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

## Lignes d'horizon

**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  def EstIntérieur(i, j):
2      res = False
3      taille = len(g)
4      for k in range(taille):
5      # on test si le point est dans le k-ième batiment
6          if j+0.5 > g[k] and j+0.5 < d[k] and i+0.5 < h[k]:
7              res = True
8      return(res);
9
10 def remplir():
11     global E, M, N;
12     for i in range(M):
13         for j in range(N):
14             if EstIntérieur(i,j):
15                 E[i,j] = 1
16             else :
17                 E[i,j] = 0
```

(PSI*)

**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* des lignes 7 et 8).

```
1  def horizon1():
2      global E, M, N;
3      horizon = (2*N+1)*[0]
4      for j in range(N):
5          horizon[2*j] = j # on stocke l'abscisse j
6          haut = 0
7          while E[haut,j] != 0:
8              haut = haut+1
9          horizon[2*j+1] = haut
10     horizon[2*N] = N
11     return(horizon)
```

**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 11-12). 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 on 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  def horizon2():
2      global E, M, N;
3      horizon = (2*N+1)*[0]
4      haut = 0
5      for j in range(N):
6          horizon[2*j] = j
7          if E[haut,j] == 1:
8              while E[haut,j] == 1 and haut < M:
9                  haut = haut+1
10         else :
11             while haut > 0 and E[haut-1,j] == 0:
12                 haut = haut-1
13         horizon[2*j+1] = haut
14     horizon[2*N] = N;
15     return(horizon)
```

**Question 4** : Dans la boucle des lignes 4 à 6, on détermine la longueur du tableau résultat en déterminant le nombre de variations des hauteurs 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 $\ell$ du tableau `hor`.

```
1  def canonique(hor):
2      N = len(hor)
3      long = 2
4      for i in range(int((N-3)/2)):
5          if hor[2*i+1] != hor[2*i+3]:
6              long = long+2
7      # on crée un tableau résultat de la bonne longueur
8      res = (long+1)*[0]
9      res[0] =  0
10     res[1] = hor[1]
11     k = 1
12     for i in range((len(hor)-3)//2):
13         if hor[2*i+1] != hor[2*i+3]:
14             res[2*k]   = hor[2*(i+1)]
15             res[2*k+1] = hor[2*i+3]
16             k = k+1
17     res[long] = hor[N-1]
18     return(res);
```

**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 23 par `return(canonique(res))` si en sortie on veut une ligne d'horizon sous forme canonique.

```
1  def fusion(hor1, hor2):
2      i = 0
3      j = 0
4      k = 0
5      l1 = len(hor1)
6      l2 = len(hor2)
7      res = (l1+l2-1)*[0]
8      h1 = hor1[1]
9      h2 = hor2[1]
10     while k < l1+l2-3 :
11         if (i<l1-1) and (hor1[i] <= hor2[j]):
12             res[k] = hor1[i]
13             h1 = hor1[i+1];
14             res[k+1] = max(h1,h2);
15             i = i+2
16         else :
17             res[k] = hor2[j]
18             h2 = hor2[j+1];
19             res[k+1] = max(h1,h2);
20             j = j+2
21         k = k+2;
22     res[k] = hor1[l1-1]
23     return(res)
```