Notes de cours

24 Décembre 2023

Table des matières

1. Notions mathématiques importantes

1.1. Distribution de probabilités discrètes

La distribution de probabilité discrète la plus élémentaire est sans doute celle de **Bernoulli**, que l'on peut interprété comme un lancer de pièce. Elle est définie comme,

$$P[X = 1] = p$$

et

$$P[X = 0] = 1 - p$$

pour un paramètre $p \in [0, 1]$ La variance d'une telle variable aléatoire X est $p(1 - p)$ et son espérance p.

Définition 1.1.1: La distribution binomiale est définie comme,

$$P[X = j] = \binom{n}{j} p^j (1 - p)^{n-j}$$

2. Algorithmes de Monte-Carlo

Définition 2.1: Un algorithme de Monte-Carlo est un algorithme randomisé dont le **temps d'exécution** est **déterministe** mais le résultat peut être incorrect avec une certaine probabilité p (généralement petite)

→ Pas de surprise sur la durée du calcul
→ Peut-être faux mais c'est très rare
→ Nous donne des algos rapides avec une faible probabilité d'échec

2.1. Biais

Un algorithme randomisé peut être érroné - il répond OUI alors que la réponse aurait dut être NON - ce qui n'est pas le cas des algorithmes déterministes. Un algorithme de Monte-Carlo peut toujours renvoyer une réponse exacte si il a un **biais**.

Un algorithme biaisé vers le faux est toujours correct lorqu'il retourne FAUX.
Un algorithme biaisé vers le vrai est toujours correct lorqu'il retourne VRAI.

Définition 2.1.1: Un algorithme est *one-side error* lorsqu'il est soit biaisé vers le faux, soit biaisé vers le vrai

Définition 2.1.2: Un algorithme est *two-side error* lorsqu'il est n'est pas biaisé, et donc qu'il répond VRAI ou FAUX selon une probabilité p.

2.2. Algorithme de Las-Vegas

Définition 2.2.1: Un algorithme de Las-Vegas est un algorithme randomisé dont l'exécution est **déterministe** mais le temps d'exécution est non-prévisible. Il s'arrête lorsqu'il trouve une réponse exacte

3. Classes de Complexités

À l'instar des algorithmes déterministes et des classes de complexités P et NP, les algortihmes randomisés ont égalements leurs classes de complexités.

3.1. RP et coRP

Randomized Polynomial-time $\rightarrow$ résolvable en $\text{TIME}(P)$ avec une probabilité bornée par un algorithme de Monte-Carlo biaisé.
Formellement, la classe RP contient les langages L tq il existe un algorithme randomisé A s'exécutant en $\text{TIME}(P)$ au pire cas de sorte que pour n'importe quelle entrée $x \in \sum^*$,

- $x \in L \Rightarrow A(x)$ accepte avec une probabilité $\geq \frac{1}{2}$
- $x \notin L \Rightarrow A(x)$ rejette

$\text{P} \subseteq \text{RP} \subseteq \text{NP}$
Suivant la même logique, la classe coRP consiste aux langages L:

- $x \in L \Rightarrow A(x)$ accepte
- $x \notin L \Rightarrow A(x)$ rejete avec une probabilité $\geq \frac{1}{2}$

3.2. ZPP

Zero-Error Probabilistic Polynomial Time $\rightarrow$ résolvable en $\text{TIME}(P)$ avec un algorithme de Las-Vegas.

Théorème 3.2.1: $\text{ZPP} = \text{RP} \cap \text{coRP}$

Question de la liste des questions d'examens
Preuve:

1. Preuve que $\text{RP} \cap \text{coRP} \subseteq \text{ZPP}$:

   Supposons qu'un langage L soit dans RP et dans coRP. Alors il existe deux algorithme A et B tq:
   - si $x \in L$ alors $A(x)$ accepte avec une probabilité $\geq \frac{1}{2}$, et $B(x)$ accepte toujours
   - si $x \notin L$ alors $A(x)$ rejette toujours, et $B(x)$ rejette avec une probabilité $\geq \frac{1}{2}$

   On peut alors exécuter les deux algorithmes sur une entrée x. Si $B(x)$ accepte, on est certain que $x \in L$ et de même, si $A(x)$ rejette alors on est sûr que $x \notin L$. Dans le cas où $A(x)$ accepterait et $B(x)$ rejeterais, on itère. Il n'est pas possible que $A(x)$ accepte toujours et $B(x)$ rejette toujours. On sera donc certains, après un nombre constant attendu d'itération, d'obtenir la bonne réponse. □
2. Preuve que $\text{ZPP} \subseteq \text{RP} \cap \text{coRP}$: Supposons qu'un langage L appartienne à ZPP. Nous pouvons créer deux algorithmes, un montrant que le langage L se trouve dans RP, et un autre montrant qu'il appartienne à coRP.

- Pour le premier algorithme, supposons que l'algorithme ZP s'exécute en temps polynomial $p(n)$ où $n = |x|$. Alors on peut faire ceci:
  1. Exécuter l'algorithme A pour $2p(n)$ étapes,
  2. Si A s'est arrêté après $2p(n)$ étapes, alors la sortie est la réponse, sinon il rejete.

