UJF - UFR-IMA - MST ESI2
ISTG - RICM2
Examen d'algorithmique distribuée
Durée 2h

Tous documents autorisés – Le barême est indicatif

18 avril 2001

I) Vague ECHO Il s'agit de proposer une variante de la vague pour réseau quelconque à liaisons bidirectionnelles (sans hypothèse FIFO). On rappelle à la suite la variante de l'algorithme ECHO proposée en cours/TD qui supporte à un instant donné :

- des vagues issues d'initiateurs quelconques
- une seule vague par initiateur. Seule la dernière vague d'un initiateur est conservée. Les autres s'éliminent par extinction.

Il y a N noeuds possibles. Tout noeud exécute le même algorithme :

```
// Wait[N] tableau d'ensembles de liaisons
// Up[N] tableau des liaisons de retour vers les initiateurs
// last[N] tableau des dates des dernières vagues
// NodeId indificateur du noeud
// HL horloge locale

état_initial ==>HL=0;  for(n=0;n<N;n++)last[n]=0, Wait[n]=L
loopALT
  decision_vague ==>  HL++ ;
      Wait[NodeId]=L;last[NodeId]=HL;
      forall l in  L l!<TOK,NodeId,last[NodeId] >

  forall l in  L l?<TOK, i, h> && L=={l}  ==> // noeud terminal
      l!< RPL, i, h>

  forall l in L l?<TOK, i, h> && L!={l}  ==>
      if(h>last[i])  // nlle vague
         last[i]=h; Up[i]=l; Wait[i]=L - {l};
         forall l in Wait[i] l!<TOK,i,h >
      elseif(h==last[i]) // boucle
         l!< RPL, i, h>

  forall l in  L l?<RPL, i, h>   ==>
      if(h==last[i])  // retour vague
         Wait[i]-={l}
         if (Wait[i]=={} && i== NodeId)   FIN_VAGUE(h)
         if (Wait[i]=={} && i!= NodeId)    up[i]!<RPL, i , h>
end loopALT
```

question 1 (4 points) Modifier l'algorithme précédent pour supporter plusieurs vagues à la fois issues d'un même initiateur. Ces vagues ne doivent donc plus être éliminées. Un des points clés de la solution est d'éviter de conserver sur les noeuds la description des vagues terminées[1].

```
// Wave[N]  table de tables associatives date --> Wait, Up
// similaire à la classe HashTable  de java....
//
// NodeId indificateur du noeud
// HL horloge locale

état_initial ==>HL=0;  for(n=0;n<N;n++)Wave[n]= new HashTable();
loopALT
  decision_vague ==>  HL++ ;
      Wave[NodeId].put(HL, <L,null>);
      forall l in  L l!<TOK,NodeId,HL >

  forall l in  L l?<TOK, i, h> && L=={l}  ==> // noeud terminal
      l!< RPL, i, h>

  forall l in L l?<TOK, i, h> && L!={l}  ==>
      if(!Wave[i].containsKey(h))  // nlle vague
          Wave[i].put(h, <L-{l},l>);
         forall lo in L-{l} lo!<TOK,i,h >
      else // boucle
         l!< RPL, i, h>

  forall l in  L l?<RPL, i, h>   ==>
        if(Wave[i].containsKey(h))   // retour vague
           <Wait,up>=Wave[i].get(h);
            Wait-=l;
            if (Wait=={})
                 Wave[i].remove(h);
                 if i== NodeId)   FIN_VAGUE(h)
                 else    up!<RPL, i , h>
end loopALT
```

question 2 (1 points) Votre algorithme supporte t il les liaisons non FIFO ? Pourquoi? (5 lignes maximum) **Oui. toutes les vagues sont enregistrées indépendamment de leur date d'arrivée et chemin d'arrivée sur un noeud. Elles sont éliminées à leur fin. Toute solution revenant à comparer 2 dates est au mieux FIFO et au pire fausse.**

II) Vague TORE 2D Il s'agit de généraliser le comportement des noeuds d'un anneau FIFO à une grille d'anneaux FIFO (cf figure 1).

On rappelle le comportement des noeuds d'un anneau FIFO :

```
// li liaison entrante et lo liaison sortante
loopALT
 //Partie routage
    décision_mess(d,data) ==>  lo!<MSG,d,data>
    li?<MSG,d,data> && d!=NodeId ==> lo!<MSG,d,data> // reémession
    li?<MSG, d,data> && d==NodeId ==>  traitement local (data)
```

[1] On se donnera les gestionnaires de données appropriés(table de hachage, etc..) en précisant simplement la sémantique des opérateurs utilisés (insertion, retrait, etc..).

