Algorithmique et programmation distribuée

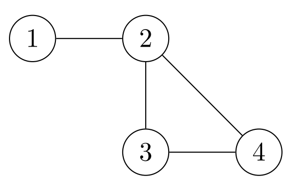
Zakaria EJJED

Contents

Wednesday February 1st 2023

Un graphe: G(V,E) $\rightarrow$ (ensemble sommet, ensembles arêtes)

V=1,2,3,4 | E=[(1,2),(2,3),(3,4)(2,4)}



degré d'un sommet x $\in$ V

d(x)=nombre de ses voisins dans le graphe

exemple: d(2)=3

soit un graphe à n sommets

G=(V,E), |V|=4 (ordre du graphe), max:n-1 min:0

Exercice: Modelisation d'un problème

Montrer que dans un groupe de personnes (***n noeuds***), il y a toujours 2 personnes qui connaissent (***arêtes***) le même nombre de membres d'un groupe.

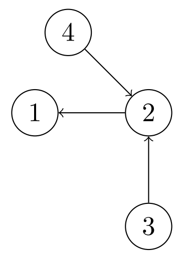
Par l'absurde, opposons que les n noeuds ont tous un degré différent

$\rightarrow$ contradiction: un noeuds doit avoir un degré n-1

$\rightarrow$ Un noeud doit avoir un degré 0 ou il y a n noeuds et n degrés différents possibles or ces 2 noeuds sont pas connecté.

Graphe orienté: Un graphe dans lequel les arêtes (*arc*) ont une direction.

G=(V,A) $\rightarrow$ (sommets,arcs)



V=1,2,3,4 | A =(2,1) (3,2) (4,2) $\rightarrow$ (*origine*,*extremité*)

Pour le somemt x

degré entrant: nombre d'arcs dans lequel x extremité

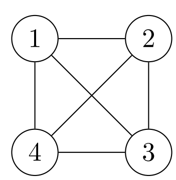
degré sortant: nombre d'arcs dans lequel x origine

$d^+(x)$ | $d^-(x)$

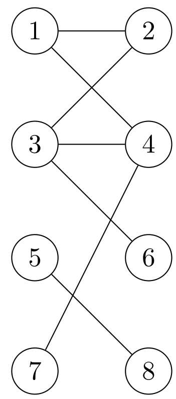
Exemple de graphes:

- ***graphe complet***: toutes les arêtes sont presentes

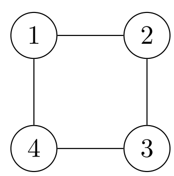
$$\text{nombres d'arête } = \frac{n(n-1}{2}$$



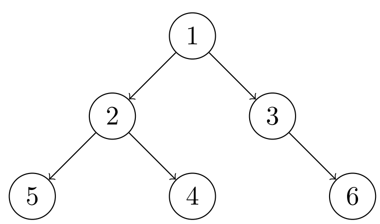
- ***graphe biparti***:



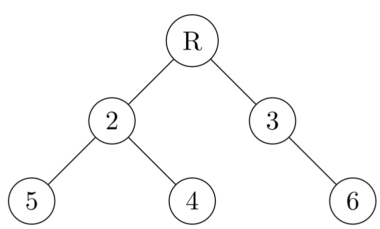
- ***Arbres***: graphe qui n'a pas de cycles.
  cycle:



  Nombre d'arêtes dans un arbre, sans sommet de degré 0: n-1



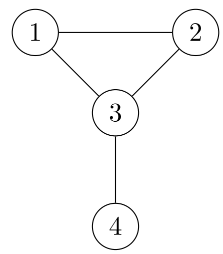
- ***arbres enracinés***: Arbres don orienté dans lequel on distingue un noeud racine.



Feuille d'un arbre: sommets de degré 1 qui n'est pas la racine.

Chemin: ensemble de sommets $x_1, x_2, x_3, ..., x_{n-1}, x_n$

tel que $(x_1, x_2, x_3, ..., x_{n-1}, x_n)$ sont des arêtes



longueur chemin: nombre de ses arêtes

père d'un noeud x: soit h(x) sa hauteur, Son père est son voisin dont la hauteur vaut h(x)-1

fils d'un noeuf: comme le père mais h(x)+1

mini exo:

Soit un arbre à n noeuds
→hauteur max ⇒ n-1 (arbre chemin)
→hauteur min ⇒ 1 (n feuilles)
→hauteur quand un noeud a exactement 2 fils ⇒ $2^{h-1} \leq n \leq 2^h$

Soit un graphe avec n sommets, $s_1, s_2, ..., s_n$

Montrer que $\sum_{i=1}^{n} d(s_i)$ est paire.

