

# ALGORITHMIQUE ET STRUCTURES DE DONNÉES

## TP 1. Introduction à l'algorithmie et la complexité

26 Janvier 2026

### 1 Le grand saut (source: [1])

Le problème est de déterminer à partir de quel étage d'un immeuble, sauter par une fenêtre est fatal. Vous êtes dans un immeuble à  $n$  étages (numérotés de 1 à  $n$ ) et vous disposez de  $k$  étudiants. Les étudiants sont classés par notes de partiel croissantes. Il n'y a qu'une opération possible pour tester si la hauteur d'un étage est fatale : faire sauter le premier étudiant de la liste par la fenêtre. S'il survit, vous pouvez le réutiliser ensuite (évidemment, l'étudiant survivant reprend sa place initiale dans la liste triée), sinon vous ne pouvez plus.

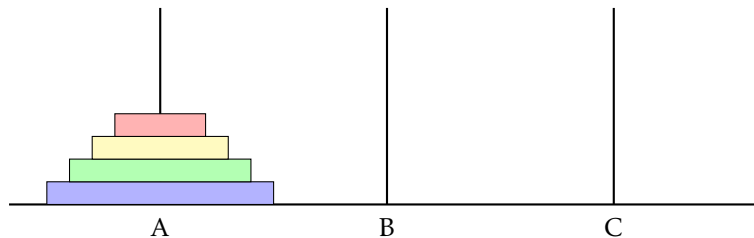
Vous devez proposer un algorithme pour trouver la hauteur à partir de laquelle un saut est fatal (renvoyer  $n + 1$  si on survit encore en sautant du  $n$ -ème étage) en faisant le minimum de sauts.

**Question 1.** Si  $k \geq \lceil \log_2(n) \rceil$ , proposer un algorithme en  $\mathcal{O}(\log_2(n))$  sauts.

**Question 2.** Si  $k < \lceil \log_2(n) \rceil$ , proposer un algorithme en  $\mathcal{O}\left(k + \frac{n}{2^{k-1}}\right)$  sauts.

**Question 3.** Si  $k = 2$ , proposer un algorithme en  $\mathcal{O}(\sqrt{n})$  sauts.

### 2 Tours d'Hanoï



Les tours d'Hanoï sont un jeu contenant trois piliers et  $n$  disques de tailles différentes. Initialement les disques se trouvent autour du premier pilier (A) triés par tailles. Le plus grand disque est en bas et le plus petit en haut. Le but du jeu est de déplacer tous les disques sur le troisième pilier (C). Pour cela, on peut retirer le disque se trouvant en haut d'un pilier pour le déplacer sur un autre pilier à condition de le poser sur un disque plus grand. Il n'est donc pas possible de poser le disque vert sur le disque jaune dans l'illustration.

Le but de cet exercice est de construire un algorithme résolvant ce problème et d'analyser sa complexité.

**Question 1.** Résoudre le problème à la main pour  $n = 1$ ,  $n = 2$  et  $n = 3$ . Compter le nombre de déplacements. Remarquez vous un paterne ?

**Question 2.** Supposons que l'on sache résoudre le problème pour  $n - 1$  disques entre n'importe quelle paire de piliers. Posez vous les questions suivantes pour déplacer  $n$  disques du pilier A au pilier C :

- Lorsque je pourrais déplacer le plus grand disque, où dois-je le mettre ?
- Pour que cela soit possible que faut-il que je fasse avant ?
- Une fois le grand disque déplacer, que faut-il ensuite que je fasse ?

En déduire un algorithme récursif pour résoudre le problème.

**Question 3.** Implémenter cette solution.

**Question 4.** Posons  $C(n)$  le nombre déplacements effectués par votre algorithme pour  $n$  disques. Donner une formule de récurrence sur  $C$ . Calculer alors  $C(n)$ .

