Algorithmes Parallèles et Distribués

3. Algorithmes distribués (suite)

Franck Butelle

LIPN, Université Paris 13
Formation Ingénieurs SupGalilée Info 3

14/12/2022

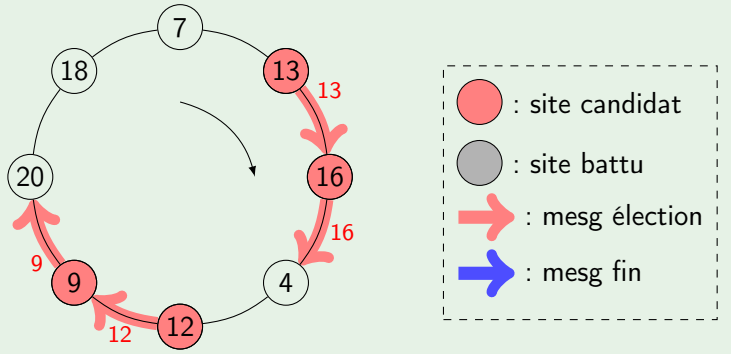
Plan

Plan

Algorithme de Chang-Roberts 1979

Hypothèse : G : anneau unidirectionnel.
Objectif : élire parmi tous les sites, l'id. max.
Idée : Les messages venant des sites d'id. inf. sont arrêtés.

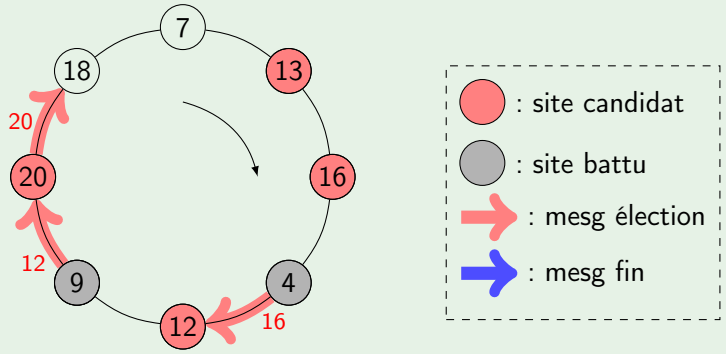
Exemple ($I = \{16,4,12,9,20,18,7,13\}$. Init=$\{9,12,13,16\}$)



Algorithme de Chang-Roberts 1979

Hypothèse : G : anneau unidirectionnel.
Objectif : élire parmi tous les sites, l'id. max.
Idée : Les messages venant des sites d'id. inf. sont arrêtés.

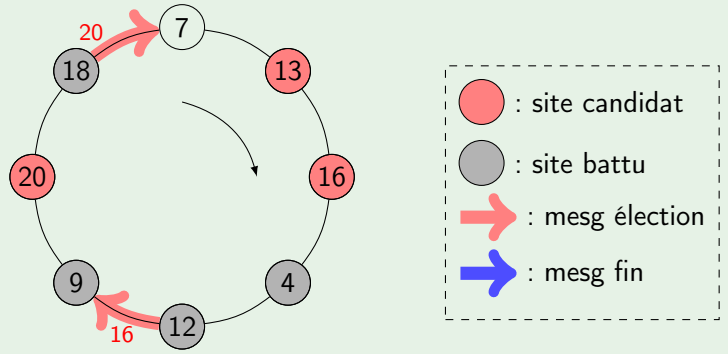
Exemple ($I = \{16,4,12,9,20,18,7,13\}$. Init=$\{9,12,13,16\}$)



Algorithme de Chang-Roberts 1979

Hypothèse : G : anneau unidirectionnel.
Objectif : élire parmi tous les sites, l'id. max.
Idée : Les messages venant des sites d'id. inf. sont arrêtés.

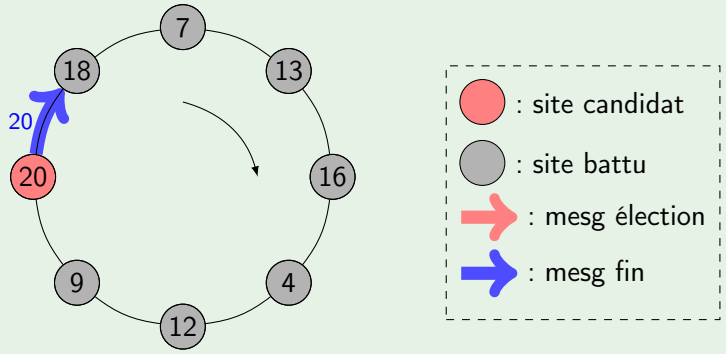
Exemple ($I = \{16,4,12,9,20,18,7,13\}$. Init=$\{9,12,13,16\}$)



Algorithme de Chang-Roberts 1979

Hypothèse : G : anneau unidirectionnel.
Objectif : élire parmi tous les sites, l'id. max.
Idée : Les messages venant des sites d'id. inf. sont arrêtés.

Exemple ($I = \{16,4,12,9,20,18,7,13\}$. Init=$\{9,12,13,16\}$)



Algorithme de Chang-Roberts 1979

```
var état ∈ {passif, candidat, élu, battu}, état←passif.
Eveil : si est_initiateur alors
            état←candidat; envoyer <élection,Myid> au succ.
        finsi

sur_reception de <élection, id> :
        si Myid < id alors
            état←battu; envoyer <élection,id> au succ.
        sinon_si Myid > id alors
            si état≠candidat alors
                état←candidat; /* si pas déjà candidat devient candidat */
                envoyer <élection,Myid> au succ.
            finsi
        sinon   /* le message a fait le tour de l'anneau */
            état← élu; envoyer <fin,Myid> au succ.
        finsi

sur_reception de <fin, id> :
        si Myid ≠id alors
            id_élu ←id; état←battu;
            envoyer <fin,id> au succ.; terminaison
        sinon
            terminaison
        finsi
```

Analyse de l'algo. de Chang-Roberts 1979

Théorème

L'algo CR79 résout le pb de l'élection sur un anneau unidirectionnel (a)synchrone.
Sa complexité en messages dans le pire des cas est en $\Theta(n^2)$.
Sa complexité en moyenne en messages est en $\Theta(n\log(n))$.
Sa complexité temporelle dans le pire des cas est en $\Theta(n)$.

