Table des matières

Petit devoir surveillé d'informatique

Durée : deux heures. Le sujet comporte deux exercices sur la récursivité, et un problème sur les piles et les tris.

Exercice 1. * Une fonction inutile

On considère la fonction `f` suivante :

```
def f(n):
    if n==0:
        return 0
    else:
        res=0
        for i in range(n):
            res += f(n-1)
        return res
```

1. Que renvoie f ? Le démontrer par récurrence.
   solution : Pour tout $n \in \mathbb{N}$ posons $P(n)$: « $f(n)$ termine et renvoie 0. ».
   - *Initialisation :* Vu le cas de base, $f(0) = 0$.
   - Soit $n \in \mathbb{N}^*$, supposons $P(n-1)$. Exécutons $f(n)$. Par hyothèse de récurrence $f(n-1)$ termine et renvoie 0. Alors la boucle consiste à sommer n fois 0. Donc `res` contient 0 à son issue. Et $f(n)$ termine et renvoie 0.
2. On note pour tout $n \in \mathbb{N}$, C_n le nombre d'additions pour exécuter `f(n)`.
   (a) Donner C_0 ainsi que la relation de récurrence vérifiée par la suite C.
   solution : C_0 et pour tout $n \in \mathbb{N}^*$, $C_n = n + n \times C_{n-1}$. Le $n+$ vient des n additions dans la boucle, et le $n \times C_{n-1}$ vient des n appels récursifs à $f(n-1)$ (ce qui est parfaitement idiot au passage : il aurait fallu calculer $f(n-1)$ une seule fois et enregistrer le résultat.).
   (b) Démontrer que pour tout $n \in N^*$, $C_n \geqslant n!$. Commentaires sur cette fonction ?
   solution : Récurrence évidente. Attention à l'initialiser à 1 car pour $n = 0$ la formule est fausse.
   (c) *Bonus :* écrire une fonction plus simple pour calculer la même chose que `f`.
   solution :

```
def f_mieux(n):
    return 0
```

Exercice 2. ** Des chiffres et des lettres, sans lettres, simplifié

Soit `t` un tableau d'entiers et $N \in \mathbb{N}$. On veut savoir s'il est possible d'obtenir N en sommant des éléments de `t`. Les répétitions sont autorisée, et on considère qu'une somme d'aucun nombre vaut 0. Par exemple pour `t==[7,5,3` et $N = 13$ la réponse est oui, mais pour `t==[7,8]` et $N = 13$ la réponse est non.

On propose d'écrire une fonction auxiliaire `sommeFaitN_aux` qui prend en entrée `t` et N mais aussi un indice i et qui indique si on peut obtenir N en sommant des éléments de `t[i:]`. On rappelle que `t[i:]` est le sous-tableau de `t` composé des éléments d'indice supérieur ou égal à i.

Indication : L'idée est qu'il y a deux possibilités : mettre i dans la somme (il reste alors à faire $N - i$) ou ne pas le mettre (il faut alors faire N avec les éléments de `t[i+1:]`).

Attention aux cas de base. La fonction peut être rédigée en six lignes.

solution :

```python
def sommeFaitAux(N,t,i):
    """ Indique si on peut faire N en sommant des éléments de t[i:]"""
    if N<0:
        return False
    elif i==len(t):
        return N==0 # t[i:] est vide, le seul nombre qu'on puisse obtenir est donc 0
    else:
        return sommeFaitAux(N-t[i],t,i)  or  sommeFaitAux(N,t,i+1)
```

Problème : calcul d'enveloppe convexe.

1 Introduction et notations

Ce sujet a pour objectif de calculer des enveloppes convexes de nuages de points dans le plan affine, un grand classique en géométrie algorithmique. On rappelle qu'un ensemble $C \subseteq \mathbb{R}^2$ est convexe si et seulement si pour toute paire de points $p, q \in C$, le segment de droite $[p, q]$ 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 . Dans le cas où P est un ensemble fini (appelé nuage de points), le bord de $Conv(P)$ est un polygône convexe dont les sommets appartiennent à P , comme illustré dans la figure 1.

