Benakli William 21960601
Le Franc Matthieu 71800858
Andrawos Saad David 21955132

Algorithmique répartie

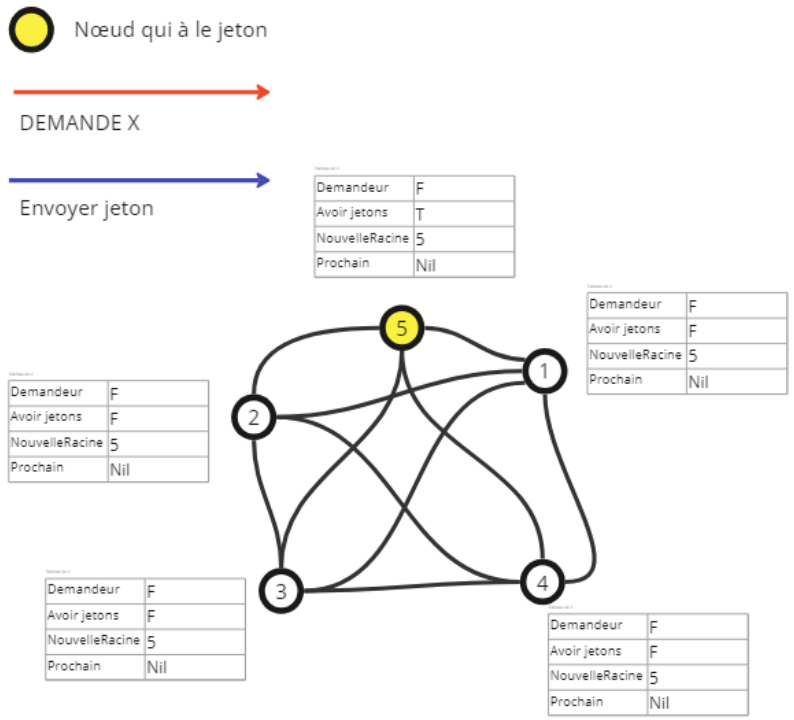
Devoir n°1

Exercice 1:

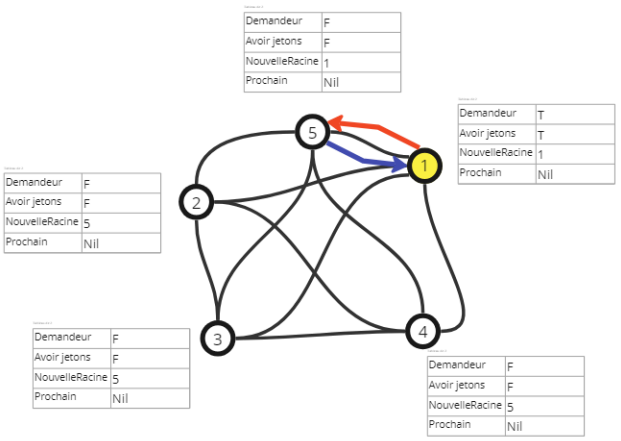
Question 1/. Donner un exemple d'exécution de cet algorithme sur un graphe de taille 5. Le jeton est sur le noeud 5. Ordre des demandes des nœuds 1,3,4.

AJ : Ajout Jeton
NR : Nouvelle Racine
P : Prochain
D : Demandeur

[INITIALISATION]

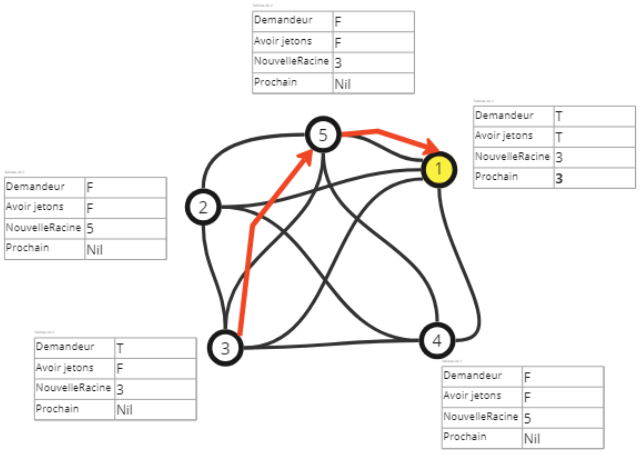


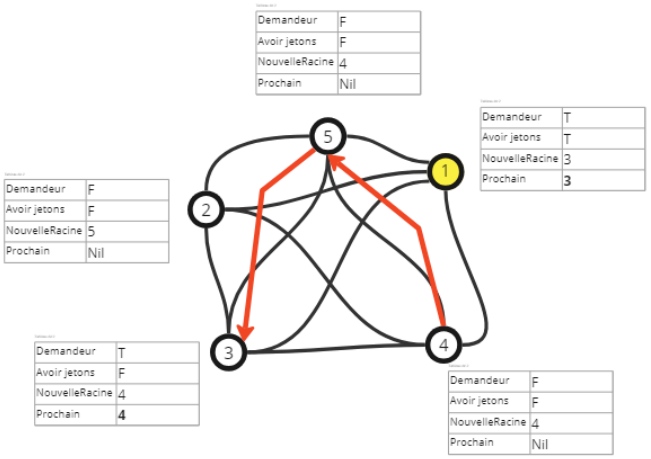
[CORPS ALGO]