Dans la $\sum$ des degrés: chaque arête est compté 2 fois, une fois pour chaque extremité.

Exercice: Montrer que le nombre de sommets de degré impaire est paire.

On sait que la somme des degrés est paire, donc pour chaque sommet de degré impaire, il doit y avoir un second sommet de degrés impaire.

Dans le cas où on aurait un nombre impaire de sommet de degré impaire, la somme des degrés serait elle aussi impaire.

Montrer que dans un arbre avec + de 2 sommet, il y a au moins deux sommets de degré 1.

→ En supposant n ≥ 2 et un noeud de degré 1.

$$\sum_{i=1}^{n} d(s_i) \geq 2 \times (n-1) + 1$$

- ***Complexité - notion :***
  Notation de ***Landeau***: $O(f(x)) = g(x)$
- ***informel:***
  à partir d'un certain x, la valeur $g(x)$ sera inférieur à $f(x)$

//COURBE

- ***formellement:***
  on écrit

$$\begin{array}{rcl} f(x) & = & O(g(x)) \\ f(x) & \in & O(g(x)) \end{array}$$

$$\text{Ssi: } \forall x \geq x_0, \exists k \in N \text{ tel que } |f(x)| \leq k|g(x)|$$

Exemple:

$$\begin{array}{rcl} f(x) & = & x \\ g(x) & = & x^x \end{array}$$

//COURBE 4

$$x_0 = 1 \quad k = 1 \quad f(x) = O(g(x))$$

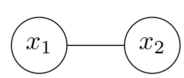
Lors d'une compétition il y a 13 joueurs, est-ce qu'il est possible que chaque joueur participe à exactement 3 matchs.

- ***système distribué:*** un ensemble de noeuds de calculs, autonomes interconnecté et pouvant communiquer
- ***représenté par un graphe:***

$$1 \text{ noeud de calcul} = 1 \text{ sommet}$$
$$1 \text{ lien de com} = 1 \text{ arête}$$

- 1 noeud n'a accès en lecture/écriture qu'à sa propre mémoire. Tout le reste lui est envoyé sous forme de message. Un noeud a une vision locale du réseau. Il faut souvent résoudre des problèmes globaux.
- ***objectif:*** résoudre problèmes globaux à l'aide d'algos locaux
  $\rightarrow$ Les noeuds vont devoir communiquer entre eux par passage de message. $\hookrightarrow$ 1 noeud peut recevoir 1 message (on sait qui nous l'a envoyé). $\hookrightarrow$ 1 noeud peut envoyer 1 message à son voisin/

mini exemple:



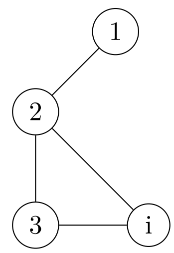
On veut un algo: à la fin de l'execution chaque noeud connait la valeur var max dans le réseau.

chaque noeud envoie son var à son voisin

calcul: sur réception de var faire le calcul de max (var x_1, var x_2)

Wednesday February 8th 2023

- **2 types d'évenements exterieur:**
  a.
  -évènement initial → bout de code
  -il est executé initialement sur un ensemble non vide, le noeud du système
  -1 seule fois
  -il ne peut pas être executé sur 1 noeud qui a déjà fait du code
  b.
  -réception



Algo: pour le noeud i
1. * description des variables
2. * event initial: (des regles gardés)
* sur réception des messages: prends code classique

- **Les types de variables:**

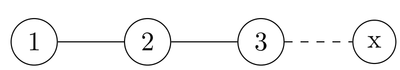
- locals: on les indices par l'identifiant du site|noeud i

- *hookrightarrow* connaissance: elles ne sont accessibles qu'en lecture, elles servent à décider le système
  par ex: voisin i, l'ensemble des voisin du sommet

- ↪ variables locales au site pour les calculs. Elles sont accessible en lecture/ecriture.

- variables de communication
  B -
  -Elle contient une info qui est transmise
  +info du site emetteur
  -elles ne sont pas indices
  -elles sont ephemere.

