Option Informatique en Spé MP et MP*
Problème du loueur de skis : le corrigé

Boîte à outils Caml

Question 1 • Voici deux formulations, la première est récursive non terminale, la deuxième est récursive terminale.

```
let rec intervalle p q =
 if p>q then [] else p::(intervalle (p+1) q) ;;

let rec intervalle p q =
 let rec aux accu = function
  | k when k<p -> accu
  | k -> aux (k::accu) (k-1)
 in aux [] q ;;
```

Question 2 • Il suffit d'appliquer (avec `map`) la fonction `q -> x::q` à chaque élément q de ℓ.

```
let ajoute_devant x = map (fun q -> x::q) ;;
```

▶ Une autre solution consisterait à utiliser l'opérateur `prefix ::`. Malheureusement, le type de cet opérateur est `'a * 'a list -> 'a list` ; or un opérateur de type `'a -> 'a list -> 'a list` serait davantage dans l'esprit de Caml. Pas de problème : on écrit d'abord une fonctionnelle `curry` qui transforme une fonction de type `a * b -> c` en une fonction de type `a -> b -> c` et le tour est joué :

```
let curry f = function x -> function y -> f(x,y);;
let ajoute_devant x = map ((curry prefix ::) x);;
```

Le mécanisme qui transforme une fonction de type `a * b * ... * x -> y` en une autre fonction, de type `a -> b -> ... -> x -> y` est appelé *curryfication*, par référence au logicien Haskell Curry. Notons que son prénom a été choisi pour baptiser un langage fonctionnel, qui, à la différence de Caml, fonctionne *paresseusement* : les arguments des fonctions ne sont évalués que quand ils deviennent nécessaires. Le rayon informatique de la bibliothèque du lycée contient un manuel d'initiation au langage Haskell.

En Caml, les arguments sont systématiquement évalués avant d'être transmis à la fonction. On trouvera sur le site Web de la *Lettre de Caml* un article de Pierre Weis expliquant coment réaliser en Caml l'évaluation paresseuse !

Question 3 • Le calcul de la somme des éléments d'une `int list` se fait très simplement, avec `it_list`. Nous rédigeons ensuite une fonction qui calcule la somme des coefficients de la ligne i, dont l'indice de colonne est compris entre k et k' :

```
let somme_liste = it_list (prefix +) 0 ;;

let somme_rangée m i k k' =
 somme_liste (map (fun j -> m.(i).(j)) (intervalle k k')) ;;

let somme_bloc m l k l' k' =
 somme_liste (map (fun i -> somme_rangée m i k k') (intervalle l l')) ;;
```

Injections, injections croissantes

Question 4 • La formule $\mathrm{a}(n,p) = \dfrac{n!}{(n-p)!}$ peut s'établir de deux façons différentes.

1. Pour choisir une injection de $[\![1,p]\!]$ dans $[\![1,n]\!]$, on commence par choisir son image F, qui est une p-partie de $[\![1,n]\!]$: il y a $\binom{n}{p}$ choix possibles. Ensuite, on choisit la bijection de $[\![1,p]\!]$ sur F : il y a $p!$ choix possibles. On obtient ainsi $p!\binom{n}{p}$, soit $\frac{n!}{(n-p)!}$.

2. Pour définir une injection φ de $[\![1,p]\!]$ dans $[\![1,n]\!]$, on commence par choisir $\varphi(1)$: il y a n choix possibles ; puis $\varphi(2)$: il y a $n-1$ choix possibles ; et ainsi de suite jusqu'à $\varphi(p)$: il y a $n-p+1$ choix possibles. Total : $n(n-1)\cdots(n-p+1) = \frac{n!}{(n-p)!}$.

• La formule $\mathrm{b}(n,p) = \binom{n}{p}$ résulte de la remarque suivante : une injection croissante φ de $[\![1,p]\!]$ dans $[\![1,n]\!]$ est parfaitement définie par son image F, qui est une p-partie de $[\![1,n]\!]$. En effet, on aura alors $\varphi(1) = \min(F)$ puis $\varphi(2) = \min\big(F \setminus \{\varphi(1)\}\big)$ et ainsi de suite.

