Algorithmique distribuée : systèmes d'agents mobiles

Arnaud Labourel

Équipe d'algorithmique distribuée
LIS, CNRS & Aix-Marseille Université, France

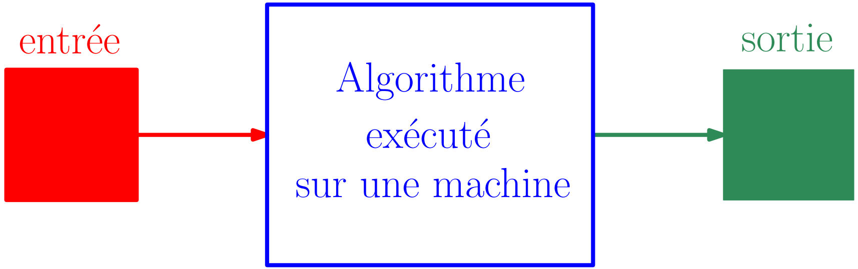
École Jeunes Chercheurs et Chercheuses en Informatique Mathématique

LABORATOIRE
D'INFORMATIQUE
& SYSTÈMES

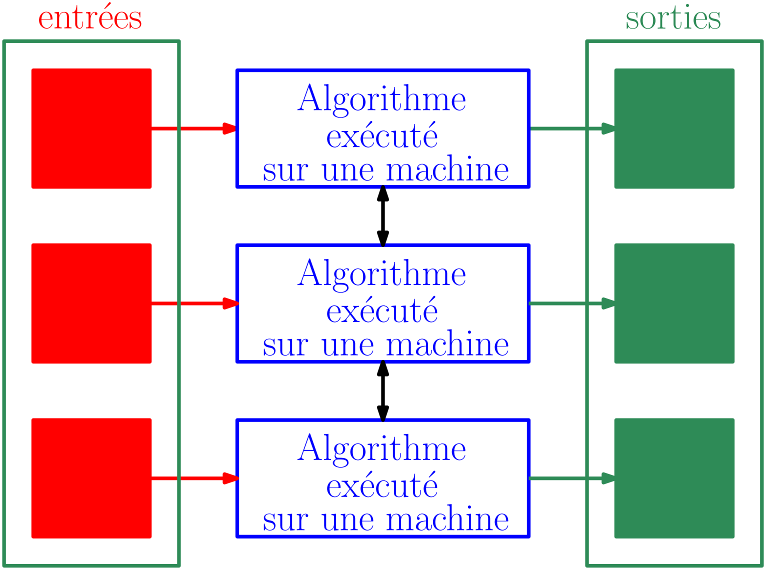
Aix*Marseille université
Initiative d'excellence

