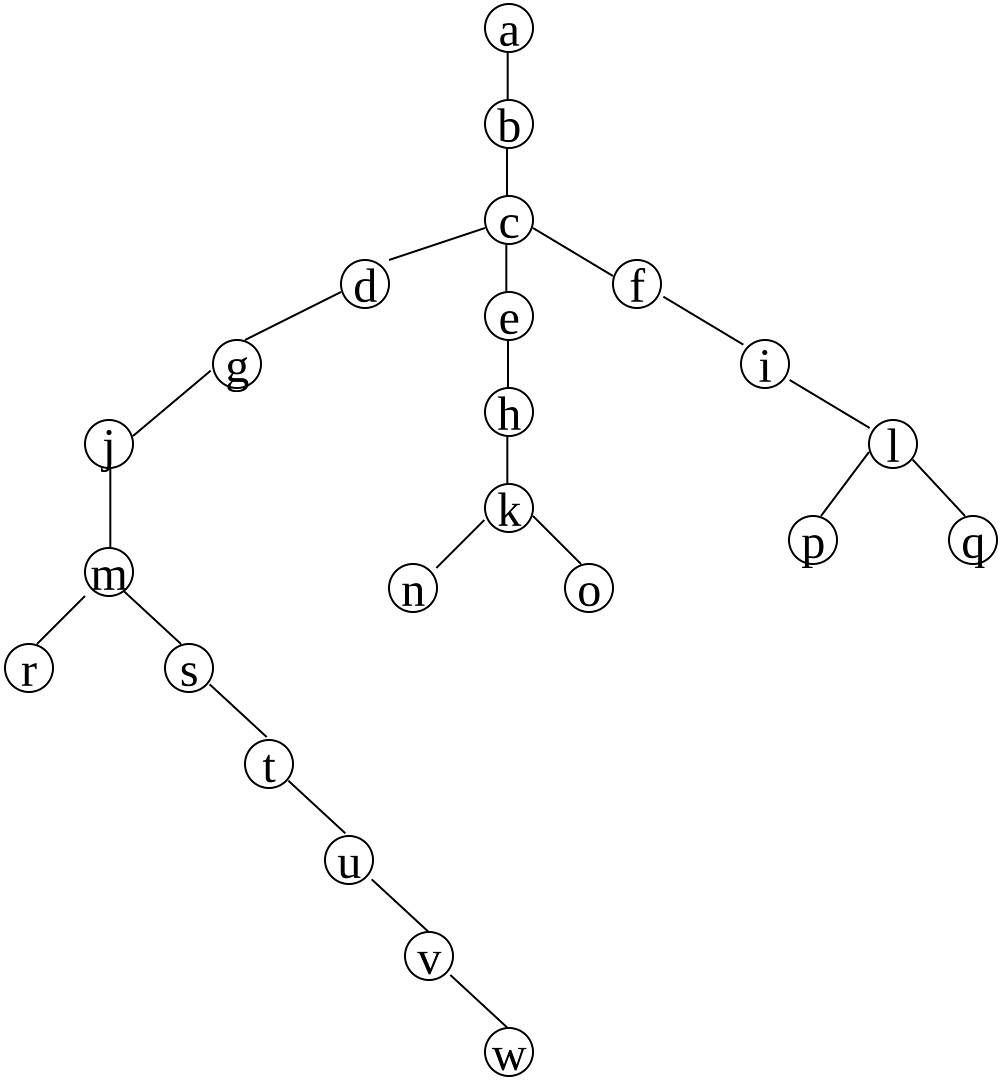
The virtual tree is a binary tree with middle children.



Think of V as partitioned into solid subtrees connected by dashed edges

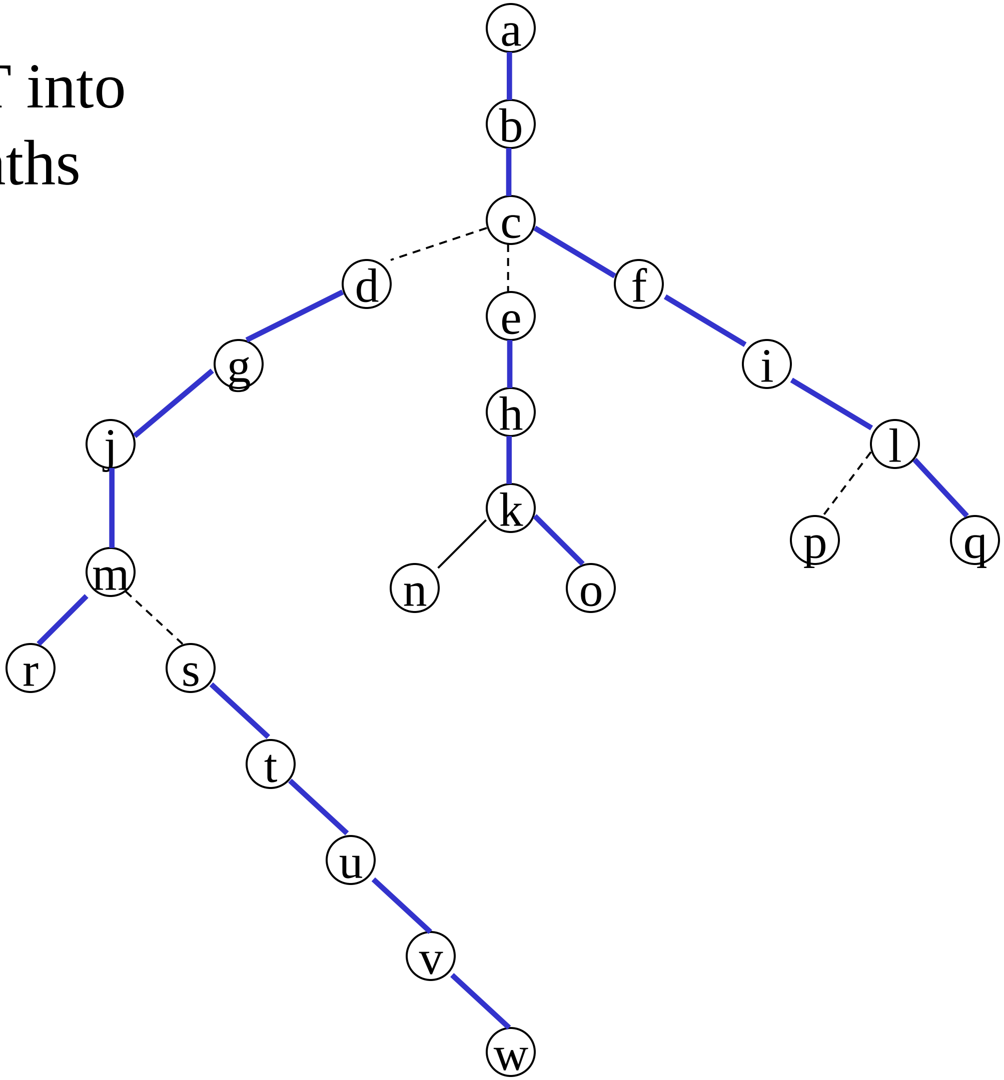
What is the relation between V and T ?

