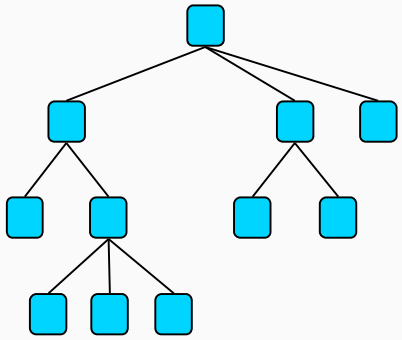
Calcul dans des topologies spécifiques:Arbres

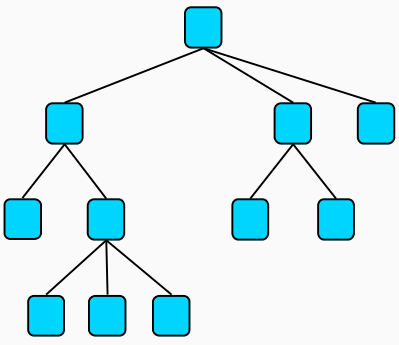
Saturation
Minimum Finding
Eccentricity
Center
Ranking



Paola Flocchini

Arbres

- Graphes acycliques
- n entités
- n - 1 liens

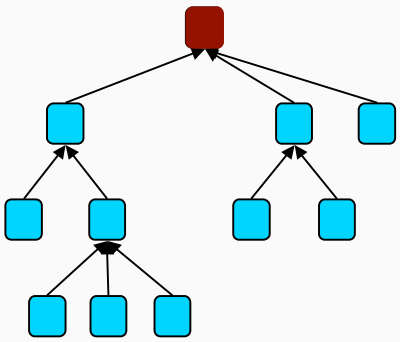


Paola Flocchini

Arbres avec racine

- Graphes acycliques
- n entités
- n - 1 liens

racine



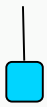
Paola Flocchini

Technique de Saturation

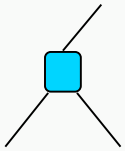
- Liens bidirectionnels
- Messages ordonnés
- Fiabilité
- Connaissance de la topologie
- Identités distinctes

Note:

Chaque entité peut comprendre:
si elle est une feuille



ou un noed interne



Paola Flocchini

Saturation

S = {available, awake, processing }

- Plusieurs initiateurs
- Au début, les noeuds sont soit initiateurs ou endormis.

Paola Flocchini

Saturation

1. Phase de réveil ("wake-up")

Commencé par les initiateurs. Le but est d' activer toutes les entités

wake-up

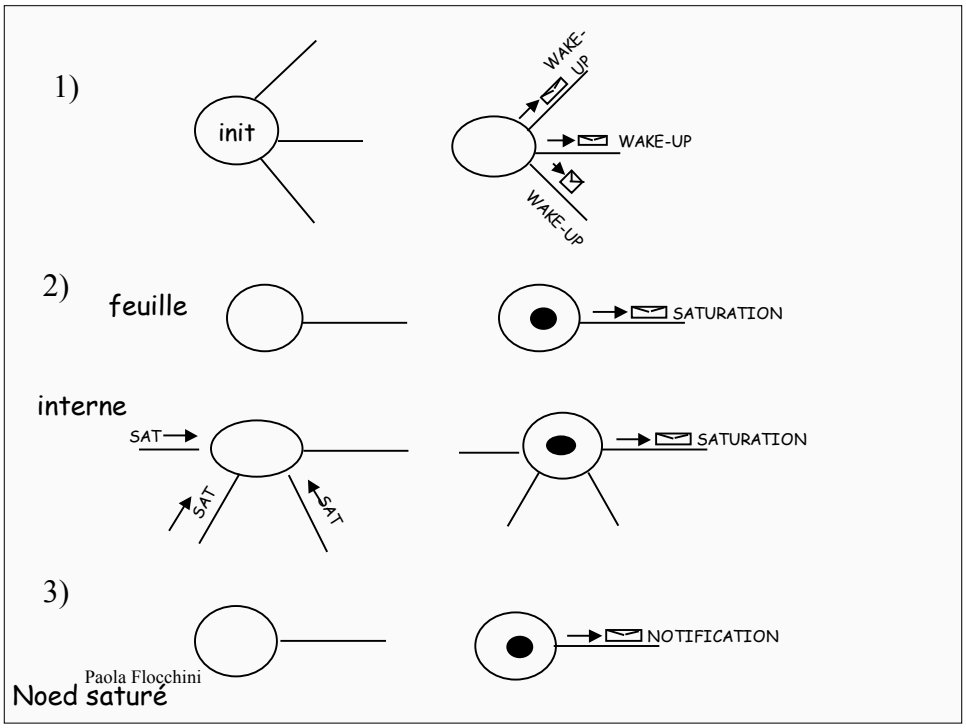
2. Phase de saturation

Commencé par les feuille. Le but est d'identifier deux entités.

