Les algorithmes gloutons

I Notion de problème d'optimisation

Le cadre de notre étude cette année se limite à des problèmes possédant toujours au moins une solution et un nombre fini de solutions. Dans ce contexte, nous nous interesserons ici plus particulièrement aux **problèmes d'optimisation, c'est à dire des problèmes dans lesquels, il faut minimiser ou maximiser une certaine quantité.** Il faut donc trouver la meilleure solution parmi l'ensemble des solutions possibles. Si on n'y parvient pas, nous tenterons d'estimer notre écart à la solution optimale.
Pour atteindre cet objectif, il y a plusieurs méthodes au choix : méthode exhaustive ou brute , méthode diviser pour régner (vue précédemment), programmation dynamique (vue ultérieurement) et méthode gloutonne (présentée ici)

II La méthode gloutonne

Définition

Un algorithme glouton est un algorithme qui fait un choix optimum local à chaque étape, dans le but d'obtenir une solution optimale globale au problème. Il n'y a pas de retour en arrière : à chaque étape de décision dans l'algorithme, le choix effectué est définitif.

But d'un algorithme glouton

Un algorithme glouton sert donc à résoudre des problèmes d'optimisation.

Vocabulaire

un algorithme glouton ne permettra pas toujours d'obtenir la solution optimale : il fait donc partie du groupe des méthodes approchées ou heuristiques gloutonnes.

Exemple 1

Approche "gloutonne" pour le surveillant de baignade et problème de rendu de monnaie (activités)

III Le problème du rendu de monnaie

A Principe

Enoncé classique du problème du rendu de monnaie

«étant donné un système de monnaie, comment rendre une somme donnée de façon optimale, c'est-à-dire avec le nombre minimal de pièces et billets ?»

A l'issue de l'activité d'introduction, nous avons écrit deux versions en pseudo-code d'une fonction **monnaie** implémentant l'algorithme :

- **Pré-conditions :**
  ⋆ somme s (entier positif) à rendre
  ⋆ liste P $[v_1, v_2, ..., v_n]$ des valeurs des pièces disponibles triées par valeur décroissante
- **Post-condition :** résultat = liste Q $[q_1; q_2; ..., q_n]$ du nombre de pièces par valeur à rendre

Version 1

```
1  Fonction( monnaie(P,s))
2  n ← taille de la liste P ;
3  Q ← [0,.... 0];
4  k ← 1;
5  tant que s > 0 et k < n faire
6      tant que s ⩾ v_k faire
7          q_k ← q_k + 1;
8          s ← s − v_k
9      fin
10     k← k+1
11 fin
12 retourner liste Q des pièces à rendre
```

Version 2

```
1  Fonction( monnaie(P,s))
2  n ← taille de la liste P ;
3  Q ← [0,.... 0];
4  k ← 1;
5  tant que s > 0 et k < n faire
6      si s ⩾ v_k alors
7          q_k ← quotient de la division euclidienne
             de s par v_k;
8          s ← reste de la division euclidienne de s
             par v_k;
9      fin
10     k← k+1 ;
11 fin
12 retourner liste Q des pièces à rendre
```

Exemple 2

Rendre la somme de 2,43€ en disposant du système de pièces de monnaie européen, à savoir (en cents) : [200,100,50,20,10,5,2,1]

B Complexité et terminaison de l'algorithme

La version 2 est plus efficace que la version 1, plus naturelle. Elle permet d'en déduire aisément la **complexité** de l'algorithme de rendu de monnaie :
Les pièces du système de monnaie étant supposées triées par ordre décroissant de valeur et au nombre de n, alors la boucle Tant que correspond à une **complexité** de l'ordre de n. Par contre si la liste des pièces n'est pas triée on passe à une **complexité** de l'ordre $n \log n$.
La **terminaison** de l'algorithme est garantie : en effet, les variants de boucle sont s et k
s est un entier positif ou nul qui décroit strictement et k est un entier positif qui augmente de 1 à chaque étape jusqu'à devenir égal à n qui est un nombre fini. La boucle Tant que s'arrête donc.

