Le Modèle et Opérations de bases

Chapitre 1 et 2

Le Modèle

Diffusion

Construction d'un arbre recouvrant

Parcours

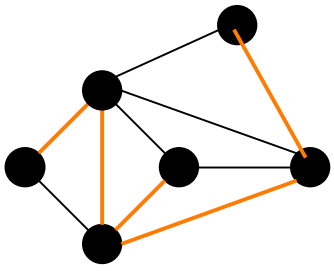
Wakeup

Construction d'un arbre recouvrant

Un arbre recouvrant T d'un graph G = (V,E) est un sous-graph acyclique de G tel que T=(V,E') et E' $\subset$ E.

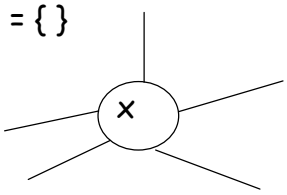
Hypothèses:

initiateur unique
liens bidirectionnels
fiabilité complète
G connexe



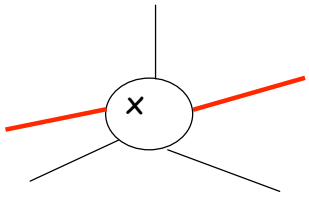
Protocole SHOUT

Initialement: ∀ x, Tree-neighbors(x) = { }



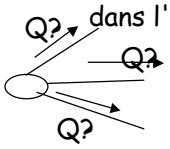
À la fin:

∀ x, Tree-neighbors(x) = {liens faisant parti de l'arbre recouvrant}



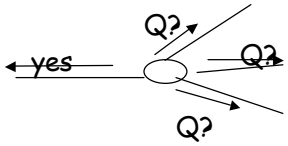
Q? = Voulez-vous être mon voisin dans l'arbre recouvrant?

1. init

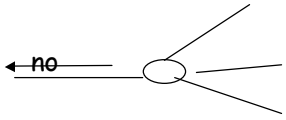


2. Q?

Si c'est la première fois:



Si le noeud a déjà répondu oui à quelqu'un d'autre:



Example

States S={INITIATOR, IDLE, ACTIVE, DONE}
Sinit = {INITIATOR, IDLE}
Sterm = {DONE}

INITIATOR
Spontaneusly
root:= true
Tree-neighbours := { }
send(Q) to N(x)
counter:=0
become ACTIVE

IDLE
receiving(Q)
root:= false
parent := sender
Tree-neighbours := {sender}
send(yes) to sender
counter := 1
if counter = |N(x)| then
become DONE
else
send(Q) to N(x) – {sender}
become ACTIVE

ACTIVE

receiving(Q)

send(no) to sender

receiving(yes)

Tree-neighbours:=
Tree-neighbours ∪ sender
counter := counter +1
if counter = |N(x)|
become DONE

receiving(no)

counter := counter +1
if counter = |N(x)|
become DONE

Note: SHOUT = FLOOD +REPLY

Exactitude et Terminaison

- Si x est un voisin de y dans l'arbre, y est aussi un voisin de x

- Si x envoie OUI à y, alors x est un voisin de y dans l'arbre et est lié à l'initiateur par une chaine de OUI

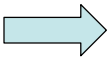
- Tout noeud x (à l'exception de l'initiateur) envoie un seul OUI

Le graph recouvrant défini par la relation "voisin dans l'arbre" est connexe et contiens tous les entités

Note: terminaison locale

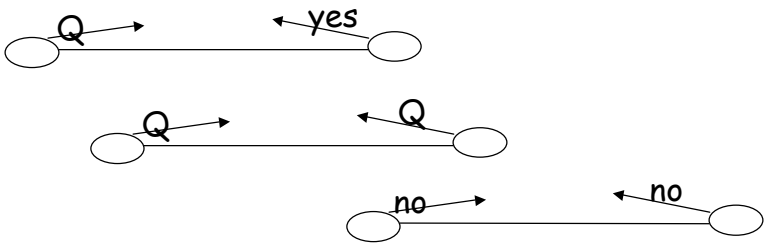
Complexité en Message

SHOUT = FLOOD + REPLY

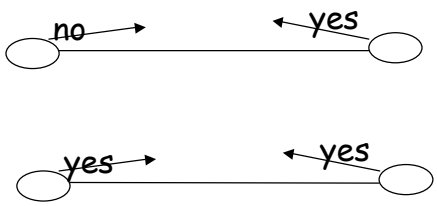


Messages(SHOUT) = 2 M(FLOOD)

Situations possibles



Situations impossibles

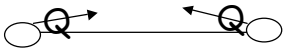


Complexité en Message – Pire Cas

Nombre de questions:

(n-1)

Une seule question sur les liens appartenant a l'arbre recouvrant



m -(n-1)

2 sur les autres

Total: $2(m-(n-1)) + (n-1)$
$= 2m - n + 1$

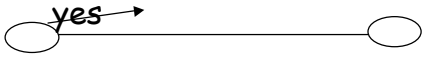
Complexité en Message – Pire Cas

Nombre de NON:

Autant que de liens où 2 questions on été échangés

$2(m - (n-1))$

Nombre de OUI:



exactement (n-1)

Complexité en Message – Pire Cas

$2m - n + 1 + 2(m - (n-1)) + n-1$
$= 2m - n + 1 + 2m - 2n + 2 + n - 1$
$= 4m - 2n + 2$

Messages(SHOUT) = $4m - 2n + 2$

En fait: M(SHOUT) = 2 M(FLOOD) = $2(2m-n+1)$

$\Omega(m)$ est aussi la limite inférieure dans ce cas

Construction d'un Arbre Recouvrant

Sans "NO"

Protocole SHOUT+

States S={INITIATOR, IDLE, ACTIVE, DONE}
Sinit = {INITIATOR, IDLE}
Sterm = {DONE}

INITIATOR
Spontaneusly
root:= true
Tree-neighbours := { }
send(Q) to N(x)
counter:=0
become ACTIVE

IDLE
receiving(Q)
root:= false
parent := sender
Tree-neighbours := {sender}
send(yes) to sender
counter := 1
if counter = |N(x)| then
become DONE
else
send(Q) to N(x) – {sender}
become ACTIVE

ACTIVE

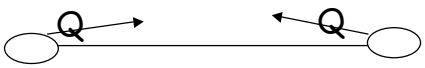
receiving(Q) **(to be interpreted as NO)**

```
counter := counter +1
if counter = |N(x)|
    become DONE
```

receiving(yes)

```
Tree-neighbours:=
    Tree-neighbours ∪ {sender}
counter := counter +1
if counter = |N(x)|
    become DONE
```

Sur chaque lien, il y aura exactement 2 messages:

Soit 

ou 

Messages(SHOUT+) = 2m

Beaucoup mieux que:

Messages(SHOUT) = 4m -2n + 2

Construction d'un Arbre Recouvrant

Avec Notification

States S={INITIATOR, IDLE, ACTIVE, DONE}
Sinit = {INITIATOR, IDLE}
Sterm = {DONE}

INITIATOR
Spontaneusly

```
root:= true
Tree-neighbours := { }
send(Q) to N(x)
counter:= 0
ack-counter:= 0
become ACTIVE
```

IDLE
receiving(Q)

```
root:= false
parent := sender
Tree-neighbours := {sender}
send(yes) to sender
counter := 1
ack-counter:= 0
if counter = |N(x)| then
    CHECK
else
    send(Q) to N(x) – {sender}
become ACTIVE
```

ACTIVE

receiving(Q)

counter := counter +1

if counter = |N(x)| and not root then

CHECK

receiving(yes)

Tree-neighbours:=

Tree-neighbours ∪ {sender}

counter := counter +1

if counter = |N(x)| and not root then

CHECK

ACTIVE (cont)

receiving(Ack)

ack-counter:= ack-counter +1

if counter = |N(x)| /* indicate tree-neighbors is done

if root then

if ack-counter = |Tree-neighbours|

send(Terminate) to Tree-neighbours

become DONE

else if ack-counter = |Tree-neighbours| - 1

send(Ack) to parent

receiving(Terminate)

send(Terminate) to Children

become DONE

CHECK

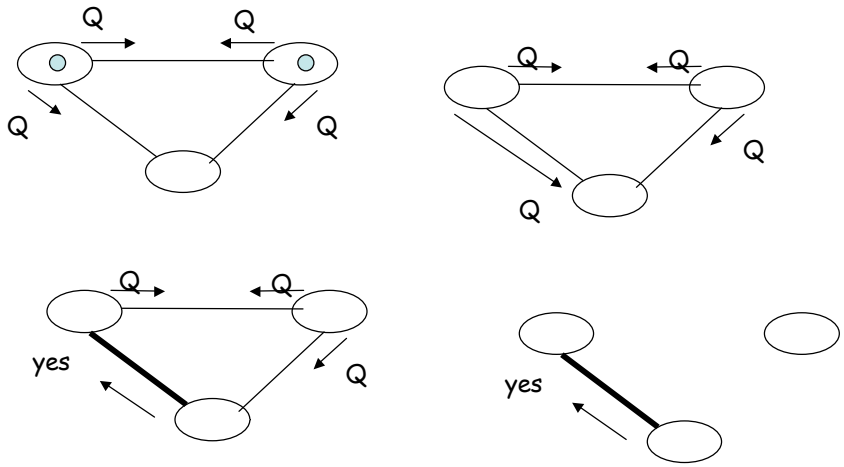
If I am a leaf

(* that is: Children:= Tree-neighbours – {parent}

if Children = emptyset *)

send(Ack) to parent

Qu'arrive-t-il s'il y a plusieurs initiateurs ?



