

Algorithmique des réseaux et des systèmes distribués

2^{ème} A. ISITcom

Mohamed Mosbah – LaBRI
Institut Polytechnique de Bordeaux
Université de Bordeaux
mosbah@labri.fr
www.labri.fr/~mosbah/Isitcom/

Objectifs du cours

- Connaître les caractéristiques d'un système distribué (SD) à l'aide d'algos
- Comprendre les concepts et les paradigmes fondamentaux d'un SD, au delà (et indépendamment) des technologies
- Etudier certains problèmes fondamentaux (élection, arbre recouvrant, exclusion mutuelle, pannes)
- Pouvoir raisonner dans un environnement distribué. Par exemple concevoir des applications distribuées, les tester, les prouver, les valider et les implanter.

Exemples de questions abordés dans ce cours

- Comment décrire une exécution répartie ?
- Comment déterminer des propriétés globales à partir d'observations locales ?
- Comment coordonner des opérations en l'absence d'horloge commune ?
- Quelles sont les critères de qualité pour une application distribuée ?
- Comment garantir la cohérence (ou la sécurité) d'informations distribuées ?
- Pas sûr que vous aurez toutes les réponses

Faut-il des connaissances préalables

?

- **Prérequis**
 - Connaissance de l'algorithmique élémentaire
 - Connaissance de la programmation (java)
- **Ce qu'on ne va pas voir**
 - Comment fonctionne le tout dernier modèle d'Ipod
 - Comment télécharger le tout dernier film de Spielberg
 - Comment pirater un serveur ?
 -

Organisation et site web

Planning :

- 20 h de cours intégré
- travail individuel

• Site Web :

- <http://www.labri.fr/~mosbah/Isitcom/>
- Supports de cours.
 - quelques exercices

Plan du cours

- Introduction aux systèmes distribués
- Algorithmique élémentaire (rappels), graphes, notion de complexité
- Modèles de l'algorithmique distribuée
- Algorithmes de diffusion,
- arbre recouvrant
- Algorithmes d'Election

Cours 1: Introduction aux systèmes distribués

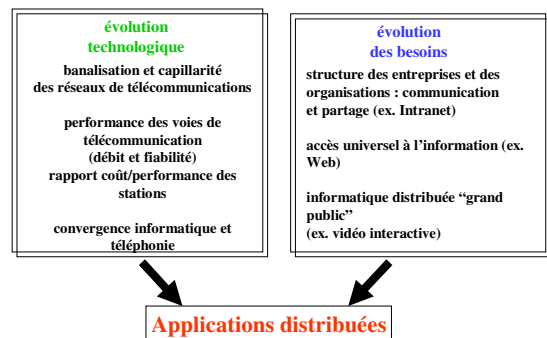
I/ Introduction à l'informatique distribuée

- L'informatique répartie : état de fait de plusieurs applications, et une mutation...
 - Besoin propre des applications
 - Intégration : applications séparées, ressources de calcul, ressources de gestion de données, etc
 - Nouvelles applications: informatique omniprésente
 - Possibilités techniques
 - Interconnexion généralisée : convergence informatique-télécom
 - Performance et coût des machines et des réseaux

Les progrès technologiques

- Avant les années 80, les ordinateurs étaient encombrants et chers (les systèmes centralisés)
- A partir de la mi-80, deux nouveautés:
 - Microprocesseurs (moins chers et très rapide)
 - LANs and WANs
- Les ordinateurs en réseaux non seulement faisables, mais simples.

Technologie + besoins



II/ Définitions d'un SD

Définition [Tanenbaum]: *Un ensemble d'ordinateurs indépendants qui apparaît à un utilisateur comme un système unique et cohérent*

- Les machines sont autonomes
- Les utilisateurs ont l'impression d'utiliser un seul système.

- Définition [Lamport]
 - *A distributed system is one on which I can't do my work some computer has failed that I never heard of.*

Un système réparti est un système qui vous empêche de travailler quand une machine dont vous n'avez jamais entendu parler tombe en panne.

Définition (pour ce cours)

- Un système distribué est un ensemble d'entités autonomes de calcul (ordinateurs, PDA, processeurs, processus, processus léger etc.) interconnectées et qui peuvent communiquer.
- Exemples:
 - réseau physique de machines
 - Un logiciel avec plusieurs processus sur une même machine.

Pourquoi des systèmes répartis ?

- Aspects économiques (rapport prix/performance)
- Adaptation de la structure d'un système à celle des applications (géographique ou fonctionnelle)
- Besoin d'intégration (applications existantes)
- Besoin de communication et de partage d'information
- Réalisation de systèmes à haute disponibilité
- Partage de ressources (programmes, données, services)
- Réalisation de systèmes à grande capacité d'évolution

Exemples:

- WWW
- Contrôle du trafic aérien
- Système de courtage
- Banques
- Super calcul distribué
- Système de fichier distribué
- DNS
- Systèmes Pair-à-pair (P2P)

Quelques domaines d'application des systèmes répartis

- CFAO, Ingénierie simultanée
 - Coopération d'équipes pour la conception d'un produit
 - Production coopérative de documents
 - Partage cohérent d'information
- Gestion intégrée des informations d'une entreprise
 - Intégration de l'existant
- Contrôle et organisation d'activités en temps réel
- Centres de documentation, bibliothèques
 - Recherche, navigation, visualisation multimédia
- Systèmes d'aide à la formation

Propriétés

- Le système doit pouvoir **fonctionner** (même en cas de pannes de certains composants), **et donner un résultat correct**
- Le système doit pouvoir **résister à des attaques contre sa sécurité (confidentialité et intégrité, déni de service, ...)**
- Le système doit pouvoir **s'adapter** à des changements (modification de composants, scalabilité, etc)
- Le système doit préserver ses **performances**

III/ Objectifs d'un système distribué

- Transparence (Masquer la répartition)
 - Uniformité des accès locaux et distants
- La séparation physique entre machines et les différences matériels/logiciels pour les accès sont invisibles par l'utilisateur.
- Localisation des ressources non perceptible
- (nom logique ex: URL <http://www.u-bordeaux.fr/>)
- Migration des ressources possible sans interférence avec la localisation physique
- (ex. transférer un objet uniquement par son nom logique sans modification de ce nom et sans modification de l'environnement d'un utilisateur)

- Réplication de ressources non visible
- Concurrence d'accès aux ressources non perceptible

(ex. accès à un même fichier ou une table dans une base de données: mécanisme de verrou ou de transaction)

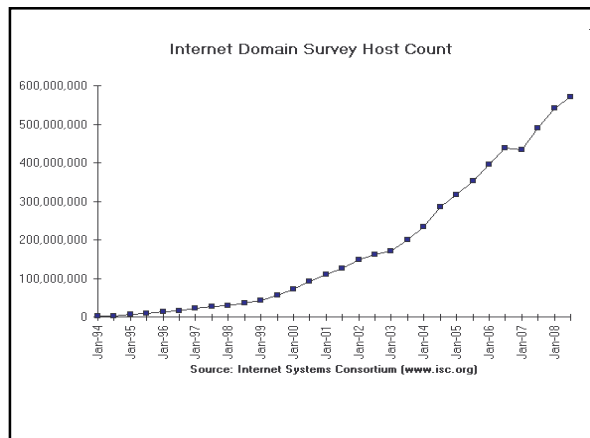
- Invisibilité du parallélisme offert par l'environnement d'exécution
- Tolérance aux pannes permettant à un utilisateur de ne pas s'interrompre (ou même se rendre compte) à cause d'une panne d'une ressource

• Ouverture

- Services offerts selon des règles standards qui décrivent la syntaxe et la sémantique de ces services (Interfaces publiées, ex. IDL)
- Interopérabilité des matériels (de fournisseurs différents)
- Portabilité
- Flexibilité (facilité d'utilisation et de configuration)
- Extensibilité (ajout/MAJ de composants sans en affecter les autres)

• Mise à l'échelle (scalability)

- fonctionne efficacement dans différentes échelles:
 - Deux postes de travail et un serveur de fichiers
 - Réseau local avec plusieurs centaines de postes de travail et serveurs de fichiers
 - Plusieurs réseaux locaux reliés pour former un Internet



• Tolérance aux pannes

- Pannes franches
- Pannes byzantines
- Détection de pannes (difficulté et même impossibilité de détection pour certains systèmes, suspicion de machines) e.g. connexion par un navigateur à un serveur distant qui répond pas !!
- Correction d'erreurs (de fichiers/messages corrompus)
- Reprise sur pannes (techniques de journalisation dans les BD)

(éventuellement système dégradé)

• Sécurité

- Confidentialité (authentification)
- Intégrité (protection contre les falsifications et les corruptions)
- Disponibilité (accès aux ressources)

e.g. commerce électronique, banque en ligne.

IV/ Systèmes distribués vs parallèles

Systèmes Parallèles. Une machine multiprocesseurs avec un environnement du type SIMD (tous les processeurs exécutent le même programme et ont une vision uniforme de l'état global du système).
Extensible à un réseau de machines asynchrones fortement couplées

Systèmes distribués. processus indépendants sur des machines distinctes et communiquant par échange de messages asynchrones (en général, des réseaux faiblement couplés).

Pas de consensus sur ces définitions...

Caractéristiques du parallélisme/distribué

- Objectifs: optimiser les solutions d'un problème (e.g. calcul scientifique, calcul matriciel, tri)
- Calcul de complexité : temps et accès mémoire (pas le temps de communication ou nombre de messages)
- La topologie est généralement fixe (grille, hypercube, grappes)

V/ Intergiciel (Middleware)

- Le *middleware* (intergiciel) est la couche logicielle située entre les couches basses (systèmes d'exploitation, protocoles de communication) et les applications dans un système informatique réparti (CORBA, EJB, COM, etc.).