3. Phase de notification

Commencé par les noeds saturés. Le but est de diffuser l'information et terminer l'algorithme

Paola Flocchini



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

AVAILABLE Pas encore activé

```
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;
```

Paola Flocchini

```
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;
```

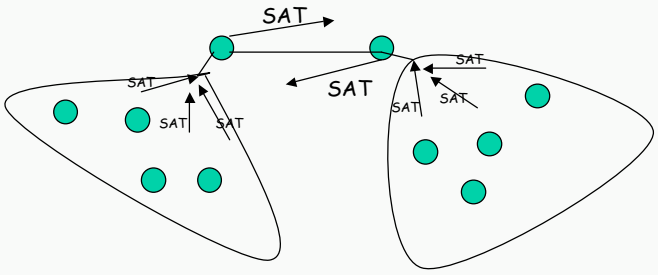

Pas encore commencé la phase de saturation

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

```
PROCESSING                    commencé la phase de
                              saturation
receiving(M)
        become SATURATED;
```

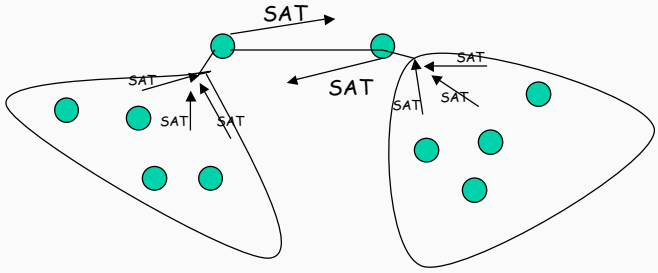
Proprieté:

À la fin, les deux entités sont saturées et elle sont des voisins



Paola Flocchini

Un noed devient PROCESSING seulement après avoir envoyé un message de saturation a son parent.



Un noed devient SATURATED seulement après avoir reçu un message dans l'état PROCESSING.

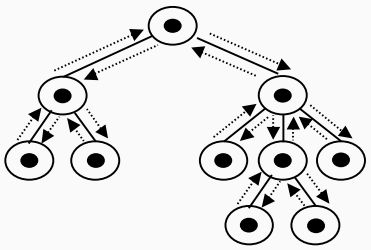
---> Deux entités voisines devient saturées

Paola Flocchini

Lesquelles devient saturées dépend de délais, qui sont inconnu.

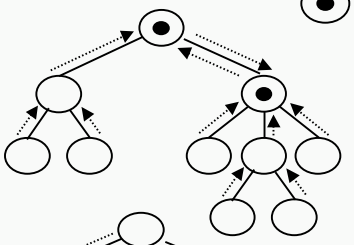
Complexité en message

Réveil: Cas pire - n initiateurs



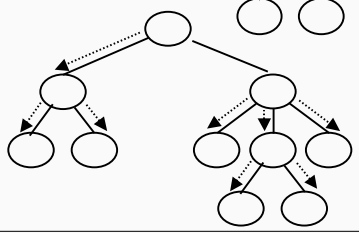
2(n-1)

Saturation:



n

Notification:



n - 2

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

Complexité en message

Réveil : En general - k* initieteurs

n +k* -2 (wake-up dans l'arbre)

Saturation: n

Il n'a rien a voir avec le nombre d'initiateurs

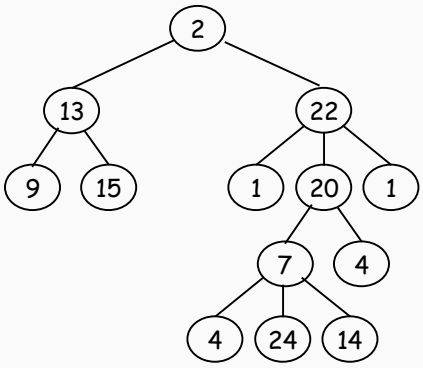
Notification: n -2

TOT: 2n+k*-2

Paola Flocchini

Mettre information dans le message de saturation

Minimum Finding



L'entité x a comme input value(x)

A la fin, chaque entité doit savoir si elle possède le minimum ou pas.

