X Informatique MP/PC 2005 — Corrigé

Ce corrigé est proposé par Jean-Baptiste Rouquier (ENS Lyon) ; il a été relu par Walter Appel (Professeur en CPGE) et Vincent Puyhaubert (Professeur en CPGE).

Cette épreuve est réservée aux candidats de la filière MP n'ayant pas choisi l'option informatique. Elle n'est comptabilisée que pour l'admission et est commune avec la filière PC.

Le thème abordé est une étude très simplifiée d'un problème de géométrie algorithmique : étant donné un ensemble d'objets en trois dimensions, déterminer le contour perçu par un observateur. En particulier, certains objets peuvent être partiellement cachés.

Toutes les questions demandent d'écrire du code. Nous l'avons rédigé en Maple car ce langage est le plus connu dans les filières MP et PC, même s'il est plus destiné au calcul formel qu'à la programmation. D'autres langages sont autorisés pour cette épreuve, parmi lesquels figurent Caml, C, C++, Java et Pascal. Les fonctions écrites n'utilisant que la syntaxe de base, vous pourrez sans peine les réécrire dans votre langage préféré.

Le sujet est assez difficile et parfois les questions ne sont pas suffisamment détaillées. De plus, il y a peu de questions et les fonctions à écrire sont longues. Il faut porter une attention particulière aux indices, à l'initialisation des variables, aux conditions d'arrêt et à la mise à jour des variables.

- La première partie définit ce qu'est une ligne d'horizon et en construit une de façon naïve puis un peu plus évoluée.
- La seconde manipule les lignes d'horizon de deux manières : mise sous forme canonique et fusion de deux lignes.

Ce corrigé introduit les fonctions locales, qui ne sont pas indispensables pour faire cette épreuve mais sont pratiques et permettent de gagner du temps.

Ce sujet est progressif et constitue une bonne préparation au concours, mais au vu de sa difficulté il est préférable de ne l'aborder qu'une fois le cours maîtrisé et après avoir pratiqué la programmation.

Indications

Pour chacune des questions, sauf la première, il y a plusieurs indications (une par phrase). Nous vous conseillons de tenter à nouveau de résoudre la question avant de toutes les lire.

1 Remplir la matrice E avec des `0` puis, pour chaque immeuble (donc avec une boucle `for`), mettre des `1` dans toutes les cases couvertes par cet immeuble.

2 Pour chaque abscisse x, chercher la hauteur y de la ligne d'horizon à cet endroit et écrire x et y dans le tableau résultat. Comme indiqué par l'énoncé, x est écrit dans la case `2*x` du tableau, y dans la case suivante.

3 Distinguer quatre cas :

- On est à l'intérieur d'un immeuble, et l'on monte jusqu'à en sortir, c'est-à-dire jusqu'à rencontrer une case contenant 0.
- On est sur un immeuble, et l'on avance (vers la droite) jusqu'à rencontrer soit un mur qui monte (c'est-à-dire que l'on est dans une case contenant 1), soit la fin du toit (c'est-à-dire que la case en dessous contient 0).
- On est au sol et l'on avance jusqu'à rencontrer un mur qui monte.
- On descend le long d'un mur, jusqu'à soit rencontrer le sol ($y = 0$), soit rencontrer un toit.

Créer une variable `direction` qui contient le cas courant et faire une boucle `while` dans laquelle, suivant la direction courante, on met à jour x et y, on change éventuellement de direction, et l'on écrit éventuellement x et y dans le tableau résultat.

4 Tenir à jour la position x dans le tableau *hor* et k dans le tableau résultat. Utiliser une boucle « `while hor[2*x] < N do ...` ». Recopier deux cases de *hor* dans le tableau résultat chaque fois que la ligne d'horizon change de hauteur.

5 Créer toutes les variables locales proposées par l'énoncé : $i, j, k, x, y, u, u', v, v'$, et les tenir à jour (chaque fois que l'on modifie i ou j). Distinguer les cas $x < y$, $x > y$ (qui est symétrique) et $x = y$. Ne pas s'attacher (au moins dans un premier temps) à renvoyer une ligne d'horizon sous forme canonique. On peut utiliser la fonction suivante qui renvoie la longueur d'un tableau :

```
longueur := proc(tableau)
local bounds;
  bounds := op(2,eval(tableau));
  op(2,bounds) - op(1,bounds) +1;
end;
```

I. Détermination de la ligne d'horizon

> On suppose dans ce corrigé que la matrice E est également une variable globale. On trouvera à la fin de ce corrigé un exemple d'utilisation de ces fonctions, qui commence par définir toutes les variables globales nécessaires.

