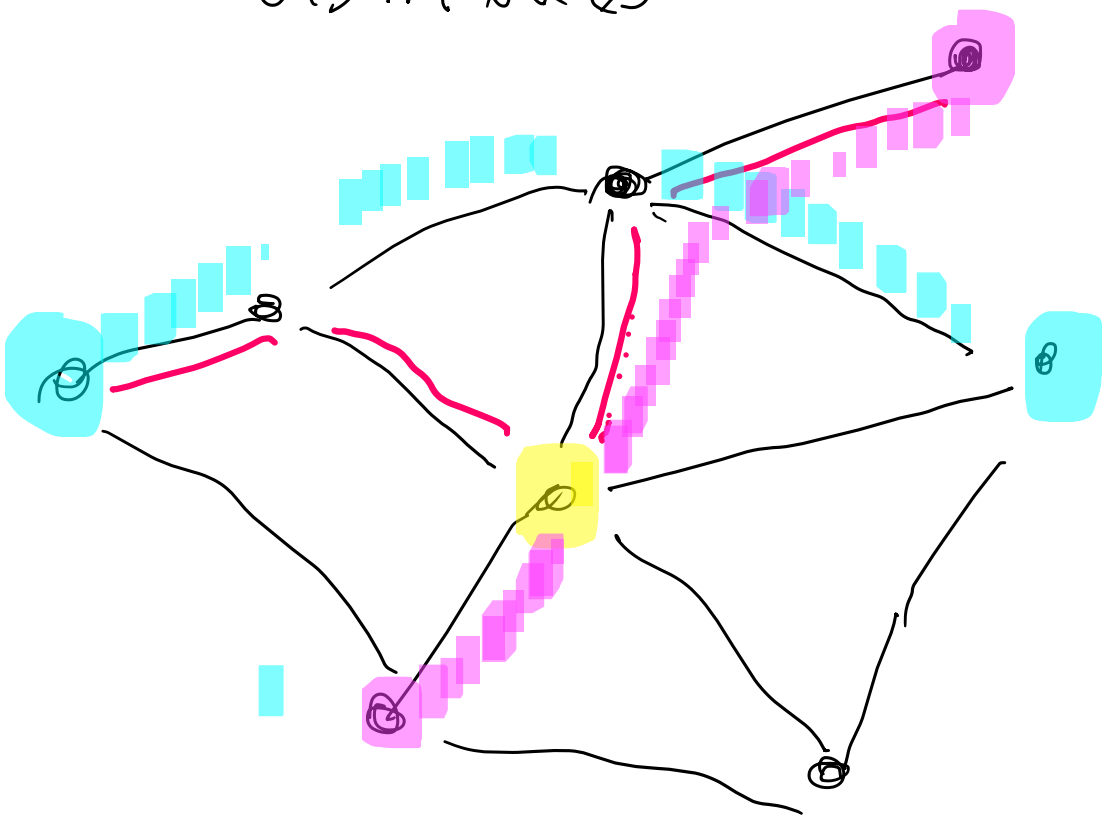
Simple tree algorithms

Mku, PWr 2021

distributed



radius

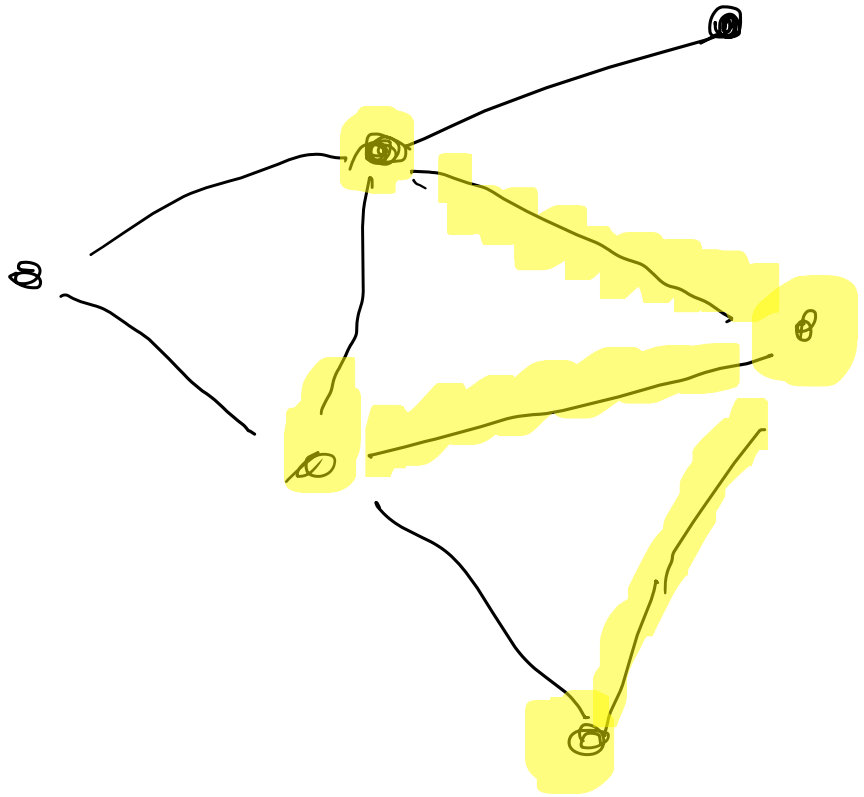
minimal distance from „center"

