Épreuve d'informatique de l'X - 2005 - MP/PC

Compte tenu de l'implémentation des tableaux en Maple, on introduit la fonction `longueur` suivante valable pour toutes les questions :

```
1  longueur := proc(a :: array)
2      return(nops(convert(a, list));
3  end;
```

Question 1 : On commence par créer une fonction auxiliaire `EstIntérieur` qui détermine si le point de coordonnées (i, j) appartient à l'un des bâtiments. Le point de coordonnées (i, j) appartient au k-ième bâtiment si et seulement si $g[k] \leqslant j + 0,5 \leqslant d[k]$ et $i + 0,5 \leqslant h[k]$. Dans la fonction `remplir`, on parcourt tout le tableau. On regarde si le point (i, j) appartient à l'un des bâtiments et si c'est le cas, on met un 1 à l'emplacement correspondant dans le tableau E. La complexité est linéaire en nMN.

```
1  EstIntérieur := proc(i :: integer, j :: integer)
2    local res, taille, k;
3    res := false;
4    taille := longueur(g);
5    for k from 0 to taille - 1 do
6      # on test si le point est dans le k-ième batiment
7      if j+0.5 > g[k] and j+0.5 < d[k] and i+0.5 < h[k]
8           then res := true fi
9    od;
10     return(res);
11   end;
12
13   remplir := proc()
14   local res, i, j;
15   global E, M, N;
16   for i from 0 to M-1 do
17      for j from 0 to N-1 do
18         if EstIntérieur(i,j)
19              then E[i,j] := 1 else E[i,j] := 0 fi;
20      od;
21   od;
22   return();
23   end;
```

Question 2 : Dans la fonction `horizon1`, pour chaque abscisse j, on détermine où passe la ligne d'horizon en repérant le premier i pour lequel $E(i, j) = 0$ (c'est l'objet de la boucle *while* de la ligne 7).

```
1  horizon1 := proc()
2    local i, j, horizon, haut;
3    horizon := array(0..2*N);
4    for j from 0 to N-1 do
5        horizon[2*j] := j; # on stocke l'abscisse j
6        haut := 0;
7        while E[haut,j] <> 0 do haut := haut+1 od;
8        horizon[2*j+1] := haut;
9    od;
10   horizon[2*N] := N;
11   return(horizon);
12   end;
```

Question 3 : Dans la fonction `horizon2`, pour chaque abscisse j, on détermine où passe la ligne d'horizon en repérant le premier i pour lequel $E(i, j) = 0$ (c'est l'objet de la boucle *while* des lignes 8-9 ou des lignes 10-11). Au lieu de partir de l'ordonnée 0, on part de la hauteur de la ligne d'horizon à l'abscisse $(j - 1)$ et on monte ou descend à partir de cette position suivant les cas. Cela revient bien à suivre la ligne d'horizon, ce qui nous permet de remplir le tableau `horizon` en $O(L)$.

```
1  horizon2 := proc()
2    local j, haut, horizon;
3    horizon := array(0..2*N);
4    haut := 0;
5    for j from 0 to N-1 do
6       horizon[2*j] := j;
7       if E[haut,j] = 1
8           then while E[haut,j] = 1 and haut < M
9                      do haut := haut+1 od;
10          else while haut > 0 and E[haut-1,j] = 0
11                     do haut := haut-1 od;
12      fi;
13      horizon[2*j+1] := haut;
14   od;
15   horizon[2*N] := N;
16   return(horizon);
17   end;
```

Question 4 : Dans la boucle des lignes 4 à 6, on détermine la longueur du tableau résultat en déterminant le nombre de variations de hauteur dans la ligne d'horizon donnée par le tableau `hor`. Dans la boucle des lignes 11 à 15, on remplit le tableau en repérant de nouveau les variations de hauteur dans la ligne d'horizon. On parcourt deux fois le tableau `hor` ce qui nous donne bien une complexité en temps linéaire par rapport à la longueur ℓ du tableau `hor`.

```
1  canonique := proc(hor :: array)
2    local i, k, long, res;
3    long := 2;
4    for i from 0 to floor((longueur(hor)-5)/2) do
5       if hor[2*i+1]<>hor[2*i+3] then long := long+2 fi;
6    od;
7  # on crée un tableau résultat de la bonne longueur
8    res := array(0..long);
9    res[0] := 0; res[1] := hor[1];
10   k := 1;
11   for i from 0 to (longueur(hor)-5)/2 do
12      if hor[2*i+1]<>hor[2*i+3]
13           then res[2*k]   := hor[2*(i+1)];
14                res[2*k+1] := hor[2*i+3]; k:=k+1 fi;
15   od;
16   res[long] := N;
17   return(res);
18 end;
```

Question 5 : Dans la fonction `fusion`, on commence par créer un tableau de longueur $\ell_1 + \ell_2 - 1$ (et non $\ell_1 + \ell_2$ comme dans l'énoncé). On parcourt progressivement et simultanément les tableaux hor_1 et hor_2 ainsi que le tableau résultat res à l'aide des indices i, j, k. Les variables `h1` et `h2` permettent de mémoriser la hauteur des lignes d'horizon associées aux tableaux `hor1` et `hor2` à l'abscisse `res[k]`. On fait progresser les indices i, j et k comme cela est suggéré dans l'énoncé. Cela revient à fusionner les abscisses des tableaux hor_1 et hor_2 en conservant l'ordre croissant et en ajustant correctement la hauteur correspondant à cet abscisse. On peut remplacer la ligne 18 par `return(canonique(res))` si en sortie on veut une ligne d'horizon sous forme canonique.

```
1  fusion := proc(hor1 :: array, hor2 :: array)
2     local l1, l2, i, j, k, res, h1, h2;
3     i := 0; j := 0; k := 0;
4     l1 := longueur(hor1); l2 := longueur(hor2);
5     res := array(0..l1+l2-2);
6     h1 := hor1[1]; h2 := hor2[1];
7     while k < l1+l2-3 do
8       if (i<l1) and (hor1[i] <= hor2[j])
9           then res[k] := hor1[i]; h1 := hor1[i+1];
10               res[k+1] := max(h1,h2);
11               i:=i+2; k:=k+2;
12          else res[k] := hor2[j]; h2 := hor2[j+1];
13               res[k+1] := max(h1,h2);
14               j:=j+2; k:=k+2;
15      fi;
16    od;
17    res[k]:=hor1[l1-1];
18    return(res);
19 end;
```