

MASTER 1 MEEF : CAPES MATHS OPTION INFORMATIQUE

ECRIT BLANC - EPREUVE DE 5 HEURES

Les résultats des questions pourront être réutilisés ainsi que les différentes fonctions demandées non traitées au cours de l'épreuve. L'usage de la calculatrice est interdit.

Problème 1

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 $\text{Conv}(P)$, est le plus petit convexe contenant P . Dans le cas où P est un ensemble fini (appelé nuage de points), le bord de $\text{Conv}(P)$ est un polygone dont les sommets appartiennent à P , comme illustré dans la figure 1.

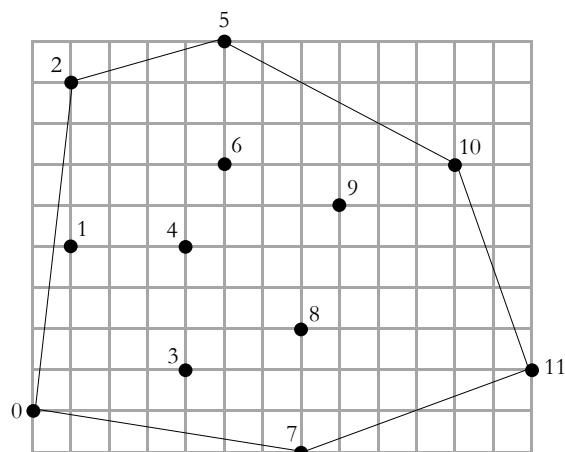


Figure 1 : Un nuage de points, numérotés de 0 à 11, et le bord de son enveloppe convexe.

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 :

- l'animation et la robotique, par exemple pour l'accélération de la détection de collisions dans le cadre de la planification de la trajectoire,
- le traitement d'images et la vision, par exemple pour la détection d'objets convexes (comme des plaques minéralogiques de voiture) dans les scènes 2D,
- l'informatique graphique, par exemple pour l'accélération du rendu de scènes 3D par lancer de rayon,
- 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 ce sujet, nous allons écrire deux algorithmes de calcul de 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 point 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 située à gauche de cette droite verticale. Les deux algorithmes sont illustrés respectivement sur les figures 3 et 4.

Le temps d’exécution du premier algorithme est majoré par une constante fois nm , celui du deuxième par une constante fois $n \log n$, où n désigne le nombre total de points de P et m le nombre de points de P appartenant au bord de $Conv(P)$.

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. Les 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 ligne 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.

Partie A : Préliminaires

On se propose d’utiliser le module `pyplot` du package `matplotlib` pour effectuer les tracés des points et du bord de l’enveloppe convexe. La fonction `plot` de ce module reçoit trois arguments : deux listes X et Y de même taille, dont on notera x_i et y_i les éléments, et un troisième paramètre représentant le style de tracé sous forme d’une chaîne de caractères. Pour une ligne noire continue reliant les points de coordonnées (x_i, y_i) , ce style est décrit par la chaîne `‘k-’` et pour dessiner des points noirs ce style est décrit par la chaîne `‘ko’`.

Ainsi le code suivant affiche trois points et les deux segments de droite les connectant.

```

from matplotlib.pyplot import *
X = [0,1,2]
Y = [-1,2,0]
plot(X,Y,'ko') # dessine les points
plot(X,Y,'k-') # dessine les segments de droites

```

1. Ecrire une fonction **dessinerPoints** qui reçoit en entrée un tableau à 2 dimensions (abscisses en première et ordonnées en seconde dimension) et qui dessine les points du nuage donnés dans le tableau.
2. Ecrire une fonction **dessinerEnveloppe** qui reçoit en entrée un tableau à 2 dimensions (abscisses en première et ordonnées en seconde dimension) et qui dessine les segments de droite reliant, dans l'ordre du tableau, tous les points.
3. Ecrire un programme qui dessine les points et l'enveloppe de la figure 1.
4. Ecrire une fonction **plusBas** qui prend en paramètre le tableau à 2 dimensions **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, votre fonction devra renvoyer l'indice du point de plus petite abscisse parmi les plus bas.

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

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

Définition 1 : Etant donnés trois points p_i, p_j et 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 vaut la moitié du déterminant de la matrice 2×2 formée par les coordonnées des vecteurs $\overrightarrow{p_i p_j}$ et $\overrightarrow{p_i p_k}$.