diameter

largest distance

$$R \leq D \leq 2R$$

distributed



Node

P

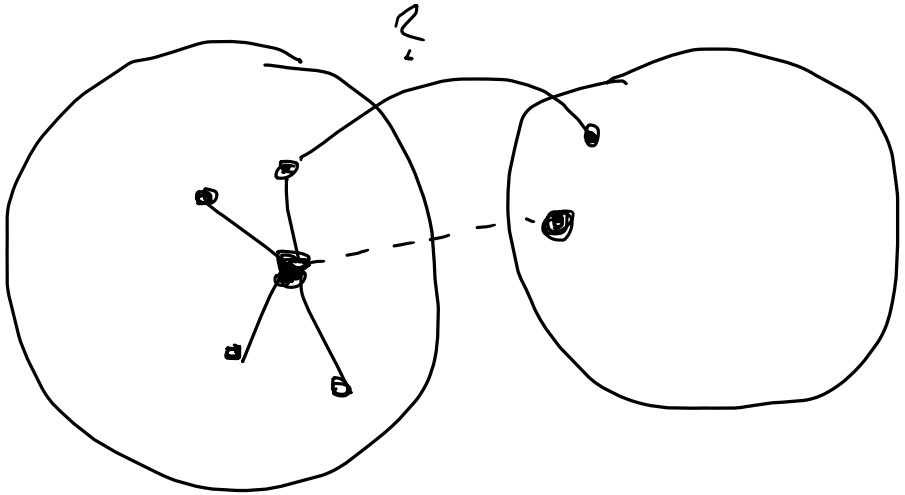
```
While
 receive
  send
End
```

Complexity: time
number of messages

Broadcast: graph with m edges
local knowledge

$\Rightarrow$ message complexity of broadcast

$\geq m$



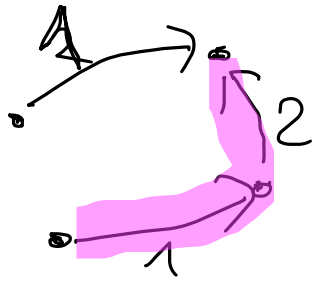
Broadcast

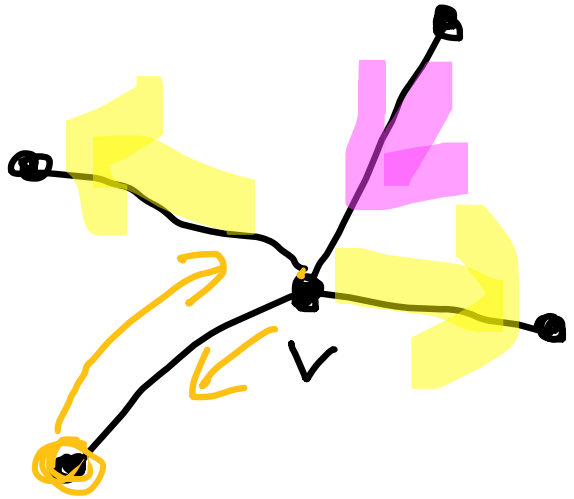
unknown topology

Algorithm 2.9 Flooding

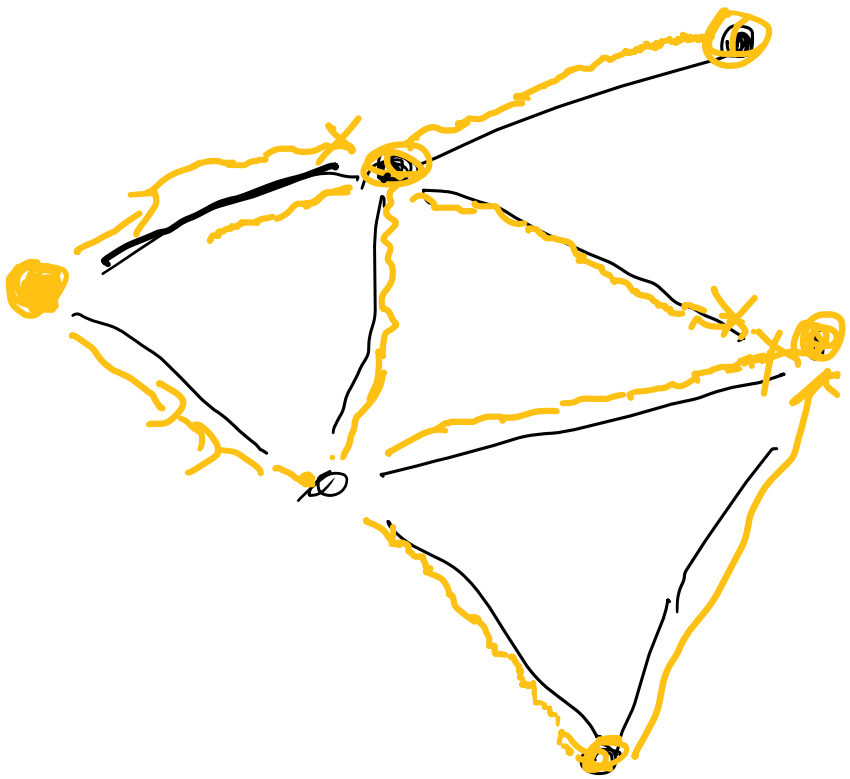
1: The source (root) sends the message to all neighbors.
2: **Each other node** v upon receiving the message the first time forwards the message to all (other) neighbors.
3: Upon later receiving the message again (over other edges), a node can discard the message.

