- Notes de cours -

Présentation du module

Objectifs du cours. Ce cours fait suite au cours d'algorithmique de L2. Son but est de poursuivre l'étude de la conception et de l'analyse d'algorithmes. Les problèmes algorithmiques que nous étudierons viendront de différents domaines : traitement informatique, recherche oprérationnelle, logique, graphes, algèbre, combinatoire...

Intervenants. S. Bessy (responsable du module), M. Bougeret, A. Perret Du Cray, L. Isenmann (mail : `prenom.nom@umontpellier.fr`)

Contenu. Quatre chapitres principaux :
- **Algorithmes exhaustifs et backtrack**,
- **Analyse en moyenne et algorithmes probabilistes**,
- **Hachage** et
- **Algorithmes d'approximation**.

Planning (a priori...) Les cours (10 séances d'1h30) ont lieu le mercredi à 13h15, les tds et tps (10 séances de 3h) le mardi à 8h00 ou le jeudi matin à 9h45 (selon votre groupe).
L'emploi du temps exact est à consulter sur l'ENT.

Sem.	**Date**	**Contenu**
37	Me 14/09	**CM1 :** *Chapitre 1 :* **Complexité et Probabilités : aléatoire en algorithmique**
38	Me 21/09	**CM2 :** *Chapitre 2 :* **Recherche exhaustive et backtrack** **TD1 :** Complexité, aléatoire en algorithmique
39	Me 28/09	**CM3 :** Fin du *Chapitre 2* **TP1 :** Algorithmes probabilistes
40	Me 05/10	**CM4 :** *Chapitre 3 :* **Analyse amortie, en moyenne et algorithmes probabilistes** **TD2 :** Recherche exhaustive
41	Me 12/10	**CM5 :** Fin du *Chapitre 3* **TP2 :** Recherche exhaustive
42	Me 19/10	**CM6 :** *Chapitre 4 :* **Hachage** **TD3 :** Analyse amortie, en moyenne et algos probabilistes
43	Me 26/10	**CM7 :** Fin du *Chapitre 4* **TD4 :** Fin des Algos probabilistes et début des exos sur le Hachage
44	Me 02/11	VACANCES D'AUTOMNE
45	Me 09/11	**CM8 :** **!!! Controle continu, type exam !!!** (Contenu : Chap. 1,2 et 3) **TD5 :** Fin des exercices sur le Hachage
46	Me 16/11	**CM9 :** *Chapitre 5 :* **Algorithmes d'approximation** **TP3 :** Hachage
47	Me 23/11	**CM10 :** Fin du *Chapitre 6* **TD6 :** Algorithmes d'approximation
14	Me 30/11	**TP4 :** Algorithmes d'approximation

Modalité de contrôle de connaissance. L'évaluation comportera un contrôle continu et un examen. Le contrôle continu est composé d'une partie type exam (sur le créneau de cours du 9 novembre a

priori) et d'une bonification sur l'avancée en tp (au plus 2pts). Le contrôle continu compte pour 30% dans la note finale avec la 'règle du max', c'est-à-dire que la note finale sera $\max\{exam; 0.7 \times exam + 0.3 \times cc\}$. Une deuxième session d'examen est prévue, pas de contrôle continu.
Pour les exams et le contrôle continu, les seuls documents autorisés sont ces notes de cours.

Prérequis. D'un point de vue algorithmique, sont attendues des connaissances sur les instructions classiques en pseudo-code, la récursivité, les heuristiques classiques de complexité (glouton, diviser pour régner, programmation dynamique), le calcul de la complexité d'un algorithme et la connaissance des structures de données usuelles (tableaux, piles, files, liste chaînées, structures arborescentes).
En programmation, les tp se feront en Python. Il faut maîtriser a minima le langage.