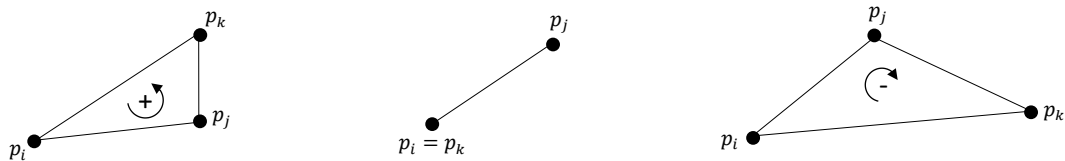


Figure 2 : Test d'orientation sur la séquence (p_i, p_j, p_k) : positif à gauche, nul au centre, négatif à droite

5. Sur l'exemple de la figure 1, donner le résultat du test d'orientation pour les choix d'indices suivants :

- $i = 0, j = 3, k = 4$
- $i = 8, j = 9, k = 10$

6. Ecrire une fonction **orient** qui prend en paramètres le tableau **tab** des points du nuage et trois indices de colonnes (i, j, k) , potentiellement égaux, et qui renvoie le résultat (-1, 0 ou +1) du test d'orientation sur la séquence (p_i, p_j, p_k) .

Partie B : 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 coordonnée (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 l'exemple de la figure 3.

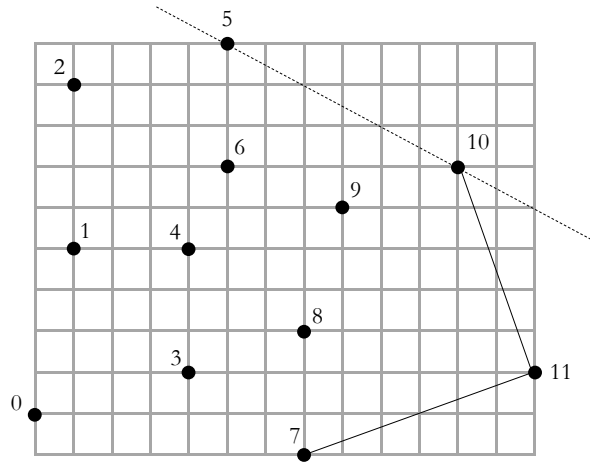


Figure 3 : Mise à jour du paquet cadeau après insertion du point p_{10} .

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

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

7. Justifier brièvement le fait que \preceq est une relation d'ordre total sur l'ensemble $P \setminus \{p_i\}$, c'est-à-dire :

- (réflexivité) pour tout $j \neq i$, $p_j \preceq p_j$
- (antisymétrie) pour tout $j, k \neq i$, $p_j \preceq p_k$ et $p_k \preceq p_j$ implique $j = k$
- (transitivité) pour tout $j, k, l \neq i$, $p_j \preceq p_k$ et $p_k \preceq p_l$ implique $p_j \preceq p_l$
- (totalité) pour tout $j, k \neq i$, $p_j \preceq p_k$ ou $p_k \preceq p_j$

Ainsi le prochain point à insérer (le point d'indice 5 dans la figure 3) est l'élément maximum pour la relation d'ordre \preceq . Il peut se calculer en temps linéaire par une simple itération sur les points de l'ensemble $P \setminus \{p_i\}$.

8. Ecrire la fonction **prochainPoint** qui prend en paramètres le tableau **tab** de taille $2 \times n$ ainsi que l'indice i du point inséré en dernier dans le paquet cadeau. La fonction renvoie l'indice du prochain point à insérer. Le temps d'exécution de cette fonction doit être majoré par une constante fois n , pour tout n et i . La constante doit être indépendante de n et i , et on ne demande pas de la préciser.

9. Décrire à la main le déroulement de la procédure **prochainPoint** sur l'exemple de la figure 3. Plus précisément, indiquer la séquence des points de l'ensemble $P \setminus \{p_i\}$ considérés et la valeur de l'indice du maximum à chaque itération.

On peut maintenant combiner la fonction **prochainPoint** avec la fonction **plusBas** de la question 4 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 . A ce moment-là, on renvoie le paquet cadeau comme résultat sans insérer p_i une seconde fois.

Un détail technique : comme la taille du paquet cadeau augmente peu à peu lors du processus, et qu'à la fin elle peut être petite par rapport au nombre n de points de P , nous stockerons les indices des points du paquet cadeau dans une liste. Par exemple, sur le nuage de la figure 1, le résultat sera la liste $[7,11,10,5,2,0]$.

10. Modifier la fonction **dessinerEnveloppe** de la question 2 afin qu'elle prenne en paramètre le tableau **tab** de taille $2 \times n$ et la liste des indices des points de l'enveloppe à dessiner.

11. Ecrire une fonction **convJarvis** qui prend en paramètre le tableau **tab** de taille $2 \times n$ représentant le nuage P , et qui renvoie une liste contenant les indices des sommets du bord de l'enveloppe convexe de P ,

sans doublon. Le temps d'exécution de cette fonction doit être majoré par une constante fois $n \times m$, où m est le nombre de points de P situés sur le bord de $Conv(P)$.