asynchrony: order of messages is arbitrary





distributed



- time: diameter
- message: each edge ≤ 2 messages

$$\#\text{edges} \cdot 2$$

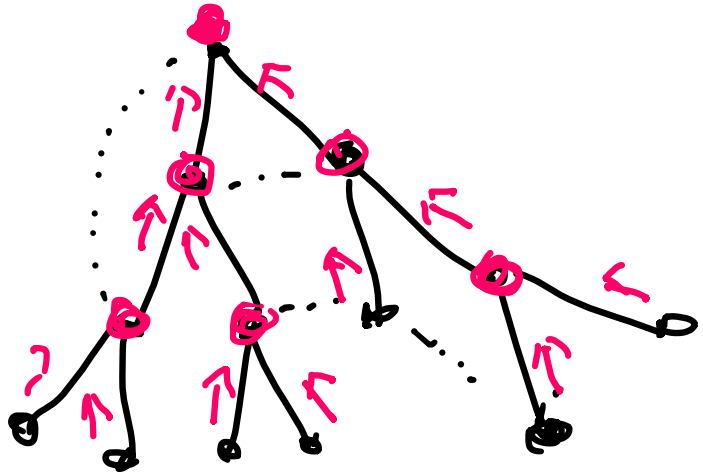
- <u>added value:</u>

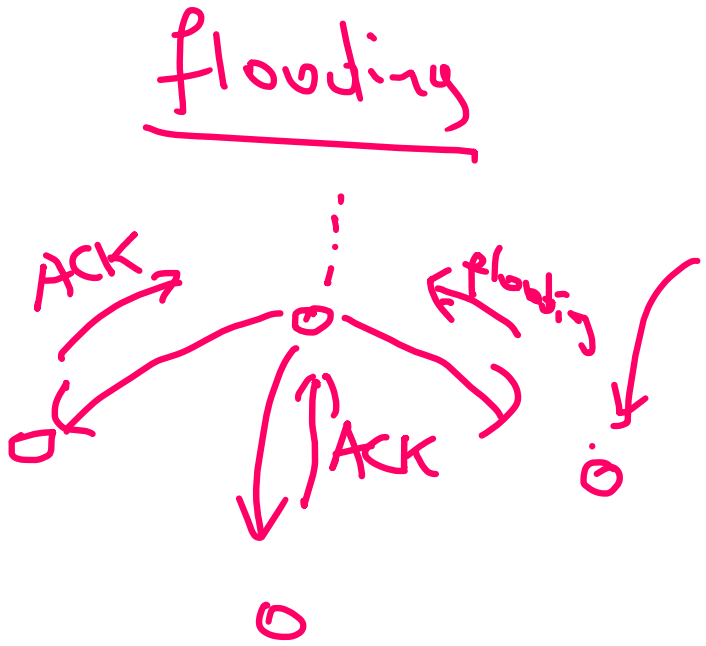
  if I got the message from node A then A is my parent in spanning tree

Convergecast

Algorithm 2.10 Echo

1: A leave sends a message to its parent.
2: If an inner node has received a message from each child, it sends a message to the parent.

