Sujet 1

Exercice 1

On considère le problème suivant `Couverture d'ensemble` :

* **Instance** : trois entiers $m, n, k \in \mathbb{N}^*$ et une collection d'ensembles $C = \{E_1, E_2, \ldots, E_n\}$ où les $E_i \subseteq [\![1, m]\!]$.
* **Question** : existe-t-il une partie de C de cardinal $\leqslant k$ qui forme une couverture de $[\![1, m]\!]$, c'est-à-dire dont l'union est égale à $[\![1, m]\!]$?

Montrer que `Couverture d'ensemble` est est NP-complet. *On pourra admettre la NP-complétude d'un problème vu en cours ou en exercices.*

Corrigé

Un certificat est une partie I de $[\![1, n]\!]$. On vérifie en temps polynomial que $|I| \leqslant k$ et $\bigcup_{i \in I} E_i = [\![1, m]\!]$. Le problème est donc bien dans NP.

On fait une réduction depuis `Couverture par sommets`. Soit $G = (S, A)$ un graphe non orienté. On numérote les sommets $S = \{1, \ldots, n\}$ arêtes $A = \{a_1, \ldots, a_m\}$. On pose, pour $s \in S$, $E_s = \{i \in [\![1, m]\!] \mid s \in a_i\}$, c'est-à-dire l'ensemble des arêtes incidents à un sommet. Alors G a une couverture par sommets de taille k si et seulement si $C = \{E_s \mid s \in S\}$ a une couverture d'ensemble de cardinal k.

Exercice 2

On admet l'existence d'un algorithme qui détermine la médiane d'un tableau de taille n en temps $\mathcal{O}(n)$. On cherche à écrire un algorithme probabiliste qui renvoie une valeur comprise entre le premier quartile et le troisième quartile d'un tableau de taille n avec une grande probabilité. On considère l'algorithme suivant :

> **Entrée :** tableau $t = [t_0, \ldots, t_{n-1}]$.
> **Début algorithme**
> Sélectionner au hasard $16 \ln n$ valeurs dans t.
> Renvoyer la médiane exacte de ces valeurs.

1. Décrire un algorithme de complexité logarithmique en n permettant de sélectionner au hasard $10 \log_2 n$ valeurs dans t. On autorisera la modification du tableau t si besoin.
2. En déduire la complexité temporelle de l'algorithme précédent.

On admet l'inégalité de Chernoff : si $X_1, \ldots, X_n$ sont des variables aléatoires indépendantes de Bernoulli de paramètre p et $\varepsilon > 0$, alors :

$$\mathbb{P}\left(\frac{1}{n}\sum_{i=1}^{n} X_i > p + \varepsilon\right) \leqslant \mathrm{e}^{-2\varepsilon^2 n}$$

3. On note $k = 16 \ln n$. Montrer que la probabilité qu'au moins $\frac{k}{2}$ éléments parmi les k choisis au hasard dans t soient inférieurs au premier quartile est inférieure ou égale à $\mathrm{e}^{-\frac{k}{8}}$.
4. En déduire que la probabilité que l'algorithme précédent renvoie une valeur comprise entre le premier quartile et le troisième quartile est supérieure ou égale à $1 - \frac{2}{n^2}$.

Corrigé

1. Pour $i = 1$ à k, on choisit un indice j entre 0 et $n - i$, on garde l'élément $t[j]$, puis on permute $t[j]$ et $t[n - i]$. Chaque passage dans la boucle est en temps constant.
2. On a une complexité en $\mathcal{O}(\log n)$ d'après l'hypothèse.
3. On pose $X_i = \begin{cases} 0 & \text{si le } i\text{-ème élément choisi est } \geqslant Q_1 \\ 1 & \text{sinon} \end{cases}$. Alors X_i est une VA de Bernoulli de paramètre $\frac{1}{4}$. En prenant $\varepsilon = \frac{1}{4}$ et en utilisant l'inégalité de Chernoff, on obtient l'inégalité voulue.
4. La probabilité précédente est $\leqslant \frac{1}{n^2}$ d'après la valeur de k.