Paola Flocchini

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;
```



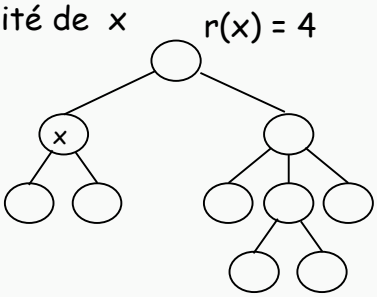
```
PROCESSING
receiving(M)
        min:= MIN{min, M}
        Notification:= ("Resolution", min)
        send (Notification)  to N(x) -parent
        if v(x)=min then
                become MINIMUM
        else
                become LARGE

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

Les Eccentricités

d(x,y) = distance entre x et y

$\max_y\{d(x,y)\}$ = r(x) eccentricité de x



Comment trouver les eccentricités de tous les noeds …

Paola Flocchini

Ideé 1:

1) Chaque noeud diffuse une requête,
2) Les feuilles envoient des requêtes qui voyagent sur l'arbre pour calculer les distances.

Complexité : $O(n^2)$

Paola Flocchini

Une autre ideé:

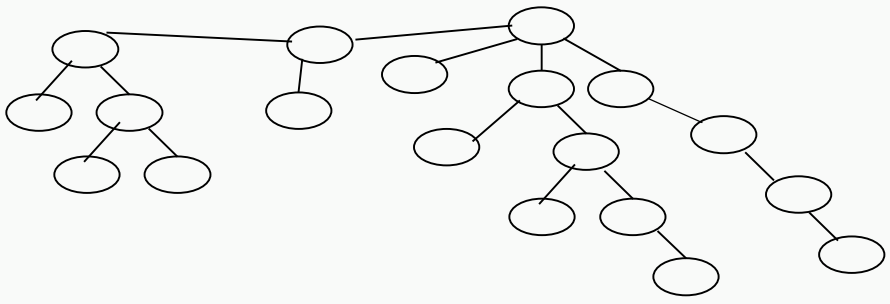
Base sur la technique de saturation

1) Trouver l'eccentricité de deux noeuds satures
2) Propager l'info nécessaire pour que les autres noeuds puissent calculer leur eccentricités (dans la phase de notification).

Complexité = complexité de la saturation

Paola Flocchini

Observations and Exemples



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

Paola Flocchini

definir 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;

Paola Flocchini

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;