Question 5 • Il faut savoir que, sur une machine à architecture 32 bits (ce qui est le cas des familles PowerPC et Pentium & Co), le type `int` de Caml permet les calculs dans $\mathbb{Z}/2^{31}\mathbb{Z}$, les représentants canoniques étant les éléments de l'intervalle discret $[\![-2^{30}, 2^{30}-1]\!] = [\![-1073741824, 1073741823]\!]$. Observons les deux dernières factorielles requises par le calcul de notre ami Jean-Maurice : $12! = 479001600 < 1073741823$ et $13! = 6227020800 > 1073741823$. Lorsqu'on lui demande `factorielle 13`, Caml rend donc comme valeur :

$$(13! + 2^{30}) \bmod 2^{31} - 2^{30} = (6227020800 + 1073741824) \bmod 2147483648 - 1073741824 = -215430144$$

Ensuite, Caml calcule (sans erreur) $12! = 479001600$ puis $(-215430144)/479001600 = -(215430144/479001600)$ qui est nul puisque $215430144 < 479001600$.

Question 6 • On utilise la relation $\binom{n}{n-p} = \binom{n}{p}$ lorsque $p > n-p$. Ensuite, pour $p \geqslant 1$, on écrit :

$$\binom{n}{p} = \frac{n!}{p!(n-p)!} = \frac{n(n-1)!}{p(p-1)!(n-p)!} = \frac{n}{p} \times \frac{(n-1)!}{(p-1)!(n-p)!} = \frac{n}{p}\binom{n-1}{p-1}$$

Comme $\binom{n}{p}$ est entier, p doit diviser le produit $n\binom{n-1}{p-1}$. Ceci, joint à $\binom{n}{0} = 1$, mène à la formulation suivante :

```
let b n p =
 let rec aux n' = function
  |  0 -> 1
  | p' -> let r = aux (n'-1) (p'-1) in (n'*r)/p'
 in aux n (min p (n-p)) ;;
```

Observons que la suite de valeurs calculées par la fonction `aux` est croissante puisqu'à chaque appel $n' > p'$; donc le plus grand résultat intermédiaire est le produit calculé tout à la fin, soit précisément $p\,\mathrm{b}(n,p)$. À titre d'exemple, la fonction `pascal` que voici construit la ligne n du triangle de Pascal :

```
let pascal n = map (fun p -> b n p) (intervalle 0 n);;
```

Question 7 • La question demande de présenter chaque injection f comme la liste $\big(f(j)\big)_{1\leqslant j\leqslant p}$. Il est clair que la nature de l'ensemble de départ est indifférente : seul compte son cardinal p. Il suffit de s'appuyer sur les deux remarques suivantes :

1. si $p = 1$, chaque injection est décrite par une liste de longueur 1 ;
2. si $p \geqslant 2$, chaque choix d'un élément i de l'ensemble d'arrivée E nous permet de construire de façon naturelle l'ensemble des injections qui envoient 1 sur i, à partir de l'ensemble des injections de $[\![2,p]\!]$ dans $E \setminus \{i\}$.

Ceci mène au programme suivant ; on y envisage aussi le cas $p > n$, par souci d'exhaustivité. La fonction `détache` construit la liste de toutes les décompositions possibles d'un ensemble E non vide (présenté sous forme d'une liste) sous forme d'un couple $(i, E \setminus \{i\})$. Ensuite, on applique récursivement `inj` pour construire la liste des injections de $[\![2,p]\!]$ dans chacun des ensembles $E \setminus \{i\}$. On place i (en tant qu'image de 1) en tête de chacune des listes représentant une injection. Enfin, comme on obtient une liste de listes de listes, il faut effectuer une mise en plat, avec `flat`.

```
let rec filtre p = function
 | [] -> []
 | t::q when p t -> t::(filtre p q)
 | _::q -> filtre p q ;;
```

```
let détache l = map (fun i -> (i,filtre (prefix <>i) l)) l ;;

let flat l = it_list (prefix @) [] l;;

let rec inj p = function
 | [] -> [[]]
 | z when p=1 -> map (fun x -> [x]) z
 | z -> let z1 = détache z in
        let z2 = map (fun (x,y) -> (x,inj (p-1) y)) z1 in
        let z3 = map (fun (x,y) -> ajoute_devant x y) z2
        in flat z3 ;;

let injections n p = if p>n then failwith "impossible"
 else inj p (intervalle 1 n) ;;
```

Question 8 • Soient $(s_i)_{1\leqslant i\leqslant p}$ et $(d_j)_{1\leqslant j\leqslant n}$ deux listes d'entiers, classées par ordre croissant. La fonction `aux` construit l'ensemble des injections croissantes de l'ensemble $S = \{s_i \mid |1 \leqslant i \leqslant p\}$ dans l'ensemble $D = \{d_j \mid |1 \leqslant j \leqslant n\}$ en procédant comme suit :

- si $p \geqslant 1$ et $n \geqslant 1$, elle construit la liste des injections f qui vérifient $f(s_1) = d_1$ et la liste des injections f qui vérifient $f(s_1) > d_1$, puis concatène ces deux listes ;
- si $p = 0$, le résultat est l'injection de l'ensemble vide dans D, représentée par la liste vide ;
- enfin, si $p > 0$ et $n = 0$, le résulat est la liste vide, puisque dans ce cas il n'existe aucune injection de S dans D.