- Buts:
 - Fournir une interface ou API de haut niveau aux applications
 - Masquer l'hétérogénéité des systèmes matériels et logiciels sous-jacents
 - Rendre la répartition aussi invisible (transparente) que possible
 - Faciliter la programmation répartie (développement, évolution, réutilisation, portabilité des applications)
- <http://www.objectweb.org/>

Couches logicielles et matérielles dans un SD (le middleware)



VI/ Voies d'études des systèmes distribués

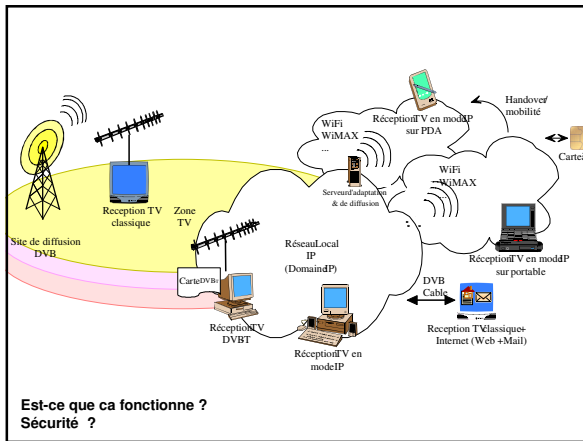
- Approche « descriptive »
 - Etude de modèles de conception d'applications réparties (client-serveur, objets répartis, composants répartis, architecture)
 - Etude de diverses classes de systèmes, intergiciels (middleware) et applications, et de leurs modes d'organisation et de fonctionnement
- Approche « fondamentale » : algorithmes distribués
 - Etude des principes de base: problèmes fondamentaux, solutions connues, limites « intrinsèques »
 - Exemples : Election d'un chef; Structure de diffusion (arbre recouvrant), Exclusion mutuelle, Nommage, Détection de la terminaison

Une application pratique

- Les virus sur nos machines...
- Solution actuelle: protection individuelle (de sa machine)....
- Limites: vulnérabilité, « oubli », mise à jour régulière, solution locale mais non globale
- Alternative: développer des stratégies globale de « destruction de virus »

Difficultés

- Pas de connaissance de l'état global
- Absence de temps universel (ou horloge globale)
- Non déterminisme (lié souvent au problème du synchronisme)
- Et surtout pas de modèle « universel » et standard pour l'algorithmique distribuée



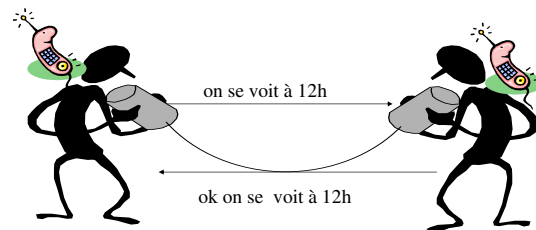
Vers des modèles

- Modéliser
 - pour **représenter** un système en simplifiant divers aspect du réel
 - pour **maîtriser la complexité**
 - pour **observer et comprendre** le comportement du système réel
 - pour **prédire** ou **aider à commander** le comportement d'un système
 - pour **prouver ce comportement** à l'aide de techniques formelles

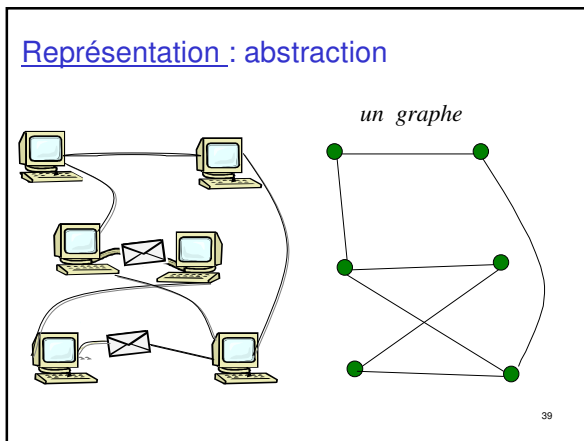
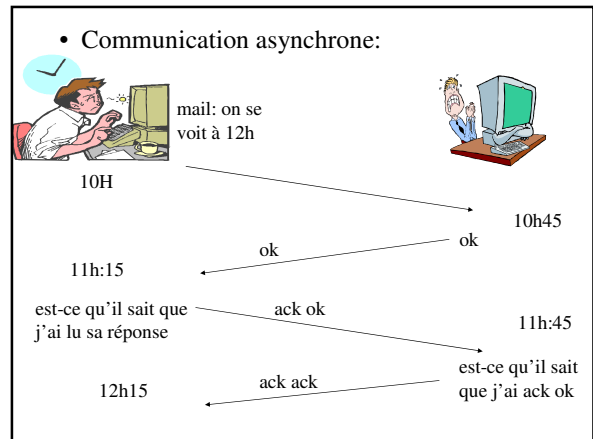
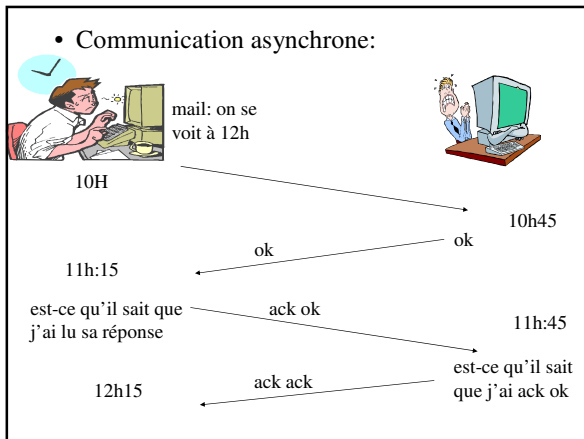
- Nécessité de simplifier et maîtriser la complexité des systèmes et des algorithmes distribués
- Difficulté de l'algorithmique distribuée / centralisé:
 - Pas de connaissance de l'état global
 - Absence de temps universel (ou horloge globale)
 - Non déterminisme (lié souvent au problème du synchronisme)
 - Et surtout pas de modèle « universel » et standard pour l'algorithmique distribuée

Modèles de communication synchrone /asynchrone

Synchrone



Même notion de temps, transmission instantanée, généralement bornée



- A venir : Modélisation d'un système distribué**
- Système distribuée : graphe (non orienté, connexe, simple)
 - sommet : processus
 - arête : canal de communication
 - algorithme distribué local : algorithme qui s'exécute sur chaque sommet (en utilisant uniquement le contexte local)
- 40

- Les réseaux anonymes /avec identités**
- anonymes: pas d'identités (numéros distincts. e.g. IP)
 - avec identités (chaque sommet possède une identité (un numéro) unique)
- En général, il est plus facile de construire un algo sur des graphes avec identités.
- 41

- L'état d'un processus est codé par une étiquette:
 - Le changement d'état : changement d'étiquette
 - Les algorithmes : arbre recouvrant, élection, terminaison, exclusion mutuelle.
- 42

Quelques rappels: Algorithmes, complexité, graphes

Algorithmes

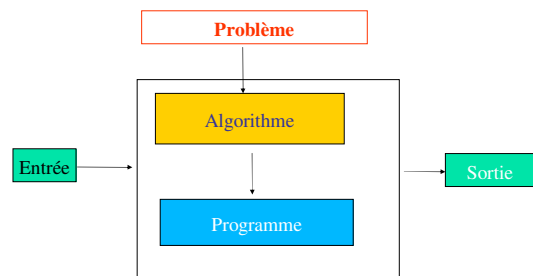
- n Qu'est-ce qu'un algorithme?
- n Efficacité des algorithmes

Qu'est-ce qu'un algorithme?

- n Un algorithme est une méthode systématique (comme une recette) pour résoudre un problème donné.
- n Il se compose d'une suite d'opérations simples à effectuer pour résoudre un problème.
- n En informatique cette méthode doit être applicable par un ordinateur.

Qu'est-ce qu'un algorithme?

Pourquoi un algorithme ?



n **Variable A en Numérique**
Début
A ← 34
A ← 12
Fin

n **Variable A en Numérique**
Début
A ← 12
A ← 34
Fin

L'ordre dans lequel les instructions sont écrites va jouer un rôle essentiel dans le résultat final.

n Exercice 1.1

n Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

n **Variables A, B en Entier**

Début
A ← 1
B ← A + 3
A ← 3
Fin

Fin

n corrigé -

n **Exercice 1.2**

n Quelles seront les valeurs des variables A, B et C après exécution des instructions suivantes ?

n **Variables A, B, C en Entier**

Début

A ← 5

B ← 3

C ← A + B

A ← 2

C ← B - A

Fin

n corrigé -

n **Exercice 1.3**

n Plus difficile, mais c'est un classique absolu, qu'il faut absolument maîtriser : écrire un algorithme permettant d'échanger les valeurs de deux variables A et B, et ce quel que soit leur contenu préalable.

n corrigé -

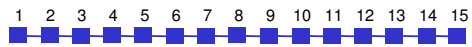
Importance de l'algorithmique

- n Pour un problème donné, il y a plusieurs algorithmes.
- n Il est facile d'écrire des algorithmes faux ou inefficaces.
- n Une erreur peut faire la différence entre plusieurs années et quelques minutes de calculs sur une même machine.
- n C'est souvent une question d'utilisation de structures de données ou d'algorithmes connus dans la littérature.
- n Une structure de données est une façon particulière d'organiser les données.