Remarques

L'algo a été défini initialement pour un réseau synchrone, mais se comporte bien sur un réseau asynchrone.

Exercice 1

Ecrire la variante : élection uniquement parmi les initiateurs. Complexités ?

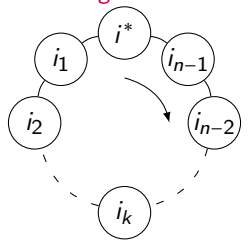
Preuve des complexités dans le pire des cas

Soit i^* le site d'identité max.

- Complexité temporelle :
  pire des cas atteint avec *un seul initiateur*, successeur immédiat de i^*...
- Complexité en nbre de messages :
  pire des cas atteint pour un anneau avec identités en ordre décroissant et *tous initiateurs*. Première «phase» : n messages...

Complexité en moyenne en nbre de mesg :

Soit une configuration aléatoire :



- On suppose que tous sont initiateurs, soit i^* id. max. et i_k le site à distance k de i^* (en partant de i_k).
- Soit X_k variable aléatoire représentant le nbre de mesg $< election, i_k >$ d'origine i_k. $P(X_k \geq k+1) = 0$. Notons que $P(X_{i^*} = n) = 1$.
- Pour que le mesg soit relayé *au moins* $t \leq k$ fois, il faut et il suffit que $i_k = \max\{i_k, ..., i_{k-t+1}\}$. Si toutes les conf. sont équiprobables alors $P(X_k \geq t) = 1/t$.

Complexité en moyenne en nbre de mesg (suite)

$C_{moy}(n) = \sum_{k=1}^{n-1} E(X_k) + 2n$ (il y a n mesg $<fin, i^*>$ et n mesg $<élection, i^*>$).

sachant que l'Espérance : $E(X) \stackrel{def}{=} \sum_{t=1}^{+\infty} t \times P(X = t)$

$$E(X) = \sum_{t=1}^{+\infty}\sum_{s=1}^{t} P(X=t) = \sum_{s=1}^{+\infty}\sum_{t=s}^{+\infty} P(X=t) = \sum_{s=1}^{+\infty} P(X \geq s)$$

or $P(X_k \geq t) = \frac{1}{t}$ et $P(X_k \geq k+1) = 0$, donc $E(X_k) = \sum_{t=1}^{k} \frac{1}{t}$.

donc $C_{moy}(n) = \sum_{k=1}^{n-1}\sum_{t=1}^{k} \frac{1}{t} + 2n = \sum_{t=1}^{n-1}\sum_{k=t}^{n-1} \frac{1}{t} + 2n = \sum_{t=1}^{n-1} \frac{n-t}{t} + 2n = n\sum_{t=1}^{n-1} \frac{1}{t} + n + 1$

$C_{moy}(n) = nH_n + n$ (H_n est la n-ieme somme partielle harmonique).

Complexité en moyenne en nbre de mesg (fin)

Calcul de H_n

$\frac{1}{t}$ décroissante donc $\int_t^{t+1} \frac{1}{x} dx \leq \frac{1}{t} \leq \int_{t-1}^{t} \frac{1}{x} dx$ pour $t \geq 2$
donc en sommant pour $t = 2$ à n (et en ajoutant 1) :

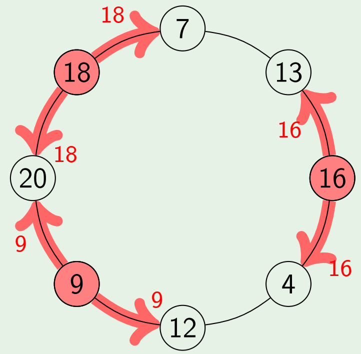
$$1 + \int_2^{n+1} \frac{1}{x} dx \leq H_n \leq 1 + \int_1^{n} \frac{1}{x} dx$$

donc $\lim_{n \to +\infty} H_n \approx ln(n)$
donc $C_{moy}(n) = \Theta(n \log(n))$

Algo. de Franklin 1982

Hyp. : anneau bidirectionnel
Objectif : Élection parmi les initiateurs.
Idée : Se comparer à chaque «tour» à ses 2 voisins actifs, les plus **petits** survivent, les passifs/battus relaient les mesgs.

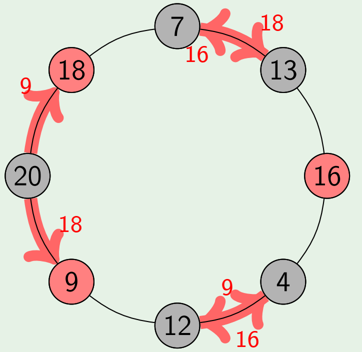
Exemple ($I = \{16, 4, 12, 9, 20, 18, 7, 13\}$. Init=$\{9, 16, 18\}$)



Algo. de Franklin 1982

Hyp. : anneau bidirectionnel
Objectif : Élection parmi les initiateurs.
Idée : Se comparer à chaque «tour» à ses 2 voisins actifs, les plus **petits** survivent, les passifs/battus relaient les mesgs.