1 On commence par remplir la matrice E avec des 0, puis, pour chaque immeuble i défini par $g[i]$, $d[i]$ et $h[i]$, on « colorie » cet immeuble dans E avec des 1. C'est la fonction `trace_rect` qui se charge de tracer un immeuble.

> C'est l'occasion de montrer que l'on peut définir une fonction *locale* (ici `trace_rect`) à l'intérieur de la fonction `remplir`. De même que la variable `i` n'existe pas à l'extérieur de la fonction `remplir` (alors que `E` si), la fonction `trace_rect` n'est utilisable que dans la fonction `remplir`. Une fonction locale n'était pas indispensable ici, mais cette construction sera très utile par la suite.

```
remplir := proc()
global g,d,h,n,M,N,E;
local i,x,y,trace_rect;
#                       On remplit E avec des 0:
  for x from 0 to N-1 do
    for y from 0 to M-1 do
      E[x,y] := 0;
    od;
  od;
#                       On définit la procédure locale trace_rect:
  trace_rect := proc (i)
    global E;
    local x,y;
      for x from g[i] to d[i] -1 do
        for y from 0 to h[i] -1 do
          E[x,y] := 1
      od;
    od;
    end;
#                       On l'utilise pour remplir les immeubles:
  for i from 0 to n-1 do trace_rect(i) od;
end;
```

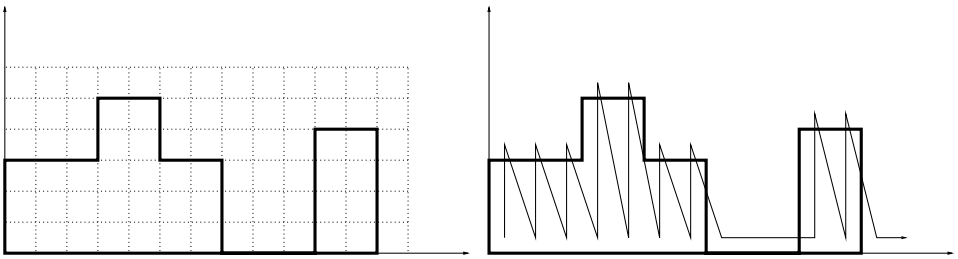
> Pour toute fonction (`remplir` comme `trace_rect`) il faut préciser à Maple les variables globales utilisées à l'aide du mot clé `global`.

2 L'idée est simple : pour chaque abscisse x, on cherche la hauteur y de la ligne d'horizon à cet endroit, et l'on écrit x et y dans le tableau résultat.

On commence donc par créer un tableau `resultat` de longueur $2N + 1$ comme l'indique l'énoncé. Trouver la hauteur de la ligne d'horizon se fait en partant de $y = 0$ et en augmentant y jusqu'à ce que la case (x, y) contienne 0 ; c'est une boucle `while`. Il suffit alors d'écrire x dans la case $2x$ du tableau, et y dans la case suivante.

> L'énoncé indique en effet que x est dans la case $2x$. Rappelons qu'une ligne d'horizon est stockée dans un tableau de la manière suivante :
>
> $$(x_1, y_1, x_2, y_2, x_3, y_3 \dots)$$
>
> L'horizon est alors constitué de l'ensemble des segments horizontaux de la forme $[(x_i, y_i); (x_{i+1}, y_i)]$.
>
> Cette notation permet d'utiliser peu de place pour stocker une ligne d'horizon lorsque `N` est grand devant le nombre de segments de la ligne. Comme la complexité des fonctions des questions 4 et 5 est linéaire en la longueur du tableau représentant la ligne d'horizon, ceci représente un gain de temps par rapport à des tableaux de longueur linéaire en `N`. Ce gain de place et de temps est rendu possible par la fonction `canonique` de la question 4.
>
> Ce que l'énoncé appelle « sentinelle » est une marque que l'on met dans la dernière case contenant des données. Ici cette marque est N, qui est connue car N est une variable globale. Le tableau peut avoir plus de cases, mais il ne faut pas les lire : la sentinelle indique la fin de la partie intéressante. Cet artifice est par exemple utilisé pour les chaînes de caractères en C : le dernier caractère est nul.

La figure suivante montre un exemple d'immeubles et le chemin parcouru par la fonction `horizon1` :



```
horizon1 := proc()
global E,N;
local resultat,x,y;
  resultat := array(0..2*N); #tableau de longueur l = 2N+1 éléments
  for x from 0 to N-1 do
    y := 0;
    while E[x,y] = 1 do y := y+1 od;
    resultat[2*x] := x; resultat[2*x+1] := y;
  od;
  resultat[2*N] := N;
  resultat
end;
```

Cette fonction examine au plus une fois chacune des cases de E, sa complexité est donc linéaire en MN.