Université de CAEN
U.F.R. de Sciences

Licence d'informatique
Semestre 6- E I61 - Algo

Séquence 1 - parcours de graphes
Jean.Saquet@info.unicaen.fr

1 Hypothèses

1.1 Sur le réseau

- Le réseau sera représenté par un graphe connexe G = (X,U) où X est l'ensemble des sommets (les processus P_i) et U est l'ensemble des lignes de communication, de topologie quelconque.
- On notera les sommets P_1, P_2, $\cdots$, P_n et l'initiateur (l'un d'entre eux) sera noté P_{init}.
- Les messages ne sont ni perdus, ni altérés.

1.2 Sur l'environnement des processus

H1 soit P_i connaît les lignes auquelles il est connecté (sans connaître l'identité de ses voisins),
H2 soit P_i connaît l'identité de ses voisins.

2 Arborescence recouvrante et signalisation de la terminaison

2.1 Le problème

Etant donné un graphe G = (X,U), un des sites que l'on note P_{init} initie la construction d'une arborescence recouvrante de G, dont il sera la racine.
Pour obtenir une arborescence recouvrante, il est nécessaire :
- (i) que tout processus autre que P_{init} ait un prédécesseur unique,
- (ii) que deux processus distincts aient des ensembles de successeurs disjoints.

Considérons les algorithmes vus en cours : le premier, A, avec contrôle de la diffusion du message, et le second, B, avec construction de l'arbre couvrant (mémorisation dans les variables Père et Fils).

⇒ **2.1.** *Montrer que la condition (ii) n'est pas vérifiée par l'algo A de diffusion si on regarde ce dernier comme un algo de mémorisation d'arbre couvrant, c-a-d :*
- *P_i reçoit son premier message, il lui est adressé depuis P_j , P_j devient son père.*
- *P_j envoie des messages à tous les processus dans y (cf algo B), ils deviennent ses successeurs.*

⇒ **2.2.** *Montrer que la variante de l'algo B vue en cours (qui consiste à mémoriser un arbre recouvrant) vérifie les deux conditions mais ne permet pas de détecter la terminaison.*

Dans la suite, on se placera dans l'hypothèse (H2).

2.2 Exploration parallèle

⇒ **2.3.** *Modifier l'algo B en utilisant une technique d'acquittement permettant, d'une part de spécifier à P_{init} que la construction est terminée, d'autre part d'assurer la condition (ii) : un processus P_i qui recoit deux messages d'exploration de P_j et P_k devra choisir son père et prévenir les deux de sa décision.*
Vous écrirez donc le code de P_i comprenant la déclaration
- *d'une constante $voisins_i$ définissant l'ensemble des voisins de P_i,*
- *des différentes variables nécessaires à la mémorisation de la structure arborescente*
- *et enfin qui prend en compte la réception de messages du type*
  - *(explorer).*
  - *(acquitter,x), où x est un booléen permettant de faire savoir au récepteur (P_j) si l'émetteur (P_i) de ce message accepte P_j comme père (0 refuse, 1 accepte).*

  sous forme événementielle.

De plus, on fera en sorte qu'un message acquitter(1) ne puisse être envoyé au père que si le site concerné a lui-même reçu tous les acquittements (positifs ou négatifs) des sites auxquel il a envoyé le message explorer (pour cela, il sera nécessaire de compter ces acquittements). Montrer que la construction est terminée lorsque P_{init} a reçu tous ses acquittements.

3 Retour d'information vers l'initiateur

L'algorithme vu ci-dessus peut être utilisé pour collecter des informations présentes dans les différents sites. on supposera donc que chaque site contient une information (par exemple, son identité), et que l'initiateur souhaite acquérir toutes ces informations (donc, dans notre exemple, apprendre les identités des sites).