Paola Flocchini

```
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;
```



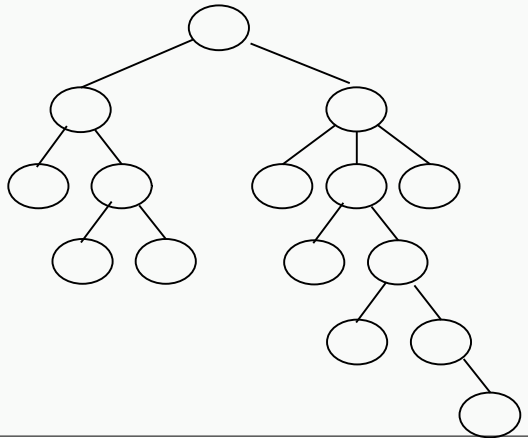
```
PROCESSING
receiving(M)
        Distance[{ sender}]:= Received_distance;
        r(x):= Max { Distance[z]: z ∈ N(x) }
        for all y ∈ N(x)-{parent} do
                maxdist:= 1+ Max{Distance[z]:
                                z∈ N(x)- {y}
                send("Resolution", maxdist) to y
 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

Recherche du centre

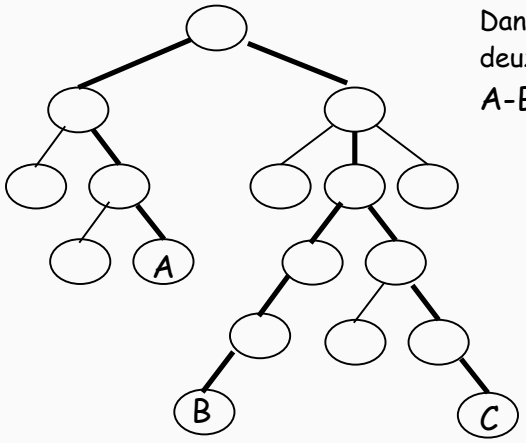
c est le centre de l'arbre si $r(c) \leq r(x)$ pour tout $x \in V$.

La distance maximale est minimizée.

Chemin diamétral: Chemin le plus long



Paola Flocchini

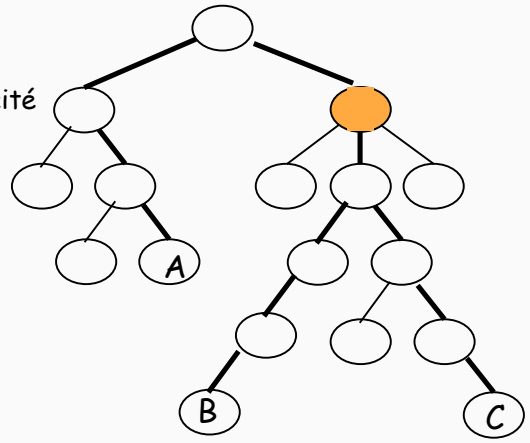


Dans cet exemple il y a deux diametres:

A-B et A-C

Paola Flocchini

Le centre est le noeud
avec la plus petite eccentricité