Une élection est nécessaire pour obtenir un seul initiateur

ou

Un autre protocole doit être utilisé

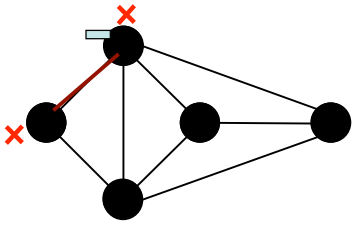
NOTE: L'élection est impossible si les entités ne possèdent pas d'identificateurs distinct

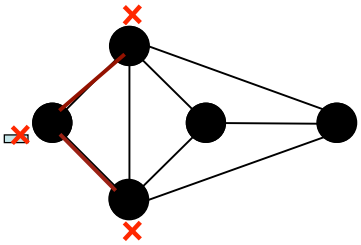
Parcours en Profondeur

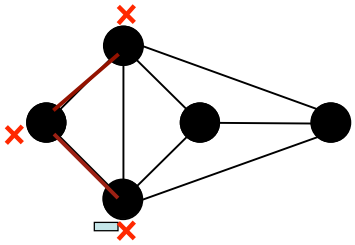
Suppositions

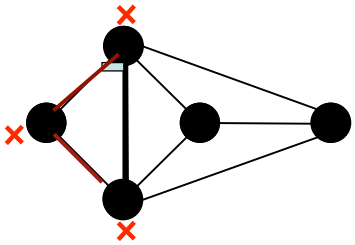
- Initiateur unique
- liens bidirectionnels
- aucune erreur
- *G* est connexe

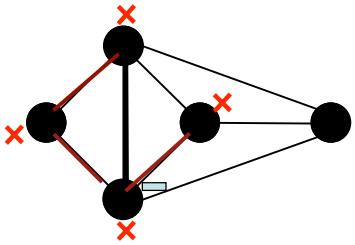
S = {INITIATOR, SLEEPING, ACTIVE, DONE}

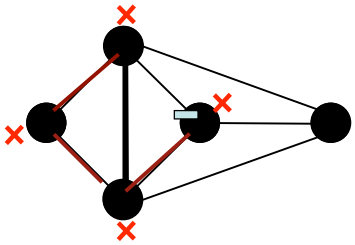


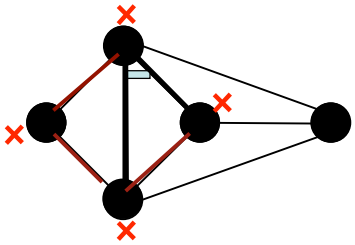


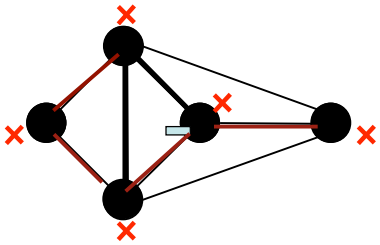


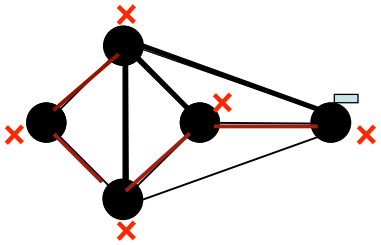


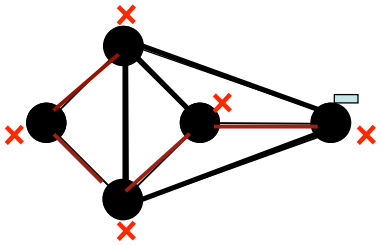


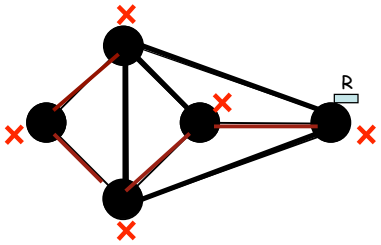


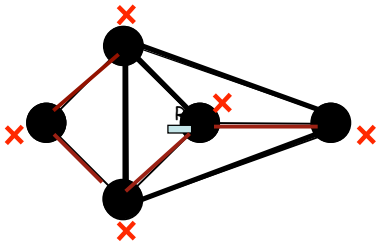









R


R

Version 1

1) Lorsque visité pour la première fois, se souvenir de l'expéditeur. Envoyer le jeton à un des voisins qui n'as pas encore été visité et attendre sa réponse.