On établit sans difficulté que la liste obtenue est classée par ordre lexicographique. La traduction en Caml est immédiate :

```
let injections_croissantes n p =
 let rec aux src dst = match src with
  | [] -> [[]]
  | t::_ when dst = [] -> []
  | t::q -> (ajoute_devant (hd dst) (aux q (tl dst))) @ (aux src (tl dst))
 in aux (intervalle 1 p) (intervalle 1 n) ;;
```

Le problème du loueur de skis

Question 9 • Supposons $p = \mathcal{O}(1)$: soit $K \in \mathbb{N}^*$ tel que tous les p envisagés soient majorés par K. Alors $\mathrm{a}(n,p) = n(n-1)\cdots(n-p+1) \leqslant n^p \leqslant n^K$, donc $\mathrm{a}(n,p) = \mathcal{O}(n^K)$: $\mathrm{a}(n,p)$ est polynomial en n.

• Supposons $n - p = \mathcal{O}(1)$: soit $K \in \mathbb{N}^*$ tel que tous les p envisagés vérifient $n - p \leqslant K$. Alors $\mathrm{a}(n,p) = \frac{n!}{(n-p)!} \geqslant \frac{n!}{K!}$ donc $n! = \mathcal{O}\big(\mathrm{a}(n,p)\big)$: comme $\mathrm{a}(n,p) \leqslant n!$, on peut affirmer que $\mathrm{a}(n,p) = \Theta(n!)$.

• Supposons $n = 2p$. Rappelons la formule de Stirling : $n! \underset{n\to\infty}{\sim} \Big(\frac{n}{e}\Big)^n \sqrt{2\pi n}$. On aura alors :

$$\mathrm{a}(n,p) = \frac{(2p)!}{p!} \underset{n\to\infty}{\sim} \frac{(2p/e)^{2p}\sqrt{4\pi p}}{(p/e)^p\sqrt{2\pi p}} = \frac{2^{2p}p^p\sqrt{2}}{e^p} = \frac{2^p(2p)^p\sqrt{2}}{e^p} = \sqrt{2}\Big(\frac{2n}{e}\Big)^{n/2}$$

Question 10 • Notons c la taille de l'unique client. Déterminer le minimum de $|c - s_j|$ lorsque j décrit l'intervalle discret $[\![1,n]\!]$ coûte $\mathcal{O}(n)$ opérations (pour une preuve, voir la question 25).

• Supposons les skis rangés par ordre croissant. S'il n'y a qu'une paire, ou deux paires, la réponse est immédiate. Sinon, notons $\nu = \lceil n/2 \rceil$. Comparons c à la taille de la paire de skis médiane s_ν. Si $c = s_\nu$, c'est fini ; si $c < s_\nu$ il suffit de poursuivre la recherche dans la liste $(s_j)_{1\leqslant j\leqslant \nu}$; enfin, si $c > s_\nu$ il suffit de poursuivre la recherche dans la liste $(s_j)_{\nu\leqslant j\leqslant n}$. Avec cette méthode dichotomique, on effectuera $\mathcal{O}(\lg n)$ opérations.

Question 11 • Quitte à permuter c et s, on peut supposer $c_1 \leqslant s_1$. On aura alors $c_1 \leqslant s_2$, donc $|c_1 - s_1| = s_1 - c_1$ et $|c_1 - s_2| = s_2 - c_1$. Par ailleurs, $|s_1 - s_2| = s_2 - s_1$. En appliquant l'inégalité triangulaire, il vient :

$$\begin{aligned} \|f\| &= |c_1 - s_1| + |c_2 - s_2| = s_1 - c_1 + |c_2 - s_2| = s_1 - c_1 + |(c_2 - s_1) + (s_1 - s_2)| \\ &\leqslant s_1 - c_1 + |c_2 - s_1| + |s_1 - s_2] = s_1 - c_1 + |c_2 - s_1| + s_2 - s_1 = s_2 - c_1 + |c_2 - s_1| = |g| \end{aligned}$$

