INF561: Utilisation de l'aléatoire en algorithmique **Hiver 2013**

Cours 5 — 4 février

Enseignant : Frédéric Magniez *Rédacteur : Svyatoslav Covanov*

5.1

5.1.1 Test d'associativité

Problème : $S = \{1, ..., n\}$
Input : $o : S \times S \longrightarrow S$ donné par une procédure
Output : Décider si o est associative, c'est-à-dire $\forall i, j, k \in S, (ioj)ok = io(jok)$

La complexité correspond au nombre d'évaluation de o.

1. **Remarque :** l'algorithme déterministe s'effectue en n^3 opérations 'o'.
2. Si l'on fixe $p = 7$, $\overrightarrow{S} = \{A \in (\mathbb{Z}/p\mathbb{Z})^n\}$ et $e_i = \begin{pmatrix} 0 \\ ... \\ 1 \\ ... \\ 0 \end{pmatrix} \longleftarrow i$

On définit $\overrightarrow{o}$:
$$\begin{array}{ccccccc} \overrightarrow{S} \times \overrightarrow{S} & \longrightarrow & \overrightarrow{S} & & & \\ A, B & \longrightarrow & C & = & A \overrightarrow{o} B & \\ & & & = & \sum_{i,j} A_i B_j (\overrightarrow{e_{ioj}}) & \end{array}$$

Propriété 1. *(S, o) est associative si et seulement si $(\overrightarrow{S}, \overrightarrow{o})$ est associative.*

Preuve: $\Longleftarrow$ Soient i, j, k. On a $\overrightarrow{e}_{(ioj)ok} = (\overrightarrow{e}_i o \overrightarrow{e}_j) o \overrightarrow{e}_k = \overrightarrow{e}_i o (\overrightarrow{e}_j o \overrightarrow{e}_k) = \overrightarrow{e}_{io(jok)}$.
Donc $(ioj)ok = io(jok)$.
$\Longrightarrow$ Réciproquement, prenons A, B, C.
On a : $Ao(BoC) = \sum_{i,j,k} A_i B_j C_k e_{io(jok)} = \sum_{i,j,k} A_i B_j C_k e_{(ioj)ok} = (AoB)oC$ □

Propriété 2. *Si (S, o) non associative alors $\mathbb{P}_{\overrightarrow{A}, \overrightarrow{B} \in \overrightarrow{S}}(\overrightarrow{A} \overrightarrow{o} (\overrightarrow{B} \overrightarrow{o} \overrightarrow{C}) = (\overrightarrow{A} \overrightarrow{o} \overrightarrow{B}) \overrightarrow{o} \overrightarrow{C})) \leq \frac{3}{8}$*

Preuve: Choisissons A, B, C.
On a : $Ao(BoC) = \sum_{t \in S} (\sum_{i,j,k \in S, io(jok)=t} A_i B_j C_k) e_t$.
Alors $P = Ao(BoC) - (AoB)oC = \sum_t (\sum_{i,j,k \in S, io(jok)=t} A_i B_j C_k - \sum_{i,j,k \in S, (ioj)ok=t} A_i B_j C_k) e_t$.

On a donc en chaque e_t un polynôme en les variables (A_i, B_j, C_k) de degré inférieur à 3.
Si o n'est pas associative, il existe i_0, j_0, k_0 tels que $t = i_0 o (j_0 o k_0) \neq (i_0 o j_0) o k_0$.
La t-ème coordonnée de P est alors non nulle pour $A = e_{i_0}, B = e_{j_0}, C = e_{k_0}$.
Donc par le lemme de Schwartz-Zippel, $\mathbb{P}_{A,B,C \in (\mathbb{Z}_p)^n}[P_t(A, B, C) = 0] \leq \frac{3}{p}$.
Donc : $\mathbb{P}_{A,B,C \in (\mathbb{Z}_p)^n}[Ao(BoC) = (AoB)oC] \leq \frac{3}{p}$. □

Algorithme : On choisit $A, B, C \in (\mathbb{Z}_p)^n$ et on accepte si $(AoB)oC = Ao(BoC)$.