  Quelle est la probabilité que A s'arrête après $2p(n)$ en sachant que, selon son espérance, il s'arrête après $p(n)$ étapes. Selon l'inégalité de Markov (voir Section 6.1), soit X le temps d'exécution de A, on a $P[X > 2p(n)] \leq \frac{E[X]}{2p(n)} = \frac{p(n)}{2p(n)} = \frac{1}{2}$. Dopc, si $x \in L$, on a une probabilité au moins $\frac{1}{2}$ que ça soit correct. Si $x \notin L$, alors il rejete toujours. Cela prouve que $L \in$ RP.
- Pour le second algorithme ($L \in$ coRP), on fait l'inverse; on accepte l'entrée lorsque A ne s'arrête pas après $2p(n)$ étapes.

3.3. PP

Polynomial Probabilistic → décidable en temps polynomial avec une probabilité d'erreur inférieure à 1/2.
Formellement, la classe PP consiste aux langages L dont il existe un algorithme randomisé A s'exécutant en temps polynomial au pire cas tel que pour n'importe quelle entrée $x \in \sum^*$,
- $x \in L \Rightarrow A(x)$ accepte avec une probabilité $> \frac{1}{2}$
- $x \notin L \Rightarrow A(x)$ accepte avec une probabilité $< \frac{1}{2}$

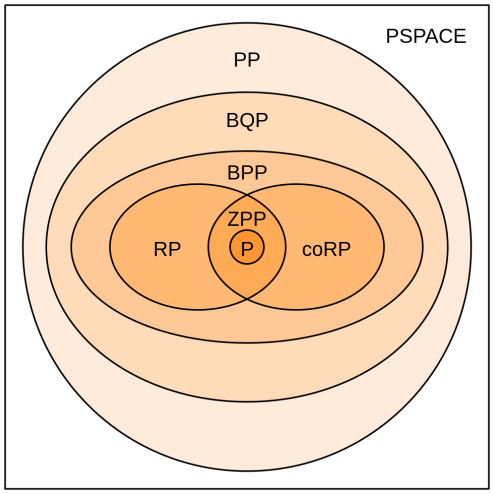

Figure 1: Relations entre les classes de complexités probabilistes

3.4. BPP

Bounded-Error Probabilistic Polynomial-time → Le problème est résolvable en TIME(P) avec un algorithme de Monte-Carlo non biaisé.
Formellement, la classe BPP consiste aux langages L tel qu'il existe un algorithme randomisé A s'exécutant en temps polynomial au pire cas tel que pour n'importe quelle entrée $x \in \sum^*$,
- $x \in L \Rightarrow A(x)$ accepte avec une probabilité $\geq \frac{3}{4}$
- $x \notin L \Rightarrow A(x)$ accepte avec une probabilité $\leq \frac{1}{4}$

BPP et PP définissent des algorithmes de Monte-Carlo non-biaisé (*two-sided error*).

Il est possible de prouver que la définition de BPP ne change pas si l'on prend une probabilité d'erreur $p < \frac{1}{2}$ à la place de $\frac{1}{4}$.

=> J'ai beau la faire, je ne comprends pas la preuve.

4. Non-uniformité

Retirer l'aléatoire dans un algorithme randomisé n'est pas toujours possible. Cela dépend de plusieurs critères et en particulier sur l'**uniformité**[i] des algorithmes. En effet, pour rendre déterministe un algorithme randomisé, il est nécessaire d'introduire une nouvelle fonctionnalité connue sous le nom de **non-uniformité**[ii], il s'agit d'un *conseil* en taille polynomiale et ne dépendant que de la taille de l'entrée n.

4.1. P/poly

→ P/poly n'est pas forcement appliquable en pratique, en effet, la classe contient des problèmes indécidables.
→ P/poly est déterministe ! On a retiré l'aléatoire.
→ On peut la définir de deux façons différents: une avec des conseils et l'autre avec des circuits booléens.

4.1.1. Conseils

La classe P/poly est la classe de complexité contenant les langages qui peuvent être décidés en temps polynomial par une machine de Turing déterministe avec une fonction de conseil a tel que $a(n)$ est bornée polynialement en n.
$a(n) : \mathbb{N} \to \sum^*$ et $|a(n)| < O(n^c)$
Il s'agit d'un mot de taille dépendant de la taille de l'entrée (un algorithme déterministe A décide un langage L avec un conseil a sur une entrée $n \in \sum^*$ tel que $a(|n|)$). Ce conseil aide énormément l'algorithme à résoudre le problème.

