

Eléments de correction du TD n° 3

Exercice 1.

C'est une procédure, car il faut retourner deux résultats. Voici une solution qui retourne le couple `Rgmin` et `Rg2min`, respectivement les indices (ou rangs) du minimum, et du second minimum des valeurs supposées toutes distinctes du tableau `T[1..N]`.

```
type tabNomb: tableau 1..beaucoup d'entiers;
procédure Bimini (ref T: tabNomb; val N: entier; ref Rgmin, Rg2min :
entiers);
var I: entier;
début
Si T[1] < T[2] // l'égalité est impossible,
  alors Rgmin:= 1; Rg2min:= 2
  sinon Rgmin:= 2; Rg2min:= 1;
finSi;
pour I:= 3 à N faire
  Si T[I] < T[Rgmin]
    alors Rg2min:= Rgmin; Rgmin:= I
    sinon // alors T[Rgmin] < T[I]
      Si T[I] < T[Rg2min]
        alors Rg2min:= I
      finSi
  finSi
finPour
```

Preuve.

Terminaison. Évidente (boucle pour).

Validité partielle. La preuve se fonde sur la remarque que le minimum de $T[1..I]$ est égal au minimum de la paire $\{\text{minimum de } \{T[1..I-1]\}, T[I]\}$, d'une part, et que le second minimum de $T[1..I]$ est égal au second minimum du triplet de 3 nombres distinctes suivant, $\{\text{minimum de } T[1..I-1], \text{second minimum de } T[1..I-1], T[I]\}$.

Soit $\mathcal{B}(I)$ l'invariant de boucle, c'est-à-dire l'affirmation que `Rgmin` pointe sur le minimum, et `Rg2min` sur le second minimum de $T[1..I-1]$. Au premier passage dans la boucle pour, ($I = 3$), $\mathcal{B}(I)$, évalué en tête de boucle, juste avant le test d'arrêt, est vrai.

S'il est vrai au dernier passage, juste avant de sortir, $I = N + 1$ et le résultat de la procédure est juste.

Il suffit donc de vérifier qu'il est invariant pour chaque tour de boucle.

Il y a 3 cas à envisager.

1er cas : $T[I] > T[Rg2min]$. Alors on a $\mathcal{B} \wedge \{T[Rg2min] < T[I]\} \Rightarrow \mathcal{B}(I + 1)$.

2ème cas : $T[Rgmin] < T[I] < T[Rg2min]$. Puisque $Rg2min := I$ le prédicat $\mathcal{B}(I + 1)$ est encore vrai.

3ème cas : $T[I] < T[Rgmin] < T[Rg2min]$. Là encore, après l'affectation, le prédicat $\mathcal{B}(I + 1)$ est vrai.

Après l'incrémentation $I:=I+1$ on retrouve le prédicat $\mathcal{B}(I)$, donc ce dernier est bien un invariant.

Si on enlève la condition que les éléments sont tous distincts, l'algorithme est insuffisamment spécifié. Il peut y avoir plus de deux valeurs minimales égales, et dans ce cas il faut préciser lesquelles choisir (les plus à gauche dans le tableau?). Tel qu'il est, l'algorithme ci-dessus donne les plus à gauche.

D'autre part on peut vouloir "les deux plus petites valeurs distinctes les plus à gauche". La seconde n'existe pas toujours. Dans ce cas on peut initialiser $Rgmin$ à 1, et $Rg2min$ à 0, et modifier un peu la séquence de la boucle pour mettre à jour correctement $Rgmin$ et $Rg2min$.

On obtient,

```
procédure Bimini2 (ref T: tabNomb; val N: entier; ref Rgmin, Rg2min :
entiers);
var I:entier;
début
Rgmin:= 1; Rg2min:=0; // la seconde valeur minimale distincte peut ne pas exister.
pour I:= 2 à N faire
  Si T[I] < T[Rgmin]
    alors Rg2min:= Rgmin; Rgmin:= I
    sinon Si T[I] > T[Rgmin]
      alors
        Si Rg2min = 0 ou (T[I] < T[Rg2min]) alors Rg2min:=I
        finSi
      finSi
    finSi
  finSi
finPour
```

Exercice 2.

```
fonction CNP (val N,P: entier): entier;
début
Si P = N ou P = 0 alors retour (1)
    sinon retour ( CNP(N-1,P-1) + CNP(N-1,P) )
finSi
finProc;
```

Complexité. Soit $\mathcal{T}^+(N, P)$ le coût d'un appel de la fonction CNP.

