Problème 3 : Le juste prix

Cercle Mathématique de Strasbourg

$\mathbb{TFJM}^2$

Résumé

Nous avons répondu entièrement à la **question 1**, pour tous les cas, que Céline joue de manière optimale ou aléatoire, et avons de plus caractérisé les stratégies optimales et calculé la moyenne du temps de jeu en jeu optimal.

Pour la **question 2**, nous avons obtenu les stratégies optimales dans le cas général pour les cas (a) et (f), avec le temps pris dans le pire des cas en ce mode de jeu, et le temps moyen pour le cas (f). Nous avons obtenus des résultats algorithmiques sur les stratégies optimales, le temps dans le pire des cas et le temps moyen pour les autres cas, pour des petits n, ce qui nous a permis de faire des conjecture sur la croissance des fonctions étudiées.

Grâce à des résultats algorithmiques pour tous les cas à la **question 3**, nous avons pu émettre des conjectures sur les mécanismes régissant l'évolution des stratégies optimales, nous permettant au passage de déterminer le temps pris dans le pire des cas au cas (f) et de proposer une méthode liée au théorème des nombres premiers pour évaluer le pire des temps au cas (a) pour des grands n.

Nous avons de plus proposé quelques **pistes de recherches**, en terme de recherche de résultats plus profonds liés au mécanismes régissant l'évolution des stratégies, ou de recherche d'indicateurs concrets plus complets, pour la gestion du risque ou la théorie des jeux, tout en fournissant une explication sur le fonctionnement des algorithmes utilisés, nous permettant au passage une exploration en profondeur de structures récursives.

Table des matières

Introduction

Dans ce jeu, Igor choisit $a \in [\![1, n]\!]$. On considère d'abord qu'il le choisit uniformément au hasard.

Céline cherche à trouver a et joue à chaque coup $b \in [\![1, n]\!]$.

Igor met à chaque coup un temps $T(b)$ à répondre avec $T : \mathbb{N} \longrightarrow \mathbb{R}_+$ une fonction fixée.

Il répond soit par $\ll a = b \gg$ si c'est le cas, et le jeu s'arrête, soit en donnant une autre information.

On étudie le temps total de jeu suivant les stratégies de Céline.

Notations

Fonctions recherchées

On s'intéresse dans ce problème à l'étude des fonctions suivantes :

Igor répond	Jeu déterministe, temps de jeu		Jeu aléatoire, temps de jeu moyen	
	Dans le pire des cas	En moyenne	Avec répétitions	Sans répétitions
$\ll = \gg$, $\ll \neq \gg$	$T_{p(i)}$	$T_{M(i)}$	$T_{M(iia)}$	$T_{M(iib)}$
$\ll = \gg$, $\ll < \gg$, $\ll > \gg$	$T_{p(2)}$	$T_{M(2)}$		
$\ll = \gg$, $\ll PGCD(a, b) \gg$	$T_{p(3)}$	$T_{M(3)}$		

Autres fonctions

On définit ici l'espérance de la variable aléatoire X qui prend ses valeurs dans l'ensemble dénombrable $\{x_i\}$, i prenant toutes les valeurs de l'ensemble I, avec $\mathbb{P}(X = x_i) = p_i$, par [1] :

$$\mathbb{E}[X] = \sum_{i \in I} x_i p_i.$$

On rappelle que l'espérance d'une variable peut se percevoir intuitivement comme la valeur moyenne de cette variable quand on répète l'expérience un très grand nombre de fois.

On note :

— $\Omega = \{u_i\}$ l'ensemble des scénarios possibles, où $u_i = (u_{i,j})_{j \in \mathbb{N}}$ est une suite de coups, c'est à dire qu'au $j^{\text{ème}}$ coup, dans le scénario u_i Céline a proposé $b = u_{i,j}$.

— U la variable de scénario, qui prend ses valeurs dans Ω.
Au $j^{\text{ème}}$ coup, Céline a proposé $b = U_j$.

— A la variable aléatoire associée au choix de a par Igor. Elle prend ses valeurs dans $[\![1, n]\!]$.
On supposera dans un premier temps que $\forall a \in [\![1, n]\!],\ \mathbb{P}(A = a) = \frac{1}{n}$.

— ω une stratégie déterministe, c'est à dire telle qu'en la suivant, $\mathbb{P}^{\omega}(U = u_i | A = a)$, est une fonction indicative sur i, càd donne 1 pour une seule valeur de i et 0 pour les autres, pour tout a fixé.

— ζ une stratégie non déterministe, appelée stratégie aléatoire.
  — $\zeta_{(iia)}$ la stratégie « Proposer b uniformément au hasard parmi tous les nombres ».

1. http ://www.bibmath.net/dico/index.php ?action=affiche&quoi=./e/esperance.html

— $\zeta_{(iib)}$ la stratégie « Proposer b uniformément au hasard les nombres non encore proposés ».
— T_t la variable aléatoire associée au temps de jeu total, fonction de U.
T_t prend ses valeurs dans $\{T_{t,i}\} \subset \mathbb{R}_+$.
— $C = \#U \in \mathbb{N}$ la variable aléatoire associée au nombre de coups total.

On s'intéresse :

— À la valeur de T_t, quand $A = a$, pour un a fixé, en suivant la stratégie déterministe ω. Cette valeur est ici déterminée et on écrit $T_t = t(\omega, a)$.
— À l'espérance $T_M(\omega)$ du temps total de jeu en suivant ω :

$$T_M(\omega) = \mathbb{E}_A^{\omega}[T_t] = \frac{1}{n}\sum_{a=1}^{n} t(\omega, a).$$

— À l'espérance de la valeur de T_t, appartenant à $\{T_{t,k}\}$, en suivant ζ, quand $A = a$, pour un a fixé.

$$\mathbb{E}_U^{\zeta}[T_t|A = a] = \sum_{T_{t,j}\in\{T_{t,k}\}} T_{t,j}\mathbb{P}^{\zeta}(T_t = T_{t,j}|A = a).$$

— À l'espérance $T_M(\zeta)$ du temps total de jeu en suivant ζ :

$$T_M(\zeta) = \mathbb{E}_A^{\zeta}[\mathbb{E}_U^{\zeta}[T_t|A = a]] = \frac{1}{n}\sum_{a=1}^{n}\sum_{T_{t,j}\in\{T_{t,k}\}} T_{t,j}\mathbb{P}^{\zeta}(T_t = T_{t,j}|A = a).$$

On évalue la qualité d'une stratégie ω ou ζ par la petitesse de la valeur de $T_M(\omega)$ ou $T_M(\zeta)$.

On appelle une stratégie déterministe ω stratégie optimale, que l'on pourra écrire ω_{opt}, si $T_M(\omega)$ est minimale parmi toutes les stratégies déterministes que l'on peut construire, pour n donné.

On peut représenter une stratégie déterministe ω par un objet de la forme :

$$\omega = [b, \omega_1, \omega_2, ...]$$

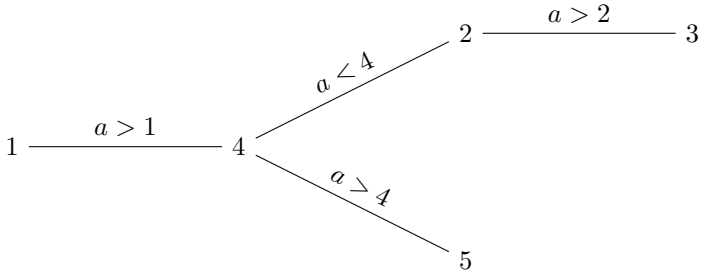
Où :

— b est le premier nombre que l'on propose.
— ω_s est la stratégie que l'on va suivre si Igor donne la $s^{\text{ème}}$ réponse autre que « $a = b$ ».

On peut représenter ces objets par des arbres, et on donne un exemple dans le cas où Igor répond par « $a = b$ » ou « $a < b$ » ou « $a > b$ ».

Voici l'arbre correspondant à la stratégie

$$\omega = [1, [4, [2, [3]], [5]]]$$



Question 1, cas où Igor répond par « $a = b$ » ou « $a \neq b$ »

Céline joue de manière optimale

Quand Igor répond par « $a = b$ » ou « $a \neq b$ », nos arbres sont des chaînes.

Il est inintéressant de poser plusieurs fois la même question, et tous les nombres $a \in [\![1, n]\!]$ doivent se retrouver sue les noeuds de la chaîne, pour qu'ils soient atteignables.

On en déduit que notre stratégie optimale se représente par une chaîne de taille n.

Dans le pire des cas

Dans le pire des cas, il faut atteindre le bout de cette chaîne, c'est à dire passer par tous les noeuds, et jouer tous les nombres de 1 à n.

On en conclut :

$$T_{p(i)}(n) = \sum_{b=1}^{n} T(b).$$

En moyenne

Chaque proposition, nous apporte la même quantité d'information : elle nous permet de supprimer un nombre de l'ensemble des valeurs de a possibles du point de vue de Céline.

À information égale, il vaut alors mieux jouer le b de plus faible coût, càd avec $T(b)$ minimal.

On nomme $(x_k)_{k\in[\![1,n]\!]},\ x_k = T(k)$, la suite donnant les temps de réponses pour chaque question, et $(y_k)_{k\in[\![1,n]\!]}$ la suite obtenue en ordonnant tous les x_k de la valeur la plus faible à la plus grande.

Alors, en suivant une stratégie optimale, si a se trouve au $i^{\text{ème}}$ noeud de l'arbre, le temps de jeu total sera :

$$\sum_{k=1}^{i} y_k.$$

On peut alors obtenir, en construisant (y_k) de la manière qui vient d'être décrite :

$$T_{M(i)}(n) = \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{i} y_k.$$

Si $T(b)$ est croissante, comme il ne sera pas nécessaire d'effectuer une permutation différente de l'identité entre (x_k) et (y_k), on pourra simplement écrire :

$$T_{M(i)}(n) = \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{i} T(k).$$

Céline joue au hasard

On rappelle :

$$T_M(\zeta) = \frac{1}{n} \mathbb{E}_A^{\zeta} \Big[\mathbb{E}_U^{\zeta}[T_t | A = a] \Big] = \frac{1}{n} \sum_{a=1}^{n} \sum_{T_{t,j} \in \{T_{t,k}\}} T_{t,j} \mathbb{P}^{\zeta}(T_{t,j} = T_{t,j} | A = a).$$

Pour obtenir ces T_M, on s'intéresse d'abord à $\mathbb{E}_U^\zeta[T_t|A=a]$.

Puisque Igor a choisi a, on peut découper le scénario en deux phases :

— Une première phase où Céline joue $C-1$ nombres tous différents de a puisque le jeu continue.
Le temps de jeu durant cette phase s'écrit $T_{\text{phase 1}}$.

— Une seconde phase où Céline joue finalement a, et le jeu s'arrête.
Le temps de jeu durant cette phase s'écrit $T_{\text{phase 2}} = T(a)$.

Ainsi, ici,

$$T_t = T_{\text{phase 1}} + T(a),$$

d'où

$$\mathbb{E}_U^\zeta[T_t|A=a] = \mathbb{E}_U^\zeta[T_{\text{phase 1}}|A=a] + T(a).$$

On fait la conjecture que durant la première phase, à chaque coup, l'espérance sur U du temps de réponse par coup T_{c1}, quand la stratégie ζ est uniforme (càd que pour la probabilité de proposer un nombre est la même pour chaque nombre jouable), s'écrit :

$$\mathbb{E}_U^\zeta[T_{c1}|A=a] = \frac{1}{n-1}\sum_{\substack{k=1\\k\neq a}}^{n} T(k).$$

Comme $T_{\text{phase 1}} = (C-1)T_{c1}$, comme C est indépendant de A en jeu aléatoire, on va supposer que l'on obtient :

$$\mathbb{E}_U^\zeta[T_{\text{phase 1}}|A=a] = \mathbb{E}_U^\zeta[C-1]\mathbb{E}_U^\zeta[T_{c1}|A=a] = (\mathbb{E}_U^\zeta[C]-1)\mathbb{E}_U^\zeta[T_{c1}|A=a].$$

D'où

$$\mathbb{E}_U^\zeta[T_t|A=a] = T(a) + (\mathbb{E}_U^\zeta[C]-1)\mathbb{E}_U^\zeta[T_{c1}|A=a] = T(a) + \frac{\mathbb{E}_U^\zeta[C]-1}{n-1}\sum_{\substack{k=1\\k\neq a}}^{n} T(k).$$

On peut à partir de là écrire :

$$T_{M(iia)}(n) = \frac{1}{n}\sum_{a=1}^{n}\ \Big(T(a) + \frac{\mathbb{E}_U^{\zeta_{(iia)}}[C]-1}{n-1}\sum_{\substack{k=1\\k\neq a}}^{n} T(k)\Big),$$

et

$$T_{M(iib)}(n) = \frac{1}{n}\sum_{a=1}^{n}\ \Big(T(a) + \frac{\mathbb{E}_U^{\zeta_{(iib)}}[C]-1}{n-1}\sum_{\substack{k=1\\k\neq a}}^{n} T(k)\Big).$$

On va donc s'intéresser à $\mathbb{E}_U^{\zeta_{(iia)}}[C]$ et $\mathbb{E}_U^{\zeta_{(iib)}}[C]$.