Propriété 3. *1. Si o est associative, l'algorithme accepte avec probabilité 1.*

2. Si o non associative, l'algorithme accepte avec probabilité inférieure à $\frac{3}{8}$.

3. La complexité est de $2n^2$ opérations o avec les additions et les multiplications modulo 7.

5.1.2 Application : 2-coloriage d'un graphe 3-coloriable

Soit $G = (X, E)$ un graphe non-orienté *3-coloriable*, c'est-à-dire qu'il existe une application $C : X \to [\![0; 2]\!]$ telle que

$$\forall\, (x, y) \in X^2,\ (x, y) \in E \Rightarrow c(x) \neq c(y)$$

On cherche un algorithme qui colorie G avec seulement 2 couleurs, de telle sorte qu'aucun triangle ne soit mono-chromatique. Ceci est toujours possible pour un graphe 3-coloriable : en effet, il suffit de "fusionner" deux couleurs pour obtenir un graphe bicolore dans lequel aucun triangle n'est monochromatique.

On remarque qu'à un triangle (u, v, w), on peut associer une clause $C_{u,v,w}$ vraie si, et seulement si, le triangle (u, v, w) n'est pas monochromatique. En effet, si $c_u \in \{0, 1\}$ est la couleur du sommet u, alors (u, v, w) n'est pas monochromatique si, et seulement si, $C_{u,v,w} = 1$ avec :

$$C_{u,v,w} = \neg(a_u = a_v = a_w) = (a_u \vee a_v \vee a_w) \wedge ((\neg a_u) \vee (\neg a_v) \vee (\neg a_w))$$

Ceci ramène la résolution du problème à un problème 3-SAT. Dans ce cas particulier, l'algorithme est cependant polynomial :

2-coloration d'un graphe 3-coloriable

1. Choisir une coloration $a : G \to \{\text{Bleu, Vert}\}$ quelconque
2. Tant qu'il existe un triangle monochromatique dans G colorié par a :

   Choisir un triangle monochromatique

   Changer la couleur de l'un des sommets au hasard
3. Retourner a

Soit $b : G \to \{\text{R, B, V}\}$ une 3-coloration du graphe. Pour $i \in \mathbb{N}$, on note X_i le cardinal, après i itérations de la boucle, de l'ensemble :

$$\{u \in V \ /\ (b(u) = B \wedge a(u) = V) \vee (b(u) = V \wedge a(u) = B)\}$$

Soit (u, v, w) un triangle monochromatique à l'étape i, par exemple $a(u) = a(v) = a(w) = B$. On peut de plus supposer, sans perte de généralité, que $b(u) = R, b(v) = V, b(w) = B$. Alors :

– Avec probabilité $\frac{1}{3}$, l'algorithme change la couleur de u : $a(u) = V$. On a alors : $X_{i+1} = X_i$;

– Avec probabilité $\frac{1}{3}$, l'algorithme change la couleur de v : $a(v) = V$. On a alors : $X_{i+1} = X_i - 1$;
– Avec probabilité $\frac{1}{3}$, l'algorithme change la couleur de w : $a(w) = V$. On a alors : $X_{i+1} = X_i + 1$.

Le temps nécessaire pour parcourir tout le graphe des possibilités est inférieur à $4|V||E| = 12(n+1)^2$.

