TD Algorithmique et Complexité n° 2
(Correction)

Algorithmes Gloutons

Les algorithmes gloutons (en anglais : greedy) construisent une solution de façon incrémentale, en choisissant à chaque étape la direction qui est la plus prometteuse. Ce choix localement optimal n'a aucune raison de conduire à une solution globalement optimale.

Tous les problème n'admettent pas une solution gloutonne.

Exercice 1 Pièces de monnaies

On considère le problème où l'on doit rendre la monnaie pour x euros avec le moins possible de pièces de monnaies.

1. On a des pièces de 1, 2, 5, 10. Ecrire l'algorithme glouton qui donne une solution optimale.

   Correction : Algorithme Glouton : Trier les types de pièces par valeurs décroissantes. Pour chaque valeur de pièce, maximiser le nombre de pièces choisies. Plus formellement, soit R la somme restante, initialisée à S. Pour chaque valeur v_i, prendre $c_i = \lfloor \frac{R}{v_i} \rfloor$ pièces, et poser $R \leftarrow R - c_i * v_i$.
   Pour prouver l'optimalité de l'algorithme glouton avec les valeurs 10, 5, 2 et 1 :
   - Au plus une pièce de 5 (sinon une de 10) Au plus une pièce de 1 (sinon une de 2)
   - Au plus deux pièces de 2 (sinon une de 5 et une de 1)
   - Au plus quatre pièces qui ne sont pas des pièces de 10 (1 5+2 2+1 = 1 10), pour un total inférieur ou égal à 9
   - Le nombre de pièces de 10 est donc $\lfloor \frac{S}{10} \rfloor$
   - Conclure avec ce qui précède

2. On dispose d'un ensemble de pièces $c^0, c^1, c^2, \ldots, c^k$ pour $c > 1$, et $k \geq 1$. écrire l'algorithme glouton qui donne la solution optimale.

   Correction : L'algorithme de la question précèdente marche.

   La preuve, comme pour la plupart des algorithmes gloutons, consiste à montrer que le problème a :
   - **La propriété du choix glouton :** une soulution globalement optimale peut être trouvée en faisant des choix localement optimaux.
   - **La propriété da la sous-structure optimale :** une solution optimale pour le problème contient également la solution optimale de n'importe quel sous-problème. i.e. Si S est une solution optimale qui contient un candidat s_1, alors l'ensemble $S \setminus \{s_1\}$ est une solution optimale du sous-problème qui correspond au problème d'origine sans s_1.

3. Donner un ensemble de pièces tel que l'algorithme glouton ne retourne pas une solution optimale.

   Correction : l'algorithme glouton n'est pas optimal pour le jeu de pièces de valeurs $(6, 4, 1)$: pour $S = 8$, le glouton renvoie $8 = 6 + 1 + 1$ alors qu'on a $8 = 4 + 4$.

4. Donner un ensemble de pièces tel que l'algorithme glouton ne trouve pas de solution.

   Correction : l'algorithme glouton ne trouve de solution pas pour le jeu de pièces de valeurs $(5, 2)$: pour $S = 6$.

5. Donne un algorithme qui retourne une solution optimale pour n'importe quel ensemble de pièces.

   Correction : Dans le cas général, on cherche m(S), le nombre minimum de pièces pour faire S avec des pièces de valeur $(a_1, a_2, \ldots, a_n)$. Pour cela on introduit la fonction z telle que :
   - $z(T, i)$ = nombre minimum de pièces choisies parmi les i premières $(a_1, a_2, \ldots, a_i)$ pour faire T

- on remarque que $z(S,n) = m(S)$, on résout un problème apparemment plus compliqué que le problème de départ
  Mais on a une relation de récurrence :

$$z(T,i) = \min \begin{cases} z(T,i) & si \quad i - emepiecenonutilisee \\ z(T-v-i,i)+1 & si \quad onautiliseunefois(aumoins)lai - emepiece \end{cases}$$