C Correction de l'algorithme

Exemple 3

Rendre la somme de 6⋆ en disposant du système de pièces de monnaie (en ⋆) : {4,3,1}

Pistes pour l'exemple 3

Le faire à la main et constater que la méthode gloutonne rendra 1 pièce de 4 ⋆ et 2 pièces de 1 ⋆ alors que la solution optimale rendra 2 pièces de 3 ⋆

L'algorithme glouton du rendu de monnaie ne fournit donc pas toujours la solution optimale :

Vocabulaire

On appelle canonique un système de pièces pour lequel l'algorithme glouton est optimal.

Le système de monnaie européen est canonique alors que le système anglais avant 1971 ne l'était pas.
Pour montrer qu'un système de monnaie n'est pas canonique, il suffit de donner un contre-exemple, par contre démontrer la correction de l'algorithme glouton pour un système de monnaie donné est moins aisé.

preuve de correction pour un système simple

Voici une démonstration de la correction de l'algorithme du rendu de monnaie dans le cas d'un système de monnaie composé des pièces 1, 2 et 5.
La liste P est donc [5,2,1] et la somme à rendre est notée s
La solution produite par l'algorithme glouton est telle que :
$s = q_1 \times 5 + r_1$ avec $0 \leqslant r_1 < 5$
$r_1 = q_2 \times 2 + r_2$ avec $0 \leqslant r_2 < 2$
$r_2 = q_3 \times 1 + r_3$ avec $\underbrace{0 \leqslant r_3 < 1}_{\text{donc } r_3=0}$ d'où $r_2 = q_3 \times 1$
On retrouve que : $s = q_1 \times 5 + r_1 = q_1 \times 5 + q_2 \times 2 + r_2 = q_1 \times 5 + q_2 \times 2 + q_3 \times 1$
On vérifie ainsi que la répartition obtenue permet d'obtenir s. Montrons qu'elle est optimale :
Soit o_1, o_2, o_3 une solution optimale dans ce système canonique.
Cette solution est donc telle que $s = o_1 \times 5 + o_2 \times 2 + o_3 \times 1$
$o_3 < 2$ sinon on pourrait remplacer 2 pièces de 1 par une pièce de 2.
De même, $o_2 < 3$ sinon on pourrait remplacer 3 pièces de 2 par une pièce de 5 et une de 5.
On a de plus $2 \times o_2 + o_1 < 5$ car sinon on pourrait remplacer par une pièce de 5.
On a donc $s = o_1 \times 5 + o_2 \times 2 + o_3 \times 1$ avec $2 \times o_2 + o_1 < 5$
D'où o_1 est le quotient q_1 de la division euclidienne de s par 5 et $o_2 \times 2 + o_3 \times 1$ est le reste r_1 de la division euclidienne de s par 5.
De même, $o_3 < 2$, d'où o_2 est le quotient q_2 de la division euclidienne de r_1 par 2 et $o_3 \times 1$ est le reste r_2 de la division euclidienne de r_1 par 2.
et on en déduit que o_3 est le quotient q_3 de la division euclidienne de r_2 par 1 ;
D'où $o_1 = q_1$, $o_2 = q_2$ et $o_3 = q_3$
L'algorithme glouton est donc correct pour le système {1,2,5} et le système {1,2,5} est canonique.

IV Le problème du sac à dos

A principe

Le problème du "Sac à Dos" ou KP(Knapsack Problem) peut s'énoncer de plusieurs façons, parfois même sans parler de sacs ! Néanmoins, voici un énoncé classique :

Enoncé classique du KP

«Étant donné plusieurs objets possédant chacun un poids et une valeur, et étant donné un poids maximum pour le sac, quels objets faut-il mettre dans le sac de manière à maximiser la valeur totale sans dépasser le poids maximal autorisé ?»

Note pour l'enseignant

Si le groupe s'y prête alors, une présentation plus formelle du problème pourra être réalisée .

Présentation formelle

Soit un ensemble E $\{(v_1;p_1),(v_2;p_2),...,(v_i;p_i),...,(v_n;p_n)\}$ de n éléments et un sac à dos. On note :
v_i : la valeur de l'élément i p_i : le poids de l'élément i p_{max} le poids maximal autorisé dans le sac à dos
Une solution au KP est un ensemble $\{x_1, x_2, ..., x_i, ..., x_n\}$ tel que :

$$x_i = \begin{cases} 1 : \text{si l'élément } (v_i;p_i) \text{ est placé dans le sac} \\ 0 : \text{sinon} \end{cases} \quad \text{et} \sum_{i=1}^{n} p_i \times x_i \leqslant p_{max}$$

Une solution optimale est une solution au KP telle que $\sum_{i=1}^{n} v_i \times x_i$ soit maximale

B Critère de décision

Pour résoudre le KP avec cette stratégie, il nous faut définir **un critère de décision**; on rappelle qu'à chaque étape décisionnelle un choix optimal doit être réalisé sur la base d'un critère défini. Il s'agit d'un choix optimal, local.