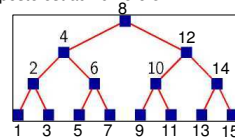
Représenter et organiser les données

Exemple : Construire une ville de 15 maisons en évitant aux facteurs qui suivent les rues un trajet trop long depuis la poste.

Organisation 1 : Linéaire. Numéros croissants. Poste au numéro 1.



Organisation 2 : Embranchements. À l'ouest de la maison k , $n^o < k$, et à l'est, $n^o > k$. La poste est au numéro 8.



Temps de calcul

- n Dans les deux organisations, le facteur a une méthode simple pour trouver une maison en partant de la poste.
- n On suppose qu'il faut une unité de temps pour passer d'une maison à une autre (en suivant une rue).
- n **Quel est, dans le cas le pire, le temps mis par un facteur pour aller jusqu'à une maison depuis la poste?**

Nombre de maisons	Temps organisation 1	Temps organisation 2
15	14	3
1023	1022	9
1073741823	1073741822	29
n	$n-1$	$\sim \log_2(n)$

t

Temps de calcul

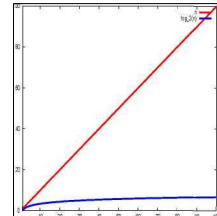
- n Le **temps de calcul** (ou **complexité**) d'un algorithme est la fonction qui à un entier n associe le **nombre maximal d'instructions élémentaires** que l'algorithme effectue, lorsqu'on travaille sur des objets de taille n .
- n En pratique, on se contente d'un ordre de grandeur.
- n **Exemples d'opérations élémentaires :**
 - n additionner, soustraire, multiplier ou diviser deux nombres,
 - n tester si une valeur est égale à une autre valeur,
 - n affecter une valeur à une variable.

Temps de calcul

- Pour déterminer si un algorithme est efficace, on compte le **nombre d'opérations** nécessaire à effectuer **dans le pire des cas** et **en fonction de la taille de la donnée**.
- Le temps de calcul d'un algorithme est une évaluation du nombre d'opérations élémentaires (opérations arithmétiques) qu'il effectue sur une donnée de taille n .
- **Exemple**
 - avec l'organisation 1 de la ville, de taille n maisons, l'algorithme naturel pour trouver une maison a une complexité $O(n)$.
 - avec l'organisation 2 d'une ville de taille n maisons, l'algorithme naturel pour trouver une maison a une complexité $O(\log_2(n))$, ce qui est bien inférieur.

Différence entre n et $\log n$

- Pour notre facteur
 - Si $n = 10^6$, alors $\log_2 n \approx 20$
 - Il fait 50 000 fois moins de déplacements si les maisons sont organisés par « embranchements »
 - Si $n = 10^9$, alors $\log_2 n \approx 30$, il fait alors 30 000 000 fois moins de déplacements.



Temps de calcul : 2ème exemple

Problème : déterminer si 2 ensembles $E1$, $E2$ de n entiers ont une valeur commune.

Algorithme 1 : comparer successivement chaque élément de $E1$ avec chaque élément de $E2 \rightarrow n^2$ comparaisons.

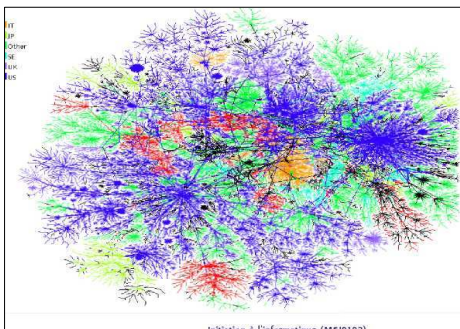
237623	5234	983	83889	9
7363	19	873	111	87321

Différence entre $O(n^2)$ et $O(n \log(n))$

Sur un ordinateur exécutant une instruction élémentaire en 10^{-8} sec

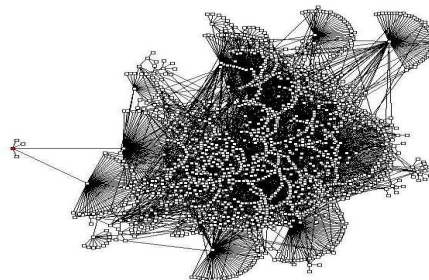
- Si les ensembles $E1$ et $E2$ ont $n = 1\,000\,000 = 10^6$ éléments
 - Exécuter $n \log_{10} n$ instructions élémentaires nécessite 0,06s.
 - Exécuter n^2 instructions élémentaires nécessite 10^4 s soit environ 2h45.
- Si les ensembles $E1$ et $E2$ ont $n = 10\,000\,000 = 10^7$ éléments
 - Exécuter $n \log_{10} n$ instructions élémentaires nécessite 0,7s.
 - Exécuter n^2 instructions élémentaires nécessite 10^6 s soit environ 11 jours.
- En informatique, on manipule parfois des ensembles énormes
 - Google indexe plusieurs milliards de pages web,
 - Google reçoit près de 200 millions de requêtes/jour.

Représentation graphique : internet

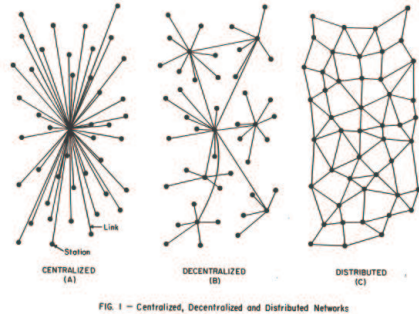


Initiation à l'informatique (MS10102)

Réseau pair-à-pair (P2P)



Représentation graphique : réseaux



Quelques rappels sur les graphes

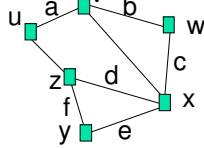
- Les graphes modélisent une grande variété de problèmes: réseaux de communication, réseaux routiers, structures de molécules, etc
- On se ramène à l'étude de sommets et d'arcs.
- Un graphe simple G est formé de deux ensembles : un ensemble V de sommets et un ensemble E d'arêtes. Chaque arête lie deux sommets (qui sont dits adjacents ou voisins)
- On note $G=(V,E)$

Exemple

- $V = \{u,v,w,x,y,z\}$
- $E = \{a,b,c,d,e,f\}$

Le degré d'un graphe:
Nombre d'arêtes adjacentes
 $\text{deg}(x) = 3$

Un graphe est régulier de degré r si tous ses sommets son de degré r .



- Lemme de poignées de main:**

Soit $G=(V,E)$, la somme des degrés des sommets de V est le double du nombre d'arêtes.

Corollaire : un graphe simple a un nombre pair de sommets de degré impair.

Application: est-il possible de relier 27 ordinateurs de sorte que chaque appareil soit relié avec exactement trois autres ?

Terminologie : graphes

- Sous-graphe: $G=(V,E)$ est sous graphe de $G'=(V',E')$ si V est sous ensemble de V' et E est sous ensemble de E' .
- Chaîne: suite de sommets reliés entre eux par une arête.
- Cycle : chaîne qui revient à son point de départ.
- Graphe connexe: pour toute paire de sommets, il existe une chaîne qui les relie
- Arbre: graphe connexe sans cycle
- Anneau: est un graphe formé d'un cycle
- Longueur d'un chaîne: nombre des arêtes qui composent la chaînes
- Distance entre deux sommets: longueur de la plus courte chaîne joignant les deux sommets
- Diamètre d'un graphe: maximum des distances entre les sommets d'un graphe
- Médian: un sommet qui minimise la somme des distances aux autres sommets

Les arbres

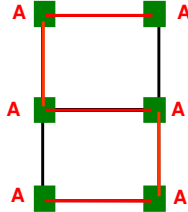
- Soit G un graphe. Nous avons les équivalences suivantes:

- G est un arbre
- Entre toute paire de nœuds, il y a un seul chemin simple
- G est connexe, mais devient non connexe si une arête est supprimée
- G est connexe et $|E|=N-1$
- G est acyclique et $|E|=N-1$
- G est acyclique, mais devient cyclique si on rajoute une arête

Arbre recouvrant

Soit G un graphe. Un arbre recouvrant de G est un sous graphe de G qui est un arbre contenant tous les sommets de G

Exemple:



Arbre recouvrant : Applications

- Structure physique et logique très utilisée en informatique et réseau
- Construire un réseau de diffusion /
 - Chaque noeud reçoit le message une seule fois (évite les duplications, innodations, ..)
 - Minimise les communications (routage efficace)

Arbres recouvrants

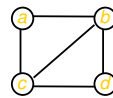
Calcul d'un arbre de coût minimal recouvrant un graphe connexe

Applications conception de réseaux (téléphonique, électrique, d'intercommunication,...) et étude de leur fonctionnement

Algorithmes
 de Prim $O(n^2)$
 (adapté aux matrices d'adjacence)
 de Kruskal $O(a \log a)$
 (adapté aux listes de successeurs et graphes contenant peu d'arêtes)

Graphes non orientés

$G=(S, A)$
 S sommets $\text{card } S = n$
 A arêtes $\text{card } A = a$
 $A = \{\{s, t\}, \dots\} \quad s \neq t, \dots$ (pas de boucle)



chemin : $(\{a,b\}, \{b,c\}, \{c,d\}, \{d,b\}, \{b,a\})$
 chemin simple : $(\{a,b\}, \{b,c\}, \{c,d\})$
 cycle simple : $(\{a,b\}, \{b,c\}, \{c,d\}, \{d,c\}, \{c,b\})$

Chemin : $c = (\{s_0, s_1\}, \{s_1, s_2\}, \dots, \{s_{k-1}, s_k\})$ où les $\{s_{i-1}, s_i\} \in A$
Chemin simple
 les sommets intermédiaires n'apparaissent qu'une seule fois
Cycle simple
 chemin simple avec $s_0 = s_k$