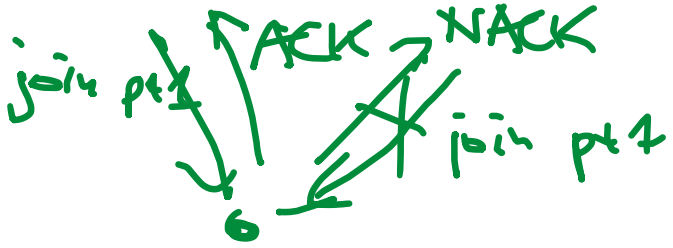


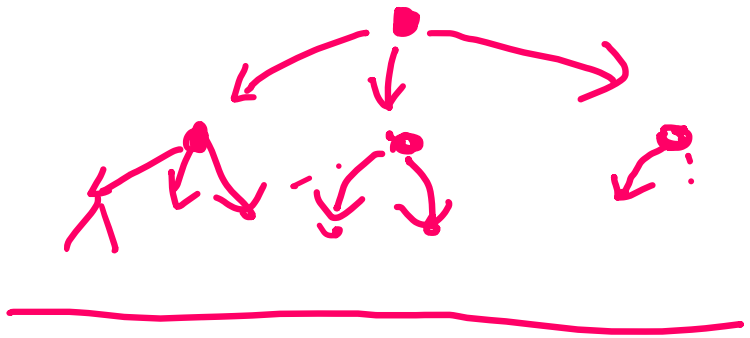


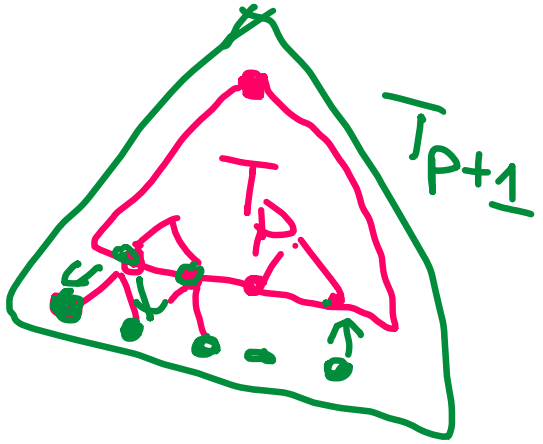
Broad-first-search BFS

Algorithm 2.11 Dijkstra BFS

1: We start with T_1 which is the root plus all direct neighbors of the root. We start with phase $p = 1$:
2: **repeat**
3: The root starts phase p by broadcasting "start p" within T_p.
4: When receiving "start p" a leaf node u of T_p (that is, a node that was newly discovered in the last phase) sends a "join $p + 1$" message to all quiet neighbors. (A neighbor v is quiet if u has not yet "talked" to v.)
5: A node v receiving the first "join p+1" message replies with "ACK" and becomes a leaf of the tree T_{p+1}.
6: A node v receiving any further "join" message replies with "NACK".
7: The leaves of T_p collect all the answers of their neighbors; then the leaves start an echo algorithm back to the root.
8: When the echo process terminates at the root, the root increments the phase
9: **until** there was no new node detected