  Il faut initialiser la récurrence correctement, par exemple en posant $z(T,i) = 0$ pour $T_l eq 0$ ou $i = 0$.
  Comment calculer toutes les $S*n$ valeurs de z dont on a besoin ? Un algorithme qui calcule par colonnes (boucle sur i externe, boucle sur T interne) permet de respecter les dépendances, i.e. de toujours avoir en membre droit des valeurs précédemment obtenues. On obtient une complexité en $O(n*S)$ alors que l'algorithme glouton avait une complexité en $o(n \log n)$ (l'exécution est linéaire, mais il faut auparavant trier les valeurs).
  Remarque Caractériser les jeux de pièces pour lesquels l'algorithme glouton est optimal est un problème ouvert. Il est facile de trouver des catégories qui marchent (par exemple des pièces $1, c, c^2, c^3, \ldots$ pour $c \geq 2$) mais le cas général résiste !

Exercice 2 Théorie des matroïdes

- (S, I) est un matroïde si S est un ensemble de n éléments, I est une famille de parties de S ($I \subset \mathcal{P}(S)$) vérifiant
  1. "Hérédité" : $X \in I$ implique $\forall Y \subset X$, $Y \subset I$
  2. "Echange" : $(A, B \in I, |A| < |B|)$ implique $\exists x \in B - A$ tel que $A \cup \{x\} \in I$.
- Les éléments de I sont appelés "indépendants".

1. Montrer que les forêts d'un graphe est un matroïde.

   Correction : Soit $G = (V, E)$ un graphe , $|V| = n$, on définit alors $S = E$ et $I = \{A \subset E / A \; sans \; cycles\}$, c.a.d. qu'un ensemble d'arêtes est un indépendant ssi c'est une forêt.
   L'hérédité est évidente, car un sous-ensemble d'une forêt est une forêt (en enlevant des arêtes à une forêt on ne crée pas de cycles).
   L'échange : Soient A et B des forêts de G tq $|A| < |B|$. Alors A (resp. B) contient $n - |A|$ (resp. $n - |B|$) arbres (avec chaque arête ajoutée à A, on relie deux arbres, donc on décrémente le nombre d'arbres de $|A|$).
   Donc B contient moins d'arbres que A, il existe alors un arbre T de B qui n'est pas inclus dans un arbre de A, c.a.d. qu'il existe deux sommets u et v de T qui n'appartiennent pas au même arbre de A. sur le chemin de u à v dans T, il existe une paire de sommets consécutifs qui ne sont pas dans le même arbre de A (ils sont de chaque côté du *pont* qui traverse d'un arbre à l'autre). Soit (x, y) l'arête en question. d'ou $A \cup \{(x, y)\}$ est sans cycle, i.e. $A \cup \{(x, y)\} \in I$

   Un indépendant est dit "maximal" s'il est maximal au sens de l'inclusion

2. Montrer que tous les indépendants maximaux ont le même cardinal.

   Correction : si ce n'était pas le cas, on pourrait les étendre par la propriété d'échange.

   Matroïde pondéré

   - On ajoute une fonction de poids $w : x \in S \mapsto w(x) \in \mathbb{N}$.
   - On pose

$$w(X) = \sum_{x \in X} w(x).$$

3. Etant donné un matroïde pondéré, donner un algorithme glouton qui construit un indépendant de poids maximal et prouver la validité de votre algorithme.

   Correction :

Algorithm 1 1

1 Trier les éléments de S par poids décroissants

$$w(s_1) \geq w(s_2) \geq \cdots \geq w(s_n).$$

2 $A := \emptyset$.

```
3   pour i := 1 jusqu'à ce que n = |S| faire
4       si A ∪ {s_i} ∈ I alors
5           A := A ∪ {s_i}
6   retourner A
```

Cet Algorithme retourne une solution optimal.

