TD 5 – Révisions

Exercice 1. *Le juste milieu*

Soit A un ensemble de n éléments distincts totalement ordonné. On cherche à trouver la médiane de l'ensemble, c'est-à-dire l'élément de rang $\lfloor \frac{n+1}{2} \rfloor$.

1. Donner un algorithme naïf. Quelle est sa complexité ?

Pour améliorer la complexité de l'algorithme, on utilise la stratégie suivante : on regroupe les éléments par paquets de 5, on calcule la médiane de chaque paquet, puis la médiane des médianes.

2. Comment terminer cet algorithme ? Quelle est sa complexité ?
3. Que se passe-t-il si on regroupe par paquets de 3 ? Et de 7 ?

Exercice 2. *Limitons la température*

Le but de cet exercice est d'illustrer la théorie des matroïdes. Nous cherchons ici à contruire un matroïde pondéré, à écrire l'algorithme glouton correspondant et à prouver qu'il renvoie la réponse optimale.

Soit $G = (V, E)$ un graphe dirigé où chaque arête $e \in E$ est munie d'une pondération entière $w(e)$, et une fonction de contrainte $f : V \to \mathbb{N}$. Le but est de trouver un sous-ensemble d'arêtes de poids maximal tel que le degré sortant de chaque noeud u est au plus $f(u)$.

1. Définir des ensembles indépendants et prouver qu'ils forment un matroïde.
2. Quel est le cardinal d'un ensemble indépendant maximal ?
3. Quel est l'algorithme glouton associé ? Pourquoi renvoie-t-il la solution optimale ? Quelle est sa complexité ?

Exercice 3. *Les deux extrêmes*

Dans cet exercice, on s'intéresse au calcul (simultané) du maximum et du minimum de n entiers. On mesure la **complexité dans le pire des cas et en nombre de comparaisons** des algorithmes.

1. Donner un algorithme naïf et sa complexité.

Une idée pour améliorer l'algorithme est de regrouper *par paires* les éléments à comparer, de manière à diminuer ensuite le nombre de comparaisons à effectuer.

2. Décrire un algorithme fonctionnant selon ce principe et analyser sa complexité.

Nous allons étudier l'optimalité d'un tel algorithme en fournissant une borne inférieure sur le nombre de comparaisons à effectuer. Nous utiliserons la méthode de l'*adversaire*.

Soit A un algorithme qui trouve le maximum et le minimum. Pour une donnée fixée, au cours du déroulement de l'algorithme, on appelle *novice* (N) un élément qui n'a jamais subi de comparaisons, *gagnant* (G) un élément qui a été comparé au moins une fois et a toujours été supérieur aux éléments auxquels il a été comparé, *perdant* (P) un élément qui a été comparé au moins une fois et a toujours été inférieur aux éléments auxquels il a été comparé, et *moyens* (M) les autres. Le nombre de ces éléments est représenté par un quadruplet d'entiers (i, j, k, l) qui vérifie bien sûr $i + j + k + l = n$.

3. Donner la valeur de ce quadruplet au début et à la fin de l'algorithme. Exhiber une stratégie pour l'adversaire, de sorte à maximiser la durée de l'exécution de l'algorithme. En déduire une borne inférieure sur le nombre de tests à effectuer.

Exercice 4. *La sieste de Maxime*

Étant donné un tableau T de n entiers relatifs, on cherche $\max\{\forall i, j \in \{1 \dots n\} : \sum_{k=i}^{j} T[k]\}$. Par exemple pour le tableau suivant :

2	18	-22	**20**	**8**	$\mathbf{-6}$	**10**	-24	13	3

l'algorithme retournerait la somme des éléments 4 à 7 soit 32.

1. Donner un algorithme retournant la somme maximale d'éléments contigus par une approche *diviser pour régner*.
2. Donner un algorithme retournant la somme maximale d'éléments contigus par *programmation dynamique*.
3. Comparer la complexité Asymptotique au pire cas des deux approches.
4. Peut-on adapter l'algorithme de programmation dynamique en dimension 2 ? Plus formellement, étant donnée une matrice M de $n \times m$ entiers relatifs, on cherche $\max\{\forall i, j \in \{1 \dots n\}, \forall k, l \in \{1 \dots m\} : \sum_{k_1=i}^{j} \sum_{k_2=k}^{l} M[k_1][k_2]\}$.