4.1.2. Circuits booléens

→ On peut représenter un conseil par un circuit.
→ Un circuit booléen est une fonction booléenne avec n entrées.
→ Une famille de circuits $C_1, C_2, ...$ est dite une famille de circuits de taille polynomiale pour un langage L si $\forall n$ le circuit C_n calcule $x \in L \forall x \in \sum^n$.
→ L'ensemble des langages ayant une famille de circuits de taille $O(n^c)$ est dite $\text{SIZE}(n^c)$ et l'ensemble des langages ayant un circuit en taille polynomiale est donc l'union de ces tailles $\bigcup_{c>0} \text{SIZE}(n^c)$

P/poly = $\bigcup_{c>0} \text{SIZE}(n^c)$
Preuve:

- La première direction est simple, le circuit C_n peut-être représenté par une description de conseil de $a(n)$ qui est ensuite exécuté par un algorithme pour simuler le circuit booléen.
- La seconde direction: On observe d'abord qu'il existe un algorithme déterministe décidant L en un temps polynomial (si $L \in$ P) et qu'alors, il existe une famille de circuits de taille polynomiale pour L. Un nombre polynomial d'étapes d'exécutions d'une Machine de Turing peuvent être modélisées par un circuit booléen.
- Donc, si il existe un circuit additionnant le conseil $a(n)$ - une chaîne de caractères polynomiale - on peut construire une famille de taille polynomiale de circuits om le conseil $a(n)$ est branché directement au circuit. Et étant donné que $a(n)$ a une taille polynomiale en n, la taille de ces circuits est aussi polynomiale en n.

5. Arbres de jeux et principe de Yao

Vision "théorie des jeux" des algorithmes randomisés. Cela permet de visualiser des algorithmes probabilistes comme une distribution de probabilités sur des algorithmes déterministes.
Le **principe de Yao** permet d'établir une borne inférieure sur les performances d'un algorithme randomisé.

5.1. Arbre min-max

Forme la plus basique des algorithmes d'intelligence artificielle permettant de représenter un jeu entre deux joueurs[iii]. Arbre où chaque valeur est binaire (l'input étant stocké dans les feuilles). Chaque noeud de l'arbre est soit un *min* ($\wedge$ booléen) soit un *max* ($\vee$ booléen).

---

[i] Algorithmes s'exécutant de la même façon pour des entrées de tailles différentes

[ii] Similairement à la définition d'uniformité. Un Algorithme non-uniforme peut largement varié selon la taille de l'entrée

[iii] On associe à chaque noeud un ensemble d'enfants représentant les options possible pour un des deux joueur

Si le jeu n'était pas binaire, le concept serait le même, chaque feuilles contient une valeur numéraire. Les noeuds à distance paires du sommet serait nommées MAX et les noeuds à une distance impaires MIN. L'évaluation de l'arbre suivrait dont le processus suivant: chaque noeud MAX retourne la valeur maximum parmis ses enfants et chaque noeud MIN la valeur minimum. Le but final étant - pour une certaine entrée - de donner la valeur de la racine.
On note T_k l'arbre de jeu de hauteur $2k$. La racine est un noeud $\wedge$, il y a donc exactement k noeuds $\wedge$ et k noeuds $\vee$. L'arbre T_k possède dont $2^2k = 4^k$ feuilles.
Évaluer un tel arbre nécessiterait donc, pour un algorithme déterministe, d'évaluer les 4^k noeuds. Cela dit, avec un algorithme probabiliste on peut augmenter la borne inférieure. Un exemple de tel algorithme est le suivant:
On commence par la racine et on évalue récursivement un des deux enfants du noeud courant. Si le noeud courant est un $\wedge$ et que le résultat de l'analyse d'un de ces enfants est 0, il n'est pas nécessaire d'évaluer l'autre enfant. Selon le même principe, si le noeud courant est un $\vee$ et qu'un des appel récursif retourne un 1 il n'est pas nécessaire d'évaluer l'autre enfant. Dans tous les autres cas, il est nécessaire de vérifier les deux enfants pour s'assurer que tout boume.

Théorème 5.1.1: Soit n'importe quelle instance d'un arbre de jeu binaire $T_{2,k}$, le nombre attendu d'étapes pour la stratégie d'évaluation randomisée est au plus 3^k

Preuve: On a 4 cas différents:
- Le noeud courant est un $\vee$:
  - Retourne 0: On a donc besoin de 2 appels récursifs, ce qui coûte
    $$2 \times 3^{k-1}$$
    lectures attendus.
  - Retourne 1: On a, avec une probabilité d'$\frac{1}{2}$, seulement un enfant à évaluer.
    $$\underbrace{\frac{1}{2} \times 2 \times 3^{k-1}}_{\text{Retourne 0}} + \underbrace{\frac{1}{2} \times 3^{k-1}}_{\text{Retourne 1}} = \frac{3}{2} \times 3^{k-1}$$
- Le noeud courant est un $\wedge$:
  - Retourne 0: Alors ses deux enfants retourne soit 0, avec une probabilité
    $$2 \times 3^{k-1}$$
    ; soit qu'un seul des deux retourne 0, avec une probabilité
    $$\frac{1}{2} \times 2 \times 3^{k-1} + \frac{1}{2} \times \left(\frac{3}{2} \times 3^{k-1} + 2 \times 3^{k-1} \right) = \frac{11}{4} \times 3^{k-1}$$
  - Retourne 1: Alors ses deux enfants doivent retourner 1, il est nécessaire de regarder les deux enfants.
    $$2 \times \underbrace{\frac{3}{2} \times 3^{k-1}}_{\text{Coût de vérification d'un noeud } \vee} = 3^k$$

Dans tous les cas, on remarque que le nombre attendu d'opération pour évaluer le $\wedge$ racine ne dépasse pas 3^k. □

5.2. Théorie des jeux

Un jeu à somme nulle pour deux joueurs est un jeu pouvant être représenté par M, une matrice de gain $n \times m$ dont les entrées sont encodées par paires de stratégies, une pour chaque joueur. M_{ij} est la quantité payée par le joueur C (colone) au joueur L (ligne) lorsque L choisi la stratégie i et C la stratégie j.