Jeu aléatoire avec répétitions

$$\mathbb{E}_U^{\zeta_{(iia)}}[C] = \sum_{i=1}^{+\infty} i\mathbb{P}(C=i).$$

Où

$$\mathbb{P}(C=k) = \mathbb{P}(\overline{C<k})\mathbb{P}(C=k|\overline{C<k}).$$

En jeu avec répétitions, $\mathbb{P}(C=k|\overline{C<k}) = \frac{1}{n}$.
On initialise

$$\mathbb{P}(C=1) = \frac{1}{n}.$$

.

On démontre alors aisément par récurrence sur k que

$$\mathbb{P}(C=k) = \frac{1}{n}\Big(\frac{n-1}{n}\Big)^{k-1}.$$

Donc,

$$\mathbb{E}_U^{\zeta_{(iia)}}[C] = \frac{1}{n}\sum_{i=1}^{+\infty} i\left(\frac{n-1}{n}\right)^{i-1} = \frac{1}{n}\sum_{i=0}^{+\infty}\sum_{j=i}^{+\infty}\left(\frac{n-1}{n}\right)^{j},$$

ce qui s'illustre en remaniant les termes de la somme, opération autorisée car tous ces termes sont positifs, et nous permet ensuite d'effectuer les sommes des lignes :

$$\begin{array}{cccccc}
1 & +\frac{n-1}{n} & +\left(\frac{n-1}{n}\right)^2 & +\left(\frac{n-1}{n}\right)^3 & +... \\
 & +\frac{n-1}{n} & +\left(\frac{n-1}{n}\right)^2 & +\left(\frac{n-1}{n}\right)^3 & +... \\
 & & +\left(\frac{n-1}{n}\right)^2 & +\left(\frac{n-1}{n}\right)^3 & +... \\
 & & & +\left(\frac{n-1}{n}\right)^3 & +... \\
 & & & & +...
\end{array}$$

Finalement, en faisant des sommes infinies de termes de suites géométriques de raison $q = \frac{n-1}{n}$, convergentes car $-1 < q < 1$, on obtient notre résultat.

$$\mathbb{E}_U^{\zeta_{(iia)}}[C] = n.$$

On injecte alors ce résultat dans notre formule :

$$T_{M(iia)}(n) = \frac{1}{n}\sum_{a=1}^{n}\ \Big(T(a) + \frac{\mathbb{E}_U^{\zeta_{(iia)}}[C]-1}{n-1}\sum_{\substack{k=1\\k\neq a}}^{n} T(k)\Big) = \frac{1}{n}\sum_{a=1}^{n}\ \sum_{k=1}^{n} T(k) = \sum_{k=1}^{n} T(k).$$

Ou en lien avec la question précédente,

$$T_{M(iia)}(n) = T_{p(i)}(n).$$

Jeu aléatoire sans répétitions

$$\mathbb{E}_U^{\zeta_{(iib)}}[C] = \sum_{i=1}^{n} i\mathbb{P}(C=i).$$

En jeu sans répétitions, pour $k \in [\![1,n]\!]$,

$$\mathbb{P}(C=k|\overline{C<k}) = \frac{1}{n+1-k},$$

sachant que l'évènement $\overline{C<k}$ est équivalent à $C \geq k$, qui sera ensuite utilisé.

On cherche à montrer que

$$\mathbb{P}(C=k) = \frac{1}{n},\ \forall k \in [\![1,n]\!]$$

On a

$$\mathbb{P}(C=k) = \mathbb{P}(C=k|C\geq k)\mathbb{P}(C\geq k) = \frac{\mathbb{P}(C\geq k)}{n+1-k},\ \forall k \in [\![1,n]\!].$$

Il suffit donc de montrer que

$$\mathbb{P}(C\geq k) = \frac{n+1-k}{n},\ \forall k \in [\![1,n]\!].$$

Procédons par récurrence :
Initialisation : $k=1$

$$\frac{n+1-1}{n} = 1 = \mathbb{P}(C\geq 1) = \mathbb{P}(C>0).$$

Hérédité : Supposons qu'on a, à un rang $k \in [\![1,n-1]\!]$ donné,

$$\mathbb{P}(C\geq k) = \frac{n+1-k}{n}.$$

$$\begin{aligned}\mathbb{P}(C\geq k+1) &= \mathbb{P}(C\geq k \cap C\neq k) = \mathbb{P}(C\geq k)\mathbb{P}(C\neq k|C\geq k)\\ &= \frac{n+1-k}{n}\frac{n-k}{n+1-k} = \frac{n+1-(k+1)}{n}\end{aligned}$$

Finalement, on a bien $\mathbb{P}(C=k) = \frac{1}{n},\ \forall k \in [\![1,n]\!]$.
Alors,

$$\mathbb{E}_U^{\zeta_{(iib)}}[C] = \frac{1}{n}\sum_{i=1}^{n} i = \frac{n+1}{2},$$

et on injecte alors ce résultat dans notre formule :

$$T_{M(iib)}(n) = \frac{1}{n}\sum_{a=1}^{n}\ \Big(T(a) + \frac{\mathbb{E}_U^{\zeta_{(iib)}}[C]-1}{n-1}\sum_{\substack{k=1\\k\neq a}}^{n} T(k)\Big) = \frac{1}{n}\sum_{a=1}^{n}\ \Big(T(a) + \frac{1}{2}\sum_{\substack{k=1\\k\neq a}}^{n} T(k)\Big),$$

$$T_{M(iib)}(n) = \frac{1}{2n} \sum_{a=1}^{n} \Big(2T(a) + \sum_{\substack{k=1 \\ k \neq a}}^{n} T(k) \Big).$$

Pour un c donné, $T(c)$ est alors compté $n-1$ fois à droite (pour tous les a sauf quand $a = c$), plus encore 2 fois pour $a = c$, soit $n+1$ fois au total.

On peut finalement écrire :

$$T_{M(iib)}(n) = \frac{n+1}{2n} \sum_{c=1}^{n} T(c) = \frac{n+1}{2n} T_{p(i)}(n).$$

Résultats finaux selon la définition de $T(b)$

Calcul de la somme des temps

Il reste alors à calculer $\sum_{b=1}^{n} T(b)$ dans les différents cas.

(a) Si $T(b) = 1$, alors $T_{p(i)}(n) = \sum_{b=1}^{n} 1 = n$.

(b) Si $T(b) = 2^b$, alors $T_{p(i)}(n) = \sum_{b=1}^{n} 2^b = 2^{n+1} - 2$ (somme d'une suite géométrique).

(c) Si $T(b) = b$, alors $T_{p(i)}(n) = \sum_{b=1}^{n} b = \frac{n(n+1)}{2}$ (somme d'une suite arithmétique).

(d) Si $T(b) = b^2$, alors $T_{p(i)}(n) = \sum_{b=1}^{n} b^2 = \frac{n(n+1)(2n+1)}{6}$ (formule classique).

Démonstration par récurrence :

Initialisation : Si $n = 1$, alors $\sum_{b=1}^{n} b^2 = 1$ et

$$\frac{n(n+1)(2n+1)}{6} = \frac{(1+1)(2+1)}{6} = 1.$$

On suppose qu'à un rang n donné,

$$\sum_{b=1}^{n} b^2 = \frac{n(n+1)(2n+1)}{6}.$$

Montrons alors que

$$\sum_{b=1}^{n+1} b^2 = \frac{(n+2)(n+1+1)(2(n+1)+1)}{6}.$$

$$\begin{aligned} \sum_{b=1}^{n+1} b^2 &= \sum_{b=1}^{n} b^2 + (n+1)^2 = \frac{n(n+1)(2n+1)}{6} + (n+1)^2 \\ &= \frac{n(n+1)(2n+1)}{6} + n^2 + 2n + 1 = \frac{2n^3 + n^2 + 2n^2 + n + 6n^2 + 12n + 6}{6} \\ &= \frac{2n^3 + n^2 + 9n^2 + 13n + 6}{6}. \end{aligned}$$

D'autre part

$$\begin{aligned}\frac{(n+2)(n+1+1)(2(n+1)+1)}{6} &= \frac{2n^3+3n^2+4n^2+6n+2n^2+3n+4n+6}{6}\\ &= \frac{2n^3+n^2+9n^2+13n+6}{6}.\end{aligned}$$

L'égalité est donc prouvée.

(e) Prenons maintenant $T(b) =$ le plus grand diviseur impair de b.
Pour tout entier naturel b il existe un entier naturel α et un entier naturel impair k impair tels que $b = 2^\alpha k$.
Donc $T(b) = k$.
Alors

$$T_{p(i)}(n) = \sum_{\substack{k=1,\\ k \text{ impair}}}^{n} k\#\left\{m|m<n,\ \exists\alpha,\ m=k2^\alpha\right\}.$$

Il reste à calculer $\#\left\{m|m<n,\ \exists\alpha,\ m=k2^\alpha\right\}$ pour chaque k.
Commençons par le calculer pour $\#\left\{m|m<n,\ \exists\alpha,\ m=2^\alpha\right\}$.
On pose $2^N \leq n < 2^{N+1}$, avec $N \in \mathbb{N}$.
On rappelle que $\log_2$ est la fonction réciproque de la fonction puissance de deux, c'est à dire $\log_2(x) = a \iff x = 2^a$.
Alors $N \leq \log_2(n) < N+1$.
D'où $N = \lfloor \log_2(n) \rfloor$, la partie entière inférieure de $\log_2(n)$.
On a alors tous les nombres $2^0, 2^1, 2^2 ... 2^N$ inférieurs ou égaux à n.
On a donc $N+1 = \lfloor \log_2(n) \rfloor + 1$ puissances de 2 entières inférieures ou égales à n :

$$\#\left\{m|m<n,\ \exists\alpha,\ m=2^\alpha\right\} = \lfloor \log_2(n) \rfloor + 1.$$

Calculons à présent pour d'autres k :

$$\#\left\{m|m<n,\ \exists\alpha,\ m=k2^\alpha\right\} = \#\left\{m|m<\frac{n}{k},\ \exists\alpha,\ m=2^\alpha\right\} = \left\lfloor \log_2(\frac{n}{k}) \right\rfloor + 1.$$

On en déduit que dans ce cas,

$$T_{p(i)}(n) = \sum_{\substack{k=1,\\ k \text{ impair}}}^{n} k\left(\left\lfloor \log_2(\frac{n}{k}) \right\rfloor + 1\right).$$

(f) Prenons enfin $T(b) =$ l'exposant de la plus grande puissance de 2 qui divise b.
Pour tout entier naturel b il existe un entier naturel α et un entier naturel impair k impair tels que $b = 2^\alpha k$.
Donc $T(b) = \alpha$.
On appelle $V_2(b) = \alpha$ la valuation 2-adique de b.
Ici,

$$T_{p(i)}(n) = \sum_{i=1}^{n} V_2(i) = \sum_{k=1}^{\lfloor \log_2(n) \rfloor} \#\left\{m|m<n,\ V_2(m) \geq k\right\}.$$

Ainsi, chaque nombre est compté le nombre de fois correspondant à sa valuation 2-adique, offrant le bon résultat.

De plus, aucun nombre dans $[\![1, n]\!]$ n'a de plus haute valuation 2-adique que la plus grande puissance de 2 dans cet intervalle. De la question précédente, on sait que ce nombre s'écrit 2^N, que $V_2(2^N) = N$, où $N = \lfloor \log_2(n) \rfloor$, qu'il est donc inutile de dépasser.

Il nous reste enfin à calculer $\#\{m | m < n,\ V_2(m) \geq k\}$.

$V_2(m) \geq k$ si et seulement si m est divisible par 2^k.

Le nombre de nombres divisibles par d dans $[\![1, n]\!]$ s'écrit $\left\lfloor \frac{n}{d} \right\rfloor$.