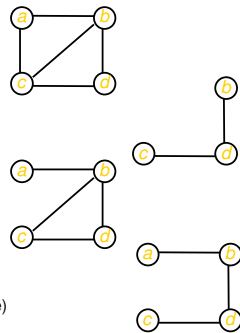
Sous-graphe

Graphe non orienté $G = (S, A)$

Sous-graphe
 $G' \subseteq G : G'$ graphe (S', A')
 avec $S' \subseteq S$ et $A' \subseteq A$

Sous-graphe recouvrant
 si $S' = S$

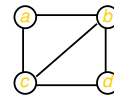
Arbre
 graphe connexe sans cycle (simple)
Arbre recouvrant pour G
 sous-graphe recouvrant qui est un arbre



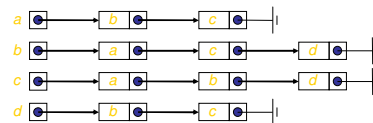
Représentations

Matrice d'adjacence
 (symétrique)

	a	b	c	d
a	0	1	1	0
b	1	0	1	1
c	1	1	0	1
d	0	1	1	0



Listes de successeurs
 (redondantes)

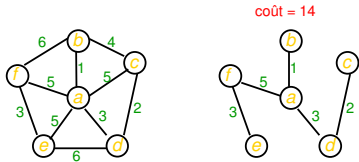


Problème ARCM

Graphe valué $G = (S, A, v)$ avec valuation $v : A \rightarrow \mathbf{R}$
non-orienté et connexe

Coût d'un sous-graphe $G' = (S', A')$: $\sum (v(p,q) \mid (p,q) \in A')$

Problème : déterminer un ARCM, arbre recouvrant de coût minimal pour G

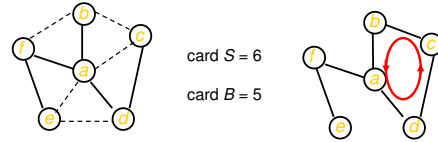


Les arbres recouvrants

Graphe non-orienté et connexe $G = (S, A)$ $\text{card } S = n$
 $T = (S, B)$ arbre recouvrant pour G

Propriétés

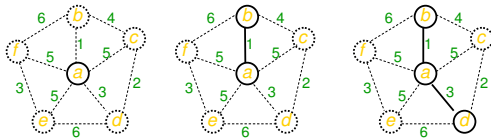
T possède $n-1$ arêtes : $\text{card } B = n-1$
 _____ si $\{p, q\} \in A-B$ alors $H = (S, B + \{u,v\})$ possède un cycle



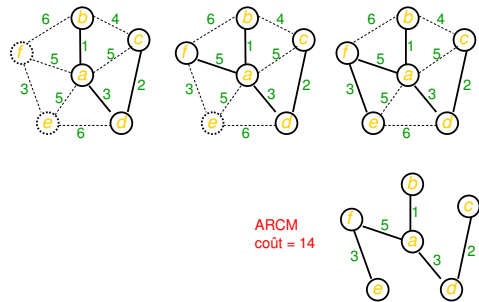
Méthode de Prim

Calcul d'un ARCM :
faire grossir un sous-arbre jusqu'au recouvrement du graphe

Exemple



Exemple (suite)



Algorithme de Prim

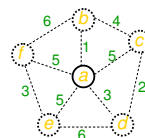
```

arbre PRIM( graphe  $(\{1, 2, \dots, n\}, A, v)$  ) {
   $T \leftarrow \{1\}$ ;
   $B \leftarrow \emptyset$ ;
  tant que  $\text{card } T < n$  faire {
     $\{p, q\} \leftarrow$  arête de coût minimal
    telle que  $p \in T$  et  $q \notin T$ ;
     $T \leftarrow T + \{q\}$ ;
     $B \leftarrow B + \{p, q\}$ ;
  }
  retour  $(T, B)$ ;
}
    
```

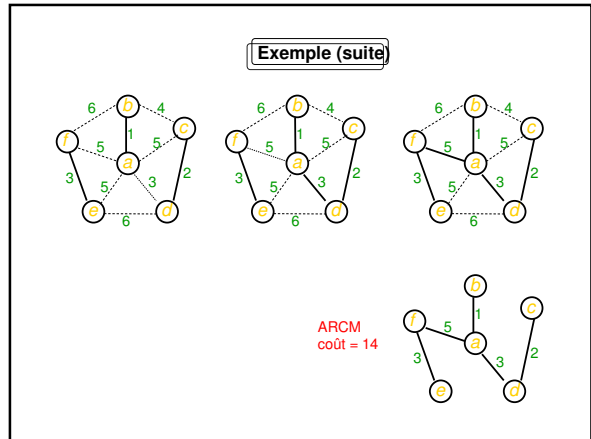
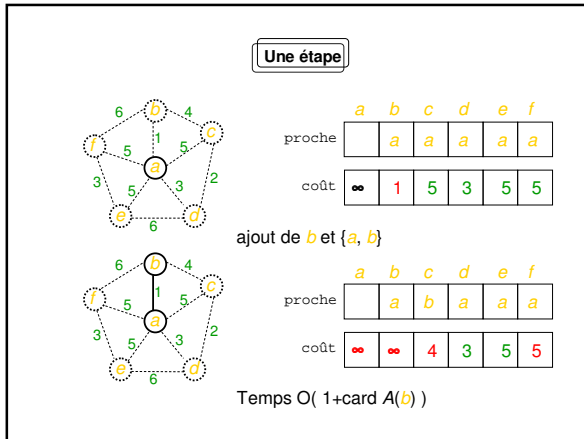
Temps d'exécution : $O(n^2)$ au moyen de deux tables indexées par les sommets

Implémentation

Tables **proche** et **coût** pour trouver l'arête $\{p, q\}$
 $q \notin T$ **proche** $[q] = p \in T$
 ssi $v(p, q) = \min \{ v(p', q) \mid p' \in T \}$
 $q \notin T$ **coût** $[q] = v(\text{proche}[q], q)$
 $q \in T$ **coût** $[q] = +\infty$

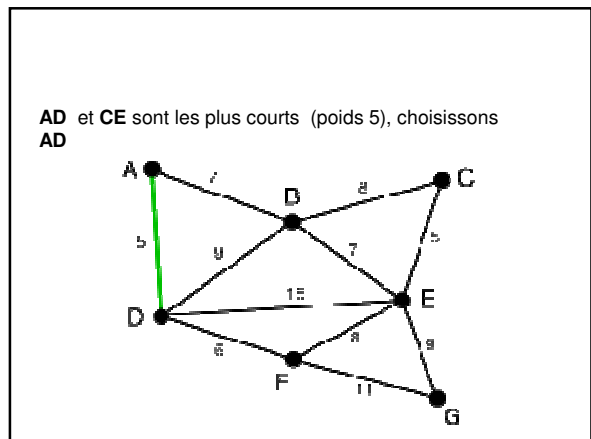
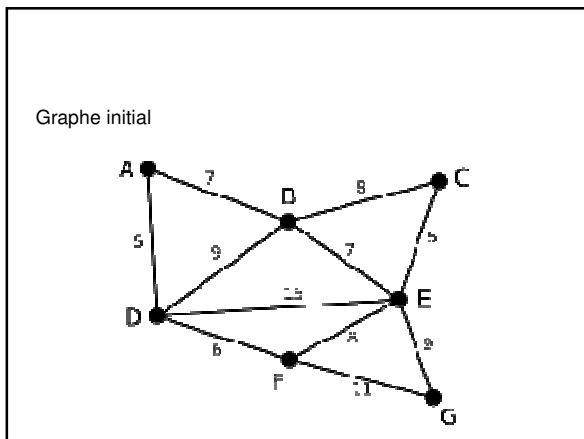
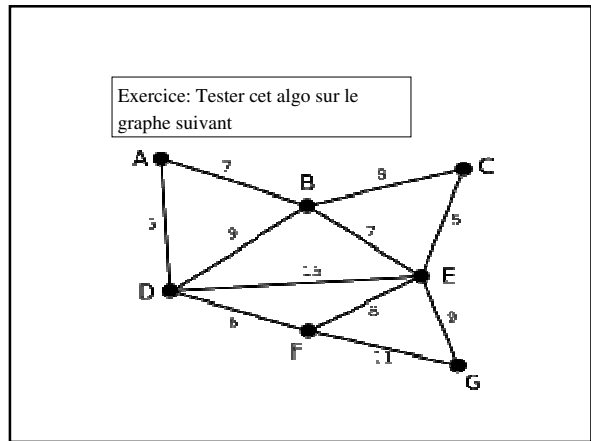


	a	b	c	d	e	f
proche		a	a	a	a	a
coût	∞	1	5	3	5	5

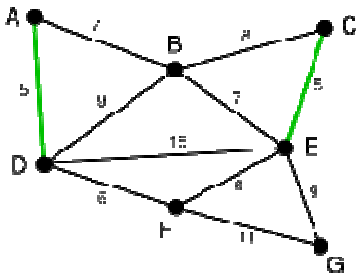


Algorithme de Kruskal

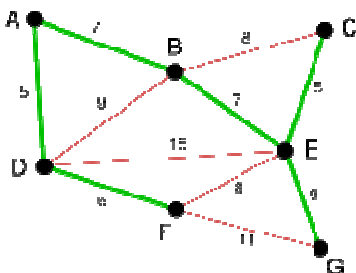
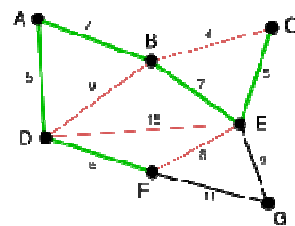
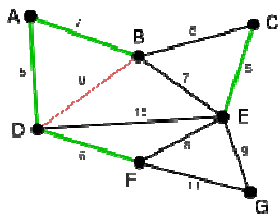
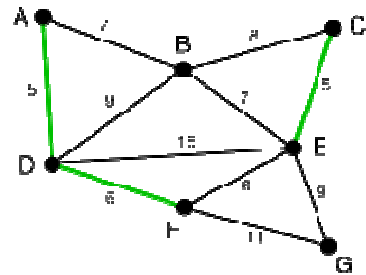
- n Créer une forêt F (un ensemble d'arbres) où (initialement) chaque sommet est un arbre séparé.
- n Créer un ensemble S contenant toutes les arêtes du graphes
- n Tant que S est non vide
 - n supprimer l'arête a de poids minimum de S
 - n Si l'arête a connecte deux arbres, alors la rajouter
 - n sinon l'ignorer



CE est la plus petite arête qui ne forme pas de cycle.