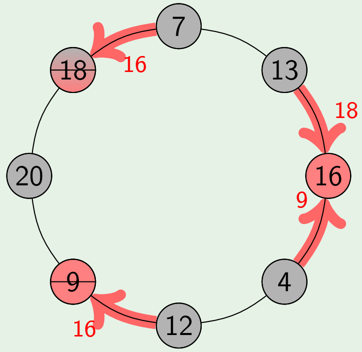
Exemple ($I = \{16, 4, 12, 9, 20, 18, 7, 13\}$. Init=$\{9, 16, 18\}$)



Algo. de Franklin 1982

Hyp. : anneau bidirectionnel
Objectif : Élection parmi les initiateurs.
Idée : Se comparer à chaque «tour» à ses 2 voisins actifs, les plus **petits** survivent, les passifs/battus relaient les mesgs.

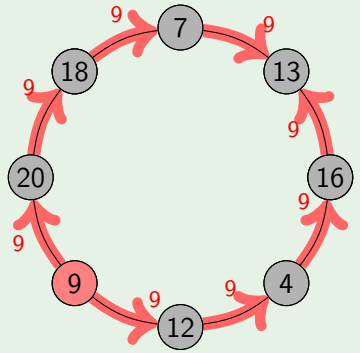
Exemple ($I = \{16, 4, 12, 9, 20, 18, 7, 13\}$. Init=$\{9, 16, 18\}$)



Algo. de Franklin 1982

Hyp. : anneau bidirectionnel
Objectif : Élection parmi les initiateurs.
Idée : Se comparer à chaque «tour» à ses 2 voisins actifs, les plus **petits** survivent, les passifs/battus relaient les mesgs.

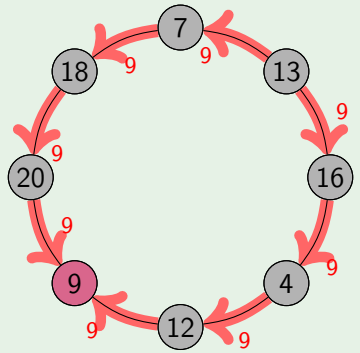
Exemple ($I = \{16,4,12,9,20,18,7,13\}$. Init=$\{9,16,18\}$)



Algo. de Franklin 1982

Hyp. : anneau bidirectionnel
Objectif : Élection parmi les initiateurs.
Idée : Se comparer à chaque «tour» à ses 2 voisins actifs, les plus **petits** survivent, les passifs/battus relaient les mesgs.

Exemple ($I = \{16,4,12,9,20,18,7,13\}$. Init=$\{9,16,18\}$)



Algo. de Franklin 1982

```
var état ∈ {passif,actif,élu,battu};
Eveil : si est_initiateur alors
          état←actif; envoyer <Myid> aux deux voisins;
        finsi

si état = actif alors /* seuls les initiateurs participent à l'élection */
     sur_reception des messages <v> et <w> des DEUX voisins :
          si min{v,w} < Myid alors état←battu; finsi
          si min{v,w} > Myid alors envoyer <Myid> aux deux voisins; finsi
          si min{v,w} = Myid alors
                  état←élu; /* Terminaison ... */
          finsi
sinon
     relayer les messages reçus
finsi
```


Complexité F82

Théorème

L'algorithme F82 termine,
Complexité en nbre de mesg. au pire en $O(n\log(n))$
Complexité en temps au pire en $O(n)$.

Remarques

La complexité en moyenne en nbre de mesg. semble être en $O(n\log(n))$ mais suivant certaines hypothèses simplificatrices (voir Janson, Lavault & Louchard DMTCS 2008).
Algorithme similaire : Hirschberg-Sinclair 1980.

Complexité F82

- Supposons que tous les sites sont initiateurs.
- Le pire des cas est atteint avec une distribution de «pics» maximale : $\lfloor n/2 \rfloor$.
- Au 1er «tour» : $2n$ messages échangés et il ne reste plus que $\lfloor n/2 \rfloor$ actifs.
- Au 2e «tour», *au pire* il reste $\lfloor n/4 \rfloor$ actifs, etc.
- En fin de phase ϕ, il reste $\left\lfloor \frac{n}{2^\phi} \right\rfloor$ candidats donc le nbre de phases est au plus $\lfloor \log_2(n) \rfloor + 1$
- La distance entre deux candidats en phase ϕ est de 2^ϕ, donc 2^ϕ unités de temps.
- Quel que soit le nbre d'actifs, un tour se termine lorsque tous les actifs ont reçu 2 messages : un «tour» comporte $2n$ messages échangés.
- Donc la complexité en message dans le pire des cas : $C_{max}(n) = 2n\log_2(n) + O(n)$
- et la complexité en temps est de $\sum_{\phi=1}^{\lfloor \log_2(n) \rfloor + 1} 2^\phi = O(n)$

Complexité de l'élection en nbre de mesg sur un anneau

Théorème

Sur un anneau d'ordre n, si les ps ne connaissent pas n, les communications sont FIFO et tous sont initiateurs alors tout algorithme d'élection par recherche d'extremum utilise au minimum $nH(n)$ messages.

Élection sur un arbre

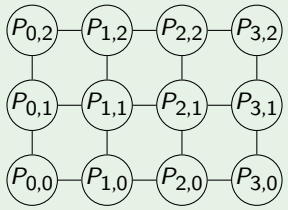
Hyp. : les sites connaissent la topologie mais pas l'ordre du graphe.
Idée : partir des feuilles...

Élection sur grille

Une grille (*Mesh*) de dimension $a \times b = n$, nbre d'arêtes : $m = a(b-1) + b(a-1) = 2ab - a - b$.

Exemple

Grille 4×3 :



Idée d'algo : réveiller tous les sites puis faire une élection sur l'anneau extérieur entre les 4 coins.