Introduction aux agents mobiles

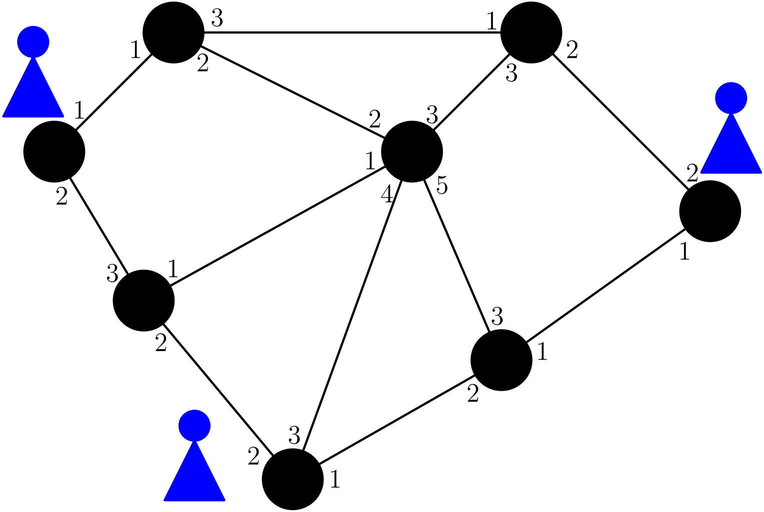
Algorithme classique



Algorithme distribuée

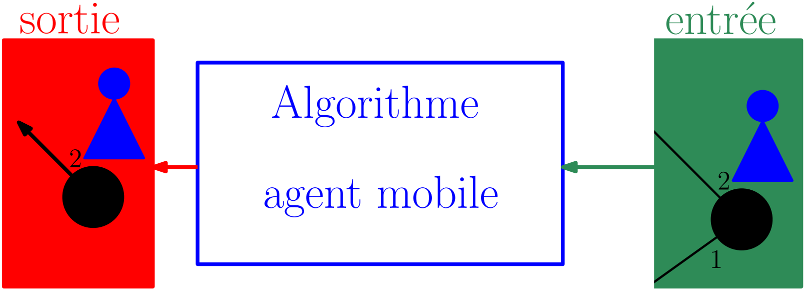


Agents mobiles

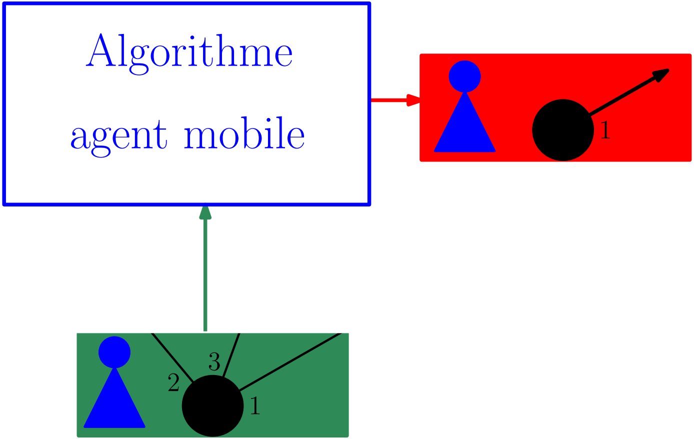


Agents mobiles

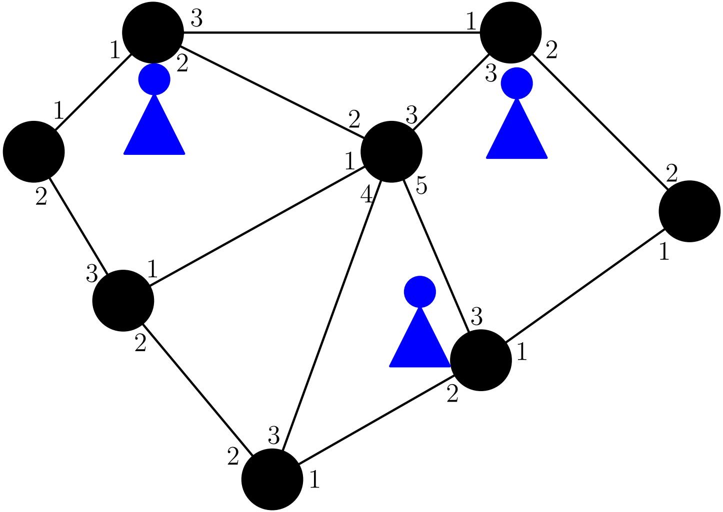
Agents mobiles



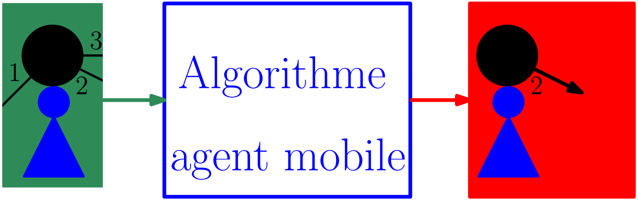
Agents mobiles



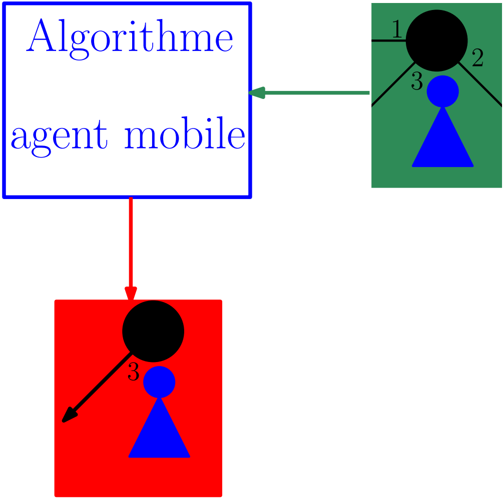
Agents mobiles



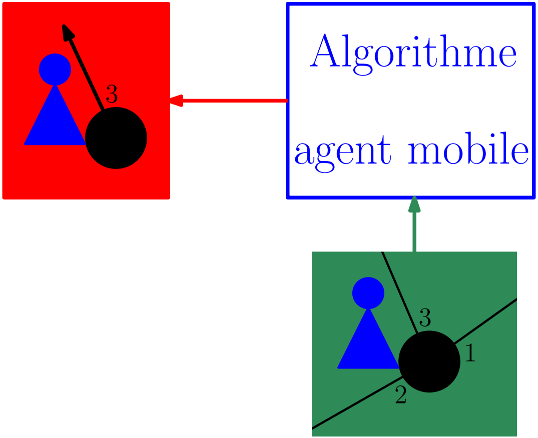
Agents mobiles



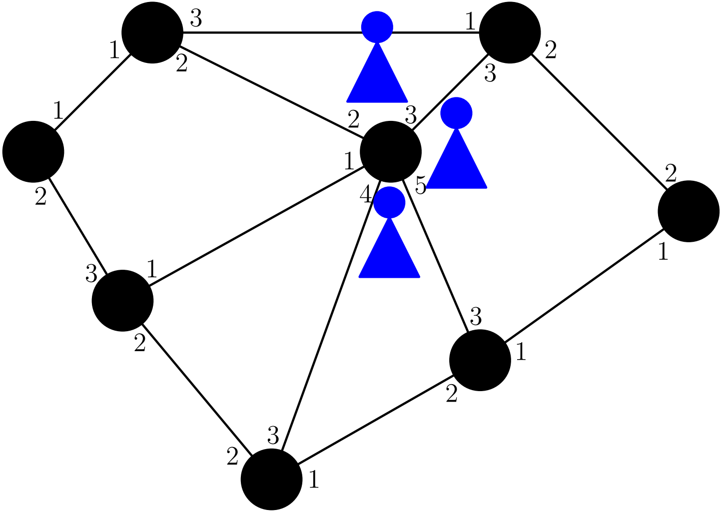
Agents mobiles



Agents mobiles



Agents mobiles



Systèmes distribués d'agents mobiles

Systèmes distribués composés d'agents mobiles se déplaçant au sein d'un réseau

Agent mobile

Processus mobile n'ayant qu'une vision locale du réseau

Réseau

Modélisé par un graphe

Algorithme pour agents mobiles

Algorithme local à chaque agent calculant son action (déplacement,attente, ...) à chaque étape à partir de sa vision locale et de l'état de sa mémoire

Navigation d'un agent dans un graphe

Les arêtes incidentes à un sommet sont numérotées localement (avec des numéros de port)

Déplacement des agents

À chaque étape :

- l'agent choisi un numéro de port sur le sommet courant
- il se déplace le long de l'arête correspondante
- il récupère le numéro de port de l'arête dans le nouveau sommet (numéro de port sortant)

Vision locale de l'agent

Pour le calcul de chaque déplacement, l'agent ne peut utiliser que :

- les informations locales :
  - le numéro de port par lequel il est arrivé
  - les numéros de ports disponibles sur le nœud courant
  - d'éventuelles informations présentes sur le nœuds courants (jetons ou message laissé par un agent)
- sa mémoire persistante, c'est-à-dire la mémoire conservée après chacun de ses déplacements.
- les informations disponibles à l'initialisation : identifiant, borne sur la taille, . . .

Tâches possibles pour des agents

- **Exploration :** visiter tous les nœuds d'un réseau (avec terminaison explicite ou non)
- **Cartographie :** construire la carte d'un réseau inconnu
- **Rendez-vous :** faire se rencontrer 2 ou plus agents
- ...

De nombreux modéles possibles

- Réseau **anonyme** ou pas (identifiants sur les nœuds)
- Agents anonymes ou **pas** (identifiants pour chaque agent)
- Réseau **synchrone** ou pas (les agents calculent et se déplace à la même vitesse ou pas)
- Connaissance préalable du réseau par l'agent ou pas (taille, topologie, ...)
- Moyens de communication entre agents :
  - Aucun
  - Tableaux blancs (possibilité d'écriture sur les nœuds)

Complexité d'un algorithme distribuée ?

Critères d'évaluation des algorithmes distribués

- la communication est le critère essentiel
- le temps de calcul local n'est pas important

Critères algorithmes d'agents mobiles

- **le nombre de mouvements des agents :** nombre total de mouvements des agents
- **la mémoire :** quantité de mémoire conservée par l'agent après chaque déplacement
- **la capacité de stockage de chaque nœud :** taille des informations que peuvent écrire les agents sur un nœud

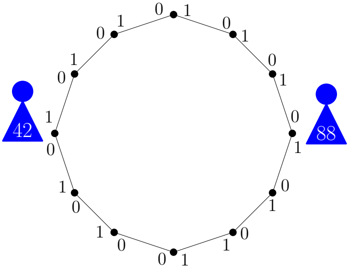
Le rendez-vous synchrone : agent **avec** d'identifiants

Rendez-vous syncrhone dans l'anneau

Le problème

Faire se rencontrer deux agents dans un anneau (cycle).