- S1 demande à S5
- AJ de S5 <- False
- S5 envoie jeton à S1
- NR de S1 <- S1
- NR de S5 <- S1
- AJ de S1 <- True

- S3 demande à S5
- NR de S3 <- S3
- S5 envoie demande de S3 à S1
- NR de S5 <- S3
- P de S1 <- S3
- NR de S1 <- S3





- S4 demande à S5
- NR de S4 <- S4
- S5 demande à S3 pour S4
- NR de S5 <- S4
- P de S3 <- S4
- NR de S3 <- S4

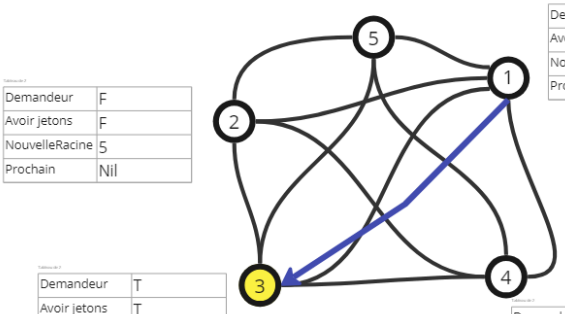
- S1 sort de la section critique
- AJ de S1 <- False
- S1 envoie jeton à S3
- P de S1 <- Nil
- D de S1 <- False
- AJ de S3 <- True

Demandeur	F
Avoir jetons	F
NouvelleRacine	4
Prochain	Nil

S1 sort de la section critique

Demandeur	F
Avoir jetons	F
NouvelleRacine	3
Prochain	NIL

Demandeur	F
Avoir jetons	F
NouvelleRacine	5
Prochain	Nil



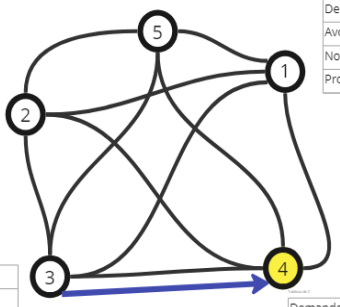
Demandeur	T
Avoir jetons	T
NouvelleRacine	4
Prochain	**4**

Demandeur	F
Avoir jetons	F
NouvelleRacine	4
Prochain	Nil

Demandeur	F
Avoir jetons	F
NouvelleRacine	4
Prochain	Nil

Demandeur	F
Avoir jetons	F
NouvelleRacine	3
Prochain	NIL

Demandeur	F
Avoir jetons	F
NouvelleRacine	5
Prochain	Nil



Demandeur	F
Avoir jetons	F
NouvelleRacine	4
Prochain	NIL

Demandeur	T
Avoir jetons	T
NouvelleRacine	4
Prochain	Nil

S3 sort de la section critique

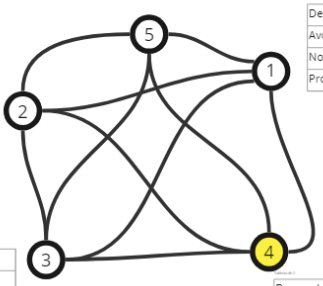
- AJ de S3 <- False
- S3 envoie jeton à S4
- P de S3 <- Nil
- D de S3 <- False
- AJ de S4 <- True
- D de S4 <- False

Etat final : S4 est dans le même état que S5 au début.

Demandeur	F
Avoir jetons	F
NouvelleRacine	4
Prochain	Nil

Demandeur	F
Avoir jetons	F
NouvelleRacine	3
Prochain	NIL

Demandeur	F
Avoir jetons	F
NouvelleRacine	5
Prochain	Nil



S4 sort de la section critique

Demandeur	F
Avoir jetons	F
NouvelleRacine	4
Prochain	NIL

Demandeur	F
Avoir jetons	T
NouvelleRacine	4
Prochain	Nil

Question 2/. A quoi correspondent les variables *NouvelleRacinep* et *Prochainp* ?

NouvelleRacinep : le dernier propriétaire du jeton connu par le sommet p. Permet notamment de transmettre la demande de jeton et de garantir que cette demande arrivera bien au propriétaire effectif du jeton.