5.3. Stratégies mixtes

Dans le cas des stratégies mixtes, la matrice M ne contient plus les gains mais une distribution de probabilité sur l'ensemble des stratégies. Le gain attendu pour un paire de stratégie p, q est :

$$E[\text{payoff}] = p^T M q = \sum_{i=1}^{n} \sum_{j=1}^{m} p_i M_{ij} q_j$$

5.4. Théorème de Minmax

Théorème 5.4.1:

$$\max_p \min_q p^T M q = \min_q \max_p p^T M q$$

Il s'agit d'un principe fondamental de théorie des jeux.
Si l'on dénote V_L la meilleure borne inférieure sur le gain attendu par le joueur ligne en choisissant une stratégie mixte p, et par V_C la meilleure borne supérieure sur le gain payé par le joueur colonne pour une stratégie mixte q.

$$V_R = \max_p \min_q p^T M q$$

$$V_C = \min_q \max_p p^T M q$$

Par le Théorème 5.4.1, nous savons que $V_R = V_C$.

5.5. Principe de Yao

Soit un problème computationnel $\sqcap$, on considère une collection finie $\mathcal{A}$ d'algorithmes et un ensemble $\mathcal{I}$ d'entrées, tel que tous les algos dans $\mathcal{A}$ exécutent correctement le probleme $\sqcap$ sur toutes les entrées dans $\mathcal{I}$. On écrit $C(I, A)$ le temps d'exécution d'un algorithme $A \in \mathcal{A}$ sur l'entrée $I \in \mathcal{I}$.

Cela défini un jeu à somme nulle entre deux joueur: un designer d'algorithme et une entrée adverse[iv]. L'ensemble $\mathcal{A}$ peut-être vu comme un ensemble de stratégies pour le designer d'algorithme, une stratégie mixte peut alors être vue comme un algorithme randomisé. Dans le cas de l'ensemble $\mathcal{I}$, une stratégie mixte peut être vue comme une distribution sur les entrées possibles.

En appliquant le Théorème 5.4.1 on remarque que le temps d'exécution attendu pour un algorithme déterministe optimal sur une entrée choisie arbitrairement sur une distribution p est une borne inférieure sur le temps d'exécution de n'importe quel algorithme randomisé exact (Las-Vegas) pour $\sqcap$.

Théorème 5.5.1: Pour toutes les distributions p sur $\mathcal{I}$ et q sur $\mathcal{A}$, nous avons

$$\min_{A \in \mathcal{A}} E\big[C(I_p, A)\big] \leq \max_{I \in \mathcal{I}} E\big[C(I, A_q)\big]$$

Donc d'une part, nous avons $C(I_p, A)$ le temps d'exécution attendu pour le meilleur algorithme déterministe[v] sur une entrée choisie aléatoirement. De l'autre côté, nous avons $C(I, A_q)$ le temps d'exécution d'un

[iv] Voir ça comme de vrais adversaires, non pas qu'ils soient méchants entre elleux, mais plus qu'iels veulent remporter ! Il n'est pas question de laisser à l'autre adversaire remporter la partie.
La théorie des jeux, c'est comme un Monopoly avec la famille, si il faut que ça se finisse dans les larmes et le sang ça se fera.

[v] Étant donné que l'algorithme est choisi par l'adversaire, il va clairement pas nous faire de cadeau en nous en donnant un pas top. Non, il prend le meilleur.

algorithme randomisé sur la pire entrée possible[vi].
Donc, pour prouver une borne inférieure sur la complexité d'un algorithme randomisé, il suffit de choisir n'importe quelle distribution d'entrée p et de prouver une borne inférieure sur le temps d'exécution attendu des algorithmes déterministes pour cette distribution p.
La force de cette technique est dans la flexibilité du choix de p et dans la réduction à une borne inférieure des algorithmes déterministes.
Attention, cela donne une borne inférieure sur les performances des algorithmes de Las-Vegas uniquement ! (C'est-à-dire ceux retournant une réponse exacte à tous les coups)

