Passeport pour l'informatique

# EXERCICES D'ALGORITHMIQUE

## Oraux d'ENS

Ouvrage collectif — Coordination Jean-Claude Bajard

INTERNATIONAL™ THOMSON PUBLISHING

# 1

# *Vers la récursivité*

« *Diviser pour régner. Faire moins, pour faire plus.* »

## *1.1 L'angoisse de la panne sèche*

Une route comporte $n$ stations-service. La première est à une distance $d_1$ du départ, la la deuxième est à une distance $d_2$ de la première, la troisième à une distance $d_3$ de la deuxième, etc. La fin de la route est à une distance $d_{n+1}$ de la $n$-ième station-service. Les distances $d_i$ sont représentées par un tableau de flottants.

Un automobiliste prend le départ de la route avec une voiture dont le réservoir d'essence est plein. Sa voiture est capable de parcourir une distance $r$ (mais pas plus !) avec un plein. On suppose que $r$ est supérieur ou égal à chacun des $d_i$ et inférieur à leur somme, sinon le problème n'a pas de sens.

### Question 1

L'automobiliste désire faire le plein le moins souvent possible. Écrire une fonction OCaml qui détermine à quelles stations-service il doit s'arrêter.

### Question 2

Maintenant, l'automobiliste part avec un réservoir vide. Il doit au départ acheter de l'essence (autant qu'il veut dans la contenance de son réservoir) au prix de $E_0$ euros par litre. Par la suite, à la station numéro $i$, l'essence coûte $E_i$ euros par litre. L'automobiliste consomme $kt$ litres pour parcourir une distance $t$, et le réservoir peut contenir $L$ litres d'essence. Écrire une fonction OCaml qui indique à quelles stations-service l'automobiliste doit s'arrêter, et combien de litres il doit prendre à chaque fois, pour que son trajet lui coûte le moins cher possible.

CORRIGÉ

Question 1

Un algorithme glouton est optimal : on regarde jusqu'où on peut aller et on s'arrête à la première station avant ce point, puis on recommence.

```
let rapide (d:int array) (r:int) : int list =
  let n = Array.length d in
  let rec aux acc i reste =
    if i=n then acc else
    if d.(i) > reste then
      aux (i::acc) (i+1) (r-d.(i))
    else
      aux acc (i+1) (reste-d.(i))
  in List.rev (aux [] 0 r)
```

Cet algorithme est bien optimal. En effet, soit une autre solution. Son premier plein est forcément au même endroit ou avant le premier plein de la solution de l'algorithme. Par récurrence, son $i$-ième plein se produit au même endroit ou avant le $i$-ième plein de l'algorithme.

Question 2

Repérons tout d'abord la station où l'essence est la moins chère, et faisons deux remarques :

- L'automobiliste doit faire en sorte que son réservoir soit vide au moment où il arrive à cette station (risqué en pratique !). En effet, s'il lui reste $x$ litres d'essence à ce moment là, il les a payés plus cher auparavant pour rien.
- Toujours à cette station, il doit prendre juste ce qu'il faut d'essence pour aller jusqu'au bout si la capacité de son réservoir le permet, soit remplir complètement son réservoir.

On recommence pour les deux sous-problèmes consistant à aller du début à cette station-service et de cette station service à la fin. Pas besoin de récursivité, la stratégie qui s'en déduit est la suivante. À chaque fois que l'on passe devant une station service :

- Soit il existe dans le rayon d'action que peut atteindre la voiture avec un plein des stations moins chères, dans ce cas, on prend le minimum d'essence (éventuellement pas du tout) permettant d'aller à la première station moins chère que celle où l'on se trouve (attention : la première, c'est-à-dire pas nécessairement la moins chère des stations accessibles).
- Soit toutes les stations que l'on peut atteindre sont plus chères, dans ce cas on remplit complètement le réservoir, sauf si on peut atteindre l'arrivée, dans quel cas on prend juste ce qu'il faut pour y aller.

L'écriture de la fonction OCaml ne représente plus aucune difficulté, elle est laissée au lecteur.

⋆⋆⋆⋆⋆⋆⋆⋆

## 1.2 Produit de plusieurs matrices

On cherche à effectuer un produit de matrices

$$M_1 \times M_2 \times ... \times M_n$$

où la matrice $i$ comporte $r_{i-1}$ lignes et $r_i$ colonnes, en effectuant la moins possible de multiplications de réels (la multiplication de deux matrices se fera par la méthode usuelle).

### Question 1

Écrire une fonction qui multiplie une matrice $A$ de taille $p \times q$ par une matrice $B$ de taille $q \times r$. Combien de multiplications de réels nécessite-t-elle ?

### Question 2

On effectue le produit $M_1 \times M_2 \times M_3 \times M_4$, où $M_1$ est de taille $10 \times 20$, $M_2$ de taille $20 \times 50$, $M_3$ de taille $50 \times 1$ et $M_4$ de taille $1 \times 100$. Combien de multiplications de réels fera-t-on si on calcule le produit dans les deux ordres (indiqués par les parenthèses) suivants :

$$M_1 \times (M_2 \times (M_3 \times M_4))$$

$$(M_1 \times (M_2 \times M_3)) \times M_4$$

### Question 3

Écrire une fonction pour trouver le plus petit nombre de multiplications de réels nécessaires pour calculer $M_0 \times M_1 \times ... \times M_{n-1}$, où la matrice $i$ comporte $r_{i-1}$ lignes et $r_i$ colonnes. Aide : définir une valeur `m.(i).(k)` égale au plus petit nombre de multiplications pour calculer $M_i \times M_{i+1} \times ... \times M_k$.

CORRIGÉ

### Question 1

On a classiquement :

```
type matrix = float array array
let produit (a:matrix) (b:matrix) : matrix =
  let p = Array.length a in
  let q = Array.length b in
  let r = Array.length b.(0) in
  let c = Array.make_matrix p r 0. in
  for i=0 to p-1 do
    for j=0 to r-1 do
      for k=0 to q-1 do
        c.(i).(j) <- c.(i).(j) +. a.(i).(k) *. b.(k).(j)
```

```
      done
    done
  done;
  c
```

Cette fonction utilise $p \times q \times r$ multiplications.

### Question 2

Il faut $50 \times 1 \times 100 + 20 \times 50 \times 100 + 10 \times 20 \times 100 = 125\,000$ multiplications pour la première méthode et $20 \times 50 \times 1 + 10 \times 20 \times 1 + 10 \times 1 \times 100 = 2200$ pour la seconde méthode.

### Question 3

Il suffit de voir comment s'obtient `m.(i).(j)`. On trouve aisément :

$$\texttt{m.(i).(j)} = \begin{cases} 0 & \text{si } i = j \\ \min_{i \leq k < j} \{\texttt{m.(i).(k)} + \texttt{m.(k+1).(j)} + r_{i-1} r_k r_j\} & \text{sinon} \end{cases}$$

Il n'est pas nécessaire d'écrire une fonction récursive, on itère sur la longueur des chaînes de multiplication.

```
let nb_min_multiplications (r:int array) : int =
  let n = Array.length r - 1 in
  let m = Array.make_matrix n n 0 in

  for l = 2 to n do
    for i = 0 to n - l do
      let j = i + l - 1 in
      m.(i).(j) <- max_int;
      for k = i to j - 1 do
        let q = m.(i).(k) + m.(k+1).(j) + r.(i) * r.(k+1) * r.(j+1) in
        if q < m.(i).(j) then
          m.(i).(j) <- q
      done
    done
  done;
  m.(0).(n-1)
```

⋆⋆⋆⋆⋆⋆⋆⋆