Partie 2. Techniques algorithmiques

7. Algorithmes probabilistes

Bruno Grenet

Université Grenoble Alpes – IM²AG
L3 Mathématiques et Informatique
UE Algorithmique

https://membres-ljk.imag.fr/Bruno.Grenet/Algorithmique.html

Table des matières

1. Qu'est-ce qu'un algorithme probabiliste ?

2. Exemple 1 : coupe minimale

3. Exemple 2 : tri rapide

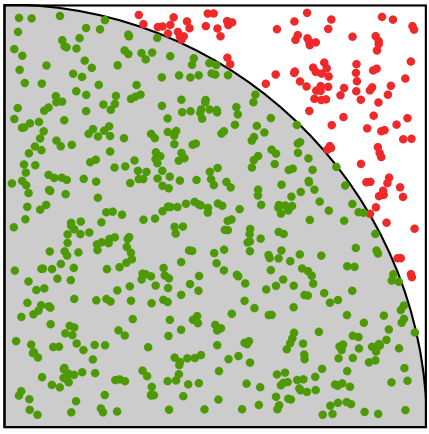
Table des matières

1. Qu'est-ce qu'un algorithme probabiliste ?

2. Exemple 1 : coupe minimale

3. Exemple 2 : tri rapide

Premier exemple : calculer π



MonteCarlo(N) :

1. $c \leftarrow 0$
2. Répéter N fois :
3. $\quad (x, y) \leftarrow$ *réels* aléatoires dans $[0, 1]$
4. $\quad$ Si $x^2 + y^2 \leq 1$: $c \leftarrow c + 1$
5. Renvoyer c/N

Propriété

$$\mathbb{E}[c/N] = \frac{\text{aire du } \frac{1}{4} \text{ de disque}}{\text{aire du carré}} = \frac{\pi}{4}$$

Remarque

- L'algorithme permet de calculer π
- Mais n'est pas la *bonne* façon de le faire

Deuxième exemple : trouver un nombre premier

Énoncé du problème

Entrée : entier N

Sortie : un nombre premier $p \geq N$

Échantillonage par rejet

1. $p \leftarrow 0$
2. Tant que p n'est pas premier :
3. $\quad p \leftarrow$ entier (impair) uniforme $\in [N, 2N]$
4. Renvoyer p

Propriétés

- Hypothèse : on sait tester si p est premier (sans erreur…)
- L'algorithme renvoie toujours un nombre premier
- L'espérance du nombre d'entiers tirés est $O(\log N)$
  - Théorème des nombres premiers $\Rightarrow \#\{p \in [N, 2N] : p \text{ premier}\} = \Omega(N/\log N)$

Troisième exemple : vérifier un produit de matrices

Énoncé du problème

Entrées : trois matrices $A, B, C \in \mathbb{K}^{n \times n}$

Sortie : est-ce que $A \cdot B = C$?

Objectif : Ne pas faire le produit $A \cdot B$ coût : $O(n^3)$ ou $O(n^\omega)$

Algorithme de Freivalds

1. $\vec{v} \leftarrow$ vecteur aléatoire uniforme $\in \{0,1\}^n$
2. $\vec{x} \leftarrow A \cdot (B \cdot \vec{v})$
3. $\vec{y} \leftarrow C \cdot \vec{v}$
4. Renvoyer $\vec{x} \stackrel{?}{=} \vec{y}$

Propriétés

- Complexité : 3 produits matrice-vecteur $\rightarrow O(n^2)$
- Correction :
  - Si $A \cdot B = C$, l'algo. renvoie VRAI
  - Sinon : *on peut montrer que* $\Pr[A \cdot B \cdot \vec{v} = C \cdot \vec{v}] \leq \frac{1}{2} \rightarrow$ renvoie FAUX avec proba. $\geq \frac{1}{2}$

Définition

Un **algorithme probabiliste** est un algorithme qui effectue des **choix aléatoires** au cours de son exécution.

Remarque

- Choix aléatoires : accès à un générateur de bits aléatoires, ou d'entiers aléatoires, etc.

Analyse d'un algorithme probabiliste

- Le comportement d'un algorithme probabiliste est une *expérience probabiliste*
- Le résultat renvoyé (*correction*) ou le nombre d'opérations effectuées (*complexité*) peuvent dépendre des choix aléatoires (parfois les deux)

Trois types d'algorithmes

1. Calcul de $\pi \rightarrow$ approximation numérique
2. Trouver un premier $\rightarrow$ toujours correct mais temps variable
3. Vérifier un produit $\rightarrow$ temps fixé mais parfois incorrect

Les algorithmes numériques

Algorithme probabiliste numérique

- Calcul d'une *valeur approchée* d'une quantité
- Souvent par échantillonnage

Analyse de l'algorithme

- Qualité de l'approximation
- En fonction du temps passé

Cadre plus général

- Algorithme d'approximation : calculer une valeur approchée *cf.* Cours 9
- Pas forcément de manière probabiliste !