Expérimentalement on trouve pour les petites valeurs de N et P , $\mathcal{T}^+(N, P) = CNP(N, P) - 1$.

D'autre part on a la récurrence évidente, en comptant le nombre d'additions effectuées,

$$\mathcal{T}^+(N, 0) = \mathcal{T}^+(N, P) = 0,$$

$$\mathcal{T}^+(N, P) = \mathcal{T}^+(N - 1, P) + \mathcal{T}^+(N - 1, P - 1) + 1,$$

récurrence très voisine de celle de $C(N, P)$, à l'initialisation près.

En fait on peut établir formellement la formule $\mathcal{T}^+(N, P) = CNP(N, P) - 1$ par récurrence sur N :

base : c'est vrai pour (1,1). récurrence : supposons la relation vraie pour tout couple d'entiers N' et P' tels que $0 \leq P' \leq N' < N$. Considérons le couple (N, P) tel que $0 \leq P \leq N$. Si

$P = 0$ ou $P = N$, on a $\mathcal{T}^+(N, P) = 0 = C(N, P) - 1$. Si $0 < P < N$ les couples $(N - 1, P)$ et $(N - 1, P - 1)$ sont tels que $0 \leq P' \leq N' < N$, donc on peut écrire,

$\mathcal{T}^+(N - 1, P) = C(N - 1, P) - 1$, et $\mathcal{T}^+(N - 1, P - 1) = C(N - 1, P - 1) - 1$.

Donc $\mathcal{T}^+(N, P) = C(N - 1, P) - 1 + C(N - 1, P - 1) - 1 + 1 = C(N, P)$.

Pour évaluer la complexité, on peut prendre sa valeur maximale, par exemple pour N pair on trouve, à l'aide de la formule de Stirling,

$$C_{\max}^+(N) = \sqrt{\frac{2}{\pi N}} \times 2^N$$

On peut également calculer la valeur moyenne car la somme en ligne des binomiaux donne $(1 + 1)^n = 2^n$. On trouve dans ce cas,

$$C_{\text{moy}}^+(N) = \frac{1}{N + 1} 2^N$$

. Dans tous les cas la complexité est exponentielle.

Le triangle de Pascal est une matrice d'entier $P[0..Nmax; 0..Nmax]$, telle que $P[N, M] = C[N, M]$, avec $P[N, P] = 0$ pour $P > N$. La procédure suivante calcule le tableau jusqu'à la ligne N :

```

procédure TriangPas(ref T: TrianglePascal; val N: entier);
val I, J: entier;
début
T[0,0] := 1;
pour I:=1 à N faire
    T[N,0]:=1; T[N,N]:= 1;
    pour J:= 1 à N-1 faire T[I,J]:= T[I-1,J] + T[I-1,J-1] finPour
finPour
finProc;

```

Cet algorithme a une complexité *triangulaire* (déjà rencontrée pour le tri par sélection), donc quadratique, c'est-à-dire $\Theta(N^2)$, et un encombrement mémoire également quadratique.

Il est possible d'économiser en place tout en restant cependant quadratique en temps, si l'on ne veut que la dernière ligne :

```

type LignePascal = tableau 0..Nmax d'entiers;
procédure LignPas (ref T: LignePascal; val N : entier);
val I, J :entier;
début
T[0] := 1;
pour I:=1 à N faire
    T[1]:=1;T[I]:=1;
    pour J:=I-1 à 1 (pas -1) faire T[J]:= T[J]+ T[J-1] finPour
finPour
finProc;

```

Exercice 3 :

```

procédure CalculRang (ref T: TabNomb; ref Rang : TabEntier; val

```

```

N:entier);
var I,J, Rm: entier;
début
// Initialisation du tableau Rang;
pour I:=1 à N faire Rang[I]:= I finPour;
//Tri
pour I:=1 à N-1 faire
  Rm:= I;
  pour J:= I+1 à N faire
    Si T[Rang[J]] < T[Rang[Rm]] alors Rm:= J finSi
  finPour
  Si Rm <> I alors Echanger (Rang[I], Rang[Rm]);
finPour
finProc;

