TP10 : programmation dynamique

Algorithmique et projet de programmation — M1 mathématiques

6 décembre 2016

1 Problème du sac à dos

Vous êtes un voleur cambriolant une bijouterie. Vous disposez d'un sac d'une contenance W et vous connaissez le poids w_i et la valeur v_i de chacun des $i \in \{1, \dots, n\}$ objets présents dans la bijouterie. Votre but est de maximiser le butin que vous pouvez transporter dans votre sac.

Pour ce faire, on va invoquer les techniques de programmation dynamique.

Exercice 1. Notons $m_{i,w}$ le butin maximal que l'on peut obtenir avec un sac de contenance w et en ne regardant que les objets $1, \dots, i$.

Exprimer $m_{i,w}$ en fonction de $m_{i-1,w}$, $m_{i-1,w-w_i}$ et v_i en considérant la dichotomie suivante :

- soit $w_i > w$, auquel cas on ne peut même pas prendre l'objet i dans le sac,
- soit $w_i \leq w$ et il faut déterminer si prendre l'objet i est plus rentable que de le laisser sur place.

Exercice 2. En déduire un algorithme récursif naïf calculant la valeur souhaitée, à savoir $m_{n,W}$. (Il n'est pas nécéssaire de l'implémenter en C.)

On effectue maintenant l'optimisation propre à la programmation dynamique, la mémoization.

Exercice 3. Écrire une fonction C de prototype :

```
int memoized_knapsack (int *values, int *weights, int i, int w,
                       int **computed);
```

Elle prendra en paramètres un tableau `values` contenant en case j la valeur v_j, un tableau `weights` contenant en case j le poids w_j, deux index i et w et une matrice `computed`. Cette dernière contient en case (i, w)

- la valeur $m_{i,w}$ définie précédemment si elle a déjà été calculée,
- la valeur -1 sinon.

La fonction doit renvoyer l'entier $m_{i,w}$.

Il s'agit maintenant de mettre en place la deuxième technique propre à la programmation dynamique : se « laisser des indices » lors du remplissage du tableau `computed` afin de pouvoir remonter les choix faits lors de la résolution récursive.

Exercice 4. Modifier légèrement la fonction précédente de façon à obtenir un fonction

```
int memoized_knapsack (int *values, int *weights, int i, int w,
                       int **computed, int **choices);
```

où tous les paramètres restent les même que précédemment et où la matrice `choices` contient en case $(i\,,w)$

- soit 0 si on sait qu'on ne choisit pas l'objet i,
- soit 1 si on sait qu'on choisit l'objet i,
- soit -1 si la valeur n'est pas encore calculée.

Exercice 5. En déduire une fonction

```
int knapsack (int *values, int *weights, int n, int W);
```

résolvant notre problème de voleur.