Épreuve d'informatique de l'X - 2013 - PT/PSI

Partie I. Planter le paysage

Question 1. Pour remplir le tableau `hauteurs`, on utilise la propriété $h_0 = 0$ ce qui se traduit `hauteurs[0] := 0.` de la ligne 4 et la propriété $h_i = h_{i-1} + \texttt{deniveles}[i]$ pour $i \in [\![1, n]\!]$ qui se traduit par la boucle des lignes 5 à 7.

```
1  calculHauteurs := proc(n::posint)
2      global hauteurs, deniveles;
3      local i;
4      hauteurs[0] := 0.;
5      for i from 1 to n do
6          hauteurs[i] := hauteurs[i-1] + deniveles[i];
7      od;
8      return()
9      end;
```

Question 2. On suppose le tableau `hauteurs` rempli par l'appel de la fonction `calculHauteurs`. On balaye le tableau des hauteurs pour trouver les hauteurs minimales et maximales ainsi que leurs indices. On commence par supposer que ces points sont en position 0 et donc que les altitudes correspondantes sont nulles (lignes 4 à 7). Dans la boucle des lignes 8 à 16, on parcourt le tableau des hauteurs. Si on trouve un point strictement plus bas que les précédents, on enregistre sa hauteur et sa position dans les variables globales `hMin` et `iMin` (ligne 11) et si on trouve un point strictement plus haut que les précédents, on enregistre sa hauteur et sa position dans les variables globales `hMax` et `iMax` (ligne 14).

```
1  calculFenetre := proc(n::posint)
2      global hMin, hMax, iMin, iMax, hauteurs;
3      local i, hauteur;
4      hMin := hauteurs[0];
5      hMax := hauteurs[0];
6      iMin := 0;
7      iMax := 0;
```

(PSI*)

```
8      for i from 1 to n do
9          hauteur := hauteurs[i];
10         if hauteur < hMin
11             then hMin := hauteur; iMin := i;
12         fi;
13         if hauteur > hMax
14             then hMax := hauteur; iMax := i;
15         fi;
16     od;
17
18     return();
19 end;
```

Question 3 : La fonction `distanceAuSol` calcule la longueur de la ligne brisée du point P_i au point P_j. On applique le théorème de Pythagore pour calculer la distance du point P_{k-1} au point P_k, cette distance étant égale à $\sqrt{1^2 + \texttt{denivele}[k]^2}$. Les lignes 4 à 7 permettent d'initialiser les variables I et J pour parcourir les points du paysage en ordre croissant dans la boucle des lignes 8 à 10.

```
1  distanceAuSol := proc(i,j :: posint)
2      global deniveles;
3      local dist := 0, k, I, J;
4      if i <= j
5          then I := i; J := j
6          else I := j; J := i
7      fi;
8      for k from I+1 to J do
9          dist := dist + sqrt(1^2+deniveles[k]^2);
10     od;
11     return(dist)
12     end;
```

Question 4 : Dans la procédure `longueurDuPlusLongBassin`, on parcourt le tableau grâce à la boucle des lignes 6 à 14. À la ligne 7, on regarde si le point P_i est un pic. Si c'est le cas, on calcule la distance au sol entre P_i et le dernier point remarquable. Si cette distance est supérieure à la plus longue distance de bassin de la partie étudiée, on actualise la variable `longueurBassinMax` (ligne 10). Dans tous les cas, on modifie la variable `debut` qui repère le premier point du bassin à l'étude (ligne 12). Les lignes 15 à 17 permettent d'étudier le dernier bassin situé, entre le dernier pic et le point remarquable de droite.

```
1  longueurDuPlusLongBassin := proc(n)
2    global hauteurs;
3    local i, debut, longueurBassinMax ;
4    debut := 0;
5    longueurBassinMax := 0;
6    for i from 1 to n-1 do
7       if hauteurs[i] > max(hauteurs[i-1], hauteurs[i+1])
8          then
9             if distanceAuSol(debut,i) > longueurBassinMax
10               then longueurBassinMax := distanceAuSol(debut,i)
11            fi;
12            debut := i
13      fi;
14   od;
15   if distanceAuSol(debut,n) > longueurBassinMax
16      then longueurBassinMax := distanceAuSol(debut,n)
17   fi;
18   return(longueurBassinMax)
19 end;
```