12. Justifier brièvement le temps d'exécution de l'algorithme du paquet cadeau de Jarvis.

Intermède : piles d'entiers

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 (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
- lire la valeur de l'entier au sommet de la pile
- retirer l'entier au sommet de la pile

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

- **nouvellePile()**, qui ne prend pas d'argument et renvoie une pile vide
- **estPileVide(p)**, qui prend une pile **p** en argument et renvoie **True** ou **False** suivant que la pile **p** est vide ou non
- **empilerPile(i,p)**, qui prend un entier **i** et une pile **p** en argument, insère **i** au sommet de la pile (c'est-à-dire à la fin de la liste), et ne renvoie rien
- **sommetPile(p)**, qui prend une pile **p** supposée non vide en argument et renvoie la valeur de l'entier au sommet de la pile
- **depilerPile(p)**, qui prend une pile **p** supposée non vide en argument, supprime l'entier au sommet de la pile et renvoie sa valeur

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

Partie C : 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, en conservant tous les points de même abscisse dans un ordre arbitraire.

13. 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 .

A partir de maintenant, 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 4.

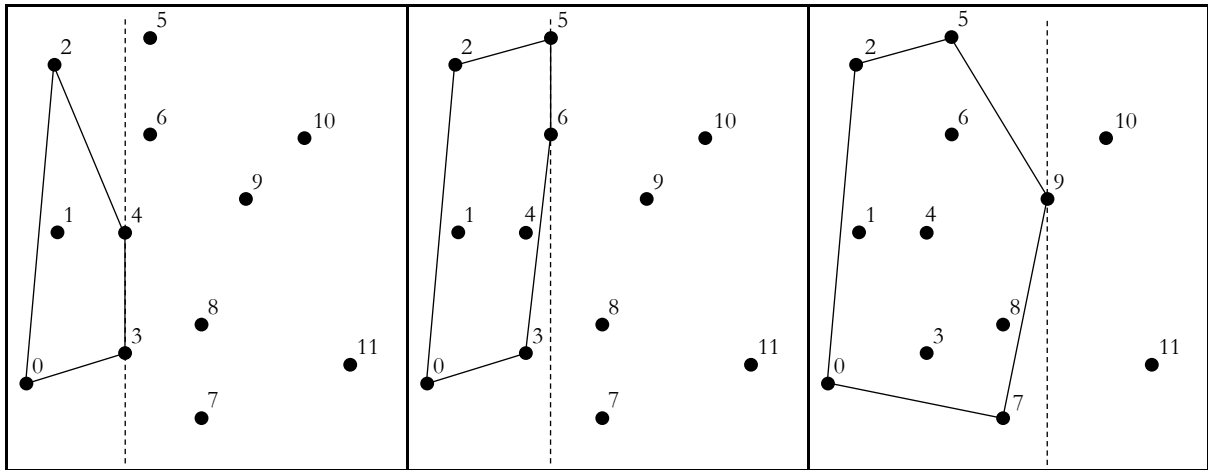


Figure 4 : Diverses étapes dans la procédure de balayage. La droite de balayage est en tirets.

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). A 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 $Conv\{p_0, \dots, p_i\}$ 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 $Conv\{p_0, \dots, p_i\}$ située au-dessous (p_0 et p_i compris). Le bord de $Conv\{p_0, \dots, p_i\}$ 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, nommées **ei** (pour enveloppe inférieure) et **es** (pour enveloppe supérieure).

La mise à jour de l'enveloppe supérieure est illustrée sur la figure 5 : tant que le point visité (p_9 dans ce cas) et les deux points dont les indices 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 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'effectue de manière symétrique.

