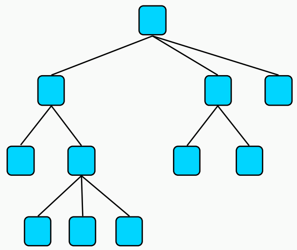
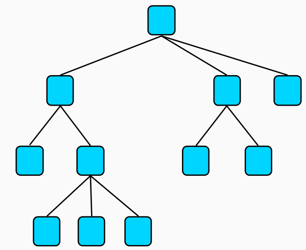
Computations in Trees

Saturation
Minimum Finding
Eccentricity
Center
Ranking



Trees

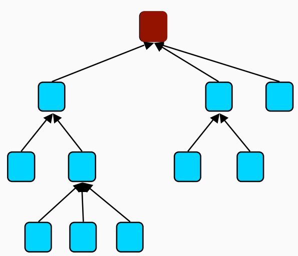
- Acyclic graph
- n entities
- n - 1 links



Rooted Trees

- Acyclic graph
- n entities
- n - 1 links

Rooted

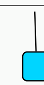


Saturation Technique

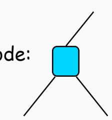
- Bidirectional links
- Ordered messages
- Full Reliability
- Knowledge of the topology

Note:

Each entity knows whether it is a leaf:



or an internal node:



SATURATION: A Basic Technique

S = {available, awake, processing }

At the beginning, all entities are available

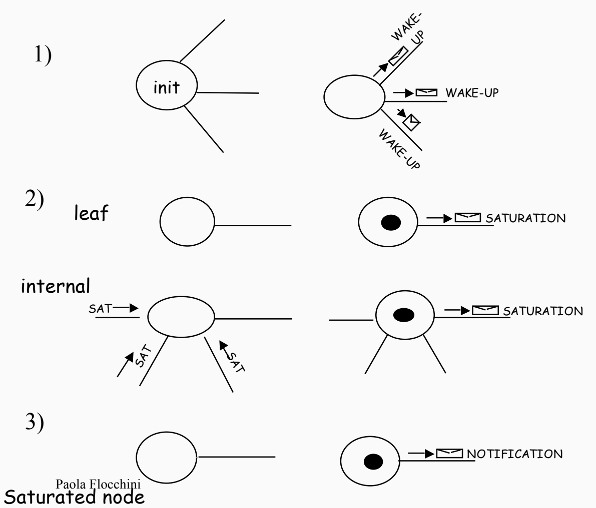
Arbitrary entities can start the computation (multiple initiators)

SATURATION: A General Technique

- **Activation phase:**
  started by the initiators: all nodes are activated *wake-up*

- **Saturation Phase:**
  started by the leaves: a unique pair of neighbours is identified (saturated nodes)

- **Resolution Phase:**
  started by the saturated nodes



S = {AVAILABLE, ACTIVE, PROCESSING, SATURATED}
Sinit = AVAILABLE

AVAILABLE `I haven't been activated yet`

```
Spontaneously
    send(Activate)  to  N(x);
    Neighbours:= N(x)
    if |Neighbours|=1  then              /* special case if
         M:=("Saturation");                  I am a leaf */
         parent ⇐ Neighbours;
         send(M) to parent;
         become PROCESSING;
    else
         become ACTIVE;
```

Receiving(Activate)

```
    send(Activate) to N(x)- {sender};
    Neighbours:=  N(x);
    if |Neighbours|=1   then
            M:=("Saturation");
            parent ⇐ Neighbours;
            send(M)  to  parent;
            become PROCESSING;
    else
            become ACTIVE;
```


I haven't started the saturation phase yet

ACTIVE

Receiving(M)

```
    Neighbours:= Neighbours - {sender};
    if |Neighbours|=1 then
            M:=("Saturation");
            parent ⇐ Neighbours;
            send(M) to parent;
            become PROCESSING;
```