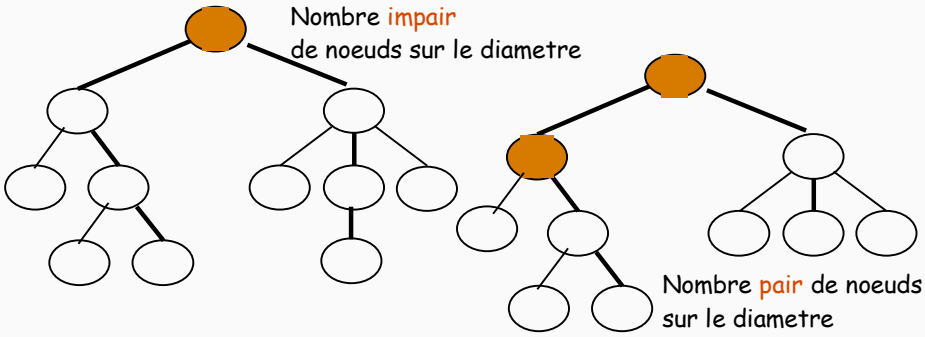
Une idée:

1) TROUVE TOUTES LES ECCENTRICITÉS
2) TROUVE LA PLUS PETITE

4n-4
2n-2
→ 6n -6

Une meilleure stratégie peut être appliquée en utilisant des propriétés du centre.

Property 1: Dans un arbre il y a un seul centre, ou il y en a deux (voisins)



Proposition 2: Les centres sont sur des chemins diamétraux.

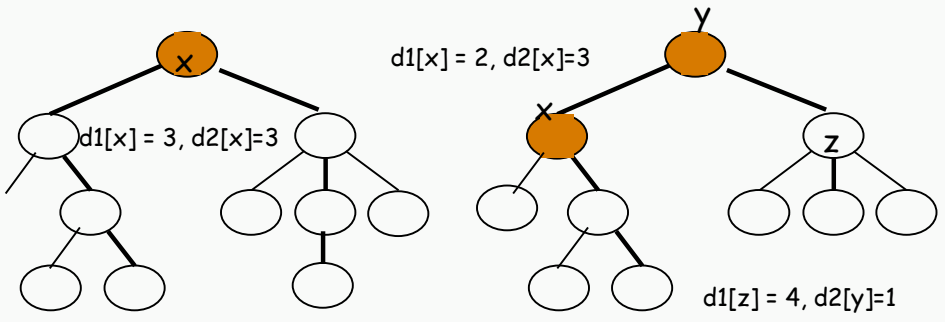
Paola Flocchini

d1[x] = max dist d2[x] = deuxieme max dist

Property 3: Un noeud x est un centre si et seulement si

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

(si d1[x]= d2[x] il y a un centre unique).



Paola Flocchini

Une autre idée:

1) Trouve toutes les eccentricités
2) Chaque noeud peut comprendre si il est un centre ou pas

4n-4

Une idée encore meilleure

1) Trouve les eccentricités des noeud saturés 3n-2