- Chaque agent a un identifiant ($\in \mathbb{N}$)
- le système est synchrone : la durée des étapes est la même pour les deux agents



Rendez-vous naïf dans l'anneau

Quelques idées naïves pour le rendez-vous dans l'anneau (cycle) :

- marche aléatoire : complexité $O(n(n-d))$
- chaque agent fait *id* tous de l'anneau et s'arrête : complexité $O(n \min\{id_1, id_2\})$
- chaque agent tourne à vitesse $\frac{1}{id}$ (se déplace puis attend $id - 1$ étapes) dans le sens des aiguilles d'une montre dans l'anneau : complexité $O\left(\frac{n.id_1.id_2}{|id_2 - id_1|}\right)$

Rendez-vous optimal dans l'anneau

Théorème [Dessmark, Fraigniaud, Pelc 2003]

Il existe un algorithme de rendez-vous déterministe en temps $O(D\log(\min\{id_1, id_2\}))$ dans l'anneau.

D : distance de départ entre les deux agents
id_1 et id_2 : les identifiants des deux agents.

Remarques :

- C'est (presque) optimal !
  $\Rightarrow$ borne inférieure en $\Omega(D\log(\min\{id_1, id_2\}))$
- On n'a pas besoin de connaître la taille de l'anneau.
- Pas besoin d'être d'accord sur un sens de rotation autour de l'anneau.

Préliminaire

Pour chaque identifiant on construit un mot binaire :

- brique de base : représentation de l'identifiant *id* en binaire.
  Exemple : $15 \rightarrow 1111$, $16 \rightarrow 10000$
- On rajoute des 0 pour que le mot soit de longueur égale à une puissance de 2.
  $\Longrightarrow$ longueur du mot $= 2^{\lfloor \log \log id \rfloor + 1}$
  Exemple : $15 \rightarrow 1111$, $16 \rightarrow 00010000$

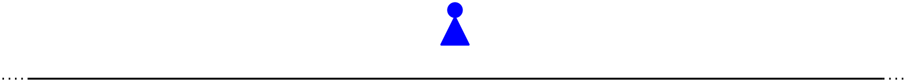
On note id^* le mot obtenu

Identifiants de tailles similaires (algo. 1)

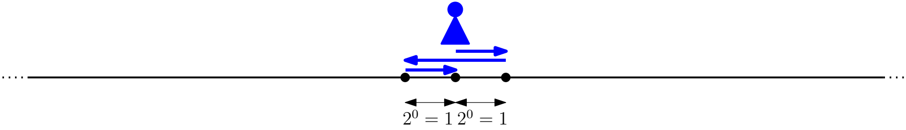
Algorithme 1 de rendez-vous dans l'anneau pour des agents id_1 et id_2 tels que $\lfloor \log \log id_1 \rfloor = \lfloor \log \log id_2 \rfloor$

pour *i de 1 à* $+\infty$ **faire**
  pour *j de 1 à* $|id^*|$ **faire**
    si $id^*[j] = 1$ **alors**
      se déplacer de 2^i pas vers la droite;
      se déplacer de 2^{i+1} pas vers la gauche;
      se déplacer de 2^i pas vers la droite;
    sinon
      attendre pendant 2^{i+2} étapes;
  fin
fin

Exécution de l'algorithme 1

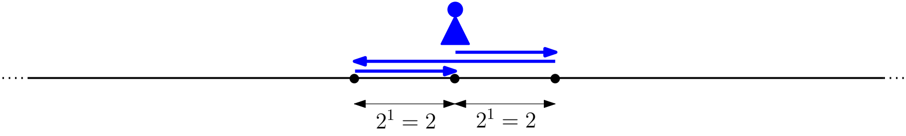


Exécution de l'algorithme 1



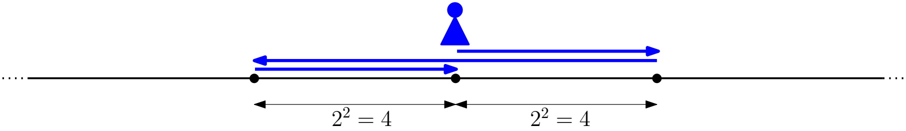
$$i = 0$$

Exécution de l'algorithme 1



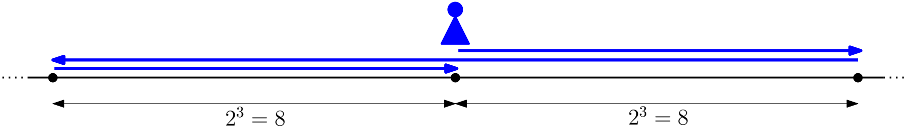
$i = 1$

Exécution de l'algorithme 1



$$i = 2$$

Exécution de l'algorithme 1



$$i = 3$$

Preuve de l'algorithme 1

On a $|id_1^*| = |id_2^*|$ et on est synchrone
$\Longrightarrow$ les deux agents ont les mêmes valeurs de i et j à tout moment

Puisque les identifiants sont différents il existe un k tel que le k-ième bit de id_1^* est différent de celui de id_2^*.
Pour $i = \lceil \log D \rceil$ et $j = k$, les deux agents se rencontrent car :

- un agent atteint la position de départ de l'autre agent
- l'autre agent ne bouge pas et reste sur sa position de départ

Complexité de l'algorithme 1

$|id_1^*| = |id_2^*|$

Coût de l'étape i **:** $2^{i+2}.|id_1^*|$

Coût total jusqu'à l'étape i **incluse :** $O(2^i \log(id_1))$

Le rendez-vous se produit à l'étape $i = \lceil \log D \rceil$

Complexité : $O(D \log(\min\{id_1, id_2\}))$

Identifiants de tailles différentes (algo. 2)

On va maintenant donner un algorithme pour le cas d'agents id_1 et id_2 tels que $\lfloor \log \log id_1 \rfloor = \lfloor \log \log id_2 \rfloor$.

