

## CORRIGÉ : ENVELOPPES CONVEXES DANS LE PLAN (X-ENS MP &amp; PC 2015)

Rédigé par Jean-Pierre Becirspahic (jp.becir@info-llg.fr).

## Partie I. Préliminaires

## Question 1.

```
def plusBas(tab, n):
    j = 0
    for k in range(1, n):
        if tab[1][k] < tab[1][j]:
            j = k
    return j
```

## Question 2.

- Si  $i = 0, j = 3, k = 4$  on a  $\det(\overrightarrow{p_i p_j}, \overrightarrow{p_i p_k}) = \begin{vmatrix} 4 & 4 \\ 1 & 4 \end{vmatrix} = 12 > 0$  donc le triangle  $p_i p_j p_k$  est orienté positivement.
- Si  $i = 8, j = 9, k = 10$  on a  $\det(\overrightarrow{p_i p_j}, \overrightarrow{p_i p_k}) = \begin{vmatrix} 1 & 4 \\ 3 & 4 \end{vmatrix} = -8 < 0$  donc le triangle  $p_i p_j p_k$  est orienté négativement.

## Question 3.

```
def orient(tab, i, j, k):
    det = (tab[0][j] - tab[0][i]) * (tab[1][k] - tab[1][i]) -
          (tab[1][j] - tab[1][i]) * (tab[0][k] - tab[0][i])
    if det > 0:
        return 1
    elif det < 0:
        return -1
    else:
        return 0
```

## Partie II. Algorithme du paquet cadeau

Question 4. La relation  $\leq$  est :

- *réflexive* car pour tout  $j \neq i$ ,  $\det(\overrightarrow{p_i p_j}, \overrightarrow{p_i p_j}) = 0$ ;
- *antisymétrique* car  $\det(\overrightarrow{p_i p_j}, \overrightarrow{p_i p_k}) = -\det(\overrightarrow{p_i p_k}, \overrightarrow{p_i p_j})$  donc  $p_j \leq p_k$  et  $p_k \leq p_j$  implique  $\det(\overrightarrow{p_i p_j}, \overrightarrow{p_i p_k}) = 0$ . Or par hypothèse  $p_i, p_j$  et  $p_k$  ne peuvent être alignés si  $j \neq k$  donc ceci implique  $j = k$ ;
- *totale* car l'une des deux conditions  $\det(\overrightarrow{p_i p_j}, \overrightarrow{p_i p_k}) \geq 0$  et  $\det(\overrightarrow{p_i p_k}, \overrightarrow{p_i p_j}) \leq 0$  est toujours vérifiée.

Il reste à montrer que la relation est *transitive*. Pour cela, on raisonne par l'absurde en supposant l'existence de trois points distincts  $p_j, p_k, p_l$  tels que  $p_j \leq p_k, p_k \leq p_l$  et  $p_l \leq p_j$ . Puisque ces trois points ne sont pas alignés, on peut considérer les coordonnées barycentriques du point  $p_i$  : l'unique triplet  $(a, b, c)$  vérifiant  $a + b + c = 1$  et tel que  $p_i$  soit le barycentre de  $(p_j, a), (p_k, b)$  et  $(p_l, c)$ . Alors :

$$\begin{aligned} \det(\overrightarrow{p_i p_j}, \overrightarrow{p_i p_k}) &= \det(b \overrightarrow{p_k p_j} + c \overrightarrow{p_l p_j}, a \overrightarrow{p_j p_k} + c \overrightarrow{p_l p_k}) = bc \det(\overrightarrow{p_k p_j}, \overrightarrow{p_l p_k}) + ac \det(\overrightarrow{p_l p_j}, \overrightarrow{p_j p_k}) + c^2 \det(\overrightarrow{p_l p_j}, \overrightarrow{p_l p_k}) \\ &= bc \det(\overrightarrow{p_l p_j}, \overrightarrow{p_l p_k}) + ac \det(\overrightarrow{p_l p_j}, \overrightarrow{p_l p_k}) + c^2 \det(\overrightarrow{p_l p_j}, \overrightarrow{p_l p_k}) = c \det(\overrightarrow{p_l p_j}, \overrightarrow{p_l p_k}). \end{aligned}$$

De même,  $\det(\overrightarrow{p_i p_k}, \overrightarrow{p_i p_l}) = a \det(\overrightarrow{p_j p_k}, \overrightarrow{p_j p_l}) = a \det(\overrightarrow{p_l p_j}, \overrightarrow{p_j p_k}) = a \det(\overrightarrow{p_l p_j}, \overrightarrow{p_l p_k})$  et  $\det(\overrightarrow{p_i p_l}, \overrightarrow{p_i p_j}) = b \det(\overrightarrow{p_l p_j}, \overrightarrow{p_l p_k})$  donc  $a, b$  et  $c$  sont de même signe (positif). Ceci prouve que  $p_i$  est à l'intérieur du triangle formé des points  $p_j, p_k$  et  $p_l$  et ne peut donc appartenir à l'enveloppe convexe du nuage de points, ce qui est absurde.

