DM1

A rendre pour le lundi 13 octobre 2014 à 10h15 dans le casier de votre chargé de TD.

Exercice 1. *Plus grand et plus petit de n entiers*

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 2. *Fusion*

On s'intéresse à la fusion de deux ensembles triés, l'un de taille m et l'autre de taille n. Les $m + n$ éléments à fusionner sont tous distincts et notés :

$$A_1 < A_2 < ... < A_m \quad \text{et} \quad B_1 < B_2 < ... < B_n$$

1. Montrer qu'il faut au moins $\lceil \log C^n_{m+n} \rceil$ comparaisons pour effectuer la fusion.
2. En déduire que pour $n = m$, il faut au moins $2n - \frac{1}{2} \log n + O(1)$ comparaisons.
3. Rappeler brièvement l'algorithme usuel de fusion et donner sa complexité.
4. Démontrer que pour $n = m$, on ne peut pas faire mieux que l'algorithme usuel. La borne $2n - \frac{1}{2} \log n + O(1)$ ne peut donc pas être atteinte.

Exercice 3. *Sous-vecteur de somme maximale*

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**	**−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 $\max\{\forall i, j \in \{1 \ldots n\}, \forall k, l \in \{1 \ldots m\} : \sum_{k_1=i}^{j} \sum_{k_2=k}^{l} M[k_1][k_2]\}$.

Exercice 4. *Utilisation de la mémoire*

On souhaite enregistrer sur une mémoire de taille L un groupe de fichiers $P = (P_1, \ldots, P_n)$. Chaque fichier P_i nécessite une place a_i. Supposons que $\sum a_i > L$: on ne peut pas enregistrer tous les fichiers. Il s'agit donc de choisir le sous ensemble Q des fichiers à enregistrer.
On pourrait souhaiter le sous-ensemble qui contient le plus grand nombre de fichiers. Un algorithme glouton pour ce problème pourrait par exemple ranger les fichiers par ordre croissant des a_i.
Supposons que les P_i soient ordonnés par taille ($a_1 \leq \ldots \leq a_n$).

1. Écrivez un algorithme (en pseudo-code) pour la stratégie présentée ci-dessus. Cet algorithme doit renvoyer un tableau booléen S tel que $S[i] = 1$ si P_i est dans Q et $S[i] = 0$ sinon. Quelle est sa complexité en nombre de comparaisons et en nombre d'opérations arithmétiques ?

2. Montrer que cette stratégie donne toujours un sous-ensemble Q maximal tel que $\sum_{P_i \in Q} a_i \leq L$.

3. Soit Q le sous-ensemble obtenu. À quel point le quotient d'utilisation $(\sum_{P_i \in Q} a_i)/L$ peut-il être petit ?

Supposons maintenant que l'on souhaite enregistrer le sous-ensemble Q de P qui maximise ce quotient d'utilisation, c'est-à-dire celui qui remplit le plus de disque. Une approche *gloutonne* consisterait à considérer les fichiers dans l'ordre décroissant des a_i et, s'il reste assez d'espace pour P_i, on l'ajoute à Q.

4. On suppose toujours les P_i ordonnés par taille croissante. Écrivez un algorithme pour cette nouvelle stratégie.

5. Montrer que cette nouvelle stratégie ne donne pas nécessairement un sous-ensemble qui maximise le quotient d'utilisation. À quel point ce quotient peut-il être petit ? Prouvez-le.

Exercice 5. *Matroïdes des degrés sortants*

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 ?