Pour cela, on utiliser la valeur suivante :

$$id^+ = \begin{cases} 1 & \text{si } id = 1 \\ \lfloor \log \log id \rfloor + 2 & \text{autrement} \end{cases}$$

Identifiants de tailles différentes (algo. 2)

Algorithme 2 de rendez-vous dans l'anneau pour des agents id_1 et id_2 tels que $\lfloor \log \log id_1 \rfloor \neq \lfloor \log \log id_2 \rfloor$

pour *i de 1 à* $+\infty$ **faire**
  pour *j de 1 à i* **faire**
    si $j = i - id^+$ **alors**
      se déplacer de 2^j pas vers la droite;
      se déplacer de 2^{j+1} pas vers la gauche;
      se déplacer de 2^j pas vers la droite;
    sinon
      attendre pendant 2^{j+2} étapes;
  fin
fin

Preuve de l'algorithme 2

On a $id_1^+ \neq id_2^+$ et on va supposer que $id_1^+ < id_2^+$.

Durant la i-ème itération de la boucle, l'agent 1 parcourt tous les nœuds de l'anneau à distance $\leq 2^{i-id_1^+}$.

Pour le plus petit s tel que $2^{s-id_1^+} \geq D$, l'agent 1 rencontre l'agent 2 puisque celui-ci attend sur sa position de départ car $id_1^+ \neq id_2^+$.

Complexité :
$O(2^s) = O(2^{id_1^+ + \log D}) = O(D \log(\min\{id_1, id_2\}))$

Algorithme final

Idée

Alterner les exécutions des deux algorithmes

Algorithme final :

pour *i de 1 à* $+\infty$ **faire**

  Exécuter l'algorithme 1 pendant 2^i étapes;
  Revenir sur la position de départ en t étapes;
  Attendre $2^i - t$ étapes;
  Exécuter l'algorithme 2 pendant 2^i étapes;
  Revenir sur la position de départ en t étapes;
  Attendre $2^i - t$ étapes

fin

Preuve de l'algorithme final

Les agents exécutent le même algorithme au même moment.

Un des deux algorithmes va réaliser le rendez-vous en $c = O(D \log(\min\{id_1, id_2\}))$.

Les agents se rencontreront donc à l'itération numéro $i = \lceil \log c \rceil$ de la boucle.

Complexité : $O(2^i) = O(D \log(\min\{id_1, id_2\}))$

Borne inférieure en $\Omega(\log(\min\{id_1, id_2\}))$

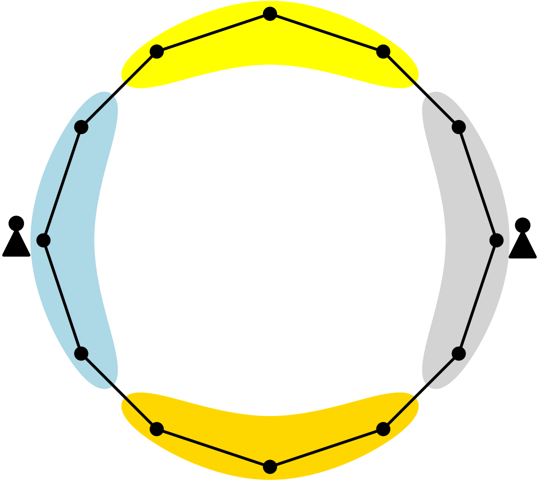
Théorème [Dessmark, Fraigniaud, Pelc 2003]

Tout algorithme de rendez-vous déterministe a une complexité en temps de $\Omega(D \log(\min\{id_1, id_2\}))$ dans l'anneau.

L'algorithme de rendez-vous vu précédemment est donc optimal à un facteur multiplicatif constant près.

Idée de la preuve

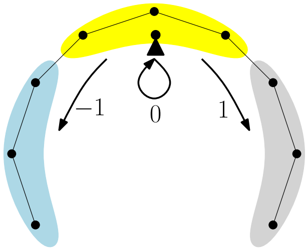
On découpe l'anneau en tranches de $D/2$ nœuds.



Idée de la preuve

On regarde la position de l'agent tout les $D/2$ étapes.

- 0 : il est dans la même tranche
- 1 : il est dans la tranche de droite
- -1 : il est dans la tranche de gauche



Cela nous donne un code de comportement
(par exemple $(0, 1, -1, 1, \ldots)$)

Idée de la preuve

Démonstration par contradiction

Supposons qu'il existe un algorithme résolvant le rendez-vous pour $Dy/4$ mouvements avec $y \in \mathbb{N}$ pour tous identifiants $id_1, id_2 \leq 2^y$.

Il y a $3^{y/2} < 2^y$ codes de comportement de longueur $y/2$.

On peut trouver deux identifiants $id_1, id_2 \leq 2^y$ qui ont le même code de comportement de longueur $y/2$.

Ces deux agents ne se rencontrent pas avant $Dy/4$ mouvements.

Contradiction !
$\Longrightarrow$ complexité en $\Omega(Dy) = \Omega(D \log(\min\{id_1, id_2\}))$

Les arbres en synchrone

Théorème [Dessmark, Fraigniaud, Kowalski et Pelc 2003]

Il existe un algorithme de rendezvous synchrone avec délai ayant comme complexité $O(n + \log(\min\{id_1, id_2\}))$ dans les arbres.

Théorème [Dessmark, Fraigniaud, Kowalski et Pelc 2003]

Il existe des arbres de n nœuds tel que tout algorithme de rendezvous synchrone avec délai a une complexité $\Omega(n + \log(\min\{id_1, id_2\}))$.

Algorithme RV-arbre

Idée de l'algorithme

- Explorer l'arbre
- Trouver le nœud ou l'arête centrale
- Rendez-vous sur le nœud ou l'arête centrale

Algorithme RV-arbre :

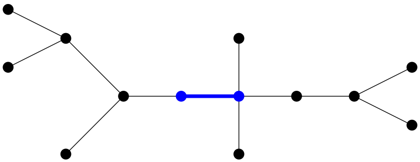
Explorer l'arbre;
si *l'arbre possède un nœud central* **alors**
    Aller sur ce nœud;
sinon
    Aller sur un nœud de l'arête centrale;
    Exécuter RV-arête;
fin

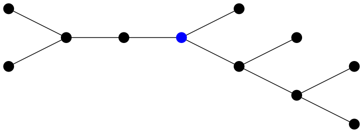
Noeud ou arête centrale

Centre : nœud minimisant la distance maximale à tout nœud du graphe

Théorème (Jordan 1869)

Un arbre a un centre ou deux centres adjacents (arête centrale).