Actual tree



Path decomposition

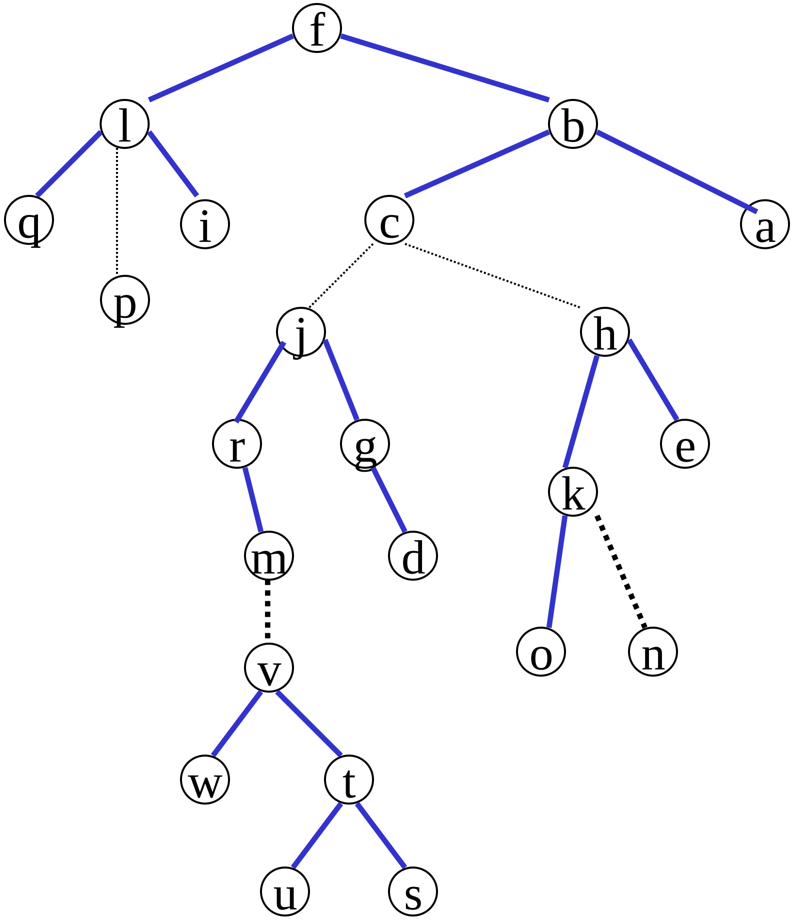
Partition T into disjoint paths



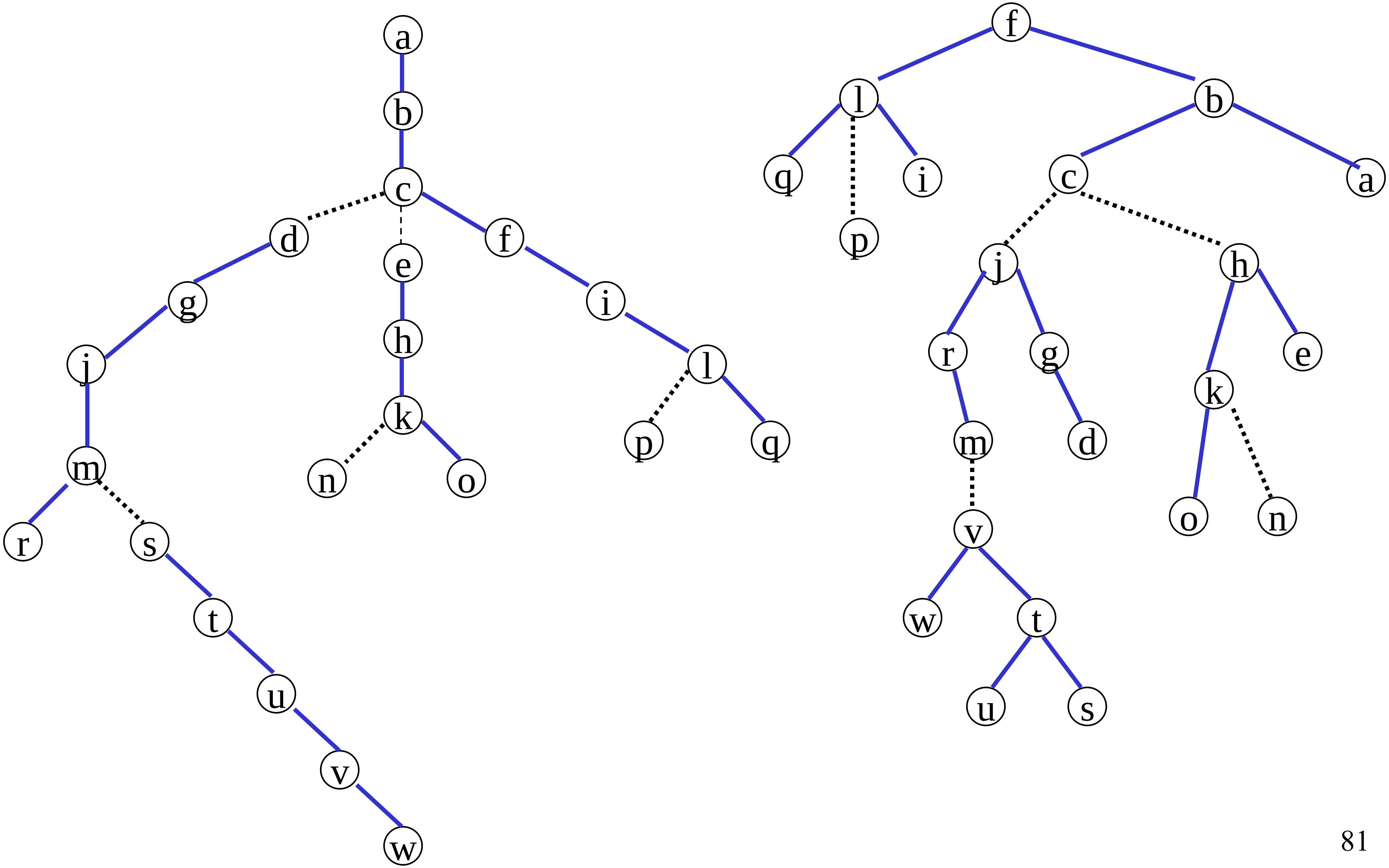
Virtual trees (cont)

Each path in T corresponds to a solid subtree in V

The parent of a vertex x in T is the successor of x (in symmetric order) in its solid subtree or the parent of the solid subtree if x is the last in symmetric order in this subtree



Virtual trees (cont)



Virtual trees (representation)

Each vertex points to p(x) to its left son l(x) and to its right son r(x).

A vertex can easily decide if it is a left child a right child or a middle child.

Each solid subtree functions like a splay tree.

The general case

Each solid subtree of a virtual tree is a splay tree.

We represent costs essentially as before.

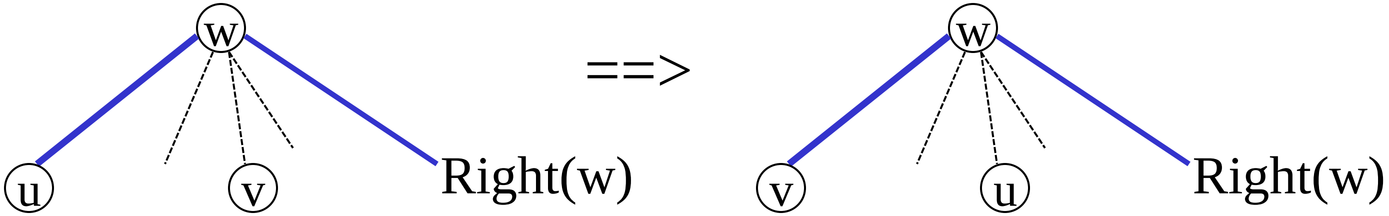
Δcost(x) = cost(x) - cost(p(x)) or cost(x) is x is a root of a solid subtree

Δmin(x) = cost(x) - mincost(x) (where mincost is the minimum cost within the subtree)

Splicing

Want to change the path decomposition such that v and the root are on the same path.

Let w be the root of a solid subtree and v a middle child of w



Want to make v the left child of w. It requires:

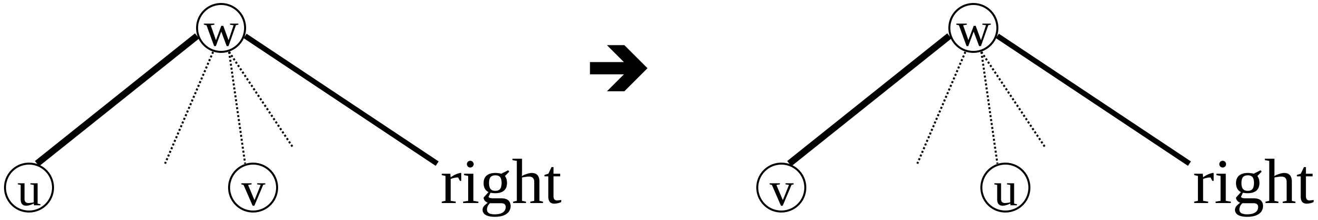
$\Delta cost'(v) = \Delta\ cost(v) - \Delta\ cost(w)$

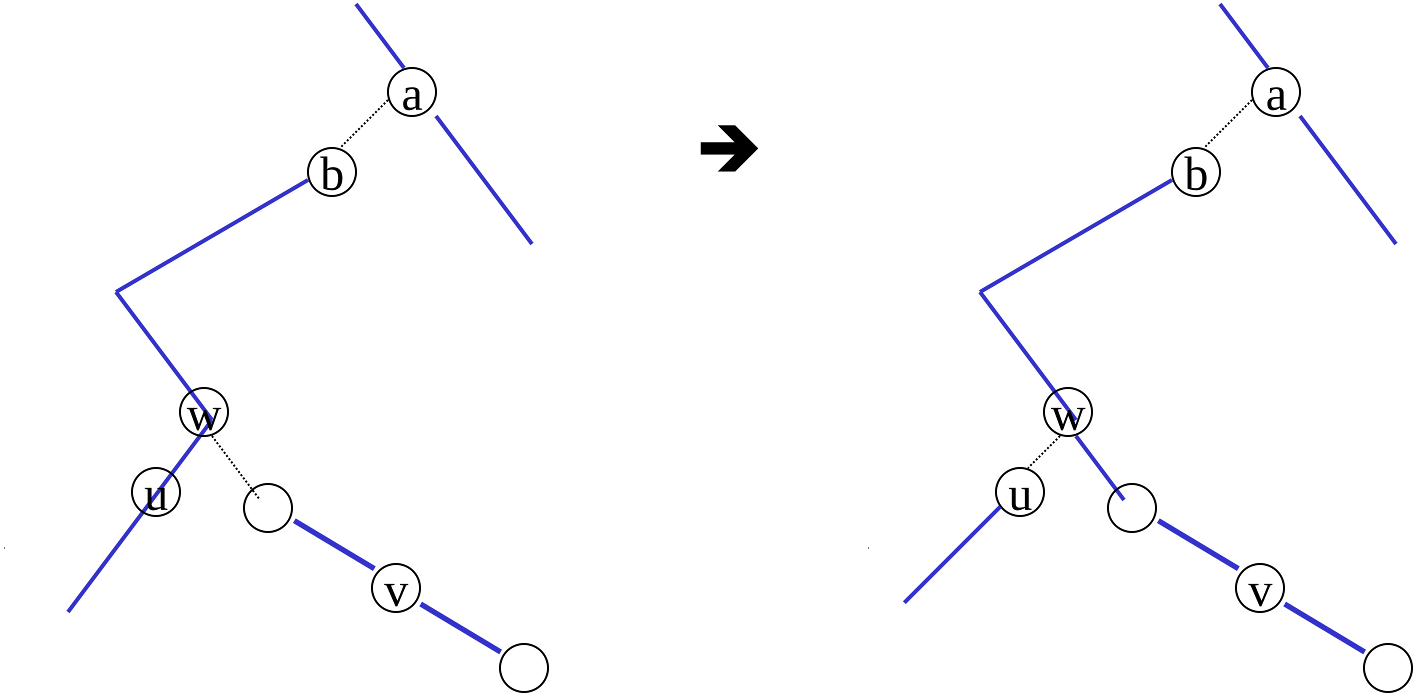
$\Delta cost'(u) = \Delta\ cost(u) + \Delta\ cost(w)$

$\Delta min'(w) = max\{0, \Delta min(v) - \Delta cost'(v), \Delta min(right(w)) - \Delta cost(right(w))\}$

Splicing (cont)

What is the effect on the path decomposition of the real tree ?