Remarques

- Algorithmes souvent rencontrés en maths applis

Les algorithmes de type *Las Vegas*

Las Vegas : « toujours correct, souvent rapide »

Algorithme probabiliste dont

- le résultat ne dépend pas des choix aléatoires
- la complexité dépend des choix aléatoires

Analyse de l'algorithme

- Correction : standard
- Complexité :
  - Espérance du nombre d'opérations, modélisé par une variable aléatoire
  - *Bornes de concentration* : « la probabilité que la complexité dépasse $10n^2$ est $\leq \frac{1}{n}$ »

Signification de l'espérance de la complexité

« L'espérance de la complexité est $O(n^2)$ »

- On *s'attend* à avoir une exécution en temps $O(n^2)$
- Si on exécute plusieurs fois l'algorithme, le temps de calcul moyen sera $O(n^2)$

Les algorithmes de type *Monte Carlo*

Monte Carlo : « toujours rapide, souvent correct »

Algorithme probabiliste dont

- la complexité ne dépend pas des choix aléatoires
- le résultat dépend des choix aléatoires

Analyse de l'algorithme

- Complexité : standard
- Correction : *probabilité de succès* (l'algorithme renvoie un résultat correct)

Améliorer la probabilité de succès

Si la probabilité de succès est p et qu'on répète N fois l'algorithme

- probabilité qu'aucune répétition ne soit correcte est $(1-p)^N \leq e^{-pN}$

Liens entre les différents types d'algorithmes

Numérique $\subset$ Monte Carlo

- On veut estimer une quantité α à $\pm 1\%$
- Algorithme correct si l'estimation renvoyée $\in [0.99 \cdot \alpha, 1.01 \cdot \alpha]$

Définition alternative de Las Vegas

- Temps fixé (non aléatoire)
- Résultat correct avec proba. p, ou ÉCHEC avec proba. $1 - p$ *pas de résultat faux*

Comparaisons Las Vegas – Monte Carlo

- Équivalence « Las Vegas $\iff$ Las Vegas alternatif »
  - ($\Leftarrow$) répétition tant que ÉCHEC $\to$ espérance de $1/p$ répétitions
  - ($\Rightarrow$) renvoyer ÉCHEC si #opérations $> k \cdot \mathbb{E}[t(n)] \to$ proba. d'ÉCHEC $\leq 1/k$ (Markov)
- Las Vegas (alternatif) $\Rightarrow$ Monte Carlo
  - remplacer ÉCHEC par une valeur arbitraire
- Monte Carlo + vérification des solutions $\Rightarrow$ Las Vegas

Bilan sur les algorithmes probabilistes

Pourquoi des algorithmes probabilistes ?

- Algorithmes souvent simples et efficaces
- Parfois : meilleure complexité que les algorithmes déterministes
- *Et pourquoi pas ?* en pratique, ils fonctionnent très bien !

Analyse des algorithmes probabilistes

- Modélisation probabiliste, avec variable aléatoire
- *Las Vegas* : étude de l'espérance de la complexité
- *Monte Carlo* : étude de la probabilité de succès

Et ensuite ?

- *Atlantic City* : « souvent correct, souvent rapide » ($\Rightarrow$ Monte Carlo)
- Algorithmes quantiques : généralisation des algorithmes probabilistes

Algorithmes probabilistes parfois difficiles à analyser… mais indispensables !

Ce que ne sont pas les algorithmes probabilistes

Algorithmes probabilistes et heuristiques

- Heuristique : méthode non prouvée, qui fonctionne souvent en pratique
  - souvent : basée sur des choix aléatoires
  - parfois : fonctionne sur *presque toutes* les entrées
- Probabiliste : méthode prouvée **quelque soit l'entrée**

Analyse en moyenne et analyse probabiliste

- Analyse en moyenne :
  - comportement d'algorithmes sur des *entrées aléatoires*
  - espérance de la complexité sur une entrée aléatoire
  - question subtile : quelle distribution sur les entrées ?
- Analyse probabiliste : espérance de la complexité **quelque soit l'entrée**
- Liens : certains algorithmes probabilistes reviennent à *rendre aléatoire* l'entrée