5.6. Principe de Yao pour les évaluation d'arbre de jeux

Si l'on reprend notre arbre de jeu AND-OR et son problème d'évaluation. Un algorithme randomisé peut-être vu comme une distribution de probabilité sur des algorithmes déterministes (car la longueur des calculs aussi bien que de nombre de choix à chaque étape sont toutes deux finies).
Imaginons un arbre de jeu T_k dont les feuilles se trouvent à une distance $2k$ de la racine, et dont tous les noeuds internes sont des NOR: Un noeud retourne un 1 uniquement si les deux entrées (enfants) sont à 0, sinon il retourne 0. L'analyse de cet arbre de NORS de hauteur $2k$ est la suivante:
Notons $p = \frac{3-\sqrt{5}}{2}$, chaque feuille de l'arbre contient 1 avec un probabilité p. Un noeud NOR retourne donc 1 avec une probabilité :

$$\left(\frac{\sqrt{5}-1}{2}\right)^2 = \frac{5+1-2\sqrt{5}}{4} = \frac{6+2\sqrt{5}}{4} = \frac{3-\sqrt{5}}{2} = p$$

La valeur de chaque noeud NOR est donc 1 avec une probabilité p et est indépendante des valeurs de tous les autres noeuds sur le même niveau.
Considérons maintenant un algorithme déterministe pour évaluer un tel arbre de jeu. Pour commencer, notons v un noeud de l'arbre dont l'algorithme cherche à obtenir la valeur. Il est simple de voir que l'on va d'abord évaluer un premier enfant de v avant d'aller voir l'autre enfant. On peut donc voir cela comme une *depth-first search* (recherche en profondeur) qui s'arrêterait d'évaluer les sous-arbre de v lorsque v sera évalué. Nous appelons un tel algorithme **depth-first pruning** (élagage en profondeur), en effet, les sous-arbres ne fournissant pas plus d'informations sur l'évaluation (par exemple, si le premier enfant retourne un 1 on sait qu'il est inutile d'aller évaluer l'autre enfant), on élague donc ce sous-arbre de l'évaluation.

Proposition 5.6.1: Si l'on prend maintenant un arbre NOR T dont les feuilles sont indépendamment définies à 1 avec une probabilité q pour une valeur fixée $q \in [0,1]$. Notons $W(T)$ le nombre d'étapes minimum (parmis tous les algorithmes déterministes) nécessaires pour évaluer T. Il y a donc un algorithme de *depth-first pruning* dont le nombre d'étapes attendues pour évaluer T est $W(T)$

Cette Proposition 5.6.1 nous dis que, pour les besion de notre borne inférieure, nous pouvons réduire notre attention aux algorithmes *depth-first pruning*. Nous définisons alors $W(h)$ comme étant le nombre attendu de feuilles inspectées par un tel algorihtme d'élagage lorsqu'il évalue un noeud se trouvant à une distance h des feuilles, quand les feuilles sont définies indépendament avec une probabilité $\frac{3-\sqrt{5}}{2}$. Alors le nombre de feuilles lues lors de l'exécution d'un tel algorithme est :

$$W(h) = \underbrace{W(h-1)}_{\substack{\text{Nombre d'étapes nécessaires} \\ \text{pour évaluer le premier enfant}}} + \underbrace{(1-p) \times W(h-1)}_{\substack{\text{Nombre d'étapes nécessaires} \\ \text{pour évaluer le second enfant} \\ \text{Cela n'a lieu que si le premier enfant vaut 0} \\ \text{ce qui arrive avec une probabilité } 1-p}}$$

[vi]Car choisie par l'adversaire, on oublie pas qu'il est pas là pour nous aider.

étant donné que l'on doit effectuer un deuxième appel récursif uniquement si l'évaluation du premier enfant retourne 0, ce qui arrive avec une probabilité $(1-p)$. On peut alors résoudre la récurrence:

$$\begin{aligned} W(h) &= W(h-1) + (1-p) \times W(h-1) \\ &= (2-p) \times W(h-1) \\ &= (2-p)^h \\ &= (2-p)^{\log_2 n} \\ &= n^{\log_2(2-p)} \\ &\simeq n^{0.694} \end{aligned}$$

Sur base du principe de Yao, nous pouvons donc déduire le théorème suivant:

Théorème 5.6.1: Le temps d'exécution attendu our n'importe quel algorithme randomisé évaluant toujours une instance de T_k correctement est au moins $n^{0.694}$, où $n = 2^2 k$ est le nombre de feuilles de T_k

5.7. Exercices d'examens

5.7.1. Exercice 2.2

Remarque préliminaire: Il s'agit d'une tentative de réponse il se peut qu'elle soit érronée !
$\rightarrow$ On a un arbre d'hauteur h
$\rightarrow$ Chaque noeud à 3 enfants
$\rightarrow$ Il y a $n = 3^h$ feuilles contenant des valeurs booléennes
$\rightarrow$ Chaque noeud retourne la valeur majoritaire parmis ses enfants (valeur booléenne donc)