5.2 Problème 3-SAT

5.2.1 Première tentative : Application de WalkSAT

Algorithme de Schoening pour 3-SAT

1. Choisir arbitrairement $a \in \{0,1\}^n$
2. Tant que $\phi(a) = 0$, répéter :
   Choisir une clause C_j telle que $C_j(a) = 0$
   Changer la valeur d'une variable de C_j au hasard
3. Renvoyer a si $\phi(a) = 1$, sinon rejeter

Si ϕ est 2-SAT et $T = 8kn^2$ alors :

1. Si ϕ est non satisfaisable alors l'algorithme rejette avec probabilité 1.
2. Si ϕ est satisfaisable alors l'algorithme renvoie a tel que $\phi(a) = 1$ avec probabilité supérieure à $1 - \frac{1}{2}^k$

Pour tout $i \in \mathbb{N}$, notons X_i le nombre de bits sur lesquels a et s diffèrent après i itérations. On a alors :

$$\mathbb{P}[X_{i+1} = n-1 | X_i = n] = 1$$
$$\mathbb{P}[X_{i+1} = 0 | X_i = 0] = 1$$
$$\forall 1 \leq j \leq n-1, \qquad \mathbb{P}[X_{i+1} = j-1 | X_i = j] \geq \frac{1}{3}$$
$$\forall 1 \leq j \leq n-1, \qquad \mathbb{P}[X_{i+1} = j+1 | X_i = j] \leq \frac{2}{3}$$

De même que dans l'analyse de l'algorithme pour 2-SAT, on considère donc le graphe linéaire dont l'ensemble des états est $\{0, 1, \ldots, n\}$, avec les probabilités de transition :

$$\forall (n,m), \qquad \mathbb{P}_{n \to m} = \mathbb{P}[X_{i+1} = m | X_i = n]$$

L'analyse du nombre d'itérations dans l'algorithme modifié se ramène au calcul du temps moyen pour atteindre 0 en effectuant une marche aléatoire sur le graphe précédent, en partant de l'état n, que l'on note h_n. Dans le pire des cas [1], ce temps moyen est donné par la relation de récurrence :

1. c'est-à-dire dans le cas où $\mathbb{P}_{n \to n+1} = \frac{2}{3}$ et où $\mathbb{P}_{n \to n-1} = \frac{1}{3}$

$$\forall 1 \leq j \leq n-1, \qquad h_j = 1 + \frac{1}{3}h_{j-1} + \frac{2}{3}h_{j+1}$$

avec les conditions aux limites $h_0 = 0$, $h_n = 1 + h_{n-1}$.

On obtient ainsi un temps moyen qui exponentiel :

$$h_n = \Theta(2^n)$$

Cette "explosion", par rapport à 2-SAT [2], du temps de calcul moyen, est liée au fait que la probabilité de s'éloigner de 0 est ici supérieure à la probabilité de s'en rapprocher. En parcourant le graphe aléatoirement, on aura naturellement tendance à s'éloigner de l'origine à long terme.

5.2.2 Une amélioration du temps de calcul moyen

Etant donné que cette tendance à s'éloigner de l'état 0, on peut penser qu'au lieu de pousser la marche aléatoire jusqu'à tomber sur l'état 0, on a intérêt à repartir, de manière régulière, d'une position aléatoire. D'où le nouvel algorithme :

ALGORITHME (WALKSAT MODIFIÉ 2)

1. $a \leftarrow (0, \ldots, 0)$
2. Tant que $\phi(a) = 0$, répeter
   (a) Choisir aléatoirement $a \in \{0,1\}^n$
   (b) Tant que $\phi(a) = 0$, répéter **au plus** $3n$ **fois** :
   Choisir une clause C_j telle que $C_j(a) = 0$
   Changer la valeur d'une variable de C_j au hasard
   Si $\phi(a) = 1$ renvoyer a sinon retourner en 1 au plus T fois.
3. Renvoyer a

Comme prévu, l'algorithme modifié est plus efficace que l'algorithme WALKSAT :

Théorème 5.1 (Schoening, 1999). *Dans l'algorithme précédent, le nombre moyen d'exécutions de la boucle 2 est majoré par :*

$$\Theta(\sqrt{n}(\frac{4}{3})^n)$$

Preuve: Soit :

$q = \mathbb{P}_{a_0}[\text{L'algorithme trouve une solution en } 3n \text{ itérations de la boucle (b), en partant de } a = a_0]$

Et :