Exemple 4

Soit un ensemble E de 5 objets $\{(1;2),(2;3),(9;4),(5;3),(4;6)\}$ où chaque objet est sous la forme (valeur en €, poids) ; voici quelques exemples de critères de décision triviaux :

— Critère n°1 : priorité pour les objets les plus légers

— Critère n°2 : priorité pour les objets les plus lourds

— Critère n°3 : priorité pour les objets de faible valeur

— Critère n°4 : priorité pour les objets de grande valeur

— ....

Le KP contiendra donc nécessairement un tri, qui peut représenter un coût important et augmenter significativement la complexité en temps de l'algorithme.

Par exemple, si on trie la liste selon le premier critère, elle devient $\{(1;2),(2;3),(5;3),(9;4),(4;6)\}$ ou $\{(1;2),(5;3),(2;3),(9;4),(4;6)\}$ puisque les objets C et D ont le même poids.

Un seul choix (poids ou valeur) peut être source d'aléas. En cas d'égalité sur le poids, il semble "naturel" de choisir comme critère secondaire "la plus grande valeur" puisque notre but est de maximiser la valeur totale du sac.

En 1957, G.Dantzig (1914-2005) introduit un critère souvent utilisé pour le KP :

— **Critère n°5 : priorité pour les objets de rapport $\frac{\text{valeur}}{\text{poids}}$ élevé**

Le choix du critère peut grandement influencer la réponse donnée (voir TP n°2) ; reprenons l'exemple précédent avec une contrainte supplémentaire inhérente au KP : le poids maximal dans le sac. On supposera $p_{max} = 10$:

— Liste initiale à trier selon un critère → $\{(1;2),(2;3),(9;4),(5;3),(4;6)\}$

— Critère n°1 → Liste triée $\{(1;2),(5;3),(2;3),(9;4),(4;6)\}$ → Réponse $\{1,1,0,1,0\}$ car $2+3+3 \leq 10 < 2+3+3+4$

— Critère n°5 → Liste triée $\{(9;4),(5;3),(4;6),(2;3),(1;2)\}$ → Réponse $\{0,1,1,1,0\}$ car $4+3+3 \leq 10 < 4+3+3+2$

Remarque : Les deux critères conduisent à des sacs avec 3 objets. La réponse indique avec des '1' la position dans la liste initiale des objets retenus et des '0' sinon. Par ailleurs, le poids total du 2^e sac est égale à p_{max} , **on dit parfois que le sac est maximal.** Enfin, la valeur du second sac est de 16 contre seulement 8 pour le premier : néanmoins, à ce stade, nous ne savons pas si 16 correspond à la solution optimale.

C Pseudo-Code : Algorithme "glouton"

voici une écriture en pseudo-code d'une fonction **KP** implémentant l'algorithme :

- **Pré-conditions :**
  ⋆ poids max p_{max} (réel positif)
  ⋆ Liste "Objets" triée $[(v_1,p_1),(v_2,p_2),...,(v_n,p_n)]$
- **Post-condition :** résultat = Liste "Reponse" contenant des 0 ou des 1 selon que l'objet est sélectionné ou pas

On suppose ici que l'on se dote d'un critère de décision et d'une liste **L d'objets triés par ordre décroissant selon notre critère.** La réponse donne la position dans la liste triée.

La complexité est d'ordre n et la terminaison est assurée par le variant i de la boucle « Pour ». .

```
1  Fonction(KP(P,s))
2  Sac ← 0;
3  Reponse← [ ];
4  pour i=0 à n-1 faire
5      si p_i+Sac ≤ p_max alors
6          Reponse[i] ← 1;
7          Sac ← Sac+p_i;
8      sinon
9          Reponse[i] ← 0;
10     fin
11 fin
12 retourner Reponse
```