Ressource. Les ressources pédagogiques seront disponibles sur le moodle de l'Université (HAI503I).
Les ouvrages suivant contiennent l'essentiel du cours (et même plus...) :

- T.H. Cormen, C.E. Leiserson, R. Rivest and C. Stein. **Introduction to Algorithms**, 3rd Edition, *MIT Press*, 2009 (une version française existe).
- S. Dasgupta, C. Papadimitriou and U. Vazirani. **Algorithms**, *McGraw-Hill Higher Education*, 2006 (version électronique gratuite).
- J. Erickson. **Algorithms**, *University of Illinois at Urbana-Champaign*, 2019, ainsi que le chapitre supplémentaire sur les probabilités (version électronique gratuite).
- J. Kleinberg, É. Tardos. **Algorithm design**, *Pearson education*, 2005.
- V.V. Vazirani. **Approximation algorithms**, *Springer*, 2001.

1 Rappels de complexité, probabilité

1.1 Rappels de complexité

- **Une notation de Landau** : Soient $f, g : \mathbb{N} \longrightarrow \mathbb{R}^+$. On dit que $\boldsymbol{f = O(g)}$ si il existe une constante $c > 0$ et un rang $n_0 \in \mathbb{N}$ tels que : $\forall n \geq n_0$ on ait $f(n) \leq c.g(n)$

Lemme 1 (O et limites) *Pour $f : \mathbb{N} \longrightarrow \mathbb{R}$ et $g : \mathbb{N} \longrightarrow \mathbb{R}^{+*}$, si il existe une constante $c \geq 0$ telle que $\frac{f(n)}{g(n)} \underset{n\to+\infty}{\longrightarrow} c$ alors on a $f = O(g)$. Et si $\frac{f(n)}{g(n)} \underset{n\to+\infty}{\longrightarrow} +\infty$ alors on a $f \neq O(g)$.*

Lemme 2 (Limites comparées) *- Pour $\alpha, \beta > 0$ on a :*
$\frac{(\log n)^\alpha}{n^\beta} \underset{n\to+\infty}{\longrightarrow} 0$ $\qquad$ $\frac{n^\alpha}{(2^n)^\beta} \underset{n\to+\infty}{\longrightarrow} 0$ $\qquad$ $\frac{(2^n)^\alpha}{n!} \underset{n\to+\infty}{\longrightarrow} 0$
- Soit $u(n)$ est une fonction telle que $u(n) \underset{n\to+\infty}{\longrightarrow} \infty$ (typiquement, $u(n) = n$ ou $u(n) = \log n$ ou $u(n) = 2^n$). Si $a.X^p$ et $b.X^q$ avec $a > 0$ et $b > 0$ sont respectivement les termes de plus haut degré de deux polynomes P et Q alors $\lim_{n\to\infty} P(u(n))/Q(u(n))$ vaut : 0 si $q > p$, a/b si $q = p$ et $+\infty$ si $p > q$.

Lemme 3 (Règles de calcul pour le $\log$) *Pour $a, b \in \mathbb{R}^{+*}$ et $c \in \mathbb{R}$, on a :*

$\log 0$ *est non défini*	$\log 1 = 0$	$\log 2 = 1$
$\log(a \times b) = \log a + \log b$	$\log(\frac{a}{b}) = \log a - \log b$	$\log(a^c) = c \times \log a$

Lemme 4 (Règles de calcul pour l'$\exp$) *Pour $a, b \in \mathbb{R}^{+*}$, on a :*

$2^0 = 1$	$2^{a+b} = 2^a \times 2^b$	$2^{a\times b} = (2^a)^b$
$2^{\log a} = a$	$\log 2^a = a$	

Lemme 5 (Sommes arithmétique et géométrique) *Pour $a, b \in \mathbb{N}$ avec $a \leq b$ et $x \in \mathbb{R} \setminus \{1\}$, on a :*

$$\sum_{i=a}^{b} i = (b - a + 1).\frac{b + a}{2} \qquad \textit{et} \qquad \sum_{i=a}^{b} x^i = \frac{x^{b+1} - x^a}{x - 1}$$