⇒ **3.1.** *Que faut-il ajouter comme paramètre, et à quel type de message, pour atteindre cet objectif ? Modifier l'algorithme de la question précédente en conséquence.*

4 Exploration séquentielle

Le but est de construire un recouvrement du graphe en parcourant les sites un par un (d'où le séquentiel). Cela permet de mieux controler l'envoi de messages d'exploration, ainsi un site n'envoie jamais un message inutile, il n'y a pas de remise en cause des successeurs.
Cet algorithme nécessite toutefois que les sites aient des identités, 2 à 2 distinctes. L'hypothèse H2 est donc indispensable ici.
Le processus P_{init} lance la construction à l'un de ses voisins par l'envoi d'un message explorer (que nous détaillerons plus bas) et le mémorise comme l'un de ses successeurs.
Un site P_i qui recoit un message d'exploration pour la première fois, mémorise l'émetteur P_j comme son père, et agit selon les trois cas possibles suivants :
- cas i) P_i sait que tous les autres processus sont dans l'arborescence. Il informe alors les autres que l'algo est terminé par l'envoi du message terminé à son père.
- cas ii) P_i sait que des processus ne sont pas encore dans l'arborescence, mais l'ensemble de ses voisins (noté $voisins_i$) le sont. Il va alors spécifier à son père que le sous arbre de racine P_i est construit (backtracking) par l'envoi du message rebrousser.
- cas iii) P_i sait qu'il a des voisins non encore atteints. Il sélectionne l'un d'eux, le mémorise comme successeur et l'informe par l'envoi d'un message explorer.

Les messages :
- (explorer,z,s) où
  z désigne l'ensemble des identités des processus déjà atteints,
  s désigne l'ensemble des identités des voisins non atteints des processus déjà atteints. ie si ($x \in z$ et ($\exists y \in voisins_x : y \notin z$)) alors $y \in s$.
- (rebrousser,z,s) où z et s ont la même signification que précédemment.
  Ce message permet de faire du backtracking lorsque tous les voisins d'un site Pi ont été atteints (Pi ne

peut pas prolonger le parcours) mais qu'il existe des sites non encore atteints.
– (terminé) annonce la terminaison.
L'initialisation
P_{init} envoie à l'un de ses voisins P_j :
(explorer, init, $voisins_{init}$- {j})

Les mises à jour des informations de contrôle (z et s) :
Lorsque P_i reçoit un message depuis P_j il agit selon les trois cas déjà mentionnés.

⇒ **4.1.** *Décrivez, en utilisant des notations ensemblistes, les trois cas de figure. Vous utiliserez les ensembles voisinsi, z et s. Et selon le cas et le message reçu, effectuez la mise à jour des informations de contrôle (z et s).*

⇒ **4.2.** *En déduire le code d'un processus P_i sous forme événementielle.*

5 Complexité sur la diffusion

On cherche ici à analyser la complexité en nombre de messages des algos vus ci-dessus.

⇒ **5.1.** *Calculer le nombre de messages de la version séquentielle (section 4), en fonction du nombre de sites du réseau. Calculer également la taille des paramètres en nombre de bits pour cette version.*

⇒ **5.2.** *Calculer une borne supérieure du nombre de messages nécessaires dans le cas d'exploration parallèle (section 2.2), avec mémorisation de l'arborescence couvrante et signalisation de la terminaison. Cette borne donne-t-elle une indication précise pour évaluer le temps d'éxecution de l'algorithme ?*

6 Exemple d'utilisation

Un algorithme de parcours peut être utilisé comme base pour établir un résultat, une structure particulière, ... Ici, nous utiliserons un parcours séquentiel pour obtenir sur le réseau une structure d'anneau virtuel couvrant.

Le but est de mémoriser les différents passages des messages dans chaque site, pour pouvoir utiliser le parcours suivi comme anneau virtuel. Il est plus facile de prendre comme sens de l'anneau le sens inverse de celui du parcours.

Ainsi, à chaque passage du message (explorer ou rebrousser) dans un site, on mémorisera la provenance du message et sa destination ; plus exactement on associera la provenance à la destination, de manière à effectuer le chemin inverse après terminaison du parcours. De plus, lors du premier passage du message dans un site, on mémorisera la provenance du message comme successeur dans l'anneau.

⇒ **6.1.** *Ajouter les variables nécessaires pour mémoriser la structure d'anneau virtuel, puis écrire l'algorithme d'utilisation de l'anneau, par exemple pour une exclusion mutuelle.*