Calcul des max dans une chaine



- **Algo pour i:**
- connaissance:
  voisin i: ensemble des voisins de i dans la chaine
  valeur i: valeur entiere du noeud i
- variables:
  max i : le maximum dourant calculé sur i

Initialement (description de l'evenement initial) *sur tous les noeuds*
envoyer Msg(val i) à voisin i

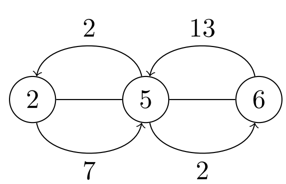
Algorithm 1 INIT

envoyer Msg(val_i) à vois_i

Algorithm 2 Sur réception de Msg(v) ($\leftarrow$ données) de j ($\leftarrow$ noeud émetteur)

if $\text{max}_i < v_i$ **then**
  $max_i \leftarrow v_i$
else if $|vois_i| = 2$ **then**
  envoyer Msg(v) à vois_i sans {j}
end if

Init:



val_2=7; val_5= 2; val_6=13

max_2=7; max_5=13; max_6=13

$$\begin{array}{ll} 2 & \rightarrow 5 \\ 5 & \rightarrow 2 \rightarrow 6 \\ 6 & \rightarrow 5 \end{array}$$

O(n^2) mémoire O(n+m) (m étant le nbres d'arêtes)

Complexité

Configuration + Execution + complexité en temps

A B C

A - Configuration

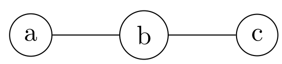
“screenshot” de votre système à 1 instant donné

	$\rightarrow$ états des variables de tous les sites
$+$	$\rightarrow$ exo des messages en transit
$=$	état local de tous les sites et les messages en transit

B - Execution

- Dans 1 config donnée, 1 site est activable si il existe 1 evenement exterieur qui a été déclenché et qui est en attente de traitement

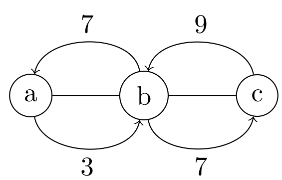
- Execution: une sequence C_1, C_2,...,C_i,C_{i+1},...,C_n
  telle que entre 2 config consecutives:
- tous les sites activables ont executé les actions associées aux event exterieur (modifs des états locaux)
- les autres ne font rien



val_a=3; val_b= 7; val_c=9

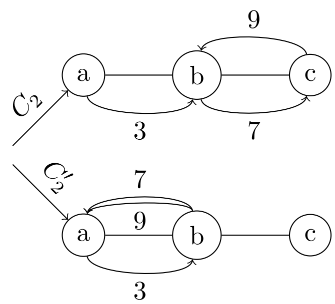
max_a=3; max_b=7; max_c=9

$\Downarrow$



le problème qui apparait est que tous les messages ne circulent pas à la meme vitesse. Par conséquent, ils n’arrivent pas en meme temps, ce qui crée de l’indeterminisme.

La complexité en tmps est donc le tmps de la plus longue exectution possible.



C_2 : Si msg(7) de b est reçu par a

C_2' : Si msg(7) de b est reçu par c, msg(9) reçu par b

→ On observe qu'il existe plusieur execution possible pour 1 meme algo/reseau. Cela est du à un non determinisme provoqué par le tps de transit des msg qui est variable.

cas de figure: (execution asynchrone)

La complexité en tps va etre la longueure de la plus longue execution possible, parmis ttes les executions.

b - hypothèse synchrone:

- l'ensemble des msgs en transit est réceptionné en 1 unité de tps
- Entre 2 config, on a 1 round
- → tous les msg sont récéptionnés + toutes les règles gardées associées …

↓

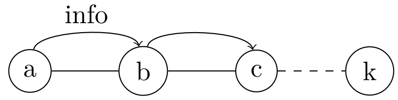
le round se termine et on est dans 1 round config.

Dans cette hypothese: la complexité en tps = nbr de noeuds (il n'y a plus qu'une exec possible)

Terminaison des algorithmes

1 algo termine quand dans toutes les exec il existe une config contenant aucun msd en transit et dans lequel aucun site n'a de regle gardées à vrai

- terminaison explicite: Il existe au moins 1 site qui sont que l'algo se termine ↪ c'est indiqué stop-globaldans le code.
- terminaison locale: 1 site sait qu'il a terminé sans que la terminaison soit associée pr le reste des noeuds → dénoté dans le code par stop-local



racine feuille

(A)

Algo pour 1 noeud i

Connaissance (lecture)

- vois_i: les voisins de i dans la chaine
- estRacine_i: bool (=1 pour noeud racine, 0 sinon)
- val_i: info à diffuser → null pour tout les noeuds sauf la racine

Variables * Save_i: sert à stocker l'info à diffuser. Vaut ndef initialement.