Prochainp : le prochain nœud dans la file d'attente pour obtenir le jeton.

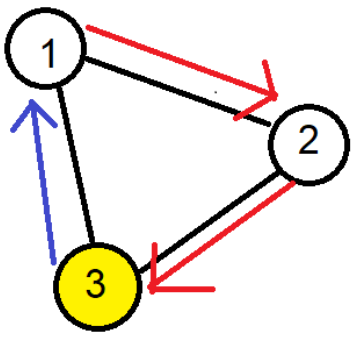
Question 3/. Pourquoi un sommet demandant le jeton l'obtiendra-t-il au bout d'un temps fini (s'il n'y a pas de panne) ?

Chaque nœud demandant le jeton, et ne le possédant pas déjà, envoie une demande au nœud qu'il sait détenteur du jeton. Le nœud potentiellement détenteur transmet le jeton au demandeur ou bien, s'il ne le possède pas, il transmet la demande au nœud qu'il estime lui-même détenteur, ainsi de suite.

Puisque chaque nœud est organisé linéairement (*NouvelleRacinep* et *Prochainp*) et qu'il y a un nombre de nœuds fini, il est garanti que chaque nœud aura la possibilité de recevoir le jeton s'il le demande.

L'algorithme garantit qu'une demande de jeton ne restera pas bloquée car la transmission du jeton est directe lorsque le détenteur reçoit la demande.

Question 4/. Quel est le nombre de messages échangés dans le pire des cas lors d'une demande ?



Dans le pire cas, le nombre de messages échangés est 2 (sans compter l'envoie de jeton).

Etant donné que le graph est complet, on peut directement envoyer la demande au sommet qui à le jeton ou qui on a enrengistré comme dernier ayant le jeton. Celui là va nous renvoyer vers celui qui à le jeton.

Exercice 2:

Question 1/.

Pourquoi dans la procédure Demander-SC attend-on n − M permissions avant d'entrer en section critique ?

On attend qu'au moins une entrée de la ressource soit libre. Cela se traduit par qu'on attend suffisamment de permissions pour accéder à la section critique. On est assuré qu'il y a au moins (n-M) places dans la section critique.

Exemple:
4 sommets, 3 entrées pour la section critique, il nous faut 4-3 permission pour avoir accès à la section critiques.

Que représente la donnée associée au message < REQUEST, k, j > ?

- L'estampille k est utilisée pour déterminer la priorité des demandes. Les demandes les plus anciennes sont les plus prioritaires.

- L'identifiant "j" est le sommet qui fait une demande.

Comment déterminer des niveaux de priorité grâce à ces données ?

Avec l'estampille on peut déterminer la demande la plus ancienne et lui donner permission si l'estampille est plus grande. Si les estampilles sont égales, on se base sur l'identifiant "j" du sommet le plus petit.

Si un sommet u reçoit un message < REQUEST, ., . > d'un sommet v plus prioritaire, et que lui-même demande une ressource, que doit il faire ?

Envoyer une requête permission au sommet v pour qu'il a un nombre de permission lui permettant d'accéder à la ressource.

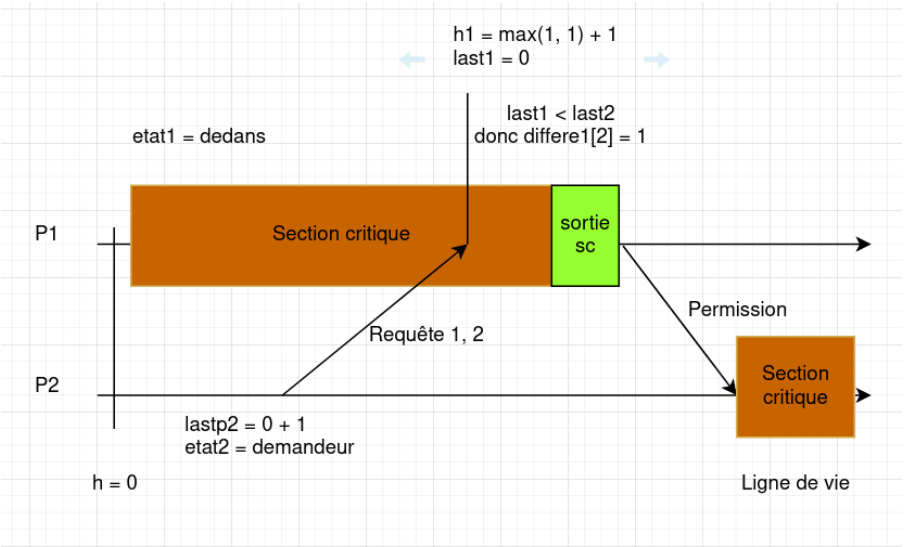
Question 2/. Écrire la procédure de réception de < REQUEST, ., . >.

