ISM MP, Informatique
Année 2025/2026 01/12/2025

Devoir surveillé n°1

L'utilisation des calculatrices n'est pas autorisée pendant cette épreuve.
On attachera une grande importance à la concision, à la clarté et à la précision de la rédaction. L'indentation devra être matérialisée par un trait vertical et les codes non triviaux devront impérativement être commentés.

⚠ L'utilisation de la fonction `min` est interdite.

Enveloppes convexes dans le plan

Ce sujet a pour objectif de calculer des enveloppes convexes de nuages de points dans le plan affine, problème très classique en géométrie algorithmique. Un ensemble $C \subseteq \mathbb{R}^2$ est *convexe* si et seulement si pour toute paire de points $(p, q) \in C^2$, le segment d'extremités p et q est inclus dans C. L'*enveloppe convexe* d'un ensemble $P \subseteq \mathbb{R}^2$, notée Conv(P) est le plus petit convexe contenant P (plus petit au sens de l'inclusion). Dans le cas où l'ensemble P est fini (appelé *nuage de points*), le bord de Conv(P) est un polygone convexe dont les sommets appartiennent à P, comme illustré dans la figure 1.

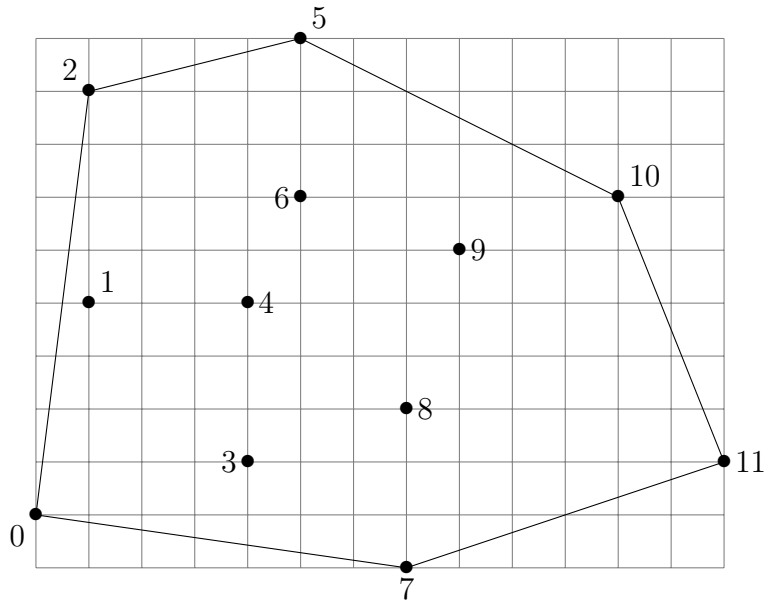

FIGURE 1 – Nuage de points numérotés de 0 à 11 et le bord de son enveloppe convexe

Dans ce sujet, nous allons décrire deux algorithmes de calcul du bord de l'enveloppe convexe d'un nuage de points P dans le plan affine. Le premier, dit *algorithme du paquet cadeau*, consiste à envelopper le nuage de points P progressivement en faisant pivoter une droite tout autour. Le deuxième, dit *de balayage*, consiste à balayer le plan horizontalement avec une droite verticale, tout en maintenant au fur et à mesure l'enveloppe convexe de la partie du nuage

Dans toute la suite, on supposera que le nuage de points P est de taille $n \geqslant 3$ et en position quelconque, c'est-à-dire qu'il ne contient pas 3 points distincts alignés.

Les nuages de points à traiter dans les programmes seront organisés dans des tableaux à deux dimensions. Les coordonnées, supposées entières, sont données dans une base orthonormée du plan, orientée dans le sens direct. La première ligne du tableau contient les abscisses tandis que la deuxième contient les ordonnées. Ainsi, la colonne d'indice j contient les deux coordonnées du point d'indice j. Ce dernier sera noté p_j dans la suite.