Partie II. Planter les poteaux

Question 5 : On initialise `beta` avec la valeur $\beta_{i,j}$ de l'énoncé (on a simplifié les « ℓ » dans l'expression de l'énoncé). Puis, grâce à la boucle des lignes 8 à 11 (ou la boucle des lignes 13 à 16), on parcourt le tableau des hauteurs. Pour chaque k de l'intervalle $[\![i+1, j-1]\!]$, on calcule $\alpha_{i,k}$ que l'on met dans la variable `alpha`. Le branchement conditionnel de la ligne 6 permet de différencier les cas $i \leqslant j$ et $i > j$. En sortie de boucle, la variable `res` vaut `true` si et seulement si $i = j$ ou bien $\beta_{i,j} \geqslant \max\limits_{i<k<j} \alpha_{i,k}$, lorsque $j > i$; ou bien $\beta_{i,j} \leqslant \min\limits_{j<k<i} \alpha_{i,k}$, lorsque $j < i$.

```
1  estDeltaAuDessusDuSol := proc(i,j,l,delta)
2     global hauteurs;
3     local alpha, beta, k, res;
4     res := true;
5     beta := (hauteurs[j]-hauteurs[i])/(j-i);
6     if j > i
7        then
8           for k from i+1 to j-1 do
9              alpha := (hauteurs[k]+delta-hauteurs[i]-l)/(k-i);
10             res := res and (beta >= alpha)
11          od
12       else
13          for k from j+1 to i-1 do
14             alpha := (hauteurs[k]+delta-hauteurs[i]-l)/(k-i);
15             res := res and (beta <= alpha)
16          od;
17    fi;
18    return(res)
19 end;
```

Question 6 : À la ligne 4, on « plante » un premier poteau en P_0. La variable `DernierPoteauPlante` garde en mémoire la position du dernier poteau « planté ». La boucle des lignes 5 à 11 permet de parcourir tous les points différents des extrémités. Si la ligne reliant le dernier poteau planté au poteau en P_k ne respecte pas la législation, c'est que le point P_{k-1} est le dernier point pour lequel la législation est satisfaite. On « plante » un poteau au point P_{k-1} en modifiant en conséquence le tableau `poteaux` (ligne 9) et on actualise la variable `DernierPoteauPlante` (ligne 8) et la variable `nb_poteaux` qui compte le nombre de poteaux plantés (ligne 7). Les lignes 12 à 14, traite le cas du poteau planté au point P_n.

```
1  placementGloutonEnAvant := proc(n,l,delta)
2     global poteaux;
3     local nb_poteaux := 1, k, DernierPoteauPlante := 0;
4     poteaux[1]:= 0;
5     for k from 1 to n do
6        if not(estDeltaAuDessusDuSol(DernierPoteauPlante,k,l,delta))
7           then nb_poteaux := nb_poteaux+1;
8                DernierPoteauPlante := k - 1;
9                poteaux[nb_poteaux] := DernierPoteauPlante;
10       fi;
11    od;
12    nb_poteaux := nb_poteaux + 1;
13    poteaux[0] := nb_poteaux;
14    poteaux[nb_poteaux] := n;
15    return()
16 end;
```

Question 7 : La complexité de `estDeltaAuDessusDuSol(pos,k,l,delta)` est en $O(k - pos + 1)$ donc au maximal en $O(k)$. La complexité de la boucle des lignes 5 à 11 est donc en $O(\sum_{k=1}^{n-1} k) = O(n^2)$. Les autres opérations se faisant à coût constant, la complexité de la fonction `placementGloutonEnAvant` est quadratique (en $O(n^2)$). La complexité quadratique vient de l'appel de la fonction `estDeltaAuDessusDuSol` dans la boucle des lignes 5 à 11. Pour améliorer l'algorithme, on peut se contenter de ne calculer qu'une seule fois les pentes que l'on peut stocker dans un tableau ou que l'on calcule une seule fois à chaque passage dans la boucle. Il suffit de garder en mémoire le maximum des pentes entre les points $P_{DernierPoteauPlante}$ et P_k. Il faudrait donc modifier la boucle des lignes 5 à 11 en supprimant l'appel de la fonction `estDeltaAuDessusDuSol` et introduire une variable `MaxPente` qui mémorise la pente maximale entre le dernier poteau planté et la position courante étudiée (d'indice k).