Question 12 • L'affectation $h : i \mapsto i$ est optimale. En effet, soit f une autre affectation. Il existe un couple (i, j) d'indices tels que $i < j$ et $f(i) > f(j)$. Considérons l'affectation g définie par $g(i) = f(j)$, $g(j) = f(i)$ et $g(k) = f(k)$ pour $k \notin \{i, j\}$. Le résultat de la question précédente montre que $\|g\| \leqslant \|f\|$; en effet :

$$\begin{aligned} \|f\| - \|g\| &= \sum_{1 \leqslant k \leqslant n} |c_k - s_{f(k)}| - \sum_{1 \leqslant k \leqslant n} |c_k - s_{g(k)}| = |c_i - s_{f(i)}| + |c_j - s_{f(j)}| - |c_i - s_{g(i)}| - |c_j - s_{g(j)}| \\ &= |c_i - s_{f(i)}| + |c_j - s_{f(j)}| - |c_i - s_{f(j)}| - |c_j - s_{f(i)}| \end{aligned}$$

On ne restreint pas la généralité en supposant $i = 1$, $j = 2$; nous sommes bien dans la situation de la question 11. On constate que l'on ne peut pas dégrader la situation, en remettant dans le bon ordre chaque couple réalisant une inversion pour f. En répétant un nombre fini de fois cette opération, on établit $\|f\| \geqslant \|h\|$.

• Une telle affectation sera obtenue en triant les deux listes, donc pour un coût $\mathcal{O}(n \ln n)$ puisque $p \leqslant n$.

Question 13 • Il existe trois types d'affectations :

- celles qui associent le client p à la paire de skis n. Elles associent les clients 1 à $p-1$ à certaines des paires de skis 1 à $n-1$; le coût minimal d'une telle affectation est donc $|c_p - s_n| + \gamma_{p-1,n-1}$.
- celles qui associent un client j autre que p à la paire de skis n. Notons k la paire de skis qui est attribuée au client p. On aura $c_j \leqslant c_p$ et $s_p \geqslant s_k$; on n'augmente pas le coût de l'affectation, en échangeant les paires attribuées aux clients j et p, donc le coût minimal d'une affectation de ce type est au moins égal à $|c_p - s_n| + \gamma_{p-1,n-1}$.
- celles qui n'associent la paire de skis n à aucun client ; le coût minimal d'une affectation de ce type est $\gamma_{p,n-1}$.

On en déduit immédiatement la formule demandée.

Question 14 • Il suffit de remplir un tableau donnant la valeur de $\gamma_{j,k}$ pour $1 \leqslant j \leqslant p$, $j \leqslant k \leqslant n$. La ligne 1 de ce tableau (qui correspond au cas où un seul client se présente) peut être remplie pour un coût $\mathcal{O}(n)$: en effet, si l'on a obtenu $\gamma_{1,k}$, alors $\gamma_{1,k+1} = \min\big(\gamma_{1,k}, |c_1 - s_{k+1}|\big)$. On peut ensuite remplir le tableau par lignes consécutives, en allant de gauche à droite dans chaque ligne ; on aura $\gamma_{j+1,j+1} = \gamma_{j,j} + |c_{j+1} - s_{j+1}|$ d'après le résultat de la question 12 ; et la formule établie à la question 13 permet de poursuivre le remplissage de la ligne. Nous constatons que le coût du remplissage de chaque case est un $\mathcal{O}(1)$; le nombre de cases est majoré par $np \leqslant n^2$, et le coût du tri préalable est $\mathcal{O}(n \ln n)$, donc dominé par n^2. Conclusion : le coût total est bien un $\mathcal{O}(n^2)$. On obtient $\gamma(c, s)$ par lecture de la case $\gamma_{p,n}$.

• Si l'on ne prend pas de précautions particulières, l'espace mémoire requis est de np cases. En fait, on constate qu'une fois la $(j+1)$-ième ligne construite, on peut se débarasser de la j-ième ; donc il suffit de conserver deux lignes en mémoire : la dernière calculée, et celle qui est en cours de calcul. L'espace mémoire requis est ainsi ramené à $2n$ cases. Avec un peu de réflexion, on peut n'utiliser que $n + \mathcal{O}(1)$ cases ...