$q_j = \mathbb{P}_{a_0}[\text{L'algorithme trouve une solution en } 3n \text{ itérations de la boucle (b), en partant de } a = a_0 | X_0 = j]$

2. On rappelle que pour 2-SAT, on avait $h_n = n^2$

On a :

$$q = \sum_j q_j \mathbb{P}[X_0 = j] = \sum_j q_j \frac{1}{2^n}\binom{n}{j}$$

Or :

$$q_j \geq \mathbb{P}_{a_0}[\text{L'algorithme trouve en au plus } 3j \text{ itérations, en partant de } a = a_0 | X_0 = j]$$

Or, partant de $X_0 = j$, et effectuant $2j$ transitions vers la gauche [3] et j transitions vers la droite, dans un ordre quelconque, on atteint forcément l'état 0 en au plus $3j$ itérations. Donc :

$$q_j \geq \mathbb{P}_{a_0}[\text{L'algorithme effectue } 2j \text{ transitions vers la gauche et } j \text{ transitions vers la droite}]$$

Cette dernière quantité se calcule facilement : la probabilité d'effectuer $2j$ transitions vers la gauche et j transitions vers la droite est $(\frac{1}{3})^{2j}(\frac{2}{3})^j$, et il y a $\binom{3j}{j}$ manières d'ordonner ces transitions. Donc :

$$q \geq \sum_j \frac{1}{2^n}\binom{n}{j}\frac{2^j}{3^{3j}}\binom{3j}{j}$$

Par la formule de Stirling, on a pour $j \neq 0$:

$$\binom{3j}{j} \quad \sim \quad \frac{1}{\sqrt{2\pi}}\sqrt{\frac{3}{2j}}\frac{3^{3j}}{2^{2j}}$$

Donc :

$$\begin{aligned}
q &\geq \Theta(1)[\frac{1}{2^n} + \sum_{j=1}^{n} \frac{1}{2^n}\binom{n}{j}\frac{1}{\sqrt{j}}\frac{1}{2^j}] \\
&\geq \Theta(1)[\frac{1}{2^n} + \sum_{j=1}^{n} \frac{1}{2^n}\binom{n}{j}\frac{1}{\sqrt{n}}\frac{1}{2^j}] \\
&\geq \Theta(1)[\frac{1}{2^n} + \frac{1}{2^n}\frac{1}{\sqrt{n}}(1+\frac{1}{2})^n] \\
&\geq \Theta[\frac{1}{\sqrt{n}}(\frac{3}{4})^n]
\end{aligned}$$

On conclut à l'aide du lemme suivant :

Lemme 5.2. *Soit $(z_k)_{k\in\mathbb{N}}$ une suite de variables aléatoire booléenne indépendante, telle que :$\forall i \in \mathbb{N}, \mathbb{P}[z_i = 1] = p$. Alors :*

$$\mathbb{E}[\min\{t/z_t = 1\}] = \frac{1}{p}$$

□

3. des transitions du type $k \to k-1$

Théorème 5.3. *Si $T = \frac{2k}{q}$ alors :*

1. *Si ϕ n'est pas satisfaisable alors l'algorithme rejette avec probabilité 1*
2. *Si ϕ a une solution, alors l'algorithme en renvoie une avec probabilité supérieure à $1 - (\frac{1}{2})^k$*

L'algorithme reste donc exponentiel, mais la constante sous l'exposant est meilleure que pour l'algorithme WALKSAT initial. Le même raisonnement donne, pour tout $k \geq 3$, un algorithme de complexité moyenne $\sqrt{n}(2 - \frac{1}{k})^n$.

5.3 Algorithmes de streaming

5.3.1 Modèle et motivation

Définition 5.4 (Masive data). *L'entrée x (Data stream) est trop grosse pour être écrite en mémoire RAM (Random Access Memory). Sa taille est* $o(n)$ *(sous-linéaire), idéalement* $O(\log(n))$.