Théorème 1 ('Master Theorem') *Si il existe trois entiers $a \geq 0$, $b > 1$ et $d \geq 0$ tels que pour tout $n > 0$ on ait $t(n) \leq a.t(\lceil n/b \rceil) + O(n^d)$ alors :*
- $t(n) = O(n^d)$ si $b^d > a$ (c'est-à-dire si $d > \log a / \log b$),
- $t(n) = O(n^d \log n)$ si $b^d = a$ (c'est-à-dire si $d = \log a / \log b$), ou
- $t(n) = O(n^{\frac{\log a}{\log b}})$ si $b^d < a$ (c'est-à-dire si $d < \log a / \log b$).

1.2 Rappel de probabilités discrètes

- L'ensemble des résultats possibles d'une expérience probabiliste est appelé son **univers**, souvent noté Ω. Un **évènement primitif** (ou **élémentaire**) est un élément de l'univers, c'est-à-dire, un résultat possible. Plus généralement, un **évènement** est un sous-ensemble de l'univers, donc un ensemble de résultats possibles.
- Étant donné un univers Ω, une **probabilités** sur Ω est une valeur pour chaque élément primitif x, notée $\Pr[x]$, telle que $\sum_{x\in\Omega} \Pr[x] = 1$.
  La **probabilité d'un évènement** est la somme des probabilités de ses éléments.

- Si tous les évènements primitifs d'un univers ont la même probabilité (c'est-à-dire que pour tout $x \in \Omega$ on a $\Pr[x] = 1/\Omega$), on parle de **probabilité uniforme** sur Ω. Dans ce cas, un évènement A a pour probabilité $|A|/|\Omega|$ ('nombres de cas favorables/nombre de cas possibles').
- Une **variable aléatoire** est une fonction $X : \Omega \to V$ avec $V \subseteq \mathbb{R}$.
  "$X = v$" est l'évènement $\{\omega \in \Omega : X(\omega) = v\}$ et $\Pr[X = v] = \sum_{\omega : X(\omega)=v} \Pr[\omega]$
  "$X \leq v$" est l'évènement $\{\omega \in \Omega : X(\omega) \leq v\}$ et $\Pr[X \leq v] = \sum_{\omega : X(\omega)\leq v} \Pr[\omega]$
- Soit $X : \Omega \to V$ une variable aléatoire. **L'espérance de X** est : $\mathbb{E}[X] = \sum_{v\in V} v \times \Pr[X = v]$.
- La **probabilité de E sachant F** est $\Pr[E \mid F] = \frac{\Pr[E\cap F]}{\Pr[F]}$. L'**espérance de X sachant F** est $\mathbb{E}[X \mid F] = \sum_{v\in V} t \times \Pr[X = v \mid F]$.
- Deux évènements sont **indépendants** si $\Pr[E \wedge F] = \Pr[E].\Pr[F]$. Deux variables aléatoires sont indépendantes si $\Pr[X = u \wedge Y = v] = \Pr[X = u].\Pr[Y = v]$ pour tous u, v.

Théorème 2 (Propriétés) *Soient E et F deux évènements et $X, Y : \Omega \to V$ deux variables aléatoires. On a les propriétés suivantes :*
- $Pr[\neg E] = 1 - Pr[E]$
- $Pr[E \vee F] \leq Pr[E] + Pr[F]$ — **Inégalité de Boole** *ou* **'Union bound'**
- $\sum_{v\in V} Pr[X = v] = 1$
- $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$ — **Linéarité de l'espérance**

Si, de plus, $\Omega = \bigsqcup_i F_i$, partition de Ω en $(F_i)_i$, on a aussi :
- $Pr[E] = \sum_i Pr[E \mid F_i].Pr[F_i]$ — *Formule des* **probabilités totales**
- $\mathbb{E}[X] = \sum_i \mathbb{E}[X \mid F_i].Pr[F_i]$ — *Formule de* **l'espérance totale**

1.3 Bits et entiers aléatoires ou pseudo-aléatoires

- Les générateurs **pseudo-aléatoires** implantés dans les machines sont considérés d'un point de vue théorique et pratique comme des générateurs acceptables de bits, d'entiers et de réels aléatoires.