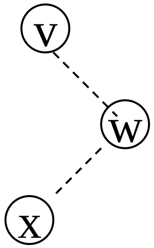
Splaying the virtual tree

Let x be the vertex in which we splay.

We do 3 passes:

1) Walk from x to the root and splay within each solid subtree

After the first pass the path from x to the root consists entirely of dashed edges



2) Walk from x to the root and splice at each proper ancestor of x.

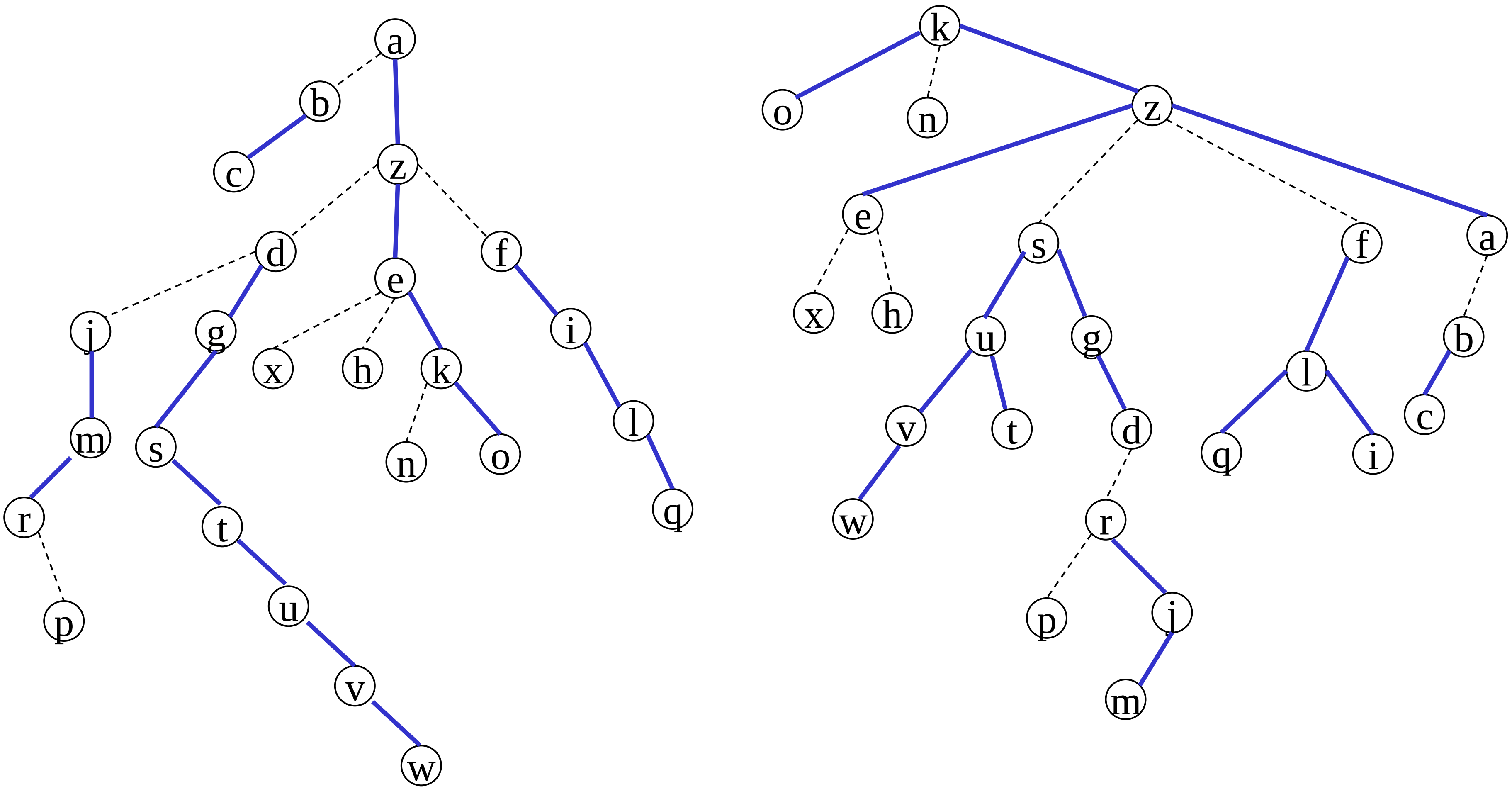
Now x and the root are in the same solid subtree

3) Splay at x

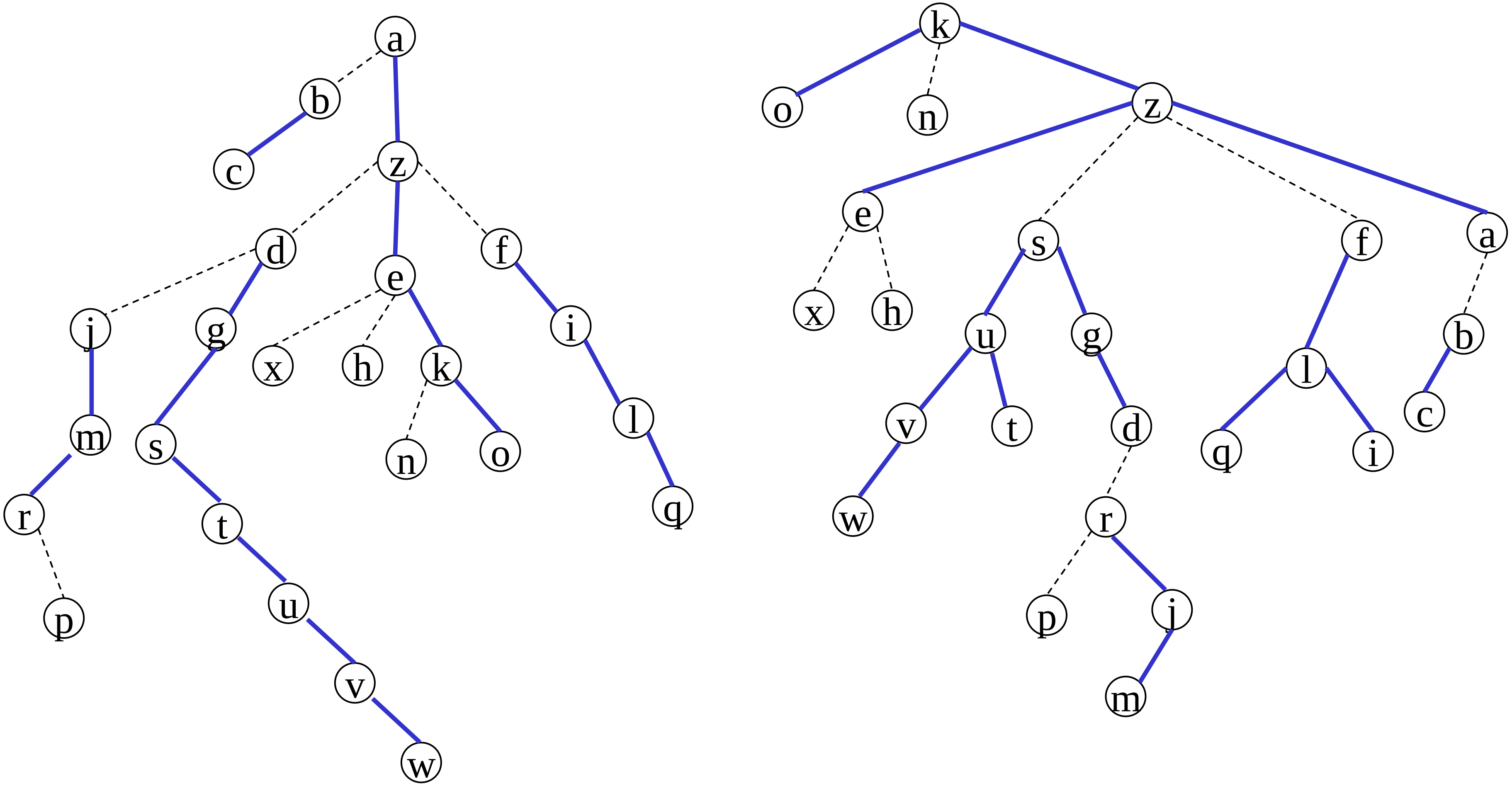
Now x is the root of the entire virtual tree.