2) Vérifie localement si je suis un centre (en regardant les deux eccentricités majeures)
3) Si je ne suis pas le centre, propage l'information Seulement dans la direction du centre

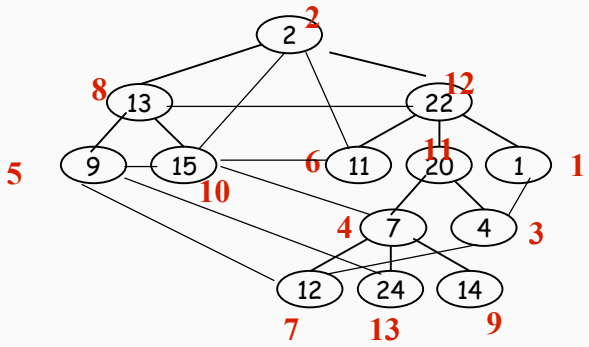
≤ n/2

Tot ≤ 3.5 n-2

Comment savoir quelle est la direction du centre ?

Exemples .....

Ranking

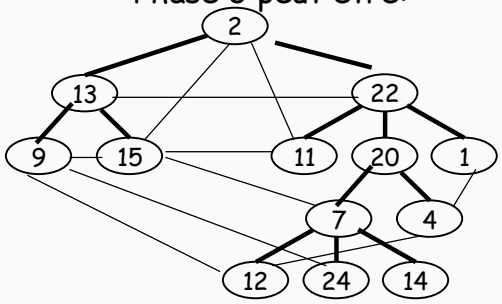


Paola Flocchini

Dans un graphe arbitraire

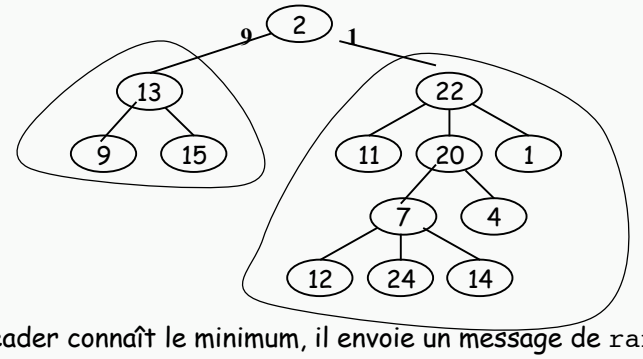
1) Trouve un arbre couvrant
2) Utilise saturation et minimum-finding pour identifier une racine (leader)
3) Applique l'algorithme de ranking

Phase 3 peut être: Centralisé Décentralisé



Paola Flocchini

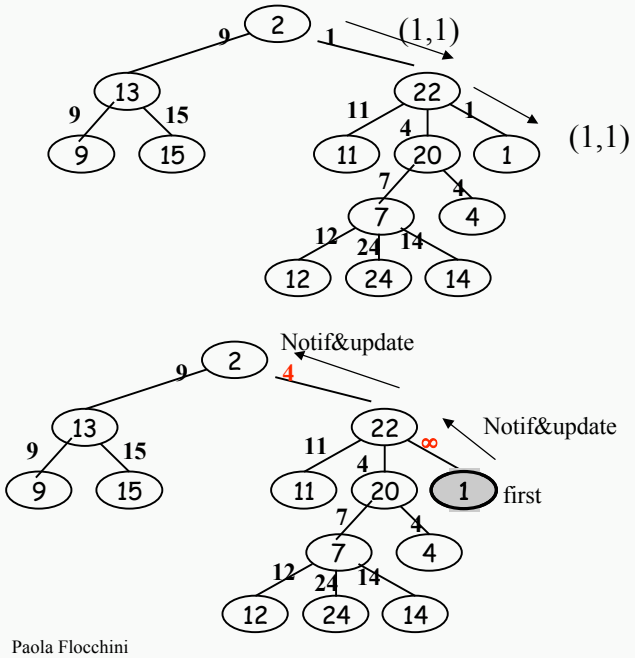
Ranking Centralisé

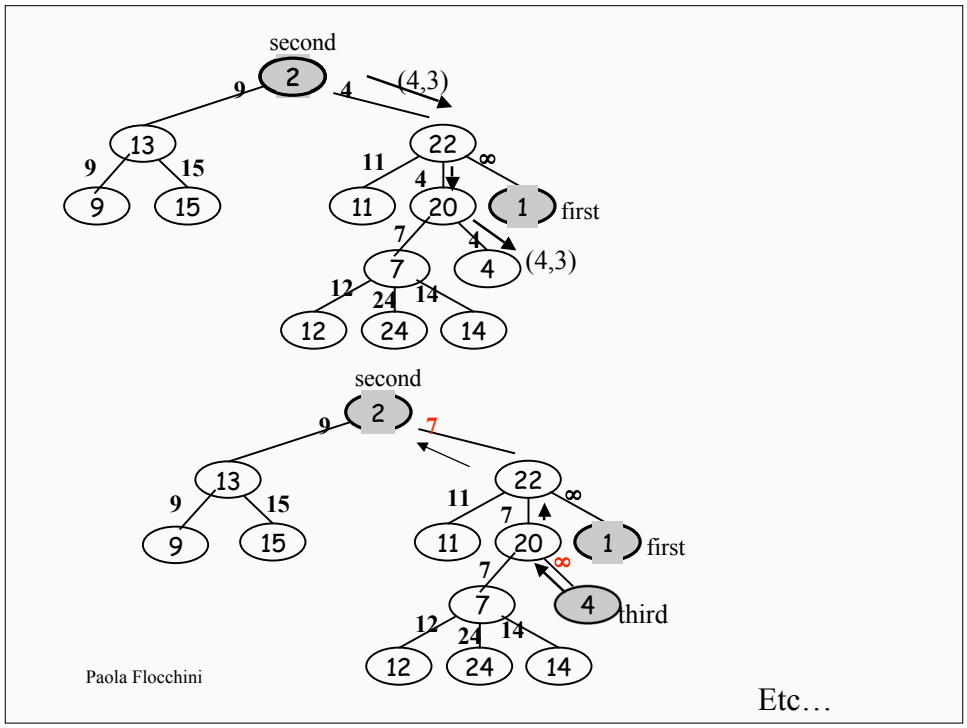


--- Le leader connaît le minimum, il envoie un message de `ranking` dans cette direction.

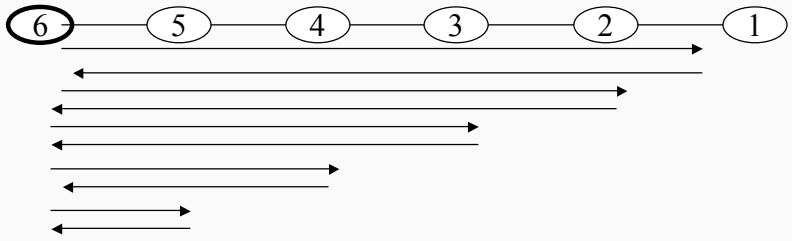
