

Algorithmique et Programmation avancée

5: Algorithmes gloutons

Principe

Les algorithmes pour les problèmes d'optimisation exécutent en général une série d'étapes, chaque étape proposant un ensemble de choix. Pour de nombreux problèmes d'optimisation, la programmation dynamique est une approche bien trop lourde pour déterminer les meilleurs choix; d'autres paradigmes, plus simples et plus efficaces, peuvent être utilisés. Un algorithme glouton fait toujours le choix qui lui semble le meilleur sur le moment: un choix localement optimal dans l'espoir que ce choix mènera à une solution globalement optimale.

Deux conditions nécessaires à un problème pour qu'un algorithme glouton se prête à sa résolution:

- **Sous-structure optimale:** Une solution optimale du problème contient des solutions optimales de sous-problèmes.
- **Propriété de choix glouton:** On peut assembler une solution globalement optimale en effectuant des choix localement optimaux (gloutons).

Exemple 1. Le problème des pièces de monnaie est résoluble par la méthode gloutonne pour des pièces de valeur $\{1, 2, 5, 10, 20, 50, 100, 200\}$.

Exemple: un problème de choix d'activité

On considère un ensemble d'activités qui rivalisent pour l'utilisation exclusive d'une ressource commune, l'objectif étant de sélectionner un sous ensemble de taille maximale d'activités mutuellement compatibles. Une activité a_i est caractérisée par une heure de début d_i et une heure de fin f_i . On dit que deux activités sont compatibles si leurs intervalles de temps sont disjoints. On veut résoudre le problème à l'aide d'un programme glouton.

Exemple:

a_i	1	2	3	4	5	6	7	8	9	10	11
d_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Idée d'algorithme:

On trie les activités par date de fin croissante, puis on gloutonne, i.e. on choisit une activité si elle est compatible avec la précédente.

Théorème 2. *Cet algorithme glouton est optimal.*

Preuve. Soit a_1 l'élément finissant le plus tôt. On va montrer qu'il existe une solution optimale contenant cette activité. Soit donc une solution optimale arbitraire O , $O = a_{i_1}, a_{i_2}, \dots, a_{i_k}$ avec k le maximum d'activités pouvant avoir lieu dans le gymnase. Il y a deux possibilités: soit $a_{i_1} = a_1$ soit $a_{i_1} \neq a_1$. Dans ce dernier cas, alors on remplace a_{i_1} par a_1 . Comme a_1 finit avant tout autre événement, alors comme a_{i_2} n'intersectait pas avec a_{i_1} , a_{i_2} n'intersecte pas avec a_1 . On peut donc bien trouver une solution optimale ayant comme premier événement l'activité finissant en premier. Ensuite, après avoir placé a_1 , on ne considère que les activités n'intersectant pas avec a_1 , et on réitère la procédure sur les événements restants, d'où la preuve par récurrence.

Matroïdes

La théorie des matroïdes fournit une condition suffisante pour qu'une méthode gloutonne aboutisse à une solution optimale.

Définition 3. Un couple $M = (E, \mathcal{I})$ avec E fini et $\mathcal{I} \subset \mathcal{P}(E)$ est un matroïde si

- **Hérédité:** $X \in \mathcal{I} \Rightarrow \forall Y \subset X, Y \in \mathcal{I}$,

- **Echange:** $(A, B \in \mathcal{I}, |A| < |B|) \Rightarrow \exists x \in B \setminus A \text{ tq } A \cup \{x\} \in \mathcal{I}$

Les éléments de $\mathcal{I}$ sont appelés les indépendants.

Exemple 4.

- Les familles libres d'un espace vectoriel.
- Les forêts (réunion acyclique d'arbres) d'un graphe $G = (S, A)$:
 $M_G = (E_G, \mathcal{I}_G)$ où $E_G = A$, l'ensemble des arêtes, et $F \in \mathcal{I}_G$ si et seulement si le graphe (S, F) forme une forêt.

On dit qu'un matroïde $(E, \mathcal{I})$ est pondéré si l'on dispose d'une fonction de pondération w qui affecte un poids strictement positif $w(x)$ à tout $x \in E$.

Propriété 5. *Les matroïdes sont les couples $(E, \mathcal{I})$ tels que pour toute fonction associant un poids à chaque élément de E , l'algorithme glouton permet de déterminer un indépendant de poids maximum.*

Pseudo-code type

Glouton(M,w):

Entrée: matroïde $M = (S, \mathcal{I})$, fonction de poids w

Sortie: indépendant maximal

$A = \emptyset$

trier S par ordre décroissant de poids

Pour chaque $x \in S$:

si $A \cup \{x\} \in \mathcal{I}$

$A = A \cup \{x\}$

renvoyer A

Le temps d'exécution de Glouton est facile à analyser: Soit $n = |E|$. La phase de tri prend $n \log n$. La boucle est exécutée n fois. Si le test $A \cup \{x\} \in \mathcal{I}$ coûte $O(f(n))$, alors l'algorithme s'exécute en $O(n \log n + nf(n))$.

Applications

Problème de l'arbre couvrant minimal, certains problème d'ordonnancement de tâches, Dijkstra