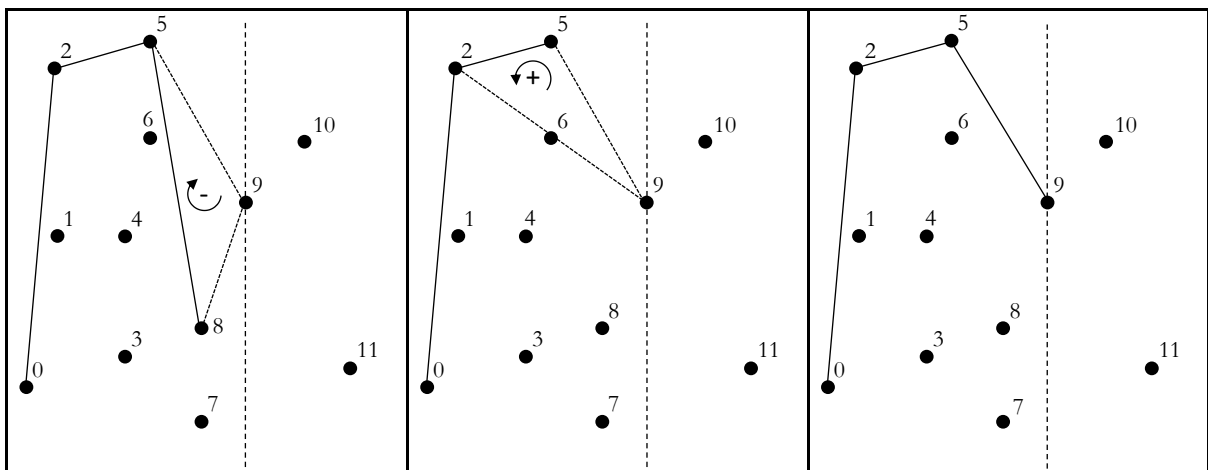


Figure 5 : Mise à jour de l'enveloppe supérieure lors de la visite du point p_9 .

14. Ecrire une fonction **majES** qui prend en paramètre le tableau **tab** ainsi que la pile **es** et l'indice **i** du point à visiter. Cette fonction met à jour l'enveloppe supérieure du sous-nuage. Le temps d'exécution de cette fonction doit être majoré par une constante fois **i**.

15. Ecrire une fonction **majEI** qui effectue la mise à jour de l'enveloppe inférieure, avec le même temps d'exécution.

16. Ecrire une fonction **convGraham** 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 déjà triées par ordre croissant d'abscisse. La fonction doit renvoyer une pile 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 contenant la suite d'indices 0, 7, 11, 10, 5, 2 dans cet ordre. L'indice 0 se trouvant au fond de la pile et l'indice 2 au sommet de la pile.

17. Analyser brièvement le temps d'exécution de l'algorithme de Graham-Andrew, 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 bien majoré par une constante fois $n \log n$.

Problème 2

Ce problème consiste en l'analyse et l'exploitation de données GPS (Global Positioning System) acquises par un récepteur équipant un véhicule. Ces dernières sont obtenues via un dispositif de réception installé à bord et fonctionnant à une fréquence allant de 1 à 4 Hertz. Chacune des mesures reçues contient (entre autres) des informations de positionnement (latitude, longitude) et de vitesse.

Le capteur inclus dans le récepteur GPS envoie les informations au calculateur embarqué via un bus de communication suivant un protocole particulier. Dans le cadre de ce sujet, le protocole retenu est le protocole NMEA 0183 (National Marine Electronics Association) utilisant des trames de type GPRMC (Recommended minimum specific GPS/Transit data) et utilisable avec la plupart des récepteurs GPS. Le format des trames GPRMC est défini en annexe 1.

Partie A : Acquisition des trames NMEA via une liaison série

L'objectif de cette première partie est d'enregistrer les trames GPS reçues par le décodeur dans un fichier nommé **tramesNMEA.txt**. Ce fichier sera utilisé en temps différé pour afficher le parcours réalisé. Le décodeur GPS reçoit des trames provenant des satellites à la fréquence de 1 Hertz. Elles sont ensuite transmises sur un port série dans le but d'être traitées par le calculateur embarqué.

A.1 Liaison série

Le décodeur GPS envoie sur la liaison série de type RS-232C une trame GPRMC. De manière générale, une liaison série permet de véhiculer, sur un nombre réduit de supports physiques, des informations élémentaires les unes à la suite des autres. Dans le protocole RS-232C, pour chaque octet à transmettre, 1 bit de start, 8 bits de données (correspondant à l'octet ou au caractère à transmettre), puis 1 bit de stop sont

envoyés sur le bus de communication. Ainsi pour chaque octet d'une trame GPRMC, 10 bits sont transmis. Le signal de communication est bivalent, chaque bit est codé par un signal électrique haut ou bas dont la durée dépend du débit choisi pour la transmission. Au débit de 57600 bauds que l'on choisira ici, 57600 bits sont transmis chaque seconde.

Nous faisons l'hypothèse que la liaison est fiable, c'est-à-dire que l'on considère qu'il n'y a pas de panne du récepteur ou de déconnexion physique de la liaison série. On considère une trame de 76 caractères codés en ASCII. On donne en annexe 1 un exemple de trame.

1. Quelle est la durée de transmission d'une trame de 76 caractères sur le bus RS-232C ?
2. Le débit de transmission du bus série est-il suffisant pour traiter en temps réel les trames provenant des satellites ?