2) Lorsque le voisin reçoit le jeton,
   -S'il a déjà été visité, il retournera le jeton avec le message "back edge"
   -Si non, il envera le jeton (séquentiellement) à tous ses voisins qui n'ont pas encore été visités.

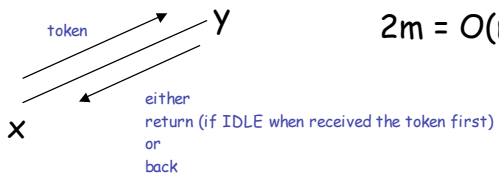
3) Si tous les voisins ont étés visités, retourner le jeton à l'expéditeur ("reply").

4) Lors de la réception d'un "reply", envoyer le jeton à un autre voisin qui n'a pas été visité

Complexité

Complexité en Message:

Types de message: `token, back, return`



2m = O(m)

Complexité temporelle: 2m = O(m)
(temps idéal) Complètement séquentiel

Ω(m) est aussi la limite inférieure

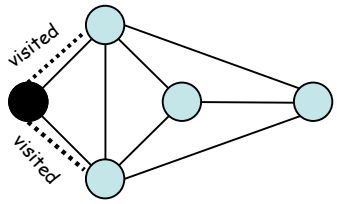
Note: La majorité des messages sont sur des liens « Back Edges»

---> La majorité du temps est perdue dû aux liens « Back Edges»

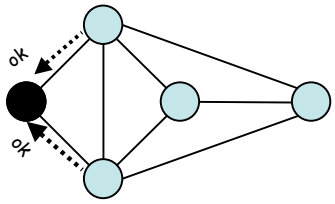
Idée: éviter d'envoyer des messages sur les liens « back edges »

Comment ?

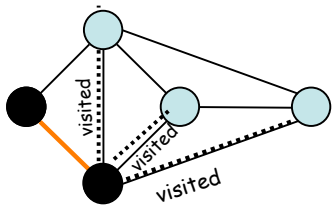
DF+ Amélioration du temps d'exécution



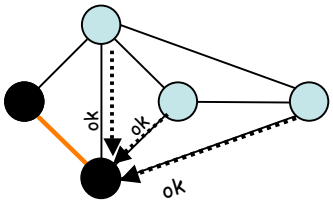
DF+ Amélioration du temps d'exécution



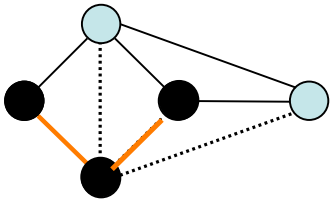
DF+ Amélioration du temps d'exécution



DF+ Amélioration du temps d'exécution



DF+ Amélioration du temps d'exécution



DF+ Complexité — Message

Messages: **`Token`**, **`Return`**, **`Visited`**, **`Ack (ok)`**

Chaque entité (sauf initiateur):

- Reçoit 1 **`Token`**
- Envoie 1 **`Return`**: 2(n-1)

Chaque entité:

- Envoie 1 **`Visited`** à tous ses voisins sauf l'expéditeur

Soit s l'initiateur,

$$|N(s)| + \sum_{x \neq s} (|N(x)|-1)$$

$$= 2m - (n-1)$$

Même calcul pour **`Ack`**)

TOT: 4m

DF+ Complexité — Temporelle (temps idéal)

Token et Return sont envoyés séquentiellement: 2(n-1)

Visited et Ack sont envoyés en parallèle: 2n

TOT: 4n -2

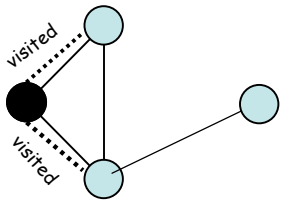
Résumé:

Parcours en profondeur (DF search)

	Messages	Temps idéal
DF:	2m	2m
DF+:	4m	4n -2

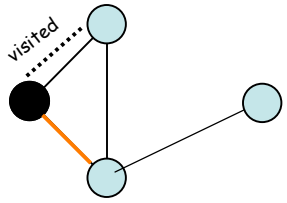
DF++

Qu'arrive-t-il si nous n'envoyons pas les ACKs?