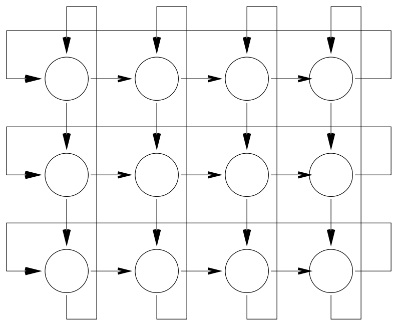

Figure 1: Exemple de tore 2D 4*3

```
  // Partie vague
    décision_vague ==> HL++; LAST =HL; lo!<TOK, NodeId, LAST>;
    li?<TOK, i, h> && i!==NodeId ==>  lo!<TOK,i,h,>
    li?<TOK, i, h> && i==NodeId ==> FIN_VAGUE(h)
end loopALT
```

Adaptez l'algorithme précédent de façon à ce qu'il fonctionne sur un tore 2D. On notera Li et Lo les deux tableaux de liaisons. L'indice 0 repère la ligne et l'indice 1 la colonne. Le problème comprend 2 parties indépendantes :

question 3 (2 points) modifier la partie routage pour router dans un tore 2D. On supposera disposer d'une fonction *Coord(int)->int,int* qui fournit à partir de l'identité d'un noeud ses coordonnées $< x, y >$ dans la grille. **on route en colonne jusqu'a la bonne puis en ligne ..**

```
int ln(int n){int x,y;<x,y>=Coord(n); return x;}// ligne
int col(int n){int x,y;<x,y>=Coord(n); return y;}// colonne

 //Partie routage

décision_mess(d,data) ==>
    if(col(d)==col(NodeId)
         Lo[0]?<MSG,d,data> //emission ligne
    else
         Lo[1]?<MSG,d,data>; //emission colonne
forall li in Li li?<MSG, d,data> && d!=NodeId ==>
    if(col(d)==col(NodeId)
         Lo[0]?<MSG,d,data> //emission ligne
    else
         Lo[1]?<MSG,d,data>; //emission colonne
forall li in Li li?<MSG, d,data> && d==NodeId ==>  traitement local (data)
```

question 4 (4 points) modifier la partie vague pour réaliser une vague sur tore 2D. On pourra introduire différents types de TOK si nécessaire. **on fait successivement tous les anneaux lignes en parcourant la colonne de l'initiateur**

```
   // Partie vague
décision_vague ==> // on débute par la ligne
   HL++; LAST =HL; Lo[0]!<TOK, NodeId, LAST>;// 1ere ligne
Li[0]?<TOK, i, h> ==>
```

```
        if(col(i)==col(NodeId)
             Lo[1]?<TOK,i,h> //fin  ligne colonne suivante
        else
             Lo[0]?<TOK,i,h>; //suite ligne
    Li[1]?<TOK, i, h> && i!==NodeId ==> // on propage sur ligne
        Lo[0]!<TOK,i,h,>
    Li[1]?<TOK, i, h> && i==NodeId ==> FIN_VAGUE(h)
```

III) Vague PHASE Le principe de la vague PHASE pour graphe quelconque à liaisons monodirectionnelles est rappelée à la suite[2]. Le diamètre du réseau est supposé connu.

```
Li (Lo)  ensemble des liaisons entrantes (sortantes) d'un noeud
CTi[Li] tableau de nombres de message reçus par liaison entrante
CTo nombre des phases d'émission

état_initial ==> CTo=0;  forall l in Lin CTi[l]=0
loopALT
  decision_vague ==>
      CTo++
      forall l in Lo l!<TOK > // première phase

  forall l in Li l?<TOK> && CTo<DiamètreRéseau ==>
      CTi[l]++
      if(minimumDes(CTi)>=CTo)  // nouvelle phase d'émission
         CTo++
         forall l in Lo l!<TOK >

  forall l in Li l?<TOK> && Cto>=DiamètreRéseau ==>
        if(minimumDes(CTi)>=CTo)  FIN_VAGUE
end loopALT
```

L'algorithme est basé sur un mécanisme de répétition de phase. Une phase (sauf la phase initiale) est la diffusion sur les liens de sortie de ce qui a été reçu sur tous les liens entrants à la phase précédente.

question 5 (3 points) Modifier l'algorithme pour le rendre capable de supporter plusieurs initiateurs ainsi qu'une vague par initiateur selon la méthode d'extinction des vagues plus anciennes. **on a besoin de tableaux Cto[N] et Cti [N] pour conserver les compteurs des N noeuds et un tableau Last[N] pour la date de derniere vague**

```
N nombre de noeuds
Li (Lo)  ensemble des liaisons entrantes (sortantes) d'un noeud
CTi[N,Li] tableau de nombres de message reçus
        par liaison entrante par noeud initiateur
CTo[N] nombre des phases d'émission  par initiateur
last[N] derniere vague vue par initiateur
HL

état_initial ==> HL=0;for(n=0;n<N;n++)last[n]=0;


loopALT
decision_vague ==>
      // initalisation compteurs
```

