I. Cours

1. Rappeler la définition de l'excentricité

Nous appelons excentricité d'un sommet s la plus grande distance entre le sommet s et les autres sommets.

2. Ecrire un algorithme à vague fonctionnant sur un arbre

(Algorithme page 27 du cours)

Code initiateur ou Racine :

```
pour tout j dans Vois faire :
      envoi jeton à j
fin pour
pour tout j dans Vois faire :
      recevoir jeton de j
fin pour
prise de décision
```

Code non initiateur :

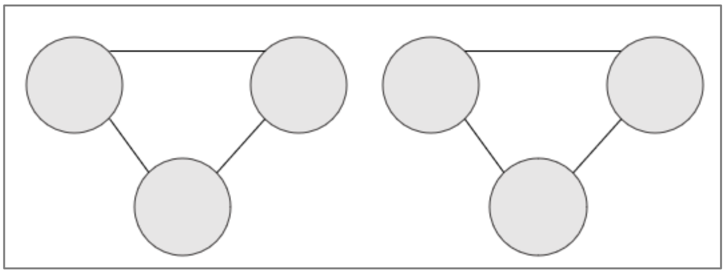
```
reçoit jeton de j0
pour tout j dans Vois sans j0 faire :
      envoi jeton à j
fin pour
pour tout j dans Vois sans j0 faire :
      recevoir jeton de j
fin pour
envoi jeton à j0
```

3. Dessiner un graphe régulier de degré 2 qui ne soit pas un anneau

Rappel :

- *Régulier :* un graphe est dit régulier si tous ses sommets ont le même degré
- *Degré :* le nombre de voisins que possède le sommet
- *Anneau :* graphe <u>non orienté</u>, <u>connexe</u>, et <u>régulier</u> de <u>degré 2</u>
- *Connexe :* un graphe est connexe s'il existe au moins un chemin menant de tout sommet à un autre sommet
- *Non orienté :* il n'y a pas de relations de successeurs et de prédécesseurs.

Il faut donc que notre graphe ne soit pas connexe, et/ou orienté. Un graphe non connexe est facilement imaginable :

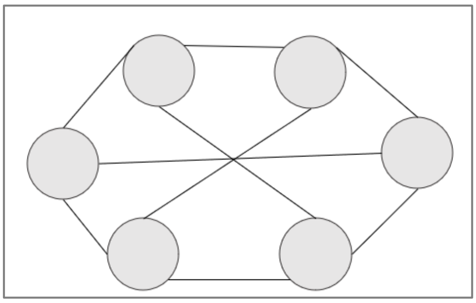


*Graphe de degré 2, régulier et **non connexe***

4. Dessiner un graphe régulier de degrés 3 de 6 sommets

Rappel :

- *Régulier :* un graphe est dit régulier si tous ses sommets ont le même degré



Degré 3, 6 sommets

II. Dérouler un algorithme : algorithme à vagues

```
Algorithme 1 Election par Vagues, Algo local au sit
  Variables
  utilise[j]( avec j ∈ vois) init. à faux
  varp init. à ⊥
  elu init. à 0

  Code initiateur
  if  varp =⊥ then
    varp ⟵ i
    elu ⟵ i
    Choix j dans vois
    utilise[j] ⟵ vrai
    Envoi msg(elect, elu)
  end if

  Code tout site
  Reception de msg(elect, val) de j0
  if elu < val then
    elu ⟵ val
    varp ⟵ j0
    for all j ∈ vois do
      utilise[j] ⟵ faux
    end for
    utilise[j0] ⟵ vrai
    if ∃k ∈ vois\varp|¬utilise[k] then
      choisir k ∈ vois\varp ∧ ¬utilise[k]
      utilise[k] ⟵ vrai
      Envoi msg(elect, elu) à k
    end if
  else
    if elu = val then
      Envoi de msg(dejavu) à j0
    end if
  end if
  Reception de msg(pere) ou msg(dejavu) de j
  if ∃k ∈ vois|¬utilise[k] then
    utilise[k] ⟵ vrai
    Envoi de msg(elect, elu) à k
  else
    if varp ≠ i then
      Envoi msg(pere) à varp
    end if
  end if
```