Table des matières

1. Qu'est-ce qu'un algorithme probabiliste ?

2. Exemple 1 : coupe minimale

3. Exemple 2 : tri rapide

Coupe minimale dans un graphe

Définition

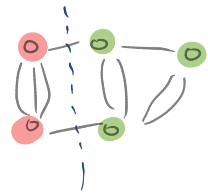
- Coupe dans un graphe $G = (S, A)$: partition de $S = S_1 \sqcup S_2$, $S_1, S_2 \neq \phi$
- Taille d'une coupe $S_1 \sqcup S_2$: nombre d'arêtes entre S_1 et S_2

$$= \#\{\{u_1, u_2\} \in A : u_1 \in S_1, u_2 \in S_2\}$$

Problème de la coupe minimale

Entrée : Graphe $G = (S, A)$

Sortie : Coupe $S = S_1 \sqcup S_2$ de taille minimale



Généralisation nécessaire : multigraphes

- Un *multigraphe* est un graphe qui autorise plusieurs arêtes entre 2 sommets
- Coupe et problème de la coupe minimale définis de manière équivalente

Algorithme probabiliste pour la coupe minimale

CoupeMin(G):

1. Tant que G possède au moins 3 sommets :
2. Choisir une arête de G, aléatoirement
3. *Contracter* l'arête G
4. Renvoyer la coupe définie par les deux sommets restants

Complexité de CoupeMin

Fait admis

Si un multigraphe G à n sommets est représenté par listes d'adjacence, la *contraction* d'une arête peut s'effectuer en temps $O(n)$

Théorème

L'algorithme CoupeMin renvoie une coupe de G en temps $O(n^2)$

Preuve

- A chaque étape, on effectue une contraction → le nb de sommets diminue de 1.
- Au total, on effectue $n-2$ contractions

↳ $\Theta(n^2)$

Correction de CoupeMin

Pourquoi cet algorithme fonctionnerait-il ?

Lemme de correction

L'algorithme CoupeMin appliqué à un multigraphe à n sommets renvoie une coupe *minimale* avec probabilité $\geq \frac{2}{n(n-1)}$

Remarque

- Cette probabilité est très petite !
- Exemple pour $n = 100$, $\frac{2}{n(n-1)} \simeq 0,02\%$…

Preuve du lemme de correction

- On fixe une coupe minimale C^*. CoupeFTin renvoie C^* $\Leftrightarrow$ aucune arête de C^* n'est contractée

- Quand il reste k sommets, la proba. de choisir une arête de C^* est $\dfrac{\#\text{arêtes dans } C^*}{\#\text{arêtes dans } G} \leq 2/k$.

  Soit $m = \#$arêtes dans C^*. Alors G possède $\geq km/2$ arêtes car chaque sommet a $\geq m$ voisins. (sinon on aurait une coupe de taille $< m$).

- Proba de renvoyer $C^* \geq \prod_{k=3}^{n} (1 - 2/k) = \prod_{k=3}^{n} \frac{k-2}{k} = \frac{(n-2)(n-3)\cdots 1}{n(n-1)\cdots 3} = \frac{2}{n(n-1)}$

$\Rightarrow$ Probabilité de renvoyer une coupe minimale $\geq$ proba. de renvoyer $C^* \geq \frac{2}{n(n-1)}$

Répétitions de CoupeMin

Probabilité de *succès* très faible $\rightarrow$ on répète l'algorithme pour améliorer cette probabilité

Lemme de répétition

Si on répète N fois CoupeMin et qu'on garde la plus petite coupe renvoyée, cette coupe est minimale avec probabilité $\geq 1 - e^{-2N/n(n-1)}$

Remarques

- Si on répète $N = 2n^2$ fois l'algorithme, on obtient
  - une complexité $O(n^4)$
  - une coupe minimale avec probabilité $\geq 1 - e^{-4} \simeq 98\%$

Preuve du lemme de répétition

- Proba de ne pas avoir une coupe minimale $\leq 1 - \frac{2}{n(n-1)}$

$\Rightarrow$ Proba de n'avoir aucune coupe minimale en N répétitions $\leq \left(1 - \frac{2}{n(n-1)}\right)^N$

Or $1 + x \leq e^x$ donc la proba est $\leq \left(e^{-2/n(n-1)}\right)^N = e^{-2N/n(n-1)}$

Bilan sur CoupeMin

Complexité

- Un appel à CoupeMin coûte toujours $O(n^2)$
- On a besoin de $O(n^2)$ *répétitions* $\rightarrow O(n^4)$

Correction

- Un appel à CoupeMin renvoie une coupe minimale avec proba. $\geq \frac{2}{n(n-1)}$
- N appels à CoupeMin renvoient une coupe minimale avec proba. $\geq 1 - e^{-2N/n(n-1)}$
- cn^2 appels à CoupeMin renvoient une coupe minimale avec proba. $\geq 1 - e^{-2c}$

CoupeMin répété est toujours en temps $O(n^4)$, et correct avec très bonne probabilité

- Type d'algorithme probabiliste : Monte Carlo

Table des matières

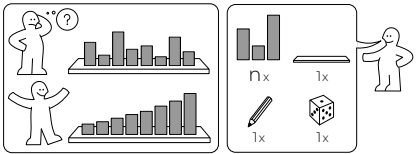
1. Qu'est-ce qu'un algorithme probabiliste ?