Example

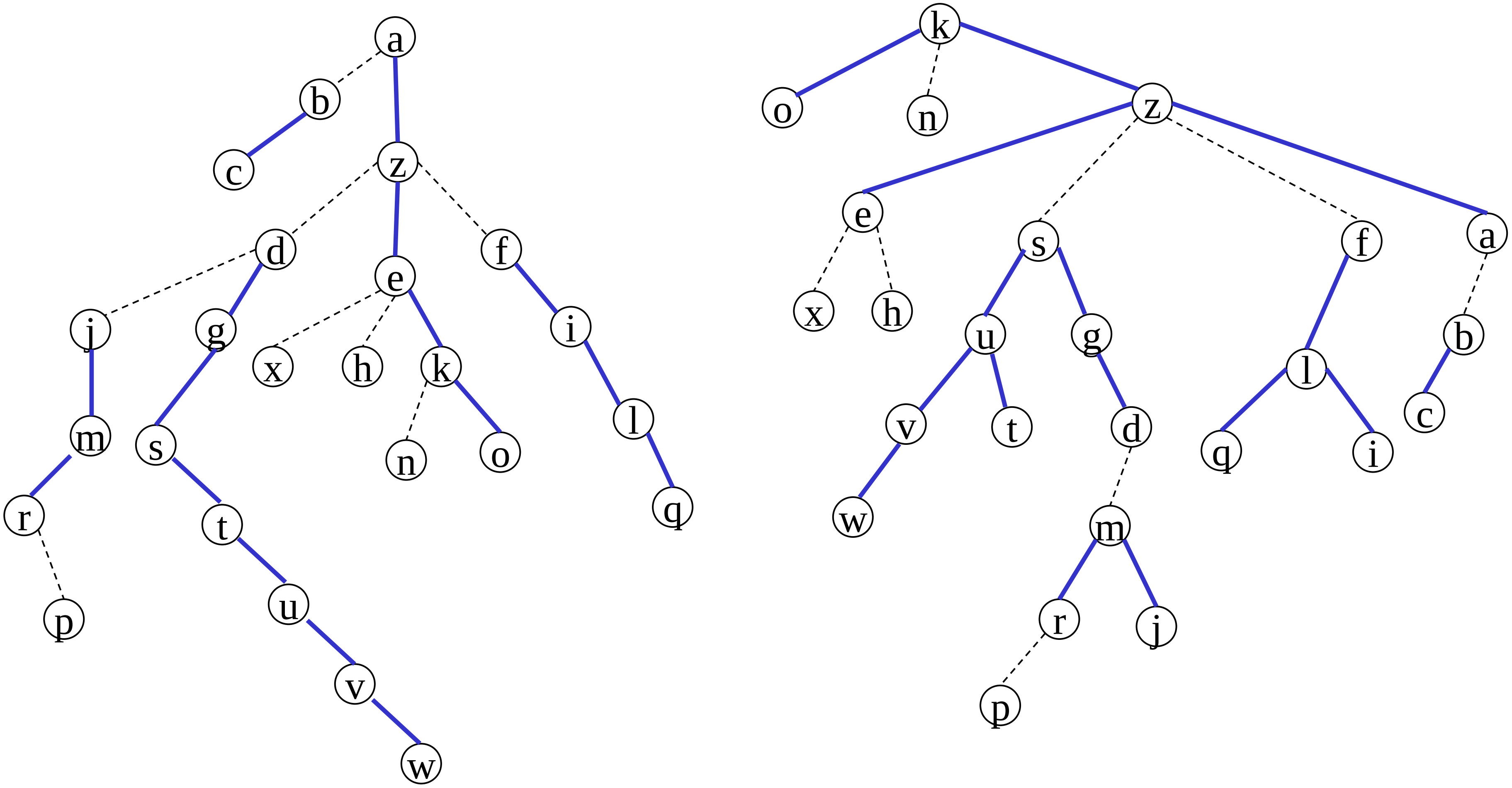
Actual and virtual trees



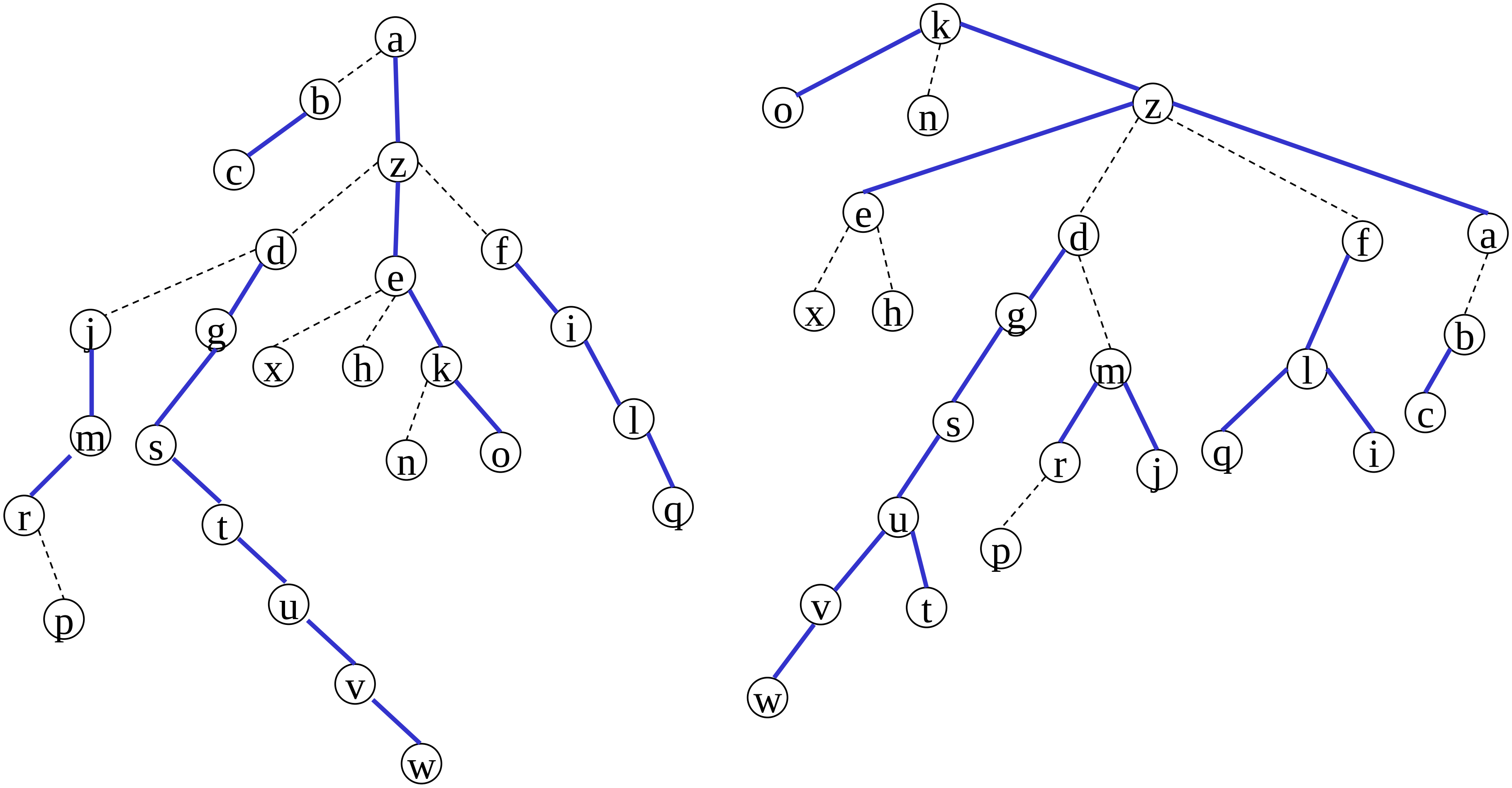
Splay at m


a
b
c
z
d
e
f
j
g
x
h
k
i
m
s
n
o
l
r
t
q
p
u
v
w
k
o
n
z
e
s
f
a
x
h
u
g
b
l
v
t
d
c
q
i
w
r
p
j
m