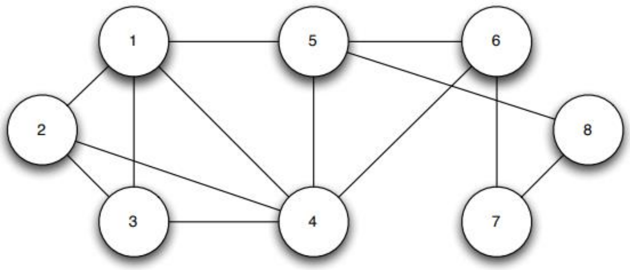


FIGURE 1 – Réseau

1. Ecrire un algorithme à vagues fonctionnant sur l'anneau

(Algorithme page 26 du cours)

Code initiateur :

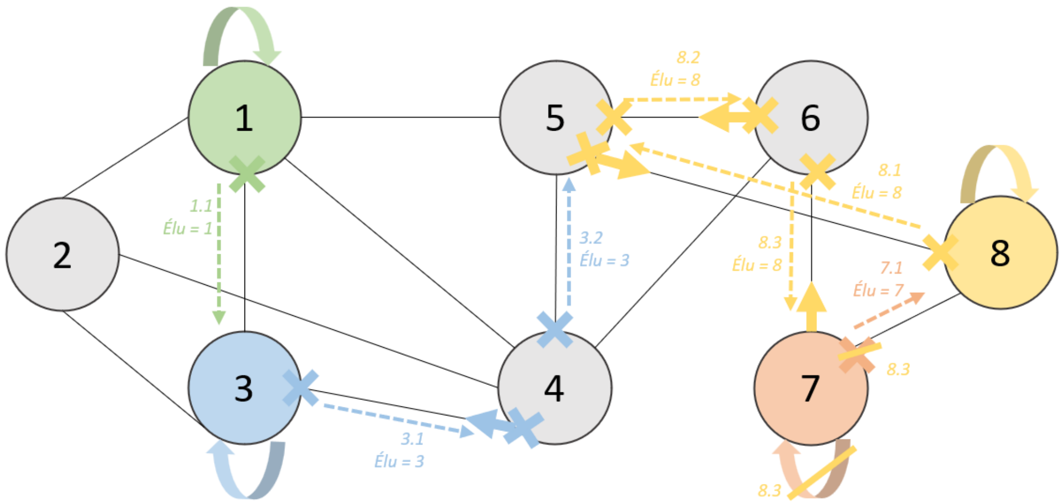
```
envoi jeton à voisin de droite
recoit jeton de voisin de gauche
prise de décision
```

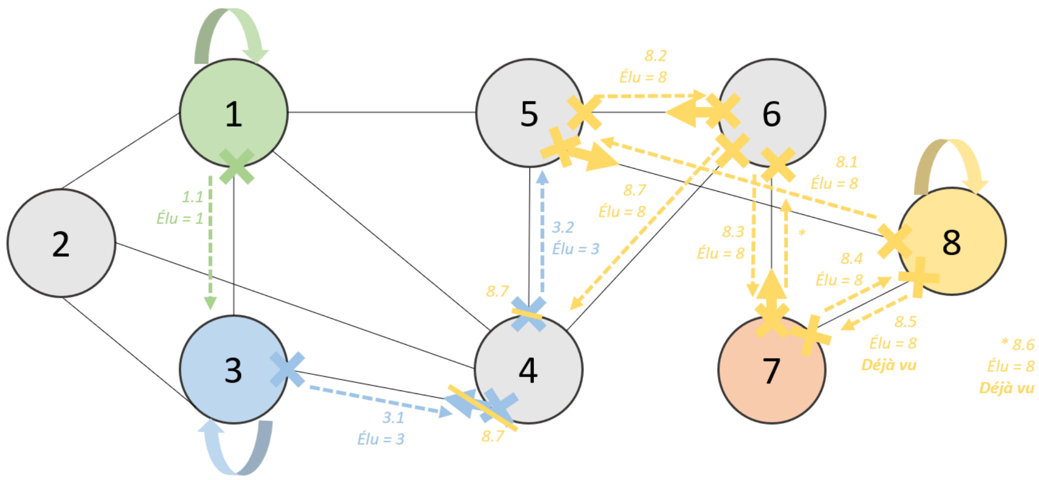
Code non initiateur :