A.2 Lecture des trames GPS

On souhaite utiliser une bibliothèque permettant la gestion des entrées/sorties sur une liaison série. Un extrait de la spécification de cette bibliothèque est donné ci-après.

```
def initSerie(port,bauds):
    """ Initialise un port série et retourne un identifiant de port idport
    Entrées :
        * port : entier >= 0, le numéro de port à initialiser
        * bauds : entier, débit du port en bauds
    Sortie :
        * idport : entier, l'identifiant du port
    """

def lireSerie(idport):
    """ Attend l'arrivée d'un octet sur le port d'identifiant idport et retourne
        cet octet sous la forme d'un entier (code ASCII)
    Entrées :
        * idport : entier, l'identifiant du port série
    Sortie :
        * car : entier, valeur de l'octet lu sur le port série (code ASCII)
    """
```

3. Donner l'instruction permettant d'initialiser le port série numéroté « 0 » à une vitesse de 57600 bauds. L'identifiant du port sera stocké dans une variable nommée **identifiant**.
4. Indiquer quel est toujours le dernier caractère lu dans une trame ainsi que la valeur numérique de son code ASCII.
5. Donner la fonction standard Python qui permet d'obtenir un caractère à partir de la valeur numérique de son code ASCII.
6. Ecrire la fonction **lireTrame** qui prend en argument l'identifiant d'un port série, qui retourne la chaîne de caractères d'une trame lue sur le port, et qui l'affiche à l'écran.

A.3 Enregistrement dans un fichier

Nous souhaitons maintenant enregistrer toutes les trames envoyées par le récepteurs GPS dans un fichier texte nommé **tramesNMEA.txt** situé dans le dossier courant (une ligne par trame). Le fichier texte sera ouvert en début de programme de sorte à être vidé s'il existait déjà auparavant. Ici, le récepteur GPS envoyant des trames en continu, le programme à réaliser n'a pas de fin et ne se terminera que lorsqu'il sera explicitement fermé par l'utilisateur. Pour implémenter ce comportement, on pourra utiliser l'expression **while True**.

7. Ecrire le programme principal qui, après initialisation du port « 0 » de la liaison série, permet de lire les trames sur ce port et d'enregistrer le contenu des trames, telles qu'elles sont reçues, dans le fichier `tramesNMEA.txt`, au format texte.

Partie B : Exploitation du fichier de trames

L'objectif de cette partie est d'extraire et d'enregistrer une trame dans une structure de données cohérente afin de pouvoir exploiter aisément les mesures. On fait l'hypothèse que les trames ne contiennent pas d'erreur.

B.1 Extraction d'une trame

La description d'une trame GPRMC, donnée en annexe 1, indique que les valeurs des différentes informations de la trame sont comprises entre les caractères « \$ » et « * ».

8. Ecrire la fonction `cherchePremiereOccurrence` qui retourne l'indice de la première occurrence d'un caractère dans une chaîne de caractères, tous les deux passés en paramètre de la fonction. Par exemple l'appel `cherchePremiereOccurrence('a', 'capes')` retourne la valeur 1. Il conviendra de choisir et de justifier ce que la fonction retourne dans le cas où le caractère recherché n'est pas présent dans la chaîne.

On cherche maintenant à concevoir l'algorithme de la fonction `extraireTrame`. Cette fonction doit permettre d'extraire et de retourner, au format chaîne de caractères, le contenu d'une trame (nommée `NMEA`) complète située strictement entre les caractères « \$ » et « * ».

A titre d'exemple, le contenu de la trame `NMEA_extraite` issue de la fonction `extraireTrame`, basée sur la trame donnée en exemple dans l'annexe 1 est de la forme :

```
GPRMC,071139.988,A,4639.8232,N,0021.6810,E,8.95,184.14,141014,1.1,W,A
```

On précise que G est en première position de la trame extraite, c'est-à-dire en position 0 dans une chaîne en Python.

9. Indiquer quelles sont les conditions nécessaires et suffisantes que doit respecter une chaîne de caractères quelconque afin de pouvoir extraire une trame complète grâce à la fonction `extraireTrame`.

10. Ecrire la fonction `extraireTrame` dans laquelle la chaîne d'entrée se nomme `NMEA` et la chaîne de sortie se nomme `NMEA_extraite`. Dans le cas où elle ne peut pas être extraite, la fonction doit retourner une chaîne vide.

B.2 Structure de données

Dans le cadre des traitements ultérieurs que nous souhaitons effectuer, il nous faut représenter une mesure GPS, que nous nommerons par la suite `pointGPS`, par sa latitude, sa longitude et sa vitesse.