Exploration d'arbre : Main Droite contre le Mur (MDM)

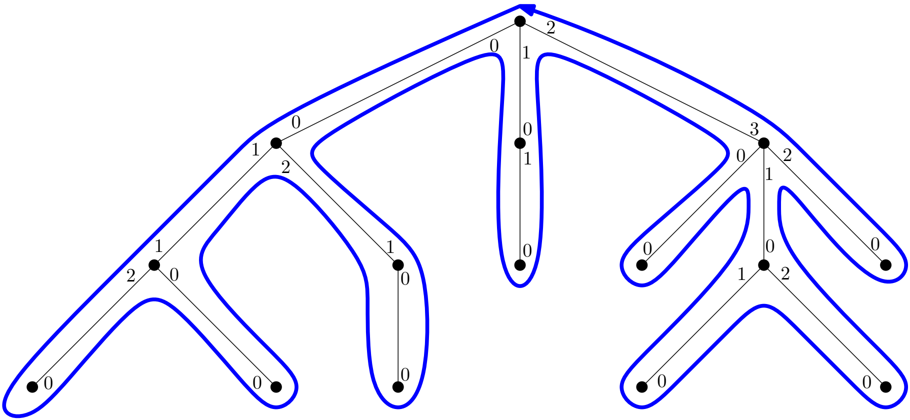
MDM pour résoudre des labyrinthes

Toujours suivre le mur de droite avec sa main lorsqu'il y a un embranchement : l'embranchement suivant dans le sens inverse des aiguilles d'une montre.

Version agent mobile

Se déplacer par le numéro de port suivant celui par lequel on est arrivé : $i + 1$ modulo le degré du nœud si on est arrivé par le port numéro i.

Main droite contre le mur



Algorithme main droite contre le mur

k : une borne sur le nombre d'arête de l'arbre

Algorithme MDM

$p :=$ numéro de port d'arrivée (initialement égal à 0);
$d :=$ degré du nœud courant;
Se déplacer via le numéro de port $(p + 1) \mod d$;
i := i+1 (initialement égal à 0);
si $i \geq 2k$ **alors**
  s'arrêter;

Remarque

On a pas besoin de la borne k si l'agent a assez de mémoire pour stocker l'arbre.

RV-arête

Algorithm RV-arête(id)

pour *j de 1 à* $+\infty$ **faire**

  traverser l'arête;

  attendre une étape;

  pour *i de 0 à* $\lfloor \log id \rfloor$ **faire**

    si $id[i] = 1$ **alors**

      traverser l'arête deux fois;

    sinon

      attendre deux étapes;

  fin

fin

RV-arête

Exemple d'exécution de RV-arête :
Pour *id* $= 101$, on a comme comportement
(1 déplacement, 0 rester sur place) :

10110011 10110011 10110011

Pour *id* $= 10011$, on a comme comportement :

101100001111 101100001111 101100001111

Lemme

Une fois les deux agents activés le rendez-vous se produit en au plus $\log(\min\{id_1, id_2\}) + 6$ étapes.

Preuve de l'algorithme RV-arête

$\tau =$ nombre d'étapes entre l'activation du premier agent A_1 et celle du second agent A_2

Cas 1 : τ est impair

A_1 : $1011110\textcolor{red}{0}\ldots$
A_2 : $\text{———}10\textcolor{red}{1}1\ldots$

A_2 exécute les bits 10 pour les deux premières étapes. A_1 doit donc exécuter 10 en même temps pour éviter le rendez-vous.

A_1 reste donc dans ce cas immobile pour la troisième étape. Le rendez-vous se produit à la troisième étape car l'étiquette de A_2 commence par un 1 et il doit donc se déplacer.

Preuve de l'algorithme RV-arête

Cas 2 : τ est pair et pas divisible par $2\lfloor \log id_1 \rfloor + 4$

A_1 : $1011000011\ldots$
A_2 : ———————$10\ldots$

A_2 exécute les bits 10 pour les deux premières étapes.

A_1 exécute 11 ou 00 pour les deux premières étapes.
$\Rightarrow$ Le rendez-vous se produit durant les deux premières étapes.

Preuve de l'algorithme RV-arête

Cas 3 : τ **est divisible par** $2\lfloor \log id_1 \rfloor + 4$ **et** $\lfloor \log id_1 \rfloor = \lfloor \log id_2 \rfloor$

A_1 : $1011000010110000\ldots$
A_2 : $\text{―――――――}10110011\ldots$

Les étiquettes de A_1 et A_2 diffèrent sur le bit numéro b.

À l'étape $2b+1$, les deux agents ont un comportement différent et le rendez-vous se produit.

Preuve de l'algorithme RV-arête

Cas 4 : τ est divisible par $2\lfloor \log id_1 \rfloor + 4$ **et** $\lfloor \log id_1 \rfloor \neq \lfloor \log id_2 \rfloor$

A_1 : $10110000101100001\textcolor{red}{0}\dots$
A_2 : $\text{———————}101100001\textcolor{red}{1}\dots$

Soit l la plus petite des étiquettes des agents.

L'agent ayant la plus petite étiquette l a un comportement différent aux étapes $2\lfloor \log l \rfloor + 5$ et $2\lfloor \log l \rfloor + 6$.

L'autre agent a le même comportement (11 ou 00)
$\Rightarrow$ Le rendez-vous se produit au plus tard à l'étape $2\lfloor \log l \rfloor + 6$.

Le rendez-vous dans les graphes généraux

Théorème [Ta-Shma et Zwick 2009]

Il existe un algorithme de rendez-vous synchrone avec délai ayant comme complexité $\tilde{O}(\Delta^2.n^3.\log l)$ pour tous les graphes.

l : plus petite des deux étiquettes des agents
n : nombre du sommets du graphe
Δ : degré maximal du graphe
L'algorithme fonctionne même si les agents ne voient pas sur quels port ils arrivent sur les sommets.

Notation $\tilde{O}$: même comportement asymptotique à un polylog près (exemple : $n(\log n)^2 = \tilde{O}(n)$)

Idée de la preuve

UTS : suite de numéro de port tels que tout agent suivant ses numéro de ports dans un graphe de taille n explore entièrement le graphe.

Théorème [Aleliunas et al. 1979]

Il existe un UTS de longueur $O(\Delta^2.n^3.\log n)$ pour tous les graphes de n sommets et de degré Δ.

En fait, il existe une suite infinie de numéros de port telle que tout sous-chaîne de longueur $\Omega(\Delta^2.n^3.\log n)$ est un UTS pour les graphes de n sommets et de degré Δ.
Idée de l'algorithme : reprendre l'algorithme dans l'anneau en suivant cette séquence de numéros de port.

Bornes inférieures pour le temps de rendez-vous

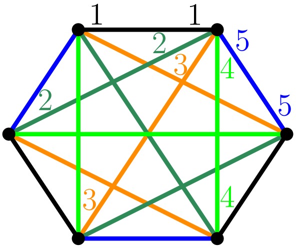
Théorème [Dessmark, Fraigniaud, Pelc 2003]

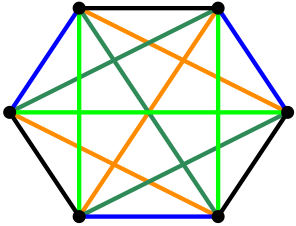
Pour tout algorithme de rendez-vous et pour toute paire d'étiquette id_1, id_2, il existe un graphe de n sommets tel que le rendez-vous se produit après un temps $\Omega(n^2)$.