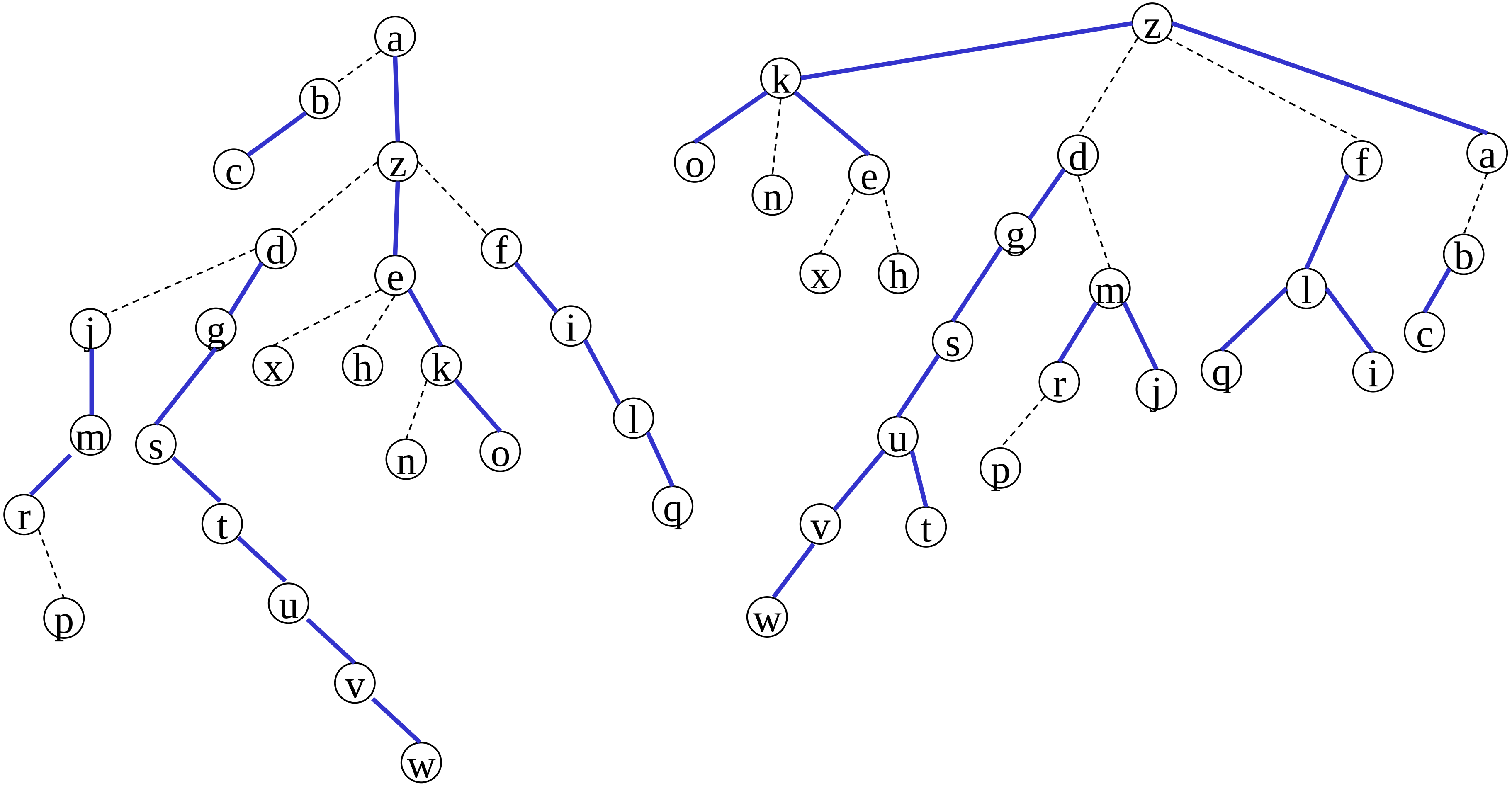
Splay at m


a
b
c
z
d
e
f
j
g
x
h
k
i
m
s
n
o
l
r
t
q
p
u
v
w
k
o
n
z
e
s
f
a
x
h
u
g
l
b
v
t
d
q
i
c
w
m
r
j
p

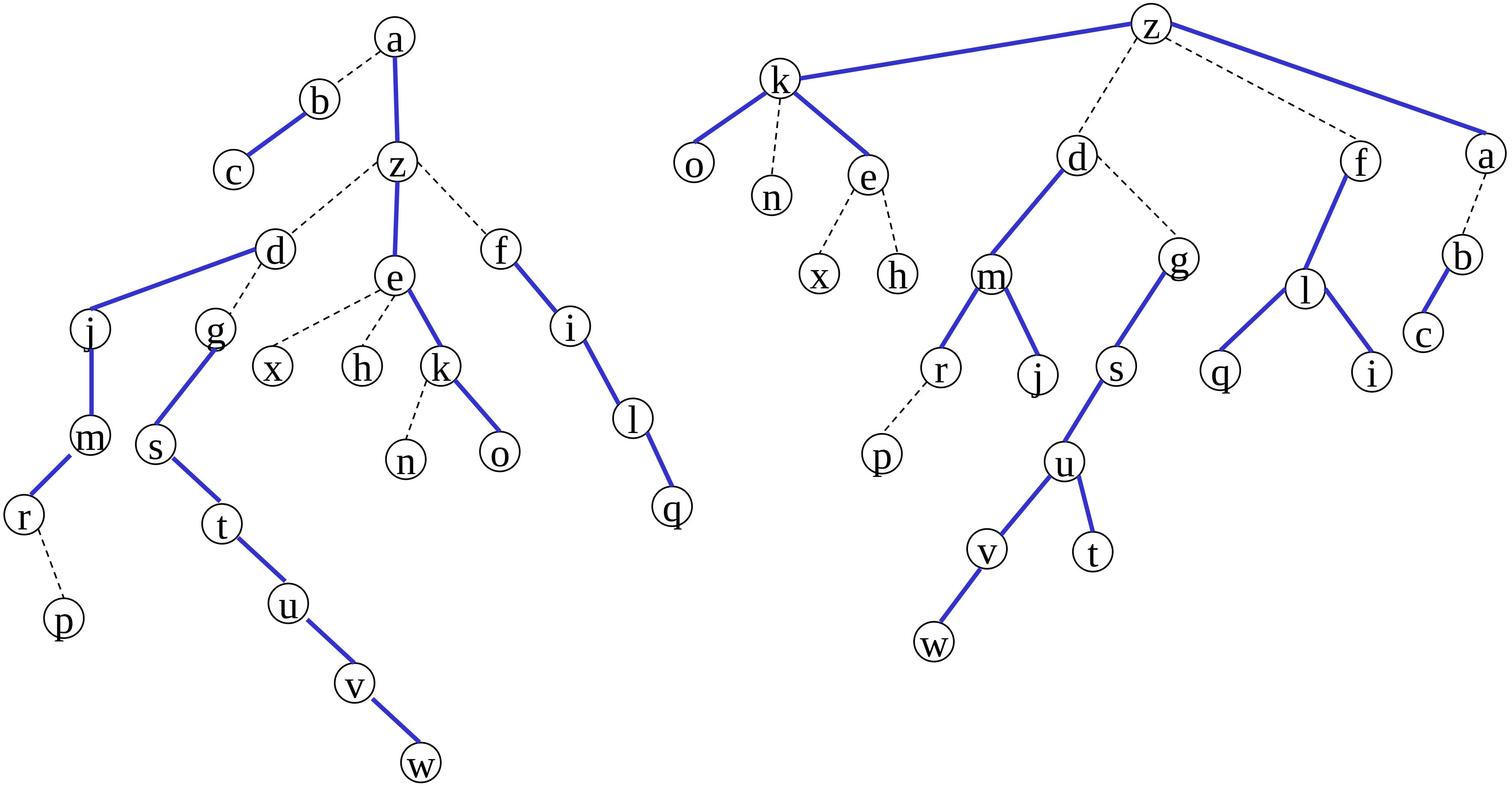
Splay at m


a
b
c
z
d
e
f
j
g
x
h
k
i
m
s
n
o
l
r
t
q
u
p
v
w
k
o
n
z
e
d
f
a
x
h
g
m
l
b
s
r
j
q
i
c
u
p
v
t
w