Question 15 • La construction de la matrice γ est facilitée si l'on convertit en vecteurs les deux listes c et s. De plus, pour alléger les écritures, nous définissons la fonction `ecart`, qui calcule $|c_j - s_k|$ pour $1 \leqslant j \leqslant p$ et $1 \leqslant k \leqslant n$. Enfin, n'oublions pas que l'indexation des vecteurs commence à 0, en Caml !

```
let gamma c s =
 let p = list_length c and vc = vect_of_list c
 and n = list_length s and vs = vect_of_list s in
 let ecart j k = abs( vc.(j) - vs.(k) ) in
 let g = make_matrix p n 0 in
 g.(0).(0) <- ecart 0 0;
 for k = 0 to n - 2 do
```

```
 g.(0).(k+1) <- min3 g.(0).(k) (ecart 0 (k+1))
done;
for j = 0 to p - 2 do
 g.(j+1).(j+1) <- g.(j).(j) + ecart (j+1) (j+1);
 for k = j+1 to n-2 do
  g.(j+1).(k+1) <- min g.(j+1).(k) (ecart (j+1) (k+1) + g.(j).(k))
 done
done;
g ;;
```

Question 16 • Une fois obtenue la matrice γ, le loueur sait quel paire de skis affecter au plus grand skieur : c'est celle dont le numéro k est le plus petit à vérifier $\gamma_{p,k} = \gamma_{p,n}$. Une fois le client c_p ainsi satisfait, il conserve le bloc formé des $p-1$ premières lignes et des $k-1$ premières colonnes passe au client suivant !

Question 17 • La fonction `meilleur_ski` détermine la valeur de k associée à la ligne j (ou plutôt, à ce qui en reste).

```
let meilleur_ski g j kmax =
 let rec aux = function
 | k when k = kmax -> k
 | k when g.(j).(k) < g.(j).(k-1) -> k
 | k -> aux (k-1)
in aux kmax ;;

let affectation c s =
 let g = gamma c s in
 let p = vect_length g and n = vect_length g.(0) in
 let ski = make_vect p 0 in
  ski.(p-1) <- meilleur_ski g (p-1) (n-1);
  for j = p-2 downto 0 do ski.(j) <- meilleur_ski g j (ski.(j+1)-1) done ;
  list_of_vect ski ;;
```

▶ Je n'ai pas réussi à déterminer l'origine exacte de ce problème. La rumeur dit qu'il provient de l'*Instructor's Book* fourni lorsque l'on acquiert un grand nombre d'exemplaires de [3]. Il a été posé à l'écrit du concours d'entrée 1997 en troisième année à l'E.N.S. de Cachan, ainsi qu'à l'oral du concours d'entrée à l'E.N.S. de Paris. On trouve une solution de ce problème dans [1].

Matrices de Monge

Question 18 • Il s'agit de prouver l'inégalité $|s_i - c_j| + |s_{i+1} - c_{j+1}| \leqslant |s_i - c_{j+1}| + |s_{i+1} - c_j|$. On sait que $s_i \leqslant s_{i+1}$ et $c_j \leqslant c_{j+1}$. On ne restreint pas la généralité en supposant $s_i \geqslant c_j$; alors $|s_i - c_j| = s_i - c_j$ et $|s_{i+1} - c_j| = s_{i+1} - c_j$. Nous sommes ainsi ramenés à établir $s_i - c_j + |s_{i+1} - c_{j+1}| \leqslant |s_i - c_{j+1}| + s_{i+1} - c_j$ soit encore $|s_{i+1} - c_{j+1}| \leqslant |s_i - c_{j+1}| + s_{i+1} - s_i$, ce qui n'est qu'une conséquence de l'inégalité triangulaire puisque $s_{i+1} - s_i = |s_{i+1} - s_i|$.

Question 19 • Prouvons d'abord le sens direct. Soient i, j, k et ℓ tels que $1 \leqslant i \leqslant \ell \leqslant n$ et $1 \leqslant j \leqslant k \leqslant p$ La matrice m vérifie $m_{r,c} + m_{r+1,c+1} \leqslant m_{r,c+1} + m_{r+1,c}$ quels que soient $r \in [\![1, n-1]\!]$ et $c \in [\![1, p-1]\!]$; cette condition peut aussi s'écrire $m_{r+1,c+1} - m_{r+1,c} \leqslant m_{r,c+1} - m_{r,c}$. Sommons cette inégalité pour $c \in [\![j, k-1]\!]$:

$$\sum_{j \leqslant c < k} \big(m_{r+1,c+1} - m_{r+1,c}\big) \leqslant \sum_{j \leqslant c < k} \big(m_{r,c+1} - m_{r,c}\big)$$