Ensuite DF



Complexité

- Complexité de l'algorithme de Kruskal $O(|E|\log|E|)$

ⁿ Exercice

Ecrire un algorithme qui affiche les valeurs contenus dans les noeuds d'un arbre ?

Ecrire une fonction qui calcule le maximum des noeuds d'un arbre

ⁿ On utilise les notations suivantes:

r: racine

Pour un noeud v, pere(v) est le père de v;

Fils (v) est la liste des fils de v.

Feuille(v) est une fonction qui renvoie une feuille

Parcours infixé, préfixé, postfixé

Parcours en largeur, en profondeur

Modèle de l'algorithmique distribuée

I/ Notion d'algorithme distribué

- Un algorithme distribué sur un système distribué S est une collection de transitions locales à réaliser séquentiellement par chaque process de S
- Ces transitions peuvent s'effectuer suite à des événements (internes ou échange de message)

- Processus:

- Les instructions d'un processus sont considérées comme atomiques.
- Il possède une mémoire locale
- Il possède un **état local (ensemble de ses données et des valeurs de ses variables locales)**
- Il peut (ou non) avoir un identifiant
- Pas ou peu de connaissance des autres processus et de leur état (il peut connaître l'état des voisins)
- Il s'exécute en parallèle que les autres processus

- Un algorithme désigne celui d'un seul process et protocole désigne l'ensemble des algos (mais on ne fait cette distinction)
- Événement: interne (changement d'état), envoi de message, réception de message
- Process p: (etat_i, evenmt_j) (etat_{i+1},evenmt_{j+1})
- Simulation d'un mémoire partagée: 1 process à part qui est chargé uniquement de la communication
- Terminaison: explicite vs implicite

II/ Modèle du réseau

Process = sommet ; Canal = arête

1. Types de réseaux

Réseaux centralisés: Il existe un seul process dans l'état centraliseur (ou élu) et tous les autres dans l'état battu

Réseaux avec identités: (named network) Chaque process de S est distingué par l'attribution d'une identité unique, distincte de tous les autres et connue par lui seul

Réseaux anonymes: les process sont symétriques du point de vue de l'exécution de l'algorithme et ne possèdent pas d'identités (connues d'eux). Ils sont indiscernables.

Propriétés du réseau

Le graphe sous-jacent est connexe

Les messages sont acheminés le long des canaux. Les délais sont finis mais non bornés (peuvent être arbitrairement longs)

Les messages reçus par un process sont traités dans leur ordre d'arrivée (si en //, ordre arbitraire)

Pour tout couple (p_i, p_j) , l'ordre de réception des messages est le même que celui de transmission (FIFO)

- Les états peuvent être codés par des étiquettes
- Topologie du réseau (arbre, anneau, complet,...)

III/ Modèle de communication (synchronisme)

1. Synchrone

- horloge globale

- L'exécution est partitionnée en rounds

(un round correspond à un envoi et à une réception de tous les messages. Envoi instantané ou borne fixe)

- Les process ont la même vitesse

Asynchrone

Pas d'horloge globale

Pas de borne sur le délai de transmissions de messages

Les process sont asynchrones (les horloges sont locales)

Modèles de pannes

- Modèle de fautes : 2 grands types de fautes ou pannes
 - Franches : le processus ne fait plus rien
 - Arbitraires ou byzantines : le processus renvoie des valeurs fausses et/ou fait « n'importe quoi »
 - Cas de fautes les plus complexes à gérer
 - Les autres fautes peuvent être considérées comme des cas particuliers des fautes byzantines
- Processus correct
 - Processus non planté, qui ne fait pas de fautes
 - Dans le cas où on considère les reprises après erreurs et la relance de processus : processus qui pouvait être incorrect précédemment mais qui est correct maintenant

- **Fautes arbitraires / byzantines**

- Fautes arbitraires
 - Appelées aussi fautes byzantines
 - Fautes quelconques et souvent difficilement détectables
- Processus
 - Calcul erroné ou non fait
 - État interne faux
 - Valeur fautive renvoyée pour une requête
- Canal
 - Message modifié, dupliqué voire supprimé
 - Message « créé »
 - En pratique peu de fautes arbitraires sur les canaux car les couches réseaux assurent la fiabilité de la communication
- Fautes arbitraires : classement en 2 catégories
 - Malicieuses : erreurs volontaires (virus, attaques ...)
 - Naturelles : problème non voulu, non déclenché

IV/ Mesures de complexité

Modèle synchrone :

- Nombre de messages : depuis le début jusqu'à la fin de l'algo.
- Taille d'un message : complexité en nombre de bits
- Temps : nombre de rounds

- **Modèle asynchrone :**

- Nombre de messages
- Taille d'un message
- Temps (moins évident) : généralement, on suppose que le temps de transmission sur un canal de communication mesure une unité

(ou bornée par une constante)

- Complexité en espace : espace pour un process
- Calcul de complexité au pire : max sur toutes les exécutions possibles (généralement le pire des exécutions synchrones)
- Complexité moyenne : valeur moyenne de la complexité (selon une distribution de probabilités)

V/ Quelques algorithmes de base

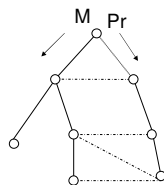
1. Un algorithme de diffusion (Broadcast, PI)

- Hypo: un arbre recouvrant donné, un message M
- Un sommet P_r distingué (racine)
- P_r envoie M à tous ses fils
- Si un sommet reçoit le message M de son père, il l'envoie à ses fils

Algo pour $P_i \quad i \neq r$

```

receive (pere( $P_i$ ),M)
pour tout fils  $P_j$  de  $P_i$ 
send ( $P_j$ ,M)
terminer
  
```



- Exercice: Calculer la complexité en nombre de messages ? (temps)
- Algorithme echo ?

2. Inondation et arbre recouvrant :

un graphe G, un message M à envoyer à tous les sommets

2.1/ Inondation, Diffusion, PI (Propagation d'Information)

- Un sommet distingué (racine) P
- P envoie M à tous ses voisins
- Un process P_i qui reçoit M de P_j pour la première fois, l'envoie à tous ses voisins sauf à P_j

Exercice:

Montrer que la complexité en messages est de $2m-n+1$

2.2/ Application à la construction d'un arbre recouvrant

Quand P_i reçoit M pour la première fois de P_j , il envoie à P_j un message <parent> et <refus> à tous les autres.

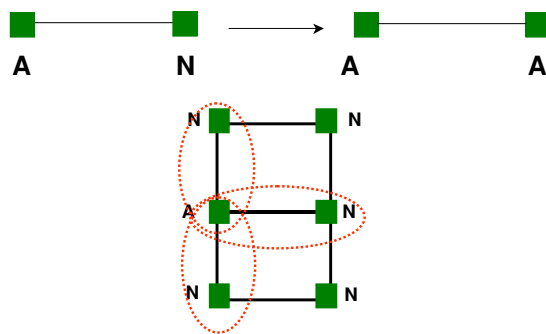
Si P_j reçoit <parent> de P_i alors fils(P_j) = P_i

Algorithmes de diffusion
Calcul d'arbres recouvrants

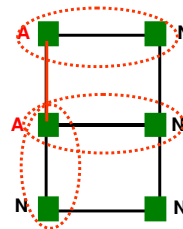
Introduction

- Etat d'un process codé par une étiquette
- (selon son état, les états de ses voisins) une étape de calcul est effectué
- Changement d'état: « réétiquetage » ou « relabeling »
- Utilisation des règles de réétiquetage
- Buts :
 - Abstraction
 - simplification
 - facilité de preuves
 - indépendance de tout modèle
 - implémentation unifiée

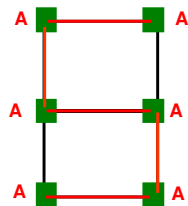
Exemple



Etape de calcul



un arbre recouvrant



Codage d'algorithmes distribués

- Un sommet étiqueté A (Actif), et tous les autres sont étiquetés N (neutre)
- Règle:
 - Si un sommet x étiqueté A possède un voisin y étiqueté N
 - Alors x est étiqueté A et l'arête (x,y) est marquée
- Résultats : arbre recouvrant
- plusieurs règles peuvent être appliquées en //

Preuves :

- Terminaison : noethérien
- Correction : invariants
- Cet algorithme est distribué ne *détecte pas localement la terminaison globale*.
- D'autres systèmes de réécriture permettent de capturer cette propriété.