Question 8 : À la ligne 4, on commence par planter un poteau en position 0. La variable `DernierPoteauPlante` permet de repérer le dernier poteau planté. La variable `nb_poteaux` donne le nombre de poteaux plantés. Dans la boucle conditionnelle des lignes 5 à 13, tant que la valeur de `DernierPoteauPlante` n'est pas égale à n, on essaye de planter un poteau le plus à droite possible. Si les poteaux en position `DernierPoteauPlante` et `suivant` respectent la législation, on plante un poteau (lignes 7 à 10), on actualise la variable `DernierPoteauPlante` et on réinitialise la variable `suivant` à n pour repartir de la droite. Si le poteau en position `suivant` ne convient pas, on essaye son voisin de gauche (ligne 11). En sortie de boucle, on stocke le nombre de poteaux plantés dans la case `poteaux[0]` (ligne 14).

```
1  placementgloutonAuPlusLoin := proc(n,l,delta)
2      global poteaux;
3      local nb_poteaux := 1, suivant := n, DernierPoteauPlante := 0;
4      poteaux[1]:= 0;
5      while DernierPoteauPlante <> n do
6          if estDeltaAuDessusDuSol(DernierPoteauPlante,suivant,l,delta)
7             then nb_poteaux := nb_poteaux+1;
8                  DernierPoteauPlante := suivant;
9                  suivant := n;
10                 poteaux[nb_poteaux] := DernierPoteauPlante;
11            else suivant := suivant - 1
12         fi;
13     od;
14     poteaux[0] := nb_poteaux;
15     return();
16 end;
```

Partie III. Minimiser la longueur du fil

Question 9 : La plus petite longueur de fil pour relier le point P_0 à lui-même est évidemment nulle, d'où `optL[0]` $= 0$. Pour relier P_0 au point P_i, il faut considérer tous les points P_j pour j compris entre 0 et $i-1$ qui peuvent être reliés à P_i en respectant la législation. Pour un tel point P_i, la longueur minimale pour relier P_0 à P_i avec P_j comme position de l'avant dernier poteau, est égale à la longueur minimale pour relier P_0 à P_j augmenté de la longueur de fil pour relier P_j à P_i ce qui nous donne $L_{i,j} +$ `optL`$[j]$. Parmi toutes ces longueurs, il faut trouver la plus petite pour trouver la longueur minimale de fil à utiliser pour relier P_0 à P_i. En déduit la formule de l'énoncé :

`optL`$[i] = \min\{L_{i,j} +$ `optL`$[j],$ où $0 \leqslant j < i$ et il est possible de tirer un fil de P_j à $P_i\}$.

Question 10 : On commence par écrire une fonction `distance(i,j)` qui calcule la distance en ligne droite entre les points P_i et P_j (donc la longueur de fil nécessaire pour relier les poteaux en position P_i et P_j).

```
1  distance := proc(i,j)
2      global hauteurs;
3      local dx, dy;
4      dx := j-i;
5      dy := hauteurs[j]-hauteurs[i];
6      return(sqrt(dx*dx+dy*dy))
7  end;
```