Par exemple, les coordonnées du nuage de points P de la figure 1 sont données dans le tableau :

i \ j	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
0	0	1	1	4	4	5	5	7	7	8	11	13
1	0	4	8	1	4	9	6	−1	2	5	6	1

implémenté en python par une liste de deux sous-listes :

```
tab=[[0, 1, 1, 4, 4, 5, 5,  7, 7, 8, 11, 13],
     [0, 4, 8, 1, 4, 9, 6, -1, 2, 5,  6,  1]]
```

Par exemple, le point p_3 a pour coordonnées `tab[0][3]`,`tab[1][3]`.

⚠ Dans toute la suite, le mot *tableau* désignera une liste de deux-sous listes comme dans l'exemple qui précède.

I Préliminaires

1 . Soit `tab` le tableau décrit en exemple précédemment. Que désigne `tab[0]` ? `tab[1]` ? `tab[0][0]` ? `tab[1][-1]` ?

2 . Écrire une fonction `ordMin(tab,n)` d'arguments `tab` un tableau de taille $2 \times n$ et `n` le nombre de points dans `tab` et qui renvoie la plus petite ordonnée.

Sur le tableau exemple précédent, le résultat de la fonction doit être la valeur -1.

3 . Écrire une fonction `occOrd(tab,n,y)` d'arguments `tab` un tableau de taille $2 \times n$, `n` le nombre de points dans `tab` et `y` une valeur d'ordonnée et qui renvoie la liste des indices des occurences de `y` dans la liste `tab[1]`.

Sur le tableau exemple précédent, le résultat de la fonction `occOrd(tab,n,1)` est `[3, 11]`.

4 . Écrire une fonction `plusBas(tab,n)` d'arguments `tab` un tableau de taille $2 \times n$ et `n` le nombre de points dans `tab` et qui renvoie l'indice j du point le plus bas (c'est-à-dire de plus petite ordonnée) parmi les points du nuage P. En cas d'égalité, la fonction doit renvoyer l'indice du point de plus petite abscisse parmi les points les plus bas. L'usage des fonctions `ordMin` et `occOrd` est impératif.

Sur le tableau exemple précédent, le résultat de la fonction doit être l'indice 7.

5 . Écrire une nouvelle version de la fonction `plusBas(tab,n)` mais sans fonction auxiliaire et avec une unique boucle `for`. Quelle version est préférable ?

Dans la suite, nous aurons besoin d'effectuer un seul type de test géométrique : celui de l'orientation.

Définition 1. *Étant donnés trois points p_i, p_j, p_k du nuage* P*, distincts ou non, le test d'orientation renvoie* $+1$ *si la séquence* (p_i, p_j, p_k) *est orientée positivement,* -1 *si elle est orientée négativement et* 0 *si les trois points sont alignés (c'est-à-dire si deux au moins sont égaux, d'après l'hypothèse de position quelconque).*

Pour déterminer l'orientation de (p_i, p_j, p_k), il suffit de calculer l'aire signée du triangle, comme illustrée sur la figure 2. Cette aire est la moitié du déterminant de la matrice 2×2 formée par les coordonnées des vecteurs $\overrightarrow{p_i p_j}$ et $\overrightarrow{p_i p_k}$.

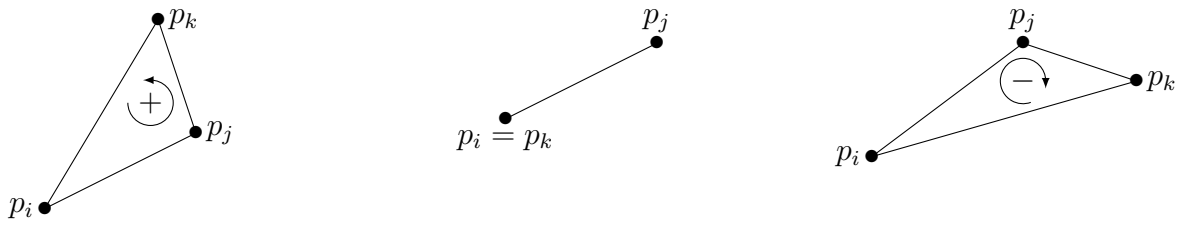

FIGURE 2 – Test d'orientation sur la séquence (p_i, p_j, p_k) : positif à gauche, nul au centre, négatif à droite

6 . Écrire une fonction `orient(tab,i,j,k)` d'arguments `tab` un tableau de taille $2 \times n$ et trois entiers `i`, `j`, `k` et qui renvoie le résultat du test d'orientation sur la séquence (p_i, p_j, p_k) de points de P. Le résultat sera de type `int`.