Dans la trame GPS, les longitudes et latitudes sont données en degrés et minutes. Cette représentation ne facilitant pas les calculs, on décide de les convertir en minutes (1 degré étant égal à 60 minutes). Par ailleurs on opte pour la convention suivante :

- pour la latitude, le signe positif est choisi pour le Nord
- pour la longitude, le signe positif est choisi pour l'Est

Ainsi par exemple, si la trame contient la latitude « 0314.2500 » suivie de l'indicateur « S », la séquence « 03 » correspond à 3 degrés soit 180 minutes d'angle, la séquence « 14.2500 » correspond à 14.25 minutes d'angle et l'indicateur « S » implique une latitude négative. On stockera donc la valeur -194.25 dans le `pointGPS`. Cette convention est assurée par la fonction `conversionLongLat2Min` qui prend en argument une chaîne de caractères sous la forme `ddmm.mmm` (latitude ou longitude) et un caractère (« N », « S », « W » ou « E »),

et qui renvoie une valeur réelle correspondant à la conversion en minutes d'angle expliquée ci-dessus. On supposera cette fonction connue et utilisable dans la suite de ce sujet.

Afin de stocker un `pointGPS`, on utilise une liste qui présente la structure interne suivante :

`pointGPS = [latitude, longitude, vitesse]`

Cette liste sera renseignée par les latitudes, longitudes et les vitesses extraites des différentes trames par le décodeur et traitées par les fonctions précédentes. Il existe deux principales méthodes pour extraire ces données, une première repose sur le comptage des séparateurs et peut s'adapter à des longueurs variables pour un même champ. La seconde, plus simple à mettre en œuvre, repose sur l'hypothèse que toutes les trames présentent exactement la même structure et que toutes les longueurs de champs restent identiques entre les différentes trames reçues. Dans la suite du sujet nous ferons l'hypothèse que toutes les longueurs de champ sont toujours identiques dans toutes les trames à traiter (celle de 76 caractères au total donnée en exemple).

11. A partir de la trame extraite issue de l'annexe 1 (donnée en section 2.1), compléter le tableau ci-dessous donnant l'indice de début et l'indice de fin des informations nécessaires à l'extraction de la latitude (y compris l'orientation), de la longitude (y compris l'orientation) et de la vitesse.

	Latitude	Orientation	Longitude	Orientation	Vitesse
Indice début					
Indice fin					

12. Ecrire la fonction `creationPointGPS` qui prend en argument la chaîne de caractères d'une trame NMEA extraite (i.e. issue de la fonction `extraireTrame`) et qui retourne un `pointGPS` contenant les informations de latitude, longitude et vitesse de la trame.

B.3 Affichage du fichier

Nous disposons maintenant de toutes les fonctions qui permettent d'exploiter le fichier de trames créé dans la première partie (`trameNMEA.txt`). Nous souhaitons à présent afficher successivement chacun des `pointGPS` présents dans le fichier à l'écran.

13. Le fichier `trameNMEA.txt` contient les éléments correspondant aux `pointGPS`. Ecrire un programme principal qui affiche à l'écran chacun de ces éléments. Chaque ligne devra être de la forme (où chaque espace correspond à une tabulation) :

latitude_Point1 longitude_Point1 vitesse_Point1

latitude_Point2 longitude_Point2 vitesse_Point2

Partie C : Structure de donnée avancée

L'objectif de cette partie est de stocker les points GPS dans un arbre afin de pouvoir accéder aux points ayant la plus grande vitesse de façon optimale.

Un **tas binaire** est une structure de données qui permet d'accéder au maximum (respectivement minimum) d'un ensemble de données en temps constant. On peut le représenter par un arbre binaire vérifiant deux contraintes :

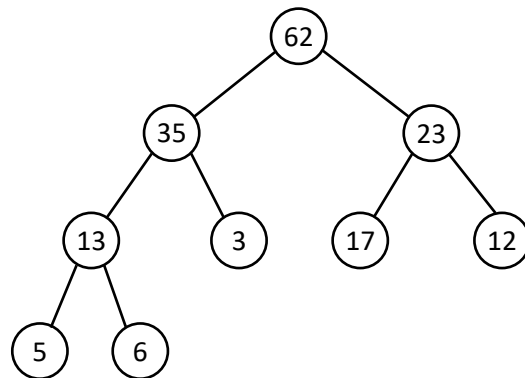
- c'est un arbre binaire parfait : tous les niveaux de l'arbre (excepté le niveau le plus bas) sont totalement remplis. Si le dernier n'est pas totalement rempli alors il doit l'être de gauche à droite
- c'est un tas : une clé est associée à chaque nœud de l'arbre. Cette dernière doit être supérieure (respectivement inférieure) ou égale aux clés de chacun de ses fils

Ainsi lorsque les clés sont des nombres (valeurs) et quand la relation d'ordre choisie est l'ordre naturel, on parle alors de tas-max (ou *max-heap*).