Sujet 2

Exercice 1

Un flux de n valeurs défilent successivement avec les défauts suivants :

- n est trop grand pour qu'on puisse garder en mémoire toutes les valeurs déjà vues ;
- n n'est pas connu à l'avance.

Écrire un algorithme qui choisit k éléments parmi les n, de manière équiprobable. La seule génération d'aléatoire dont on dispose est la sélection d'un entier compris entre 0 et $i-1$, pour tout entier i.

Corrigé

On propose l'algorithme suivant (où `foo` est la fonction de génération d'aléatoire).

Entrée : entier k, flux de valeurs F
Début algorithme
  $t \leftarrow$ tableau de taille k contenant des valeurs arbitraires.
  $i \leftarrow 1$.
  Tant que F n'est pas terminé **Faire**
    $x \leftarrow$ valeur suivante de F.
    $j \leftarrow$ `foo`(i).
    Si $i \leqslant k$ **Alors**
      $t[i] \leftarrow x$.
    Sinon si $j \leqslant k$ **Alors**
      $t[j] \leftarrow x$.
    $i \leftarrow i+1$.
  Renvoyer t.

Il s'agit maintenant de montrer que cet algorithme renvoie bien une sélection uniforme de k valeurs parmi les n du flux F :

1. Commençons par les éléments après les k premiers :
   - Pour $i > k$, la probabilité que le i-ème élément du flux soit ajouté au tableau (disons à un indice j) est $\frac{k}{i}$.
   - Pour que cet élément ne soit jamais remplacé, il faut que l'indice j ne soit jamais tiré lors des appels à `baz` qui vont suivre. Cette probabilité vaudra successivement $1-\frac{1}{i+1}$, $1-\frac{1}{i+2}$, …, $1-\frac{1}{n}$. Ainsi la probabilité que l'indice j ne soit jamais tiré est le produit de ces probabilités, soit $\prod_{m=i+1}^{n}\left(1-\frac{1}{m}\right) = \prod_{m=i+1}^{n}\frac{m-1}{m} = \frac{i}{n}$.
   - Finalement, la probabilité que le i-ème élément du flux soit dans le tableau à la fin de l'algorithme est le produit des deux probabilités précédentes, soit $\frac{k}{i}\times\frac{i}{n} = \frac{k}{n}$.
2. Pour les k premiers éléments du flux, on se contente de calculer la probabilité qu'ils ne soient jamais remplacés, soit $\prod_{m=k+1}^{n}\left(1-\frac{1}{m}\right) = \frac{k}{n}$.

Ainsi, tous les éléments du flux ont bien la même probabilité de faire partie des k restants.

Exercice 2

On considère le problème `1LIT-3SAT` :

* **Instance** : une formule φ en 3-FNC.
* **Question** : existe-t-il une valuation μ telle qu'exactement un littéral par clause de φ est évalué à « vrai » ?

Pour une clause $C = x \vee y \vee z$, on définit $\psi_C = (\overline{x} \vee a \vee b) \wedge (\overline{y} \vee c \vee d) \wedge (\overline{z} \vee e \vee f) \wedge (a \vee c \vee e)$.

- Montrer que pour une valuation μ, $\mu(C) = 1$ si et seulement s'il existe une valuation μ_C telle que μ et μ_C coïncident en x, y et z, et μ_C rend vrai exactement un littéral par clause de ψ_C.
- En déduire que `1LIT-3SAT` est NP-complet.

Corrigé

1. On distingue selon le nombre de littéraux vrais dans C, sans perte de généralité pour le sens direct. Réciproquement, ça marche aussi car une seule variable parmi a, c et e est vraie.
2. On fait une réduction depuis `3SAT`. Pour φ une formule en 3-FNC, on remplace chaque clause C par ψ_C, avec 6 nouvelles variables à chaque fois. On utilise la question précédente pour conclure la réduction. L'appartenance à NP est claire.

Sujet 3

Exercice 1

On considère le problème `ARBRE COUVRANT` :

* **Instance** : un graphe $G = (S, A, f)$ non orienté pondéré et un seuil $P \in \mathbb{R}$.
* **Question** : existe-t-il un arbre couvrant T de G tel que $f(T) \leqslant P$?