Algorithm 3 INIT

```
if estRacine_i == 1 then
    envoyer Msg(info_i) à vois_i
    save_i ← info_i
end if
```

Algorithm 4 Sur reception de message Msg(v) de j

```
if vois_i == 1 then                                  ▷ (A)
    save_i ← v
    stop-global
end if
if vois_i == 2 then
    save_i ← v
    envoyer Msg(v) à vois_i sans {j}
    stop-global
end if
```

complexité en msg: n-1

en temps ..

- synchrone: n-1
- asynchrone: n-1

Exercie 2

1.

compléxité: O(2n) → O(n)

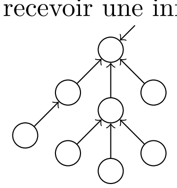
Algorithm 5 Sur reception de message Msg(v) de j)

```
if vois_i == 1 then                                    ▷ (A)
    save_i ← v
    envoyer Retour(u) à j
    stop-global
end if
if vois_i == 2 then
    save_i ← v
    envoyer Msg(v) à vois_i sans {j}
    stop-global
end if
```

Algorithm 6 Sur reception de Retour(v) de j)

```
if estRacine_i == 1 then
    stop-global
end if
if vois_i == 2 then
    envoyer Msg(v) à vois_i sans {j}
    stop-global
end if
```

la racine doit recevoir une info de chacune des feuilles



Connaissance:

père_i le père du noeud_i. Vaut null pour la racine

fils_i les fils du noeud_i: vaut ndef pour les feuilles

info_i: l'information à diffuser: ndef sauf pour les noeuds

Variables:

cpt_i: compter le nombre de messages reçus

save_i: init à null, sert à sauvegarder l'info

Algorithm 7 INIT

```
if fils_i == 0 then
    envoyer Msg(info_i) à père_i
    stop-local
end if
```

Algorithm 8 Sur réception de Msg(v) de j)

cpt_i++
if cpt_i ==|fils_i| **then**
  envoyer Msg(info_i) à père_i
  $\text{save}_i \leftarrow v$
  if $\text{père}_i \neq \textit{null}$ **then**
    envoyer Msg(v) à père_i ▷ top-local
  else
    stop-global
  end if
end if

compléxité en message: n-1 : 1 message par canal de comm

n noeuds ⇒ n-1 arêtes

en temps: hauteur de l'arbre

2.

Connaissance:

estRacine_i

vois_i voisins du sommet

info_i: l'information à diffuser: ndef sauf pour les noeuds

Variables:

cpt_i: compter le nombre de messages reçus

save_i: init à null, sert à sauvegarder l'info

Algorithm 9 INIT

if vois_i == 1 **then**
  envoyer Msg(v) à vois_i
  $\text{père}_i \leftarrow \text{vois}_i$
  $\text{fils}_i \leftarrow \textit{null}$
end if

Exercice 4

Connaissance:

vois_i

estRacine_i

info_i

Algorithm 10 Sur réception de Msg(v) de j)

cpt_i++
ajouter j dans fils i
if $\text{estRacine}_i == 0$ **then**
  if $|\text{vois}_i| - cpt_i == 1$ **then**
    enoie msg(v) à vois_i sans fils_i
    père ← vois_i sans fils_i
    stop-local
  end if
else
  if $\text{père}_i == null$ AND cpt == $|\text{vois}_i|$ **then**
    stop-global
  end if
end if

Variables:

$\text{père}_i = NULL$

$\text{fils}_i = []$

save_i: init à null, sert à sauvegarder l'info

Algorithm 11 INIT

if $\text{vois}_i == 1$ **then**
  envoyer Msg(v) à vois_i
  $\text{père}_i \leftarrow \text{vois}_i$
  $\text{fils}_i \leftarrow null$
end if

Algorithm 12 Sur réception de Msg(v) de j)

```
cpt_i++
ajouter j dans fils i
if estRacine_i == 0 then
    if |vois_i| - cpt_i == 1 then
        enoie msg(v) à vois_i sans fils_i
        père ← vois_i sans fils_i
        stop-local
    end if
else
    if père_i == null AND cpt == |vois_i| then
        stop-global
    end if
end if
```