1.4 Lois de probabilités usuelles

- La loi **Uniforme** sur l'ensemble $V = \{1, \ldots, n\}$ est donné par $\Pr[X = v] = 1/|V| = 1/n$ pour tout $v \in V$. Elle se simule par le tirage d'un entier entre 1 et $|V|$.
- La loi **Bernoulli**(p) sur l'ensemble $V = \{0, 1\}$ est donnée par $\Pr[X = 1] = p$ et $\Pr[X = 0] = 1 - p$. Elle se simule par letirage d'un bit aléatoire biaisé.

- La loi **Binomiale**(p,n) sur l'ensemble $V = \{0,\dots,n\}$ est donnée par $\Pr[X = k] = \binom{n}{k}p^k(1-p)^k$ pour $k = 0,\dots,n$. Elle se simule comme la somme de n variables de Bernoulli de paramètre p.
- La loi **Géométrique**(p) sur l'ensemble $V = \mathbb{N}$ est donnée par $\Pr[X = n] = p(1-p)^{n-1}$ pour $n \in \mathbb{N}$. Elle se simule comme la loi de la première apparition du '1' dans une suite de variables de Bernoulli de paramètre p.

1.5 Borne des queues de distribution

Lemme 6 (Inégalité de Markov) *Soit* $X : \Omega \to V$ *une variable aléatoire* **positive** *(*$V \subseteq \mathbb{R}^+$*).*
Pour tout $t > 0$*, on a :* $Pr[X \geq t] \leq \frac{1}{t}.\mathbb{E}[X]$

2 Recherche exhaustive et backtrack

2.1 Recherche exhaustive

- Un **littéral** est une variable logique ou sa négation. Une **clause** est une disjonction de de littéraux. Et une **formule SAT** est une conjonction de clauses.
  Ex : $\varphi(x_1, x_2, x_3) = (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge \neg x_2$
- Le problème SAT prend en entrée une formule SAT φ et décide si il existe ou non une affection des variables à `vrai` ou `faux` qui satifasse φ.
  Chaque clause C de φ est codé par un tableau où chaque case contient l'indice de la variable correspondante, précédée d'un signe moins si la variable est négative dans C. Une affectation A est codée par un tableau tel que $A[i] = 1$ (resp. -1) si x_i est affecté à `vrai` (resp. `faux`).
- Les algorithmes TestAff et AffSuivante respectivement testent une affectation et passe à l'affectation suivante.

```
Algorithme : TestAff(φ, A)
pour C dans φ faire
    OK ← faux;
    pour ℓ dans C faire
        si ℓ × A[|ℓ|−1] > 0 alors
            OK ← vrai;
    si OK = faux alors retourner faux;
retourner vrai;
```

```
Algorithme : AffSuivante(A)
i ← 0;
tant que i < n et A[i] = 1 faire
    A[i] ← −1;
    i ← i + 1;
    si i = n alors
        retourner « Fin »
A[i] ← 1;
retourner A;
```

- L'algorithme RechercheExhaustiveSAT cherche une affectation positive pour une formule SAT.

```
Algorithme : RechercheExhaustiveSAT(φ)
A ← tableau de longueur n (nb de variables dans φ), initialisé à −1;
tant que non(TestAff(φ, A)) faire
    A ← AffSuivante(A);
    si AffSuivante a renvoyé « Fin » alors
        retourner « Insatisfiable »
Renvoyer A;
```

Théorème 3 (Resolution SAT exhaustive) *L'algorithme* RechercheExhaustiveSAT *trouve une affectation satisfaisante s'il en existe une, et renvoie « Insatisfiable » sinon, en temps* $O(|\varphi|2^n)$.