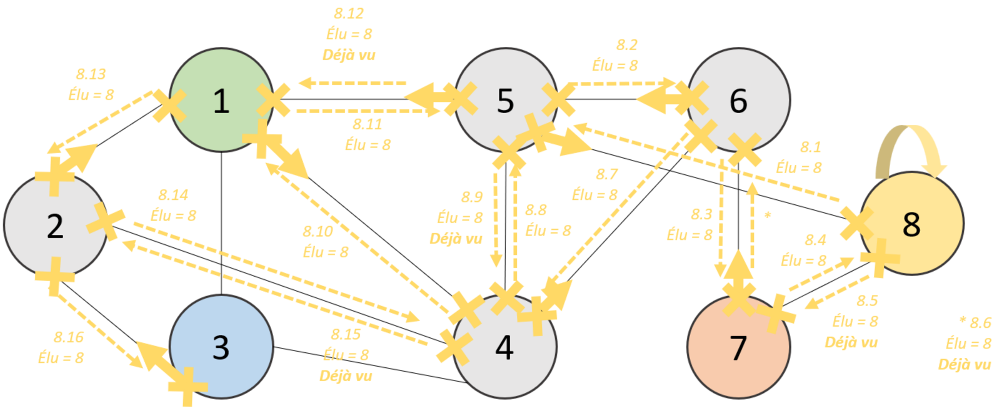
```
recoit jeton de voisin de gauche
envoi jeton à voisin de droite
```

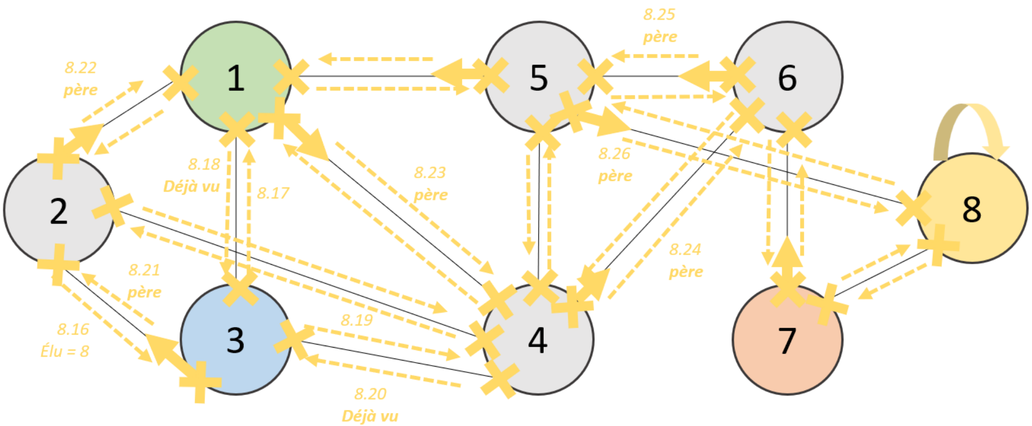
2. Dérouler l'algorithme 1 sur la figure 1 en considérant les nœuds 1,3,7 et 8 comme initiateurs (utiliser des schémas)

On remarque que c'est algorithme est l'algorithme « Election DFS » page 36 du cours. Pour un autre exemple et plus d'explications cf. un autre déroulement de cet algorithme.









3. Donner la complexité en temps et en message de cet algorithme en fonction du nombre d'initiateurs

La complexité de cet algorithme est 2cm messages, avec c le nombre de candidats. Donc la complexité est de 16.

III. Imaginer un algorithme : algorithme de parcours

On définit un hypercube de dimension n par les $2n$ points dans un repère orthonormal E ayant des coordonnées égales à 0 ou 1 reliés par des segments de droite (les arêtes). Les hypercubes sont les figures obtenues à partir de l'hypercube unité par des similitudes.

Soit l'hypercube donné figure 2. Cet hypercube est de dimension 3.

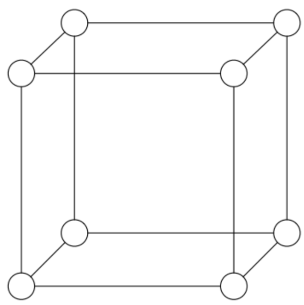
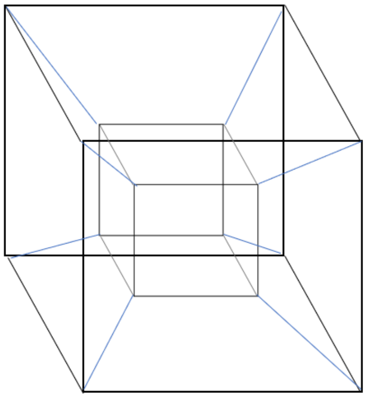


FIGURE 2 – Hypercube de dimensions 3

1. Dessiner un hypercube de dimension 4

Attention, erreur dans l'énoncé ! C'est 2^n points et non 2n points.



