Épreuve d'informatique de l'X - 2015 - MP/PC

Enveloppes convexes dans le plan

Partie I. Préliminaires

Question 1 On parcourt le tableau des points dans la boucle des lignes 4 à 8. Le tableau `sol` contient les coordonnées du point le plus bas parmi les points explorés et j son indice. Si dans l'exécution de la boucle on trouve un point plus bas, on modifie en conséquence les variables `sol` et j (lignes 7 à 9).

```
1  def plusBas(tab,n):
2      sol = np.array([tab[0,0],tab[0,1]])
3      j = 0
4      for i in range(1,n):
5          if tab[1,i] < sol[1]
6          or (tab[1,i] == sol[1] and tab[0,i] < sol[0]):
7              j = i
8              sol[0] = tab[0,i]
9              sol[1] = tab[1,i]
10     return(j)
```

Question 2 Sur le tableau exemple, le test d'orientation pour les choix d'indices $i = 0$, $j = 3$ et $k = 4$ est positif (car $\begin{vmatrix} 4-0 & 4-0 \\ 1-0 & 4-0 \end{vmatrix} = \begin{vmatrix} 4 & 4 \\ 1 & 4 \end{vmatrix} = 16 - 4 = 12 > 0$) et il est négatif pour les choix d'indices $i = 8$, $j = 9$ et $k = 10$ (car $\begin{vmatrix} 8-7 & 11-7 \\ 5-2 & 6-2 \end{vmatrix} = \begin{vmatrix} 1 & 4 \\ 3 & 4 \end{vmatrix} = 4 - 12 = -8 < 0$).

(PSI*)

Question 3 Si on note (x_1, y_1), (x_2, y_2) et (x_3, y_3) les coordonnées respectives des points p_i, p_j et p_k, on calcule le déterminant suivant :

$$\begin{vmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{vmatrix} = (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1).$$

On retourne 1, 0 ou -1 suivant le signe ou la nullité de ce déterminant.

```
1  def orient(tab,i,j,k):
2      x1,y1 = tab[:,i]
3      x2,y2 = tab[:,j]
4      x3,y3 = tab[:,k]
5      det = (x2-x1)*(y3-y1)-(y2-y1)*(x3-x1)
6      if det == 0:
7          return(0)
8      elif det > 0:
9          return(1)
10     else :
11         return(-1)
```

Partie II. Algorithme du paquet cadeau

Question 4

On utilise les propriétés du déterminant :

- (réflexivité) $\forall j \neq i,\ \texttt{orient}(tab, i, j, j) = 0 \Rightarrow p_j \preceq p_j$.
- (antisymétrie) $\forall j \neq i,\ \forall k \neq i,\ p_j \preceq p_k$ et $p_k \preceq p_j$
  $\Rightarrow \texttt{orient}(tab, i, j, k) \leqslant 0$ et $\texttt{orient}(tab, i, k, j) = -\texttt{orient}(tab, i, j, k) \leqslant 0$
  $\Rightarrow \texttt{orient}(tab, i, j, k) = 0 \Rightarrow p_i, p_j, p_k$ sont alignés et donc, d'après les hypothèses, comme $j \neq i$ et $k \neq i$, on a nécessairement $j = k$.
- (transitivité) $\forall j \neq i,\ \forall k \neq i,\ \forall l \neq i,\ p_j \preceq p_k$ et $p_k \preceq p_l$
  $\Rightarrow \texttt{orient}(tab, i, j, k) \leqslant 0$ et $\texttt{orient}(tab, i, k, l) \leqslant 0$. Or
  $$\begin{vmatrix} x_j - x_i & x_l - x_i \\ y_j - y_i & y_l - y_i \end{vmatrix} = \begin{vmatrix} x_j - x_i & x_l - x_k + x_k - x_i \\ y_j - y_i & y_l - y_k + y_k - y_i \end{vmatrix}$$
  $$= \underbrace{\begin{vmatrix} x_j - x_i & x_l - x_k \\ y_j - y_i & y_l - y_k \end{vmatrix}}_{\leqslant 0} + \underbrace{\begin{vmatrix} x_j - x_i & x_k - x_i \\ y_j - y_i & y_k - y_i \end{vmatrix}}_{\leqslant 0} \leqslant 0.$$
- (totalité) $\forall j \neq i,\ \forall k \neq i,\ \texttt{orient}(tab, i, j, k) \leqslant 0$ ou $\texttt{orient}(tab, i, j, k) > 0$
  $\Rightarrow \texttt{orient}(tab, i, j, k) \leqslant 0$ ou $\texttt{orient}(tab, i, k, j) = -\texttt{orient}(tab, i, j, k) < 0$
  $\Rightarrow p_j \preceq p_k$ ou $p_k \preceq p_j$

Question 5 On commence par initialiser j à 0 (ligne 2). Puis dans la boucle des lignes 3 à 5, si on trouve un point $p_k \neq p_i$ vérifiant $p_k \preceq p_j$, on actualise la valeur de j. En sortie de boucle, j est l'indice du point cherché. Si $i = 0$, j va prendre la valeur 1 pour $k = 1$ dans la boucle.

```
1 def prochainPoint(tab, n, i):
2     j = 0
3     for k in range(1,n):
4         if k != i and (orient(tab,i,j,k) <= 0):
5             j = k
6     return(j)
```