**Question 5.** Il s'agit de chercher le maximum de  $\{p_j \mid j \neq i\}$  pour la relation  $\leq$  :

```
def prochainPoint(tab, n, i):  
    if i > 0:  
        j = 0  
    else:  
        j = 1  
    for k in range(n):  
        if k != i and k != j and orient(tab, i, j, k) < 0:  
            j = k  
    return j
```

**Question 6.** La fonction ci-dessus commence par affecter à  $j$  la valeur 0 puis parcourt les valeurs  $k \in \llbracket 0, 11 \rrbracket$  à la recherche d'un élément  $p_k$  strictement plus grand que  $p_j$  pour la relation  $\leq$ .

- pour  $k = 1$  la séquence  $(p_{10}, p_0, p_1)$  est orientée négativement donc  $j$  prend la valeur 1 ;
  - pour  $k = 2$  la séquence  $(p_{10}, p_1, p_2)$  est orientée négativement donc  $j$  prend la valeur 2 ;
  - pour  $k = 5$  la séquence  $(p_{10}, p_2, p_5)$  est orientée négativement donc  $j$  prend la valeur 5 ;
- et ensuite la valeur de  $j$  n'est plus modifiée.

**Question 7.**

```
def convJarvis(tab, n):  
    i = plusBas(tab, n)  
    j = prochainPoint(tab, n, i)  
    env = [i]  
    while j != i:  
        env.append(j)  
        j = prochainPoint(tab, n, j)  
    return env
```

**Question 8.** La fonction `plusBas` a un coût temporel en  $O(n)$  et n'est utilisée qu'une fois ; la fonction `prochainPoint` a un coût temporel en  $O(n)$  et est utilisée  $m + 1$  fois où  $m$  est le cardinal de l'enveloppe convexe. Enfin, la méthode `append` est de coût (amorti) constant donc le coût total de la fonction `convJarvis` est un  $O(n) + O(mn) + O(m) = O(mn)$ .

### Partie III. Algorithme de balayage

**Question 9.** L'algorithme de tri fusion (*mergesort*) a un coût temporel en  $O(n \log n)$ .

**Question 10.** On écrit une version récursive de `majES` :

```
def majES(tab, es, i):  
    j = pop(es)  
    if not isEmpty(es):  
        k = top(es)  
        if orient(tab, i, j, k) < 0:  
            return majES(tab, es, i)  
    push(j, es)  
    push(i, es)
```

**Question 11.** À la différence de la fonction précédente, la fonction `majEI` se termine dès lors que la séquence  $(p_i, p_j, p_k)$  devient négative.

```
def majEI(tab, ei, i):  
    j = pop(ei)  
    if not isEmpty(ei):  
        k = top(ei)  
        if orient(tab, i, j, k) > 0:  
            return majEI(tab, ei, i)  
    push(j, ei)  
    push(i, ei)
```

**Question 12.** La fonction `convGraham` effectue tout d'abord les mises à jour successives des piles *ei* et *es* en parallèle puis transfère les éléments de *es* (à l'exception de ses extrémités) dans la pile *ei* qui sera finalement renvoyée.

```
def convGraham(tab, n):
    ei = newStack()
    es = newStack()
    push(0, ei)
    push(0, es)
    for i in range(1, n):
        majES(tab, es, i)
        majEI(tab, ei, i)
    x = pop(es)
    while not isEmpty(es):
        push(pop(es), ei)
    pop(ei)
    return ei
```

**Question 13.** Chaque point du nuage entre une seule fois dans chacune des piles *ei* et *es* et n'en sort que zéro ou une fois. les opérations élémentaires sur les piles ainsi que la fonction `orient` ayant un coût constant, le coût total du balayage est un  $O(n)$ . Sachant que le tri par abscisse croissante du nuage de points peut être effectué en  $O(n \log n)$ , le coût total de l'algorithme de GRAHAM-ANDREW est en  $O(n \log n) + O(n) = O(n \log n)$ .