II Algorithme du paquet cadeau

Cet algorithme a été proposé par R. Jarvis en 1973. Il consiste à envelopper peu à peu le nuage de points P dans une sorte de paquet cadeau qui, à la fin du processus, est exactement le bord de Conv(P). On commence par insérer le point de plus petite ordonnée (celui d'indice 7 dans l'exemple de la figure 1) dans le paquet cadeau puis, à chaque étape de la procédure, on sélectionne le prochain point du nuage P à insérer.

La procédure de sélection fonctionne comme suit. Soit p_i le dernier point inséré dans le paquet cadeau à cet instant, par exemple $i = 10$ dans la configuration de la figure 3.

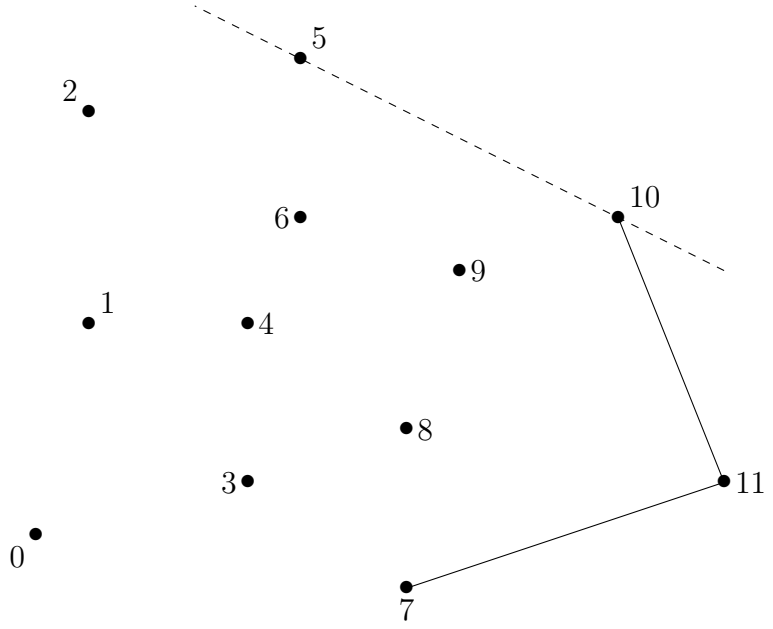

FIGURE 3 – Mise à jour du paquet cadeau après insertion du point p_{10}

Considérons la relation binaire $\preceq$ définie sur l'ensemble $\mathrm{P} \setminus \{p_i\}$ par

$$p_j \preceq p_k \iff \mathrm{orient}(tab, i, j, k) \leqslant 0$$

Il s'agit d'une relation *d'ordre total* sur l'ensemble $\mathrm{P} \setminus \{p_i\}$, c'est-à-dire :

- réfléxivité : pour tout $j \neq i$, $p_j \preceq p_j$;
- antisymétrie : pour tous $j, k \neq i$, $p_j \preceq p_k$ et $p_k \preceq p_j$ implique $j = k$;
- transitivité : pour tous $j, k, \ell \neq i$, $p_j \preceq p_k$ et $p_k \preceq p_\ell$ implique $p_j \preceq p_\ell$;
- totalité : pour tous $j, k \neq i$, $p_j \preceq p_k$ ou $p_k \preceq p_j$.

Avec cette relation d'ordre total, on peut toujours comparer deux points.

Ainsi, le prochain point à insérer est l'élément maximum pour la relation d'ordre $\preceq$. Dans la configuration de la figure 3, c'est le point d'indice 5 qui est le prochain point à insérer puisqu'il est maximum pour la relation $\preceq$, c'est-à-dire

$$\forall j \in [\![\, 0\,;\, 11\,]\!] \setminus \{10\} \qquad p_j \preceq p_5$$

Le prochain point à insérer peut se calculer en $\mathrm{O}(n)$ par une simple itération sur les points de $\mathrm{P} \setminus \{p_i\}$.