//Les demandes les plus anciennes sont les plus prioritaires

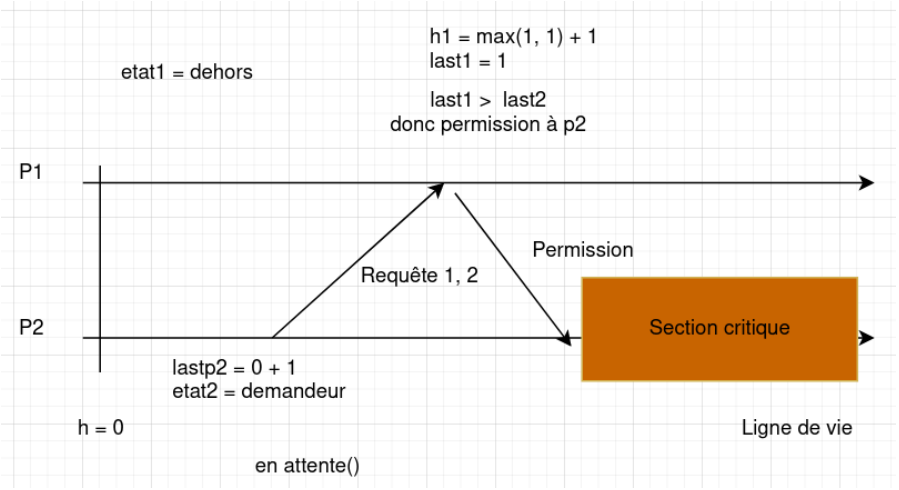
```
Lors de la réception de < REQUEST, k, j> :
  hi := max(k, hi)+1

  if etati == {dedans, demandeur}  et (lasti < k ou i (lasti =
k et i < j)):
    différéi[j] := différéi[j] + 1
  Sinon envoyer <PERMISSION i, 1> à j
```

Exemple d'exécution de notre procédure REQUEST :

P1 est plus prioritaire que P2:



P2 est plus prioritaire que P1:



Question 3/. Montrez (par contradiction) qu'au plus M sommets peuvent simultanément utiliser la ressource.
Preuve par l'absurde:

Supposons que :

P = "Plus de M sommets peuvent simultanément utiliser la ressource"

Cela signifie qu'il existe :

Q = "Il existe un ensemble de M+1 sommets qui utilisent la ressource en même temps"

Ainsi : P => Q

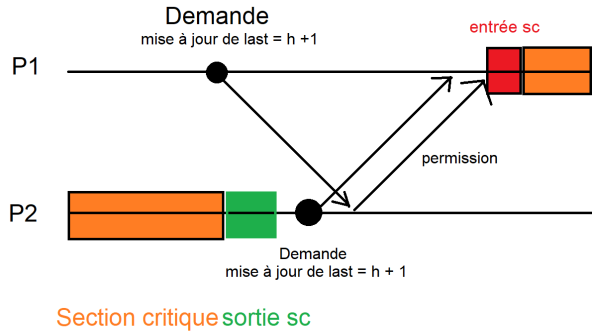
Un sommet ne peut entrer en section critique que lorsqu'il a reçu la permission de tous les autres sommets, on sait qu'il est nécessaire d'avoir au moins n-M permissions.

Alors, avec plus de M sommets pouvant entrer en section critique simultanément, il faudrait plus de n-M sommets donnant leur permission, ce qui contredit l'algorithme.

Donc, au plus M sommets peuvent simultanément utiliser la ressource.

Question 4/. Avec un tel algorithme, un site peut-il attendre indéfiniment avant d'entrer en section critique ?

En supposant que la phase d'utilisation de la section critique a une durée finie (cf cours), la propriété qu'il faut avoir pour répondre à la question est la Vivacité, chaque nœud demandant un accès à la ressource l'obtiendra au bout d'un temps fini car nous avons un système de file d'attente. On parle d'absence de famine.



Lorsqu'un nœud (p1) fait la demande il aura un timestamp donné et sera ajouté à la liste d'attente de (p2). Lorsqu'un autre nœud(p2) sort sa section critique. Il va envoyer les permissions à sa file d'attente différe i. S'il refait une demande d'accès à la ressource au même moment, il va mettre à jour son last et donc l'accès à la ressource pour le nœud (p1) qui fait la demande est assurée.

Dans la procédure REQUEST par `(lasti < k ou i (lasti = k et i < j))` et le tableau diffère i assure que la ressources soient reçues. Donc la propriété de **vivacité est vraie**.