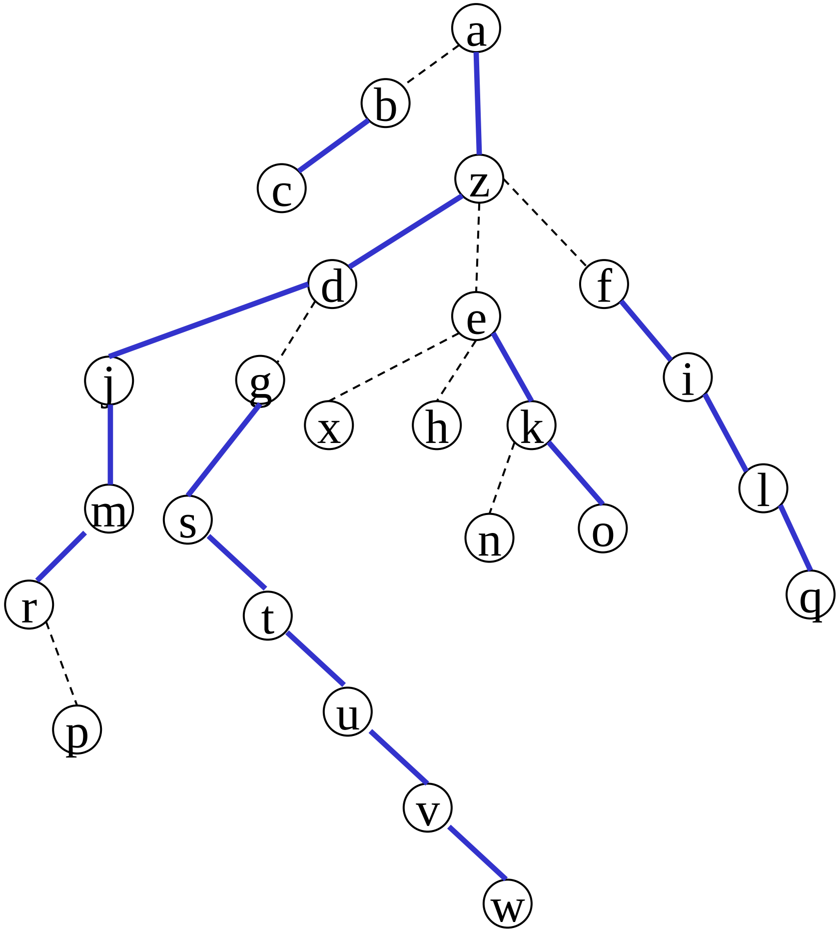
Splay at m

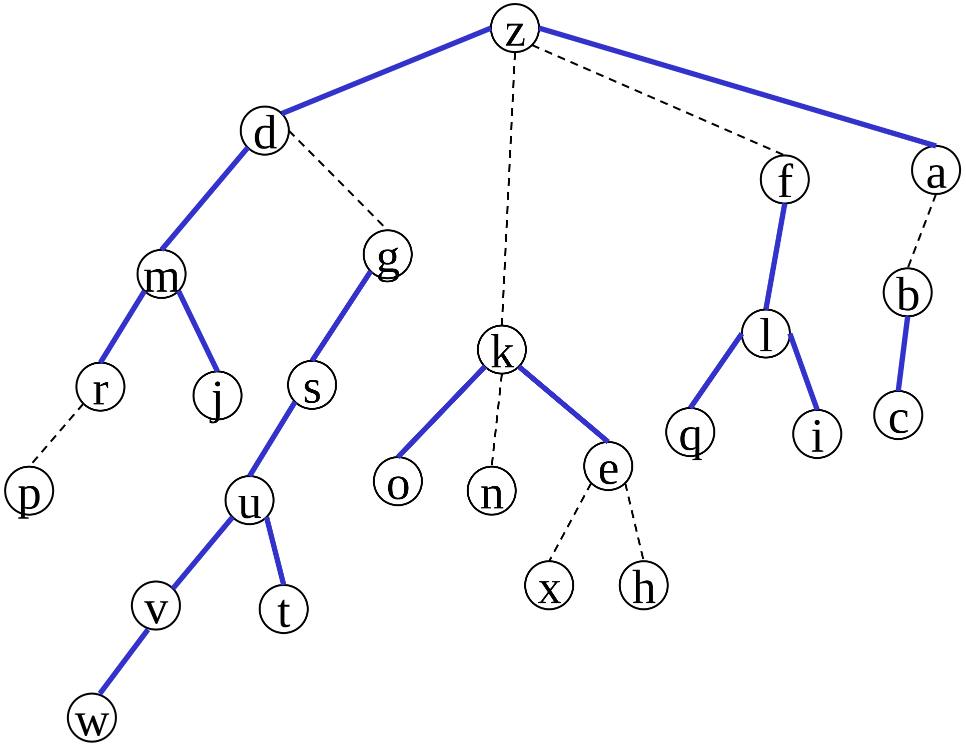

a
b
c
z
d
e
f
j
g
x
h
k
i
m
s
n
o
l
r
t
q
u
p
v
w
z
k
o
n
e
d
f
a
x
h
g
m
l
b
s
r
j
q
i
c
u
p
v
t
w

Splay at m

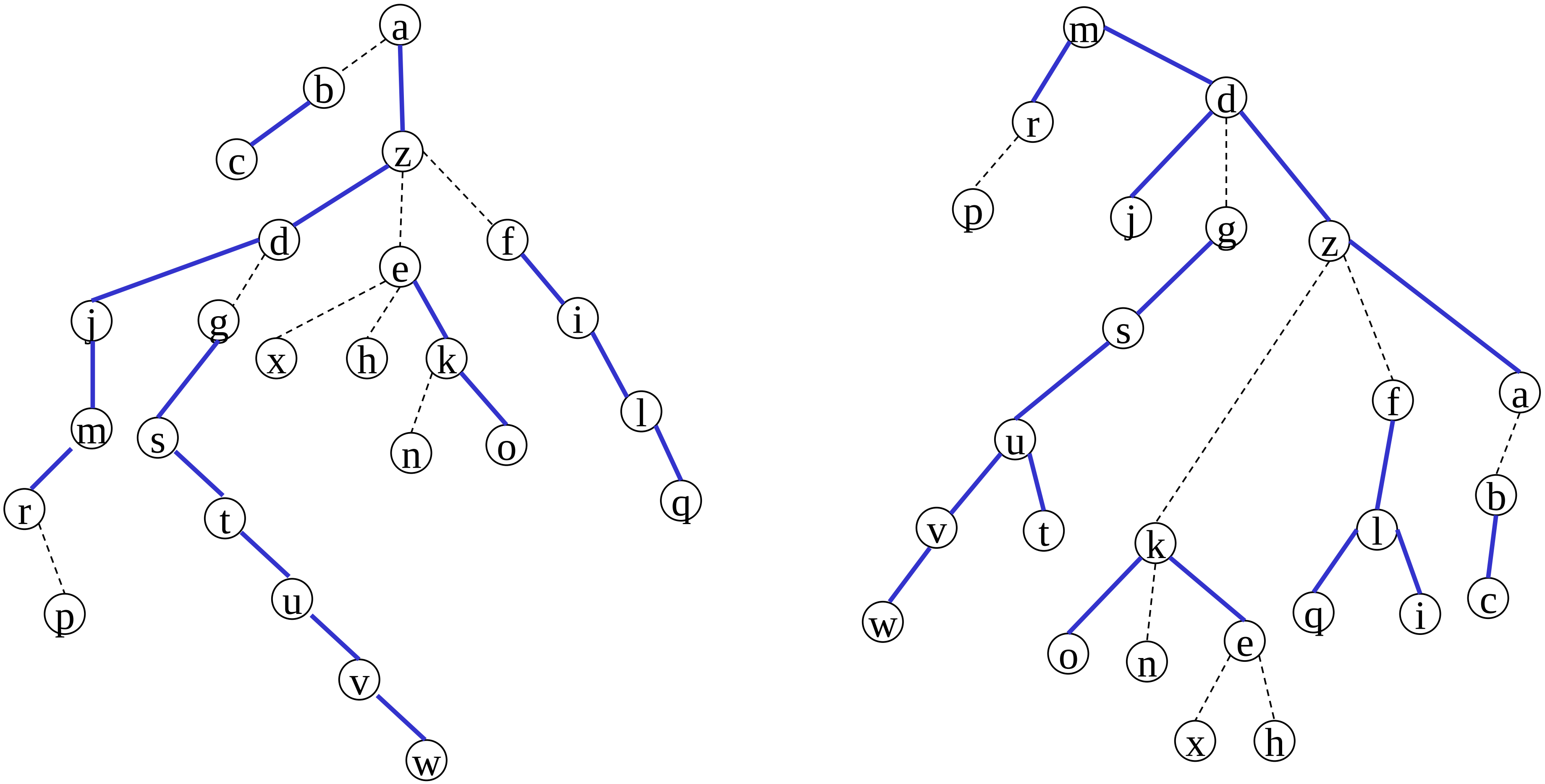

a
b
z
c
d
e
f
j
g
x
h
k
i
m
s
n
o
l
r
t
q
u
p
v
w
z
k
o
n
e
d
f
a
x
h
m
g
l
b
r
j
s
q
i
c
p
u
v
t
w

Splay at m


a
b
c
z
d
e
f
j
g
x
h
k
i
m
s
n
o
l
r
t
q
u
p
v
w


z
d
f
a
m
g
b
l
k
r
j
s
q
i
c
o
n
e
p
u
x
h
v
t
w

Splay at m


a
b
c
z
d
e
f
j
g
x
h
k
i
m
s
n
o
l
r
t
q
u
p
v
w
m
r
d
p
j
g
z
s
f
a
u
v
t
k
l
b
w
o
n
e
q
i
c
x
h

Dynamic tree operations

w = findroot(v) : Splay at v, follow right pointers until reaching the last node w, splay at w, and return w.

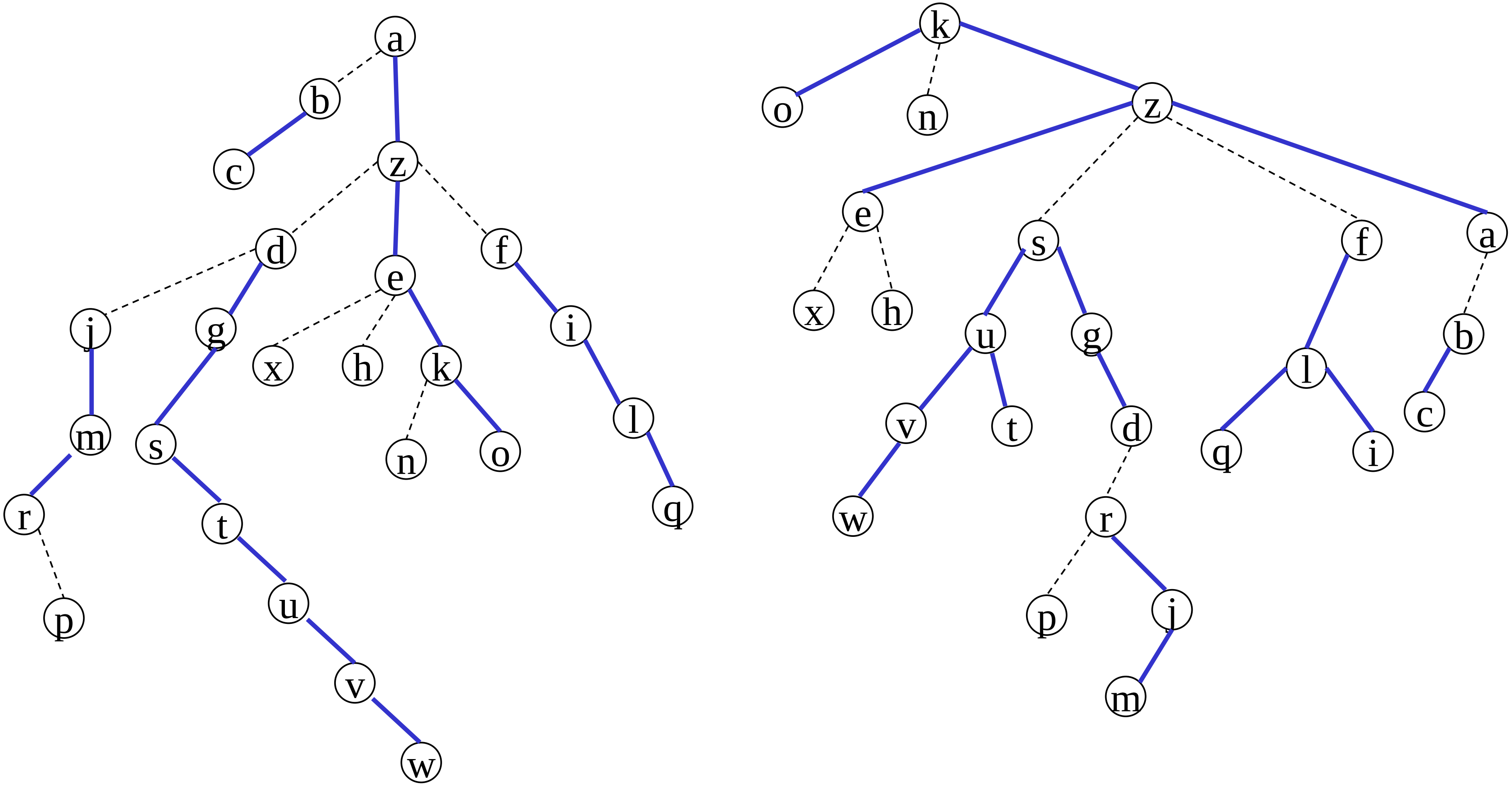
(v,c) = mincost(v) : Splay at v and use Δcost and Δmin to follow pointers to the smallest node after v on its path (its in the right subtree of v). Let w be this node, splay at w.