- $\emptyset$ est un independant, par hérédité.
- Soit s_k le premier élément indépendant de S (= le premier indice i de l'algorithme précédent tel que $\{s_i\} \in I$.
- Lemme : Il existe une solution optimale qui contient s_k.
- Soit B un indépendant de poids maximal, et A la solution retournée par l'algorithme. On a $s_k \in A$.
- Supposons $s_k \notin B$ (sinon, rien à prouver) : on applique $|B|-1$ fois la propriété d'échange pour compléter $\{s_k\}$ par des éléments de B : on obtient un indépendant A' qui contient tous les éléments de B sauf un seul que l'on peut appeler b_i.
- On a $w(A') = w(B) - w(b_i) + w(s_k)$, or $w(s_k) \geq w(b_i)$, car $\{b_i\} \in I$ (hérédité) et b_i a été considéré après s_k par ordre décroissant.
- Donc $w(A') \geq w(B)$, donc $w(A') = w(B)$, et A' est une solution optimale qui contient s_k.
- Puis par récurrence, on obtient que glouton donne la solution optimale : on se restreint à une solution contenant $\{s_k\}$, et on recommence avec

$$S' = S - \{s_k\}$$

et

$$I' = \{X \subset S' | X \cup \{s_k\} \in I\}.$$

Exercice 3 Application : ordonnancement

Le problème de l'ordonnancement de tâches sur un seul processeur se compose

- d'un ensemble de taches $S = \{1..n\}$ de durée 1,
- avec des dates limites $d_1, \ldots, d_n$: la tâche i doit finir avant la date d_i, sinon on doit payer la pénalité w_i.

Problème : comment ordonnancer pour minimiser la somme des pénalités.

- Définition : Un ensemble A de tâches est dit "indépendant" si il est possible de les ordonnancer tel que personne ne soit en retard.
- $N_t(A)$: nombre de tâches de l'ensemble A dont la date limite est $\leq t$.

1. Montrer que les 3 propriétés suivantes sont équivalentes
   - A est indépendant
   - Pour $t = 1..n$, on a $N_t(A) \leq t$.
   - Si les tâches de A sont ordonnancées dans l'ordre croissant de leur dates limites, alors aucune tache n'est en retard.

   Correction :

   Preuve
   - $1 \rightarrow 2$: supposons $N_t(A) > t$, alors il y a au moins une tâche qui se déroulera après le temps t, et donc A ne peut pas être indépendant.
   - $2 \rightarrow 3$: trivial.
   - $3 \rightarrow 1$: par définition.

2. Montrer que (S, I), où S est l'ensemble des tâches, I la famille des sous-ensembles de tâches indépendantes est un matroïde.

   Correction : Remarque
   - Minimiser les pénalités des tâches en retard = Maximaliser les pénalités des tâches qui ne sont pas en retard.
   - Théorème : (S, I), où S est l'ensemble des tâches, I la famille des sous-ensembles de tâches indépendantes est un matroïde.

   Preuve
   - Hérédité : triviale.

- échange : Soit A, B indépendants avec $|A| < |B|$. Il faut montrer qu'il existe $x \in B$ tel que $A \cup \{x\}$ soit indépendant.
  Idée : comparer $N_t(A)$ et $N_t(B)$ pour $t = 1, 2, \ldots, n$.
  On cherche le plus grand $t \leq n$ tel que $N_t(A) \geq N_t(B)$. On sait que $t < n$ donc $N_t(A) \geq N_t(B)$, $N_{t+1}(A) < N_{t+1}(B)$, ..., $N_n(A) < N_n(B)$.
  Dans B il y a plus de tâches de date limite $t+1$ que dans A : on en prend une x qui n'est pas déjà dans A : $A \cup \{x\}$ est indépendant.

3. En déduire un algorithme glouton pour tester l'indépendance d'un ensemble A.

Correction :

Algorithm 2 1

```
1  Trier les éléments de S par poids décroissants
                    w(s_1) ≥ w(s_2) ≥ ··· ≥ w(s_n).
2  A := ∅.
3  pour i := 1 jusqu'à ce que n = |S| faire
4      si A ∪ {s_i} ∈ I alors
5          A := A ∪ {s_i}
6  retourner A
```

4. Application numérique
   Entrée : 7 tâches

i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	7	6	5	4	3	2	1

Correction :

5. Itérations :
   - $A = \{1\}$;
   - $A = \{2, 1\}$;
   - $A = \{2, 1, 3\}$
   - $A = \{2, 4, 1, 3\}$;
   - $A = \{2, 4, 1, 3, 7\}$.
   - Résultat : $\{2, 4, 1, 3, 7\}$ et la pinalité maximale est 5.