On a donc finalement :

$$T_{p(i)}(n) = \sum_{k=1}^{\lfloor \log_2(n) \rfloor} \left\lfloor \frac{n}{2^k} \right\rfloor.$$

Obtention des temps moyens en jeu optimal

On rappelle :

$$T_{M(i)}(n) = \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{i} y_k.$$

Où $(y_k)_{k \in [\![1,n]\!]}$ est une liste croissante obtenue par permutation de $(x_k)_{k \in [\![1,n]\!]},\ x_k = T(k)$. Ce qui nous donne, si $T(k)$ est croissante,

$$T_{M(i)}(n) = \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{i} T(k).$$

Pour les définitions (a), (b), (c) et (d), la fonction $T(b)$ est croissante, on va donc pouvoir utiliser cette égalité, dans laquelle on se servira des sommes des temps établies à la question précédente. On en déduit :

(a) Quand $T(b) = 1$,

$$T_{M(i)}(n) = \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{i} 1 = \frac{1}{n} \sum_{i=1}^{n} i = \frac{n+1}{2}.$$

(b) Quand $T(b) = b$,

$$T_{M(i)}(n) = \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{i} i = \frac{1}{n} \sum_{i=1}^{n} \frac{(i+1)i}{2} = \frac{1}{2n} \Big(\sum_{i=1}^{n} i^2 + \sum_{i=1}^{n} i \Big),$$

d'où

$$T_{M(i)}(n) = \frac{(n+1)(n+2)}{6}.$$

(c) Quand $T(b) = 2^b$,

$$T_{M(i)}(n) = \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{i} 2^i = \frac{1}{n} \sum_{i=1}^{n} \Big(2^{i+1} - 2 \Big) = \frac{2}{n} \Big(\sum_{i=1}^{n} 2^i - \sum_{i=1}^{n} 1 \Big),$$

d'où

$$T_{M(i)}(n) = \frac{2^{n+2} - 4 - 2n}{n}.$$

(d) Quand $T(b) = b^2$,

$$T_{M(i)}(n) = \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{i} i^2 = \frac{1}{n}\sum_{i=1}^{n}\frac{i(i+1)(2i+1)}{6} = \frac{1}{6n}\Big(2\sum_{i=1}^{n} i^3 + 3\sum_{i=1}^{n} i^2 + \sum_{i=1}^{n} i\Big).$$

On peut démontrer par récurrence la formule classique $\sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}$, mais on propose ici une preuve plus déductive :

$$\sum_{i=1}^{n} i^4 - (i-1)^4 = 1^4 - 0^4 + 2^4 - 1^4 + 3^4 - 2^4 + ... + n^4 - (n-1)^4 = n^4.$$

De plus,

$$\sum_{i=1}^{n} i^4 - (i-1)^4 = \sum_{i=1}^{n} 4i^3 - 6i^2 + 4i - 1.$$

A partir de là, on peut retrouver la somme des cubes en connaissant celle des carrés et des entiers :

$$\begin{aligned} n^4 &= 4\sum_{i=1}^{n} i^3 - 6\sum_{i=1}^{n} i^2 + 4\sum_{i=1}^{n} i - n \\ \sum_{i=1}^{n} i^3 &= \frac{1}{4}\Big(n^4 + 6\sum_{i=1}^{n} i^2 - 4\sum_{i=1}^{n} i + n\Big) \\ &= \frac{1}{4}\Big(n^4 + n(n+1)(2n+1) - 2n(n+1) + n\Big) \\ &= \frac{n^2(n+1)^2}{4} \end{aligned}$$

On utilise ce résultat, on développe, et on en déduit :

$$T_{M(i)}(n) = \frac{n^3 + 4n^2 + 5n + 2}{12}.$$

Question 2, cas où Igor répond par « $a = b$ », « $a < b$ » ou « $a > b$ »

Cas (f)

Dans cette question, Igor peut répondre à Céline par "<", "=", ou ">". Il devient donc possible d'obtenir des informations permettant à Céline d'éliminer plusieurs cas par une seule question.

On commence par considérer le cas (f), quand $T(b) = V_2(b)$.

Pour tout nombre impair k, on aura $T(k) = 0$, car un impair n'est jamais divisible par aucune autre puissance de 2 que $1 = 2^0$.

Une stratégie optimale consistera donc à jouer tous les impairs en un temps nul. Se présentent deux cas :

— a est un impair. Le jeu est terminé.

— a est un pair. Il est parfaitement connu car encadré par deux impairs, reste à le jouer, et le jeu s'arrête.

Dans les deux cas, $T_t = T(a) = V_2(a)$.

Dans le pire des cas

Quand on choisit $a \in [\![1, n]\!]$, celui donnant la plus grande valuation 2-adique est la plus grande puissance de 2 inférieure ou égale à n, càd $a = 2^N$ avec $N = \lfloor \log_2(n) \rfloor$, le pire des cas sera celui où Igor choisit $a = 2^{\lfloor \log_2(n) \rfloor}$.

Alors,

$$T_{p(2)}(n) = \lfloor \log_2(n) \rfloor .$$

En moyenne

$$T_{M(2)}(n) = \frac{1}{n} \sum_{a=1}^{n} t(\omega_{\text{opt}}, a).$$

Ici, on a donc

$$T_{M(2)}(n) = \frac{1}{n} \sum_{a=1}^{n} T(a),$$

d'où, grâce à la somme des temps déjà réalisée,

$$T_{M(2)}(n) = \sum_{k=1}^{\lfloor \log_2(n) \rfloor} \left\lfloor \frac{n}{2^k} \right\rfloor .$$

Cas (a)

On étudie ensuite le cas (a), quand $T(b) = 1$.

On pose l'hypothèse que le temps de recherche d'un nombre dans un petit intervalle est inférieur ou égal à celui dans un grand intervalle.

Démonstration de la dichotomie comme stratégie optimale

Démontrons qu'à n'importe quelle étape, la stratégie optimale pour Céline est de suivre un algorithme de dichotomie.

Posons nous dans le cas où Céline sait que $a \in [\![c_0, d_0]\!]$ avec $[\![c_0, d_0]\!]$ un intervalle quelconque.

Elle joue b. Il y a alors 3 scénarios :

— Soit Igor répond « $a = b$ » et le jeu s'arrête.

— Soit Igor répond « $a < b$ » et on sait maintenant que $a \in [\![c_0, b-1]\!]$.

— Soit Igor répond « $a > b$ » et on sait maintenant que $a \in [\![b+1, d_0]\!]$.

On nomme $[\![c_1, d_1]\!]$ le nouvel intervalle obtenu si le jeu continue.

On cherche donc à minimiser

$$\mathbb{E}_b[\#[\![c_1, d_1]\!]] = \#[\![c_0, b-1]\!]\mathbb{P}(a \in [\![c_0, b-1]\!]) + \#[\![b+1, d_0]\!]\mathbb{P}(a \in [\![b+1, d_0]\!]).$$

Cela revient à chercher le minimum de

$$M(b) = \frac{(b-c_0)^2 + (d_0-b)^2}{d_0 - c_0 + 1}.$$

Comme $d - c + 1 \in \mathbb{N}$, le minimum de $M(b)$ est atteint pour la même valeur de b que pour la fonction

$$f(b) = (b-c_0)^2 + (d_0-b)^2 = 2b^2 + b(-2c_0 - 2d_0) + (c_0^2 + d_0^2).$$

Alors, on a $f'(b) = 4b - 2c_0 - 2d_0$.

On peut alors résoudre $f'(b) = 0$ et obtenir $b = \frac{c_0+d_0}{2}$.

Cette valeur correspond bien à un minimum puisque le coefficient 2 devant b^2 est strictement positif.

Il convient de noter qu'il faut jouer dans $\mathbb{N}$, et que si $c_0 + d_0$ impair, il faudra jouer un des deux minimums équivalents, $b = \frac{c_0+d_0+1}{2}$ ou $b = \frac{c_0+d_0-1}{2}$.

Application de la stratégie optimale

On étudie maintenant le déroulement d'une partie, dans le pire des cas, avec Céline jouant de manière optimale. Initialement, on sait que $a \in [\![1, n]\!]$, donc $c_0 = 1$ et $d_0 = n$, on part donc d'un intervalle de taille $F_0 = n$.

— Si $F_0 = 2k+1$, $c_0 + d_0 = 2k+2$ est pair. Si le jeu continue, on a alors un nouvel intervalle de taille $F_1 = k$.

— Si $F_0 = 2k$, $c_0 + d_0 = 2k+1$ est impair. Dans le pire des cas, on a alors un nouvel intervalle de taille $F_1 = k$.

On passe donc de $F_0 = 2k+1$ à $F_1 = k$, et de $F_0 = 2k$ à $F_1 = k$.

On pose alors $2^N \leq F_0 < 2^{N+1}$.

— Si $F_0 = 2F_1$, on peut écrire $2^N \leq 2F_1 < 2^{N+1}$.

— Si $F_0 = 2F_1 + 1$, cela implique $2^N + 1 \leq 2F_1 + 1 < 2^{N+1} + 1$.

Dans les deux cas, on passe alors à $2^{N-1} \leq F_1 < 2^N$.

En N étapes, on arrive alors à $2^0 \leq F_N < 2^1$, soit $F_N = 1$. Il ne reste donc qu'une seule question à poser, et le jeu parvient à sa fin.

On conclut que dans le pire des cas, il faut $N + 1$ étapes pour atteindre la fin du jeu.

On a donc, puisque $T(b) = 1, \forall b \in [1, n]$,

$$T_{p(2)}(n) = \lfloor \log_2(n) \rfloor + 1$$

Question 3, cas où Igor répond par « $a = b$ » ou « $PGCD(a, b)$ »

Cas (f), $T(b) = V_2(b)$

Pour b impair, $T(b) = 0$. On peut donc jouer tous les impairs jusqu'à n avec un coût nul. On sait que l'on peut écrire $a = 2^\alpha k \in [\![1, n]\!]$ avec $\alpha \in \mathbb{N}$ et k impair.

— Si a est impair, on a atteint la fin du jeu, et $T_t = 0$.

— Si a est pair, le jeu continue, et on connaît alors parfaitement k.
On sait alors que

$$a \in \left\{ 2^\alpha k |\ \alpha \in [\![1, \left\lfloor \log_2(\frac{n}{k}) \right\rfloor]\!] \right\}.$$

— Si $\left\lfloor \log_2(\frac{n}{k}) \right\rfloor = 1$, on joue $2^1 k$, la réponse arrive en un temps $V_2(2^1 k) = 1$, et le jeu s'arrête. Alors, $T_t = 1$.

— Sinon, il faut chercher au cas par cas comment jouer de manière optimale. On toujours considérer de manière équivalente que l'on joue à partir de l'ensemble P des possibilités

$$P = \left\{ 2^\alpha |\ \alpha \in [\![1, \left\lfloor \log_2(\frac{n}{k}) \right\rfloor]\!] \right\}.$$

On remarque si les petits α ont un faible coût, les grands α apportent plus d'information : par exemple, à partir de l'ensemble $\{2, 4, 8, 16\}$, si l'on joue 8, on détermine parfaitement la réponse.

On se réfère à nos résultats algorithmiques pour la caractérisation d'une stratégie optimale à partir d'un ensemble P. Si on admet que l'algorithme fonctionne, les stratégies optimales présentées à partir le l'ensemble P sont les bonnes (on pourra mesurer des contres-exemples que l'on voudrait proposer..).

On suppose que le schéma observé au niveau des résultats algorithmiques continue pour des $\alpha_{\max}$ plus grands, et conjecture à partir de là que la stratégie optimale s'écrit :

— Jouer tous les impairs.
Le temps de jeu total est alors nul.

— Si le jeu continue, une fois k caractérisé, jouer $2^1 k$.
Le temps de jeu total vaut alors 1.

— Si le jeu continue :

— Si l'ensemble des possibles ne possède plus qu'un élément, le jouer, et le jeu s'arrête.
Le temps de jeu total vaut alors $1 + 2 = 3$.

— Sinon, jouer le deuxième plus grand nombre de l'ensemble des possibles.
Le temps de jeu total vaut alors $1 + \left(\left\lfloor \log_2(\frac{n}{k}) \right\rfloor - 1 \right) = \left\lfloor \log_2(\frac{n}{k}) \right\rfloor$.

— a est alors parfaitement caractérisé, en effet :

$$PGCD\left(2^\alpha k, 2^{\left\lfloor \log_2(\frac{n}{k}) \right\rfloor - 1} k\right) = k 2^{\min\left(\left\lfloor \log_2(\frac{n}{k}) \right\rfloor - 1, \alpha \right)}.$$

Igor aura donc répondu :

— $a = b$ si $a = 2^{\left\lfloor \log_2(\frac{n}{k}) \right\rfloor - 1} k$

— $PGCD(a, b) = 2^{\left\lfloor \log_2(\frac{n}{k}) \right\rfloor - 1} k$ si $a = 2^{\left\lfloor \log_2(\frac{n}{k}) \right\rfloor} k$

— $PGCD(a, b) = a$ sinon.