- Le principe de la recherche exhaustive est de parcourir toutes les solutions possibles et pour chacune de tester si elle répond au problème.
  La complexité des algos correspondants est $O\left(\textsc{NombreSolutions} \times (\textsc{CoûtTest} + \textsc{CoûtPassageSuivant})\right)$
- Le problème du **Voyageur de commerce** a pour entrée un graphe $G = (S, A)$ avec une longueur $\ell(u, v)$ pour chaque arête, et pour sortie une numérotation $u_0, \ldots, u_{n-1}$ des sommets qui minimise la longueur totale $\sum_{i=0}^{n-1} \ell(u_i, u_{i+1}) + \ell(u_{n-1}, u_0)$.
  La résolution du Voyageur de commerce par recherche exhaustive demande de générer toutes les permutations des entiers 0 à $n - 1$. Pour cela, on part de la permutation $\pi = 0, 1, 2, \ldots, n - 1$ et on applique l'algorithme suivant.

> **Algorithme :** $\textsc{PermSuivante}(\pi)$
> **si** $\pi_{[0]} > \cdots > \pi_{[n-1]}$ **alors retourner** *'Fin'*;
> Trouver j maximal tel que $\pi_{[j]} < \pi_{[j+1]}$;
> Trouver $\ell > j$ maximal tel que $\pi_{[j]} < \pi_{[\ell]}$;
> Échanger $\pi_{[j]}$ et $\pi_{[\ell]}$;
> **pour** $0 < k < \frac{n-j}{2}$ **faire**
>   Échanger $\pi_{[j+k]}$ et $\pi_{[n-k]}$; *Retournement de* $\pi_{[j+1,n[}$
> **retourner** π;

- L'algorithme suivant résout alors le problème du voyageur de commerce sur le graphe $G = (S, A)$ muni d'une fonction de longueur ℓ sur ses arêtes.

> **Algorithme :** $\textsc{VoyageurDeCommerce}(S, A, \ell)$
> $\pi \leftarrow$ tableau de taille n, initialisé à $[0, 1, \ldots, n - 1]$;
> $L_{\min} \leftarrow +\infty$;
> $\pi_{\min} \leftarrow \pi$;
> **tant que** $\textsc{PermSuivante}$ *ne retourne pas 'Fin'* **faire**
> $L \leftarrow \sum_{i=0}^{n-1} \ell(S[\pi_{[i]}], S[\pi_{[i+1 \bmod n]}])$;
>   **si** $L < L_{\min}$ **alors**
> $(L_{\min}, \pi_{\min}) \leftarrow (L, \pi)$;
> $\pi \leftarrow \textsc{PermSuivante}(\pi)$;
> **retourner** $\pi_{\min}$;

> **Théorème 4 (Résolution exhaustive de** $\textsc{VoyageurDeCommerce}$**)** *L'algorithme* $\textsc{VoyageurDeCommerce}$ *résout le problème du voyageur de commerce en temps* $O(n \times n!)$.

2.2 Backtrack ou 'retour sur trace'

- La résolution de SAT par backtrack se fait en affectant récursivement chaque variable en testant à chaque étape la solution partielle obtenue. On utilise les algorithmes suivants, où φ est une formule SAT à n variables et b est une valeur booléenne (*Vrai* ou *Faux*).