1. Étant donné que l'on a 3 enfants et que les valeurs sont booléennes (0 ou 1), on sait qu'il ne peut y avoir d'égalité parmis la majorité (car nombre impair d'enfants). L'algorithme d'évaluation peut donc être défini comme suit: Pour évaluer un noeud v, on prend 2 de ses enfants aléatoirement et on les compare, si ils sont identiques c'est super, on a trouvé la majorité, v est donc évalué à la valeur des deux enfants sélectionnés, il n'est plus nécessaire d'évaluer le troisième étant donné qu'il n'impactera pas la majorité (qu'il soit identique ou différent). Si la valeur des deux enfants est différente, on va alors évaluer le troisième pour trancher sur la majorité (et les trois enfants sont donc évalués).
   Il est facile de remarquer qu'il n'y a que deux cas possibles:
   - Les enfants sont 3 b[vii]: la majorité est obtenue lorsqu'on évalue 2 enfants. Le nombre d'évaluation est donc de 2.
   - Les enfants sont 2 b et 1 $\neg b$: Soit on sélectionne les deux b du premier coup et c'est niquel, soit on sélectionne un b et le $\neg b$ et dans ce cas là il faut évaluer le dernier enfant pour savoir quelle est la valeur majoritaire. Le nombre d'évaluation est donc de $\underbrace{\frac{1}{3} \times 2}_{2b \text{ du premier coup}} + \underbrace{\frac{1}{3} \times (2+1)}_{1b \text{ et } 1 \neg b \text{ puis un } b} = \frac{5}{3}$

   Le pire cas est donc le premier cas $= 2$ donc $T(h) = 2T(h-1) = 2^h$, $n = 3^h \Rightarrow h = \log_3 n$, $2^{\log_3 n} = n^{\log_3 2}$
2. On part du principe que chaque feuilles prend la valeur 1 avec une probabilité $p = \frac{1}{2}$. Notons $W(h)$ le nombre de feuilles lues lorqu'on analyse un noeud se trouvant à une distance h des feuilles. On sait qu'il est nécessaire d'évaluer à minima 2 enfants ($2W(h-1)$) et qu'il y a une probabilité $\frac{1}{2}$ que les deux valeurs soient différentes car:
   - Proba d'avoir 2 valeurs identiques $\circledast$: $\underbrace{\frac{1}{2} \times \frac{1}{2}}_{\text{On a deux 1}} + \underbrace{\frac{1}{2} \times \frac{1}{2}}_{\text{On a deux 0}} = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$
   - Proba d'avoir 2 valeurs différentes : $1 - \underbrace{\frac{1}{2}}_{\circledast} = \frac{1}{2}$

On sait alors qu'il est nécessaire d'effectuer $\frac{1}{2}W(h-1)$ évaluations.

$$\begin{aligned} W(h) &= 2W(h-1) + \frac{1}{2}W(h-1) \\ &= \frac{5}{2}W(h-1) \\ &= \left(\frac{5}{2}\right)^h = \left(\frac{5}{2}\right)^{\log_3 n} \\ &= n^{\log_3 5} \simeq n^{1.464} \end{aligned}$$

6. Inégalités de concentrations

Définition 6.1 : En probabilité, les **inégalités de concentration** fournissent des bornes sur la probabilité qu'une variable aléatoire dévie d'une certaine valeur (généralement l'espérance de cette variable aléatoire)

En d'autres termes, ces inégalités nous donnent des bornes sur les variations que peuvent prendre une variable aléatoire. Dans le cas des algorithmes randomisés celles-ci sont intéressantes pour prouver qu'un tel algorithme nous donne une réponse correcte avec une grande probabilité.

6.1. Inégalité de Markov

Il s'agit de l'inégalité la plus simple. On l'utilisera comme base pour d'autres inégalités. Elle utilise l'espérance de la variable aléatoire.

Définition 6.1.1 : Si X est une variable aléatoire et $f(x)$ une fonction réelle. Alors l'**espérance** de $f(x)$ est donnée par

$$E[f(X)] = \sum_{x \in X} f(x) P[X = x]$$

L'inégalité de Markov nous donne la borne la plus serrée possible quand nous savons que X est non-négatif et a une espérance donnée.

Théorème 6.1.1 : Si X est une variable aléatoire non-négative et $a > 0$ alors

$$P[X \geq a] \leq \frac{E[X]}{a}$$

Et de manière équivalente

$$P[X \geq kE[X]] \leq \frac{1}{k}$$

Preuve: Définissons une fonction

$$f(x) = \begin{cases} 1 \text{ si } x \geq a \\ 0 \text{ sinon} \end{cases}$$

Alors $P[X \geq a] = E[f(X)]$. Étant donné que $f(x) \leq \frac{x}{a}$ pour n'importe quel x, on a

[vii] b est une valeur booléenne, on se moque de laquelle.

$$E[f(X)] \leq E\left[\frac{X}{a}\right] = \frac{E[X]}{a}$$

□

Malheureusement, cette borne est trop faible pour apporter des résultats utiles. Cependant, elle peut être utilisée pour trouver de meilleures bornes sur la probabilité de la queue en utilisant plus d'informations sur la distribution de la variable aléatoire.

6.2. Inégalité de Chebyshev

Cette inégalité est basée sur la conaissance de la variance de la distribution.

Définition 6.2.1: Pour une variable aléatoire X avec une espérance μ_X, sa **variance** est définie comme

$$\sigma_X^2 = E\left[(X - \mu_X)^2\right]$$

Définition 6.2.2: L'**écart-type** d'une variable aléatoire X, notée σ_X est la racine carrée positive de la variance de X.