1. Justifier que `ARBRE COUVRANT` $\in$ P.

On considère le problème `ARBRE COUVRANT BORNÉ` :

* **Instance** : un graphe $G = (S, A, f)$ non orienté pondéré, un seuil $P \in \mathbb{R}$ et un entier k.
* **Question** : existe-t-il un arbre couvrant T de G tel que $f(T) \leqslant P$ et tous les sommets de T sont de degré au plus k ?

2. Montrer que `ARBRE COUVRANT BORNÉ` est NP-complet. *On pourra admettre la NP-complétude d'un problème vu en cours ou en exercices.*

Corrigé

1. L'algorithme de Kruskal répond au problème en temps polynomial.
2. On fait une réduction depuis `CHEMIN HAMILTONIEN` : on pose tous les poids égaux à 1, $P = |S| - 1$ et $k = 2$.

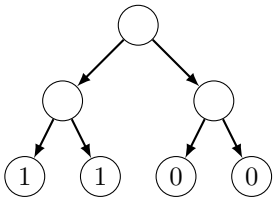
Exercice 2

On considère un arbre correspondant à un graphe de jeu à deux joueurs où les coups alternent entre les deux joueurs. On suppose que les états terminaux sont gagnants pour l'un ou l'autre des joueurs (pas de match nul). On cherche à déterminer si la racine de l'arbre est gagnante pour 1 ou pour 2. On se limite au cas d'un arbre binaire où il reste k coups à jouer pour chacun des deux joueurs, en commençant par le joueur 1.

1. Quelle est la complexité temporelle en fonction de k d'un algorithme déterministe pour trouver qui est le joueur gagnant ?
2. Montrer qu'il n'est pas nécessaire de déterminer l'état de chaque nœud de l'arbre pour trouver qui est le joueur gagnant.
3. En déduire un algorithme Las Vegas simple permettant de déterminer le joueur gagnant.
4. Montrer que l'espérance du nombre de nœuds dont l'état est déterminé par l'algorithme précédent est inférieur ou égal à 3^k.
5. En déduire que la complexité temporelle de cet algorithme est en $\mathcal{O}(n^{0.8})$, où n est le nombre de nœuds.

Corrigé

1. Un algorithme déterministe consisterait à évaluer chaque nœud de l'arbre. Comme c'est un arbre complet de hauteur $2k$, il y a environ 4^k nœuds, soit une complexité en $\mathcal{O}(4^k)$.
2. Par exemple dans l'arbre suivant :



On peut n'évaluer que 4 nœuds sur les 7 (les 4 premiers dans un ordre préfixe).

3. L'idée est la suivante :
   - pour chaque nœud :
     * on évalue récursivement un de ses deux fils au hasard ;
     * si c'est au joueur 1 de jouer et que le premier fils est gagnant pour 1, alors le nœud aussi ;
     * si c'est au joueur 2 de jouer et que le premier fils est gagnant pour 2, alors le nœud aussi ;
     * sinon, on évalue récursivement le deuxième fils avant de conclure.
4. On montre ce résultat par récurrence sur k :
   - pour $k = 0$, on évalue effectivement un seul nœud ;
   - supposons le résultat établi pour un certain k. Alors, pour un nœud donné, les deux fils seront évalués avec probabilité $\frac{1}{2}$, et un seul avec probabilité $\frac{1}{2}$. Soit x un nœud du joueur 1 et y, z ses deux fils.
     * pour l'évaluation de y (resp. z), le nombre moyen de nœuds étudiés vaudra $\frac{1}{2}3^{k-1} + \frac{1}{2} \times 2 \times 3^{k-1} = \frac{1}{2}3^k$;
     * pour l'évaluation de x, le nombre moyen de nœuds étudiés vaudra $\frac{1}{2}\frac{1}{2}3^k + \frac{1}{2} \times 2 \times \frac{1}{2}3^k = 3^k\left(\frac{1}{4} + \frac{1}{2}\right) \leqslant 3^k$.

   On conclut par récurrence.
5. Sachant que $n \simeq 4^k$, on a $k \simeq \log_4 n$, et $3^k = n^{\log_4 3} \simeq n^{0.8}$.