> **Théorème 5 (Resolution** SAT **par backtrack)** *L'algorithme* $\textsc{SatBacktrack}$ *trouve une affectation satisfaisante s'il en existe une, et renvoie ≪ Insatisfiable ≫ sinon, en temps* $O(|\varphi| 2^n)$ *(mais est beaucoup plus efficace en pratique que l'algorithme* $\textsc{RechercheExhaustiveSAT}$*).*

```
Algorithme : Élimination(φ, n, b)
ψ ← formule vide;
pour C dans φ faire
    C′ ← clause vide;
    sat ← FAUX;
    pour ℓ dans C faire
        si |ℓ| = n et ℓ × b > 0 alors
            sat ← VRAI
        sinon
            si |ℓ| ≠ n alors
                Ajouter ℓ à C′;
    si NON(sat) alors
        Ajouter C′ à ψ;
retourner ψ;
```

```
Algorithme : SatBacktrack(φ, n)
si φ est vide alors
    retourner A = [1, . . . , 1];
si φ possède une clause vide alors
    retourner "Insatisfiable";
pour b ∈ {1, −1} faire
    ψ ← Élimination(φ, n, b);
    A ← SatBacktrack(ψ, n − 1);
    si ψ n'est pas insatisfiable alors
        retourner A + [b];
retourner "Insatisfiable";
```

- Le principe des algorithmes de Backtrack est de constuire récursivement des solutions partielles et de tester celles-ci à chaque étape (ce qui permet d'éliminer des branches de l'exploration). Cela correspond à parcourir l'arbre des solutions en profondeur en évitant de parcourir certaines branches.
- Le problème de **Sudoku généralisé** est le suivant.
  ENTRÉE : Une grille G de dimensions $n^2 \times n^2$, remplie d'entiers de 0 (= vide) à n^2
  SORTIE : La même grille G sans 0, tel que $G_{[i,j]} \neq G_{[k,\ell]}$ dès que : $i = k$ (ligne), ou $j = \ell$ (colonne), ou $\lfloor i/n \rfloor = \lfloor k/n \rfloor$ et $\lfloor j/n \rfloor = \lfloor \ell/n \rfloor$ (zone), ou alors *'aucune solution'*.
- La grille du Sudoku s'encode de façon linéaire. La grille G de dimensions $n^2 \times n^2$ est représentée par un tableau T de taille n^4. La case $G_{[i,j]}$ correspond à la case $T_{[in^2+j]}$ (stockage *ligne par ligne*). Inversement, la case $T_{[u]}$ correspond à la case $G_{[u \operatorname{div} n^2, u \bmod n^2]}$ où $u \operatorname{div} n^2 = \lfloor u/n^2 \rfloor$.
- La résolution par backtrack du Sodoku généralisé se fait par les deux algorithmes suivants où pour le premier u est un numéro de case et x une valeur à tester dans la case u.

```
Algorithme : Valide(G, n, u, x)
(i, j) ← (u div n², u mod n²);
pour k = 0 à n² − 1 faire
    si (k ≠ i et G[kn²+j] = x) alors
        retourner Faux;
    si (k ≠ j et G[in²+k] = x) alors
        retourner Faux;
(z_i, z_j) ← (n⌊i/n⌋, n⌊j/n⌋);
pour k = 0 à n − 1 faire
    pour ℓ = 0 à n − 1 faire
        si ((z_i + k)n² + (z_j + ℓ) ≠ u et
          G[(z_i+k)n²+(z_j+ℓ)] = x) alors
            retourner Faux;
retourner Vrai;
```

```
Algorithme : SudokuBacktrack(G, n)
u ← 0;
tant que (u < n⁴ et G[u] ≠ 0) faire
    u ← u + 1;
si u = n⁴ alors
    retourner Vrai;
pour x de 1 à n² faire
    si Valide(G, n, u, x) alors
        G[u] ← x;
        si SudokuBacktrack(G, n, u + 1)
          alors
            retourner Vrai;
G[u] ← 0;
retourner Faux;
```

3 Analyse amortie, analyse d'algorithmes probabilistes

3.1 Analyse amortie

- On revient sur le compteur binaire, qui prend un tableau T de taille k en entier représentant l'entier N encodé en binaire et retourne le tableau correspondant à l'entier $N+1$.

```
Algorithme : Incrément(T)
i ← 0;
tant que i < k et T[i] = 1 faire
    T[i] ← 0;
    i ← i + 1;
    si i = k alors retourner 'Fin';
si i < k alors
    T[i] ← 1;
retourner T;
```