Élection sur un graphe qcq : Algorithme Phase

G. Tel 1990

Hyp. : connaissance du diamètre du graphe : D
Idées :

- Rechercher l'id. max parmi ses voisins, puis parmi les voisins de ses voisins etc.
- Notion de synchronisation logique par phase
- Exactement D messages envoyés à chaque voisin
- Terminaison si on est sûr d'avoir vu des sites à distance $\geq D$

Résultat : optimal en temps.


```
var NbRecus[x ∈ Voisins]←0 : entiers /* nb de mesgs recus de x*/
    NbDiffusions ←0 : entier
    maxid ←Myid : entier

procedure diffusion() :
        envoyer <élection,maxid> aux Voisins; NbDiffusions ←NbDiffusions + 1;

procedure fin() : ...

Eveil : si initiateur alors diffusion(); finsi

sur_reception <élection,id> de x :
        maxid ←max{maxid,id}; NbRecus[x] ←NbRecus[x]+1;
        si min_{q∈Voisins}{NbRecus[q]} ≥ NbDiffusions et NbDiffusions< D alors
                diffusion();
        finsi
        si min_{q∈Voisins}{NbRecus[q]} ≥ D alors fin(); finsi
```


Complexité de l'algo Phase

Théorème

Complexité en messages : $2mD + O(m)$
complexité en temps : $\Theta(D)$ *: optimal.*

Nbre max de phases : 1 seul initiateur, id max à dist D de cet initiateur : $2D$ pour l'élection avec terminaison implicite. $3D$ pour la terminaison par processus (diffusion de «fin»).
Chaque site a envoyé D messages à chaque voisin, donc $2mD$ messages échangés sans la terminaison qui prend au plus $2m$ messages.

Théorème

Tout algo d'élection dans un graphe quelconque utilise $\Omega(m + n \log n)$ *messages.*

Pour les anneaux la borne inférieure est de $\Omega(n \log n)$.
Toute arête du graphe doit être visitée au moins une fois : $\Omega(m)$.
Ce résultat est valide aussi pour la construction d'Arbre Couvrant et d'AC de poids min.

Plan

Définitions et propriétés sur les Arbres Couvrants

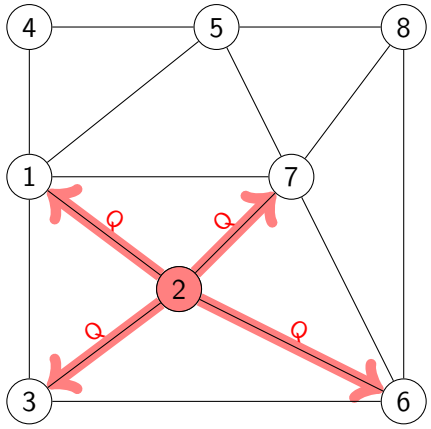
- Chaque site doit choisir un sous-ens. de voisins : $Voisins_{AC} \subseteq Voisins$
- L'ens. de tous les liens correspondants forme un AC T de G : $E(T) \subseteq E(G)$ et $V(T) = V(G)$
- Un AC de G contient donc n *nœuds* et $n-1$ *arêtes*.
- Si on ôte des arêtes à un AC on obtient une *forêt couvrante*
- Une fois l'AC construit, on peut profiter de tous les algos conçus pour fonctionner sur des réseaux en arbre
- Construction distribuée d'un AC
  - 1 initiateur : facile, plusieurs : moins facile

Un seul initiateur : Algo AC1

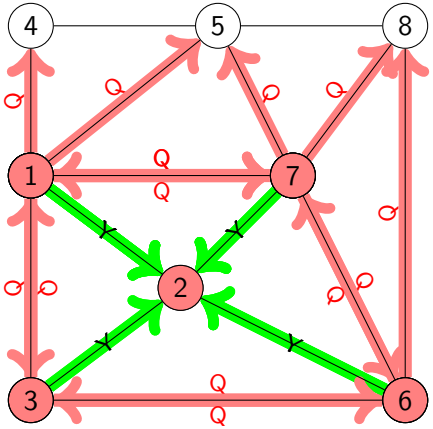
Méthode (inondation avec réponses) :

1. L'initiateur demande «Q : es-tu mon voisin dans l'AC ?» à tous ses voisins
2. un site non initiateur répond «Y : oui» qu'à sa première visite et alors demande aussi à tous ses autres voisins «Q : es-tu mon voisin dans l'AC ?».
   Si ce n'est pas la première visite alors «N : non». L'initiateur réponds toujours «N : non».
3. Un site termine quand il a reçu une réponse de tous ses voisins à qui il a posé la question.

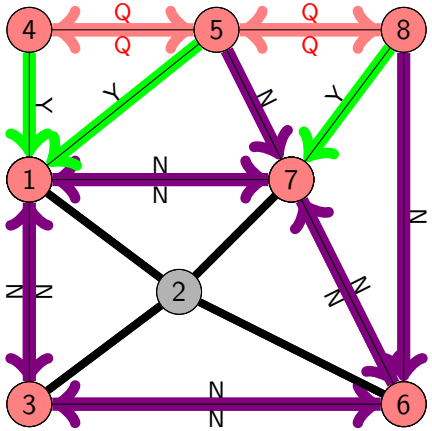
Exemple d'exécution AC1



Exemple d'exécution AC1



Exemple d'exécution AC1




```
/* Algo AC1 : On suppose connaissance des Voisins */
Eveil :
    marque←Faux; cpt←0; parent← −1;
    si initiateur alors
        envoyer<Q> à tous x ∈ Voisins
        marque←Vrai; cpt← |Voisins|;
    finsi
sur_reception de <Q> de y :
    si marque=Faux alors /* première visite */
        marque←Vrai;
        envoyer<Q> à x ∈ Voisins − {y}
        envoyer<Y> à y ;
        Cpt← |Voisins| −1 ; parent←y
    sinon
        envoyer<N> à y
    finsi
sur_reception de <Y> de y :
    cpt−− ; si cpt=0 alors terminaison
sur_reception de <N> de y :
    cpt−− ; si cpt=0 alors terminaison
```


Complexités de AC1

1 seul initiateur.