[2]Il ne supporte qu'une vague dans le réseau.

```
        HL++; last[NodeId]=HL;CTo[NodeId]=0;
        forall l in Lin CTi[NodeId,l]=0
        CTo[NodeId]++       // première phase
        forall l in Lo l!<TOK,NodeId,last[NodeId] >

forall l in Li l?<TOK,i,h> ==>
   if(last[i]<h)// nouvelle vague : on reinitialise
       CTo[i]=0;
       forall l in Lin CTi[i,l]=0
       last[i]=h
   // on traite la nouvelle vague ou continue celle en cours
   if(last[i]==h)&& CTo[i]<DiamètreRéseau
        CTi[i,l]++
        if(minimumDes(CTi[i])>=CTo[i])  // nouvelle phase d'émission
          CTo[i]++
          forall l in Lo l!<TOK,i,h >
   if(last[i]==h)&& && CTo[i]>=DiamètreRéseau
        FIN_VAGUE(i,h)
   // h<last[i] vieille vague : on oublie....

end loopALT
```

Dans un algo de vague, un noeud répèe le token au moins une fois sur ses sorties. Ceci garantit que le token atteint tous les noeuds du graphe en D étapes (D diamètre). Le problème dans un réeau monodirectionnel est simplement de déterminer au bout de combien de réception il a acquis toutes les réonses. Les réponses lui reviennent des cycles du réseau qui sont nécessaires pour permettre la communication entre toutes les paires de noeuds du réseau

question 6 (2 points) Pourquoi l'algorithme consistant à reémettre sur les liaisons de sortie chaque message arrivant sur une liaison entrante ne marche-t-il pas ? **Du fait des cycles et des délais de propagation inconnus, l'initiateur (ou tout noeud relai) ne peut savoir au bout de combien de réceptions il a reçu toutes les infos émises des noeuds de ses sous réseaux. On ne peut donc décider d'un critère d'arrêt. On ne peut qu'engorger le réseau en espérant avoir tout acquis au bout d'un temps correspondant aux délais de propagation les plus longs.**

question 7 (2 points) Pourquoi l'algorithme de phase ne marche-t-il pas avec une seule répétition[3]? **Si on émet et propage une seule fois le jeton, celui-ci revient à l'initiateur sur les entées via les cycles auquels il appartientdirectement. Dans la figure 2, il obtient les réponses de 1 et 2 mais pas de 3 et 4 qui sont seulement acquises par le noeud 0 qui ne les a pas répétées.**

question 8 (2 points) Pourquoi le nombre de phases nécessaire est il la valeur du diamètre du réseau ?**Il faut donc répéter par phases successives pour extraire les données des cycles auxquels l'initiateur n'appartient pas directement. Une répétition va extraire les données acquises par 0 en provenance du cycle auquel 3 et 4 appartiennent. Une autre remontera les acquis du cycle du noeud 4. En R phases, on extrait les réponses des cycles distants de R de l'initiateur. Comme cette distance ne peut être supérieure au diamètre D du réseau(cf. figure 2), le nombre de répétition est D**

[3] équivalent à un diamètre de 1.

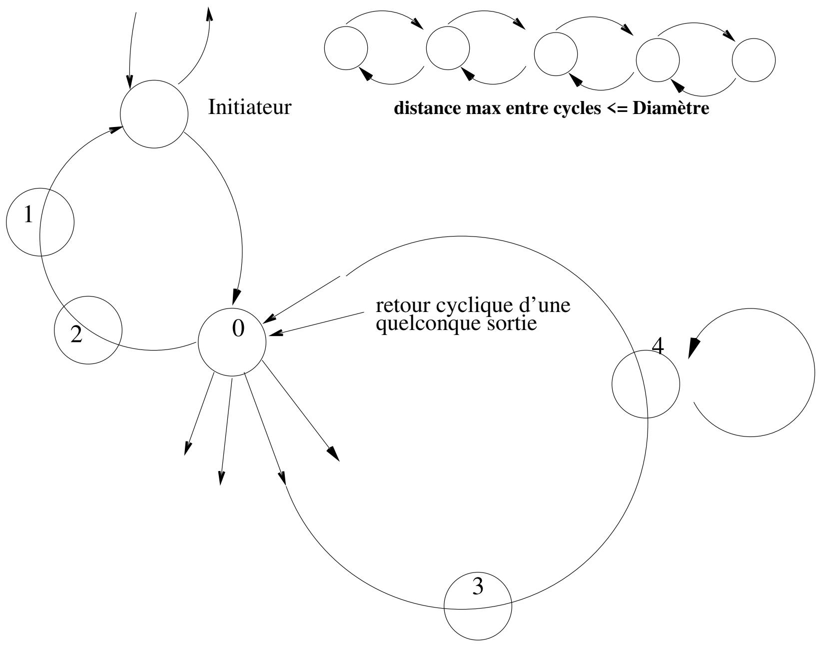

Figure 2: Exemple de sous-réseau monodirectionnel