Algorithme probabiliste

On parle **d'algorithme probabiliste (ou algorithme randomisé)** dès que celui-ci utilise une source de hasard : dès que le déroulement de l'algorithme fait appel à des données tirées au hasard, dès qu'on fait appel à la fonction *random*. Par opposition, un algo n'étant soumis à aucun hasard est dit déterministe.

La méthode PERT, la méthode du simplexe, l'algorithme Branch and Bound sont déterministes.

Le recuit simulé est un algo probabiliste. L'algo génétique le devient rapidement (dès qu'on tire au sort les gènes retenus des 2 parents ou qu'il y a de l'aléatoire dans la composition de la population initiale, ...)

Un algo probabiliste exécuté 2 fois sur le même jeu de données peut rendre deux réponses différentes.

Pourquoi des algos probabilistes ?

De nombreux algorithmes nécéssitent naturellement des tirages aléatoires, que ce soit pour la constitution des données ou paramètres ou pour la résolution même du problème. Mais de plus, le hasard permet souvent de défaire l'adversaire.

- de nombreuses applications en crypto avec un adversaire qui essaye de "craquer" le code,
- de nombreuses applications dans les jeux à deux joueurs avec un adversaire qui gagnera facilement s'il finit par comprendre ce que fait votre algo déterministe (exemple : Pierre, Feuille, Ciseau)
- permet souvent de baisser la complexité dans le pire des cas de l'algorithme - le pire des cas revient à retenir la configuration la plus déforable à l'algo comme si un adversaire allait trouver le cas le plus nefaste.

Vous jouez dans quel casino ?

Parmi les algos probabilistes, on distingue ceux de Monte-Carlo, ceux de Las-Vegas et ceux d'Atlantic City.

- Un algo de **Las Vegas** donne toujours le même résultat exact, mais le temps de calcul varie d'une exécution à l'autre. On espère ce temps petit avec une forte probabilité.
- Le résultat d'un algo de **Monte-Carlo** peut être incertain ou approché dans certains cas et sûr ou exact dans d'autres. On espère la probabilité du premier cas ou l'erreur commise faible. La complexité en temps doit être faible, certainement polynomiale.
- Un algo d'**Atlantic-City** est un mélange des deux : très forte proba sur l'exactitude de la réponse avec temps de calcul en moyenne faible.

Monte-Carlo - exemple 1 : on cherche un homme

On a en entrée un très gros tableau ($N = 100000$). Chaque case correspond à un individu. Les $\frac{3}{4}$ des enregistrements sont des hommes. On veut récupérer les données d'un homme.

Algo déterministe :

$i = 0$; Tant Que $(T[i].sexe \; != \; H)$ Faire $i++$;

Complexité très bonne en moyenne - Mais dans le pire des cas, on doit lire un quart du tableau (25000 cases).

Monte-Carlo :

Pour $j = 1..K$ faire $i = rand(N)$; Si $(T[i].sexe == H)$ break

Si $(j == K + 1)$ Alors Ecrire("Echec")

Le résultat n'est pas sûr. La probabilité d'échec est de $\frac{1}{4}^K$. Mais cette proba devient très vite très faible dès que K croit.

Pour $K = 5$ Prob echec $< 10^{-3}$, pour $K = 17$ Prob echec $< 10^{-10}$

Donc très faible probabilité d'echec et complexité en temps en $O(K)$, donc polynomiale et très petite.

Monte-Carlo - exemple 2 : Test de primalité

Le problème est de reconnaitre si un nombre est premier ou non avec de nombreuses applis en crypto sur des nombres très grands

L'algo déterministe est polynomial mais très couteux en temps de calcul et pas vraiment applicable pour les tailles généralement visées.

Propriété arithmétique (non démontrée) : cas particulier du petit théorème de Fermat
Si un entier n est premier alors $\forall a < n$, $a^{n-1}\ mod\ n = 1$

D'où le test de primalité de Miller-Rabin :
$i = 0$; $Res = 1$; Tant Que $(Res\ \&\&\ i < K)$ Faire
$i++$; Choisir a aléatoirement $(1 < a < n)$; Si $a^{n-1}\ mod\ n\ != 1$ Alors
$Res = 0$;

Monte-Carlo - exemple 2 : Test de primalité

- Le calcul de a^{n-1} se fait par exponentiation rapide,
- Si $Res = 0$ Alors on est sur que n n'est pas premier.
- Si $Res = 1$ on n'est pas sur de la primalité de n
- *Propriete 2* : Si n non premier et impair, au moins $\frac{3}{4}$ des entiers inférieurs à n sont des témoins de non-primalité.
- Meme proba d'échec que dans l'exemple 1

Monte-Carlo - exemple 3 : calcul d'une aire

Le problème est de calculer l'intégrale sur $[0, 1]$ d'une fonction f qui a été normalisée et dont les valeurs sont aussi sur $[0, 1]$ On suppose le calcul de l'intégrale de cette fonction très complexe.