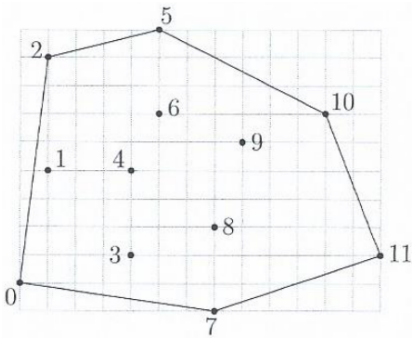

FIGURE 1 – Un nuage de points, numérotés de 0 à 11, et le bord de son enveloppe convexe.

Le calcul de l'enveloppe convexe d'un nuage de points est un problème fondamental en informatique, qui trouve des applications dans de nombreux domaines comme :

- la robotique, par exemple pour l'accélération de la détection de collisions dans le cadre de la planification de trajectoire,
- le traitement d'images et la vision, par exemple pour la détection d'objets convexes (comme des plaques minéralogiques de voiture) dans des scènes 2d,
- l'informatique graphique, par exemple pour l'accélération du rendu de scènes 3d par lancer de rayons,
- la théorie des jeux, par exemple pour déterminer l'existence d'équilibres de Nash,
- la vérification formelle, par exemple pour déterminer si une variable risque de dépasser sa capacité de stockage ou d'atteindre un ensemble de valeurs interdites lors de l'exécution d'une boucle dans un programme,

et bien d'autres encore.

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

Ces hypothèses vont permettre de simplifier les calculs en ignorant les cas pathologiques, comme par exemple la présence de 3 points alignés sur le bord de l'enveloppe convexe. Nos programmes prendront en entrée un nuage de points P dont les coordonnées sont stockées dans un tableau `tab` à 2 dimensions, comme dans l'exemple ci-dessous qui contient les coordonnées du nuage de points de la figure 1 :

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

Précisons que 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 nommé p_j dans la suite.

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

Définition 1.1. *Une pile d'entiers est une structure de données permettant de stocker des entiers et d'effectuer les opérations suivantes en temps constant (indépendant de la taille de la pile) :*

- *créer une nouvelle pile vide,*
- *déterminer si la pile est vide,*
- *insérer un entier au sommet de la pile,*
- *déterminer la valeur de l'entier au sommet de la pile,*
- *retirer l'entier au sommet de la pile.*

Nous supposerons fournies les fonctions suivantes, qui réalisent les opérations ci-dessus et s'executent chacune en temps constant :

- `newStack()`, qui ne prend pas d'argument et renvoie une pile vide,
- `isEmpty(s)`, qui prend une pile `s` en argument et renvoie `True` ou `False` suivant que `s` est vide ou non,
- `push(i, s)`, qui prend un entier i et une pile `s` en argument, insère i au sommet de `s`, et ne renvoie rien,
- `top(s)`, qui prend une pile `s` (supposée non vide) en argument et renvoie la valeur de l'entier au sommet de `s`,
- `pop(s)`, qui prend une pile `s` (supposée non vide) en argument, supprime l'entier au sommet de `s` (c'est-à-dire à la fin de la liste) et renvoie sa valeur.

Dans la suite il est demandé aux candidats de manipuler les piles uniquement au travers de ces fonctions, sans aucune hypothèse sur la représentation effective des piles en mémoire.

2 Préliminaires

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

Définition 2.1. *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 générale).*

Pour déterminer l'orientation de (p_i, p_j, p_k), il suffit de calculer l'aire signée du triangle, comme illustré 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 $\overrightarrow{p_i p_k}$.