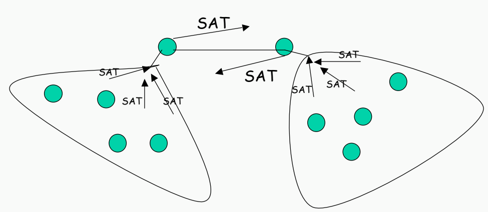
PROCESSING — **I have already started the saturation phase**

receiving(M)

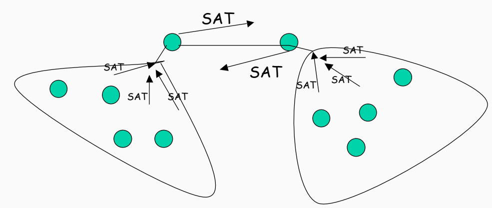
```
    become SATURATED;
```


Property:

Exactly two processing nodes become saturated, and they are neighbours.



A node becomes PROCESSING only after sending saturation to its parent



A node become SATURATED only after receiving a message in the state PROCESSING from its parent

TWO neighbouring entities become saturated

Which entities become saturated depends on the unpredictable delays

Observations and Examples

Message Complexity

Activation: Worst case - n initiators

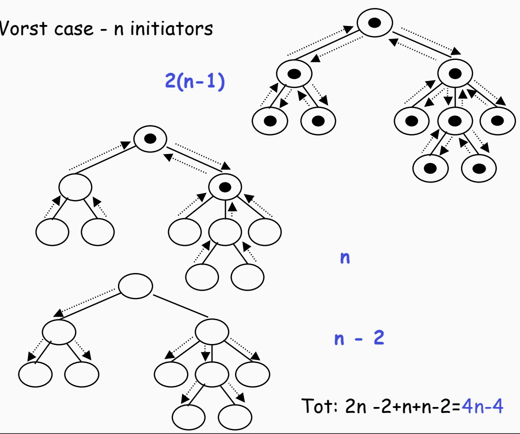
2(n-1)

Saturation:

n

Notification:

n - 2



Tot: 2n -2+n+n-2=4n-4

Message Complexity

Activation: In general - k* initiators

n +k* -2 (wake-up in the tree)

Saturation: n

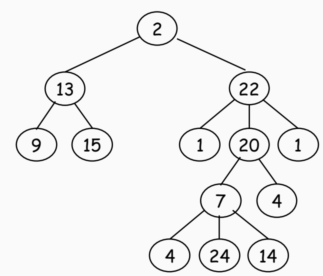
do not depend on number of initiators

Notification: n -2

TOT: 2n+k*-2

Put information in the saturation message

Minimum Finding



Entity x has in input value(x)

At the end each entity should know whether it is the minimum or not.

States S {AVAILABLE, ACTIVE, PROCESSING, SATURATED} Sinit = AVAILABLE

```
AVAILABLE

Spontaneously
        send(Activate) to N(x);
        min := v(x);
        Neighbours:= N(x)
        if |Neighbours|=1 then
                M:=("Saturation", min);
                parent ⇐ Neighbours;
                send(M) to parent;
                become PROCESSING;
        else become ACTIVE;
```



```
Receiving(Activate )
        send(Activate) to N(x) – {sender};
        min:=v(x);
        Neighbours:= N(x);
        if |Neighbours|=1 then
            M:=("Saturation", min);
            parent ⇐ Neighbours;
            send(M) to parent;
            become PROCESSING;
        else become ACTIVE;
```



```
ACTIVE
Receiving(M)
        min:= MIN{min, M}
        Neighbours:= Neighbours - {sender}};
        if |Neighbours|=1 then
                M:=("Saturation", min);
                parent ⇐ Neighbours;
                send(M) to parent;
                become PROCESSING;
```