```

Exercice 4.

```

fonction RanInserDicho (ref T: tabNomb; val N: entier; val X: Nombre): entier;
début // on suppose N >= 1
Si X < T[1] alors retour (0)
  sinon si X >= T[N] alors retour(N)
    sinon // obligatoirement N >= 2 et T[1] <= X < T[N]
      retour(RIDRec (T,1,N,X))
    finSi
  finSi
finFonc;
fonction RIDRec( ref T: tabNomb; val D,F: entier; val X: Nombre): entier;
var M:entier;
début
Si D+ 1 = F alors retour (D) //on s'arrête sur un tableau à 2 éléments.
  sinon M:= (D+F)div 2
  Si X < T[M] alors F:= M sinon D:= M finSi;
  retour (RIDRec(T,D,F,X))
finSi

```

Preuve :

Prouvons uniquement la fonction récursive, par récurrence sur la longueur $L = F - D + 1$ du tableau. La preuve de la fonction itérative de lancement s'en déduit aussitôt. Cette preuve utilise le fait que $D + 1 < F \Rightarrow D < M < F$ inégalités strictes .

a- Terminaison. Si $L = 2$ la fonction termine (pas de nouvel appel récursif lancé).

Soit $L > 2$ et supposons que la fonction termine pour tout appel avec une longueur L' tq $2 \leq L' < L$. Comme $L > 2 \Rightarrow D < M < F$ (inégalités strictes), l'appel en dernière ligne se fait avec une longueur L' inférieure à L donc termine (hypothèse de récurrence), de même que l'appel qui l'a lancé.

b- Validité Partielle. Par récurrence sur L . Appelons K le résultat retourné et $\mathcal{P}(L)$ le prédicat suivant :

$$\mathcal{P}(L) = \{T[D] \leq X < T[F] \Rightarrow T[K] \leq X < T[K + 1]\}.$$

base : $\mathcal{P}(2)$ est vrai, car alors le résultat retourné est D et $D = K$.

réurrence : supposons $\mathcal{P}(L')$ vrai $\forall L'$ tq $2 \leq L' < L$.

Si $\{X < T[M]\}$ alors

$$\{T[D] \leq X < T[F]\} \text{ et } \{X < T[M]\} \Rightarrow \{T[D] \leq X < T[M]\}.$$

. L'appel suivant se fait sur un tableau de longueur $L' = M - D + 1 < L$, donc $\mathcal{P}(L')$ est vrai (hypothèse de récurrence). La valeur retournée vérifie $T[K] \leq X < T[K + 1]$ donc $\mathcal{P}(L)$ est vrai. L'autre cas est symétrique.

Complexité.

Comptons le nombre de comparaisons $X < T[M]$. Il y en a autant que d'appels. Or ce nombre est de l'ordre de $\log N$. Plus précisément on vérifie qu'il est $\lceil \log N \rceil$, valeur atteinte lorsque N est impair et que l'on poursuit systématiquement la recherche dans la moitié droite du sous-tableau. Donc $\mathcal{C}^<(N) = \Theta(\log N)$.

Exercice 5.

La procédure de fusion de 2 portions consécutives du tableau $T[D..M]$ et $T[M+1..F]$ nécessite clairement une mémoire auxiliaire supplémentaire : en effet supposons que $T[D..M] = [5, 8, \dots]$ et $T[M+1..F] = [7, \dots]$, si l'on fusionne sur place, le 7 de la partie droite va écraser le 8 de la partie gauche.

. On peut mettre cette déclaration de variable auxiliaire partout, sauf dans une procédure récursive, par exemple dans une procédure d'appel non récursive, qui appelle le premier appel récursif (solution présentée en cours), ou dans la procédure fusion elle-même.

Dans cette seconde solution, il n'est alors pas nécessaire de transmettre le tableau auxiliaire en paramètre.

```
procédure FUSION3(ref T,D,M);
var Taux, tabClé;
début
I1:=D; I2:=M+1; J:=D-1;
tant que I1<= M et I2 <= F faire
    sI t[I J:=J+1; Taux[J
```