--- Chaque noeud connaît le minimum dans son sous arbre, il peut alors acheminer le message de `ranking` dans la direction correcte.

--- Quand le noeud reçoit son message de `ranking` , il envoie un message de `notification&update` qui va voyager jusqu'au leader.



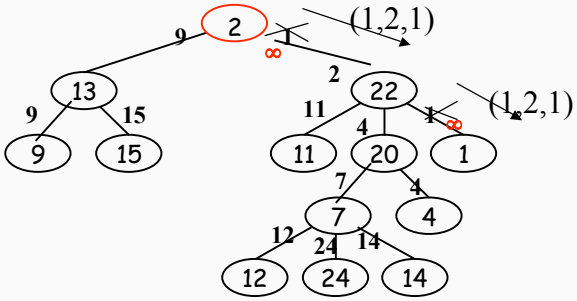


Complexité: Cas Pire



Paola Flocchini

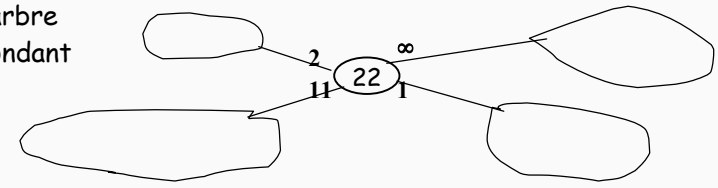
Ranking Décentralisé



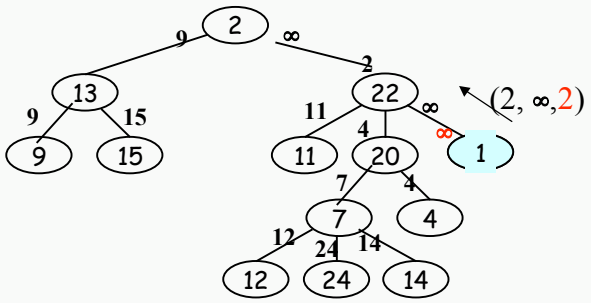
Le noeud de départ envoie un message de ranking de la forme: (**first**, **second**,**rank**) dans la direction du minimum.

first: valeur la plus petite
second: deuxième plus petite connue JUSQU'A MAINTENANT
(une devine de la valeur a classer après **first**)

La valeur sur un lien indique la plus petite valeur dans le sous-arbre correspondant



Si la valeur est le symbole spécial ∞
La plus petite valeur dans le sous-arbre est inconnue (pour l'instant)