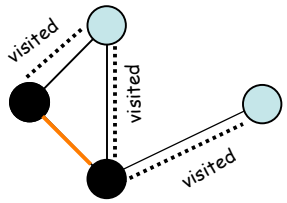
DF++

Qu'arrive-t-il si nous n'envoyons pas les ACKs?



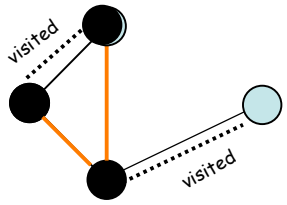
DF++

Qu'arrive-t-il si nous n'envoyons pas les ACKs?



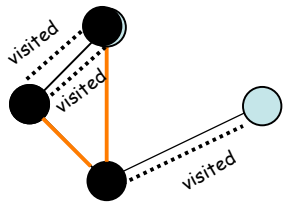
DF++

Qu'arrive-t-il si nous n'envoyons pas les ACKs?



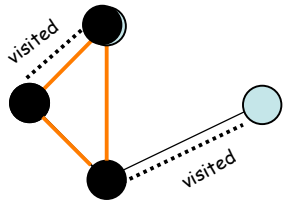
DF++

Qu'arrive-t-il si nous n'envoyons pas les ACKs?



DF++

Qu'arrive-t-il si nous n'envoyons pas les ACKs?

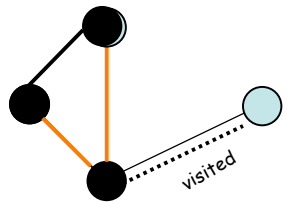


Un jeton est envoyé à un noeud déjà visité (= back edge)

Éventuellement, les deux noeuds seront conscient de "l'erreur"

DF++

Qu'arrive-t-il si nous n'envoyons pas les ACKs?



Un jeton est envoyé à un noeud déjà visité (= back edge)

Éventuellement, les deux noeuds seront conscient de "l'erreur"

Et continueront comme si l'erreur n'avait pas eu lieue

DF++ Complexité

Dans le pire cas, il y a une "erreur" sur chaque lien à l'exception des liens appartenant à l'arbre couvrant.

Messages = 4m -(n-1)

Mais lorque nous mesurons le temps idéal:

Il y aura aucune "erreur"

Temps = 2(n-1)

Résumé

	Messages	Temps idéal
DF:	2m	2m
DF+:	4m	4n -2
DF++	4m-n+1	2n-1

Observations

Temps ...

Terminaison ...

Utilisation:

Problèmes de permissions d'accès, e.g., Exclusion Mutuelle

Tout parcours d'un graph effectue aussi une diffusion(pas efficacement). L'inverse n'est pas vrai.

Un Autre Parcours: Smart Traversal

1- construire un arbre couvrant avec SHOUT+

Messages = 2m

2- Effectuer un parcours DF

Messages = 2(n-1)

Total des Messages = 2(m+n-1)

Un Autre Parcours: Smart Traversal

1- construire un arbre couvrant avec SHOUT+

Temps ≤ d+1 d: diamètre

2- Effectuer un parcours DF

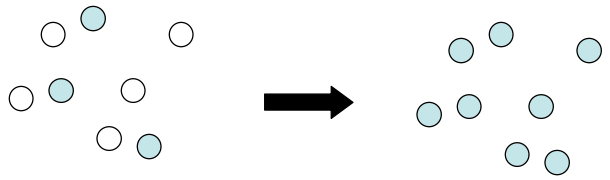
Temps = 2(n-1)

Temp total ≤ 2n+d-1

Résumé

	Messages	Temp Idéal
DF:	2m	2m
DF+:	4m	4n -2
DF++	4m-n+1	2n-1
Smart	2m+2n-2	2n+d-1

Opérations avec initiateurs multiples: WAKE-UP



FLOOD résous le problème.

Algorithme générique FLOOD : $O(m)$

Plus Précisément: $2m - n + k^*$ (↘ n. d' initiateurs)

POURQUOI ?

1 initiateur = broadcast = $2m - n + 1$ Tous des initiateurs = $2m$

Opérations avec initiateurs multiples: WAKE-UP

Dans des topologies particulières ?

Arbre

Flood est optimal $n + k^* - 2$

Opérations avec initiateurs multiples: WAKE-UP

COMPLETE GRAPH

Broadcast

Flood $O(n^2)$ → Specific $O(n)$

Wakeup

Flood Specific

$\Omega(n^2)$

Certaines suppositions sont nécessaires pour réduire la complexité

HYPERCUBE

Broadcast

Flood $O(n \log n)$ → Specific $O(n)$

Wakeup

Flood Specific

$\Omega(n \log n)$