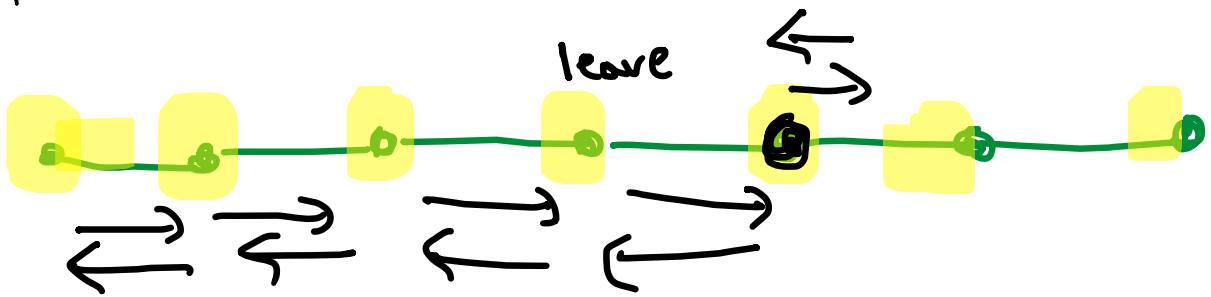




Time: $O(D^2)$ D-diameter

Message: $O(m + n \cdot D)$

- D phases



$\leq \frac{D \cdot (D-1)}{2} \approx D^2$

messages 1. each edge join
ACK or NACK

2. broadcast + echo

$2n \cdot D$

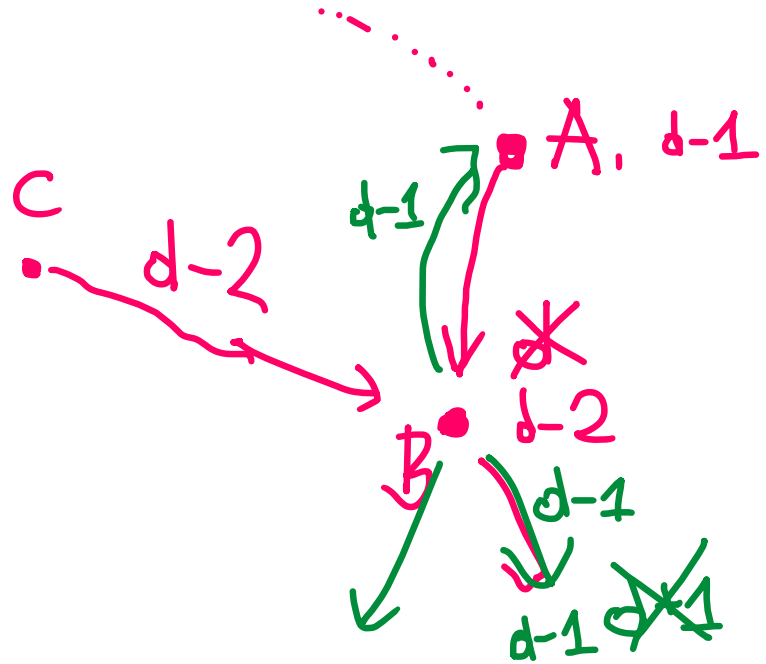
Bellman – Ford

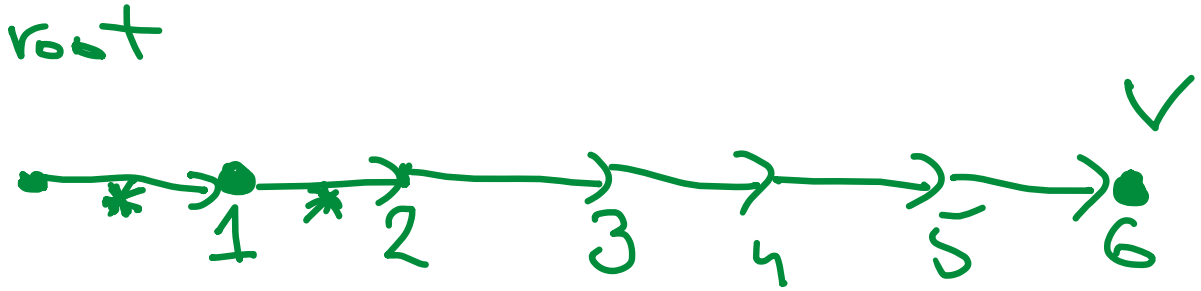
Algorithm 2.13 Bellman-Ford BFS

1: Each node u stores an integer d_u which corresponds to the distance from u to the root. Initially $d_{\text{root}} = 0$, and $d_u = \infty$ for every other node u.
2: The root starts the algorithm by sending "1" to all neighbors.
3: **if** a node u receives a message "y" with $y < d_u$ from a neighbor v **then**
4: node u sets $d_u := y$
5: node u sends "$y + 1$" to all neighbors (except v)
6: **end if**