Théorème [Dessmark, Fraigniaud, Pelc 2003]

Tout algorithme de rendez-vous déterministe a une complexité en temps de $\Omega(n \log(\min\{id_1, id_2\}))$ dans l'anneau.

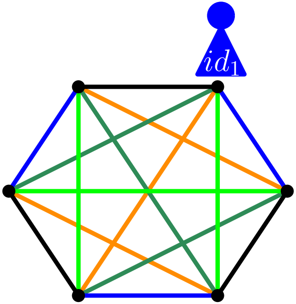
Preuve de la borne inférieure en $\Omega(n^2)$





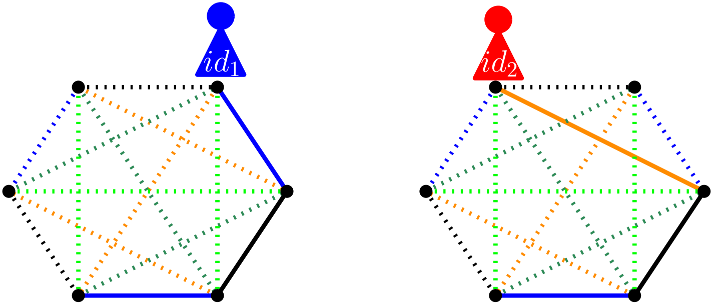
Prendre deux copies d'un graphe complet à k sommets

Preuve de la borne inférieure en $\Omega(n^2)$



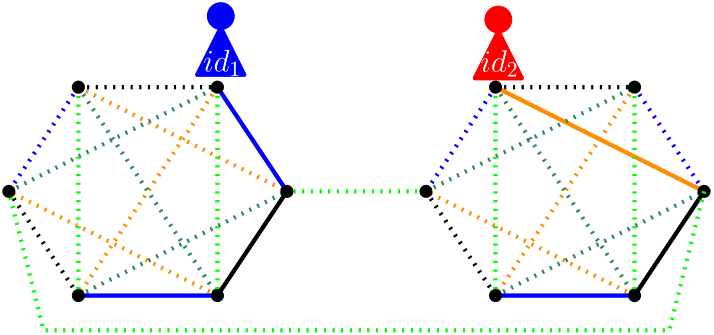
Mettre un agent dans chaque graphe complet

Preuve de la borne inférieure en $\Omega(n^2)$



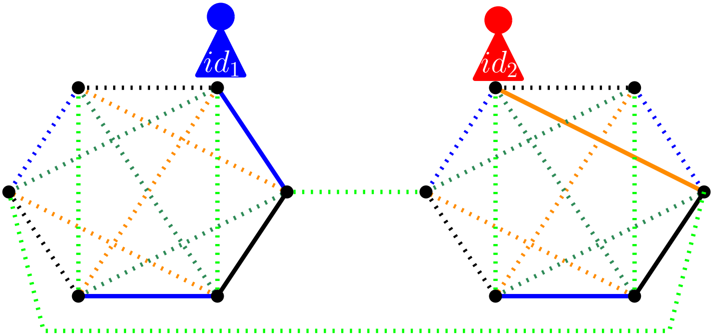
Considérer une simulation de l'algorithme avec $\frac{k^2}{12}$ mouvements pour chaque agent

Preuve de la borne inférieure en $\Omega(n^2)$



Connecter les deux graphes en utilisant des arêtes non-explorées

Preuve de la borne inférieure en $\Omega(n^2)$



Le rendez-vous ne peut pas se produire avant $\frac{k^2}{12} = \Omega(n^2)$ étapes

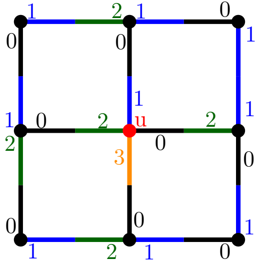
Le rendez-vous synchrone : agent **sans** d'identifiants

Rendez-vous avec des agents anonymes

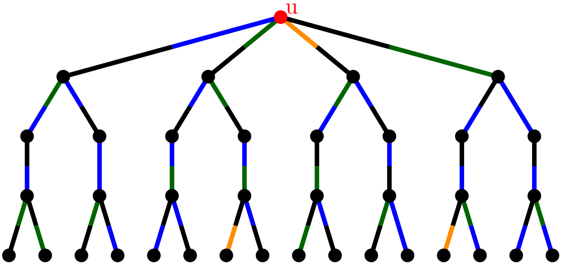
Qu'est-ce qu'il est possible de faire avec des agents anonymes (sans identifiants) ?

Un concept important : la **vue** d'un sommet

Graphe avec numéros de port



Vue du sommet u

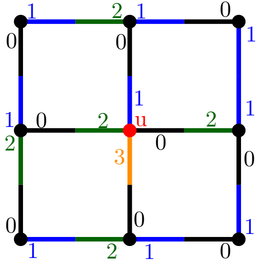


Vue d'un sommet

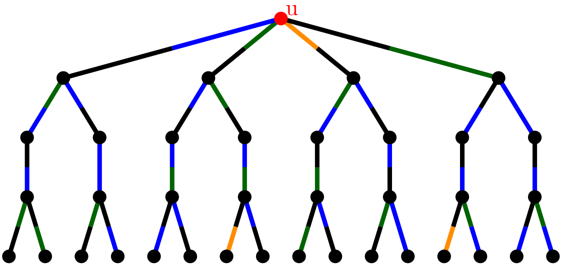
Vue d'un sommet

Arbre de tous les chemins (suffisant de considérer les chemins de longueur *n*) partants du sommet avec les numéros de ports.

Graphe avec numéros de port



Vue du sommet u



Possibilité du rendez-vous

Lemme

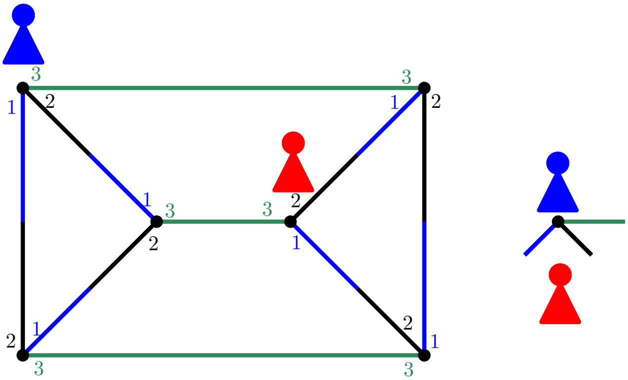
Le rendez-vous est impossible pour deux agents anonymes qui commencent dans deux sommets ayant la même vue.