- en messages : $4(m-(n-1))+2(n-1)=4m-2n+2$
- en temps : $D+1$.
- optimal en temps et en messages mais améliorable vis-à-vis des constantes
- amélioration : supprimer les messages «Non» : finalement seulement $2m$ messages.

Conclusions sur AC1

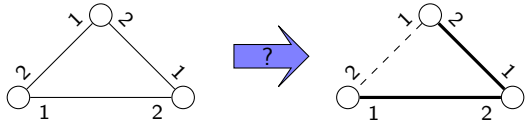
Hyp. 1 seul initiateur

- Tout Algo de diffusion peut être facilement transformé en algo de création d'AC
  - En fait tout algo de diff. assure que l'info est reçue par tous, soit le parent celui par lequel un site reçoit l'info en premier...
- Tout Algo d'AC fournit une structure permettant la diffusion.
- Donc les pbs AC et Diffusion sont équivalents en distribué avec 1 seul initiateur
- Tout Algo global (qui utilise tous les sites) peut résoudre la diffusion donc peut construire un AC

AC avec plusieurs initiateurs

Théorème (d'impossibilité déterministe)

Il n'existe pas d'algo. distribué déterministe de construction d'AC dans un réseau sans identités et avec plusieurs initiateurs.



Démonstration.

Tous les sites exécutent le même algo qui déclenchera l'envoi/réception des mêmes messages et donc les mêmes changements d'état... □

Construction d'AC avec plusieurs initiateurs

Hyp. Identités toutes distinctes, graphe quelconque, plusieurs initiateurs.

- *Difficultés* : risque de création de cycle, risque de création d'une forêt couvrante et non d'un AC
- *Idée 1* : faire tourner en parallèle autant de AC1 que d'initiateurs mais en les marquant de l'identité de l'initiateur : on construit alors k AC indép.
  - $O(2mk)$ messages si k initiateurs donc au pire $O(n^3)$.
  - Amélioration : créer 1 seul AC en utilisant l'ordre sur les id. des sites pour arrêter ceux porteurs de plus grande identité. complexité toujours en $O(n^3)$ messages *si on ne réutilise pas les fragments déjà construits...*

Algo ACx

- Idée : faire croître *simultanément* des arbres fragments de l'AC final.
- *Chaque fragment a une identité* stockée sur chaque site dans la variable *id_frag*.
- Un site maintient un tableau de *l'état de ses voisins : candidat*, *actif*, *inactif*
- Suivant l'état de ses voisins, un site est dans l'état *ouvert*, *libre*, *fermé*, *terminé*.
- Chaque fragment dispose d'un *nœud privilégié* : sa racine.
- La racine d'un fragment est déplacée pour initialiser les tentatives de rattachement de son fragment à un autre.
- En plus de la création de l'AC on obtient une élection : la racine de l'AC.
- L'algo est conçu pour être non déterministe sur le choix de l'élu.

Algo ACx : messages échangés

Algo de x pour un message en provenance de y

- *<conn,idfx>* : tentative de connexion à un voisin. Les réponses possibles sont :
  - *<cousin>* : si $idfy = id\_frag$ (ils appartiennent au même fragment), l'arête ne doit plus être visitée
  - *<nok>* : si $idfy > id\_frag$: la fusion doit se faire dans l'autre sens
  - *<ok,idfy>* : si $idfy < id\_frag$: le fragment de y est «absorbé» : l'identité de ce fragment est changé par un message *<nrac,idf>*, éventuellement il faut mettre à jour les variables d'état du père par *<maj>*.
- *<nrac,idf>* : mise à jour de l'id. de frag, à propager de père en fils.
- *<jeton,idf>* : déplacement de la racine vers un fils lorsque tous les voisins ont été testés, le sens fils-père doit être inversé.
- *<maj,idf>* : mise à jour de l'état d'un site, à propager vers le père au besoin.
- *<fin>* : terminaison explicite

Algo ACx : variables d'état d'un site / *état* d'un voisin

- état= *ouvert* ssi $\exists$ un voisin y tel que $état[y] = candidat$
- état= *libre* ssi $\nexists y$ tel que $état[y] = candidat$ et $\exists z$ tel que $état[z] = actif$.
- état= *fermé* ssi $\nexists y$ tel que $état[y] = actif$
- état= *terminé* ssi état= *fermé* et racine ou réception de *<fin>*

Algo ACx détaillé

```
Eveil père← nil;fils=∅;att←faux; id_frag← Myid; état←ouvert; ∀y ∈ Voisins état[y]←candidat;

fini←faux;
tant_que non fini faire
   si père= nil et att=faux et état=fermé alors fin();
   sinon si père= nil et att=faux et état=ouvert alors
         choix(); /* Choix d'un candidat pour connexion ou déplacement jeton */
   sinon {
         sur_reception <fin,id> : envoyer <fin,id> à tous les fils ; fini←vrai;
         sur_reception <conn,id> : réception_conn(y,id);
         sur_reception <ok,id> :
            père←y; att←faux; id_frag←id; envoyer <nrac,id> à tous les fils; MAJ(y,inactif);
         sur_reception <nok,id> : att←faux; MAJ(y,actif);
         sur_reception <cousin,id> : att←faux; MAJ(y, inactif);
         sur_reception <nrac,id> : id_frag←id; envoyer <nrac,id> à tous les fils;
         sur_reception <maj,etat_mesg> : MAJ(y, etat_mesg);
         sur_reception <jeton,etat_mesg> : père←nil; fils←fils∪{y}; MAJ(y, etat_mesg);
   }
fintq
```