S'il le jeu continue, il reste à demander a, et le temps de jeu total vaut alors $\left\lfloor \log_2(\frac{n}{k}) \right\rfloor + \alpha$.

Ceci nous permet de donner la valeur de T_t selon a :

— Si a est impair ($\alpha = 0$),

$$T_t = 0.$$

— Si $\alpha = 1$,

$$T_t = 1.$$

— Si $\left\lfloor \log_2(\frac{n}{k}) \right\rfloor = 2$ et $\alpha = 2$;

$$T_t = 3.$$

— Si $\left\lfloor \log_2(\frac{n}{k}) \right\rfloor > 2$ et $\alpha = \left\lfloor \log_2(\frac{n}{k}) - 1 \right\rfloor$,

$$T_t = \left\lfloor \log_2(\frac{n}{k}) \right\rfloor .$$

— Si $\left\lfloor \log_2(\frac{n}{k}) \right\rfloor > 2$ et $\alpha \neq \left\lfloor \log_2(\frac{n}{k}) - 1 \right\rfloor$,

$$T_t = \left\lfloor \log_2(\frac{n}{k}) \right\rfloor + \alpha.$$

A partir de ces T_t pour tout a, on peut obtenir $T_{p(3)}$, le temps dans le pire des cas.

On voit que $\left\lfloor \log_2(\frac{n}{k}) \right\rfloor$ est maximal pour $k = 1$. On s'intéresse donc uniquement aux cas où $a = 2^\alpha$.

— Si $n = 1$,

$$T_{p(3)} = 0.$$

— Si $\lfloor \log_2(n) \rfloor = 1$, soit pour $n = 2$ ou $n = 3$,

$$T_{p(3)} = 1.$$

— Si $\lfloor \log_2(n) \rfloor = 2$, soit pour $n \in [\![4, 7]\!]$,

$$T_{p(3)} = 3.$$

— Si $\lfloor \log_2(n) \rfloor > 2$, soit $n \geq 8$, le pire des cas est atteint quand $\alpha = \lfloor \log_2(n) \rfloor$, et alors

$$T_{p(3)} = 2 \lfloor \log_2(n) \rfloor .$$

On pourra également se servir de ces résultats pour étudier $T_{M(3)}$.

Conjecture sur l'évolution des stratégies et évaluation du pire cas

Sauf dans le cas (f) déjà décrit, on remarque que les nombres premiers p ont un rôle tout particulier dans l'arbre représentant une stratégie optimale.

En effet, si p est petit, de nombreux nombres dans notre ensemble de recherche sont multiples de ce nombre premier, et on peut donc jouer $b = p$ pour séparer entre multiples de p et autres nombres.

Un produit de petits nombres premiers sera une question encore plus intéressante, à condition que la fonction $T(b)$ ne désavantage pas trop les grands nombres.

En revanche, si p est grand, il y a très peu, voir aucun nombre de la forme kp, $k \in \mathbb{N}^*$ dans notre ensemble de recherche, et on aura tout intérêt à isoler p sur une branche.

Les fonctions $T(b)$ croissantes amplifient cet intérêt pour les petits et ce rejet des grands.

Pour étudier d'autres effets, on ajoutera dans nos résultats algorithmiques l'étude de deux autres fonctions de temps, décroissantes.

On se retrouve donc souvent avec une très longue branche-chaîne sur l'arbre, dont les noeuds sont des nombres premiers « relativement grands ».

Il semble que pour la plupart des fonctions de temps « lisses », le pire des cas est de visiter cette branche jusqu'au bout.

On se retrouverait donc à calculer, avec $\mathcal{P}$ l'ensemble des premiers relativement grands entre 1 et n, $\mathcal{D}$ l'ensemble des nombres à la base :

$$T_{p(3)} = \sum_{d\in\mathcal{D}} T(d) + \sum_{p\in\mathcal{P}} T(p).$$

La question d'estimer $\mathcal{D}$ et de comprendre quand un nombre premier est « relativement grand » semble difficile et complexe, mais constitue une piste de recherche très intéressante, surtout de ses connections avec de nombreux thèmes en théorie des nombres : un pont supplémentaire entre analyse et arithmétique mérite d'être examiné.

Dans le cas (a)

Après avoir examiné les stratégies optimales obtenues algorithmiquement (cf partie suivante), on va faire l'hypothèse d'étude, hardie sans doute, que le premier coup en jeu optimal est le plus grand nombre dans $[\![1, n]\!]$, tel que ce nombre soit le plus petit commun multiple de tous les entiers d'une suite d'entiers consécutifs commençant par 1.

En effet, un tel nombre contiendrait des diviseurs variés, d'un grand nombre d'éléments de $[\![1, n]\!]$.

Si cela est vrai, on observera un changement de premier dans les stratégies optimales à partir des nombres M_s :

$$1,\ 2,\ 6 = 2\cdot3,\ 12 = 2^2\cdot3,\ 60 = 2^2\cdot3\cdot5,\ 420 = 2^2\cdot3\cdot5\cdot7,\ 840 = 2^3\cdot3\cdot5\cdot7,\ 2520 = 2^3\cdot3^2\cdot5\cdot7....$$

À chaque fois que cela advient, si le nombre obtenu est multiple d'un nouveau premier p, il faudra le supprimer de $\mathcal{P}$, ce qui entraînera une augmentation légèrement ralentie de $T_{p(3)}$.

Si l'on suppose que $\mathcal{D}$ est composé uniquement du nombre M que l'on vient de décrire, on obtient :

$$T_{p(3)} = T(M) + \sum_{p\in\mathcal{P}} T(p),\ \mathcal{P} = \text{ l'ensemble des nombres premiers qui ne divisent pas M.}$$

Et comme $T(b) = 1$, on obtient

$$T_{p(3)} = \pi_M(n) + 1,$$

où $\pi_M n = \pi(n) - D(n)$, avec $\pi(n)$ le nombre de nombres premiers de 1 à n, et $D(n)$ le plus grand nombre entre 1 et n qui est multiple de tous les nombres d'une suite d'entiers consécutifs commençant par 1.

On conjecture que $\pi(n)$ croît beaucoup plus rapidement que $D(n)$, et que donc,

$$\pi_M(n) \sim \pi(n)\ (n \to +\infty).$$

D'après le théorème des nombres premiers :

$$\pi(n) \sim \frac{n}{\ln(n)}\ (n \to +\infty).$$

Toutes ces conjectures nous permettent de proposer :

$$T_{p(3)}(n) \sim \frac{n}{\ln(n)} (n \to +\infty).$$

Résultats algorithmiques

Cas 2

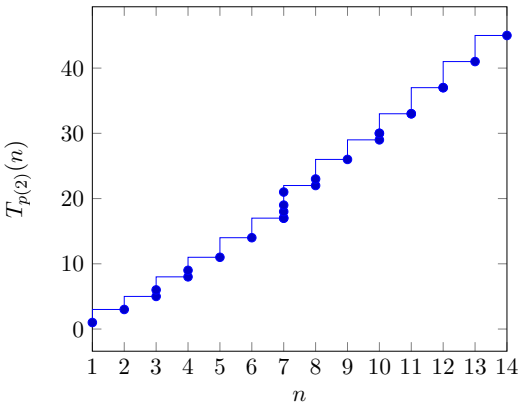
Présentons maintenant des résultats d'un algorithme présenté en annexe sous forme de graphique, avec à chaque fois, une représentation de $T_{p(2)}$, et à droite de $T_{M(2)}$.

On ajoutera en-dessous une proposition obtenue de manière expérimentale et empirique grâce au logiciel Regressi pour approximer les fonctions décrites, en notant bien sûr que notre manque de recul peut nous conduire à des erreurs (comme si l'on mesurait la courbure de la Terre ou de l'espace-temps).

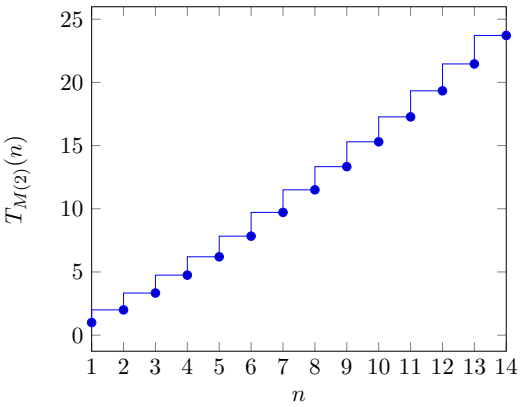
On pourra proposer plusieurs fonctions, classées des plus simples aux plus proches des quelques valeurs proposées.

On donnera également quelques exemples d'arbres correspondant à une stratégie optimale.

Cas (b), $T(b) = b$

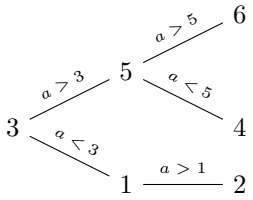


$$T_{p(2)}(n) \approx 3.48n - 5.20$$
$$T_{p(2)}(n) \approx 0.0616n^2 + 2.56n - 2.51$$

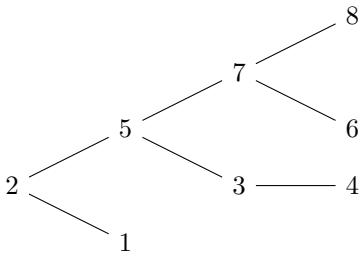


$$T_{M(2)}(n) \approx 1.77n - 2.07$$
$$T_{M(2)}(n) \approx 0.0422n^2 + 1.14n - 0.382$$

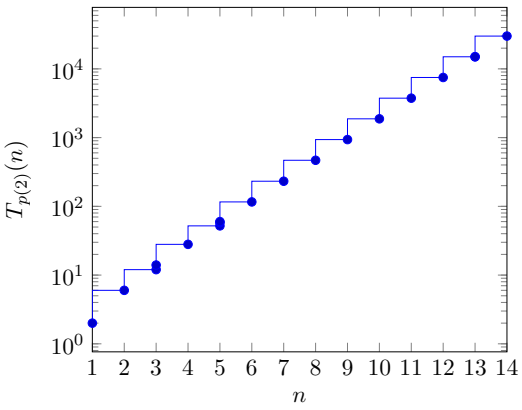
Pour $n = 6$, une stratégie optimale, avec un temps de jeu moyen de $\frac{47}{6}$ secondes et un temps de jeu dans le pire des cas de 14 secondes.



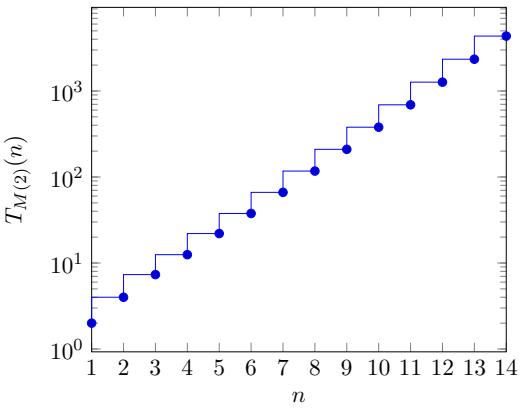
Pour $n = 8$, une stratégie optimale, avec un temps de jeu moyen de 11.5 secondes et un temps de jeu dans le pire des cas de 22 secondes.



Cas (c), $T(b) = 2^b$

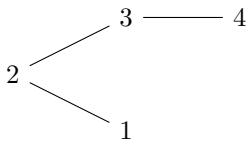


$T_{p(2)}(n) \approx 2.09^n$

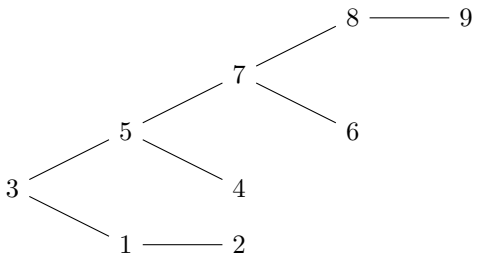


$T_{M(2)}(n) \approx 1.82^n$

Pour $n = 4$, une stratégie optimale, avec un temps de jeu moyen de 12.5 secondes et un temps de jeu dans le pire des cas de 28 secondes.