Un tas binaire étant un arbre binaire complet, on peut l'implémenter à l'aide d'un tableau tel que :

- la racine de l'arbre (niveau le plus haut) se trouve à l'indice 0
- en considérant un nœud à l'indice i :
 - son fils gauche se trouve à l'indice $2i + 1$
 - son fils droit se trouve à l'indice $2i + 2$
- en considérant un nœud à l'indice $i > 0$:
 - son père se trouve à l'indice $(i - 1)/2$, le symbole / désignant la division entière

La figure 1 illustre un tas binaire (max) et le tableau d'indices associés.

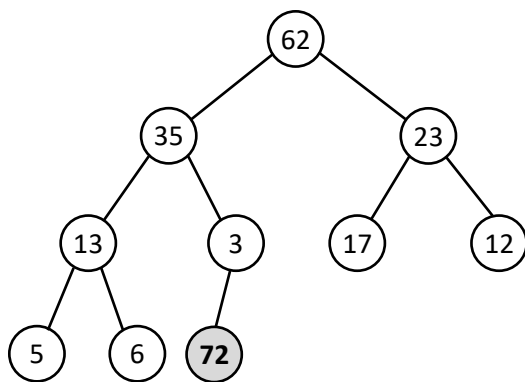


Valeur (clé)	62	35	23	13	3	17	12	5	6
Indice	0	1	2	3	4	5	6	7	8

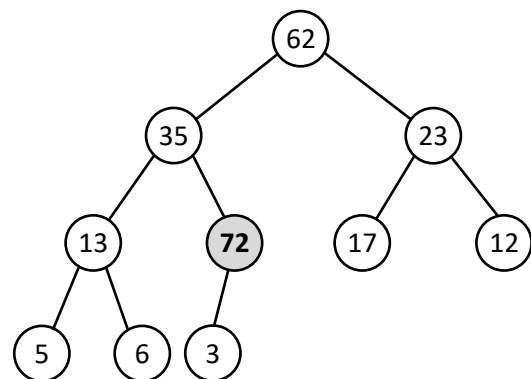
Le nœud ayant la valeur 62 est la racine de l'arbre (niveau le plus haut) et les nœuds ayant les valeurs 5 et 6 sont au niveau le plus bas.

Figure 1 : Tas binaire (max) et son implémentation en tableau

Dans un tas binaire, on insère l'élément à la prochaine position libre (la position libre la plus à gauche possible sur le dernier niveau) puis on effectue l'opération `retablirOrdreTasBinaire` pour rétablir si nécessaire la propriété d'ordre du tas binaire. Dans cette opération, tant que l'élément n'est pas la racine de l'arbre et que la clé de l'élément est strictement supérieure à son père, alors on échange les positions entre l'élément et son père. L'exemple de la figure 2 montre l'insertion d'un élément ayant une valeur 72 dans l'arbre de la figure 1.



Insertion de la clé 72



Echange de la clé 72 avec son père (clé 3)

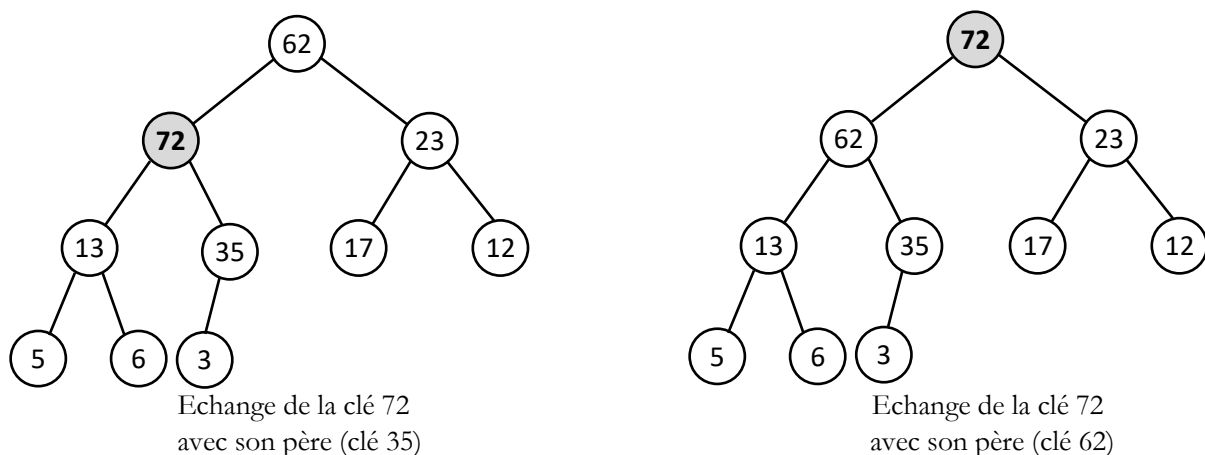


Figure 2 : Insertion de l'élément de clé 72

On souhaite stocker les `pointGPS` dans un tas binaire. La liste (i.e. le tas) sera donc de la forme :

```
[[latitude_0, longitude_0, vitesse_0], [latitude_1, longitude_1, vitesse_1], ...]
```