Théorème [Czyzowicz, Kosowski et Pelc 2012]

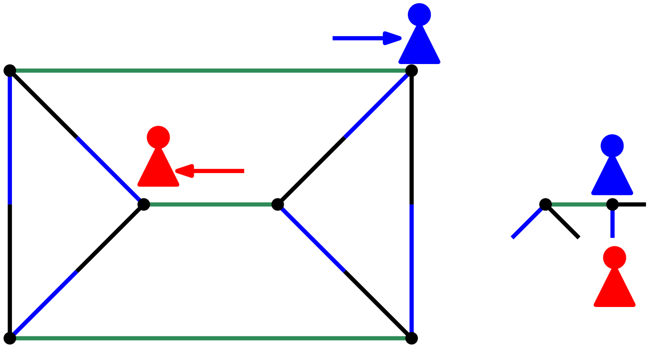
Il existe un algorithme pour deux agents anonymes qui garantit le rendez-vous avec délai si les deux agents commencent dans deux sommets ayant des vues différentes.

Mémoire : $O(\log n)$ bits
Temps polynomial en n si départ simultané

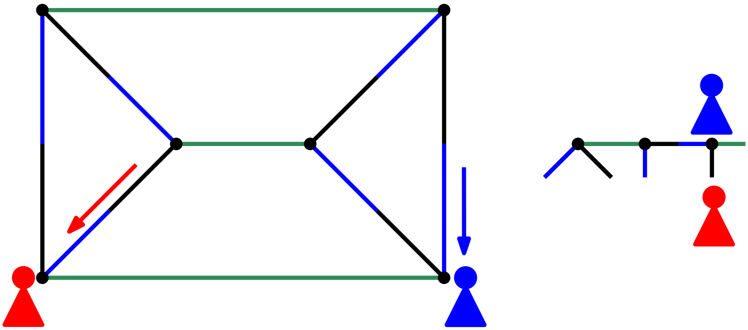
Impossibilité du rendez-vous en cas de vues identiques



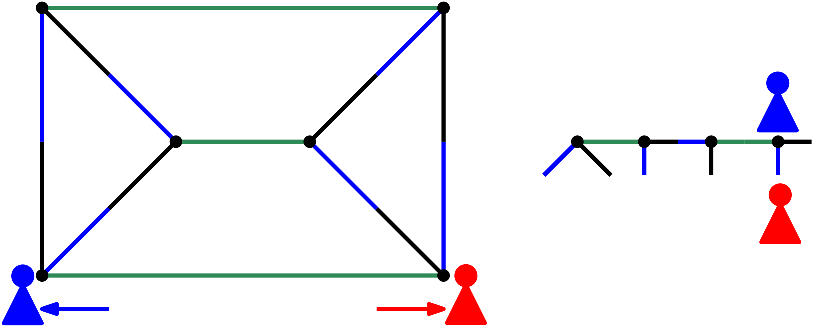
Impossibilité du rendez-vous en cas de vues identiques



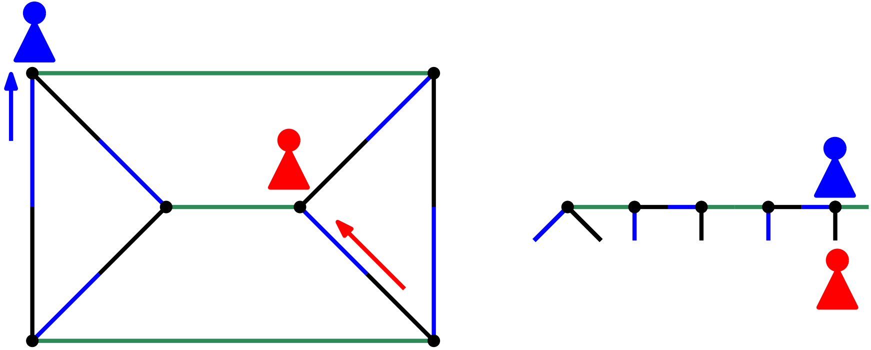
Impossibilité du rendez-vous en cas de vues identiques



Impossibilité du rendez-vous en cas de vues identiques



Impossibilité du rendez-vous en cas de vues identiques



Idée de l'algorithme de rendez-vous

Utiliser les vues pour générer un identifiant pour chaque agent qui permettra d'exécuter un algorithme de rendez-vous.
Ingrédients clés pour obtenir un algorithme utilisant $O(\log n)$ bits :

- UXS : Universal eXploration Sequence constructible avec $O(\log n)$ bits [Reingold 2008]
- Signature : suite des étiquettes (couple de numéro de port) des arêtes traversées par un UXS
- 2 vues différentes $\Rightarrow$ signatures des 2 UXS (pour un graphe de $O(n^2)$ sommets) différentes

Mémoire pour le rendezvous dans les arbres

On se place dans le cas d'agents anonymes dont les positions de départ ne sont pas symétriques.

Théorème

Le RV avec délai dans l'arbre nécessite $\Omega(\log n)$ bits de mémoire même pour la ligne.
Le RV sans délai dans l'arbre nécessite $\Omega(\log\log n + \log f)$ bits de mémoire avec f=#feuilles.

Théorème [Fraigniaud et Pelc 2008]

Il existe un algorithme de RV utilisant $O(\log\log n + \log f)$ bits de mémoire.

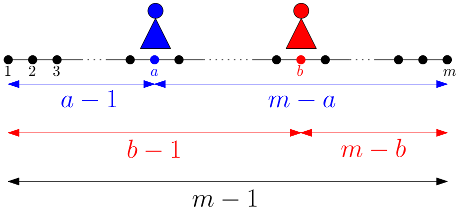
Algorithme RV-ligne (sans délai)

Algorithm RV-ligne

Se déplacer vers une des extrémités du chemin ;
$p := 2$;
Tant que le RV ne s'est pas produit
traverser le chemin 2 fois à vitesse $\frac{1}{p}$;
($\forall$ arête attendre $p - 1$ étapes puis traverser)
$p :=$ plus petit entier premier $> p$;

Mémoire utilisée : $O(\log \log n)$

Preuve de l'algorithme RV-ligne



Lemme

Si le rendez-vous d'agent anonyme est possible pour des agents anonymes, c'est-à-dire :

- m est impair **ou**
- m est pair et $a-1 \neq m-b$

Alors le rendez-vous se produit lors de l'itération $t = \log(m)$ de la boucle de l'algorithme RV-ligne.

