

Algorithmique

TD n° 8 : programmation dynamique

Exercice 1 : le rendu de monnaie

On considère le problème suivant : dans une monnaie donnée, il y a un certain ensemble P de pièces ayant cours. Par exemple, pour l'euro (en centimes),

$$P = \{1, 2, 5, 10, 20, 50, 100, 200\}$$

Le but de cet exercice est de parvenir à produire une somme S en utilisant le moins de pièces possible. Ainsi, pour $S = 60$ centimes, on préférera $50 + 10$ à $20 + 20 + 20$. On considère que la somme est toujours réalisable avec (une montagne de) la plus petite pièce.

1. Proposer un algorithme simple pour les pièces en centimes d'euros et analyser sa complexité. Est-ce qu'il est toujours optimal ?
2. Imaginer un pays où cet algorithme ne marche pas (il faut produire un P et un S où il échoue).
3. Vers l'algorithme 1 : donner une relation de récurrence exprimant le nombre minimal de pièces à utiliser en fonction de l'ensemble des pièces autorisées (au début on n'a droit qu'à un seul type de pièce, puis on introduit progressivement les autres) et de la somme à atteindre.
4. Vers l'algorithme 2 : donner une relation de récurrence exprimant le nombre de pièces à utiliser en fonction de la somme à atteindre uniquement.
5. Constaté que l'implémentation directe de ces récurrences donne des programmes de mauvaise complexité.
6. Donner deux algorithmes de programmation dynamique pour les deux versions. Comparez leur complexité.
7. Comment faire pour avoir la liste et non plus le nombre de pièces à utiliser ?

Exercice 2 : mise en page et impression d'un texte

On dispose d'un fichier texte sans passage à la ligne (il s'agit d'un seul paragraphe) que l'on veut imprimer sur une feuille de taille donnée. Chaque caractère imprimé occupe la même largeur, y compris les espaces, comme sur une machine à écrire mécanique. Le problème est de découper le texte en lignes, pour minimiser les espaces qui restent à la fin de chaque ligne. Un mot ne peut pas être découpé.

Formellement, le texte est une suite de n mots de longueurs $\ell_1, \ell_2, \dots, \ell_n$, mesurées en nombre de caractères. Chaque ligne contient exactement M caractères, blancs compris. On suppose que toutes les longueurs $\ell_1, \ell_2, \dots, \ell_n$ sont inférieures ou égales à M (un mot peut tenir entièrement dans une ligne). Si une ligne contient les mots i à j (inclus), où $i \leq j$, et qu'on laisse exactement un espace entre deux mots, le nombre de caractères blancs à la fin de la ligne est

$$B := M - j + i - \sum_{k=i}^j \ell_k.$$

On veut minimiser la somme, sur toutes les lignes, des cubes¹ des B (nombres de caractères blancs présents à la fin de la ligne). Cette somme est appelé coût. On dira alors que la présentation du texte est « équilibrée ».

Voici un exemple du problème avec longueur de la ligne $M = 14$. Soit donné le texte suivant :

1. des grandes espaces à la fin de la ligne sont fortement pénalisées

La récursion est un concept important

On a $l_1 = 2, l_2 = 9, l_3 = 3, l_4 = 2, l_5 = 6$ et $l_7 = 9$. Pour l'alignement

```
La récursion   | B = 2
est un concept | B = 0
important      | B = 5
```

on a successivement des valeurs de B de 2, 0 et 5 et donc une somme des cubes de 133.

1. Donnez un algorithme naïf (glouton) pour le problème.
2. Montrez que cet algorithme peut échouer en l'appliquant sur l'exemple suivant pour des lignes de longueur 17 :

La programmation dynamique est une des bases algorithmiques

et donner la solution optimale.

3. Donner un algorithme de programmation dynamique qui résout le problème.
Indication : donner une relation de récurrence pour $\text{cout}(j)$ qui dénote le coût de placer les mots de 1 à j . Pour cela utiliser $\text{cout}(i, j)$ le coût d'une ligne avec les mots i à j .
4. Donner la complexité en temps et en espace de l'algorithme précédent.
5. Comment faut-il modifier l'algorithme pour que la dernière ligne ne soit pas prise en compte dans la somme des cubes des nombres de caractères blancs ?

Exercice 3 : le jeu de Double-Nim

On considère le jeu à deux joueurs suivant. Au début de la partie il y a n allumettes sur la table. Les deux joueurs jouent à tour de rôle en enlevant des allumettes. Celui qui enlève la dernière allumette a gagné. Les règles sont :

- au premier tour, le premier joueur ne peut pas prendre toutes les allumettes d'un coup ;
- aux tours suivants, un joueur doit prendre au minimum une allumette et au maximum le double du nombre pris par l'autre joueur au tour précédent.

1. Donnez une récurrence $J(n, m)$ disant si une position donnée est gagnante ou perdante selon le nombre n d'allumettes restantes et le nombre maximum m d'allumettes que l'on peut prendre.
2. Donner la table de J pour $n = 8$
3. Programmez cette récurrence sous forme d'une fonction récursive. Affichez pour tout n si le premier joueur a une stratégie gagnante, et combien d'appels récursifs ont été faits.
4. Interpréter l'affichage :
 - proposer une hypothèse sur les nombres n pour lesquels le premier joueur gagne ;
 - constater que le nombre d'appels récursifs est exponentiel en n .
5. Améliorez votre programme à l'aide de la programmation dynamique.
6. Affichez le déroulement d'une partie où un joueur qui peut gagner prend le plus petit nombre possible d'allumettes tel qu'il gagne, et un joueur qui ne peut pas gagner fait n'importe quoi.