Théorème 6 (Compteur binaire armorti) Incrément *est correct, a une complexité en $O(k)$ et une complexité amortie en $O(1)$ lorsqu'on l'appelle 2^k fois depuis l'appel initial sur $T = [0, 0, \ldots, 0]$.*

- Si on appelle un algorithme sur N instances résultant en un nombre c_i d'opérations élémentaires sur la $i$ème instance, le **coût amorti** (ou **complexité amortie**) sur la séquence d'instances est $\frac{1}{N}\sum_{i=1}^{N} c_i$. Classiquement, il y a trois méthode pour caluler ou borner cela.
- La **méthode de l'agrégat** consiste à calculer explicitement $\sum_{i=1}^{N} c_i$ et à retourner $\frac{1}{N}\sum_{i=1}^{N} c_i$.
- La **méthode comptable** consiste à considérer un 'compte' qui ne doit jamais être en négatif et surlequel, à l'étape i, on verse la somme a_i et on prélève la somme c_i. Plus précisément, on détermine des entiers a_i positifs ou négatifs, vérifiant pour tout $i = 0, \ldots, N-1$ l'inégalité $\sum_{0\leq j\leq i} c_j \leq \sum_{0\leq j\leq i} a_j$. Le coût amorti est alors borné par $\frac{1}{N}\sum_{0\leq i\leq N-1} a_i$.
- La **méthode du potentiel** consiste à associer à l'instance produite après le $i$ème appel un potentiel Φ_i et à estimer le coût $a_i = c_i + (\Phi_i - \Phi_{i-1})$ après chaque appel. Le coût amorti est alors borné par $\frac{1}{N}\sum_{0\leq i\leq N-1} a_i$
- Une **liste dynamique** est donné par un tableau T de taille N et ayant une taille utile n. Les algorithmes d'ajout et de suppression d'un élément dans une telle liste sont données ci-dessous.

```
Algorithme : Ajout(T, N, n, x)
si n < N alors
    T[n] ← x;
    retourner (T,N,n+1)
U ← tableau de taille 2N;
pour i = 0 à N − 1 faire
    U[i] ← T[i];
U[N] ← x;
retourner (U, 2N, n + 1)
```

```
Algorithme : Suppression(T, N, n)
si n = 1 ou n > N/4 alors
    retourner (T, N, n − 1)
U ← tableau de taille N/2;
pour i = 0 à n − 2 faire
    U[i] ← T[i]
retourner (U, N/2, n − 1)
```

Théorème 7 (Tableaux dynamiques) Ajout *et* Suppression *sont corrects, de complexité $O(n)$ dans le pire des cas, mais avec un coût amorti en $O(1)$ pour n'importe quelle séquence d'opérations en partant d'une liste vide.*

3.2 Analyse d'algorithmes probabilistes

- Le problème de **sélection** (ou du $k$ième rang) consiste à trouver le $k$ième plus petit élément d'un tableau T donné, de taille n. L'algorithme suivant permet de répondre au problème.

> **Algorithme :** QuickSelect(T, k)
>
> $p \leftarrow T[i]$ avec i choisi aléatoirement entre 0 et $n-1$;
> $n_0 \leftarrow$ nombre d'élément $< p$ dans T *(boucle Pour)*;
> **si** $n_0 = k-1$ **alors retourner** p;
> **si** $n_0 \geq k$ **alors**
> $T_0 \leftarrow$ tableau des éléments de T qui sont $< p$ *(boucle Pour)*;
>   **retourner** QuickSelect(T_0, k)
> $T_1 \leftarrow$ tableau des éléments de T qui sont $> p$ *(boucle Pour)*;
> **retourner** QuickSelect$(T_1, k - n_0 - 1)$;

> **Théorème 8 (Espérance de complexité pour** QuickSelect**)** *Soit C_n le nombre de comparaisons effectuées par* QuickSelect(T, k) *où T est de taille n. Alors $\mathbb{E}[C_n] \leq 4n$, quelque soit k.*