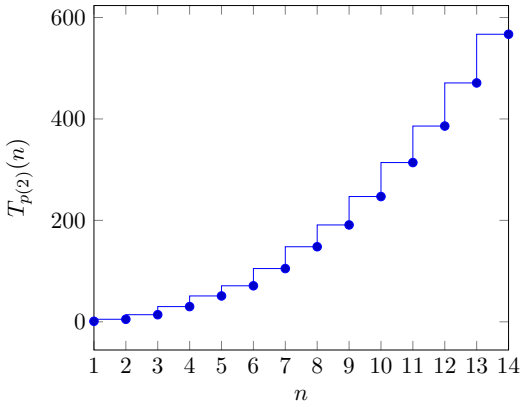


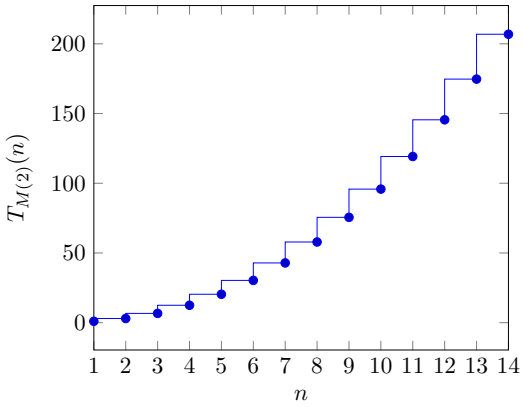
Pour $n = 9$, une stratégie optimale, avec un temps de jeu moyen de $\frac{1888}{9}$ secondes et un temps de jeu dans le pire des cas de 936 secondes.



On a la stratégie optimale [2, [1], [4, [3], [6, [5], [8, [7], [9, [10]]]]]] pour $n = 10$, dont le premier coup est 2, ce qui entame nos espoirs d'obtenir une méthode coup par coup permettant de jouer de manière optimale.

Cas (d), $T(b) = b^2$

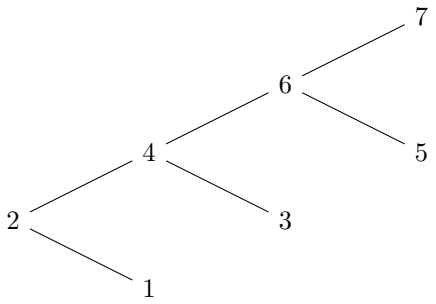




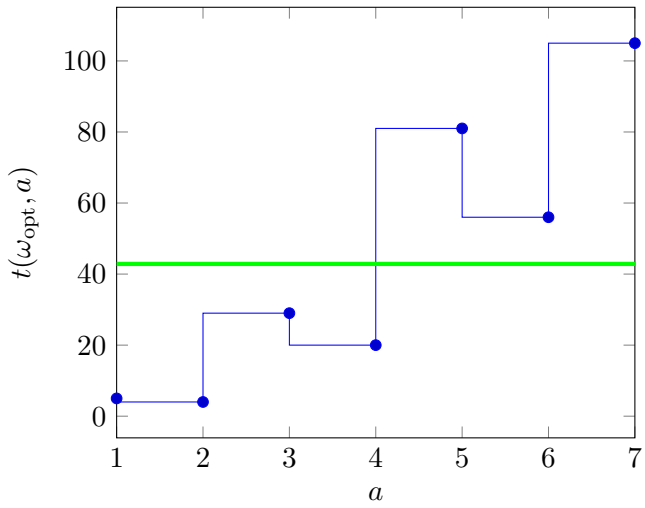
$T_{p(2)}(n) \approx 0.103n^3 + 1.45n^2 + 0.0861n - 1.13$ $\quad T_{M(2)}(n) \approx 0.0198n^3 + 0.831n^2 - 0.806n + 1.06$

Pour $n = 4$, une stratégie optimale, avec un temps de jeu moyen de 12.5 secondes et un temps de jeu dans le pire des cas de 30 secondes, est la chaîne $[1, [2, [3, [4]]]]$.

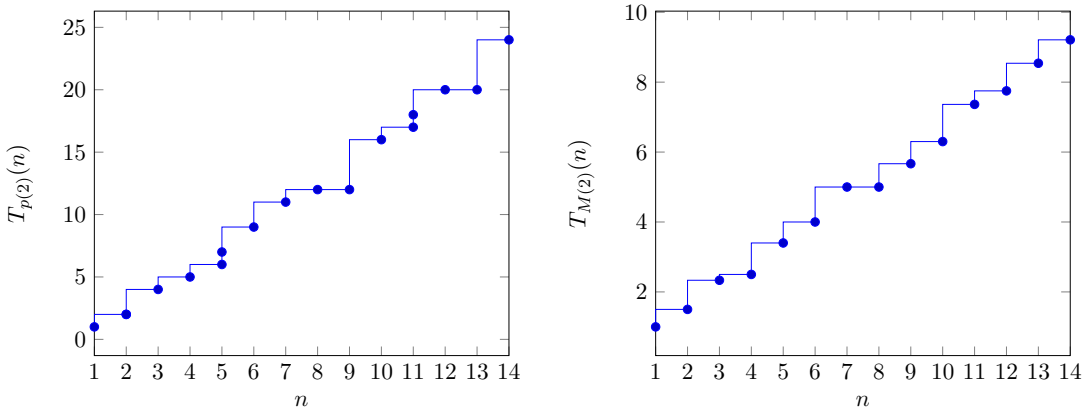
Pour $n = 7$, une stratégie optimale, avec un temps de jeu moyen d'environ 42.86 secondes et un temps de jeu dans le pire des cas de 105 secondes.



Voici un graphe donnant en abscisse la valeur de a et en ordonnée le temps total de jeu en suivant cette stratégie quand Igor a choisi a. Peut-être Céline devrait-elle chercher une stratégie avec une variance moins élevée, minimisant les risques ?

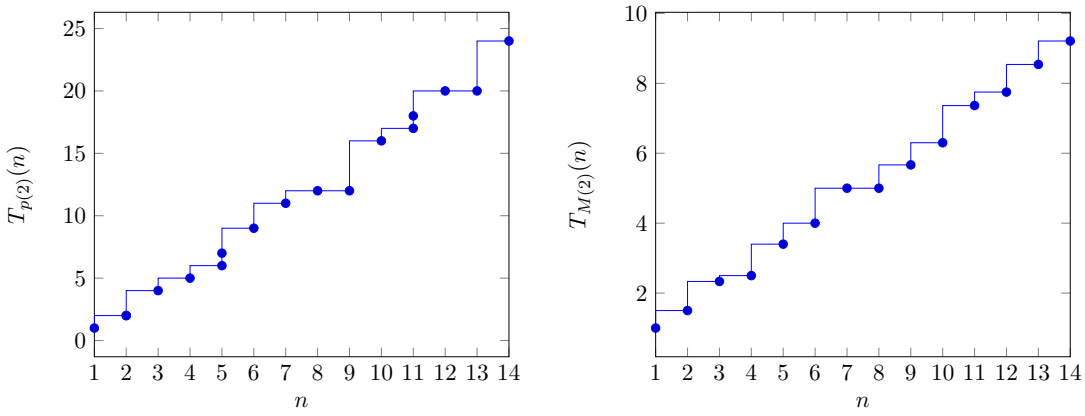


Cas (e), $T(b) =$ plus grand diviseur impair de b



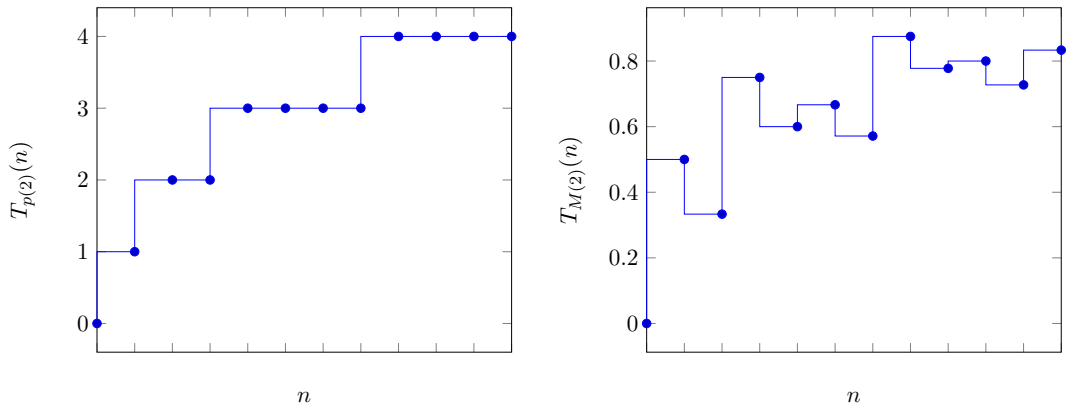
Cas (a), $T(b) = 1$

On a à gauche la fonction déjà obtenue :



Cas (f), $T(b) = V_2(b)$

On trace les fonctions déjà obtenues :



Cas 3

Cas (f) à partir de P

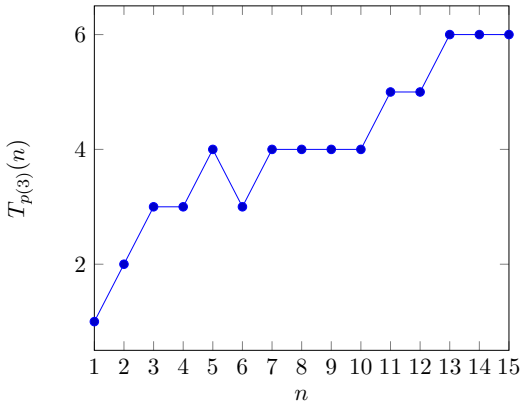
Voici quelques stratégies optimales avec $P = \{2^\alpha | \ \alpha \in [\![1, \alpha_{\max}]\!]\}$.

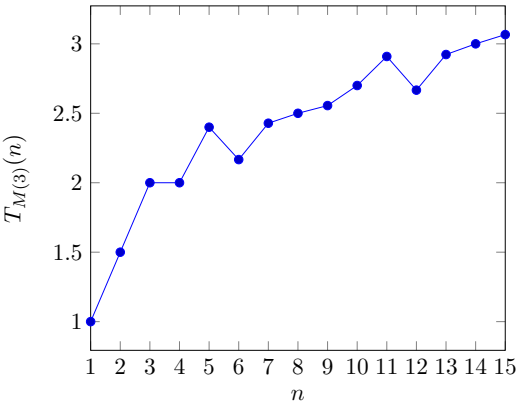
$\alpha_{\max}$	2	3	4	5
	$2 \text{ — } 4$	$2 \text{ — } 4 \text{ — } 8$	$2 \text{ — } 8 \begin{matrix} \diagup 16 \\ \diagdown 4 \end{matrix}$	$2 \text{ — } 16 \text{ — } 8 \begin{matrix} \diagup 32 \\ \diagdown 4 \end{matrix}$

$\alpha_{\max}$	6	7	8	9
	$2 \text{ — } 32 \begin{matrix} \diagup 64 \\ \diagup 16 \\ \diagdown 8 \\ \diagdown 4 \end{matrix}$	$2 \text{ — } 64 \text{ — } 16 \begin{matrix} \diagup 128 \\ \diagup 32 \\ \diagdown 8 \\ \diagdown 4 \end{matrix}$	$2 \text{ — } 128 \begin{matrix} \diagup 256 \\ \diagup 64 \\ \diagup 32 \\ \diagdown 16 \\ \diagdown 8 \\ \diagdown 4 \end{matrix}$	$2 \text{ — } 256 - 32 \begin{matrix} \diagup 512 \\ \diagup 128 \\ \diagup 64 \\ \diagup 16 \\ \diagdown 8 \\ \diagdown 4 \end{matrix}$

Cas (a), $T(b) = 1$

Dans ce cas comme dans les suivants, on présente, pour une seule stratégie optimale par n, un tracé de $T_{p(3)}$ à gauche, $T_{M(3)}$ à droite, et un arbre représentant cette stratégie dans le tableau du bas.