Quand un tas binaire est réalisé sur un ensemble structuré de variables comme c'est le cas sur les variables de type `pointGPS`, un des paramètres de cette variable structurée devient la clé. Les comparaisons présentes dans la fonction `retablirOrdreTasBinaire` se font alors sur cette clé.

14. Ecrire la fonction `retablirOrdreTasBinaire` qui prend en argument le tas binaire de points GPS dont l'ordre des clés doit être rétabli après l'insertion d'un élément en première position libre. Le paramètre vitesse jouera le rôle de clé.

15. Ecrire une fonction `insérerPointGPS` qui ajoute un point GPS dans un tas binaire de points. Tous les deux seront passés en paramètre de la fonction.

16. Ecrire une fonction `extraireMax` qui retourne le `pointGPS` de plus grande vitesse dans un tas binaire de points GPS passé en paramètre. Donner la complexité de cette fonction en fonction du nombre de points GPS dans le tas.

17. *Question bonus.* Ecrire la classe `Parcours` définie par :

- un constructeur initialisant une instance à partir du nom d'un fichier de trames (ex. `tramesNMEA.txt`)
 - deux attributs `listePointsGPS` et `tasPointsGPS` stockant les points GPS de la trame lue d'une part sous forme d'une liste où les points sont triés dans le même ordre que la trame (parcours chronologique), et d'autre part sous forme d'un tas binaire où les clés sont les vitesses
 - une méthode `afficherPointsChrono` sans paramètre permettant d'afficher les points GPS dans l'ordre chronologique (un point par ligne, même format que la question 13)
 - une méthode `afficherPointsVitesseMinMax` sans paramètre permettant d'afficher le point GPS de vitesse minimale et le point de vitesse maximale (un point par ligne, même format que la question 13)
 - toutes autres méthodes internes que vous jugerez utiles
-

Annexe 1 – Le protocole NMEA 0183, type GPRMC

Dans le protocole NMEA 0183, chaque trame est constituée de chaînes de caractères, terminées par un retour chariot et un saut de ligne. Les caractères constituant la trame ont un code ASCII compris entre 20 et 127 mais sont codés sur un octet. Toute trame commence par les caractères \$GP, suivis de trois caractères définissant le type de trame. Chaque trame de type GPRMC est donc préfixée par \$GPRMC.

On donne ci-dessous un exemple de trame :

```
$GPRMC,071139.988,A,4639.8232,N,0021.6810,E,8.95,184.14,141014,1.1,W,A*65
```

Les trames sont constituées de 13 champs texte, séparés par des virgules et terminés par une somme de contrôle de parité, résultat d'un « ou exclusif » de chaque octet compris strictement entre le caractère initial \$ et le caractère *. L'interprétation de chaque champ dépend de sa position.

Position	Exemple	Interprétation
0	\$GPRMC	Type de trame. Nous considérons uniquement le type RMC.
1	071139.988	Heure UTC de la mesure sous forme hhmmss.sss (avec hh heures, mm minutes et ss.sss secondes et secondes décimales).
2	A	Caractère 'A' pour mesure valide, 'V' pour mesure invalide
3	4639.8232	Latitude donnée sous la forme ddm.mmm (avec dd degrés, mm.mmm minutes et minutes décimales d'angle).
4	N	Complète le champ 3 pour préciser s'il s'agit de la latitude Nord ('N') ou Sud ('S').
5	0021.6810	Longitude donnée sous la forme ddm.mmm
6	E	Complète le champ 5 pour préciser s'il s'agit de la longitude Est ('E') ou Ouest ('W').
7	8.95	Vitesse en nœuds.
8	184.14	Cap du déplacement par rapport au pôle Nord magnétique en degré décimaux.
9	141014	Date universelle sous la forme jjmmaa (avec jj jour du mois, mm mois, et 20aa année).
10	1.1	Variation magnétique sous la forme d.d en degrés décimaux.
11	W	Sens de la variation magnétique ('E' = Est, 'W' = Ouest).
12	A *65	Mode de fonctionnement de récepteur. Somme de contrôle exprimée en base hexadécimale sur deux chiffres.
fin	<CR><LF>	La trame se termine par deux caractères : un retour chariot (code ASCII 13) puis un saut de ligne (code ASCII 10).