Après télescopage, nous obtenons $m_{r+1,k} - m_{r+1,j} \leqslant m_{r,k} - m_{r,j}$ soit encore $m_{r+1,k} - m_{r,k} \leqslant m_{r+1,j} - m_{r,j}$. Sommons cette fois pour $r \in [\![i, \ell-1]\!]$:

$$\sum_{i \leqslant r < \ell} \big(m_{r+1,k} - m_{r,k}\big) \leqslant \sum_{i \leqslant r < \ell} \big(m_{r+1,j} - m_{r,j}\big)$$

Après un deuxième télescopage, nous obtenons $m_{\ell,k} - m_{i,k} \leqslant m_{\ell,j} - m_{i,j}$, soit enfin $m_{i,j} + m_{\ell,k} \leqslant m_{i,k} + m_{\ell,j}$.

• La réciproque est claire, en vertu de la règle «qui peut le plus peut le moins».

Question 20 • Nous exploitons le résultat de la question 19 pour obtenir une complexité $\mathcal{O}(np)$; en utilisant la définition, la complexité serait $\mathcal{O}(n^2p^2)$. La technique de programmation utilisée est classique : on visite l'ensemble des blocs carrés d'ordre 2 ; la rencontre d'un bloc qui ne satisfait pas la propriété de MONGE lève une exception, laquelle est récupérée par le `try .. with`.

```
let est_de_monge m =
 let n = vect_length m and p = vect_length m.(0) in
 try
  for i = 0 to n - 2 do
   for j = 0 to p - 2 do
    if m.(i).(j) + m.(i+1).(j+1) > m.(i).(j+1) + m.(i+1).(j)
      then failwith ""
   done
  done; true
 with _ -> false ;;
```

Question 21 • On trouve $\widehat{d} = \begin{pmatrix} 4 & 9 & 17 \\ 3 & 4 & 10 \end{pmatrix}$.

Question 22 • La première solution utilise la fonction `somme_bloc` rédigée à la question 3 ; sa complexité est $\mathcal{O}(n^2p^2)$, ce qui est déplorable !

```
let chapeau d =
 let n = vect_length d and p = vect_length d.(0) in
 let d' = make_matrix n p 0 in
  for i = 0 to n - 1 do
   for j = 0 to p - 1 do
    d'.(i).(j) <- somme_bloc d i 0 (n-1) j
   done
  done;
 d';;
```

Améliorons ceci : lorsque l'on calcule les $\widehat{d}_{i,j}$ avec la méthode naïve, on ne profite absolument pas des relations qui existent entre ces coefficients. Or $\widehat{d}_{i,j}$ est la somme des coefficients situés dans le bloc de d dont le coin supérieur droit est le coefficient $d_{i,j}$. Il est clair que $\widehat{d}_{n,1} = d_{n,1}$, $\widehat{d}_{n,j+1} = \widehat{d}_{n,j} + d_{n,j+1}$ pour $1 \leqslant j < p$ et $\widehat{d}_{i,1} = \widehat{d}_{i+1,1} + d_{i,1}$ pour $1 \leqslant i < n$. Ceci permet de remplir la dernière ligne et la première colonne de $\widehat{d}$. Ensuite, pour $1 \leqslant j < p$ et $1 \leqslant i < n$, on utilise la formule :

$$\widehat{d}_{i,j} = \widehat{d}_{i,j-1} + \widehat{d}_{i+1,j} - \widehat{d}_{i+1,j-1} + d_{i,j}$$

Notons bien que cette formule doit être utilisée dans un ordre précis : pour évaluer $\widehat{d}_{i,j}$, il faut disposer de $\widehat{d}_{i,j-1}$, $\widehat{d}_{i+1,j}$ et $\widehat{d}_{i+1,j-1}$. Dans le programme ci-dessous, nous décrivons chaque ligne de la gauche vers la droite, et les lignes successives sont traitées de bas en haut.

```
let chapeau_efficace d =
 let n = vect_length d and p = vect_length d.(0) in
 let d' = make_matrix n p 0 in
 d'.(n-1).(0) <- d.(n-1).(0);
 for j = 1 to p - 1 do
  d'.(n-1).(j) <- d'.(n-1).(j-1) + d.(n-1).(j)
 done;
 for i = n - 2 downto 0 do
  d'.(i).(0) <- d'.(i+1).(0) + d.(i).(0);
  for j = 1 to p - 1 do
   d'.(i).(j) <- d'.(i).(j-1) + d'.(i+1).(j) - d'.(i+1).(j-1) + d.(i).(j)
  done
 done;
 d';;
```

