

Épreuve d'Informatique 4

Correction

Exercice 1 (E3A 2013)

1. Deux fonctions possibles en $O(N^2)$:

```
1 def listeSommesPartielles1(L):  
2     return [sum(L[:k+1]) for k in range(len(L))]
```

ou en $O(N)$ (donc mieux) :

```
1 def listeSommesPartielles2(L):  
2     if len(L)==0:  
3         return []  
4     resultat=[L[0]]  
5     for k in range(1, len(L)):  
6         resultat.append(resultat[k-1]+L[k])  
7     return resultat
```

2. On utilise la fonction `max` :

```
1 def listeSommesPartielles1(L):  
2     return [sum(L[:k+1]) for k in range(len(L))]
```

ou on la recode (attention, `max` est un mot réservé du langage, ne pas l'utiliser comme nom de variable) :

```
1 def maxSommesPartiellesMain(L):  
2     if len(L)==0:  
3         return 'Erreur'  
4     SP=listeSommesPartielles2(L)  
5     maximum=SP[0]  
6     for element in SP[1:]:  
7         if element > maximum:  
8             maximum = element  
9     return maximum
```

3. Cet algorithme ne prétend pas être optimal.

On considère les listes de la forme $L[k_1:N]$ lorsque $k_1 \in \llbracket 1, N \rrbracket$. Pour chacune de ces listes, on cherche la somme partielle maximale (donc $\sum_{i=k_1}^k \ell_i$ maximale lorsque k bouge). Puis on stocke les N résultats dans une liste, dont on prend le maximum.

```

1 def maxSommres(L):
2     if len(L)==0:
3         return 'Erreur'
4     return max([maxSommresPartielles(L[k1:]) for k1 in range(len(L))])

```

Exercice 2 (Poteaux télégraphiques – corrigé UPS)

Partie I. Planter le paysage

Question 1. Pour créer et remplir le tableau `hauteurs`, on utilise la propriété $h_0 = 0$ ce qui se traduit par `hauteurs = [0.]` en ligne 2 et la propriété $h_i = h_{i-1} + \text{deniveles}[i]$ pour $i \in \llbracket 1, n \rrbracket$ qui se traduit par la boucle des lignes 3 à 4.

```

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

```

On pouvait aussi utiliser la fonction de l'exercice 1.

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 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 (hMin, hMax, iMin, iMax)

```

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 $\sqrt{1^2 + \text{denivele}[k]^2}$. 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 + (1**2+deniveles[k]**2)**(1/2.)
10    return dist

```

Question 4.1 : Dans la procédure `pointRemarquable(i, n)` (où l'on avait besoin de $n...$), on commence par tester si on est au bord (ligne 4). Sinon, on teste ligne 6 si les deux points P_{i-1} et P_{i+1} qui entourent P_i sont plus bas que P_i .

```

1 def pointRemarquable(i, n):
2     global hauteurs
3     res = False
4     if (i == 0) or (i == n):
5         res = True
6     elif hauteurs[i] > max(hauteurs[i-1], hauteurs[i+1]):
7         res = True
8     return res

```

On pouvait aussi tout tester d'une seule phrase logique, grace à l'évaluation paresseuse :

```

1 def pointRemarquable2(i, n):
2     global hauteurs
3     return (i == 0) or (i == n) or (hauteurs[i] > max(hauteurs[i-1], hauteurs[i+1]))

```

Si $i = 0$ ou $i = n$, python ne va pas plus loin (à cause du `or`), donc la formule appelant `i-1` et `i+1` ne provoque pas une erreur.

Question 4.2 : Dans la procédure `longueurDuPlusLongBassin`, on parcourt le tableau grâce à la boucle des lignes 5 à 10. 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 9). Dans tous les cas, on modifie la variable `debut` qui repère le premier point du bassin à l'étude (ligne 10). Les lignes 12 à 13 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             longueur = distanceAuSol(debut, i)
8             if longueur > longueurBassinMax:
9                 longueurBassinMax = longueur
10            debut = i
11    longueur = distanceAuSol(debut, n)
12    if longueur > longueurBassinMax:
13        longueurBassinMax = longueur
14    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 $\llbracket i+1, j-1 \rrbracket$, 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 $i \leq j$ et $i > j$. En sortie de boucle, la variable `res` vaut `true` si et seulement si $i = j$ ou bien $\beta_{i,j} \geq \max_{i < k < j} \alpha_{i,k}$, lorsque $j > i$; ou bien $\beta_{i,j} \leq \min_{j < k < i} \alpha_{i,k}$, 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]-1)/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]-1)/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 None

```

(On pouvait aussi écrire une fonction qui retourne une liste (locale) `poteaux`).

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 $O(n^2)$)

La 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 $P_{\text{DernierPoteauPlante}}$ 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).