7 . Écrire une fonction `prochainPoint(tab,n,i)` d'arguments `tab` un tableau de taille $2 \times n$, `n` le nombre de points de `tab` et `i` l'indice du point inséré en dernier dans le paquet cadeau qui renvoie l'indice du prochain point à insérer. La fonction doit être de complexité temporelle en $\mathrm{O}(n)$.

On peut maintenant combiner la fonction `prochainPoint` avec la fonction `plusBas` pour calculer le bord de l'enveloppe convexe de P. On commence par insérer le point p_i d'ordonnée la plus basse puis on itère le processus de mise à jour du paquet cadeau jusqu'à ce que le prochain point à insérer soit de nouveau p_i. À ce moment là, on renvoie le paquet cadeau comme résultat sans insérer p_i une seconde fois.

8 . Écrire une fonction itérative `convJarvis(tab,n)` d'arguments `tab` un tableau de taille $2 \times n$ et `n` le nombre de points dans `tab` et qui renvoie une liste contenant les indices des sommets du

bord de l'enveloppe convexe, sans doublon.

Par exemple, pour la configuration décrite en figure 1, le résultat est `[7, 11, 10, 5, 2, 0]`.

On se propose d'écrire une version récursive du calcul de l'enveloppe convexe selon l'algorithme du paquet cadeau. On initialise une liste contenant le point le plus bas puis on calcule récursivement le prochain point de l'enveloppe à insérer dans cette liste jusqu'à atteindre le premier point inséré dans cette liste.

9 . Écrire une fonction récursive `conv_rec(tab,n,res)` d'arguments `tab` un tableau de taille $2\times n$, `n` le nombre de points dans `tab`, `res` une liste de points de l'enveloppe convexe en construction et qui complète la liste `res` jusqu'à atteindre à nouveau le point `res[0]`.

10 . Écrire une fonction `convJarvisRec(tab,n)` d'arguments `tab` un tableau de taille $2 \times n$ et `n` le nombre de points dans `tab` et qui renvoie une liste contenant les indices des sommets du bord de l'enveloppe convexe, sans doublon. L'usage de la fonction `conv_rec` est impératif. La fonction `convJarvisRec` amorce la construction récursive de l'enveloppe convexe en réalisant le premier appel de `conv_rec`.

III Algorithme de tri

11 . Citer trois algorithmes de tri dont deux s'appuyant sur le paradigme « diviser pour régner ». Préciser leurs complexités temporelles en détaillant, si besoin, le meilleur cas et le pire cas.

12 . Écrire une fonction `tri(L)` d'argument `L` une liste d'entiers qui renvoie la liste triée. L'algorithme de tri implémenté, pas nécessairement en place, devra s'appuyer sur le paradigme « diviser pour régner ».

13 . En s'inspirant de la fonction `tri`, écrire une fonction récursive `tri_ind(L,I)` d'arguments `L` une liste de n entier et `I` la liste `[0, 1, ..., n-1]` contenant les indices des éléments de `L` qui renvoie un couple contenant la liste triée et la liste des indices permutés par le tri. Par exemple, l'appel `tri_ind([3,0,6,2],[0,1,2,3])` renvoie `([0, 2, 3, 6], [1, 3, 0, 2])`.

14 . Écrire une fonction `tri_tab(tab)` d'argument `tab` un tableau de taille $2 \times n$ qui renvoie le tableau réordonné par abscisses croissantes.

IV Modèle de piles

Dans la suite, nous aurons besoin d'utiliser des piles d'entiers dont on rappelle la définition ci-dessous.

Définition 2. *Une* pile d'entiers *est une structure de données permettant de stocker des entiers et d'effectuer les opérations suivantes en temps constant, indépendamment de la taille de la pile :*

- *créer une nouvelle pile vide ;*
- *déterminer si une pile est vide ;*
- *empiler un entier au sommet de la pile ;*
- *déterminer la valeur de l'entier au sommet de la pile ;*
- *dépiler l'entier au sommet de la pile.*

15 . Citer une méthode de programmation qui utilise des structures de piles.