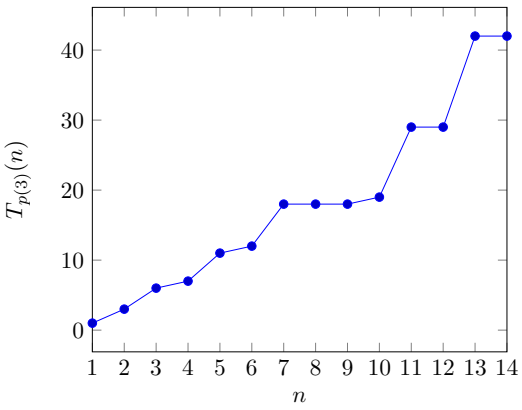
n	1	2	3	4
	1	1 — 2	1 — 2 — 3	2 → 4, 4 → 1 — 3

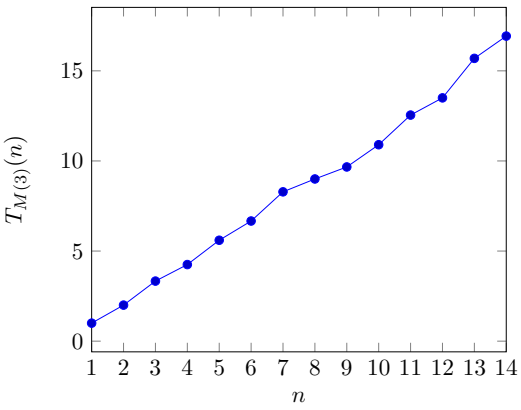
n	5	6	7	8
	2 → 4, 4 → 1 — 3 — 5	6 → 3, 6 → 2 — 4, 4 → 1 — 5	6 → 3, 6 → 2 — 4, 4 → 1 — 5 — 7	6 → 3, 6 → 4 → 8, 4 → 2, 2 → 1 — 5 — 7

n	9	10	11	12
	6 → 3 — 9, 6 — 4 → 8, 4 → 2, 2 → 1 — 5 — 7	6 → 3 — 9, 6 — 4 → 8, 4 → 2 − 10, 2 → 1 — 5 — 7	6 → 3 — 9, 6 — 4 → 8, 4 → 2 − 10, 2 → 1 — 5 — 7 − 11	12 → 6, 6 → 4 − 8, 4 → 3 − 9, 3 → 2 · 10, 2 → 1 − 5 − 7 · 11

n	13	14	15	
	12 → 6, 6 → 4 − 8, 4 → 3 − 9, 3 → 2 · 10, 2 → 1 − 5 − 7 · 11 · 13	12 → 6, 6 → 4 − 8, 4 → 3 − 9, 3 → 2 · 10 · 14, 2 → 1 − 5 − 7 · 11 · 13	12 → 6, 6 → 4 − 8, 4 → 3 − 9 · 15, 3 → 2 · 10 · 14, 2 → 1 − 5 − 7 · 11 · 13	

Cas (b), $T(b) = b$





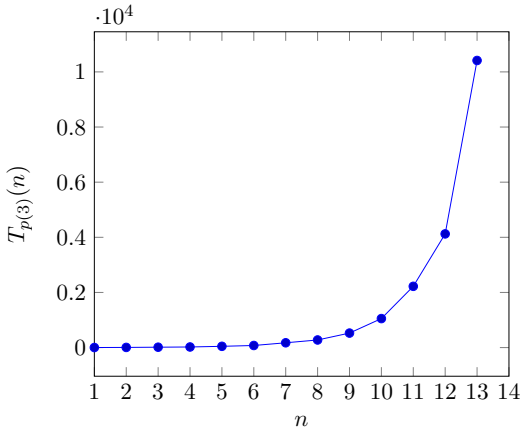
n	1	2	3	4
	1	1 – 2	1 – 2 – 3	1 – 2 < 4, 3

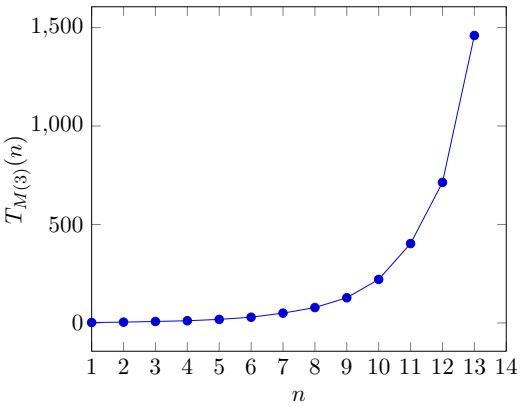
n	5	6	7	8
	1 – 2 < 4, 3 – 5	2 < 4 – 6, 1 – 3 – 5	2 < 4 – 6, 1 – 3 – 5 – 7	2 < 4 < 8, 6, 1 – 3 – 5 – 7

n	9	10	11	12
	2 < 4 < 8, 6, 1 – 3 < 9, 5 – 7	2 < 4 < 8, 3 < 6, 10, 1 – 3 < 9, 5 – 7	2 < 4 < 8, 3 < 6, 10, 1 – 3 < 9, 5 – 7 · 11	2 < 4 < 3 < 12, 8, 3 < 6, 10, 1 – 3 < 9, 5 – 7 · 11

n	13	14		
	2 < 4 < 3 < 12, 8, 3 < 6, 10, 1 – 3 < 9, 5 – 7 · 11 · 13	2 < 4 < 3 < 12, 8, 3 < 6, 5 < 10, 14, 1 – 3 < 9, 5 – 7 · 11 · 13		

Cas (c), $T(b) = 2^b$





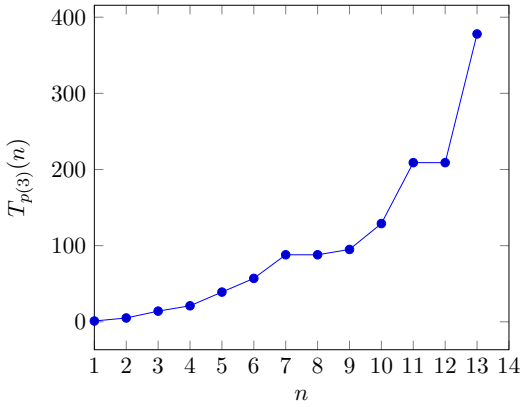
n	1	2	3	4
	1	1 – 2	1 – 2 – 3	1 – 2 < 4, 3

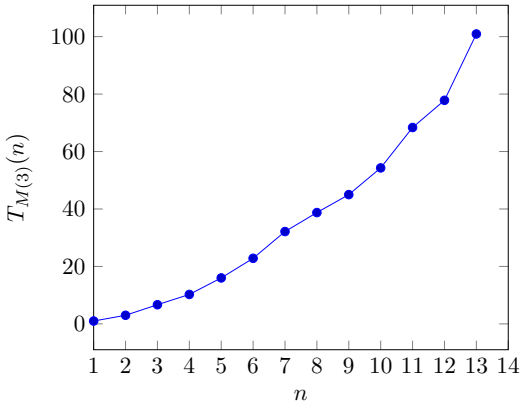
n	5	6	7	8
	1 – 2 < 4, 3 – 5	2 < 3 < 6, 4, 1 – 3 – 5	2 < 3 < 6, 4, 1 – 3 – 5 – 7	2 < 4 < 8, 6, 1 – 3 – 5 – 7

n	9	10	11	12
	2 < 4 < 8, 6, 1 – 3 < 9, 5 – 7	2 < 3 < 6, 4 < 8, 10, 1 – 3 < 9, 5 – 7	2 < 3 < 6, 4 < 8, 10, 1 – 3 < 9, 5 – 7 · 11	2 < 4 < 3 < 12, 8, 3 < 6, 10, 1 – 3 < 9, 5 – 7 · 11

n	13			
	2 < 4 < 3 < 12, 8, 3 < 6, 10, 1 – 3 < 9, 5 – 7 · 11 · 13			

Cas (d), $T(b) = b^2$





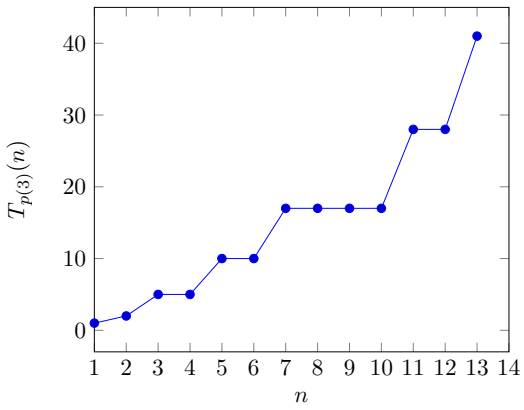
n	1	2	3	4
	1	1 – 2	1 – 2 – 3	1 – 2 < 4, 3

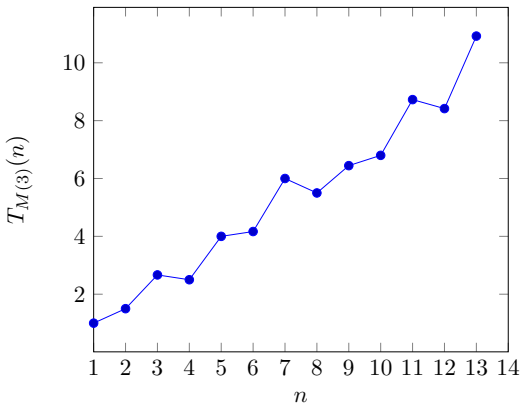
n	5	6	7	8
	1 – 2 < 4, 3 – 5	1 – 2 < 4 – 6, 3 – 5	1 – 2 < 4 – 6, 3 – 5 – 7	1 – 2 < 4 < 8, 6, 3 – 5 – 7

n	9	10	11	12
	1 – 2 < 4 < 8, 6, 3 < 9, 5 – 7	2 < 4 < 8, 6, 3 < 6, 10, 1 – 3 < 9, 5 – 7	2 < 4 < 8, 6, 3 < 6, 10, 1 – 3 < 9, 5 – 7 · 11	2 < 4 < 3 < 12, 8, 3 < 6, 10, 1 – 3 < 9, 5 – 7 · 11

n	13			
	2 < 4 < 3 < 12, 8, 3 < 6, 10, 1 – 3 < 9, 5 – 7 · 11 · 13			

Cas (e), $T(b) =$ plus grand diviseur impair de b





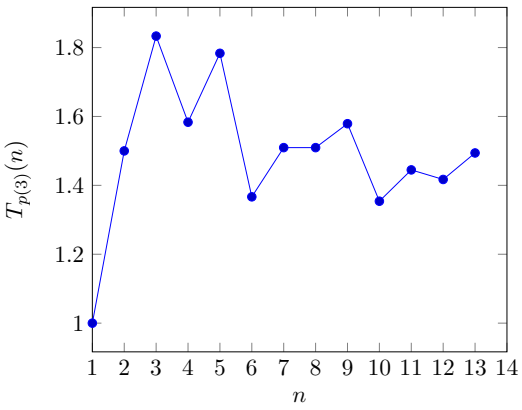
n	1	2	3	4
	1	1 – 2	1 – 2 – 3	2 ⟨ 4 1 – 3

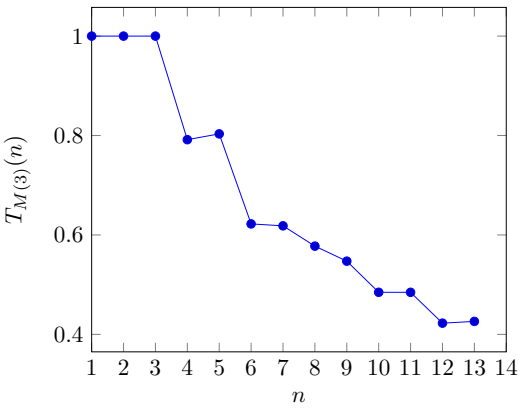
n	5	6	7	8
	2 ⟨ 4 1 – 3 – 5	2 ⟨ 4 – 6 1 – 3 – 5	2 ⟨ 4 – 6 1 – 3 – 5 – 7	8 4 – 2 – 6 1 – 3 – 5 – 7

n	9	10	11	12
	8 4 – 2 – 6 1 – 3 ⟨ 9 5 – 7	8 4 – 2 – 6 · 10 1 – 3 ⟨ 9 5 – 7	8 4 – 2 – 6 · 10 1 – 3 ⟨ 9 5 – 7 · 11	8 · 12 4 – 2 – 6 · 10 1 – 3 ⟨ 9 5 – 7 · 11

n	13			
	8 · 12 4 – 2 – 6 · 10 1 – 3 ⟨ 9 5 – 7 · 11 · 13			

Fonction additionnelle décroissante : fonction inverse



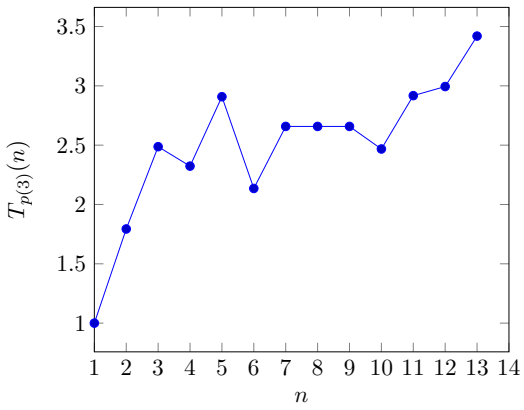


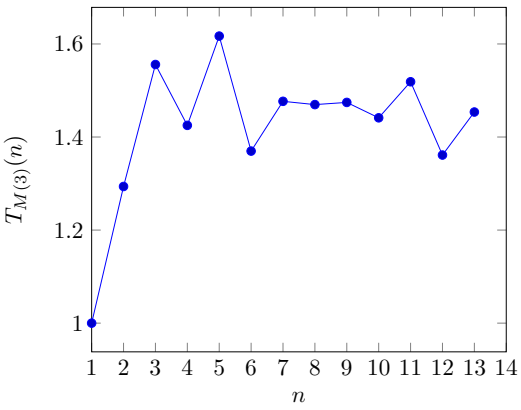
n	1	2	3	4
	1	2 - 1	3 - 2 - 1	4 < 2, 3 - 1

n	5	6	7	8
	4 < 2, 5 - 3 - 1	6 < 3, 4 - 2, 5 - 1	6 < 3, 4 - 2, 7 - 5 - 1	6 < 3, 8 < 4, 2, 7 - 5 - 1

n	9	10	11	12
	8 < 4, 6 - 2, 9 < 3, 7 - 5 - 1	10 < 5, 8 < 4, 6 - 2, 9 < 3, 7 - 1	10 < 5, 8 < 4, 6 - 2, 9 < 3, 11 - 7 - 1	12 < 6, 8 - 4, 9 - 3, 10 - 2, 11 < 10 < 5, 7 - 1