Théorème 6.2.1: Soit X une variable aléatoire ayant une espérance μ_X et un écart-type σ_X, pour n'importe quel $t \in \mathbb{R}^+$,

$$P[|X - \mu_X| \geq t\sigma_X] \leq \frac{1}{t^2}$$

Preuve: On observe que

$$P[|X - \mu_X| \geq t\sigma_X] = P\left[(X - \mu_X)^2 \geq t^2\sigma_x^2\right]$$

Définissons une variable aléatoire $Y = (X - \mu_X)^2$, son espérance vaut alors $E[Y] = \sigma_X^2$, si l'on applique l'inégalité de Markov à cette variable aléatoire avec $a = t^2$, clairement[viii]:

$$\begin{aligned}
P[Y \geq t^2] &\leq \frac{E[Y]}{t^2} \\
P\left[(X - \mu)^2 \geq t^2\right] &\leq \frac{\sigma^2}{t^2} \\
P\left[(X - \mu)^2 \geq \sigma^2 t^2\right] &\leq \frac{1}{t^2} \\
P[(X - \mu) \geq \sigma t] &\leq \frac{1}{t^2}
\end{aligned}$$

□

6.3. Borne de Chernoff

Borne sur la queue de la distribution bien plus précises que celles de Markov ou Chebyshev. Ces bornes sont utilisées dans le cas où les variables aléatoires ne peuvent pas être modélisées comme une somme de variables aléatoires indépendantes (par exemple des variables de Bernoulli[ix]).

[viii] Je dis clairement, mais j'avais tellement pas compris la preuve (qui pourtant est super simple) que j'ai passé 1h à la refaire. Oui parfois y'a des trucs débiles sur lesquels on passe une plombe, pour la peine je vais écrire les étapes.

[ix] Résultat d'un pile ou face

Définition 6.3.1: Soit $X_1, ..., X_n$ des **variables aléatoires indépendantes de Bernoulli** tel que $1 \leq i \leq n, P[X_i = 1] = p$ et $P[X_i = 0] = (1 - p)$. Si l'on défini $X = \sum_{i=1}^{n} X_i$, alors X suit une distribution binomiale.

Définition 6.3.2: Des essais de **Poisson** font références à des variables aléatoires indépendantes dont les probabilités sont propres aux variables. Il s'agit d'une généralisation de Bernoulli. Soit $X_1, ..., X_n$ des variables de Bernoulli tel que pour $1 \leq i \leq n, P[X_i = 1] = p_i$ et $P[X_i = 0] = (1 - p_i)$

Nous nous concentrons sur le cas général (Poisson) mais il fonctionne bien évidemment aussi dans le cas "Bernoulli" où toutes les variables partagent la même probabilité p. On peut se poser des questions sur la déviation de l'espérance μ de X, $\mu = \sum_i = 1^n p_i$ telles que "Quelle est la probabilité que X dépasse $(1 + \delta)\mu$?". Une réponse à cette question est utile pour analyser un algorithme randomisé, cela montre que les chance que l'algorithme échoue une certaine performances sont faibles. Un autre type de question que l'on peut se poser est "Quelle valeur maximale δ peut prendre de sorte que la probabilité de la queue soit plus petite qu'une valeur ε ?"

Théorème 6.3.1: Soit $X_1, ..., X_n$ une collection d'essais de Poissons indépendants[x] tq pour tout $1 \leq i \leq n, P[X_i = 1] = p_i$ où $0 < p_i < 1$. Alors, pour un $X = \sum_{i=1}^{n} X_i, \mu = E[X] = \sum_i^n p_i$ et un $\delta > 0$,

$$P[X > (1 + \delta)\mu] < \left[\frac{e^\delta}{(1 + \delta)^{1+\delta}}\right]^\mu$$

Remarque: le membre de droite de l'inégalité est une fonction ne dépendant que des paramètres δ et μ

L'idée derrière cette formule est d'appliquer les inégalités de Markov sur les *fonction génératrices des moments* des variables aléatoires X définies comme $M_{X(t)} = E[e^{tX}]$.
Preuve:

$$P[X > (1 + \delta)\mu] = P\left[e^{tX} > e^{t(1+\delta)\mu}\right]$$
$$\Rightarrow P\left[e^{tX} > e^{t(1+\delta)\mu}\right] < \frac{E[e^{tX}]}{e^{t(1+\delta)\mu}} \qquad \text{Inégalité de Markov}$$

Ici, une petite étape tranformation du numérateur dans le terme de droite s'impose

$$\begin{aligned} E[e^{tX}] &= E\left[e^{t\sum_i x_i}\right] \\ &= E\left[\prod_i e^{tx_i}\right] \qquad \text{Propriété des exposants} \\ &= \prod_i E[e^{tx_i}] \qquad \text{Linéarité de l'espérance} \end{aligned}$$

Ici aussi, une petite transformation est nécessaire pour nous simplifier la vie, elle est pas super triviale pour un étudiant en info qui comprend pas grand chose en proba. En gros, l'idée est quand même simple mais il faut l'avoir: l'espérance représentant la valeur que peut prendre une variable aléatoire et qu'on a des variables de Poissons (c'est à dire dont les valeurs x_i ont chacunes une probabilité p_i), on peut observer que e^{tX_i} prend la valeur e^t avec une proba p_i et 1 avec une proba $1 - p_i$, en effet:

[x] Il s'agit donc simplement de variables aléatoires indépendantes de Bernouilli x_i ayant chacunes une probabilité p_i

- si la proba est p_i alors $X = 1$ et donc $e^{t1} = e^t$ et,
- si la proba est $1 - p_i$ alors $X_i = 0$ et $e^{t0} = 1$.