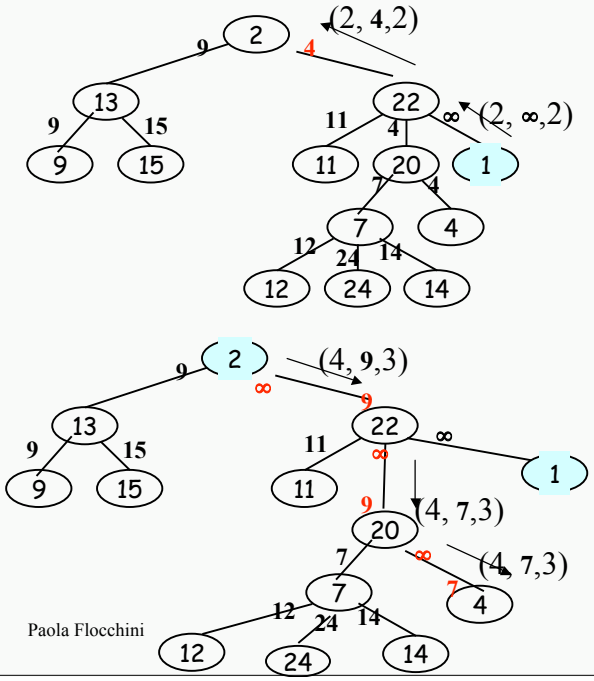


--- Le noeud classé essaye d'envoyer le message au prochain (second).

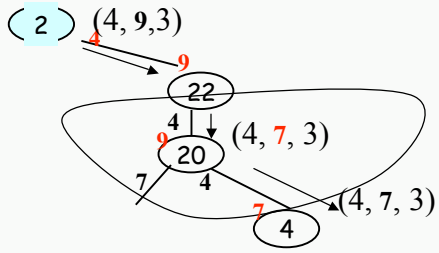
--- **Second** pourrait être inconnu, dans ce cas la valeur ∞ est utilisé.

--- La deuxième variable du message est mise a jour pendant le voyage du message de ranking et en même temps les valeurs sur les liens sont mises a jour aussi.

Paola Flocchini

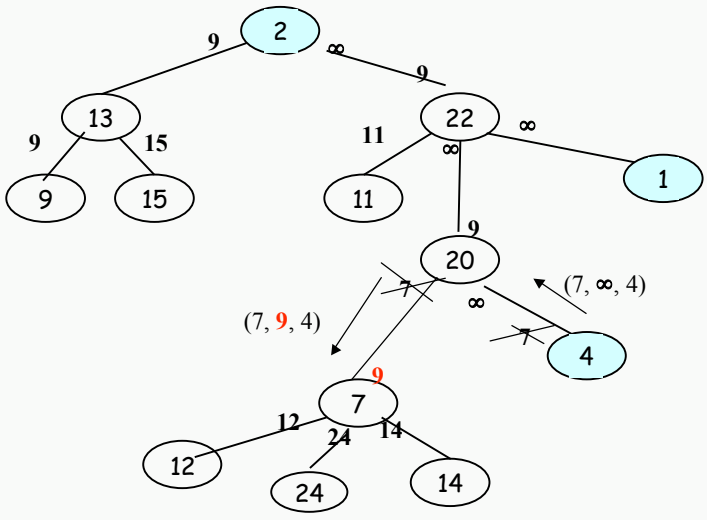


Paola Flocchini

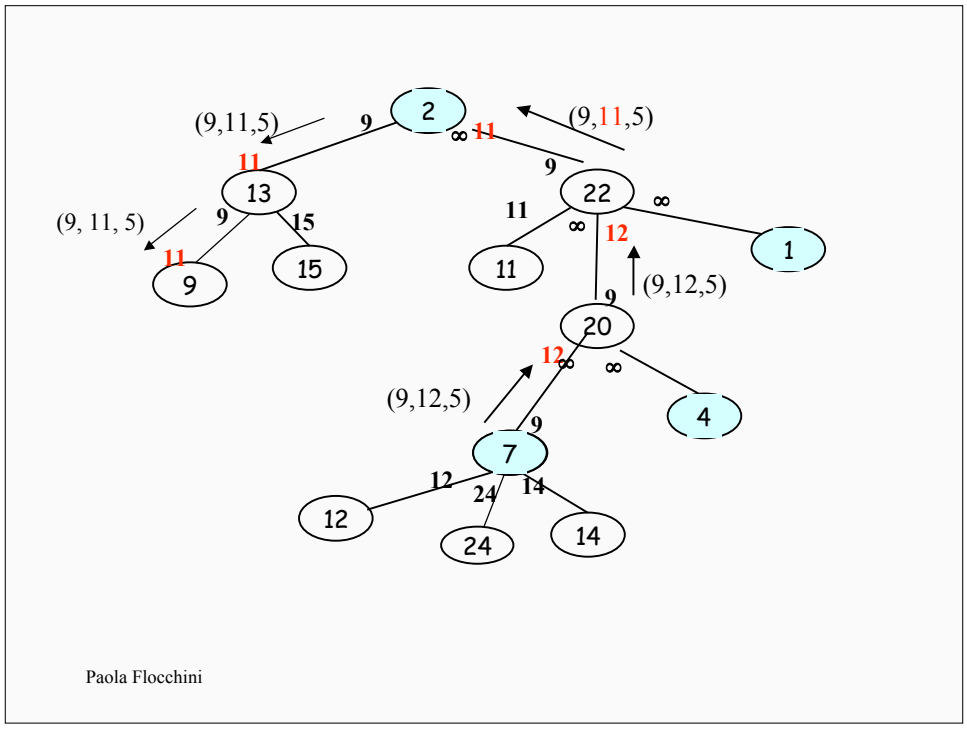


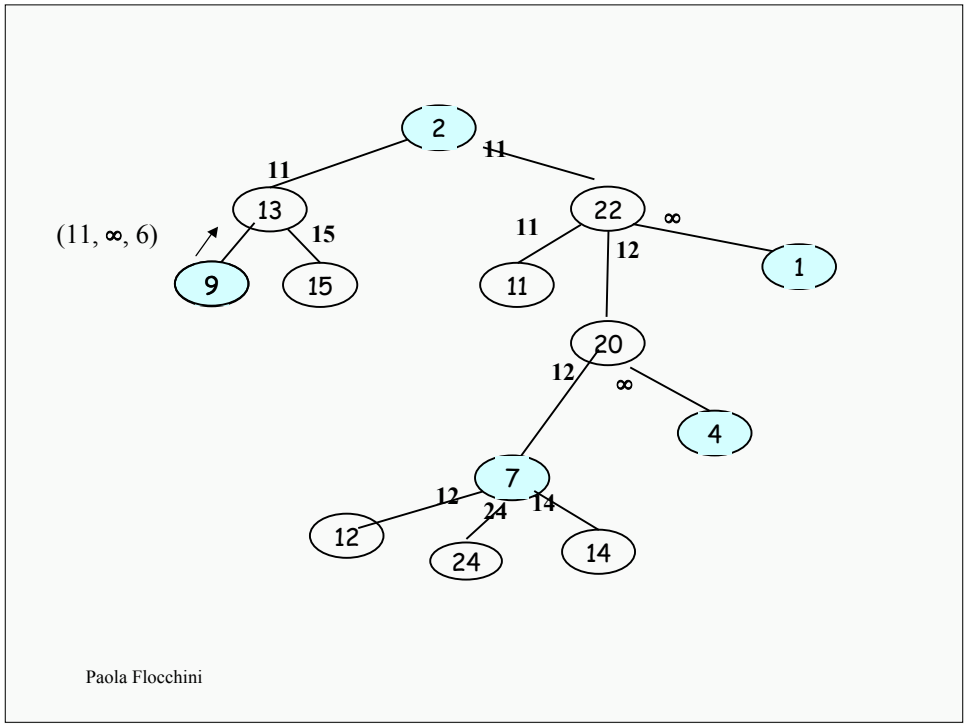
Remarquez la mise-a-jour

Paola Flocchini

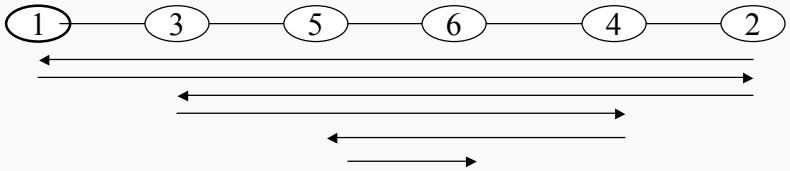


Paola Flocchini


(9,11,5)
(9,11,5)
(9, 11, 5)
(9,12,5)
(9,12,5)
2
13
22
9
15
11
1
20
4
7
12
24
14
Paola Flocchini


(11, ∞, 6)
2
13
22
9
15
11
1
20
4
7
12
24
14
Paola Flocchini

Complexité: Cas Pire



Paola Flocchini