Question 23 • Il s'agit de montrer que la quantité $\varepsilon_{i,j} = \widehat{d}_{i,j+1} + \widehat{d}_{i+1,j} - \widehat{d}_{i+1,j+1} - \widehat{d}_{i,j}$ est positive ou nulle. Observons la façon dont les coefficients de d contribuent aux quatre sommes qui constituent cette quantité :

- les coefficients des lignes 1 à $i-1$ n'interviennent dans aucune somme, tout comme ceux des colonnes $j+2$ à p ;
- les coefficients situé dans le bloc dont le coin supérieur droit est $d_{i-1,j}$ interviennent deux fois affectés du signe «plus», et deux fois affectés du signe «moins» ;
- les coefficients $d_{i,1}$ à $d_{i,j}$ interviennent une fois affectés du signe «plus», et une fois affectés du signe «moins» ; il en est de même des coefficients $d_{i+1,j+1}$ à $d_{n,j+1}$
- enfin, $d_{i+1,j}$ intervient une fois, affecté du signe «plus».

Ceci montre que $\varepsilon_{i,j} = d_{i+1,j} \geqslant 0$ et termine la preuve.

Question 24 • Soient i, j, k et ℓ tels que $1 \leqslant i \leqslant \ell \leqslant n$ et $1 \leqslant j \leqslant k \leqslant p$; le quadrilatère $A_iA_\ell B_kB_j$ est convexe, donc la somme $\Delta_{i,k} + \Delta_{\ell,j}$ des longueurs de ses diagonales est au moins égale à la somme $\Delta_{i,j} + \Delta_{\ell,k}$ des longueurs de deux côtés opposés de ce quadrilatère. La figure 1 illustre cette propriété.

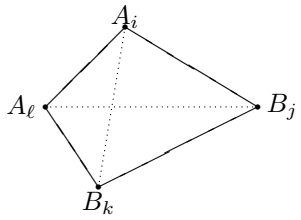

Figure 1: l'inégalité du quadrangle convexe

Question 25 • Chaque comparaison permet d'éliminer un élément de l'ensemble (le plus grand des deux en cas d'inégalité, n'importe lequel sinon) ; en partant de n éléments, il faudra donc en éliminer $n-1$ pour obtenir le minimum. Autre point de vue : considérons les n éléments comme les sommets d'un graphe ; chaque comparaison est une arête de ce graphe. S'il y a moins de $n-1$ arêtes, le graphe n'est pas connexe ; dans ce cas, deux éléments qui appartiennent à des composantes connexes distinctes n'ont pas été comparés (directement ou indirectement) et on ne sait donc pas quel est le plus petit.

Question 26 • Soit $i \in [\![1, n-1]\!]$. Notons $j = \gamma_i$, $k = \gamma_{i+1}$ et supposons $k < j$. Par définition, $m_{i,k} > m_{i,j}$ et $m_{i+1,k} \leqslant m_{i+1,j}$. Donc $m_{i,k} + m_{i+1,j} > m_{i,j} + m_{i+1,k}$ ce qui contredit le fait que m possède la propriété de Monge. Donc $j \leqslant k$, soit $\gamma_i \leqslant \gamma_{i+1}$: la suite $(\gamma_i)_{1\leqslant i\leqslant n}$ est croissante.

Question 27 • Soit $n \geqslant 3$. Notons $i = \left\lceil \dfrac{n}{2} \right\rceil$; on peut alors déterminer $j = \gamma_i$ au prix de $p-1$ comparaisons. Exploitons le résultat précédent :

- si $\ell \in [\![1, i-1]\!]$, alors γ_ℓ est dans le bloc dont le coin inférieur droit est $m_{i,j}$; ce bloc compte i lignes et j colonnes ;
- en revanche, si $\ell \in [\![i+1, n]\!]$, alors γ_ℓ est dans le bloc dont le coin supérieur gauche est $m_{i,j}$; ce bloc compte $n-i+1$ lignes et $p-j+1$ colonnes.