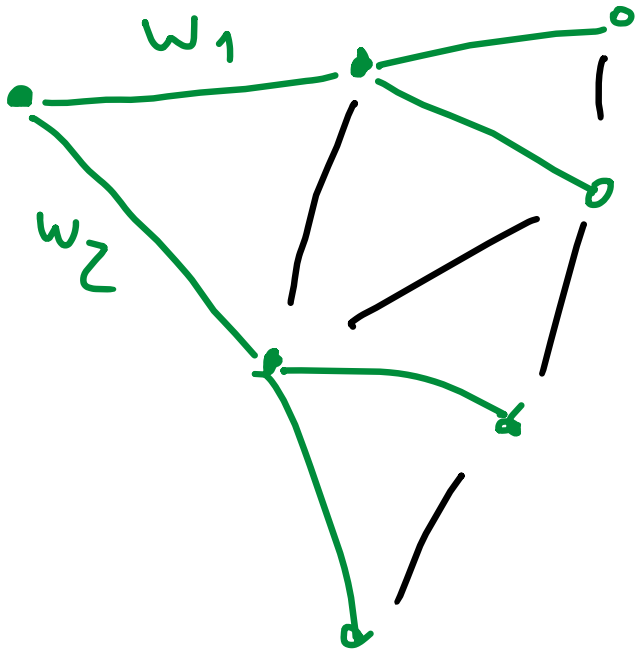
+∞

- any delivery time for messages ⇒ still correct
- eventually you terminate


A, d-1
C
d-2
d-1
B
d-2
d-1
d-1
d-1

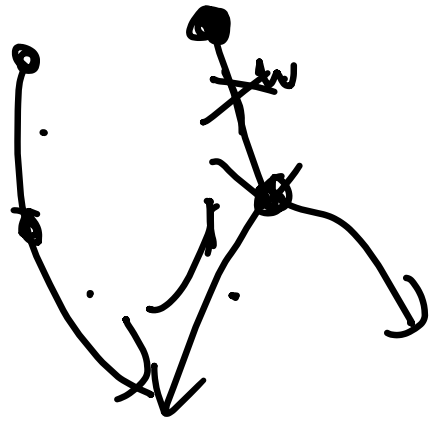

root
1
2
3
4
5
6
V

Minimum Spanning Tree

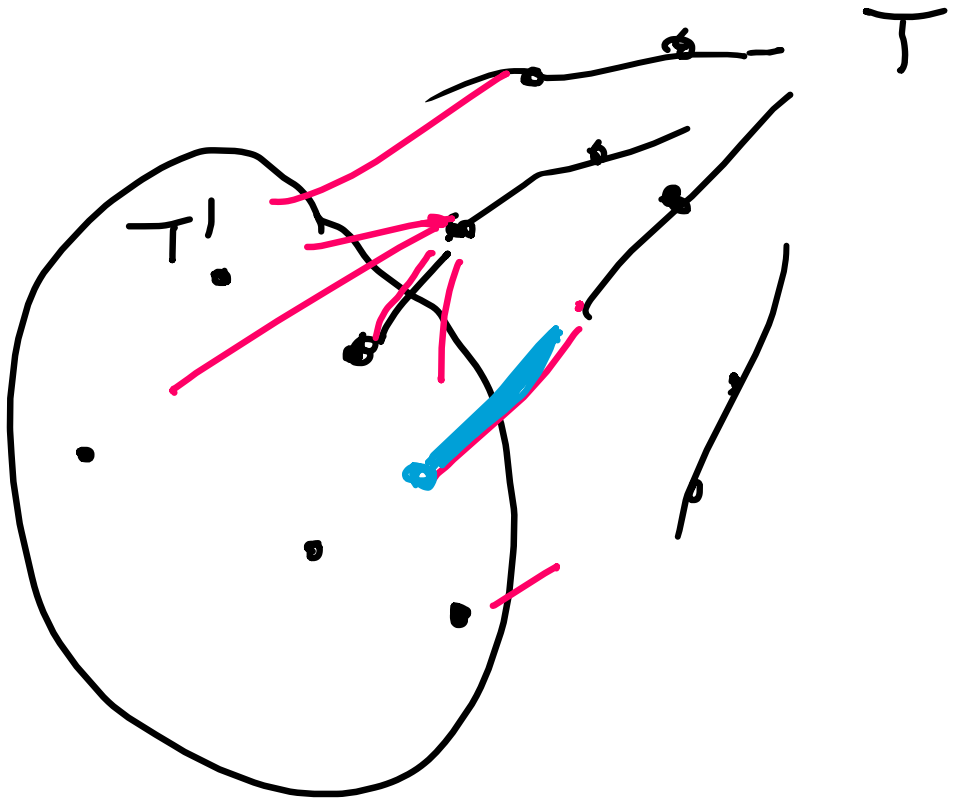


$$\sum w_i = \text{minimal}$$

w_i from ST



Blue edges

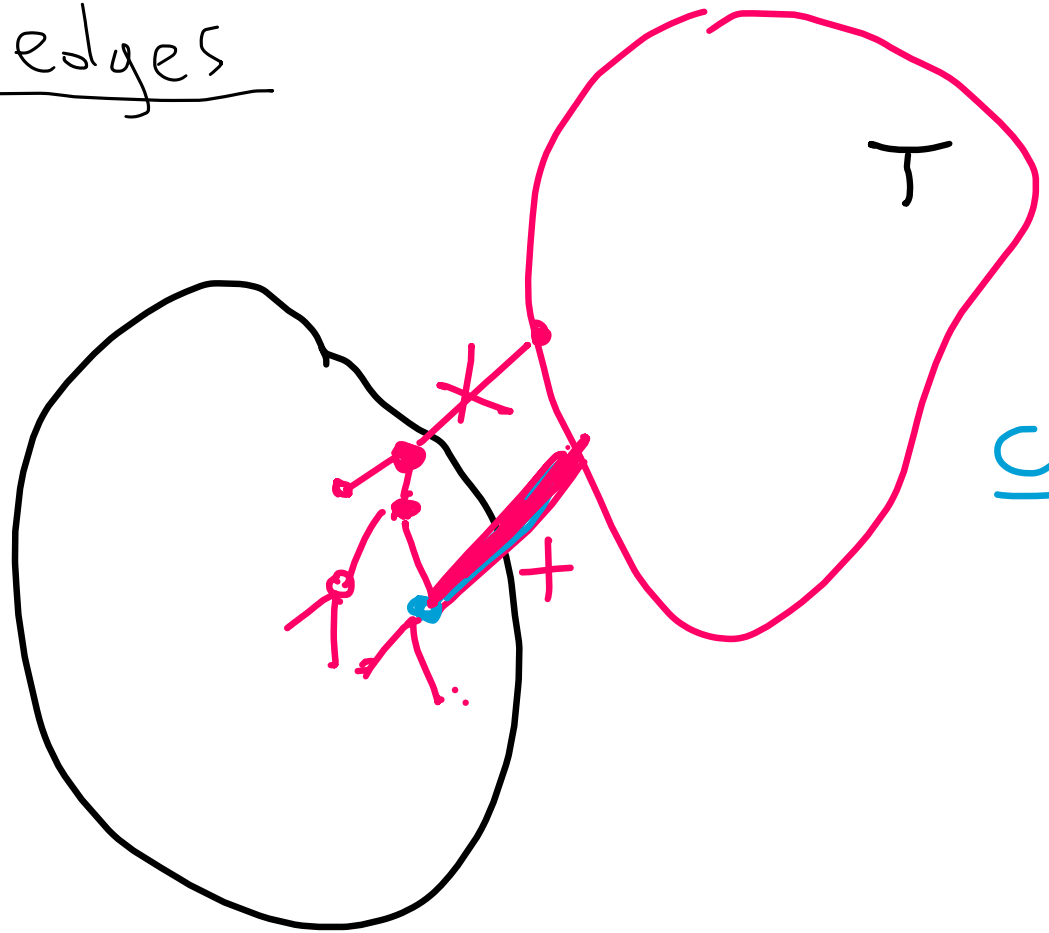


red edge with minimal weight

Claim

blue edge must belong to the min. ST

Blue edges



red edge with minimal weight

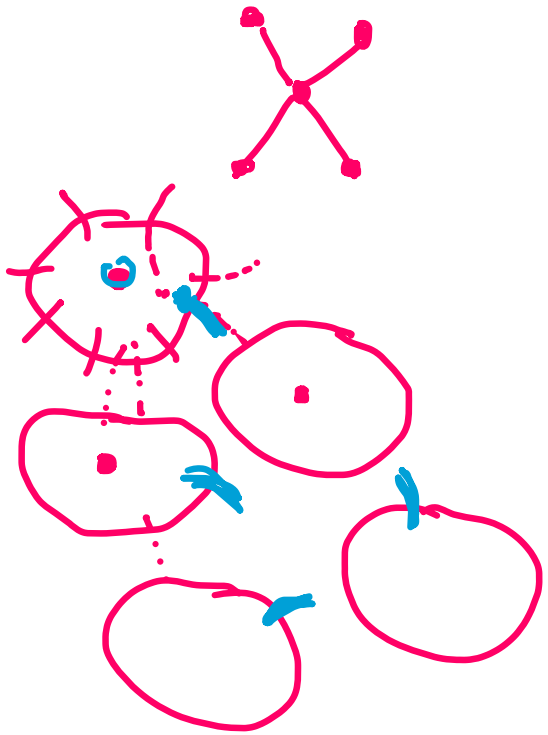
Claim

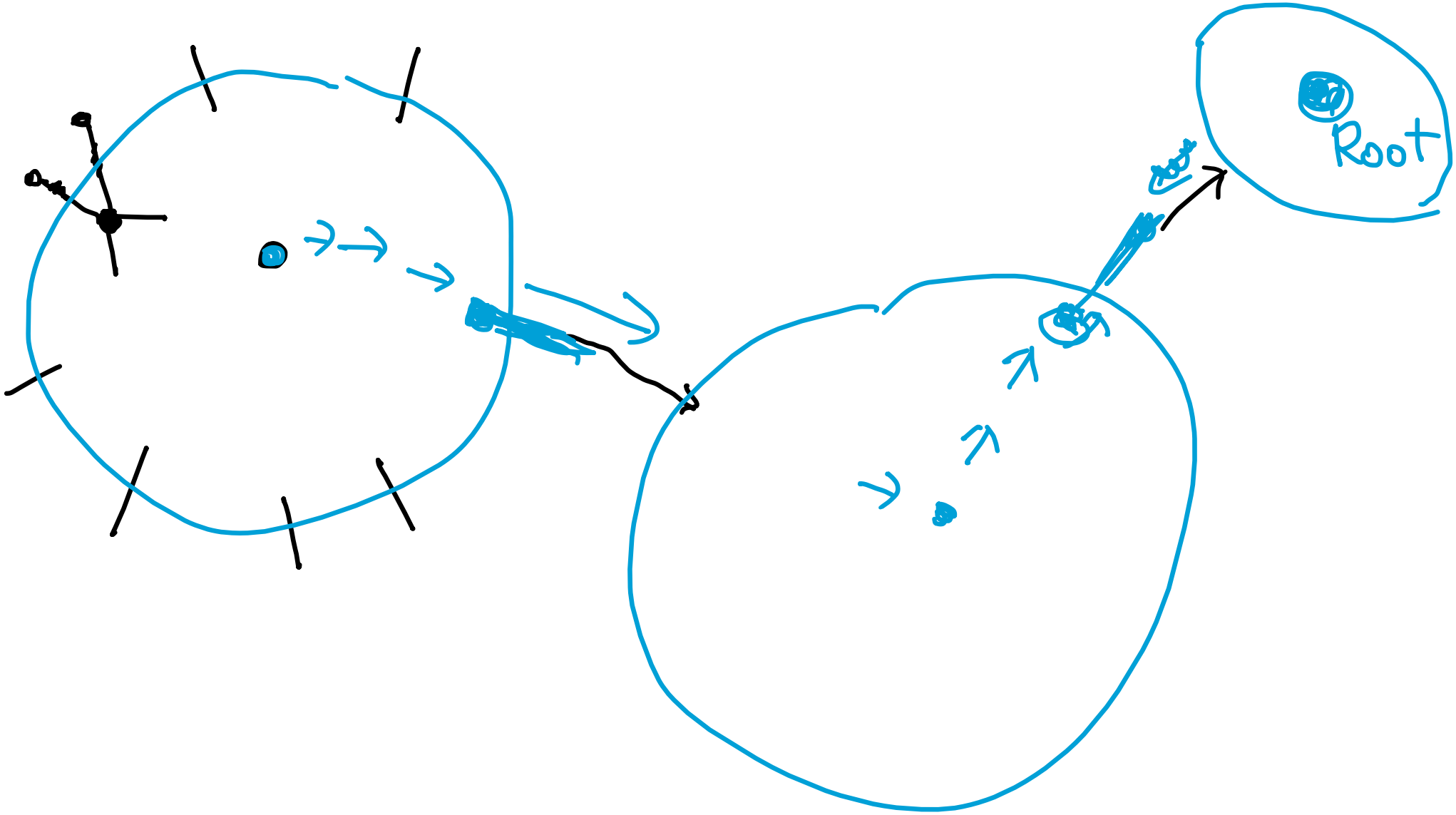
blue edge must belong to the min. ST

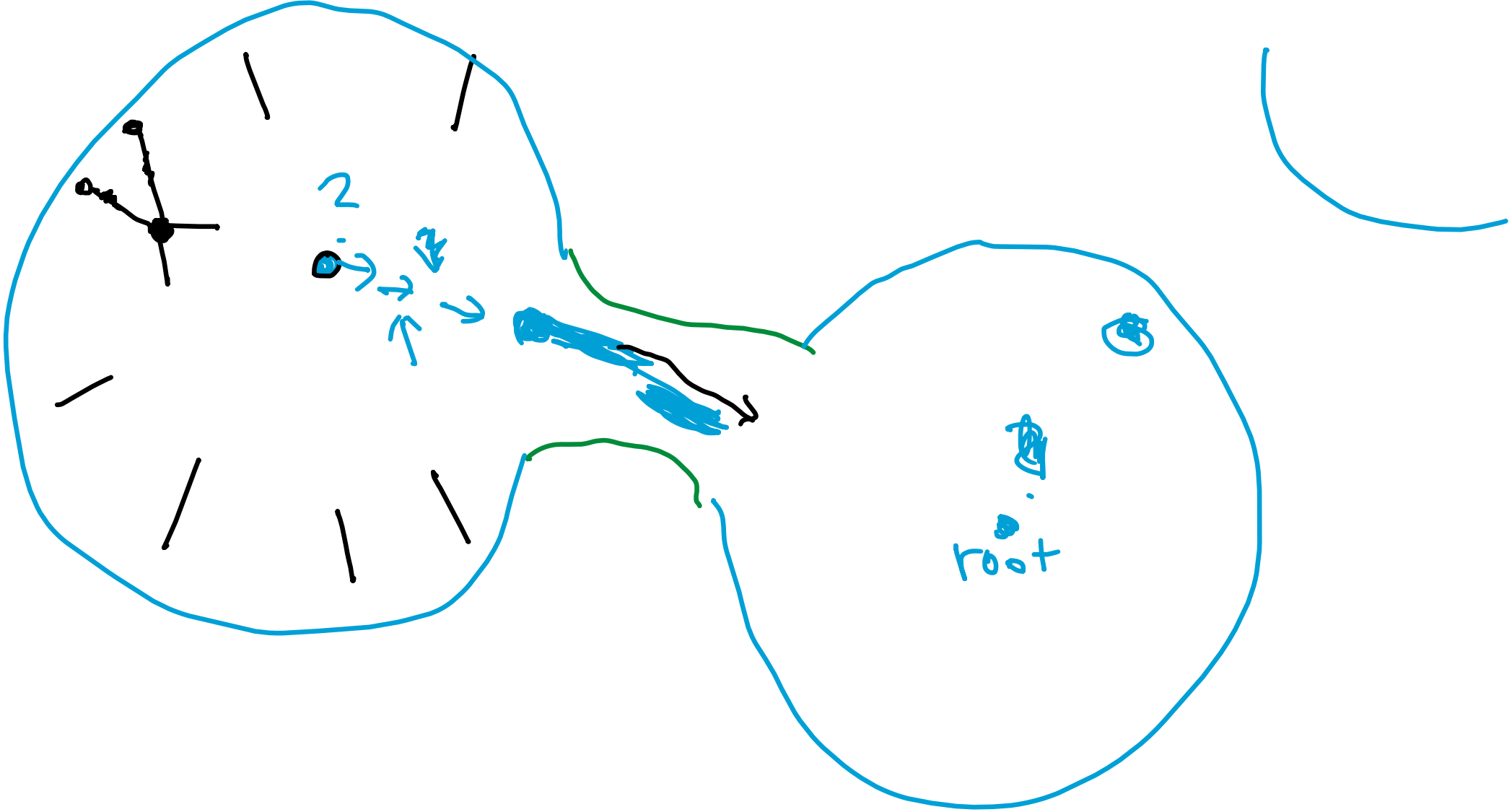