Définitions : systèmes de réétiquetage de graphes

- Un système de réétiquetage (GRS) est un triplet $R=(L,I,P)$ où L est un alphabet, I sous ensemble de L , P est un ensemble de règles de réétiquetage. Chaque règle est de la forme (Gr, λ_r) (Gr, λ'_r) , notée aussi $(Gr, \lambda_r, \lambda'_r)$
- Une étape de réétiquetage est une application d'une règle
- un graphe est (G, λ) est R -irréductible si aucune règle de R n'est applicable
- **Système de réétiquetage avec priorités**
 $R=(L,I,P,>)$ est un système de réétiquetage avec priorités si (L,I,P) est un SR et $>$ est un ordre partiel défini sur l'ensemble P (relation de priorité). Une règle prioritaire s'applique en 1er en cas de conflit.

- **Contexte interdit**: Un contexte d'un graphe (G, λ) est un sous graphe de (G, λ) à isomorphisme près.

Un règle de réétiquetage avec contexte interdit est une $(Gr, \lambda_r, \lambda'_r, F_r)$ avec F_r : contexte interdit

Un système de réécriture avec contexte interdit est un système de réécriture ou chaque règle est avec contexte interdit

Une règle est appliquée sur un sous graphe s'il ne contient aucun de ses contextes interdits

Application à l'étude des algos distribués

- Ramener l'étude d'un algo distribué à l'étude d'un système de réétiquetage qui le code
 - plus simple, facile à comprendre, à prouver etc.
- 1/ **Terminaison** : un algorithme distribué termine si le SR n'induit pas de chaîne de réécriture infinie (s'il est noethérien). Pour prouver qu'un système est noethérien, il suffit d'exhiber un ordre noethérien compatible avec le SR: trouver un ensemble partiellement ordonné $(S, <)$ qui n'a pas de chaîne infinie décroissante et une fonction $f : GL \rightarrow S / (G, \lambda) \rightarrow (G, \lambda')$ alors $f(G, \lambda) > f(G, \lambda')$
- 2/ **Correction** : invariants
- 3/ **Complexité** : nombre de réécritures

- Exemple (Arbre recouvrant avec SR1)
- $SR1 = (L1, I1, P1)$

Théorème:

1. SR1 est noethérien
2. Soit (G, λ) un graphe étiqueté dans $I1$, avec un seul sommet étiqueté A , et soit (G, λ') un graphe SR1-irréductible. Le sous-graphe induit par les arêtes étiquetées 1 est un arbre de G .
3. La longueur d'une séquence est au plus $n-1$ (avec n le nombre de sommets de G)

- SR1 est noethérien

- Preuve

$f : GL1 \rightarrow N$

$(G, \lambda) \rightarrow |VN|$

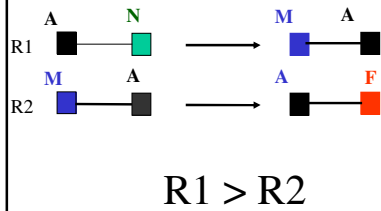
L'ordre $>$ sur N est compatible avec le système de réécriture SR1

- Invariants:

- P1 : Toute arête incidente à un sommet étiqueté N est étiquetée 0
- P2 : Tout sommet étiqueté A (à part la racine) est incident à au moins une arête étiquetée 1
- P3 : Le sous graphe induit par les arêtes étiquetées 1 est un arbre

Preuve : par récurrence sur les chaînes de réécriture

Calcul d'un arbre recouvrant séquentiel avec détection de la terminaison (système SR2)

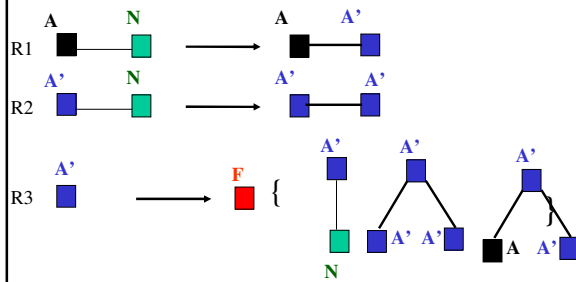


• **Théorème** : SR2 calcule un arbre recouvrant. Il détecte localement la terminaison globale.

• Preuve:

- SR2 est noethérien
- SR2 est correct (le calcul est un arbre recouvrant)
 - P1: Toutes les arêtes incidentes à N sont étiquetées 0
 - P2: Tout sommet étiqueté A, F ou M est incident à moins une arête étiquetée 1
 - P3: le sous graphe induit par les arêtes étiquetées 1 est un arbre
 - P4: il y a exactement un seul sommet étiqueté A
 - P5: le sous graphe induit par les sommets étiquetés A et M et les arêtes étiquetées 1 est un chemin ayant une extrémité étiquetée A
 - P6: tout sommet étiqueté F n'a aucun voisin étiqueté N

Distribué avec terminaison (SR3)



• **Théorème**: le système SR3 calcule un arbre recouvrant avec détection locale de la terminaison globale

Preuve:

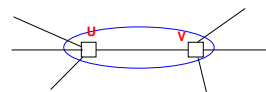
- SR3 est noethérien
- le résultat est un arbre recouvrant
- détection locale de la terminaison globale: A entourée par des F

Calculs locaux

- généralisation des SR (réécriture de boules)
- conditions:
 - C1 : ne modifie pas le graphe sous-jacent. Seul l'étiquetage est modifié.
 - C2 : ils sont locaux, chaque étape ne concerne qu'un sous graphe de taille fixe dans le graphe sous-jacent
 - C3 : les réétiquetages sont localement contrôlés (ou engendrés); l'application d'une règle ne dépend que du contexte local du sous graphe

Implémentation de calculs locaux.

- Implémentation par des procédures
- Trois types de calculs locaux:
- computations:
 1. **Rendez vous** : Etiquettes d'un sous graphe k2 sont modifiés selon des règles qui dépendent de ces étiquettes.



Implémentation (Procédure Rendezvous).

Chaque sommet exécute les actions suivantes:

- un sommet v choisit au hasard un de ses voisins $c(v)$;
- v envoie 1 à $c(v)$;
- v envoie 0 à tous les sommets différents de $c(v)$;
- v reçoit les messages de tous ses voisins.

(* Il y a un rendez-vous entre v et $c(v)$ si v reçoit 1 de $c(v)$. Une étape du calcul peut avoir lieu*)

1. **Calcul Local 1 (LC1):** Pour une boule de rayon 1 (une étoile), l'étiquette du centre de la boule est modifiée selon une règle qui dépend de celle des sommets de la boule.
2. **Calcul Local 2 (LC2):** Comme pour LC1 sauf que tous chaque sommet de la boule modifie son étiquette.

Implémentation de LC1 (Election locale probabiliste):

Le sommet v tire au hasard un entier $rand(v)$;

v envoie $rand(v)$ à tous ses voisins;

v reçoit les entiers de ses voisins;

(* le sommet v est élu dans $B(v,1)$ si $rand(v)$ est strictement plus grand que tous les entiers reçus par v ; dans ce cas un pas de calcul LC1 peut être effectué sur $B(v,1)$ *)

Implémentation de LC2 (Election locale probabiliste):

Le sommet v tire au hasard un entier $rand(v)$;

v envoie $rand(v)$ à tous ses voisins;

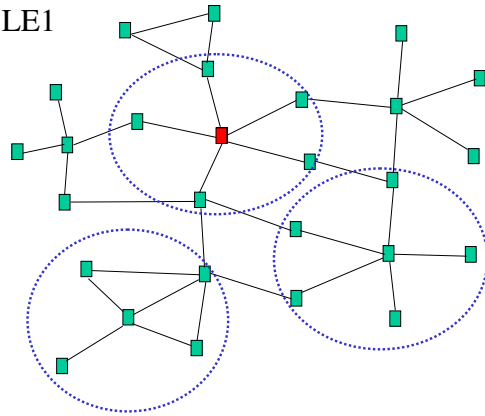
v reçoit les entiers de ses voisins;

v envoie le max des entiers reçus à chacun de ces voisins;

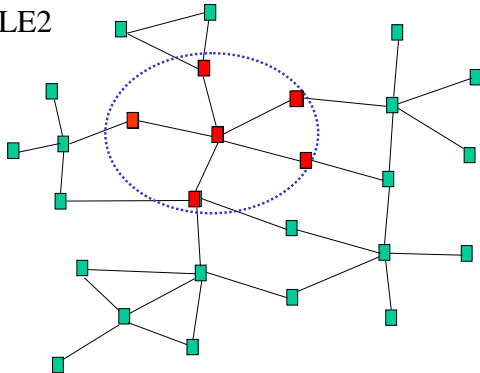
v reçoit les entiers de ses voisins.

(* le sommet v est élu dans $B(v,2)$ si $rand(v)$ est strictement plus grand que tous les entiers reçus par v ; dans ce cas un pas de calcul LC2 peut être effectué sur $B(v,1)$ *)

LE1



LE2



Un méta-algorithme

Chaque processeur exécute

```
While (run){
  \ Synchronisation (rendezvous, LE1, LE2);
  \ Echange d'étiquettes, attributs,
  \ calculs;
  \ Mise à jour des étiquettes, attributs, états;
  \ Arrêt de la Synchronisation;
}
```

Exemple (Calcul d'un arbre recouvrant – Rendez-vous)

```
while (run) {
    neighbour = rendezVous();
    sendTo(neighbour,myLabel);
    neighbourLabel=receiveFrom(neighbour);
    if (myLabel == 'N') && (neighbourLabel == 'A'){
        myLabel = 'A';
        edge[neighbour]=1
    }
    breakSynchro();
}
```

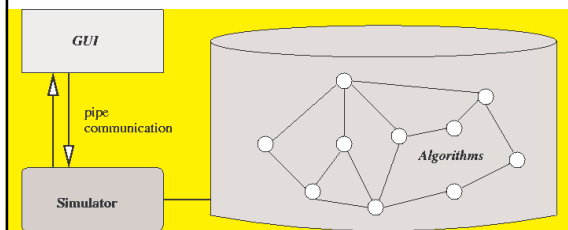
Application à la visualisation de l'exécution d'un algorithme distribué

- Implémentation à la visualisation de tous les évènements.
- Bibliothèque de primitives.
- Les propriétés du système de réécriture sont préservées: : consistance, ordre des évènements, terminaison, résultats.