Fonction additionnelle faiblement décroissante : $T(b) = b^{-1/3}$





n	1	2	3	4
	1	2 - 1	3 - 2 - 1	4 < 2, 3 - 1

n	5	6	7	8
	4 < 2, 5 - 3 - 1	6 < 3, 4 - 2, 5 - 1	6 < 3, 4 - 2, 7 - 5 - 1	6 < 3, 8 < 4, 2, 7 - 5 - 1

n	9	10	11	12
	6 < 9 - 3, 8 < 4, 7 - 5 - 1	10 < 5, 8 < 4, 6 - 2, 9 < 3, 7 - 1	10 < 5, 8 < 4, 6 - 2, 9 < 3, 11 - 7 - 1	12 < 6, 8 - 4, 9 - 3, 10 - 2, 11 - 7 - 5 - 1

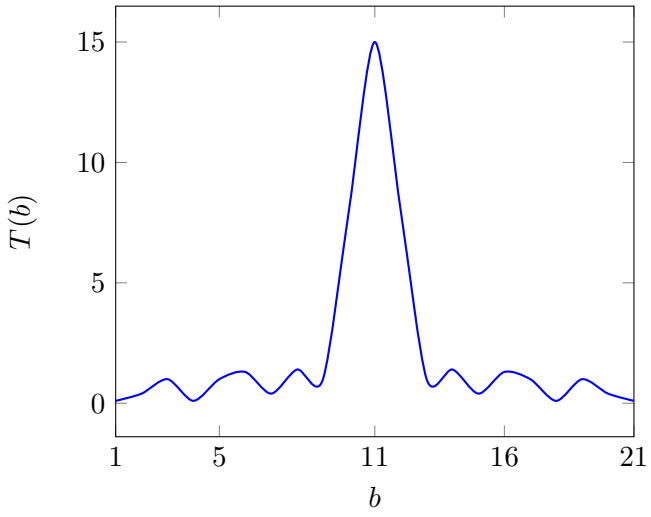
Remarques

Méthodes générales

Toute méthode générale (exacte) capable de générer des stratégies optimales, ou toutes les stratégies optimales, devra être capable trouver les bons compromis entre obtention d'information et coût de cette information.

Des stratégies intuitives que l'on pourrait nommer « Milieu des masses » ou « Milieu du carré des masses », calquées sur la solution par dichotomie du cas (a) à la question 2, ne pourront fonctionner.

Il est par exemple ici bien plus intéressant, pour minimiser la moyenne de temps de jeu, de jouer d'abord les périphéries dont le coût est quasi nul, avant le milieu, bien trop cher.



Variance et gestion du risque

On notera, comme cela peut être vu dans la lecture des résultats algorithmiques dans le cas 2, qu'il peut exister plusieurs stratégies optimales pour un même n.

Si elles donnent la même moyenne, optimisée, de temps de jeu, elle ne donnent pas nécessairement la même répartition, la même variance, le même temps dans le pire des cas. Ce sont des données plus complexes, mais qu'il faudrait prendre en compte pour une vraie prise de décision avec gestion du risque.

Théorie des jeux, nouveaux problèmes

On a dans ce problème considéré qu'Igor choisissait a uniformément au hasard.

Mais qu'en serait-il si nos deux joueurs avait des motivations ?

Si l'objectif d'Igor était de maximiser le temps de jeu, alors que celui de Céline est de le minimiser, il est à priori plus intéressant de choisir un a tel que le temps mis par Céline en jeu optimal est très long.

Cependant, si Céline le sait, elle pourra tenir compte de cette information pour modifier son jeu.

On quitte donc totalement le choix aléatoire uniforme de a, et il faut alors tenir compte de l'information qu'ont chaque joueur sur leur adversaire, ses mobiles, ses capacités.

Il faudra par exemple imaginer que, pour jouer, Céline se demandera quelles sont les informations qu'Igor a sur le jeu de Céline...

On arrive finalement dans des boucles, sans doutes fort complexes, qu'il pourra être intéressant de dénouer, ou au moins d'explorer...

Annexes

Algorithme de l'ensemble d'arbres mesurés

Dans l'algorithme implémenté en Python ci-dessus, on cherche à construire, à un rang n donné, dans le cas où Igor répond par $\ll a = b \gg$, $\ll a < b \gg$ ou $\ll a > b \gg$ (jeu en intervalle), un ensemble contenant toutes les stratégies optimales.

On appelle *stratégie valide* une stratégie telle que chaque coup joué nous apporte une information non nulle.

La structure des stratégies construite sera encore :

$$\omega = [b, \omega_1, \omega_2, ...].$$

On rappelle que l'on se représente un tel objet par un arbre.

Résumé de la méthode

On va donc construire l'ensemble de tous les arbres représentant des stratégies valides.

On va ensuite extraire **toutes** les stratégies optimales de cet ensemble (sachant que l'on pourrait procéder à d'autres mesures et extractions).

Pour ce faire, on va, pour chaque arbre de l'ensemble, le $\ll$ peser $\gg$, c'est à dire évaluer $T_{M(2)}$ en suivant cette stratégie.

Cette mesure se fait en calculant la moyenne des T_t obtenus pour tous les choix de a possibles.

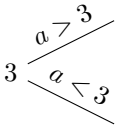
Cette extraction réalisée, on se retrouve avec l'ensemble de toutes les stratégies optimales au rang n pour une définition de $T(b)$. On les enregistre dans un fichier, accompagnées d'informations sur $T_{M(2)}$ et $T_{p(2)}$.

Description de fonctions

La fonction *ConstruireEnsembleStrategie* repose sur le fait que l'on peut ici construire un arbre valide de rang n à partir d'arbres valides de rang inférieurs.

On illustre ce fait :

Si l'on veut par exemple construire un arbre valide au rang $n = 6$, on choisit d'abord le premier coup entre 1 et 6. On choisit dans cet exemple le nombre 3.



Si le jeu continue, il y a deux réponses possibles pour Igor, à représenter par deux branches.

En bas, on peut alors choisir un nombre entre 1 et 2 : la stratégie à suivre ensuite est une stratégie au rang 2 : on en choisit une dans un ensemble déjà construit et on colle.

En haut, on peut alors choisir un nombre entre 4 et 6 : cela revient à suivre une stratégie au rang 3, mais en renommant : 1 deviendra 4, 2 deviendra 5 et 3 deviendra 6. Ce nouveau baptême est fourni par la fonction *Decaler*.

En faisant ceci pour tous les coup initiaux, et en collant toutes les combinaisons possibles d'arbres à gauche et arbre à droite, on se retrouve finalement avec notre nouvel ensemble !

En jouant ainsi, le nombre d'élément de l'ensemble d'arbres de rang n est :

$$\begin{cases} c_0 & = 1 \\ c_{n+1} & = \sum_{i=1}^{n+1} c_{i-1} c_{n+1-i}, \ \forall n \in \mathbb{N} \end{cases}$$

La suite obtenue est celle des nombres de Catalan.

Code

```
import math
import time
import sys
###
def ConstruireEnsembleStrategie(n) :
    if(n==0) :
        return [[]]
    elif(n==1):
        return [[1]]
    else :
        EnsembleStrat=[]
        EnsembleStratPrecedents=[]
        for i in range(n) : #On cherche tous les arbres de taille<n
            EnsembleStratPrecedents.append(ConstruireEnsembleStrategie(i))
        for i in range(1,n+1) :
            for j in range(1,len(EnsembleStratPrecedents[i-1])+1) :
                for k in range(1,len(EnsembleStratPrecedents[n-i])+1) :
                    EnsembleStrat.append([i,EnsembleStratPrecedents[i-1][j-1],Decaler(
                        EnsembleStratPrecedents[n-i][k-1],i)])
                    return EnsembleStrat

def Decaler(E, x) :
    if (E==0) :
        return [];
    else :
        D = E.copy()
        for i in range(len(E)) :
            if (isinstance(E[i], int)) :
                D[i]=E[i]+x
            else :
                D[i]=Decaler(E[i],x)
        return D;

def MoyenneTemps(w,n) :
    m=0
```

```
    for a in range(1,n+1) :
        m+=t(w,a)
    return float(m/n);

def PireCas(w,n) :
    pireTemps = 0
    for a in range(1,n+1) :
        tempsActuel = t(w,a)
        if tempsActuel>pireTemps : pireTemps=tempsActuel;
    return pireTemps;

def ExtraireStrategiesOptimales(E,n) :
    meilleureMoyenne = MoyenneTemps(E[0],n)
    meilleuresStrategies = [E[0]]
    if (len(E)>1) :
        for i in range(1,len(E)) :
            moyenneActuelle=MoyenneTemps(E[i],n)
            if moyenneActuelle<meilleureMoyenne :
                meilleureMoyenne = moyenneActuelle
                meilleursStrategies = [E[i]]
            elif moyenneActuelle==meilleureMoyenne :
                meilleuresStrategies.append(E[i])
    return[meilleureMoyenne, meilleuresStrategies];

def t(w, a) :
    TempsTotal=0
    while(w[0]!=a) :
        TempsTotal+=T(w[0])
        if (a<w[0]) : w=w[1];
        else : w=w[2];
    TempsTotal+=T(w[0])
    return TempsTotal;

#Definitions de T(b)

def Ta(b) :
    return 1;
def Tb(b) :
    return b;
def Tc(b) :
    return 2**b;
def Td(b) :
    return b**2;
def Te(b) :
    n=float(b)
    a=0
    while n%2 == 0 :
```

```
        a+=1
        n/=2
    return int(2**(-a)*b);
def Tf(b) :
    n=float(b)
    a=0
    while n%2 == 0 :
        a+=1
        n/=2
    return a;

def T(b) : #On modifie ici la definition de T(b)
    return Tb(b)

###

saveout = sys.stdout
fsock = open("Data\Intervalle\Tb.csv","a")
sys.stdout = fsock
print("N","Pire","Moyen","Strat",sep=';')
sys.stdout = saveout
fsock.close()

for n in range(1,14) :
    saveout = sys.stdout
    fsock = open("Data\Intervalle\Tb.csv","a")
    sys.stdout = fsock
    EnsembleStrategie = ConstruireEnsembleStrategie(n);
    StrategiesOptimales = ExtraireStrategiesOptimales(EnsembleStrategie,n);
    for i in range(len(StrategiesOptimales[1])) :
        print(n,PireCas(StrategiesOptimales[1][i],n),StrategiesOptimales[0],
            StrategiesOptimales[1][i],sep=';')
    sys.stdout = saveout
    fsock.close()
    print (n)
```