- Un **multigraphe** est un graphe dans lequel chaque paire de sommets est relié par aucune, une *ou plusieurs* arêtes.
- Une **coupe** dans un multigraphe $G = (V, A)$ est une partition (V_1, V_2) de l'ensemble de ses sommets en deux ensembles non vides.
  La **taille** de la coupe (V_1, V_2) est le nombre d'arêtes entre V_1 et V_2, c'est-à-dire $|\{u_1u_2 \in A : u_1 \in V_1, u_2 \in V_2\}|$
  Le **problème de la coupe minimale** prend en entrée un multigraphe G et retourne une coupe de taille minimale.
- Soit $G = (V, A)$ un multigraphe et uv une arête de G. Le multigraphe G/uv, obtenu par contraction de l'arête uv, a pour sommets $V \setminus v$ et pour ensemble d'arêtes $(A \setminus \{xv \ : \ xv \in A\}) \cup \{xu \ : \ xv \in A, x \neq u\}$
- l'algorithme suivant prend en entrée un multigraphe G et retourne une coupe de G.

> **Algorithme :** CoupeMin(G)
>
> **tant que** *G possède au moins 3 sommets* **faire**
>   Choisir une arête uv de G, aléatoirement et uniformément;
>   Contracter l'arête uv dans G;
> **retourner** la coupe définie par les deux sommets restants

> **Théorème 9 (Analyse de** CoupeMin**)** *En admettant que l'opération de contraction d'arête puisse s'effectuer en temps $O(n)$, où n est le nombre de sommets de G, alors* CoupeMin *retourne une coupe de G en temps $O(n^2)$.*
> *De plus, cette coupe est une coupe minimale de G avec probabilité $\geq \frac{2}{n(n-1)}$.*

> **Lemme 7 (Lemme de répétition)** *Si on répète N fois* CoupeMin *et qu'on garde la plus petite coupe renvoyée, cette coupe est minimale avec probabilité $\geq 1 - e^{-2N/n(n-1)}$*

> **Théorème 10 (**CoupeMin **répété)** *Si on répète $N = 2n^2$ fois l'algorithme* CoupeMin *et que l'on retourne la coupe trouvée de plus petite taille, alors, en temps $O(n^4)$ une coupe minimale est retournée avec probabilité $\geq 98\%$.*

- Les algorithmes probabilistes se divisent classiquement en deux grandes familles :
  - Les algorithmes de type **Las Vegas** dont le résultat ne dépend pas des choix aléatoires, mais

la complexité si (ex : QuickSelect).
- Les algorithmes de type **Monte Carlo** dont le résultat dépend des choix aléatoires, mais pas la complexité (ex : CoupeMin).

- L'algorithme suivant prend en entrée un tableau T et retourne ce tableau, trié.

Algorithme : TriRapide(T)
si *taille*$(T) = 1$ **alors retourner** T;
$p \leftarrow T_{[i]}$ avec i choisi aléatoirement entre 0 et $n-1$; *# p est appelé 'le pivot'*
$n_p \leftarrow$ nombre d'indices i tels que $T_{[i]} = p$; *# Boucle 'Pour' pour cela*
$T_0 \leftarrow$ tableau des éléments de T qui sont $< p$; *# Boucle 'Pour' pour cela*
$T_1 \leftarrow$ tableau des éléments de T qui sont $> p$; *# Boucle 'Pour' pour cela*
$T_0 \leftarrow$ TriRapide(T_0);
$T_1 \leftarrow$ TriRapide(T_1);
retourner la concaténation T_0, n_p fois p, et T_1

Théorème 11 (TriRapide) TriRapide *retourne bien le tableau initial, trié. L'espérance du nombre de comparaisons effectuées par* TriRapide *est* $O(n \log n)$

4 Tables de hachage

4.1 Introduction