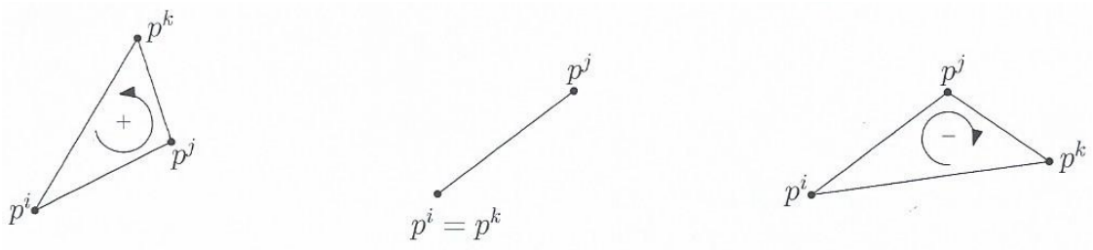

FIGURE 2 – Test d'orientation sur la séquence (p_i, p_j, p_k) : positif à gauche, nul au centre, négatif à droite.

1. Sur le tableau exemple précédent, donner le résultat du test d'orientation pour les choix d'indices suivants :
   (a) $i = 0$, $j = 3$, $k = 4$,
   (b) $i = 8$, $j = 9$, $k = 10$.
2. Écrire une fonction `orient(tab, i, j, k)` qui prend en paramètres le tableau `tab` et trois indices de colonnes, potentiellement égaux, et qui renvoie le résultat (−1, 0 ou +1) du test d'orientation sur la séquence (p_i, p_j, p_k) de points de P .

3 Tri

1. Parmi les algorithmes de tri que vous connaissez, mentionnez-en un qui a un temps d'exécution majoré par une constante fois $n \log n$ sur les entrées de taille n.
2. Écrire une procédure `fusionEntre(t, deb, m, fin)` qui prend en paramètres un tableau `t` et trois indices `de`, `m`, et `fin`, le tout vérifiant que `t[deb:m]` et `t[m:fin]` sont triés dans l'ordre croissant, et ayant pour effet qu'à son issue `t[deb:fin]` contient les mêmes éléments qu'initialement mais dans l'ordre croissant.
3. En déduire une procédure `triFusion` qui prend un tableau `t` et le trie selon l'algorithme du tri par partition-fusion. Votre programme doit modifier t et non renvoyer un nouveau tableau.
   On pourra commencer par une procédure `trieEntre` qui prend en entrée `t`, `deb`, et `fin` et qui trie la portion `t[deb:fin]`.
4. La relation `<` de Python lorsqu'appliquée à des tableaux consiste à comparer le premier élément des tableaux, puis en cas d'égalité les deuxièmes éléments, etc...
   Par exemple `[1, 15] < [2,0]` renvoie `True`, de même que `[2, 2] < [2,3]`.
   Soit `t` le tableau défini par `t = [ [1,8], [4,1], [7,-1], [0,0], [4,0], [1,4], [8,5]]`. Que vaut `t` après qu'on lui aie appliqué la procédure `triFusion` ?

4 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.

La première étape consiste à trier les n points du nuage P par ordre croissant d'abscisse.

À partir de maintenant, sauf mention du contraire, on supposera 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 3.

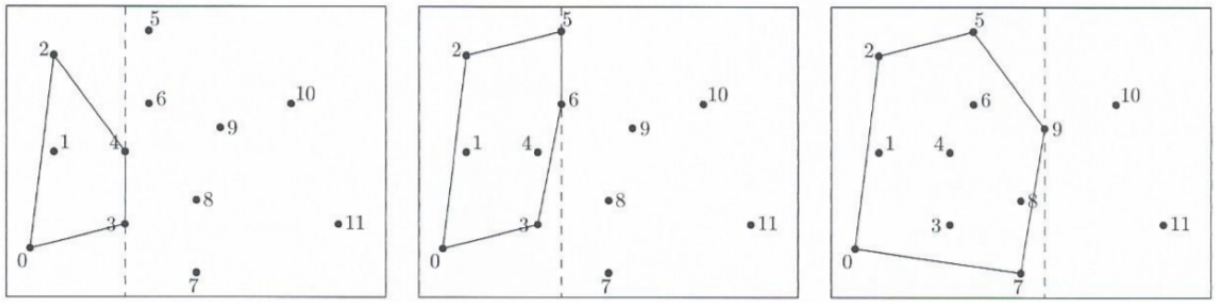

FIGURE 3 – Diverses étapes dans la procédure de balayage. La droite de balayage est en tirets.