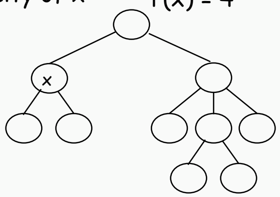
```
PROCESSING
receiving(M)
        min:= MIN{min, M}
        Notification:= ("Resolution", min)
        send (Notification) to N(x) -parent
        if v(x)=min then
                become MINIMUM
        else
                become LARGE

receiving(Notification)
        send(Notification) to N(x) -parent
        if v(x)=Received_Value then
                become MINIMUM;
        else
                become LARGE;
```

Finding Eccentricities

d(x,y) = distance between x and y

$\text{Max}_y\{d(x,y)\}$ = r(x) eccentricity of x

r(x) = 4



How to find the eccentricity of all nodes

Idea 1:

1) Every node broadcasts a request,
2) the leaves send up a message to collect the distances.

Complexity: $O(n^2)$

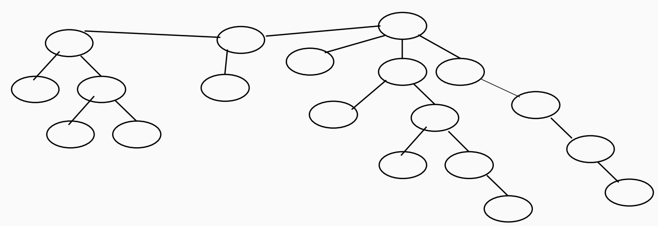
Other Idea:

Based on the saturation technique:

1) Find the eccentricity of the two saturated nodes
2) Propagate the needed info so that the other nodes can find their eccentricity (in the notification phase)

Complexity = saturation

Observations and Examples



States S {AVAILABLE, ACTIVE, PROCESSING, SATURATED} Sinit = AVAILABLE define Distance[ ]

AVAILABLE

Spontaneously

```
        send(Activate) to N(x);
        Distance[x]:= 0;
        Neighbours:=N(x)
        if |Neighbours|=1  then
                maxdist:= 1+ Max{Distance[*]}

                M:=("Saturation", maxdist);
                parent ⇐ Neighbours;
                send(M) to parent;
                become PROCESSING;
         else  become ACTIVE;
```


Receiving(Activate)

```
        send(Activate)to N(x) – {sender};
        Distance[x]:= 0;
        Neighbours:=  N(x);
        if |Neighbours|=1  then
                maxdist:= 1+ Max{Distance[*]}
                M:=("Saturation", maxdist);
                parent ⇐ Neighbours;
                send(M)  to  parent;
                become PROCESSING;
        else  become ACTIVE;
```


ACTIVE

Receiving(M)

```
        Distance[{sender}]:= Received_distance;
        Neighbours:= Neighbours - {sender}};
        if |Neighbours|=1 then
                maxdist:= 1+ Max{Distance[*]}
                M:=("Saturation", maxdist);
                parent ⇐ Neighbours;
                send(M) to parent;
                become PROCESSING;
```


PROCESSING

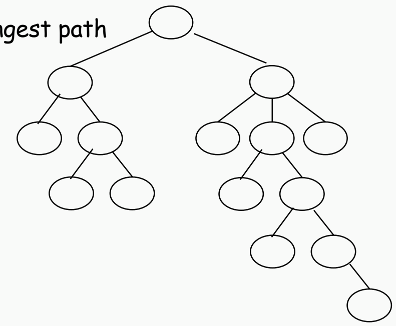
receiving(M)

```
        Distance[{ sender}]:= Received_distance;
        r(x):= Max { Distance[z]: z ∈  N(x) }
        for all  y ∈ N(x)-{parent} do
                maxdist:= 1+ Max{Distance[z]:
                                        z∈ N(x)- {y}}
                send("Resolution", maxdist)  to y
        endfor
        become DONE
```

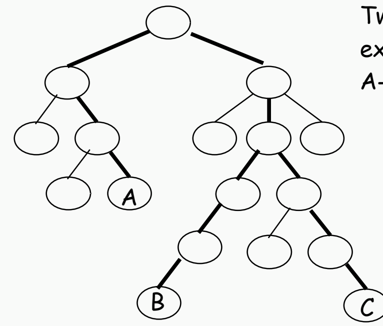
Center Finding

c is the center if $r(c) \leq r(x)$ for all x belonging to V. Max distance is minimized.

