TD d'algorithmique avancée
Corrigé du TD 6 : Algorithmes gloutons

Jean-Michel Dischler et Frédéric Vivien

Le coût de la non panne sèche

Le professeur Bell conduit une voiture entre Amsterdam et Lisbonne sur l'autoroute E10. Son réservoir, quand il est plein, contient assez d'essence pour faire n kilomètres, et sa carte lui donne les distances entre les stations-service sur la route.

1. Donnez une méthode efficace grâce à laquelle Joseph Bell pourra déterminer les stations-service où il peut s'arrêter, sachant qu'il souhaite faire le moins d'arrêts possible.
   La solution consiste à s'arrêter le plus tard possible, c'est-à-dire à la station la plus éloignée du dernier point d'arrêt parmi celles à moins de n kilomètres de ce même point.
2. Démontrez que votre stratégie est optimale.
   On procède de la même manière que pour prouver l'optimalité de l'algorithme glouton pour la location de voiture : on considère une solution $F = (x_1, x_2, ..., x_p)$ fournie par notre algorithme et une solution optimale $G = (y_1, y_2, ..., y_q)$, on a donc $p \geq q$ et on cherche à montrer que $p = q$ —ici les deux suites sont bien sûr des suites de stations-service, triées de celle la plus proche du point de départ à celle la plus éloignée.
   Soit k le plus petit entier tel que :

$$\forall i < k, x_i = y_i, \text{ et } x_k \neq y_k.$$

   Par définition de notre algorithme, on a $x_k > y_k$. On construit à partir de la solution optimale $G = (y_1, y_2, ..., y_{k-1}, y_k, y_{k+1}, ..., y_q)$ la suite de stations-service $G' = (y_1, y_2, ..., y_{k-1}, x_k, y_{k+1}, ..., y_q)$. G' est une liste de même taille. Montrons que c'est aussi une solution —et donc une solution optimale. Il nous faut vérifier qu'il n'y a pas de risque de panne sèche c'est-à-dire que :
   - *$x_k - y_{k-1} \leq n$. F est une solution, donc $x_k - x_{k-1} \leq n$. Par conséquent, $x_k - y_{k-1} = x_k - x_{k-1} \leq n$.*
   - *$y_{k+1} - x_k \leq n$. G est une solution, donc $y_{k+1} - y_k \leq n$. D'où $y_{k+1} - x_k < y_{k+1} - y_k \leq n$. Si $y_{k+1} < x_k$, on peut en fait supprimer y_{k+1} de G' et obtenir une solution strictement meilleure ce qui contredirait l'optimalité de G.*

   G' est donc bien une solution optimale de notre problème.
   Nous avons donc construit à partir de G une solution G' qui contient un élément en commun de plus avec F. On itère jusqu'à ce que l'on ait une solution optimale contenant F. Alors $p \leq q$, ce qui prouve le résultat attendu.

Problème du sac à dos

Variante « tout ou rien »

Un voleur dévalisant un magasin trouve n objets, le i^{e} de ces objets valant v_i euros et pesant w_i kilos, v_i et w_i étant des entiers. Le voleur veut bien évidemment emporter un butin de plus grande valeur possible mais il ne peut porter que W kilos dans son sac à dos. Quels objets devra-t-il prendre ?

Variante fractionnaire

Le problème est le même que le précédent, sauf qu'ici le voleur peut ne prendre qu'une *fraction* d'un objet et n'est plus obligé de prendre l'objet tout entier comme précédemment, ou de le laisser.

1. Proposez un algorithme glouton pour la variante fractionnaire.

   On commence par trier les objets suivant leur valeur au kilo, v_i/w_i, décroissante. L'algorithme consiste alors à prendre autant que faire se peut de l'objet de plus grande valeur au kilo —sa totalité si le sac à dos peut le contenir en entier et sinon la quantité nécessaire à remplir le sac— puis recommencer avec les autres objets jusqu'à ce que le sac soit plein.

