Algorithmes Probabilistes

1. Généralités
2. La “méthode probabiliste” sur un exemple: Ramsey
3. Arrondi aléatoire: MAXSAT
4. Echantilonnage: Points les plus proches
5. Propriétés particulières des données: Hachage parfait
6. Permutation aléatoire des données: routage sur l’hypercube
7. Désymétrisation en algorithmique distribuée: les philosophes

Algorithme Probabiliste

Lors de l'exécution, l'algorithme fait des choix probabilistes guidés par des tirages aléatoires : par exemple des lancers de pièces pile ou face.

Deux catégories

1. Monte Carlo Le résultat du calcul est aléatoire. Toujours rapide, Probablement correct
2. Las Vegas Le temps d'exécution est aléatoire. Toujours correct, Probablement rapide

Propriété de Ramsey

Dans tout groupe de $n \geq 6$ personnes, il existe toujours, ou bien **k=3** personnes qui se connaissent, ou bien **3** personnes qui ne se connaissent pas.

Faux pour les groupes de **n=5** personnes.

Quelle doit être la taille du groupe pour garantir l'existence de **k=4** personnes qui se connaissent toutes mutuellement ou telles qu'aucune n'en connaisse une autre?

Réponse: n=18.

Théorème : Soit n tel que :

$$2\binom{n}{k} < 2^{k(k-1)/2}$$

alors il existe des groupes de n personnes sans sous-groupe de k personnes qui se connaissent toutes mutuellement ou ne se connaissent pas du tout mutuellement.

Preuve :

1. On prend une configuration **aléatoire** de n personnes. $\Pr\{i \text{ connaît } j\} = 1/2$.
2. On montre que

$$\Pr\{\exists \text{ sous-groupe de } k \text{ personnes satisfaisant la propriété }\} < 1.$$

3. On en déduit qu'il existe une configuration démontrant le théorème.

Problème ouvert : construction explicite.

Outils d'analyse probabiliste

Inégalité de Markov. Soit X une variable aléatoire positive de moyenne $E(X)$. Alors pour tout t :

$$Pr(X \geq t \cdot E(X)) \leq \frac{1}{t}.$$

Inégalité de Chebychev : Soit X une variable aléatoire X de moyenne μ et d'écart-type σ. Alors pour tout a :

$$Pr(|X - \mu| \geq a\sigma) \leq \frac{1}{a^2}.$$

Grandes déviations Soient $X_1, \ldots, X_n$ n variables aléatoires indépendantes valant 1 avec probabilité p et 0 avec probabilité $1 - p$, et soit $X = \sum X_i$, $E(X) = np$. Alors :

$$Pr(X \geq (1+\delta)np) < \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^{np}.$$

Analyse probabiliste ou algorithme probabiliste

Analyse de quicksort : Complexité $\Theta(n^2)$ dans le pire cas, mais $O(n \log n)$ en moyenne si les nombres à trier forment une permutation aléatoire.

Quicksort randomisé :

1. Faire une permutation aléatoire des nombres donnés en entrée, puis
2. Exécuter Quicksort

Analyse de quicksort randomisé : La complexité est une variable aléatoire. Quels que soient les nombres donnés en entrée, la complexité est $O(n \log n)$ en moyenne, où la moyenne porte sur le choix de la permutation aléatoire.

Avantages : contrôle de la distribution, donc pas de données pathologiques

Le problème MAX-SAT, la méthode de l'arrondi probabiliste

Données

- Un ensemble de clauses, c'est à dire de formule booléennes ne comportant que des $\vee$ et portant sur des variables ou leur complément

Résultat attendu

- Une affectation de valeurs aux variables telle que le nombre maximum de clauses prennent la valeur $vrai$
- Un algorithme naïf consiste à tirer au hasard une valeur pour chaque variable
- Donne une solution $\frac{1}{2}$-approchée avec forte probabilité