Question 6 L'indice j change de valeur pour $k = 1$, $k = 2$ et $k = 5$ car $p_0 \preceq p_1$, $p_1 \preceq p_2$, $p_2 \preceq p_5$.

valeur de k	valeur de j en sortie de boucle
1	1
2	2
3	2
4	2
5	5
6	5
7	5
8	5
9	5
10	5
11	5

Question 7 : On commence par déterminer le point le plus bas (ligne 2) et son successeur (ligne 3). On initialise la liste des indices des sommets du bord (ligne 4). Dans la boucle conditionnelle des lignes 5 à 7, tant que le point suivant est différent du point le plus bas, on l'ajoute à la liste des points du bord et on calcule son successeur. On retourne la liste des indices des points du bord en ligne 8.

```
1 def convJarvis(tab,n):
2     iplusBas = plusBas(tab,n)
3     suivant = prochainPoint(tab, n, iplusBas)
4     l = [iplusBas]
5     while suivant != iplusBas :
6         l.append(suivant)
7         suivant = prochainPoint(tab, n, suivant)
8     return(l)
```

Question 8 On commence par déterminer le point le plus bas avec la fonction `plusBas` qui a une complexité en $O(n)$ (un parcours du tableau des points).
Pour chaque point de l'enveloppe convexe, on parcourt entièrement le tableau des points pour déterminer le point suivant. Comme il y a m points dans l'enveloppe convexe et n points au total, la complexité est en $O(n) + O(mn) = O(mn)$.

Partie III. Algorithme de balayage

Question 9 : Le tri fusion est un algorithme de tri dont le temps d'exécution est majoré par une constante fois $n \log n$.

Question 10 : Si la pile de l'enveloppe supérieure est vide, on met i dans la pile (ligne 3). Si la pile ne contient qu'un seul élément, on ajoute i à la pile (cas des lignes 6 à 8).
On note `p2` l'indice du point en haut de la pile et `p1` l'indice du point en deuxième position. Si le triangle $P_{p1}P_{p2}P_i$ est direct, on remet `p2` dans la pile ainsi que i (lignes 12 et 13). Sinon, on rappelle récursivement la fonction `majES`, l'indice `p2` ayant été supprimé de la pile `es`.

```
1  def majES(tab,es,i):
2      if isEmpty(es):
3          push(i,es)
4      else:
5          p2 = pop(es)
6          if isEmpty(es):
7              push(p2,es)
8              push(i,es)
9          else :
10             p1 = top(es)
11             if orient(tab,i,p2,p1)>0:
12                 push(p2,es)
13                 push(i,es)
14             else :
15                 majES(tab,es,i)
```

Question 11 : Le principe de la fonction `majEI` est le même que celui de la fonction `majES` sauf que le test sur le triangle $P_{p1}P_{p2}P_i$ est inversé.

```
1  def majEI(tab,ei,i):
2      if isEmpty(ei):
3          push(i,ei)
4      else:
5          p2 = pop(ei)
6          if isEmpty(ei):
7              push(p2,ei)
8              push(i,ei)
9          else :
10             p1 = top(ei)
11             if orient(tab,i,p1,p2)<0:
12                 push(p2,ei)
13                 push(i,ei)
14             else :
15                 majEI(tab,ei,i)
```

Question 12 On commence par créer deux piles, l'une pour l'enveloppe supérieure (`es`), l'autre pour l'enveloppe inférieure (`ei`). Dans la boucle des lignes 4 à 6, on parcourt l'ensemble des points en mettant à jour les piles `es` et `ei`. Les lignes 7 à 10, permettent de verser le contenu de la pile de l'enveloppe supérieure dans la pile de l'enveloppe inférieure, en supprimant les points en double (ligne 7 et 10). En ligne 11, on retourne la pile `ei` correspondant au résultat attendu.

```
1  def convGraham(tab,n):
2      es = newstack()
3      ei = newstack()
4      for i in range(n):
5          majEI(tab,ei,i)
6          majES(tab,es,i)
7      dummy = pop(es)
8      while(not(isEmpty(es))):
9          push(pop(es),ei)
10     dummy = pop(ei)
11     return(ei)
```

Question 13 : On peut trier les points par ordre croissant d'abscisse en $O(n \log n)$.
Le coût des fonctions `majEI` et `majES` est en $O(i)$ donc le coût de la boucle des lignes 4 à 6 de la fonction `convGraham` est en $O(\sum_{i=1}^{n} i) = O(n^2)$. En fait, chaque point est au plus ajouté une fois dans chacune des piles et au plus enlevé une fois donc le coût de la boucle des lignes 4 à 6 est en fait en $O(n)$.
Le recollement des deux piles est en $O(m)$ où m est le nombre de points du bord. Comme $m = O(n)$, on en déduit que le coût de la fonction `convGraham` est en $O(n)$ et donc le coût global de l'algorithme de Graham-Andrew est en $O(n \log n)$.