ViSiDiA

- Un environnement pour simuler, visualiser et expérimenter les algorithmes distribués.
- Un processeur est implémenté par un thread Java.
- Une bibliothèque de primitives de haut niveau est disponible pour le programmeur.
- Une interface graphique (contrôlable par l'utilisateur) permet à l'utilisateur de
 1. Construire d'éditer un réseau (avec GML export/import).
 2. visualiser l'exécution d'un algorithme ou de l'expérimenter.

Architecture



- *Modulaire, orienté-objet, portable, interactif, limites (celles de la java)..*

Algorithmes d'Election

Election

- Chaque process décide s'il est élu ou non
- Résultat de l'algo: - un seul sommet au élu
- tous les autres non élus
- Applications :
 - un algorithme centralisé doit être exécuté dans un environnement distribué
 - après un crash
 - Sauvegarde des ressources
- Une classe de problèmes pour briser la symétrie

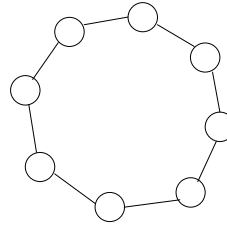
Définitions :

un algorithme d'élection est un algorithme qui satisfait les propriétés suivantes:

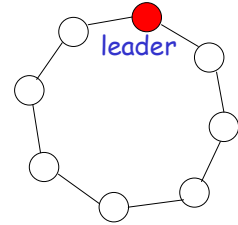
- chaque processeur exécute le même algo. local
- l'algorithme est décentralisé, i.e. le calcul peut être initialisé par un sous ensemble de process
- l'algorithme termine dans tout calcul, et la configuration terminale
 - un seul élu (ou leader)
 - tous les autres : non élu (ou perdant)

Election dans un anneau

Etat initial



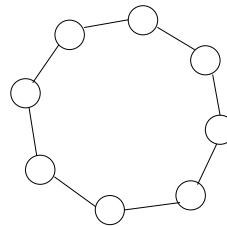
Etat final



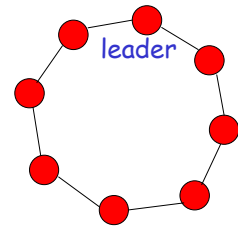
Election dans les anneaux anonymes

- Les processeurs d'un anneau anonyme ne possèdent pas d'identifiants uniques (utilisés par l'algorithme)
- Tous les processeurs sont symétriques (ils envoient les mêmes messages, ils reçoivent les mêmes réponses, ils sont dans le même état)
- Il n'existe pas d'algorithmes d'élection dans un anneau anonyme (dans un modèle synchrone)

Etat initial



Etat final

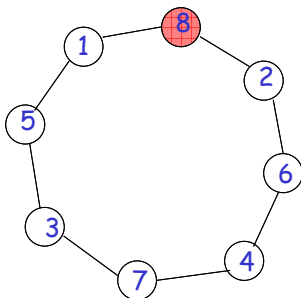


Si un noeud est élu,
alors tout noeud est élu

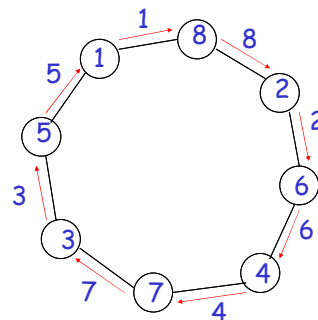
De même pour le modèle asynchrone

Anneau avec identités

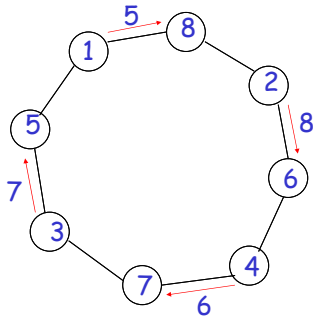
Le noeud avec id max est élu



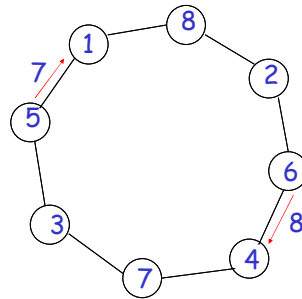
Chaque noeud envoie un message contenant
Son id au voisin suivant



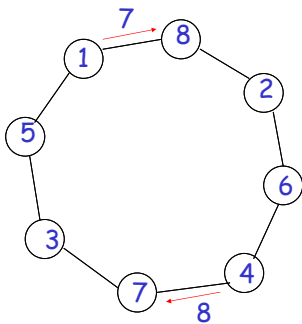
si: id reçu est > id courant
lors: faire passer id



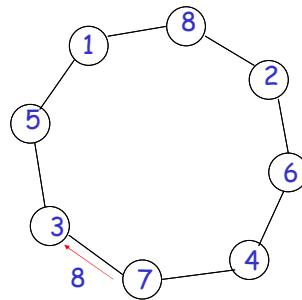
si: id reçu est > id courant
lors: faire passer id



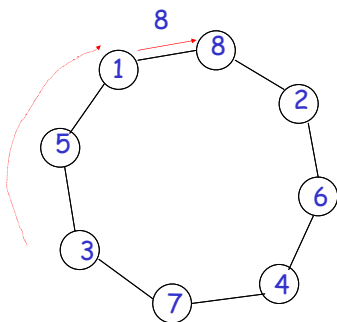
si: id reçu est > id courant
lors: faire passer id



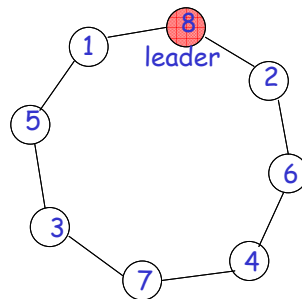
si: id reçu est > id courant
lors: faire passer id



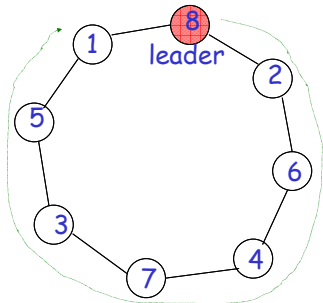
si: id reçu est > id courant
lors: faire passer id



si: un noeud reçoit son propre id
lors: il est élu

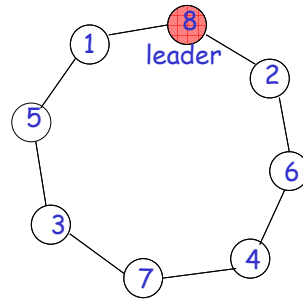


Le leader envoie un message au réseau pour informer qu'il est élu



Complexité temps: $O(n)$ n noeuds

Remarque: n peut ne pas être connu par l'algo

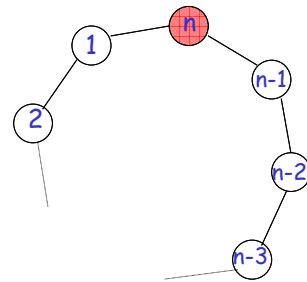


Exercice:

Soit un anneau avec des identités de sommets numérotés de 1 à n . Quel est la configuration de l'anneau qui représente le pire des cas pour le nombre de messages ?

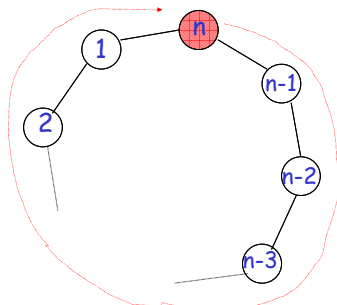
complexité messages: $O(n^2)$ n noeuds

Pire cas:



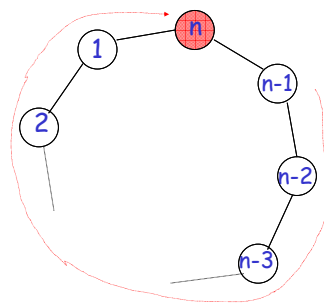
complexité messages: $O(n^2)$ n noeuds

n messages



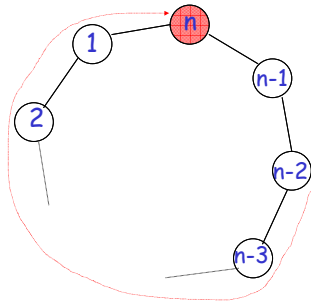
complexité messages: $O(n^2)$ n noeuds

$n-1$ messages



complexité messages: $O(n^2)$ n noeuds

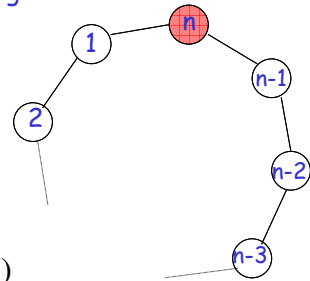
$n - 2$ messages



complexité messages: $O(n^2)$ n noeuds

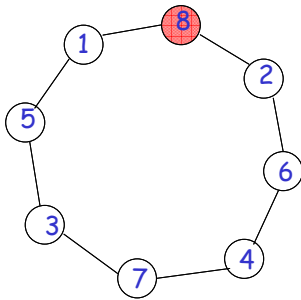
Nbre total messages:

$n +$
 $n - 1 +$
 $n - 2 +$
 \dots
 $2 +$
 $1 = O(n^2)$



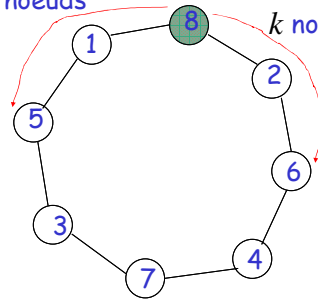
Un algorithme $O(n \log n)$

Election du noeud avec id maximal

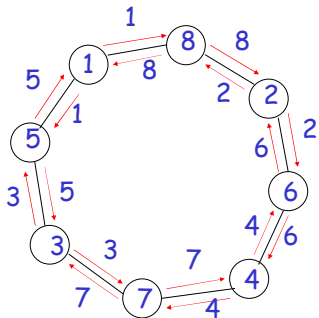


k - voisin

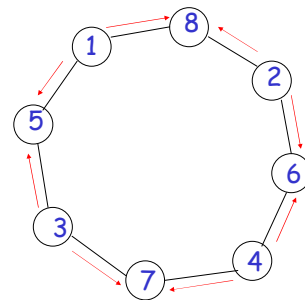
k noeuds k noeuds



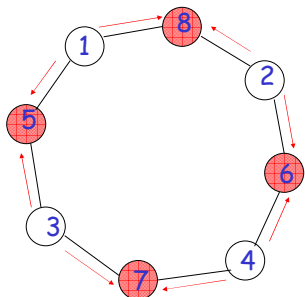
hase 1: envoyer id aux 1-voisins



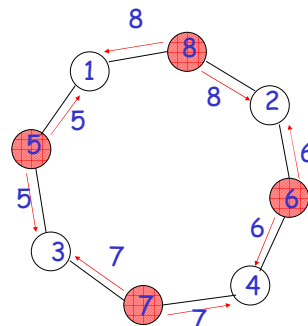
Si: id reçu > id courant
 alors: envoyer une réponse



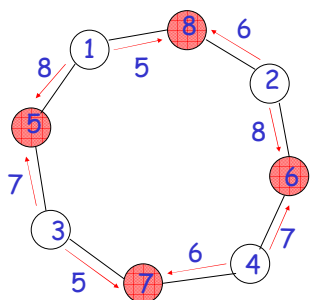
Si: un oeu^d recoit les deux réponses
 alors: le noeud devient un leader local



Phase 2: envoyer id aux 2-voisins

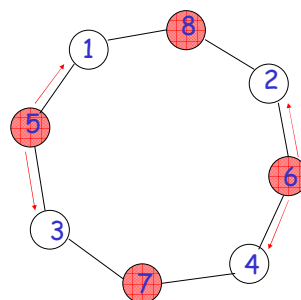


Si: id recu > id courant
 alors: faire passer le message

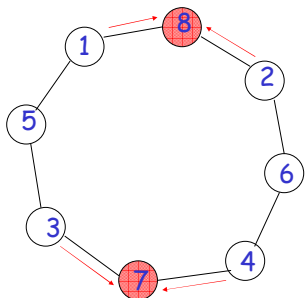


Seconde étape:

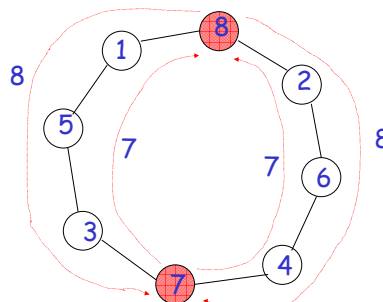
Si: id recu > id courant
 alors: envoyer une réponse



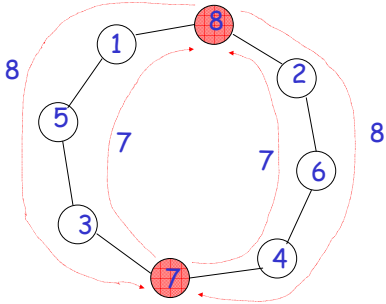
Si: un noeud recoit deux réponses
 alors: il devient leader temporaire



Phase 3: envoyer aux 2²voisins

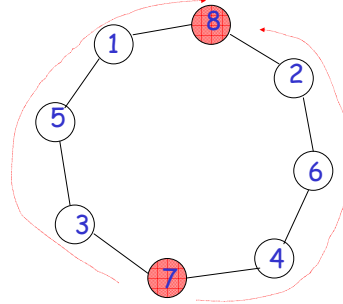


Si: id reçu > id courant
 alors: faire passer the message

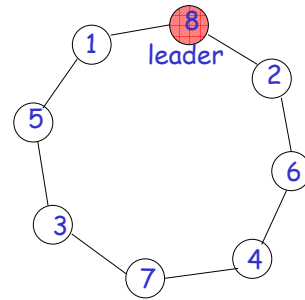
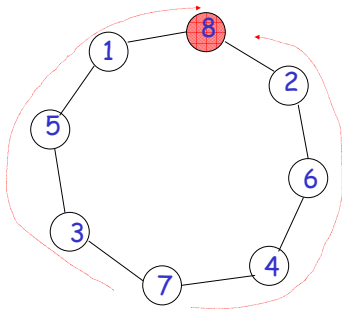


A l'étape: 2^2

Si: id reçu > id courant
 alors: envoyer une réponse

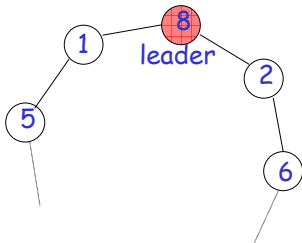


Si: un noeud recoit les deux réponses
 alors: il devient leader

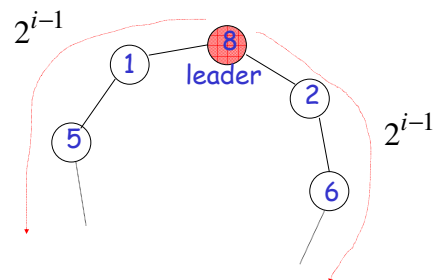


En général:

n noeuds \rightarrow $\log n$ phases



Phase i : envoyer id au 2^{i-1} -voisins



Complexité en temps

Le coût du calcul du leader en temps

Phase 1: 2
 Phase 2: 4
 ...
 Phase i: 2^i
 ...
 Phase log n: $2^{\log n}$

Total : $2^{\log n + 1} - 2 = O(2^{\log n}) = O(n)$

complexité en messages

Nombre de messages
 par leader

	leaders
Phase 1: 2	$n/2$
Phase 2: 4	$n/4$
...	
Phase i: 2^i	$n/2^i$
...	
Phase log n: $2^{\log n}$	$n/2^{\log n}$

Nombre de messages
 par leader

	leaders	
Phase 1: 2	×	$n/2 = n$
Phase 2: 4	×	$n/4 = n$
...		
Phase i: 2^i	×	$n/2^i = n$
...		
Phase log n: $2^{\log n}$	×	$n/2^{\log n} = n$

Nbre de messages: $O(n \cdot \log n)$

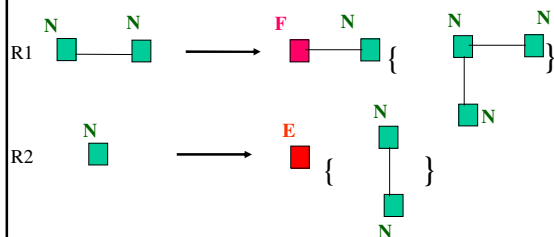
Formulation à l'aide des systèmes de réécriture

On dit qu'un SR résout le problème d'élection pour une classe de graphes C si

- il est noéthérien
- Il existe deux labels N et T / pout tout G dans C, initialement étiqueté N, si (G, λ') tel que Irred (G, λ) alors il existe exactement un sommet v dans $V(G)$ étiqueté T, $\lambda(v) = T$

A/ Election dans un arbre

A.1/ Election dans un arbre.

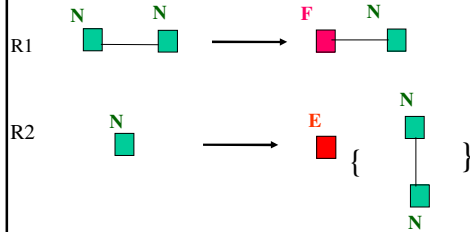


Implémentation de l'élection dans un arbre

```
while(run){
    synchro=starSynchro();
    Si( synchro==starCenter ){
        int n_Count=0;

        for (int door=0;door<arity;door++) {
            neighbourState[door]=receiveFrom(door);
            Si (neighbourState[door]=="N")
                n_Count++;
        }
        Si ((myState=="N") && (n_Count ==1))
            changeState(F);
        else
            Si ((myState=="N") && (n_Count==0)) {
                changeState(E);
                run=false;
            }
        breakSynchro();
    }
    else {
        Si (synchro == starLeaf) {
            sendTo(center,myState);
        }
    }
}
/* demo */
```

A.2/ Election dans un graphe complet

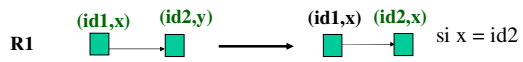
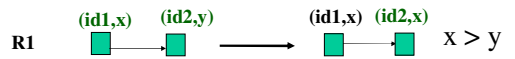


Implémentation de l'élection dans un graphe complet

```

while(run){
    neighbour=rendezVous();
    sendTo(neighbour,myState);
    neighbourLabel=receiveFrom(neighbour);
    Si ((myState == "N") && (noNeighbourN))
        myState ="E"
    Si ((myState == "N") && (neighbourLabel== "N")) {
        myState="F";
    }
}
    
```

Election dans un anneau avec identités



- Initialement : (i,i)
- garder la plus grande identité