Mise en forme de problème linéaire pour MAX-SAT

Transformation des données

- On considère des variables y_i à valeurs 0 (pour $faux$) ou 1 (pour vra), une pour chaque variable booléenne x_i et des variables z_j, une pour chaque clause, indiquant si elle est satisfaite.
- On doit optimiser :

$$Max \ \ (z_1 + z_2 + \ldots + z_m)$$

- Avec une contrainte par clause C_j :

$$z_j \leq \sum_{x_i \in C_j} y_i + \sum_{\bar{x_k} \in C_j} (1 - y_k)$$

Arrondi probabiliste

Résultat intermédiaire

- Résolution du problème linéaire, en remplaçant la contrainte $y_i, z_j = 0, 1$ par $0 \leq y_i \leq 1, 0 \leq z_j \leq 1$
- Donne une solution qui pas nécessairement en nombre entiers

Transformation du résultat

- Si on obtient comme résultat du problème linéaire $\hat{y_i}$ on donne alors à la variable x_i une valeur obtenu par tirage aléatoire donnant à $vrai$ la probabilité $\hat{y_i}$
- On a alors une solution telle que le nombre de clauses satisfaites est inférieure ou égale à :

$$\hat{z_1} + \hat{z_2} + \ldots + \hat{z_m}$$

Arrondi probabiliste (suite)

- On peut noter que le nombre maximum de clauses satisfaisables est aussi majoré par cette valeur
- On peut montrer qu'une clause C_j comportant k variables est alors satisfaite avec une probabilité supérieure ou égale à :

$$(1 - (1 - \frac{1}{k})^k)\hat{z_j}$$

- où $\hat{z_j}$ est la solution trouvée dans la solution du problème linéaire

Conclusion

Théorème La méthode de la programmation linéaire (relaxée) suivie de l'arrondi probabiliste donne une solution qui en moyenne est $(1 - \frac{1}{e})$ approchée

- On a vu que le nombre maximum de clauses satisfaisables est aussi majoré par :

$$\hat{z_1} + \hat{z_2} + \ldots + \hat{z_m}$$

- et qu'une clause C_j comportant k variables est alors satisfaite avec une probabilité supérieure ou égale à :

$$(1 - (1 - \frac{1}{k})^k)\hat{z_j}$$

- En moyenne sur la clause est donc satisfaite $(1 - (1 - \frac{1}{k})^k)\hat{z_j}$ fois
- en additionnant on trouve donc : $(1 - (1 - \frac{1}{k})^k) \quad \leq (1 - \frac{1}{e})$ fois la valeur de l'optimum.

Echantillonnage

Principe

- On traite le problème à l'aide d'un sous ensemble de l'ensemble des données

Exemple :

- On se donne un ensemble de nombre $a_1, a_2, \ldots, a_n$ où n est très grand, on veut trouver un a_m qui soit plus grand que plus de la moitié des $a - i$
- Algorithme naïf en $O(n)$: on prend le plus grand parmi les $\frac{n}{2} + 1$ premiers éléments
- Meilleur on choisi $\sqrt{n}$ élements au hasard et on prend le plus grand. Probabilité d'echec $(\frac{1}{2})^{\sqrt{n}}$

Les points les plus proches

Données

- Un ensemble de points du plan (ou d'un espace de dimension supérieure à 2) :

$$A_1, A_2, \ldots A_n$$

Résultat

- Deux points A_p et A_q tels que $d(A_p, A_q)$ soit minimum
- Algorithme naïf en $n^2 d$ où d est la dimension
- Problème si n est très grand et si on a un nombre considérable de configurations à traiter

Principe

Diviser en sous-domaines contenant un faible nombre de points

- Chercher les deux points les plus proches dans chaque sous-domaine
- On compare les minima des sous-domaines
- Commment trouver les sous-domaines pour que :

$$\sum_{i=1}^{k} n_i^2 + k = O(n)$$

Difficultés

- Les points les plus proches ne sont pas nécessairement dans le même sous-domaine