CasDeuxEnsemble.py

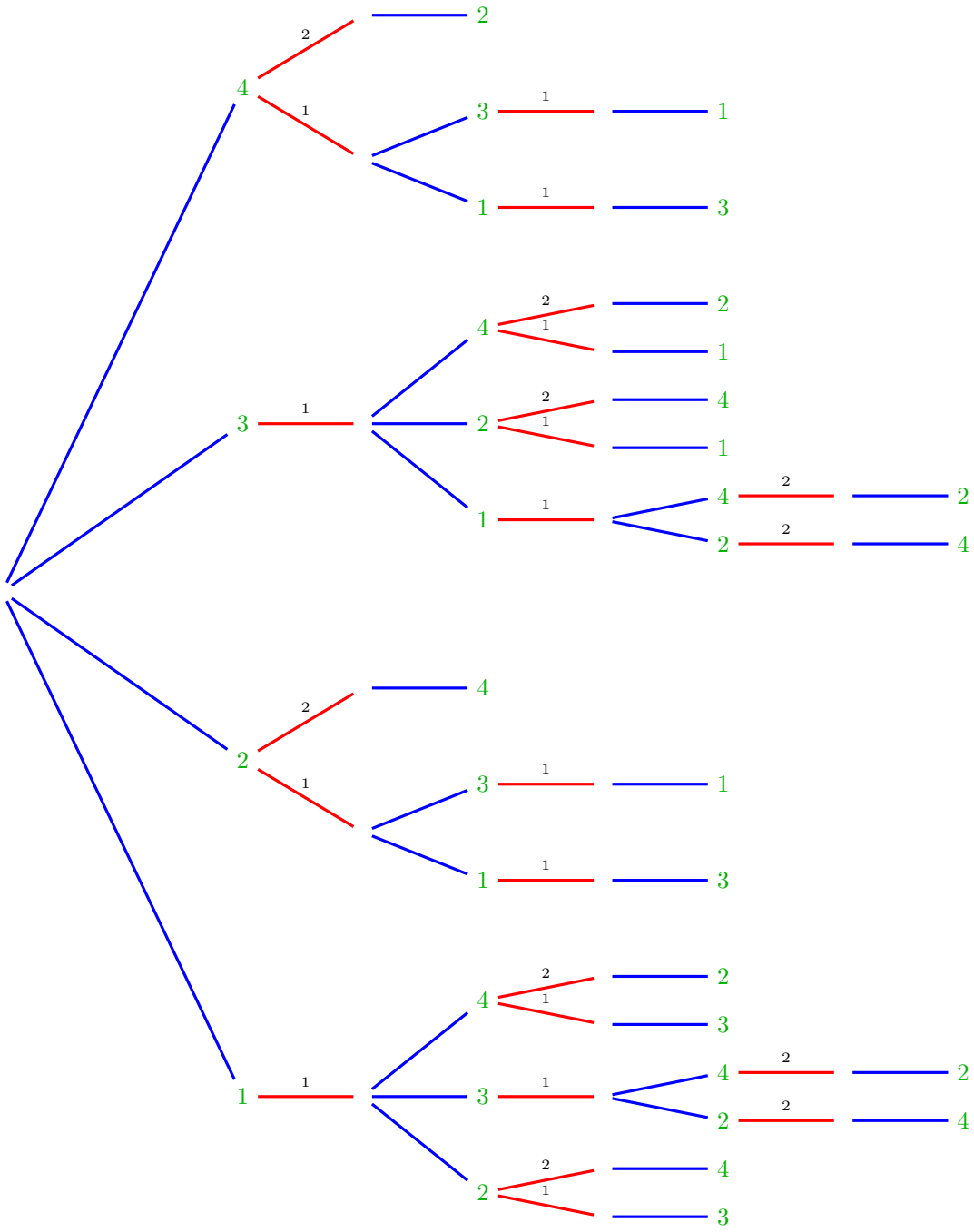
Algorithme de l'arbre noué taillé

On cherche ici à trouver *une* stratégie optimale.

Pour ce faire, on va construire un gros objet appelé *arbre noué*, sur lequel se trouvent des « branches bleues » qui représentent les choix de Céline, des « branches rouges » qui représentent les réponses possibles d'Igor, et des « noeuds verts », les nombres à proposer.

En extraire une stratégie déterministe revient à extraire un arbre rouge avec des noeuds verts en fixant à l'avance les choix de Céline.

Voici un exemple d'arbre noué pour se fixer une image mentale :



Dans le code, on stockera l'information ainsi, pour :

— Une branche bleue B_{bleue} :

$$B_{\text{bleue}} = [b, B_{\text{rouge }1}, B_{\text{rouge }2}, ...].$$

— Une branche rouge B_{rouge} :

$$B_{\text{rouge}} = [B_{\text{bleue }1}, B_{\text{bleue }2}, ...],$$

ou bien, après « écrasement » :

$$B_{\text{rouge}} = [b, B_{\text{rouge }1}, B_{\text{rouge }2}, ...].$$

Résumé de la méthode

On va donc construire l'arbre noué, et en extraire une stratégie optimale.

Pour ce faire, on va « tailler » l'arbre noué, de droite à gauche c'est à dire :

— Si il y a plusieurs branches bleues filles d'une branche rouge, on les pèse toutes, et on n'en garde qu'une de poids minimal.

— Une fois la coupe réalisée, on « écrase » : on met le contenu de la branche bleue choisie dans la branche rouge mère.

Finalement, en répétant cette méthode récursivement (on réalise en fait un parcours en profondeur), jusqu'à la pesée des branches bleues les plus à gauche et le choix d'une de poids minimale, on extrait un arbre rouge : une stratégie déterministe, optimale.

Description de fonctions

On se sert entre autre de deux variables qui nous permettent de savoir quelles branches bleues construire, et quelles sont les branches rouges à construire ou examiner (*ReponsesPossibles* fait une liste, et *ProbaReponsesPossibles* pondère).

— J, l'ensemble des b jouables : on ne construit des stratégies *valides*.

— P, l'ensemble des a possibles du point de vue de Céline.

Les fonctions *ActualiserJouables* et *ActualiserPossibilites* nous permettent de mettre à jour J et P en fonction des informations données par Igor.

Pour actualiser les possibles, on compare la réponse d'Igor avec les réponses qu'il aurait donné pour chaque nombre (sachant que si on avait donné la bonne réponse, il aurait dit « $a = b$ » ce qui ne correspond pas à une branche supplémentaire).

Pour actualiser les jouables, on s'autorise des coups qui peuvent apporter une information non nulle.

— Soit par leur présence dans P : Igor peut répondre par l'égalité ou autre chose.

— Soit par des diviseurs communs autres que 1 avec des éléments de P.

On construit ainsi l'arbre récursivement avec *ConstruireArbreNoue*.

Pour extraire une stratégie optimale, la fonction *StrategieOptimale* utilise la méthode décrite précédemment de pesées, tailles et écrasements (qui nous permettent de faire des pesées sur des stratégies optimales).

Cette fonction nous renvoie la stratégie optimale trouvée dans le sous-arbre, ainsi que le poids de cette stratégie, jusqu'à la surface.

Code

```
import math
import time
import sys
####
def pgcd(a,b) :
    r=a%b
    if r==0 :
        return b
    else :
        return pgcd(b,r)
def ConstruireArbreNoue(P,J) :
    if len(P)==1 :
        return [P[0]] #Si il ne reste qu'un a possible, on le joue
    else :
        ArbreNoue = []
        for i in range(len(J)) : #Sinon on teste tous les nombres jouables
            b = J[i]
            ArbreNoue.append([b])
            R=ReponsesPossibles(b,P)
            for j in range(len(R)):
                P2 = ActualiserPossibilites(b,P,R[j])
                J2 = ActualiserJouables(b,P2,J)
                BrancheNouee = ConstruireArbreNoue(P2,J2)
                ArbreNoue[i].append(BrancheNouee)
        return ArbreNoue
def ActualiserPossibilites(b,P,r) : #Mettre a jour P en fonction de la reponse d'Igor
    P2=[]
    for k in range(len(P)) :
        if pgcd(P[k],b)==r :
            P2.append(P[k])
    if b in P2 : P2.remove(b)
    return P2

def ActualiserJouables(b,P,J) :
    J2=[]
    J2=J.copy()
    if b in J2 : J2.remove(b)
    JTemp = J2.copy()
    for i in range(len(J2)) :
        if J2[i] not in P : #On peut jouer tout nombre qui peut etre a
            NbASupprimer = True
            class CoupInteressant(Exception) : pass
            try:
                for j in range(len(P)) :
                    if pgcd(J2[i],P[j])>1 : #Sinon, un nombre doit avoir des diviseurs
```

```
                        communs autres que 1 avec des a possibles pour apporter une
                        information
                    raise CoupInteressant
            except CoupInteressant :
                NbASupprimer = False
            if NbASupprimer :
                JTemp.remove(J2[i])
    J2=JTemp.copy()
    return J2

def ReponsesPossibles(b,P) : #Savoir combien on a de branches rouges
    R=[]
    P2=P.copy()
    if b in P2 :
        P2.remove(b)
    for i in range(len(P2)) :
        r=pgcd(P2[i],b)
        if r not in R :
            R.append(r)
    R.sort()
    return R

def ProbaReponsesPossibles(b,R,P) :
    Prob=[]
    RList=[] #Reponses pour chaque P[i]
    P2=P.copy()
    if b in P2 :
        P2.remove(b)
    for i in range(len(P2)) :
        RList.append(pgcd(P2[i],b))
    for i in range(len(R)) :
        Prob.append(RList.count(R[i]))
    return Prob #NB : n*proba

def StrategieOptimale(Arbre,P) :#Fonction de mesure + ecrasement recursif
    if isinstance(Arbre[0],int) :  #Le tronc est bleu, ou rouge ecrase
        if len(Arbre)==1 :  #Quand il n'y a qu'une feuille, on la pese.
            return [T(Arbre[0]),Arbre]
        else :  #Sinon, on pese le tronc bleu et on garde la Strategie Ecrasee en Memoire
            poids = 0.0
            R = ReponsesPossibles(Arbre[0],P)
            Count = ProbaReponsesPossibles(Arbre[0],R,P)
            StratDetSortie=[Arbre[0]] #On pose le vert au debut
            for i in range(len(R)) :#On pese les branches rouges en pondere
                RougeAnalyse=StrategieOptimale(Arbre[i+1],ActualiserPossibilites(Arbre[0],P,
                    R[i]))
                poids+=Count[i]*RougeAnalyse[0]
```

```
                StratDetSortie.append(RougeAnalyse[1])
            poids=(len(P)*T(Arbre[0])+poids)/len(P)
            return [poids,StratDetSortie]
    else :  #Le tronc est rouge (et porte des bleus)
        BleuAnalyse=StrategieOptimale(Arbre[0],P)
        meilleurPoids=BleuAnalyse[0]
        StratDuMeilleurBleu=BleuAnalyse[1]
        for i in range(1,len(Arbre)):#On taille et on ecrase
            BleuAnalyse=StrategieOptimale(Arbre[i],P)
            if BleuAnalyse[0]<meilleurPoids :
                meilleurPoids=BleuAnalyse[0]
                StratDuMeilleurBleu=BleuAnalyse[1]
        return [meilleurPoids,StratDuMeilleurBleu]

def t(Strat,a,P) : #Temps total
    StratSuivie=Strat
    PActuel=P
    TempsTotal=0.0
    b=0
    while a!=b : #On joue, Igor repond, et on parcours la Strat
        b=StratSuivie[0]
        TempsTotal+=T(b)
        if a==b :
            break;
        r=pgcd(a,b)
        R=ReponsesPossibles(b,PActuel)
        PActuel=ActualiserPossibilites(b,PActuel,r)
        StratSuivie=StratSuivie[R.index(r)+1]
    return TempsTotal

def TempsPireCas(Strat,P) :
    tmax=t(Strat,P[0],P)
    for i in range(1,len(P)) :
        ti=t(Strat,P[i],P)
        if ti>tmax :
            tmax=ti
    return tmax

###Definitions de T(b)

def Ta(b) :
    return 1;
def Tb(b) :
    return b;
def Tc(b) :
    return 2**b;
def Td(b) :
```

```
    return b**2;
def Te(b) :
    n=float(b)
    a=0
    while n%2 == 0 :
        a+=1
        n/=2
    return int(2**(-a)*b);
def Tf(b) :
    n=float(b)
    a=0
    while n%2 == 0 :
        a+=1
        n/=2
    return a;
def Th(b) :
    return 1.0/b
def Tj(b):
    return b**(-0.5)

def T(b):
    return Ta(b)    #On modifie la definition de T(b) ici
###
saveout = sys.stdout
fsock = open("Ta.csv","a")
sys.stdout = fsock
print("N","Pire","Moyen","Strat","PerfAlgo",sep=';')
sys.stdout = saveout
fsock.close()
for n in range(1,20) :
    start_time = time.time()
    saveout = sys.stdout
    fsock = open("Ta.csv","a")
    sys.stdout = fsock
    P=[] ###Possibilites pour a et possibilite de jeu au debut
    for i in range(1,n+1) : P.append(i)
    ArbreNoue=ConstruireArbreNoue(P,P)
    StratOpt=StrategieOptimale(ArbreNoue,P)
    PireTemps=TempsPireCas(StratOpt[1],P)
    print(n,PireTemps,StratOpt[0],StratOpt[1],time.time() - start_time,sep=';')
    sys.stdout = saveout
    fsock.close()
```

CasTroisTaille.py