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 par `hauteurs[0] = 0.` en ligne 3 et la propriété $h_i = h_{i-1} + \texttt{deniveles}[i]$ pour $i \in [\![1, n]\!]$ qui se traduit par la boucle des lignes 4 à 5.

```
1  def calculHauteurs(n):
2      global hauteurs, deniveles;
3      hauteurs[0] = 0.;
4      for i in range(1,n+1):
5          hauteurs[i] = hauteurs[i-1] + deniveles[i]
6      return()
```

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 3 à 6). Dans la boucle des lignes 7 à 12, 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 10) 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 12).

```
1  def calculFenetre(n):
2      global hMin, hMax, iMin, iMax, hauteurs;
3      hMin = hauteurs[0]
4      hMax = hauteurs[0]
5      iMin = 0
6      iMax = 0
7      for i in range(1,n+1):
8          hauteur = hauteurs[i]
9          if hauteur < hMin:
10             hMin = hauteur; iMin = i
11         if hauteur > hMax:
12             hMax = hauteur; iMax = i
13     return()
```

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 9.

```
1  def distanceAuSol(i,j):
2      global deniveles
3      dist = 0
4      if i <= j :
5          I = i; J = j
6      else :
7          I = j; J = i
8      for k in range(I+1,J+1) :
9         dist = dist + sqrt(100**2+deniveles[k]**2)
10     return(dist)
```

Question 4 : Dans la procédure `longueurDuPlusLongBassin`, on parcourt le tableau grâce à la boucle des lignes 5 à 9. À la ligne 6, 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 8). Dans tous les cas, on modifie la variable `debut` qui repère le premier point du bassin à l'étude (ligne 9). Les lignes 10 à 11 permettent d'étudier le dernier bassin situé, entre le dernier pic et le point remarquable de droite.

```
1  def longueurDuPlusLongBassin(n):
2    global hauteurs
3    debut = 0
4    longueurBassinMax = 0
5    for i in range(1,n-1):
6        if hauteurs[i] > max(hauteurs[i-1], hauteurs[i+1]):
7            if i - debut > longueurBassinMax:
8                longueurBassinMax = i - debut
9            debut = i
10   if n - debut > longueurBassinMax:
11       longueurBassinMax = n - debut
12   return(longueurBassinMax)
```

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 6 à 8 (ou la boucle des lignes 10 à 12), 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 5 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  def estDeltaAuDessusDuSol(i,j,l,delta):
2      global hauteurs
3      res = True
4      beta = (hauteurs[j]-hauteurs[i])/(j-i)
5      if j>i :
6          for k in range(i+1,j):
7              alpha = (hauteurs[k]+delta-hauteurs[i]-l)/float(k-i);
8              res = res and (beta >= alpha)
9      else:
10         for k in range(j+1,i):
11             alpha = (hauteurs[k]+delta-hauteurs[i]-l)/float(k-i);
12             res = res and (beta <= alpha)
13     return(res)
```

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 6 à 10 permet de parcourir tous les points différents des extrémités. Si le segment de droite 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 10) et on actualise la variable `DernierPoteauPlante` (ligne 9) et la variable `nb_poteaux` qui compte le nombre de poteaux plantés (ligne 8). Les lignes 11 à 13, traite le cas du poteau planté au point P_n.

```
1  def placementGloutonEnAvant(n,l,delta):
2      global poteaux
3      nb_poteaux = 1
4      poteaux[1]= 0
5      DernierPoteauPlante = 0
6      for k in range(1,n+1):
7          if not(estDeltaAuDessusDuSol(DernierPoteauPlante,k,l,delta)):
8              nb_poteaux = nb_poteaux+1
9              DernierPoteauPlante = k - 1
10             poteaux[nb_poteaux] = DernierPoteauPlante
11     nb_poteaux = nb_poteaux + 1
12     poteaux[0] = nb_poteaux
13     poteaux[nb_poteaux] = n
14     return()
```

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 6 à 10 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 $O(n^2)$). Le complexité quadratique vient de l'appel de la fonction `estDeltaAuDessusDuSol` dans la boucle des lignes 6 à 10. 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 6 à 10 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 3, 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 7 à 14, 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 9 à 12), 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 14). En sortie de boucle, on stocke le nombre de poteaux plantés dans la case `poteaux[0]` (ligne 15).

```
def placementgloutonAuPlusLoin(n,l,delta):
    global poteaux
    poteaux[1] = 0
    DernierPoteauPlante = 0
    nb_poteaux = 1
    suivant = n
    while DernierPoteauPlante != n:
        if estDeltaAuDessusDuSol(DernierPoteauPlante,suivant,l,delta):
            nb_poteaux = nb_poteaux+1
            DernierPoteauPlante = suivant
            suivant = n
            poteaux[nb_poteaux] = DernierPoteauPlante
        else:
            suivant = suivant - 1
    poteaux[0] = nb_poteaux
    poteaux[nb_poteaux] = n
    return()
```

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).

```
def distance(i,j) :
    global hauteurs;
    dx = j-i
    dy = hauteurs[j]-hauteurs[i]
    return(np.sqrt(dx*dx+dy*dy))
```

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 4 à 11 permet de remplir le tableau `optL` en appliquant la formule donnée par l'énoncé. La boucle des lignes 6 à 10 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 7.

```
def longueurMinimale(n,l,delta):
    global optL
    optL[0] = 0
    for i in range(1,n+1) :
        distance_min= optL[i-1] + distance(i,i-1)
        for j in range(i-1,-1,-1):
            distance_i_j = optL[j] + distance(i,j)
            if estDeltaAuDessusDuSol(j,i,l,delta) \
                        and distance_min>= distance_i_j:
                distance_min= distance_i_j
        optL[i] = distance_min
    return()
```

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 5, 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 12.

```
def longueurMinimale(n,l,delta):
    global optL, precOptL
    optL[0] = 0
    for i in range(1,n+1) :
        precOptL[i] = i-1
        distance_minimum = optL[i-1] + distanceAuSol(i,i-1)
        for j in range(i-2,-1,-1) :
            distance_i_j = optL[j] + distanceAuSol(i,i-1)
            if estDeltaAuDessusDuSol(j,i,l,delta)  \
                        and distance_minimum >= distance_i_j :
                distance_minimum = distance_i_j;
                precOptL[i] = j
        optL[i] = distance_minimum;
    return()
```

Question 12 : La boucle conditionnelle des lignes 5 à 7 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 6). 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 10 à 11, 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  def placement() :
2      global precOptL, poteaux
3      nbPoteaux=1
4      pos = n
5      while pos != 0 :
6          pos = precOptL[pos]
7          nbPoteaux = nbPoteaux + 1
8      poteaux[0] = nbPoteaux
9      poteaux[nbPoteaux] = n
10     for i in range(nbPoteaux-1,0,-1):
11         poteaux[i] = precOptL[poteaux[i+1]]
12     return()
```

Le tableau « `precOptL` » étant supposé rempli, la complexité de la fonction `placement` est en $O(n)$. En effet, dans la boucle `while` des lignes 5 à 7, 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 10 à 11, 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 :

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