16 . Proposer une implémentation de ce modèle de piles d'entiers en écrivant les fonctions suivantes :

- `Pile()` sans argument et qui renvoie une pile vide ;
- `estVide(P)` d'argument `P` une pile qui renvoie `True` si celle-ci est vide et `False` sinon ;
- `empiler(P,x)` d'arguments `P` une pile, `x` un entier qui empile `x` au sommet de `P` en modifiant `P` directement et ne renvoie rien ;
- `sommet(P)` d'argument `P` une pile qui renvoie l'entier au sommet de la pile sans modifier celle-ci ;
- `depiler(P)` d'argument `P` une pile qui dépile l'élément au sommet de la pile et le renvoie (la pile est donc modifiée).

⚠ Dans la suite, les manipulations sur les piles se font uniquement avec les fonctions listées à la question précédente, complètement indépendamment des choix d'implémentation de celles-ci.

V Algorithme de balayage

Cet algorithme a été proposé par R. Graham en 1972. Nous allons écrire la variante (plus simple) proposée par A. Andrew quelques années plus tard.

⚠ On suppose que les points fournis en entrée sont triés par abscisse croissante comme c'est le cas dans l'exemple du tableau `tab` donné au début du sujet.

L'idée de l'algorithme est de balayer le nuage de points horizontalement de gauche à droite par une droite verticale, tout en mettant à jour l'enveloppe convexe des points de P situés à gauche de cette droite comme illustré dans la figure 4.

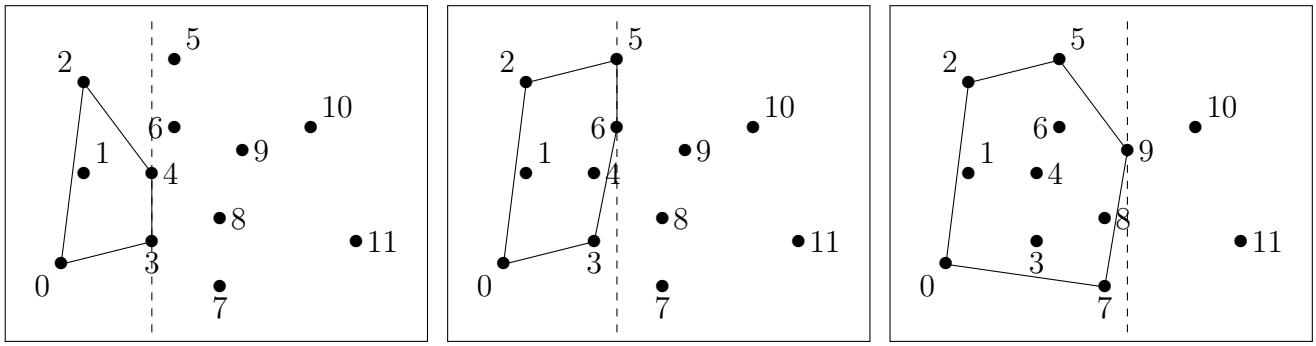

FIGURE 4 – Diverses étapes dans la procédure de balayage - la droite de balayage est en tirets

Plus précisément, l'algorithme visite chaque point de P une fois, par ordre croissant d'abscisse (donc par ordre croissant d'indice de colonne dans le tableau `tab` car celui-ci est trié). À chaque nouveau point p_i visité, il met à jour le bord de l'enveloppe convexe du sous-nuage $\{p_0, \dots, p_i\}$ situé à gauche de p_i. On remarque que les points p_0 et p_i sont sur ce bord et on appelle *enveloppe supérieure* la partie du bord de $\text{Conv}\{p_0, \dots, p_i\}$ située au-dessus de la droite passant par p_0 et p_i (p_0 et p_i compris) et *enveloppe inférieure* la partie du bord de $\text{Conv}\{p_0, \dots, p_i\}$ située au-dessous (p_0 et p_i compris). Le bord de $\text{Conv}\{p_0, \dots, p_i\}$ est donc constitué de l'union de ces deux enveloppes, après suppression des doublons de p_0 et p_i.