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 $\{p_0, \cdots, p_i\}$ 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 $Conv\{p_0, \cdots, p_i\}$ 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 $Conv\{p_0, \cdots, p_i\}$ située au-dessous (p_0 et p_i compris). Le bord de $Conv\{p_0, \cdots, p_i\}$ 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 3 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 enveloppe inférieure) et `es` (pour enveloppe supérieure).

La mise à jour de l'enveloppe supérieure est illustrée dans la figure 4 : 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 (p_9, p_8, p_5)

d'orientation négative (voir la définition 2.1 pour rappel de l'orientation), on dépile l'indice situé au sommet de `es` (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.

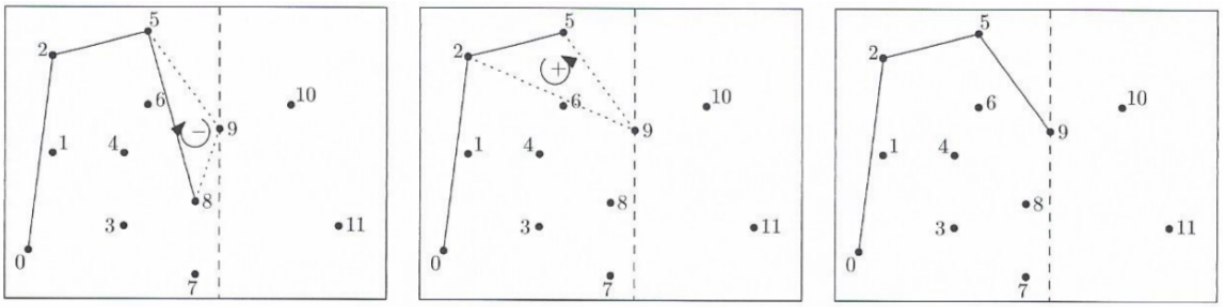

FIGURE 4 – Mise à jour de l'enveloppe supérieure lors de la visite du point p 9 .

5. Écrire une fonction `majES(tab, es, i)` qui prend en paramètre le tableau `tab` ainsi que la pile `es` et l'indice i du point à visiter, et qui met à jour l'enveloppe supérieure du sous- nuage. Le temps d'exécution de votre fonction doit être majoré par une constante fois i.
6. Écrire une fonction `majEI(tab, ei, i)` qui effectue la mise à jour de l'enveloppe inférieure, avec le même temps d'exécution.
7. Écrire maintenant une fonction `convGraham(tab, n)` qui prend en paramètre le tableau `tab` de taille $2 \times n$ représentant le nuage P , et qui effectue le balayage des points de P comme décrit précédemment. On supposera les colonnes du tableau tab déjà triées par ordre croissant d'abscisse. La fonction doit renvoyer une pile `s` contenant les indices des sommets du bord de $Conv(P)$ triés dans l'ordre positif d'orientation, à commencer par le point p_0 .

   Par exemple, sur le nuage de la figure 1, le résultat de la fonction `convGraham` doit être la pile `s` contenant la suite d'indices 0, 7, 11, 10, 5, 2 dans cet ordre, l'indice 0 se trouvant au fond de la pile s et l'indice 2 au sommet de `s`.
8. Écrire enfin la fonction finale, identique à la précédente hormis le fait que le tableau `tab` passé en argument n'est plus supposé trié par ordre d'abscisses croissantes.
9. Analyser brièvement le temps d'exécution de l'algorithme de balayage décrit précédemment, en supposant une fois encore que les points du nuage fourni en entrée sont déjà triés par abscisse croissante. En déduire que le temps d'exécution total de l'algorithme de Graham-Andrew est majoré par une constante fois $n \log n$.