La fonction `longueurMinimale` remplit le tableau `optL` et renvoie la valeur `optL[n]` qui correspond à la longueur minimale de fil pour relier le bord gauche au bord droit. La boucle des lignes 5 à 15 permet de remplir le tableau `optL` en appliquant la formule donnée par l'énoncé. La boucle des lignes 7 à 13 permet de calculer le minimum de l'ensemble $\{L_{i,j} +$ `optL`$[j]$ où $0 \leqslant j < 0$ et il est possible de tiré un fil de P_j à $P_i\}$. On est sûr de pouvoir tirer un fil entre les points P_i et P_{i-1} ce qui explique l'initialisation de la distance minimale à la ligne 6.

```
1  longueurMinimale := proc(n,l,delta)
2      global optL;
3      local distance_minimum, distance_i_j, i, j;
4      optL[0] := 0;
5      for i from 1 to n do
6          distance_minimum := optL[i-1] + distance(i,i-1);
7          for j from 0 to i-2 do
8              distance_i_j := optL[j] + distance(j,i);
9              if estDeltaAuDessusDuSol(j,i,l,delta)
10                 and distance_minimum > distance_i_j
11                     then distance_minimum := distance_i_j;
12             fi;
13         od;
14         optL[i] := distance_minimum;
15     od;
16     return(optL[n])
17 end;
```

Question 11 : On garde la même structure de programme qu'à la question précédente en rajoutant deux lignes permettant de remplir le tableau `precOptL`. À la ligne 6, on part de l'hypothèse que le poteau à gauche du poteau au point P_i est planté au point P_{i-1}. À chaque fois que l'on trouve un meilleur emplacement, on met à jour la case `precOptL` en ligne 13.

```
1  longueurMinimale := proc(n,l,delta)
2      global optL, precOptL;
3      local distance_minimum, distance_i_j, i, j;
4      optL[0] := 0;
5      for i from 1 to n do
6          precOptL[i] := i-1;
7          distance_minimum := optL[i-1] + distance(i,i-1);
8          for j from 0 to i-2 do
9              distance_i_j := optL[j] + distance(j,i);
10             if estDeltaAuDessusDuSol(j,i,l,delta)
11                and distance_minimum >= distance_i_j
12                    then distance_minimum := distance_i_j;
13                         precOptL[i] := j
14             fi;
15         od;
16         optL[i] := distance_minimum;
17     od;
18     return(optL[n]) end;
```

Question 12 : La boucle conditionnelle des lignes 6 à 9 permet de compter les poteaux nécessaires. On part du dernier poteau et tant que l'on a pas atteint le bord gauche, on considère le poteau précédent (ligne 7). En sortie de boucle, on dispose du nombre de poteaux à planter, on peut remplir le tableau `poteaux`. On initialise le tableau `poteaux` en remplissant les cases d'indice 0 et $nbPoteaux$. Puis, par la boucle des lignes 12 à 14, on remplit le tableau en partant de la fin en passant d'un poteau au poteau précédent en suivant le même principe que dans la boucle conditionnelle précédente.

```
1  placement := proc()
2      global precOptL, poteaux;
3      local nbPoteaux, DernierPoteauPlante, i;
4      nbPoteaux:=1;
5      DernierPoteauPlante := n;
6      while DernierPoteauPlante <> 0 do
7          DernierPoteauPlante := precOptL[DernierPoteauPlante];
8          nbPoteaux := nbPoteaux + 1;
9      od;
10     poteaux[0] := nbPoteaux;
11     poteaux[nbPoteaux] := n;
12     for i from nbPoteaux-1 to 1 by -1 do
13         poteaux[i] := precOptL[poteaux[i+1]];
14     od;
15     return()
16 end;
```

Le tableau « `precOptL` » étant supposé rempli, la complexité de la fonction `placement` est en $O(n)$. En effet, dans la boucle `while` des lignes 6 à 9, on visite au plus une fois chaque case du tableau et pour chaque case on effectue deux opérations d'affectation. Dans la boucle itérative des lignes 12 à 13, on effectue une affectation et le nombre d'opérations est égal au nombre de poteaux plantés ce qui n'excède pas $(n+1)$. Le pire des cas est obtenu quand on est obligé de planter un poteau par point.

Le remplissage du tableau « `precOptL` » est réalisé par la fonction `longueurMinimale` de la question 11. En prenant en compte la complexité de la fonction `estDeltaAuDessusDuSol` qui est en $O(|i-j|)$, en comptant le nombre d'affectations, on trouve la complexité suivante :

$$\sum_{i=1}^{n} 3 + \sum_{i=2}^{n} \sum_{j=0}^{i-2} [3 + \alpha.(i-j)] = O(n^3)$$