IF223 - Algorithmique Distribuée - TD3

Rohan Fossé

rfosse@labri.fr

Exercice 1

On considère l'Algorithme vu en cours. Cet algorithme construit un arbre couvrant (*spanning tree* en anglais), étant donnée la racine de l'arbre. On considère un système distribué représenté par le graphe en Figure 1, où chaque processus a un identifiant unique dans l'ensemble {a, b, ... f } et le processus a est la racine de l'arbre à construire, c'est-à-dire pr = a dans le pseudo-code.

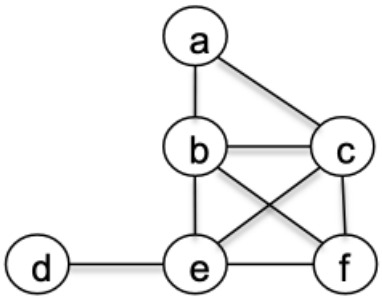


Figure 1: Figure 1

1. Donner deux arbres couvrants différents enracinés en a pour le graphe en Figure 1.
2. On suppose que le système est synchrone. Donner une exécution de l'algorithme et montrer l'arbre couvrant généré par cette exécution. Est-il possible de construire un autre arbre couvrant enraciné en a en exécutant le même algorithme sur le même graphe ?
3. On considère maintenant un système asynchrone. Est-il possible que deux exécutions différentes construisent deux arbres couvrants différents ? Si oui, donnez ces exécutions en forme de suite de graphes.
4. On considère les exécutions admissibles pour le modèle asynchrone. pourquoi tout processus aura un et un seul parent ?
   Pourquoi ce n'est pas possible qu'un cycle soit crée?

Exercice 2

On considère l'algorithme en Figure 2. Quel problème résout cet algorithme ? Expliquer en français le fonctionnement de l'algorithme. Il y a un processus p_r désigné avant l'exécution. Les valeurs initiales pour les variables locales à un processus p_i pour tout $i \in \{0, \ldots, n-1\}$ sont :

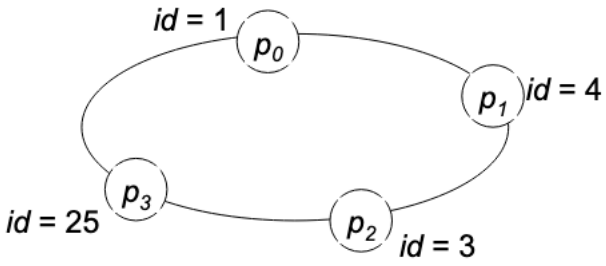
- parent = $\perp$
- children = $\emptyset$
- unexplored = tout les voisins de p_i

```
1:  upon receiving no message:
2:      if p_i = p_r and parent = ⊥ then
3:      parent := p_i
4:      explore()

5:  upon receiving ⟨M⟩ from p_j:
6:      if parent = ⊥ then                    // p_i has not received ⟨M⟩ before
8:          parent := p_j
9:          remove p_j from unexplored
10:         explore()
11:     else
12:         send ⟨already⟩ to p_j
13:         remove p_j from unexplored
14: upon receiving ⟨already⟩ from p_j:
15:     explore()

16: upon receiving ⟨parent⟩ from p_j:
17:     add p_j to children
18:     explore()

19: procedure explore():
20:     if unexplored ≠ ∅ then
21:         let p_k be a processor in unexplored
22:         remove p_k from unexplored
23:         send ⟨M⟩ to p_k
24:     else
25:         if parent ≠ p_i then send ⟨parent⟩ to parent
26:         terminate
```

Figure 2: Algorithme mystère: pseudo-code pour le processus p_i

Exercice 3

On considère le système distribué en Figure 3. En supposant que le système est synchrone, donner une exécution de l'algorithme pour l'élection d'un leader avec complexité $\Omega(nlogn)$ où n est le nombre de processus dans le système. Expliquer le rôle des informations insérées dans les messages < probe, j, k, d > et < reply, j, k >.



Exercice 4

On considère un anneau anonyme de n processus où chaque processus a une entrée 1 ou 0. Tous les processus doivent retourner 0 s'il existe un processus dont le bit d'entrée est 0, sinon ils doivent retourner 1.

- Concevoir un algorithme non uniforme asynchrone qui résout le problème avec une complexité en message en O(n2).
- Concevoir un algorithme non uniforme synchrone qui résout le problème avec une complexité en message en O(n).