Par exemple, dans le cas du nuage P de la figure 4 gauche, le sous-nuage $\{p_0, p_1, p_2, p_3, p_4\}$ a pour enveloppe supérieure la séquence (p_0, p_2, p_4) et pour enveloppe inférieure la séquence (p_0, p_3, p_4), le bord de son enveloppe convexe étant donné par la séquence (p_0, p_3, p_4, p_2).

Informatiquement, les indices des sommets des enveloppes inférieure et supérieure seront stockés dans deux piles d'entiers séparées : `ei` pour l'enveloppe inférieure et `es` pour l'enveloppe supérieure.

La mise à jour de l'enveloppe supérieure est illustrée dans la figure 5 : tant que le point visité (p_9 dans ce cas) et les deux points dont les indices sont situés au sommet de la pile `es` (dans l'ordre : p_8 et p_5) forme une séquence d'orientation négative (p_9, p_8, p_5), on dépile l'indice situé au somme de `es` (p_8 dans ce cas). On poursuit ce processus d'élimination jusqu'à ce que l'orientation devienne positive ou qu'il ne reste plus qu'un seul indice dans la pile. L'indice du point visité (p_9 dans ce cas) est alors inséré au sommet de `es`. La mise à jour de l'enveloppe inférieure s'opère de manière symétrique.

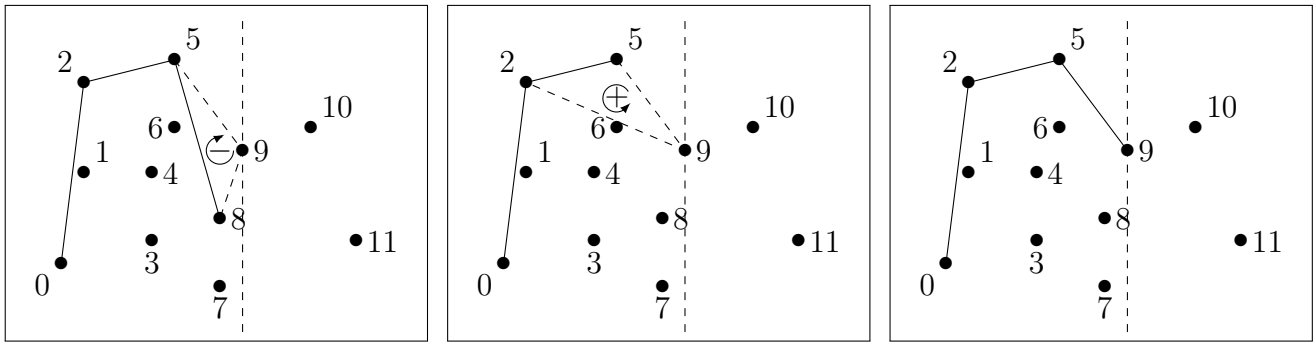

FIGURE 5 – Mise à jour de l'enveloppe supérieure lors de la visite du point p_9

17 . Écrire une fonction `majES(tab,es,i)` d'arguments le tableau `tab`, la pile `es` et l'indice du point `i` à visiter qui met à jour l'enveloppe supérieure du sous-nuage. La complexité temporelle doit être en $O(i)$.

18 . Écrire une fonction `majEI(tab,ei,i)` d'arguments le tableau `tab`, la pile `ei` et l'indice du point `i` à visiter qui met à jour l'enveloppe inférieure du sous-nuage en $O(i)$.

19 . Écrire une fonction `convGraham(tab,n)` d'arguments `tab` un tableau de taille $2 \times n$ représentant le nuage P, `n` le nombre de points dans `tab`, qui effectue le balayage des points de P comme décrit précédemment et renvoie une pile contenant les indices du sommet de Conv(P) triés dans l'ordre positif d'orientation.

Par exemple, sur le nuage de la figure 1, le résultat de la fonction est une pile contenant de la base au sommet : 0, 7, 11, 10, 5, 2.

20 . Déterminer la complexité temporelle de `convGraham`. Proposer une nouvelle version de la fonction pour le cas où les abscisses du tableau transmis en argument ne sont pas préalablement triées par ordre croissant et préciser l'impact éventuel sur la complexité temporelle.