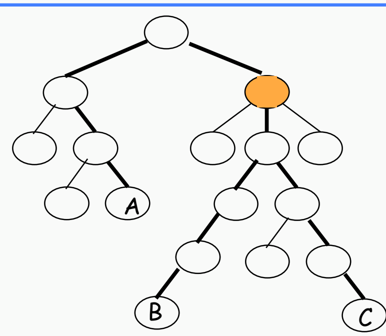
Diametral path: Longest path



Two diameters in this example:
A-B, and A-C



The center is the node with smallest eccentricity



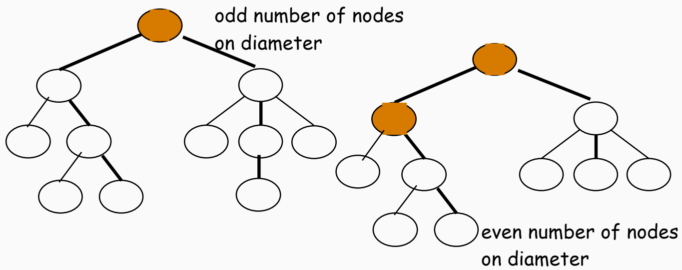
One Idea:

1) Find all the eccentricities — 4n-4
2) Find the smallest — 2n-2

→ 6n -6

A better strategy can be devised exploiting properties of the center.

Property 1: In a tree there is a unique center, or there are two centers that are neighbours



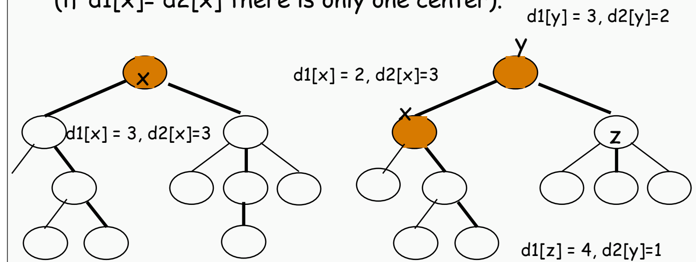
Proposition 2: Centers are on diametral paths.

d1[x] = max dist   d2[x] = second max dist (through different neighbours)

Property 3: A node x is a center iff

$$d1[x] - d2[x] \le 1$$

(if d1[x]= d2[x] there is only one center).



Another Idea:

1) Find all the eccentricities
2) Each node can find out locally whether it is the center or not

4n-4

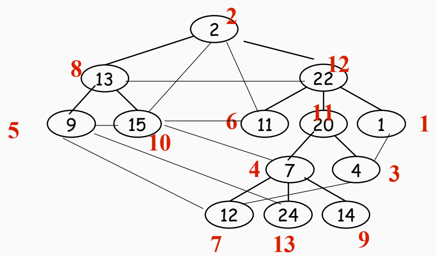
Yet another better idea:

1) Find the eccentricities of the saturated nodes   3n-2
2) Locally check if I am the center (checking largest and second largest)
3) If I am NOT the center, propagate the distance information ONLY in the direction of the center
≤ n/2

How do I know the direction of the center ?