Procédures de ACx

```
procedure réception_conn(y,id) :
        si id < id_frag alors envoyer <nok,id_frag> à y; MAJ(y, candidat); finsi
        si id = id_frag alors envoyer <cousin,id_frag> à y; MAJ(y, inactif); finsi
        si id > id_frag alors envoyer <ok, id_frag> à y; fils←fils∪{y}; MAJ(y, candidat); finsi

procedure MAJ(y,etat_mesg) :
        état[y]← etat_mesg;
        /* recalcul de l'état du site: en cas de chgt d'état faire suivre */
        si ∃y ∈ voisins état[y]←candidat alors nouv_état← ouvert;
        sinon si ∃y ∈ voisins état[y]←actif alors nouv_état← libre;
        sinon nouv_état←fermé.
        finsi
        si état≠ nouv_état alors
          état←nouv_état; envoyer <maj, état> à père s'il existe.
        finsi
```

Complexités de ACx

- en messages : à cause des *<maj>* et *<nrac>* : $O(n^2)$
  - on peut construire un exemple ou les fusions sont faites avec des tailles de frag. croissant les *<nrac>* sont alors en $1+2+3+4+...+(n-1)=\frac{n(n-1)}{2}$. Sur un graphe complet on peut aussi avoir les *<nok>* en $O(n^2)$.
- en temps : en $O(n^2)$
  - avec le même exemple de fusions de frag de taille croissante.

Réduire le nbre de messages d'ACx

- pb : il faut *réduire les maj au minimum*. Pour se faire il faut une croissance a peu près *équilibrée* des fragments : introduction de phase logique et peut introduire des attentes mais tant pis
  - historiquement le premier du genre est un algo de Gallager, Humblet et Spira en 1983 qui construit en fait un AC de poids min.
- L'optimum est en $\Omega(m+n\log n)$ en messages et $\Omega(D)$ en temps. Il existe des algos atteignant l'optimum pour l'un des deux mais très difficile pour les 2 en même temps.
  - Par exemple pour un algo optimal en nbre de messages voir algo GaHS83, et pour le temps voir phase (élection) puis AC1 et Awerbuch 1987 pour un algo optimal en nbre de mesg et $O(n)$ en temps.
  - Il a par ailleurs été prouvé que tout algorithme d'élection en $O(D)$ en temps, dans un graphe quelconque, requiert $\Omega(mlogD)$ messages.

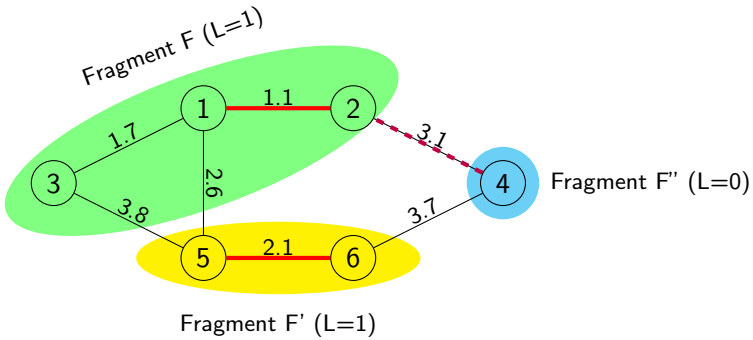
AC de poids minimal

- premiers algos (séquentiels) : Borůvka (1926), l'algorithme de Prim et l'algorithme de Kruskal.
- Propriétés si les arêtes sont toutes de poids diff. :
  - alors l'ACPM est unique.
  - localité : pour un nœud x quelconque, son arête adjacente de poids min appartient à l'ACPM.
  - étant donné un fragment F de l'ACPM, soit e l'arête sortante de poids min. L'ajout de e (et de son nœud adjacent) à F forme un autre frag. F' de l'ACPM.

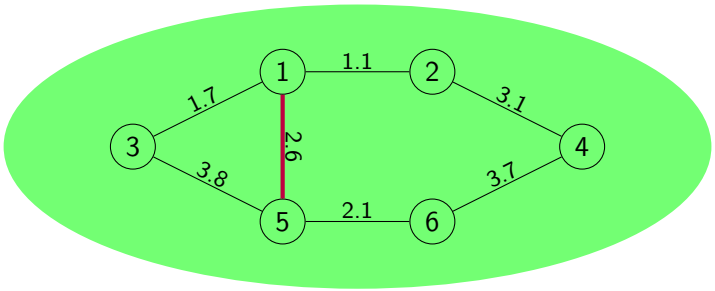
GaHS83

- *Hyp. :* graphe quelconque, poids des arêtes connues par les extrémités des arêtes et poids des arêtes *toutes diff.*,
- *Méthode :*
  - Fusion et absorption de fragments,
  - pas de racine de fragment mais une arête cœur avec orientation fils/père vers cette arête.
  - Décisions de fusion et terminaison prises par les extrémités de l'arête cœur.
  - Un frag. qui ne contient qu'un nœud est de niveau 0.
  - L'identité d'un frag. est donné par le poids de son cœur.
- *Règles de croissance des fragments* chaque frag. F recherche son arête sortante de poids min. Le frag. tente une fusion par cette arête $e=(i,j)$. Soit L le niveau de F et L' le niveau de F'.
  - Si $L<L'$ alors F est *absorbé* par F'. Le niveau de F' ne change pas.
  - Si $L=L'$ et F et F' ont la même arête sortante de poids min. alors *fusion*, e devient le nouveau cœur et le niveau est $L+1$.
  - Sinon F doit attendre que F' grossisse.