- *Nécessite un accès séquentiel à l'entrée : on lit x par morceaux (un bit, un entier, une lettre, ... un élément de taille fixée petite).*
- *Une seule passe ou plusieurs possibles selon les cas (mais toujours un nombre constant de passes).*
- *A la fin des passes, l'algorithme doit calculer ou approcher une fonction.*

Exemple: Analyse de l'ADN, du graphe du WEB ... les données sont trop grandes pour être stoqués en mémoire RAM. Nombre constant de passes.

Exemple (Missing number):

Stream: suite de n entiers distincts de $[[1; n+1]]$

Sortie: trouver l'entier manquant

Contrainte: mémoire en $O(\log(n))$ bits, 1 passe

ALGORITHME

$s \leftarrow 0$

Tant que Stream non vide

    $x \leftarrow$ lire Stream

    $s \leftarrow s + x$

retourner $\frac{(n+1)(n+2)}{2} - s$

Définition 5.5 (Paramètres d'un algorithme de streaming).

paramètre	*valeur idéale*
nombre de passes	1 *ou* O(1) *passes*
mémoire RAM	$O(\log^{O(1)}(n))$ *bits*
temps par morceaux	$O(\log^{O(1)}(n))$ *opérations*
temps de post-processing (*pour donner la réponse après passage du stream*)	$O(\log^{O(1)}(n))$ *opérations*

Lemme 1. *Tout langage régulier peut être reconnu en une passe par un algorithme de streaming de mémoire constante.*

5.3.2 Moments de Fréquence

Définition 5.6.

Stream: $a_1, a_2, ..., a_n \in [[1, m]]$. *n est inconnu et m est connu*

Fréquences: $f_j = |\{i \in [[1; n]], a_i = j\}|, j \in [[1; m]]$

Moments: $F_k = \sum_{j=1}^{m} (f_j)^k$

- $F_0 = |\{j \in [[1; m]], f_j \neq 0\}|$
- $F_1 = n$
- F_2 = *"repeat rate" ou "surprise index". F_2 grand $\Rightarrow$ f_j anormalement grand (possibilité d'attaque).*
- $F_\infty = \max_j f_j$

Algorithme pour estimer F_1 :

Déterministe: espace en $O(\log(n))$ bits

Probabiliste: espace en $O(\log(\log(n)))$ bits

ALGORITHME

$a \leftarrow 0$

Tant que Stream non vide

    lire Stream

    $a \leftarrow a + 1$ avec probabilité $1/2^a$

retourner $2^a - 1$

Théorème 5.7. *Soit X_i la valeur de a après i éléments ($X_0 = 0, X_1 = 1$, ...)*

$$\mathbb{E}(2^{X_i}) = i + 1 \tag{5.1}$$

Preuve: Soit $\mathbb{P}_{i,j} = \mathbb{P}(X_i = j)$

$$\mathbb{E}(2^{X_i}) = \sum_j \mathbb{P}_{i,j} 2^j$$

$$\begin{aligned}
\mathbb{P}_{i,j} &= \mathbb{P}(X_i = j | X_{i-1} = j)\mathbb{P}_{i-1,j} + \mathbb{P}(X_i = j | X_{i-1} = j-1)\mathbb{P}_{i-1,j-1} \\
&= (1 - \frac{1}{2^j})\mathbb{P}_{i-1,j} + \frac{1}{2^{j-1}}\mathbb{P}_{i-1,j-1}
\end{aligned}$$

$$\begin{aligned}
\mathbb{E}(2^{X_i}) &= \sum_j (2^j - 1)\mathbb{P}_{i-1,j} + 2^{1-j}\mathbb{P}_{i-1,j-1} \\
&= \sum_j \mathbb{P}_{i-1,j} 2^j - \sum_j \mathbb{P}_{i-1,j} + 2\sum_j \mathbb{P}_{i-1,j-1} \\
&= \mathbb{E}(2^{X_{i-1}}) - 1 + 2 \\
\mathbb{E}(2^{X_i}) &= \mathbb{E}(2^{X_{i-1}}) - 1
\end{aligned}$$

□