Examples ....   ≤ 3.5 n-2

Ranking

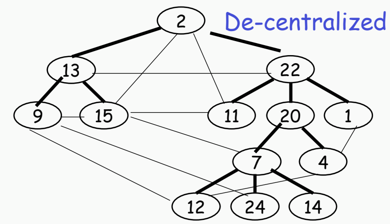


Paola Flocchini

In an arbitrary network:

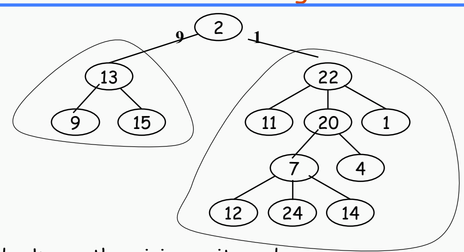
1) Find a spanning tree
2) Use saturation+ minimum finding to find a starting node
3) Do-ranking

Phase 3 can be: Centralized / De-centralized



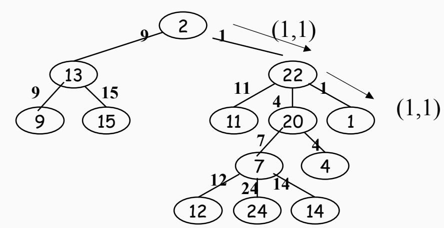
Paola Flocchini

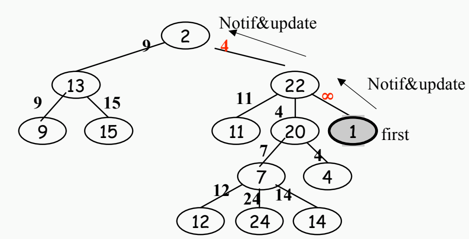
Centralized Ranking



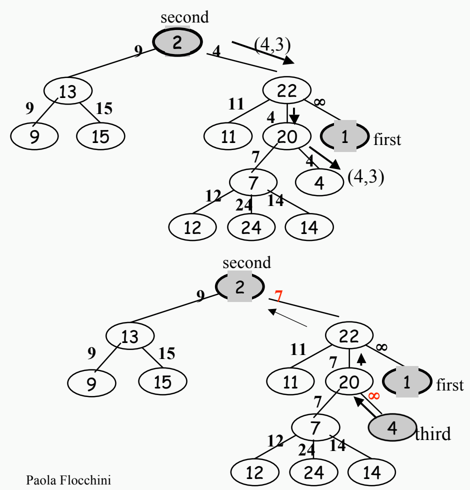
The leader knows the minimum, it sends in that direction a ranking message
Every node knows the minimum in its subtrees, they can then forward the ranking message in the right direction
When the node to be ranked receives the message It sends up a notification&update message that will travel up to the leader

Paola Flocchini



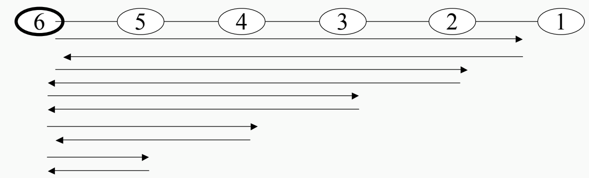


Paola Flocchini

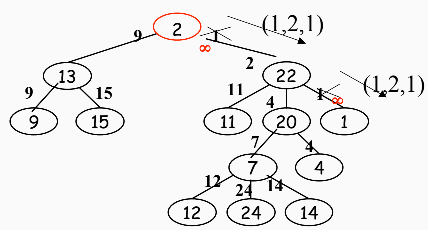


Etc…

Complexity: worst case



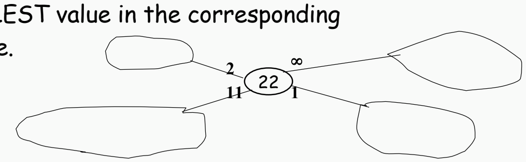
Decentralized Ranking



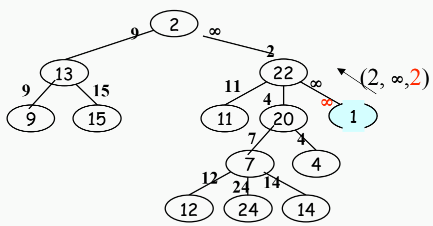
The starter node send a ranking message of the form:
(**first**, **second**,**rank**) in the direction of first.

first: smallest value
second: second smallest known **SO FAR**
(this is a guess on the value that has to be ranked after first)

The value on a link indicates the SMALLEST value in the corresponding subtree.