  $\Rightarrow$ 4 grilles

- Trouver un pas de grille proche du δ le plus petit
- Répeter deux fois l'algorithme de divison en sous-domaines
- Pour un petit nombre de points appliquer l'algorithme de calcul exhaustif

Résultat principal

Théorème Soit un ensemble S de n points répartis uniformément dans le plan, et soit S_1 un sous-ensemble de S formé de $n^{2/3}$ points déterminés au hasard, si on choisit δ_1 par :

$$\delta_1 = \min_{i,j \in S_1} d(A_i, A_j)$$

Alors les nombres de points n_i dans chaque carré de coté δ_1 satisfont (en moyenne) :

$$\sum n_i^2 = O(n)$$

Conséquences pour obtenir δ_1 on itére le résultat ce qui permet de passer de $n^{2/3}$ points à $n^{4/9}$ pour ces $n^{4/9}$ points on procède en comparant tous les couples

Hachage parfait

Etant donné un univers $U = \{1, 2, \ldots, N\}$ de *clés* et un sous-ensemble X de U formé de n clés, on cherche une application injective $X \longrightarrow \{0, 1, 2, \ldots, m-1\}$ de la forme :

$$h_k(x) \quad = \quad (kx \bmod N) \bmod m,$$

où k est un entier à choisir compris entre 1 et $N - 1$

Collisions

A chaque fonction h_k on associe sa mesure de collision : soit $c(i,k)$ le nombre de clés d'image i par h_k. Soit

$$\gamma(k) = \sum_{i=0}^{m-1} c(i,k)^2 \quad - n.$$

Alors h_k est une injection si et seulement si

$$\gamma(k) = 0$$

Lemme : Si N est premier alors on a :

$$\sum_k \gamma(k) \leq \frac{2n(n-1)(N-1)}{m}$$

Conséquence : Si k est tiré au hasard alors :

$$E(\gamma(k)) \leq \frac{2n(n-1)}{m}$$

et donc par Markov :

$$Pr(\gamma(k) \leq \frac{4n(n-1)}{m}) > \frac{1}{2}.$$

Algorithme

- `Repeter`
  - Choisir k au hasard dans $1, \ldots N-1$
  - jusqu'à $\gamma(k) < \frac{4n(n-1)}{m}$
- Le temps moyen de calcul est $2n$
- On trouve en moyenne le résultat après deux essais

Remarques

- On a un hachage parfait si $m > 4n^2$
- Pour avoir une place en $O(n)$ on procède en deux étapes on commence par prendre $m = 4n$ puis on refait un hachage dans les cases où c'est nécessaire, mais avec nos hypothèses les nouveuax hachage nécessite une place en $O(n)$

Routage sur l'hypercube

Dans un réseau représenté par un graphe $G = (X, E)$, chaque sommet i détient un message M_i qu'il doit envoyer à un autre sommet $\sigma(i)$

- Un algorithme de routage consiste à choisir à chaque étape un certain nombres de messages qui vont passer d'un sommet à l'un de ses voisins
- Au cours d'une étape deux messages ne peuvent pas emprunter la même arête
- Il s'agit de minimiser le nombre d'étapes
- On suppose que σ est une permutation (pour simplifier les calculs).

Graphe de l'Hypercube

- Les sommets sont toutes les suites de n bits
- Une arête entre chaque couple de sommets qui diffèrent sur 1 bit

Algorithme déterministe

- A chaque étape un message cherche à se rapprocher de son but en modifiant le bit le plus à gauche du sommet qui diffère de celui du but
- Si deux messages veulent emprunter la même arête on donne priorité à celui qui attend depuis le plus long temps
- Les ex aequo sont départagés au hasard

Algorithme probabiliste

1. **Initialisation** Chaque sommet détermine au hasard une destination intermédiaire $\tau(i)$
2. **Phase 1** On applique l'algorithme déterministe pour aller de chaque i à $\tau(i)$
3. **Phase 2** On applique l'algorithme déterministe pour aller de chaque $\tau(i)$ à $\sigma(i)$