On peut donc effectuer la transformation suivante:

$$\begin{aligned} E\left[e^{tX_i}\right] &= e^t p_i + (1 - p_i) \\ &= 1 + p_i(e^t - 1) \end{aligned}$$

Enfin, on va encore faire une ultime transformation, bien moins triviale; elle se base sur l'inégalité bien connue[xi] : $1 + x < e^x$

$$\begin{aligned} \prod_{i(1+p_i(e^t-1))} &< \prod_{i\left(e^{p_i(e^t-1)}\right)} \\ &= e^{\sum_i p_i(e^t-1)} && \text{Propriétés des exposants} \\ &= e^{(e^t-1)\mu} && \text{car } \sum_i p_i = \mu \end{aligned}$$

Remettons toutes ces transformations dans l'équation initiale:

$$\begin{aligned} P\left[e^{tX} > e^{t(1+\delta)\mu}\right] &< \prod_i \frac{1 + p_i(e^t - 1)}{e^{t(1+\delta)\mu}} \\ &< \frac{e^{(e^t-1)\mu}}{e^{t(1+\delta)\mu}} \end{aligned}$$

On doit donc maintenant une valeur pour t de sorte à obtenir la meilleure borne possible, pour ça on a juste à prendre, il s'agit d'un problème d'optimisation simple sans contraintes, on prend la valeur obtenue lorsque la dérivée est nulle. Cela donne

$$\frac{d}{dt}\left(\frac{e^{(e^t-1)\mu}}{e^{t(1+\delta)\mu}}\right) = \mu * \left(e^t - \delta - 1\right) * e^{\mu*(e^t-1)-(\delta+1)*\mu*t} = 0$$

La valeur obtimale est $e^t = \delta + 1 \Rightarrow t = \ln(\delta + 1)$ pour $\delta > 0$, en remplacant t dans l'inégalité, on a obtient notre borne:

$$\begin{aligned} P[X > (1 + \delta)\mu] &< \frac{e^{\left(e^{\ln(1+\delta)}-1\right)\mu}}{e^{\ln(1+\delta)(1+\delta)\mu}} \\ &< \frac{e^{((1+\delta)-1)\mu}}{(1 + \delta)^{(1+\delta)\mu}} \\ &< \left[\frac{e^{\delta}}{(1 + \delta)^{(1+\delta)}}\right]^{\mu} \end{aligned}$$

□

Nous noterons le membre de droite de l'inéquation

$$F(\delta, \mu) := \left[\frac{e^{\delta}}{(1 + \delta)^{(1+\delta)}}\right]^{\mu}$$

Il est important de remarquer que la valeur se trouvant entre les parenthèses doit toujours être strictement plus petit que 1.

[xi] Non

Théorème 6.3.2: Soit $X_1, ..., X_n$ un ensemble d'essais de Poisson indepéndants où $X_i \in \{0, 1\}$ et où pour tout $1 \leq i \leq n, P[X_i = 1] = p_i$ où $0 < p_i < 1$. Définissons $X = \sum_i X_i$ et $\mu = E[X] = \sum_i p_i$. Alors pour tout$0 < \delta < 1$,

$$P[X < (1 - \delta)\mu] < e^{-\mu\frac{\delta^2}{2}}$$

Le Théorème 6.3.2 permet de borner supérieurement la probabilité que la somme est un facteur $(1 - \delta)$ **plus petit** que l'espérance.
En effet, le Théorème 6.3.1 permettait de borner uniquement la probabilité que la somme des variables indépendantes de Bernoulli ne dépassent pas un facteur $(1 - \delta)$ **plus grand** que son espérance.

6.4. Bornes de Chernoff simplifiées

On peut obtenir des bornes simplifiées en jouant avec l'inégalité logarithmique $\ln(1 + x) \geq \frac{x}{1+\frac{x}{2}}$ pour tout $x > 0$ et en l'appliquant à $F(\delta, \mu)$. En effetuant ces étapes, on obtient,

$$F(\delta, \mu) \leq e^{-\mu\frac{\delta^2}{2+\delta}}$$

Il est alors nécessaire de tenir compte de la valeur de δ, en effet celle-ci ne peut prendre la valeur 2, ce qui nous donne une formulation simplifiée de la borne de Chernoff,

Proposition 6.4.1:
Si $\delta > 2$, alors

$$F(\delta, \mu) \leq e^{-\mu\frac{\delta}{2}}$$

Si $\delta < 2$, alors

$$F(\delta, \mu) \leq e^{-\mu\frac{\delta^2}{4}}$$

6.5. Boules et boîtes