Un algorithme simple probabiliste peut etre le suivant :
On tire aléatoirement deux points dans $[0, 1]$ (x_1, x_2). Si le point est sous la courbe $(x_2 < f(x_1))$, il fait partie de l'aire recherchée. Du coup, le rapport entre le nombre de points sous la courbe sur le nombre de points tirés au total tend vers la valeur de l'integrale.

C'est un algorithme de Monte-Carlo, puisqu'il est probabiliste, que le résultat n'est pas exact et que la complexité en temps de calcul est faible. Plus on tire de points, plus on est capable de borner l'erreur commise sur la valeur de l'integrale (je vous passe les calculs mathématiques). Par contre, la vitesse de convergence est ici bien plus lente que dans les deux exemples precedents.

Las Vegas - exemple 1 : QuickSort

Rappel de l'algorithme de tri rapide : pour simplification, on suppose toutes les valeurs du tableau à trier distinctes.

fonction QuickSort(S)

- Si $(Card(S) \leq 1)$ Alors Renvoyer(S) Sinon
  - Prendre p le premier element de S
  - Determiner $S_1 = \{s \in S \; tels \; que \; s < p\}$
  - Determiner $S_2 = \{s \in S \; tels \; que \; s > p\}$
  - Renvoyer(QuickSort(S_1),p,Quicksort(S_2))

La complexité de l'algorithme dépend des valeurs p. Plus elles sont médianes et séparent en deux sous-ensembles S_1 et S_2 de tailles equivalentes, plus la complexité est bonne.

Las Vegas - exemple 1 : QuickSort

Le meilleur cas est celui ou $Card(S_1) = Card(S_2)$. Complexite en $O(n.logn)$.

Le pire cas est celui ou, à chaque coup, S_1 ou S_2 est vide : p est toujours le plus petit ou le plus grand element de S.
La complexite est alors en $n + (n-1) + (n-2) + ... = O(n^2)$. Dans ce cas, Quicksort s'avere un mauvais algo de tri.

Quelle que soit la stratégie déterministe de choix du pivot, il existe une entrée S qui exige $O(n^2)$ comparaisons. Un "attaquant" pourrait en tirer profit (voir "A killer adversary for Quicksort, Mcliroy, 99").

Las Vegas - exemple 1 : QuickSort randomisé

On choisit aléatoirement le pivot :

fonction QuickSort(S)

- Si $(Card(S) \leq 1)$ Alors Renvoyer(S) Sinon
  - **Choisir aleatoirement p dans S**
  - Determiner $S_1 = \{s \in S \; tels \; que \; s < p\}$
  - Determiner $S_2 = \{s \in S \; tels \; que \; s > p\}$
  - Renvoyer(QuickSort(S_1),p,Quicksort(S_2))

Le hasard permet de défaire l'adversaire.

Las Vegas - exemple 1 : QSort randomisé

Complexité = Nombre de comparaisons effectuées.
$C_{ij} = 1$ si les éléments de rang i et j sont comparés (et 0 sinon).

$$E(C) = \sum_{i,j>i} E(C_{ij}) = \sum_{i,j>i} \frac{2}{j-i+1}$$
$$E(C) = \sum_i \sum_{k=1}^{n-i+1} \frac{2}{k} \leq 2 \sum_i \sum_{k=1}^{n} \frac{1}{k} = 2nH_n$$

H_n : série Harmonique en $O(logn)$. Donc $E(C)$ est en $O(n.logn)$.
La performance de QS randomisé est la même que celle du QS deterministe en moyenne.

Las Vegas - exemple 2 : Election

La problématique de l'election (très utile en algorithmique distribuée) est la suivante :
On a n éléments (machines) et on veut qu'une seule d'entre elle soit élue. Mais pas d'oracle extérieur qui peut tirer au hasard entre 1 et n qui est l'élue : toutes les machines doivent avoir le même code.

Un algo LV pour l'election : A un tour donné, chacun des n individus tire au hasard un nombre entre 1 et n. S'il n'y a aucun 1 tiré, on recommence. S'il n'y a qu'un seul 1 tiré, on a notre chef. S'il y a plusieurs 1 tirés, on recommence avec les individus qui ont tiré 1.

- Quand l'algorithme se termine, le résultat est correct et on a bien 1 seul chef : l'algorithme fournit toujours le résultat escompté
- On montre assez facilement que la probabilité que l'algo n'ait pas terminé après L étapes décroit vite et tend vers 0 quand L grandit.
- Par contre, le temps de l'algorithme est incertain : on montre qu'il est petit avec une forte proba, mais avec une certaine malchance, cela peut durer très longtemps