addcost(v,c) : Splay at v, increase Δcost(v) by c and decrease Δcost(left(v)) by c, update Δmin(v)

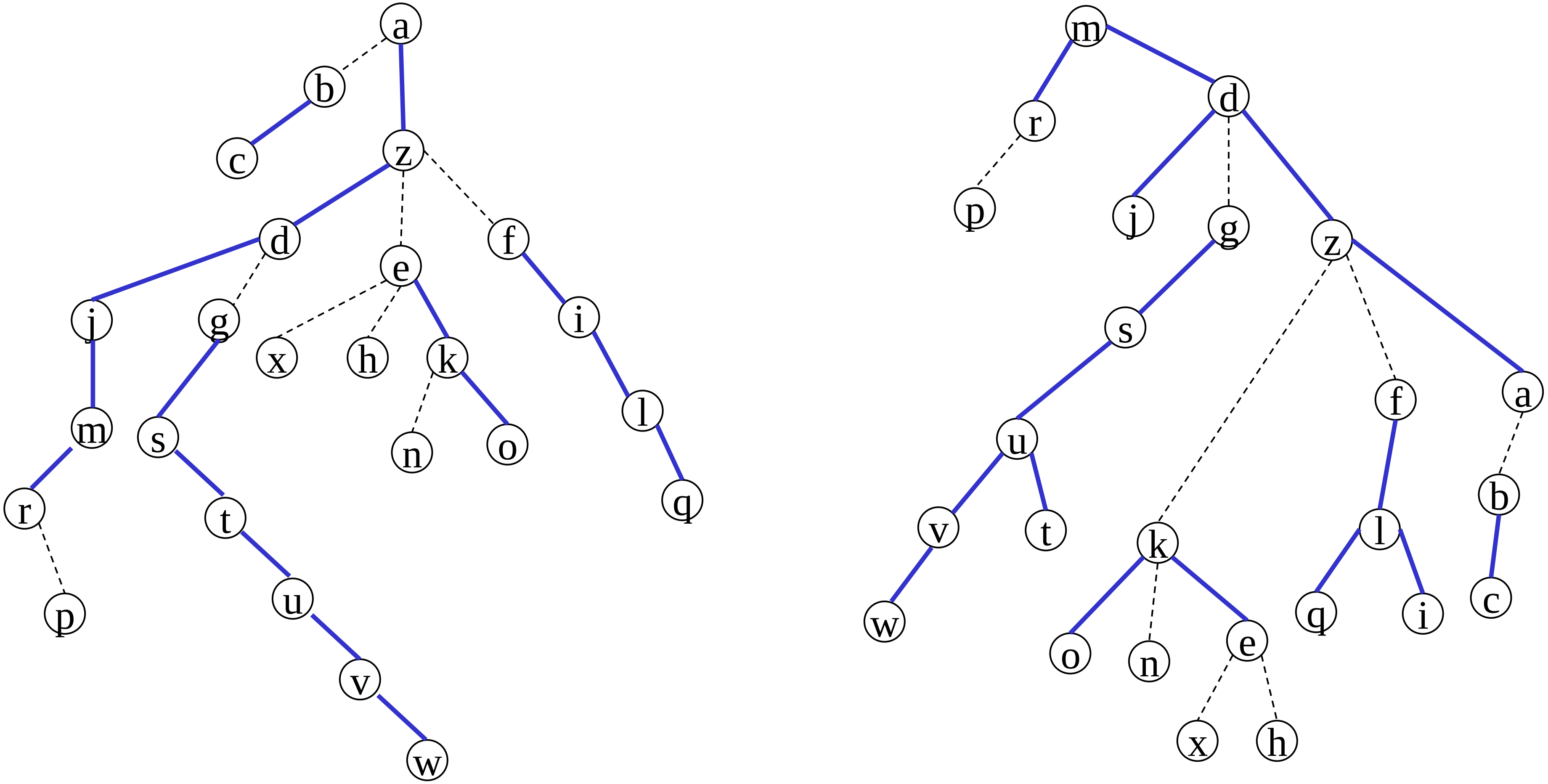
link(v,w,c(v,w)) : Splay at v, update the cost of v to be c(v,w) (requires updates to Δcost(v), Δmin(v), Δcost(left(v)), and Δcost(right(v)), splay at w (so potential does not increase too much when we add v as a child) and make v a middle child of w

cut(v) : Splay at v, break the link between v and right(v), set Δcost(right(v)) += Δcost(v)

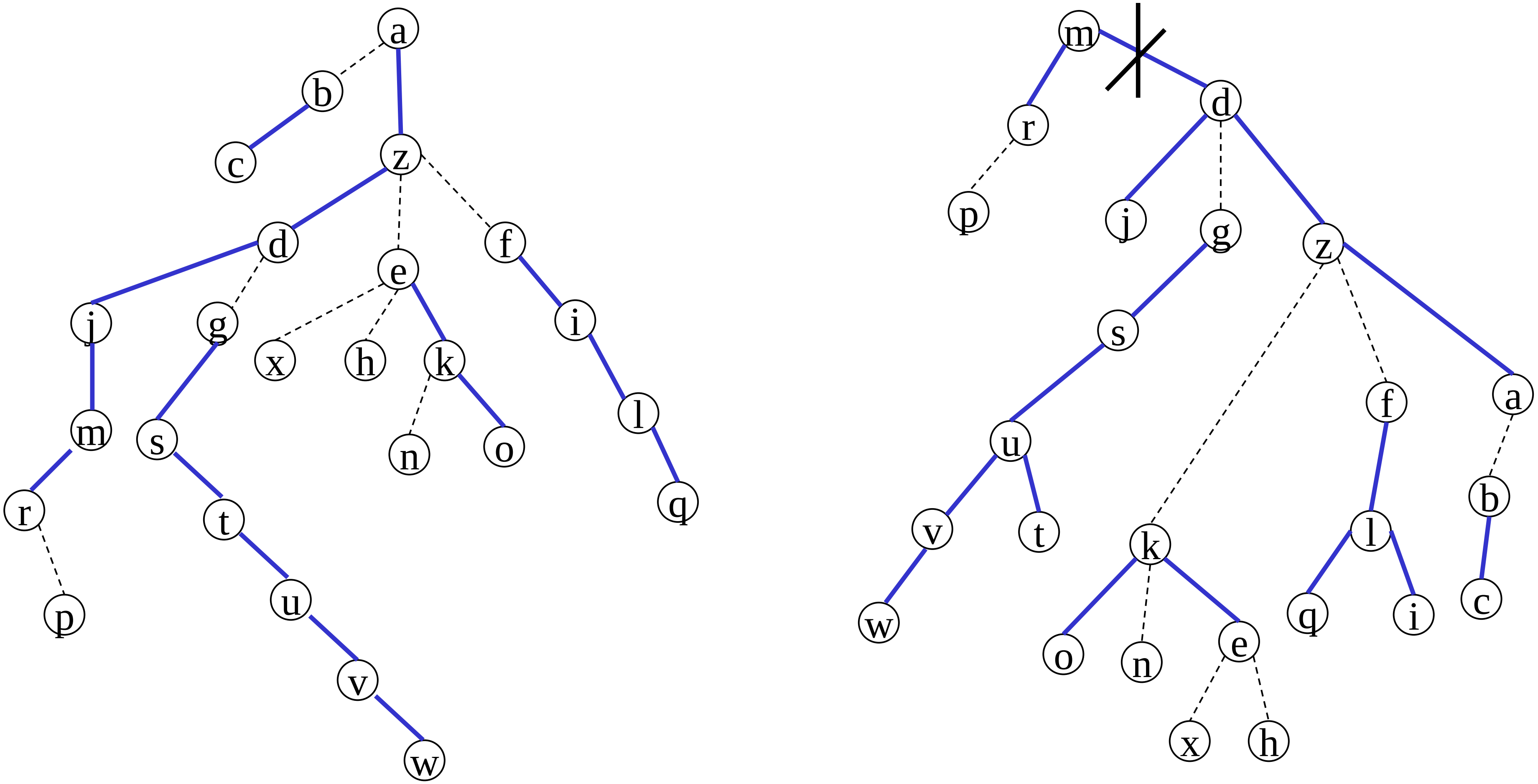
Cut(m)


a
b
c
z
d
e
f
j
g
x
h
k
i
m
s
n
o
l
r
t
q
p
u
v
w
k
o
n
z
e
s
f
a
x
h
u
g
l
b
v
t
d
q
i
c
w
r
p
j
m

Splay at m


a
b
c
z
d
e
f
j
g
x
h
k
i
m
s
n
o
l
r
t
q
u
p
v
w
m
r
d
p
j
g
z
s
f
a
u
b
v
t
k
l
w
q
i
c
o
n
e
x
h

