TD14. Stratégies algorithmiques avancées

Exercice 1. Approximation du problème Somme Max
On considère le problème d'optimisation suivant :

$$\texttt{SommeMax}\begin{cases}\text{Entrée :} & \text{n objets } \{a_1,...,a_n\}\in\mathbb{N}^n, \text{ un entier } B\in\mathbb{N}\\ \text{Sortie :} & \max\{\sum_{i\in I}a_i \mid I\subseteq\{1,\dots n\} \text{ et } \sum_{i\in I}a_i\leq B\}\end{cases}$$

1. Donner le problème de décision $\texttt{SommeMax}_D$ associé au problème d'optimisation `SommeMax`.

2. On admet que le problème de décision suivant est NP-difficile :

$$\texttt{Sous-Ensemble}\begin{cases}\text{Entrée :} & \text{n entiers } \{e_1,...,e_n\}\in\mathbb{N}^n, \text{ un entier } W\in\mathbb{N}\\ \text{Sortie :} & \text{Existe-t-il } I\subseteq\{1,\dots,n\} \text{ tq } \sum_{i\in I}e_i=W \text{ ?}\end{cases}$$

Montrer que $\texttt{SommeMax}_D$ est NP-difficile.

3. Proposer un algorithme glouton en temps **linéaire** qui construit une solution pour le problème `SommeMax`.

4. Montrer que $\forall\alpha\in\mathbb{R}$, l'algorithme précédent n'est **pas** une α-approximation.
Indication : Il s'agit d'exhiber une instance pour laquelle le rapport n'est pas respecté.

5. Proposer un algorithme glouton en $O(n\log(n))$ qui évite l'écueil de l'instance précédente.

6. Montrer que l'algorithme proposé à la question précédente est une $\frac{1}{2}$ -approximation pour `SommeMax`.
Indication : On pourra commencer par trouver une borne supérieure de la valeur optimale M, et ensuite raisonner par disjonction de cas selon que l'algorithme a produit une solution de valeur $\geq\frac{M}{2}$ ou $<\frac{M}{2}$.

Exercice 2. Problème d'ordonnancement
On considère le problème suivant :

$$\texttt{Ordo}\begin{cases}\text{Entrée :} & n \text{ tâches de durées } \{t_1,...,t_n\}\in\mathbb{N}^n, \text{ un entier } m\in\mathbb{N}^*\\ \text{Sortie :} & \text{Trouver une affectation des } n \text{ tâches sur } m \text{ machines}\\ & \text{de façon à minimiser la date de fin d'exécution de toutes les tâches}\end{cases}$$

On propose l'algorithme consistant à considérer les tâches dans l'ordre, et d'exécuter la tâche courante sur la machine la moins utilisée jusqu'ici.

1. Quel ordonnancement donne cet algorithme pour les tâches de durées $\{4,1,2,2,4,3,8\}$ sur 3 machines ? Comparer à la solution optimale.

2. Montrer que cet algorithme est une 2-approximation.
Indication : On pourra considérer le moment où la dernière tâche de la machine la plus tardive est attribuée.

Exercice 3. Approximation de π
La proportion de la surface du carré unité $[0,1]\times[0,1]$ qui est recouverte par le disque unité, c'est-à-dire $\{(x,y)\in\mathbb{R}^2 \mid x^2+y^2\leq 1\}$ est de $\frac{\pi}{4}$.

1. Déduire de la remarque précédente une fonction `C double approximation_pi(int k)` qui tire uniformément k points dans le carré unité, et en observant en quelle proportion ceux-ci sont dans le disque, en déduit une valeur approchée de π.

2. À quelle famille d'algorithmes appartient cette fonction ?

Exercice 4. Problème MAX-3SAT
On s'intéresse au problème d'optimisation suivant :

$$\texttt{Max-3SAT}\begin{cases}\text{Entrée :} & m \text{ clauses } C_1,\dots,C_m \text{ de taille 3 portant sur les variables } x_1,\dots,x_n\\ \text{Sortie :} & \text{Le nombre max de clauses pouvant être satisfaites simultanément}\end{cases}$$

1. On affecte à chaque variable, de manière indépendante, une valeur aléatoire de $\{V,F\}$. On note Z_i la variable aléatoire qui vaut 1 si la clause C_i est satisfaite et 0 sinon.
Donner l'espérance de $\mathbb{E}(Z_i)$.

2. On note Z la variable aléatoire indiquant le nombre de clause satisfaite. Donner l'espérance $\mathbb{E}(Z)$.

Nous proposons d'améliorer la stratégie en utilisant une règle simple pour aboutir de manière sure à la valeur $\mathbb{E}(Z)$ (c'est possible, une variable aléatoire ne peut pas être partout inférieure à son espérance) : on répète l'affectation des variables propositionnelles jusqu'à en obtenir une qui satisfait au moins l'espérance précédente.

3. Donner la probabilité $\mathbb{P}(Z>\mathbb{E}(Z))$.

4. En déduire, en espérance, le nombre de répétitions à effectuer.

5. À quelle famille d'algorithme appartient chacune des deux méthodes présentées ?

Exercice 5. Échantillonnage
Cet exercice manipule des tableaux. Si T est un tableau de taille n, on pourra utiliser le raccourci syntaxique $x\in T$ pour désigner : "$\exists i\in\{0,\dots,n-1\}, T[i]=x$".
On s'intéresse au problème suivant :

$$\texttt{Échantillonage}\begin{cases}\text{Entrée :} & \text{Un tableau de } n \text{ éléments et un entier } k\in\{0,\dots,n\}\\ \text{Sortie :} & \text{Un tableau contenant } k \text{ éléments de } T\end{cases}$$

Le tableau résultat R doit s'envoyer injectivement dans le tableau initial. Ainsi, il existe une injection $\varphi:[\![0,k-1]\!]\to[\![0,n-1]\!]$ telle que $\forall i\in[\![0,k-1]\!], R[i]=T[\varphi(i)]$. On remarque que φ n'est pas supposée croissante, l'ordre dans R n'importe donc pas.

On supposes dans la suite, pour simplifier les raisonnements, que le tableau T ne contient aucun doublon. On souhaite finalement s'assurer que les éléments soient choisis de manière équiprobable, ie $\forall p\in[\![0,n-1]\!], \mathbb{P}(T[p]\in R)=\frac{k}{n}$. On se propose ici d'étudier l'algorithme suivant :

```
def echantillonage(int* tab, int n, int k){
    int* res = malloc(k * sizeof(int));
    int i = k;
    while (i < n){
        int j = rand() % (i+1); // tirage uniforme dans [0, i]
        if (j < k){
            res[j] = tab[i];
        }
        i = i + 1;
    }
    return res;
}
```

1. Montrer l'invariant de boucle suivant :

$$\text{Pour tout } 0 \leq l < i,\ \mathbb{P}(\texttt{tab[l]} \text{ sélectionné}) = \frac{k}{i}$$

2. En déduire que l'algorithme proposé convient.

Exercice 6. Mélange de Knuth

Pour mélanger les éléments d'un tableau aléatoirement, le mélange de Knuth procède ainsi : on parcourt le tableau de la gauche vers la droite, et pour chaque élément à l'indice i, on l'échange avec l'élément situé à un indice tiré aléatoirement entre 0 et i inclus.

1. Écrire une fonction OCaml `melange_knuth : 'a array -> unit` qui réalise cet algorithme.

2. Montrer que le mélange de Knuth est un bon mélange, au sens où la probabilité que l'élément initialement dans la case i se retrouve au final dans la case j est exactement $\frac{1}{n}$, où n est la taille du tableau.