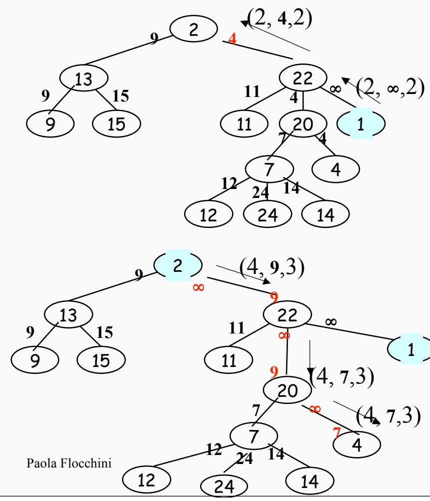
If no value is indicated (or the value is ∞) it means that the smallest in the corresponding subtree is unknown (for the moment)

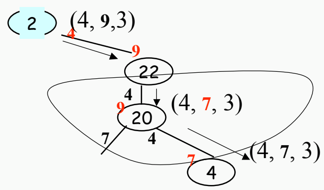


The ranked node attempts to send a ranking message to the next node to be ranked

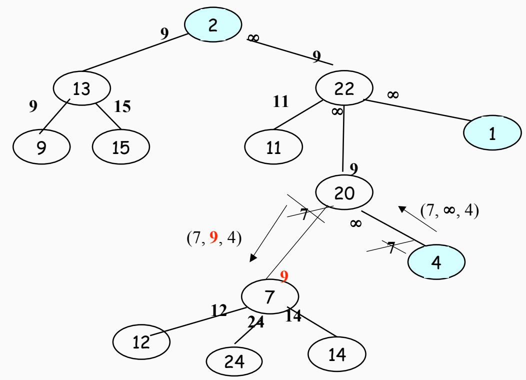
Second might now be unknown, in this case the value ∞ is used

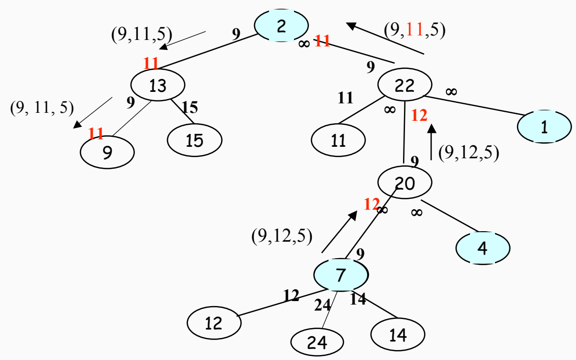
The second variable of the rank message is updated during its travel and the minimum values on the links of the tree are also updates

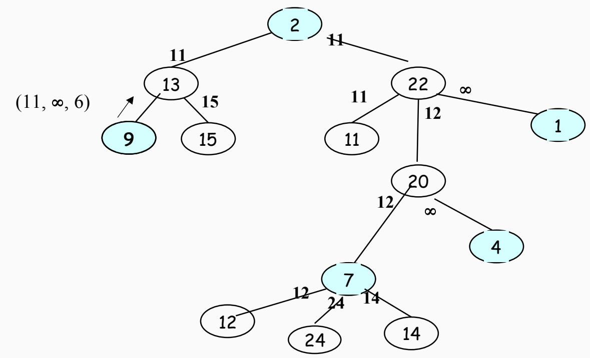




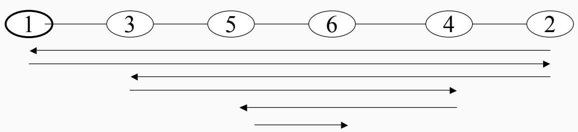
Notice the update







Complexity: worst case



Paola Flocchini