2. Inventer une numérotation des sites en fonction de la dimension de l'hypercube

Des meilleures/autres solutions existent sûrement

On remarque qu'un site dans une démission n aura n voisins (avec n > 1). On pourrait donc numéroter les sites dans l'ordre. C'est à dire en partant d'un site et en suivant ses arrêtes.

Variables :

`valeur` : entier qui est le numéro du sommet, initialisé à 0
`dim` : entier qui représente la dimension
`Vois` : qui sont les voisins du sommet
`taille_tab` : entier initialisé avec la valeur 2^{dim}
`tab_valeurs` : tableau de booléens de taille `taille_tab` initialisé entièrement à faux. Chaque case représente une valeur à attribuer à un site.

Code initiateur :

```
valeur = 1
si dim != 0 :
       tab_valeurs[0]=true
       envoi message à un j dans Vois avec comme données tab_valeurs
sinon :
       fin
fin si
```

Code non initiateur :

```
reçoit message de j0
si tab_valeurs[taille_tab-1]==false :
       si valeurs !=0
              pour i allant de 0 à (taille_tab-1) faire :
                     si tab_valeurs[i]==false :
                            valeur = i+1
                            tab_valeurs[i]=true
                            envoi message à un j dans Vois\j0 avec comme données
tab_valeurs
                     fin si
              fin pour
       sinon :
              envoi message à un j dans Vois\j0 comme données tab_valeurs
```

```
        fin si
sinon :
        fin
fin si
```

3. Déterminer un parcours en fonction de cette numérotation

Si on souhaite parcourir les sites dans l'ordre de la numérotation attribuée :

Variables :

`valeur` : entier qui représente le numéro du site
`dim` : entier qui représente la dimension
`visite` : booléen initialisé à faux
`Vois` : qui sont les voisins du sommet

Code initiateur :

```
visite=true
si dim != 0 :
        envoi message à un j dans Vois avec comme données valeur
sinon :
        fin
fin si
```

Code non initiateur :

```
reçoit message de j0 valeur_j0
si valeur==(valeur_j0+1) :
        visite=true
        si valeur==(taille_tab-1) :
                fin //tous les sites ont été parcouru
        sinon :
                envoi message à un j dans Vois\j0 avec comme données valeur
sinon :
        envoi message à j0
fin si
```

IV. Comprendre un algorithme : exclusion mutuelle

Algorithme 2 Alorithme de Ricart Agrawala, local au site i

Variables
$etat \longleftarrow S$ E, SC, S éttat du site i
$h \longleftarrow 0$ entier date des demandes
$last \longleftarrow 0$ entier date de la derniere demande
$attendu \longleftarrow \phi$ ensemble de sites qui attendent la permission
$differe \longleftarrow \phi$ ensemble de sites qui retardent l'envoi d'une perm
$priorit \longleftarrow faux$ booleen si i prioritaire ou non

demande d'entree en section critique

$etat \longleftarrow E$
$last \longleftarrow h+1$
$attendu \longleftarrow R$
for all $j \in attendu$ **do**
  Envoi $msg(dem, (lasti, i))$ à j
end for
while $attendu \neq \phi$ **do**
  reception $msg(perm(j))$ de j
  $attendu \longleftarrow attendu \backslash \{j\}$
end while
$etati \longleftarrow SC$

Sortie de section critique

$etat \longleftarrow S$
for all $j \in differe$ **do**
  Envoi $msg(perm(i))$ à j
end for
$differe \longleftarrow \phi$

Réception de $msg(dem, (h', j))$ **de** j

$h \longleftarrow \max(h, h')$
$priorit \longleftarrow (etat \neq sortie) \wedge (last, i) < (h', j)$
if $priorit = vrai$ **then**
  $differe \longleftarrow differe \cup j$
else
  Envoi $msg(perm(i))$ à j
end if

1. Expliquer comment fonctionne cet algorithme et donner sa complexité

Pour rentrer en section critique, tout site doit obtenir la permission de chacun d'entre eux.La complexité en message pour une demande de section critique est de 2n-2.

2. Montrer la propriété de sureté de cet algorithme

La propriété de sûreté est assurée car si deux sites sont en section critique, cela signifierait que les deux sites se sont accordés leurs permissions mutuelles, ce qui est exclu.

3. Montrer la propriété de vivacité de cet algorithme

La propriété de vivacité est aussi respectée car une estampille finira toujours par devenir la plus ancienne et donc la demande d'entrée en section critique sera toujours permise.