Lemme 1 Deux chemins déterminés par l'algorithme déterministe se rencontrent sur une partie constituée d'arêtes consécutives

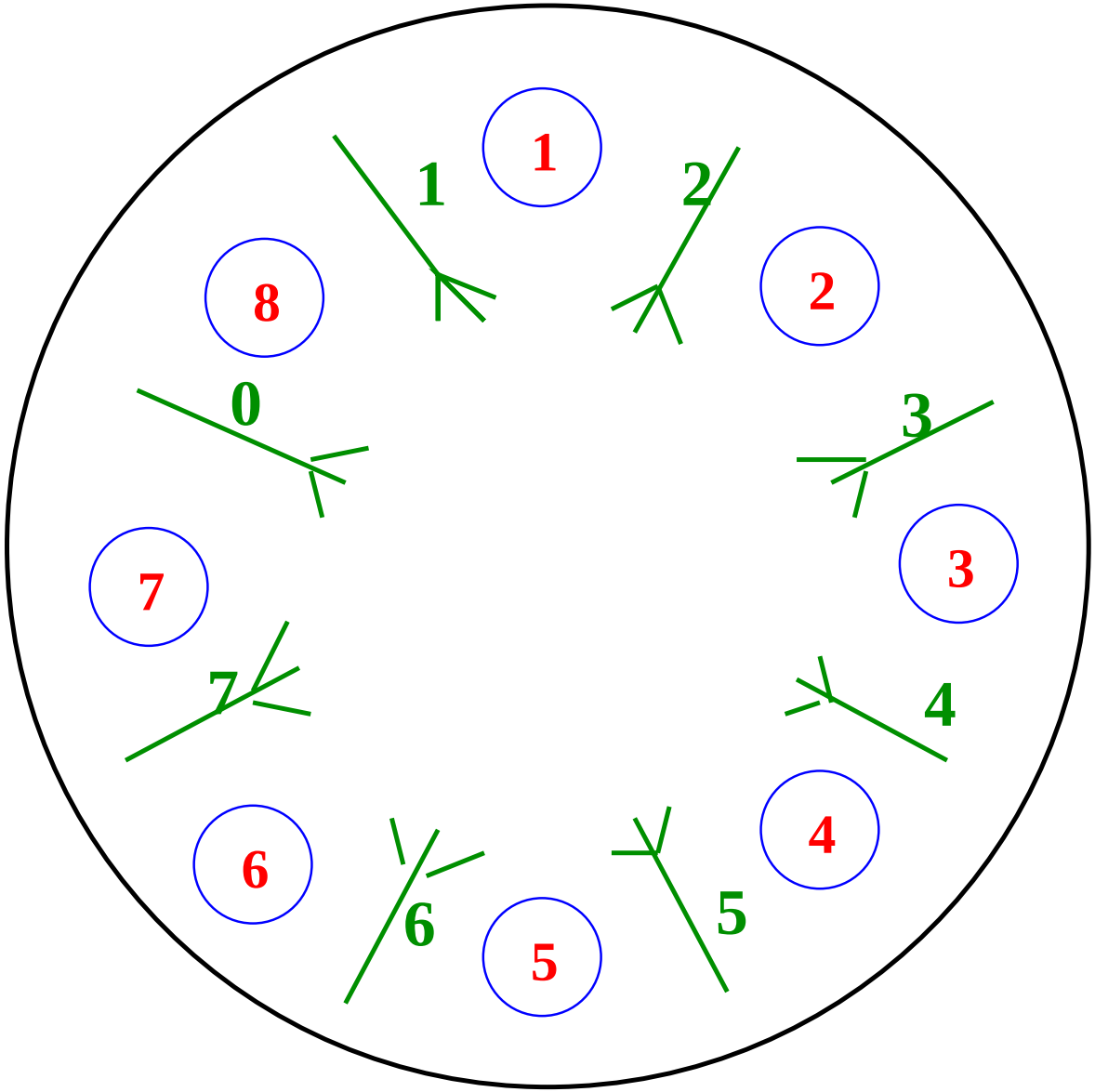
Lemme 2 Pour une phase le retard accumulé par chaque message est inférieur ou égal au nombre de messages différents qui empruntent une même arête que lui.