Remarquons que $n-i+1 = \left\lceil \dfrac{n+1}{2} \right\rceil$. Notons $C(n,p)$ le coût, dans le pire des cas, de la construction de la suite $(\gamma_i)_{1\leqslant i\leqslant n}$ associée à une matrice de Monge à n lignes et p colonnes. Le découpage précédent n'est intéressant que si $n \geqslant 3$; pour $n \leqslant 2$, nous avons $C(n,p) = n(p-1)$; pour $n \geqslant 3$, nous obtenons l'équation de récurrence suivante :

$$C(n,p) = p - 1 + \max_{1\leqslant j\leqslant p} \left(C\Big(\left\lceil \frac{n}{2} \right\rceil, j\Big) + C\Big(\left\lceil \frac{n+1}{2} \right\rceil, p-j+1\Big)\right)$$

Observons que $\dfrac{C(n,p)}{p-1}$ ne dépend que de n : en effet, cette propriété est vraie pour les conditions initiales, et est conservée par l'équation de récurrence. Notons donc $\varphi(n)$ cette quantité ; $\varphi(1) = 1$, $\varphi(2) = 2$ et φ est manifestement croissante, si bien que $\varphi(n) = 1 + \varphi\Big(\left\lceil \dfrac{n+1}{2} \right\rceil\Big)$. Dressons un tableau donnant les premières valeurs de $\varphi(n)$:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	2	3	4	4	5	5	5	5	6	6	6	6	6	6	6	6	7	7	7

Ne tenons pas compte des cas particuliers $n = 1$ et $n = 2$. On constate qu'ensuite les longueurs des «paliers» sont les puissances successives de 2, les changements de palier se faisant pour $n = 4, 6, 10, 18$, soit $n = 2^k + 2$. $\varphi(3) = \varphi(2^0 + 2) = 3$ suggère la formule $\varphi(n) = 3 + \lambda(n)$, où $\lambda(n) = \lfloor \lg(n-2) \rfloor$. Prouvons-la par récurrence, pour $n \geqslant 3$. Supposons cette relation acquise jusqu'au rang n inclus. Notons que $\lambda(2n) = \lambda(2n+1) = 1 + \lambda(n)$, et que $n \geqslant 3$ implique $\left\lfloor \frac{n+2}{2} \right\rfloor \leqslant \frac{n+2}{2} < \frac{n+n}{2} = n$. Alors :

$$\varphi(n+1) = 1 + \varphi\Big(\Big\lceil \frac{n+2}{2} \Big\rceil\Big) = 1 + 3 + \lambda\Big(\Big\lceil \frac{n+2}{2} - 2 \Big\rceil\Big)\Big) = 4 + \lambda\Big(\Big\lceil \frac{n-2}{2} \Big\rceil\Big)\Big)$$

À ce stade, nous distinguerons deux cas de figure selon la parité de n :

$$\begin{aligned} n = 2p \quad &\Rightarrow \quad \varphi(n+1) = 4 + \lambda\Big(\Big\lceil \frac{2p-2}{2} \Big\rceil\Big)\Big) = 4 + \lambda(p-1) = 3 + \lambda(2p-2) = 3 + \lambda(n-2) \\ n = 2p+1 \quad &\Rightarrow \quad \varphi(n+1) = 4 + \lambda\Big(\Big\lceil \frac{2p-1}{2} \Big\rceil\Big)\Big) = 4 + \lambda(p) = 3 + \lambda(2p) = 3 + \lambda(2p+1) = 3 + \lambda(n) \end{aligned}$$

Concluons : $C(n,p) = (p-1)\lambda(n)$, où $\lambda(n) = \mathcal{O}(\lg n)$. Donc $C(n,p) = \mathcal{O}(p \lg n)$.

▶ Voici deux autres exemples de mise en œuvre de la technique employée à la question 27 :

- l'algorithme linéaire de recherche du k-ième élément d'un ensemble ; voir [2] ou [3] ;
- l'algorithme proposé par Hirschberg, pour la détermination d'un plus long sous-mot commun à deux mots donnés ; voir [5] (volume 2, chapitre 8), ou [4] (problème 10).

Références

[1] Éric Badouel, Stéphane Boucheron, Anne Dicky, Antoine Petit, Miklos Santha, Pascal Weil et Marc Zeitoun. Problèmes d'informatique fondamentale. SCOPOS-Springer, 2001.

[2] Sara Baase et Allen Van Gelder. Computer Algorithms. Addison-Wesley, 2000.

[3] Thomas H. Cormen, Charles E. Leiserson et Ronald L. Rivest. Introduction to Algorithms. The MIT Press, 1990.

[4] Bruno Petazzoni. 16 problèmes d'informatique. SCOPOS-Springer, 2001.

[5] Grzegorz Rozenberg et Arto Salomaa (éd.). Handbook of Formal Languages. Springer, 1997.

FIN