Exemple d'exécution de GaHS83



Exemple d'exécution de GaHS83



Complexités de GaHS83

- nbre de messages : $2m + \frac{5}{2} n \log n + O(n)$
- temps : $O(n^2)$

Plan

Rappels définitions terminaison

Rappels :

Définition

Un algo. distribué se *termine par processus (explicitement)* lorsque pour toute exéc. tous les sites de S atteignent un état stable quiescent : terminaison classique avec détection de terminaison. A partir d'une certaine date, tous les sites savent que l'algo est terminé ou en phase de terminaison.

Définition

La *terminaison par message (implicite)* : il n'y a plus de message en circulation. L'algo. est dit alors *sans détection de terminaison*.

Précisions sur la terminaison

Terminaison explicite :

- Tous les sites s'arrêtent sur réception d'un certain message
- Tous les canaux de communication sont vides
- Tous les sites atteignent l'état final dans lequel il n'y a aucun risque de recevoir un nouveau message.

Transformation terminaison implicite en explicite :

- Détection de la terminaison
  - Propriétés nécessaires d'un tel algo :
    - ⋆ Non-interférence : pas d'influence sur l'algo qu'il surveille
    - ⋆ Vivacité : si term. effective alors appel de terminaison en temps fini
    - ⋆ Correction : si l'algo est appelé alors termine.
    - ⋆ Début d'exécution algo term. avant sinon canaux FIFO obligatoirement.
  - 1 initiateur : par calcul du graphe d'exécution (Dijkstra & Sholten 1980)
  - par vagues ou par inondation
- Diffusion d'un message de terminaison

Algo de diffusion du message de terminaison

Hyp. nbre qcq d'initiateurs, graphe qcq, pas besoin d'identités distinctes.
A un instant un site appelle annonceTerm()...

```
var envoiStop←faux; nbStop←0;

procedure annonceTerm():
        si envoiStop=faux alors
                envoiStop←vrai;
                envoyer à tous les Voisins <stop>
        finsi

sur_reception de <stop>:
        nbStop++; annonceTerm();
        si nbStop= |Voisins| alors terminaison finsi
```

Complexité : $2m$ messages échangés en D unités de temps (inondation).

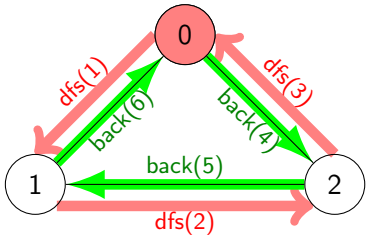
Détecter la terminaison par graphe d'exécution

- *Hyp.* 1 initiateur : P_0
- Gestion dynamique de l'arbre d'exécution *virtuel* (distribué) $T = (V_T, E_T)$ tel que :
  - Soit T est vide, soit T est orienté avec racine P_0
  - V_T inclut sites actifs et messages en transit de l'algo de l'utilisateur
- *Méthode*
  - Les sites maintiennent la variable *père* pour leur père dans T
  - Les sites maintiennent un compteur de fils dans T : nbfils.
  - Si i envoie le message <m> alors i est virtuellement père de <m> dans T (juste nbfils++).
  - Si $j \notin T$ reçoit un message de i alors i devient père de j dans T (idem).
  - Lorsque un message arrive à destination, il est ôté de T (voir <sig>).
  - Un site i qui devient passif et qui n'a plus de fils est ôté de T
  - Si T est vide pour P_0 alors P_0 peut lancer l'algo de terminaison

Algo Dijkstra Sholten 1980

```
Eveil: état←passif; nbfils←0; père← undef;
    si Myid = P0 alors état←passif; père← P0; finsi

A chaque envoi de message <m> de l'algo contrôlé faire
        nbfils++;

sur_reception de <m> de x: /* message de l'algo contrôlé */
        état←actif; si père=undef alors père←x sinon envoyer <sig> à x; finsi
sur_reception de <sig> de x:
        nbfils−−; si nbfils=0 et état=passif alors seDesactive(); finsi

si le site devient passif dans l'algo de base alors
        état←passif; si nbfils==0 alors seDesactive(); finsi
finsi

procedure seDesactive():
        si père=Myid alors annonceTerm() /* Voir diapos précédentes */
                sinon envoyer <sig> à père
        finsi
        père←undef; /* le site sort de l'arbre d'exécution */
```


Exemple d'exécution de DFS1 avec surveillance



Complexité de Dijkstra Sholten 1980

- Complexité : autant de mesg *<sig>* que de mesg. *<m>* (sorte d'accusés de réception), conservation complexité temporelle + hauteur max. de l'arbre T (sans compter annonceTerm).
- Généralisé pour prendre en compte plus d'un initiateur par Shavit & Francez en 1986
- Finalement préférable d'organiser la détection en utilisant un arbre couvrant

Plan

Conclusion de cette partie

Des algorithmes liés

- Si on a un algo d'élection alors on a un algo d'AC (en ajoutant $2m$ messages et D unités de temps)
- AC $\longrightarrow$ élection (en ajoutant $n-1$ messages et h unités de temps, où h est la hauteur de l'arbre)
- Recherche d'extremum $\longrightarrow$ élection
- AC $\longrightarrow$ rech. d'extremum (en ajoutant $n-1$ messages et h unités de temps)
- Recherche d'extremum $\longrightarrow$ terminaison (1= travail, 0=terminé, et toutes les arêtes visitées)
- AC et AC de poids min ont même complexité.