Idée de la preuve

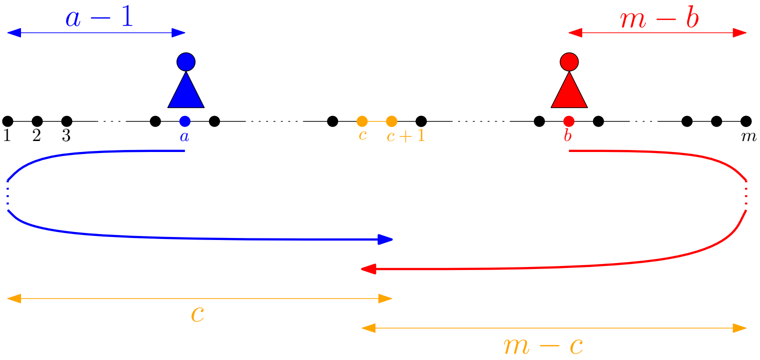
Si le rendez-vous ne se produit pas à la j-ème itération alors les deux agents se sont croisés sur l'arête entre le sommet c et le sommet $c+1$.

Il y a quatre cas possibles suivant la direction prise au départ par les deux agents.

p_i : i-ème nombre premier

On pose $S = 2(m-1)\sum_{i=1}^{j-1} p_i$ pour le nombre d'étapes des itération pour i de 1 à j.

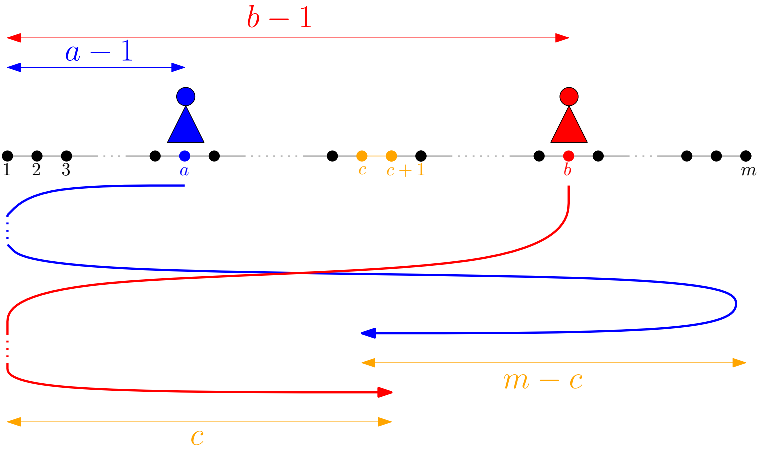
Idée de la preuve : cas 1



On a :

$$(a-1) + S + cp_j = (m-b) + S + (m-c)p_j$$

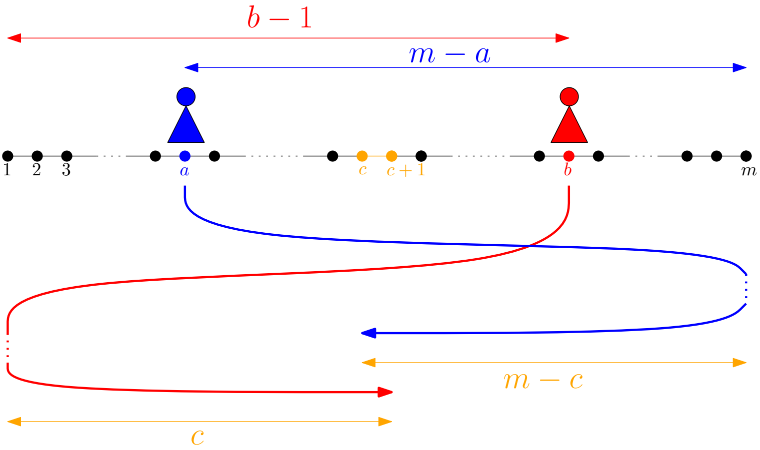
Idée de la preuve : cas 2



On a :

$$(a-1)+S+(m-1)p_j+(m-c)p_j=(b-1)+S+cp_j$$

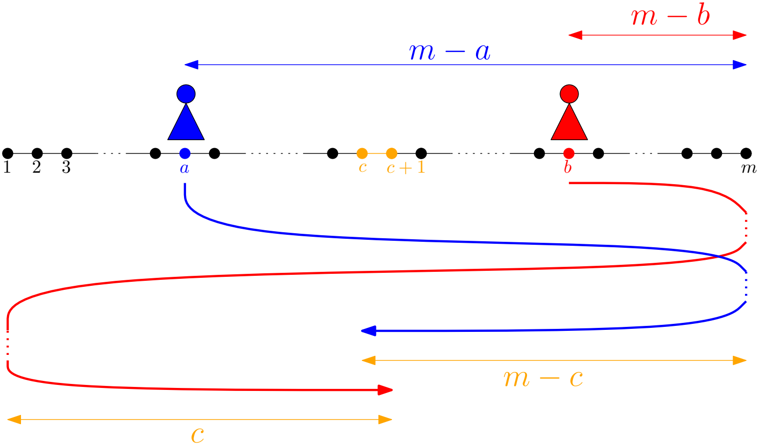
Idée de la preuve : cas 3



On a :

$$(m-a)+S+(m-c)p_j=(b-1)+S+cp_j$$

Idée de la preuve : cas 4



On a :

$$(m-a)+S+(m-c)p_j = (m-b)+S+(m-1)p_j + cp_j$$

Idée de la preuve

Les 4 cas possible nous donnent les égalités suivantes :

- $(2c - m)p_j = m - a - b + 1$
- $(2c - 2m + 1)p_j = a - b$
- $(2c - m)p_j = m - a - b + 1$
- $(2m - 2c + 1)p_j = a - b$

Dans tous les cas p_j divise $|a - b|$ ou $|m - a - b + 1|$
$\prod_{i=1}^{j} p_i$ divise $|a - b| \times |m - a - b + 1|$
Le rendez-vous se produit avant l'itération t avec t plus grand entier tel que $\prod_{i=1}^{t} p_i \leq m^2$

Idée de la preuve

$\pi(x)$: # entiers premier plus petit que x
$\prod_{i=1}^{j} p_i \geq 2^{\pi(P_j)}$
Le rendez-vous se produit avant l'itération t avec t plus grand entier tel que :
$2^{\pi(p_t)} \leq m^2$ et donc $\pi(p_t) \leq 2 \log m$

Pour x assez grand on a $\pi(x) \geq \frac{x}{2 \ln x}$
Le rendez-vous se produit avant l'itération t avec t plus grand entier tel que : $p_t / ln(p_t) \leq 4 \log m$
On a besoin de $O(\log p_t) = O(\log \log m)$ bits de mémoire.