2. Montrez que cet algorithme est optimal.

   On procède comme pour le problème de la location d'une voiture ou celui de la minimisation des arrêts en station service : on compare ici la solution construite par notre algorithme avec une solution optimale. On compare ces deux solutions après avoir triés les objets qu'elles contiennent par valeur au kilo décroissante. Pour le premier objet pour lequel les quantités diffèrent, on rajoute dans la solution optimale la quantité manquante de l'objet en question en enlevant le poids équivalent en objets de valeur au kilo inférieure —ce qui est toujours possible vu l'ordre dans lequel on compare les deux solutions. On conclut comme précédemment.

3. Quelle est sa complexité ?

   La complexité du tri est en $O(n \log n)$ (voir le cours) et celle de l'algorithme proprement dit en n, où n est le nombre d'objets. La complexité de l'ensemble est donc en $O(n \log n)$.

4. Montrez au moyen d'un contre-exemple que l'algorithme glouton équivalent pour la variante « tout ou rien » n'est pas optimal.

   On considère trois objets : le premier pèse 10 kilos et vaut 60 euros (valeur de 6 euros par kilo), le deuxième pèse 20 kilos et vaut 100 euros (valeur de 5 euros par kilo) et le troisième pèse 30 kilos et vaut 120 euros (valeur de 4 euros par kilo). La stratégie gloutonne précédente prendra initialement le premier objet et ne pourra plus alors en prendre d'autre, quand la solution optimale ici aurait été de prendre les deux autres objets...

5. Proposez un algorithme de programmation dynamique résolvant la variante « tout ou rien ».

   On commence par établir une récurrence définissant la valeur du butin emporté. $SAD(i, w)$ est la valeur maximale du butin s'il n'est composé que de certains des i premiers objets (les objets sont ici numérotés mais pas ordonnés) et qu'il pèse au maximum w kilos. On a plusieurs cas à considérer :

   (a) Le i^e objet est plus lourd que la capacité du sac : on ne peut pas le prendre et le problème se ramène à la recherche du meilleur butin parmi les $i-1$ premiers objets.

   (b) Il est possible de prendre le i^e objet :

   i. On ne le prend pas et le problème se ramène à la recherche du meilleur butin parmi les $i-1$ premiers objets.

   ii. On le prend et la valeur du butin est celle du i^e objet plus celle du butin que l'on peut constituer à partir des $i-1$ premiers objets compte tenu que l'on a pris le i^e objet et que le poids portable est diminué d'autant.

   De ce qui précède on tire l'équation :

$$SAD(i,w) = \begin{cases} SAD(i-1,w) & \text{si } w < w_i, \\ \max(v_i + SAD(i-1, w-w_i), SAD(i-1,w)) & \text{sinon.} \end{cases} \quad (1)$$

   Il nous reste à transcrire cette définition récursive sous la forme d'un algorithme de programmation dynamique :

```
SacÀDos(W, w, v)
  pour w ← 0 à W faire Valeur[0, w] ← 0
  pour i ← 1 à n faire
    pour w ← 0 à W faire
      si w_i > w
        alors Valeur[i, w] ← Valeur[i − 1, w]
        sinon Valeur[i, w] ← max(v_i + Valeur[i − 1, w − w_i], Valeur[i − 1, w])
renvoyer Valeur
```

Cet algorithme nous calcule la valeur optimale du butin (Valeur(n, W)) mais pas sa composition, ce que fait l'algorithme suivant :

BUTIN(*Valeur*, i, W, w, v)

si $w_i > W$
alors renvoyer BUTIN(*Valeur*, $i-1$, W, w, v)
sinon si $v_i + S[i-1, W-w_i] > S[i-1, W]$
alors renvoyer $\{i\} \cup$ BUTIN(*Valeur*, $i-1$, $W-w_i$, w, v)
sinon renvoyer BUTIN(*Valeur*, $i-1$, W, w, v)

Cet algorithme est appelé initialement : BUTIN*(*SACÀDOS*(w, v), n, W, w, v).*

6. Quelle est sa complexité ?

La complexité de l'algorithme SACÀDOS *est en* $\Theta(nW)$ *et celle de l'algorithme* BUTIN *est en* $\Theta(n)$ *(à chaque étape on décrémente i de 1). La complexité totale est donc en* $\Theta(nW)$.