### 3 Produit de deux polynômes

Le but est de multiplier deux polynômes le plus efficacement possible. On note  $\mathbb{P}_n[X]$  l'ensemble des polynômes de degré strictement inférieur à  $n$  et de variable  $X$ . Soient  $P, Q \in \mathbb{P}_n[X]$  tel que :

$$P(X) = \sum_{i=0}^{n-1} a_i X^i, \quad Q(X) = \sum_{i=0}^{n-1} a_i X^i$$

On cherche à calculer  $R = P \cdot Q$  dans  $\mathbb{P}_{2n-1}$ . Pour rappel,  $R$  peut s'écrire :

$$R(X) = \sum_{i=0}^{2n-2} c_i X^i, \quad \text{avec } c_i = \sum_{j=0}^i a_j b_{i-j}, \quad \text{où } a_j = b_j = 0 \text{ pour } j \geq n$$

**Question 1.** Quelle est la complexité d'un algorithme naïf calculant le produit de deux polynômes de degré strictement inférieur à  $n$ .

**Question 2.** Essayons désormais de faire mieux. Supposons  $n$  pair et  $n = 2m$ . On décompose alors  $P$  et  $Q$  comme suit :

$$\begin{cases} P &= P_1 + X^m \cdot P_2 \\ Q &= Q_1 + X^m \cdot Q_2 \end{cases}, \quad \text{avec } P_1, P_2, Q_1, Q_2 \in \mathbb{P}_m[X]$$

Exprimer  $R$  en fonction de  $P_1, P_2, Q_1, Q_2$ .

**Question 3.** On introduit maintenant 3 nouvelles valeurs :

$$\begin{cases} R_1 &= P_1 \cdot Q_1 \\ R_2 &= P_2 \cdot Q_2 \\ R_3 &= (P_1 + P_2) \cdot (Q_1 + Q_2) \end{cases}$$

Exprimer  $R$  en fonction de  $R_1, R_2, R_3$ .

**Question 4.** En remarquant que  $R_1, R_2$  et  $R_3$  sont obtenus par produits de polynômes de  $\mathbb{P}_m[X]$  avec  $m = \frac{n}{2}$ , en déduire un algorithme récursif permettant de calculer le produit de deux polynômes de degré strictement inférieur à  $n$ . Exprimer la complexité de cet algorithme à l'aide d'une formule de récurrence.

**Question 5.** Appliquer le théorème maître pour obtenir la complexité.

**Remarque** Il existe un algorithme permettant de multiplier deux polynômes de degrés strictement inférieur à  $n$  avec une complexité en  $\mathcal{O}(n \log(n))$ . Cet algorithme s'appuie sur la transformée de Fourier rapide (FFT) [voir [https://fr.wikipedia.org/wiki/Transformation\\_de\\_Fourier\\_rapide](https://fr.wikipedia.org/wiki/Transformation_de_Fourier_rapide)].

## 4 Élément majoritaire (source: [1])

Soit  $E$  un ensemble de  $n$  éléments rangés dans un tableau numéroté de 0 à  $n-1$ . On suppose que la seule opération qu'on sait effectuer sur les éléments est de vérifier si deux éléments sont égaux ou non. On dit qu'un élément  $x \in E$  est majoritaire si l'ensemble  $E_x = \{y \in E \mid y = x\}$  a strictement plus de  $n/2$  éléments. On suppose que  $n$  est une puissance de 2, et on s'intéresse à la complexité dans le pire des cas.

**Question 1.** Écrire un algorithme calculant le cardinal de  $E_x$  pour un  $x$  donné. En déduire un algorithme naïf pour vérifier si  $E$  possède un élément majoritaire. Quelle est la complexité de cet algorithme ?

**Question 2.** Donner un autre algorithme récursif basé sur un découpage de  $E$  en deux tableaux de même taille. Quelle est sa complexité ?

**Question 3.** Pour améliorer l'algorithme précédent, on va se contenter dans un premier temps de mettre au point un algorithme possédant la propriété suivante :

- soit l'algorithme garantit que  $E$  ne possède pas d'élément majoritaire,
- soit l'algorithme fournit un entier  $p > n/2$  et un élément  $x$  tels que  $x$  apparaisse au plus  $p$  fois dans  $E$  et tout élément autre que  $x$  apparaît au plus  $n-p$  fois dans  $E$ .

Donner un algorithme récursif possédant cette propriété. Quelle est sa complexité ?  
En déduire un algorithme efficace vérifiant si  $E$  possède un élément majoritaire.

## References

- [1] ANNE BENOIT, BENJAMIN DEPARDON, C. M. C. R. Algorithmique i - cours et travaux dirigés.  
<https://graal.ens-lyon.fr/~abenoit/algo09/poly09.pdf>, Septembre 2009.