Partiel – Durée 2h

Exercice 1. *Glouton*

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 $\frac{\sum_{P_i \in Q} a_i}{L}$ 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. 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 2. *Programmation dynamique*

On se donne n réels positifs $s_1, \ldots, s_n$ de somme 1. On veut partitionner le carré unité en n rectangles de surfaces $s_1, \ldots, s_n$. On cherche un partitionnement en colonnes comme illustré sur la figure 1. On veut minimiser la longueur des traits dessinés (donc à une constante près la somme des demi-périmètres des n rectangles.

1. Résoudre le problème dans le cas $n = 4$ avec $s_1 = s_2 = s_3 = s_4 = 0,25$.

2. Résoudre le problème dans le cas $n = 6$ avec $s_1 = 0,5$ et $s_2 = s_3 = s_4 = s_5 = s_6 = 0,1$.

3. Dans le cas général, on ne connaît pas le nombre de colonnes qu'on va utiliser. Mais :

(a) Montrer que l'on peut se ramener à placer les s_i dans l'ordre croissant.

(b) Résoudre le problème par programmation dynamique.

FIG. 1 – Partitionnement du carré unité en colonnes de rectangles.

Exercice 3. *Le tri-diot*

On se donne un ensemble de n cartes numérotées de 1 à n. On veut trier le paquet de cartes de la manière (stupide) suivante :
- On parcourt le paquet jusqu'à trouver la première carte i qui n'est pas à la place i ;
- On insère cette carte à sa place, c'est-à-dire que si la carte que l'on range est la carte i on la met en i-ème position, en décalant alors toutes celles qui sont au-dessus d'elle ;
- On recommence jusqu'à ce que le paquet soit trié.

Par exemple, si le paquet a 4 cartes et est initialement dans l'ordre $(2, 1, 4, 3)$, on prend la carte 2 qui n'est pas à sa place et on la met en deuxième position : $(1, 2, 4, 3)$, puis comme les cartes 1 et 2 sont à leur place on prend la carte 4 qui est la première qui n'est pas à sa place et on la met en quatrième position : $(1, 2, 3, 4)$ et l'on s'arrête car le paquet est trié.

1. Donner la suite des étapes successives du tri d'un paquet de 5 cartes initialement dans l'ordre $(3, 2, 5, 4, 1)$.

2. Quelle est la complexité de l'algorithme (en nombre de déplacements de cartes) dans le cas où le paquet est initialement trié dans l'ordre inverse $(n, n-1, \ldots, 2, 1)$?

On veut montrer que cet algorithme termine dans tous les cas, et calculer sa complexité dans le pire des cas. Pour cela, on associe à chaque ordre possible du paquet un poids $p = \sum_{i=1}^{n} 2^{i-1} d_i$ où $d_i = 1$ si la carte i est à sa place dans le paquet et 0 sinon. Par exemple $p(1, 2, 4, 3, 5) = 1 + 2 + 16 = 19$ car 1, 2 et 5 sont à leur place mais pas 3 et 4.

3. Montrer qu'à chaque fois qu'on déplace une carte selon l'algorithme considéré le poids de l'ordre du paquet augmente strictement.

4. En déduire une preuve que l'algorithme termine toujours et donner une majoration de sa complexité dans le pire des cas (en nombre de déplacements de cartes).

On veut maintenant affiner cette borne et montrer qu'elle est atteinte.

5. Justifier que le poids augmente d'au moins 2 à chaque déplacement de carte.

6. En déduire que cet algorithme effectue au plus $(2^{n-1} - 1)$ déplacements de cartes sur un paquet de n cartes, quel que soit l'ordre initial du paquet.

7. Donner une configuration initiale qui atteint cette borne (en le prouvant si possible).