Cut at m


a
b
c
z
d
e
f
j
g
x
h
k
i
m
s
n
o
l
r
t
q
p
u
v
w
m
r
d
p
j
g
z
s
f
a
u
b
v
t
k
l
c
w
q
i
o
n
e
x
h

Dynamic tree (analysis)

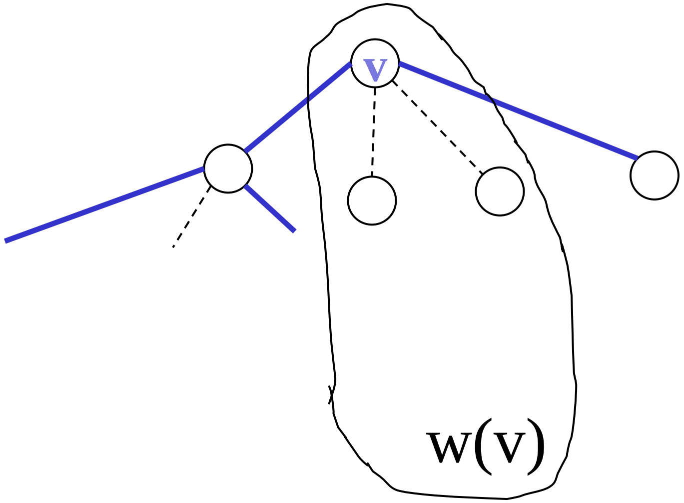
It suffices to analyze the amortized time of splay in the virtual tree

Use the access lemma as follows:

The weight assigned to each node/item v is

1 + ∑sizes of subtrees (in the virtual tree) rooted at middle children of v

➔The size of v is the #elements in v’s subtree in the virtual tree

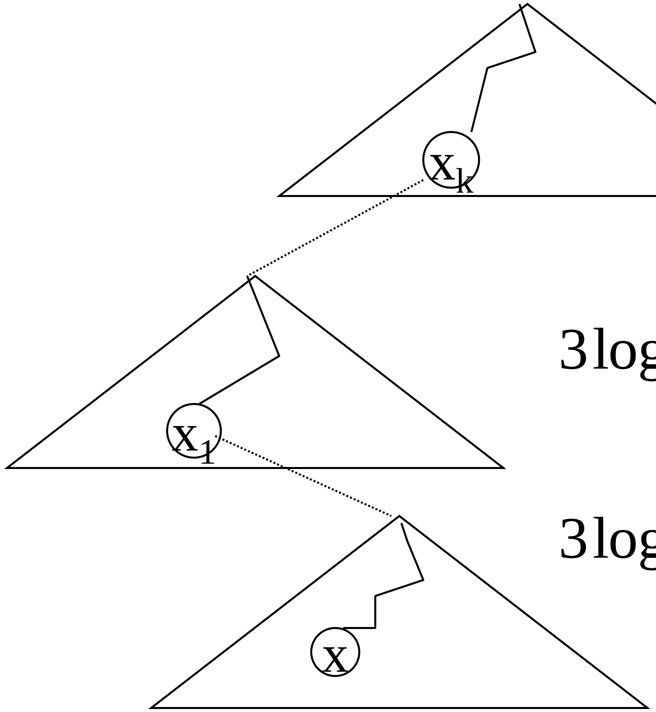


Note: Splices do not affect the size of v

Dynamic tree (analysis)

Analysis of the step (1) of a splay of a node in the virtual tree:

Apply the access lemma to each splay and sum up



k=#solid subtree along the path

$$3\log\left(\frac{s(T_k)}{s(x_k)}\right)+.....+3\log\left(\frac{s(T_1)}{s(x_1)}\right)+3\log\left(\frac{s(T_x)}{s(x)}\right)+k\le$$

$$3\log\left(\frac{s(T_k)}{s(x)}\right)+k$$

Dynamic tree (analysis)

pass 1 takes 3logn + k

pass 2 takes k

pass 3 takes 3logn + 1

How do we get rid of this k ?

Refining the access lemma

Original version: **The amortized time to splay a node x in a tree with root t is at most $3(r(t) - r(x)) + 1 = 3\log(s(t)/s(x)) + 1$**

Modified version: **For any constant $c \geq 1$, the amortized time to splay a node x in a tree with root t is at most $3c(r(t) - r(x)) + 1 = 3c\log(s(t)/s(x)) + 1-(\ell-1)(c-1)$, where ℓ is the length of the splay path**

Dynamic tree (analysis)

pass 1 takes 3clogn + k

pass 2 takes k-1

pass 3 takes 3clogn + 1 – (k-2)(c-1)

➔ O(log n)

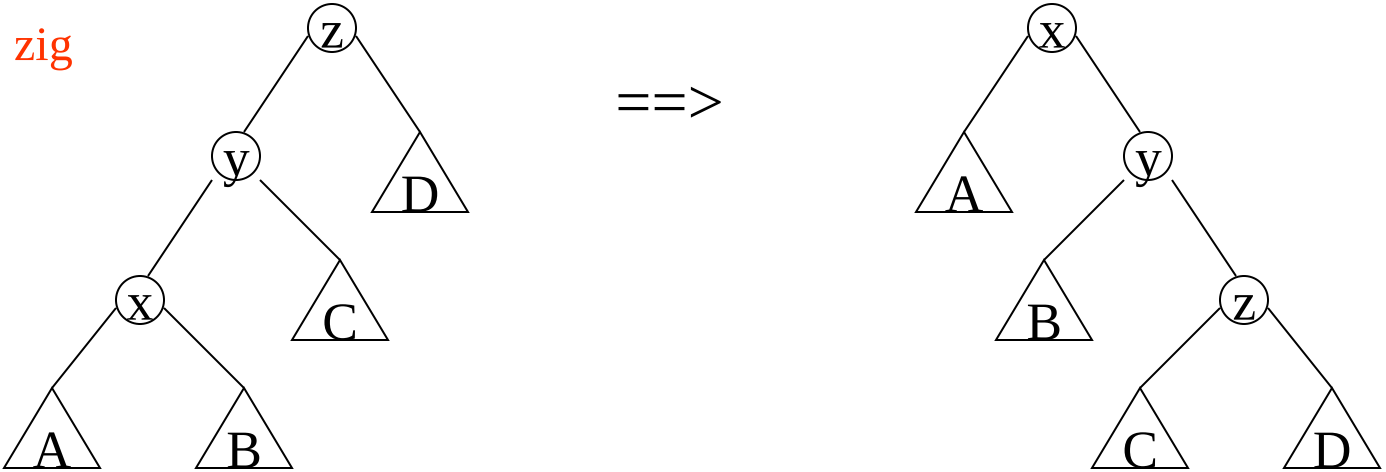
Proving the modified access lemma

- Same proof, multiply the potential by c:

Potential is: $\mathbf{c \cdot \sum r(x) = c \cdot \sum \log_2(s(x))}$

Proof of the access lemma (cont)

(1) zig - zig



amortized time(zig-zig) = $2 + \Delta\Phi =$

$2 + c(r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)) \leq$

$2 + c(r'(x) + r'(z) - r(x) - r(y)) \quad \leq \quad 2 + c(r'(x) + r'(z) - r(x) - r(x)) =$

$2 + c(r(x) - r'(x) + r'(z) - r'(x) + 3(r'(x) - r(x))) \leq$

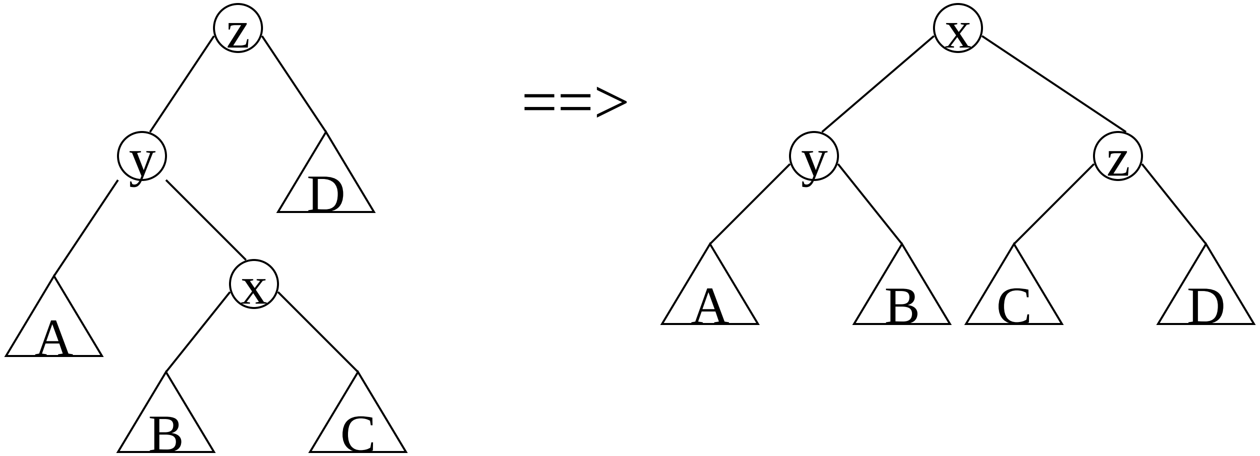
$2 + c(\log(s(x)/s'(x)) + \log(s'(z)/s'(x))) + 3c(r'(x) - r(x)) \leq$

$2 + c(\log([s'(x)/2]/s'(x)) + \log([s'(x)/2]/s'(x))) + 3c(r'(x) - r(x)) =$

$3c(r'(x) - r(x)) - 2(c-1)$

Proof of the access lemma (cont)

(2) zig - zag



Same modification