Nombre moyen d'étapes

- $H_{i,j}$ une variable aléatoire égale à 1 si les chemins de i à $\tau(i)$ et de j à $\tau(j)$ se rencontrent
- Le nombre total d'arêtes est $Nn = 2^n n$
- Le nombre moyen de chemins empruntant une arête donnée est $1/2$
- La valeur moyenne de la somme des $H_{i,j}$ est inférieure ou égale à $n/2$.
- La probabilité que cette variable dépasse $(\theta + 1)n$ est au plus $2^{-(\theta+1)n}$
- Avec probabilité $1 - \frac{2}{N^\theta}$ chaque message atteint sa destination temporaire en moins de $(\theta + 2)n$ étapes.
- Pour la destination finale il faut multiplier par 2 le nombre d'étapes

Le problème des philosophes

- Des philospohes sont assis autour d'une table, une assiette de spaghetti devant eux
- Chacun partage une fourchette avec son voisin de droite et une autre avec son voisin de gauche
- Il peut, vérifier si une fourchette est libre et s'en saisir, ou en relacher une
- Il mange s'il détient une fourchette dans chaque main



Philosophes autour d'une table

Problème

Trouver un algorithme pour chaque philosophe tel que :

- Deux philosophes ne détiennent pas ensemble la même fourchette Correction
- Chaque philosophe mange au bout d'un temps raisonable vivacité
- Pas de solution symétrique!

Une solution incorrecte

1. Attendre que la fourchette de gauche soit libre, la saisir
2. Attendre que la fourchette de droite soit libre, la saisir
3. Manger
4. Reposer les deux fourchettes

Algorithme probabiliste

Pincipe intuitif de l'algo :

- Chaque philosophe tire au sort la fourchette dont il va se saisir en premier
- Si elle est libre il la prend, sinon il attend qu'elle soit libre
- Une fois qu'il obtient une fourchette, il examine si l'autre est libre
- Si tel est le cas il la prend sinon il repose la première

Algorithme

1. **Variables partagées** `f[i]` initialisée à `true` accessible en lecture-écriture par les processus P_i et P_{i-1} par l'instruction `Test-And-Set`
2. **Algorithme du processus i :**

   `do forever`

   - `first = random(0,1);`
   - `wait until f[i+first]`
   - `f[i+first] = false;`
   - `if (f[i+1-first]) then`
     - `f[i+1-first] = false`
     - `go to` Critical
   - `else f[i+first] = true`