Lemma 2.17. *For a given weighted graph G (such that no two weights are the same), let T denote the MST, and T' be a fragment of T. Then the blue edge of T' is also part of T, i.e., $T' \cup b(T') \subseteq T$.*

Algorithm 2.18 GHS (Gallager–Humblet–Spira)

1: Initially each node is the root of its own fragment. We proceed in phases:
2: **repeat**
3: All nodes learn the fragment IDs of their neighbors.
4: The root of each fragment uses flooding/echo in its fragment to determine the blue edge $b = (u, v)$ of the fragment.
5: The root sends a message to node u; while forwarding the message on the path from the root to node u all parent-child relations are inverted {such that u is the new temporary root of the fragment}
6: node u sends a merge request over the blue edge $b = (u, v)$.
7: **if** node v also sent a merge request over the same blue edge $b = (v, u)$ **then**
8: either u or v (whichever has the smaller ID) is the new fragment root
9: the blue edge b is directed accordingly
10: **else**
11: node v is the new parent of node u
12: **end if**
13: the newly elected root node informs all nodes in its fragment (again using flooding/echo) about its identity
14: **until** all nodes are in the same fragment (i.e., there is no outgoing edge)




Root


2
root

rounds: logarithmic number in number of nodes

round: broadcast + echo + reordering

⋮