TD 03 – Algorithmes Gloutons

Exercice 1. *Couverture par intervalles*

Étant donné un ensemble $\{x_1, \ldots, x_n\}$ de n points sur une droite, décrire un algorithme qui détermine le plus petit ensemble d'intervalles fermés de longueur 1 qui contient tous les points donnés. Prouver la correction de votre algorithme et donner sa complexité.

Exercice 2. *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 3. *Ordonnancement de tâches indépendantes avec priorité*

On souhaite ordonnancer n tâches indépendantes, T_1, T_2, …, T_n, sur un processeur. Chaque tâche T_i a un temps d'exécution w_i et une priorité p_i. Comme les tâches sont exécutées séquentiellement et que l'on souhaite trouver un ordonnancement optimal, on peut se restreindre aux ordonnancements qui exécutent les tâches dès que possible. Un ordonnancement est alors entièrement défini par l'ordre d'exécution des tâches. En d'autres termes, un ordonnancement des tâches $T_1, \ldots, T_n$ est une permutation $T_{\sigma(1)}$, $T_{\sigma(2)}$, …, $T_{\sigma(n)}$, déterminant dans quel ordre les tâches sont exécutées. On suppose que la première tâche dans l'ordre d'exécution est traitée au temps 0. Le coût d'un ordonnancement est alors défini par $\sum_{i=1}^{n} p_i C_i$, où C_i est le temps de complétion de la tâche T_i, c'est-à-dire, le moment où son traitement est terminé. On cherche à trouver un ordonnancement minimisant ce coût.

1. Soit deux tâches T_i et T_j qui sont exécutées successivement dans un ordonnancement. Quelle tâche devrait être exécutée en premier pour minimiser le coût ?
2. Donner un algorithme glouton résolvant ce problème, et démontrer son optimalité. Quelle est sa complexité ?