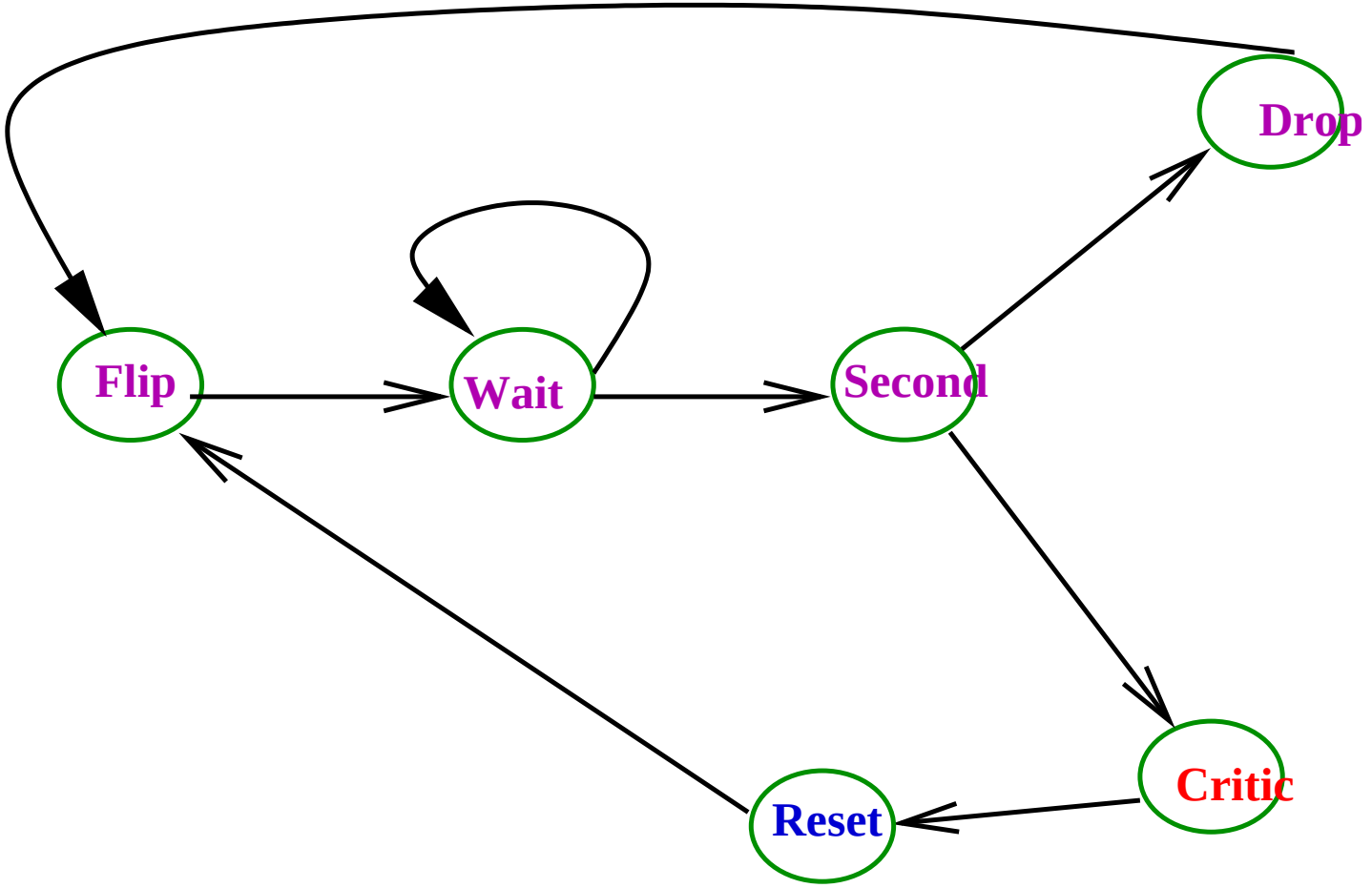
Analyse de l'algorithme

- Il existe des situations de blocage mais qui interviennent avec une très faible probabilité
- Si ℓ est le temps maximum d'exécution d'une étape, on montre que la probabilité pour que l'on arrive en temps 10ℓ à un état dans lequel au moins un philosophe mange est supérieur ou égale à

$$\frac{1}{16}$$

Modélisation

1. Les fourchettes sont des variables partagées $f[i]$ accessibles par P_i et P_{i-1}
2. Les états : Flip, Wait, Second, Drop, Critical, Reset
3. Transition : changement d'état, modification des variables partagées
4. Modèle avec des automates


Drop
Flip
Wait
Second
Critic
Reset

Etats Locaux

- Etats d'essai:

$$T = \{Flip, Wait, Second, Drop\}$$

- $\overleftarrow{W}$ Etat W avec attente de la fourchette à gauche (`first = 1`)
- $\overleftarrow{S}$ Etat S avec en main la fourchette gauche
- $\overleftarrow{D}$ Repose la fourchette gauche car celle à droite n'est pas libre
- $\overrightarrow{W}, \overrightarrow{S}, \overrightarrow{D}$ Les symétriques gauche/droite