Questions supplémentaires

1. On considère `Ensemble couvrant` :
   * **Instance** : un ensemble $E = \{x_1, \ldots, x_N\}$ et une collections d'ensembles $S_1, \ldots, S_n \in \mathcal{P}(E)$.
   * **Solution** : un sous-ensemble $S \subseteq E$ tel que $S \cap S_j \neq \emptyset$ pour tout $j \in [\![1, n]\!]$.
   * **Optimisation** : minimiser $|S|$.

   Proposer un algorithme de type Branch and Bound qui résout ce problème. On détaillera la forme des solutions partielles, ainsi que les heuristiques d'évaluation et de branchement.
2. On considère `CLUSTERING` :
   * **Instance** : un ensemble de données $E = \{x_1, \ldots, x_N\} \subseteq \mathbb{R}^d$ et un entier m.
   * **Solution** : une partition de E en m classes $C_1, \ldots, C_m$.
   * **Optimisation** : minimiser $\max_{j=1}^{m} \max_{x,y \in C_j} \|x - y\|_2$.

   On admet que ce problème est NP-difficile, même pour $d = 2$. On considère l'algorithme suivant :

   Entrée : ensemble de données $E = \{x_1, \ldots, x_N\}$ et un entier m.
   Début algorithme
   Choisir y_1 arbitrairement parmi les x_i.
   Pour $j = 2$ à m **Faire**
   $y_j \leftarrow$ point le plus loin de $\{y_1, \ldots, y_{j-1}\}$ (c'est-à-dire le x_i qui maximise $\min_{j'=1}^{j-1} \|x_i - y_{j'}\|$).
   Associer chaque x_i au y_j le plus proche et renvoyer les classes ainsi créées.

   Montrer que cet algorithme est une 2-approximation de `CLUSTERING`.
3. On considère `Multicoupe minimale` :
   * **Instance** : un graphe $G = (S, A)$ non orienté et un ensemble $\{s_1, \ldots, s_k\} \subseteq S$ de sommets terminaux.
   * **Solution** : un ensemble d'arêtes $A' \subseteq A$ tel que dans $G - A'$, tous les s_i sont dans des composantes connexes différentes.

* **Optimisation** : minimiser $|A'|$.

(a) Justifier que si $k = 2$, le problème peut être résolu en temps polynomial.

(b) Décrire une 2-approximation du problème lorsque $k = 3$.

Corrigé

1. Pour $x \in E$, on appelle degré de x le nombre de S_j tels que $x \in S_j$. On commence par trier les x_i par degré décroissant. Une solution partielle est un ensemble $\tilde{S} \subseteq \{x_1, \ldots, x_{i-1}\}$ (qui indique si on garde ou non chaque élément). Une heuristique de branchement consiste à commencer par garder x_i puis à ne pas le garder (d'où l'intérêt d'avoir trié par degré décroissant). Comme heuristique d'évaluation, en notant n' le nombre d'ensemble S_j non atteints par $\tilde{S}$, il faudra choisir au moins k sommets tels que $\sum_{i'=i}^{i+k} \deg(x'_i) \geqslant n'$ (un sommet de degré d peut atteindre au plus d ensembles).
2. Soit y_{m+1} le point le plus loin de $\{y_1, \ldots, y_m\}$ et r la distance entre y_{m+1} et le y_j le plus proche. Alors chaque cluster est de diamètre au plus $2r$. De plus, $\{y_1, \ldots, y_{m+1}\}$ représentent $m+1$ points dont la distance minimale entre 2 est r. Par le principe des tiroirs, deux de ces points doivent être dans le même cluster dans un clustering optimal, donc le diamètre maximal vaut au moins r.
3. (a) C'est le problème de la coupe minimale, équivalent au flot maximal (avec des poids d'arêtes égaux à 1), on a montré en DM que ce problème pouvait être résolu en temps polynomial.

   (b) On fait une coupe exacte entre deux des trois sommets, puis une coupe exacte de la partie qui contient le sommet restant. Le nombre d'arêtes coupées est au plus le double d'une coupe minimale (et une tri-coupe minimale contient au moins autant d'arêtes qu'une bi-coupe minimale).