Plan

Exclusion mutuelle

- pb : accéder à des ressources partagées
- Une solution simple : définir des sections critiques en exclusion mutuelle
  - A tout instant au plus 1 ps en section critique (sûreté)
  - Tout ps qui demande à entrer en section critique finit par y arriver au bout d'un temps fini (vivacité)
  - en distribué, solutions souvent basées sur *Horloges logiques*

Quelques algorithmes distribués d'exclusion mutuelle :

- Algorithme de Lamport 1978
- amélioré par Ricart-Agrawala 1981
- amélioré par Carvalho-Roucairol 1983
- Naïmi-Trehel 1987 (suppose création préalable d'arbre couvrant)

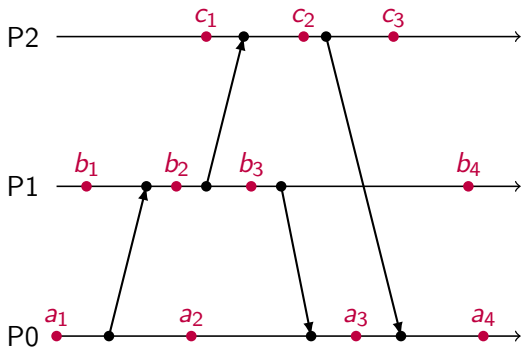
Dépendance causale

La relation «happened-before» (Lamport 78) est un ordre partiel des événements dans un système distribué tel que :

1. Si a et b sont des événements du même ps et a a été exécuté avant b alors $a \rightarrow b$
2. Si a est un événement d'envoi et b est l'événement de réception de ce mesg, alors $a \rightarrow b$
3. Si $a \rightarrow b$ et $b \rightarrow c$ alors $a \rightarrow c$

Si 2 événements a et b sont tels que $a \not\rightarrow b$ et $b \not\rightarrow a$ alors a et b sont 2 événements concurrents.

Exemple de dépendances causales



$a_1 \rightarrow b_2 \rightarrow c_2$

$b_3 \rightarrow a_3$

$c_2 \rightarrow a_4$

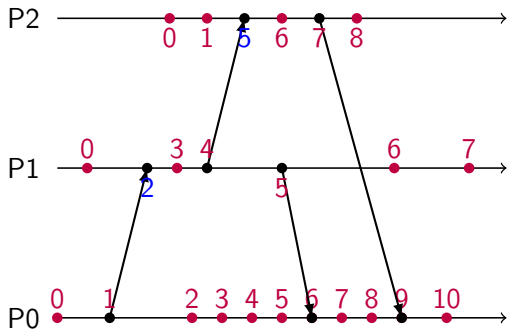
Horloges logiques (Lamport 78)

Objectif : profiter des algos synchrones (plus rapides) sur des systèmes asynchrones. Les horloges physiques dérivent, il faut baser les horloges logiques sur la dépendance causale et non le vrai temps.
Idée : une horloge logique est un compteur à croissance monotone.

- Chaque site gère un compteur H : incrémenté avant chaque événement sur ce site
- Quand un site envoie un mesg, il y ajoute ce compteur.
- sur réception de mesg, $H \leftarrow 1 + \max(H_{recu}, H)$.

Conséquence : si $a \rightarrow b$ (b dépends de a, a *happened-before* b) alors $H(a) < H(b)$ (réciproque fausse)
Avec des entiers sur 64 bits, peu de risques de débordement pour H.

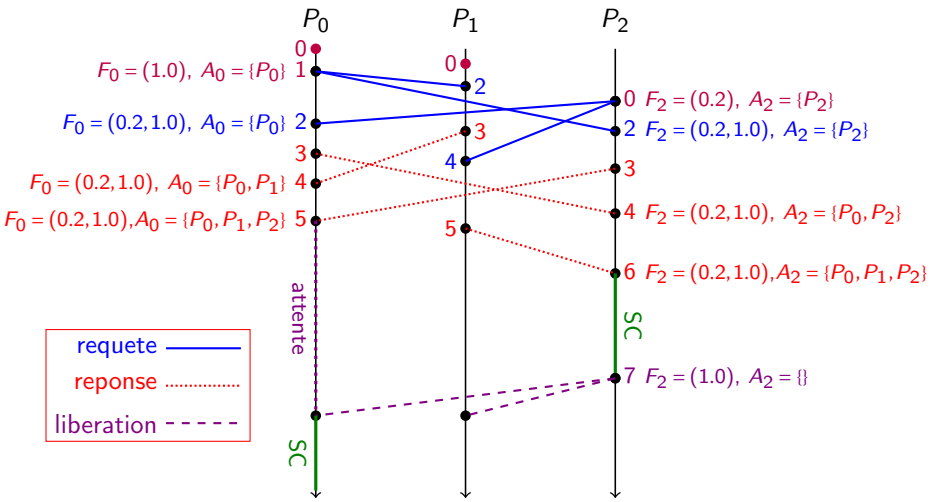
Exemple d'utilisation des horloges logiques



Exclusion mutuelle (algo. L78)

Hyp. : graphe complet, ordre du graphe connu par tous, canaux FIFO.
Idées : chaque ps diffuse à tous les autres sa demande d'entrée en SC (et sa sortie) et maintient une file d'attente des requêtes d'entrées en section critique (y compris la sienne) ordonnée par valeur d'horloge logique croissante.
On passe d'un ordre partiel à un ordre total en effectuant des comparaisons sur le couple (H, id).

Exemple de déroulement de l'algo. d'EM de L78



Détail de l'algo. d'EM de L78

```
/* F: file d'attente, A: sites ayant accusé réception de ma requête.
   On suppose que l'horloge logique H est gérée par une sous–couche logicielle. */
init: F := (), A := ∅

si Myid veut entrer en SC alors diffusion <requete,Myid>; A := ∅ finsi

sur_reception <requete,id>
        ajout_croissant(H.id,F)
        envoyer <reponse,Myid> à id

sur_reception <reponse,id>
        A := A ∪ {id}
        SC()

sur_reception <liberation,id>
        defile(H.id,F)
        SC()

procedure SC:
        si |A| = n−1 et tete_de_file(F)=Myid alors
                entrer en SC...
                diffusion <liberation, Myid>
        finsi
```