Comportements globaux

- $\mathcal{T}$ un des processus est dans l'état: $= \{Flip, Wait, Second, Drop\}$
- $\mathcal{RT}$ tous les processus sont dans l'un des états: $= \{F, W, S, D\}$
- $\mathcal{C}$ un des processus est dans l'état: $Critical$ il tient les deux fourchettes
- $\mathcal{F}$ un des processus est dans l'état: $Flip$

Les bons états

- $\mathcal{G}$

  $\exists i$ tel que P_i a une fourchette en main et l'autre est pour l'instant libre. Soit plus précisement:

- $$Etat(P_i) \in \{\overleftarrow{W}, \overleftarrow{S}\} \quad Etat(P_{i-1}) \in \{F, \overrightarrow{W}, \overrightarrow{S}, \overrightarrow{D}\}$$

  ou :

- $$Etat(P_i) \in \{\overrightarrow{W}, \overrightarrow{S}\} \quad Etat(P_{i+1}) \in \{F, \overleftarrow{W}, \overleftarrow{S}, \overleftarrow{D}\}$$

- On note

$$\mathcal{U} \xrightarrow{t}_{p} \mathcal{V}$$

si on passe d'un état de $\mathcal{U}$ à un état de $\mathcal{V}$ en moins de t étapes avec une probabilité supérieure ou égale à p. On va montrer :

$$\mathcal{RT} \xrightarrow{10}_{1/16} \mathcal{C}$$

Lemme 1

$$\mathcal{RT} \xrightarrow{3}_{1} \mathcal{C} \cup \mathcal{F}$$

- Si l'un des processus est dans l'état S en une ou deux étapes il atteint soit F soit C
- Si l'un des processus est dans l'état D il atteint F en une étape
- Si tous les processus sont dans l'état W, aucun n'a de fourchette,le premier a regarder si elle est libre la prendra, passera dans l'état S en une étape, puis à F ou C en deux autres.

Lemme 2

$$\mathcal{F} \xrightarrow{2}_{1/4} \mathcal{G} \cup \mathcal{C}$$

Preuve: Soit i tel que $etat(P_i) = F$

- Cas 1: $Etat(P_{i-1}) \in \{F, \overrightarrow{W}, \overrightarrow{S}, \overrightarrow{D}\}$.
  Alors avec probabilité $\frac{1}{4}$ P_i choisit la gauche et le prochain choix de P_{i-1} est la droite. Alors dans les deux étapes qui suivent, si P_{i-1} ne détient pas sa fourchette de gauche $etat(P_i) \in \mathcal{G}$ sinon $etat(P_{i-1}) \in \mathcal{C}$
- Cas 2: $Etat(P_{i+1}) \in \{F, \overleftarrow{W}, \overleftarrow{S}, \overleftarrow{D}\}$. Symétrique
- Cas 3: $Etat(P_{i-1}) \in \{\overleftarrow{W}, \overleftarrow{S}, \overleftarrow{D}\}$ $Etat(P_{i+1}) \in \{\overrightarrow{W}, \overrightarrow{S}, \overrightarrow{D}\}$.

Lemme 3

$$\mathcal{G} \xrightarrow{5}_{1/4} \mathcal{C}$$

Preuve:

Composition

On a vu

- $$\mathcal{RT} \xrightarrow{3\ell}_1 \mathcal{F} \cup \mathcal{C}$$
- $$\mathcal{F} \xrightarrow{2\ell}_{1/4} \mathcal{G} \cup \mathcal{C}$$
- On vérifie
$$\mathcal{G} \xrightarrow{5\ell}_{1/4} \mathcal{C}$$

Donc

$$\mathcal{RT} \xrightarrow{10\ell}_{1/16} \mathcal{C}$$