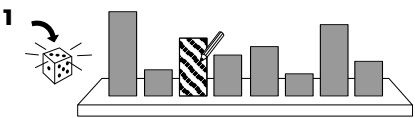
2. Exemple 1 : coupe minimale

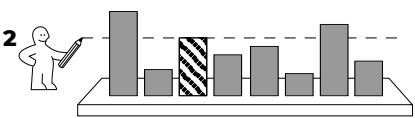
3. Exemple 2 : tri rapide

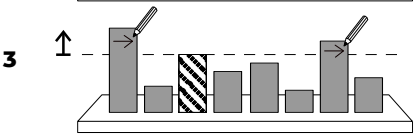
KVICK SÖRT

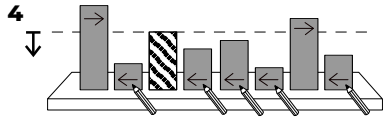
idea-instructions.com/quick-sort/
v1.2, CC by-nc-sa 4.0

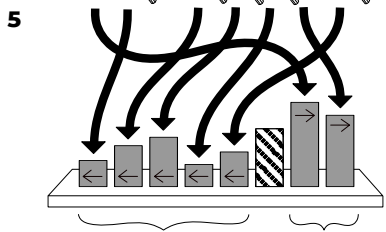


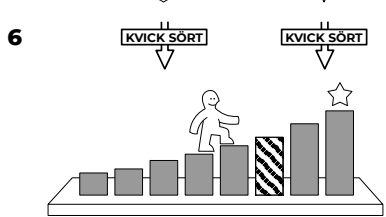












Le tri rapide

TriRapide(T) :

1. Si taille(T) $= 1$: renvoyer T
2. $p \leftarrow T_{[j]}$ avec j choisi aléatoirement entre 0 et $n-1$ *(pivot)*
3. T_{INF}, $T_{\text{SUP}} \leftarrow$ tableaux vides
4. Pour $i = 0$ à $n-1$, $i \neq j$:
5. Si $T_{[i]} \leq p$: ajouter $T_{[i]}$ à T_{INF}
6. Sinon : ajouter $T_{[i]}$ à T_{SUP}
7. $T_{\text{INF}} \leftarrow$ TriRapide(T_{INF})
8. $T_{\text{SUP}} \leftarrow$ TriRapide(T_{SUP})
9. Renvoyer la concaténation T_{INF}, p et T_{SUP}

Implantation

- Pas de tableaux annexes T_{INF} et T_{SUP}
- *Partition* en T_{INF} et T_{SUP} en place dans T en échangeant des éléments

cf. TP Programmation

Première analyse du tri rapide

Lemme (correction)

TriRapide(T) renvoie T trié

- Algo correct si $\#T = 1$
- Si $\#T > 1$, $\#T_{inf}, \#T_{sup} < \#T$ $\Rightarrow$ par hyp. T_{inf} et T_{sup} sont correctement triés. Donc l'algo renvoie T trié.

Lemme (complexité)

~~Dans le pire des cas~~, le nombre de comparaisons effectuées est $O(n^2)$

Si les choix proba. sont très malchanceux,

Par exemple, si $\#T_{inf} = \#T - 1$ à chaque étape, alors le nb C_n de comparaisons vérifie $C_n = n - 1 + C_{n-1}$ $\rightsquigarrow C_n = \Theta(n^2)$

Espérance du nombre de comparaisons

Notations

- $T^{(i)}$: $i^{\text{ème}}$ plus petit élément de T
- $X_{ij} = \begin{cases} 1 & \text{si } T^{(i)} \text{ est comparé à } T^{(j)} \text{ au cours de l'algorithme} \\ 0 & \text{sinon} \end{cases}$
- X : nombre total de comparaisons $\rightarrow X = \sum_{i<j} X_{ij}$

Lemme

Pour $1 \leq i < j \leq n$, $\mathbb{E}[X_{ij}] = \Pr[X_{ij} = 1] = 2/(j - i + 1)$

Théorème

L'espérance du nombre de comparaisons effetuées par TriRapide est $O(n \log n)$

$$\mathbb{E}[X] = \sum_{i<j} \mathbb{E}[X_{ij}] = \sum_{i<j} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k} \leq \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$$

Or $\sum_{k=1}^{n} \frac{1}{k} \simeq \ln(n)$ donc $\mathbb{E}[X] = \Theta(n \log n)$

Preuve du lemme

au tableau pendant le TD

Bilan sur le TriRapide

Propriétés du TriRapide

- L'algorithme est toujours correct
- L'espérance de sa complexité est $O(n \log n)$
- Type d'algorithme probabiliste : Las Vegas

Comportement pratique

- Le TriRapide est efficace en pratique, s'il est implanté *en place*
- Tri *stable*