Correction du DM

Nicolas Gast

J'ai choisi pour ce sujet de représenter les variales E et horizon par des variables globales.

Question 1

La question 1 ne présente pas de difficulté particulière. La méthode la plus simple possible consiste à remplir un tableau M*N avec des 0 puis à ajouter des '1' aux bons endroits pour chaque batiment.

On peut remarquer que la condition $e_{j,k} = 1$ est équivalent à l'existance d'un batiment i tel que le batiment commence à gauche de j, ($g[i] \leq j$), finisse à droite (strictement) de j ($j < d[i]$) et dont la hauteur soit au moins $h[i] + 1$.

```
remplir := proc(M,N, g,d,h)
  local i,j,k;
  global E;
  for i from 0 to N-1 do
    for j from 1 to M-1 do
      E[i][j] = 0;
    od;
  od;
  for i from 1 to nops(g) do
    for j from g[i] to d[i]-1 do
      for k from 0 to h[i]-1 do
        E[j][k] = 1;
   od;od;od;
end;
```

Question 2

Pour calculer la ligne d'horizon, on peut parcourir chaque colonne dans l'ordre.

```
horizon1 := proc( M,N, E)
   local k,h; global hor;
   for k from 0 to N-1 do
     hor[2*k] := k;
     hor[2*k+1] := 0;
     for h from 0 to M-1 do
       if E[k][h] = 1 then
         hor[2*k+1] := h+1;
       fi;
     od;
   od;
   hor[2*N] := N;
end:
```

Question 3

Justification du nombre d'opération

```
horizon2 := proc(N,M,E)
   local k,h; global hor;
   h := 0;
   for k from 0 to N-1 do
     hor[2*k] := k;
     hor[2*k+1] := h;
```

```
      if E[k][h] = 0
       then
        while h > 0 and E[k][h-1] = 0 do
            h := h-1;
        od;
       else
        while h < M and E[k][h] = 1 do
            h := h+1;
        od;
      fi;
      hor[2*k+1] := h;
     od;
   hor[2*N] := N;
end
```

Question 4

L'énoncé n'est pas très clair dans la définition de la ligne d'horizon. Une définition logique pour la ligne d'horizon est de supposer que $\forall k$ `hor[2*k]` $<$ `hor[2*k+2]` (ou réciproquement $\forall k$ `hor[2*k]` $>$ `hor[2*k+2]`) Si ce n'est pas le cas, il peut y avoir un motif de la forme $(\ldots, 3, x, 5, y, 4, \ldots)$ mais dans ce cas nécessairement $x = y$ (car la partie entre 4 et 5 a une hauteur de x et de y).

Il ne faut donc pas commencer par trier le tableau comme ont pu le faire certains. Si un tri était nécessaire, la question serait impossible. En effet : un tri a une complexité d'au moins $n\log(n)$ ce qui n'est pas linéaire.

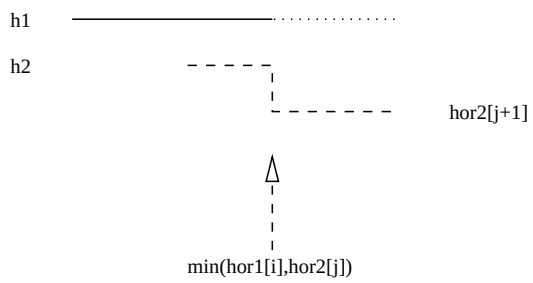
```
canonique := proc( hor)
  local ho,h, i,res;
  # La première partie consiste à remettre dnas l'ordre pour que ho[2*k] < ho[2*k+2]
  # elle est laissée en exercice.
  ho := hor;

  # maintenant : ho[2*k] < ho[2*k+2]
  res := ho[1], ho[2]; h := ho[2];
  for i from 1 to nops(ho)/2-1 do
      if ho[2*i+2] <> h
       then
          res := res,ho[2*i+1],ho[2*i+2];
          h := ho[2*i+2];
      fi;
  od;
  res,ho[nops(hor)];
end:
```

Question 5

Cette question est la plus difficile de l'énoncé. Elle est faite pour occuper les élèves brillants et départager les candidats. L'idée de l'algorithme est de parcourir les deux listes hor1 et hor2 en parallèle et de fusionner les deux listes au fur et à mesure.

Le problème est que les listes ne sont pas nécessairement de la même taille, ainsi si on veut fusionner l'horizon de l'énoncé et [0,1,7,0,11], on est obligé de commencer par parcourir la liste de l'énoncé puis l'autre puis de retourner sur la première.

C'est pour celà qu'on choisit à chaque fois `min(hor1[i], hor2[i])`. L'étape à l'intérieur du `while` est décrite ci dessous :



À chaque étape, `h1` (resp `h2`) désigne la hauteur du dernier batiment rencontré dans `hor1` (resp `hor2`). On détermine ensuite quel est la prochaine limite de batiment qui arrive. Dans l'exemple ci contre, c'est `hor2[j]` qui arrive avant. La hauteur du prochain batiment est donc soit `h1` dans le cas où la hauteur du prochain batiment de `hor2` est $<$ `h1` car soit le nouveau batiment est plus petit comme dans l'exemple (et la hauteur de l'horizon reste donc `h1`), soit le batiment est plus grand est la hauteur devient `hor2[j+1]`.
À la fin de la boucle, on ajoute à `res` la nouvelle horizon et on modifie `i` ou `j` afin de faire progresser l'algorithme.

```
fusion := proc( hor1,hor2)
  local i,j,h1,h2,res;
  h1,h2 := hor1[2], hor2[2];
  res := 0, max(h1,h2);
  i := 3; j := 3;

  while i <= nops(hor1) and j <= nops(hor2) and not(i=nops(hor1) and j =nops(hor2)) do
      if hor1[i] < hor2[j] then
        if h2 < hor1[i+1]
          then res := res,hor1[i],hor1[i+1];
          else res := res, hor1[i],h2;
        fi;
        h1 := hor1[i+1];
        i := i+2;
      else
        if h1 < hor2[j+1]
          then res := res,hor2[j],hor2[j+1];
          else res := res, hor2[j],h1;
        fi;
        h2 := hor2[j+1];
        j := j+2;
     fi;
   od;
  canonique( [res, hor1[nops(hor1)] ]);
end:
```