

## *En guise de brève introduction*

*50 kilos de patates, un sac de sciure de bois, il te sortait  
25 litres de 3 étoiles à l'alambic. Un vrai magicien ce  
Jo. Et c'est pour ça que je me permets... .*

MICHEL AUDIARD – *Les tontons flingueurs*

Voici le recueil de la plupart des sujets d'informatique — ou comportant au moins une question d'informatique — posés aux concours des grandes écoles scientifiques en 2017 et disponibles actuellement.

Le recueil est partagé en trois grandes parties. D'abord, les épreuves de l'option info de nos classes, auxquels s'ajoutent les épreuves d'informatique des seconds concours des ENS. Ensuite, les épreuves d'informatique commune. Enfin, les épreuves des disciplines autres que l'informatique (mathématiques, modélisation, sciences de l'ingénieur) comportant au moins une question d'informatique.

Comme l'an dernier, le bulletin des concours Informatique ne sera proposé aux adhérents que sous forme électronique.

Dans ce recueil, le nom du fichier contenant chaque sujet figure dans la table des matières et en tête de chaque page. Les fichiers sont disponibles sur le site de l'UPS, soit à l'adresse <http://prepas.org/ups.php?module=Maths&voir=recherche>, soit dans la rubrique Ressources / Informatique <http://prepas.org/ups.php?rubrique=146> (pour les sujets de modélisation ou de sciences de l'ingénieur).

Vous trouverez ce recueil, au format pdf, sur le site de l'UPS, à l'adresse <http://prepas.org/ups.php?rubrique=146>. Les ajouts d'énoncés y seront signalés en allant.

Antoine Pichoff    [antoine.pichoff@prepas.org](mailto:antoine.pichoff@prepas.org)  
Philippe Patte    [philippe.patte@prepas.org](mailto:philippe.patte@prepas.org)

*Juillet 2017*



# ÉCOLE POLYTECHNIQUE — ÉCOLES NORMALES SUPÉRIEURES

CONCOURS D'ADMISSION 2017

FILIÈRES MP SPECIALITÉ INFO

## COMPOSITION D'INFORMATIQUE – A – (XULCR)

(Durée : 4 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.

Le langage de programmation sera **obligatoirement Caml**.

\*  
\* \*

## Jeux à un joueur et solutions optimales

Nous nous intéressons ici à des jeux à un joueur où une configuration initiale est donnée et où le joueur effectue une série de déplacements pour parvenir à une configuration gagnante. Des casse-tête tels que le *Rubik's Cube*, le solitaire, l'âne rouge ou encore le taquin entrent dans cette catégorie. L'objectif de ce problème est d'étudier différents algorithmes pour trouver des solutions à de tels jeux qui minimisent le nombre de déplacements effectués.

La partie I introduit la notion de jeu à un joueur et un premier algorithme pour trouver une solution optimale. Les parties II et III proposent d'autres algorithmes, plus efficaces. La partie IV a pour objectif de trouver une solution optimale au jeu du taquin.

Les parties I, II et III peuvent être traitées indépendamment. Néanmoins, chaque partie utilise des notations et des fonctions introduites dans les parties précédentes. La partie IV suppose qu'on a lu entièrement l'énoncé de la partie III.

### Complexité

Par *complexité en temps* d'un algorithme  $A$  on entend le nombre d'opérations élémentaires (comparaison, addition, soustraction, multiplication, division, affectation, test, etc) nécessaires à l'exécution de  $A$  dans le cas le pire. Par *complexité en espace* d'un algorithme  $A$  on entend l'espace mémoire minimal nécessaire à l'exécution de  $A$  dans le cas le pire. Lorsque la complexité en temps ou en espace dépend d'un ou plusieurs paramètres  $\kappa_0, \dots, \kappa_{r-1}$ , on dit que  $A$  a une complexité en  $\mathcal{O}(f(\kappa_0, \dots, \kappa_{r-1}))$  s'il existe une constante  $C > 0$  telle que, pour toutes les valeurs de  $\kappa_0, \dots, \kappa_{r-1}$  suffisamment grandes (c'est-à-dire plus grandes qu'un certain seuil), pour toute instance du problème de paramètres  $\kappa_0, \dots, \kappa_{r-1}$ , la complexité est au plus  $C f(\kappa_0, \dots, \kappa_{r-1})$ .

```

BFS()
   $A \leftarrow \{e_0\}$ 
   $p \leftarrow 0$ 
  tant que  $A \neq \emptyset$ 
     $B \leftarrow \emptyset$ 
    pour tout  $x \in A$ 
      si  $x \in F$  alors
        renvoyer VRAI
       $B \leftarrow s(x) \cup B$ 
     $A \leftarrow B$ 
     $p \leftarrow p + 1$ 
  renvoyer FAUX

```

FIGURE 1 – Parcours en largeur.

## Partie I. Jeu à un joueur, parcours en largeur

Un jeu à un joueur est la donnée d'un ensemble non vide  $E$ , d'un élément  $e_0 \in E$ , d'une fonction  $s : E \rightarrow \mathcal{P}(E)$  et d'un sous-ensemble  $F$  de  $E$ . L'ensemble  $E$  représente les états possibles du jeu. L'élément  $e_0$  est l'état initial. Pour un état  $e$ , l'ensemble  $s(e)$  représente tous les états atteignables en un coup à partir de  $e$ . Enfin,  $F$  est l'ensemble des états gagnants du jeu. On dit qu'un état  $e_p$  est à la *profondeur*  $p$  s'il existe une séquence finie de  $p + 1$  états

$$e_0 \ e_1 \ \dots \ e_p$$

avec  $e_{i+1} \in s(e_i)$  pour tout  $0 \leq i < p$ . Si par ailleurs  $e_p \in F$ , une telle séquence est appelée une *solution* du jeu, de profondeur  $p$ . Une solution *optimale* est une solution de profondeur minimale. On notera qu'un même état peut être à plusieurs profondeurs différentes.

Voici un exemple de jeu :

$$\begin{aligned} E &= \mathbb{N}^* \\ e_0 &= 1 \\ s(n) &= \{2n, n+1\} \end{aligned} \tag{1}$$

**Question 1.** Donner une solution optimale pour ce jeu lorsque  $F = \{42\}$ .

**Parcours en largeur.** Pour chercher une solution optimale pour un jeu quelconque, on peut utiliser un parcours en largeur. Un pseudo-code pour un tel parcours est donné figure 1.

**Question 2.** Montrer que le parcours en largeur renvoie VRAI si et seulement si une solution existe.

**Question 3.** On se place dans le cas particulier du jeu (1) pour un ensemble  $F$  arbitraire pour lequel le parcours en largeur de la figure 1 termine. Montrer alors que la complexité en temps et en espace est exponentielle en la profondeur  $p$  de la solution trouvée. On demande de montrer que la complexité est bornée à la fois inférieurement et supérieurement par deux fonctions exponentielles en  $p$ .

```

DFS( $m, e, p$ )
  si  $p > m$  alors
    renvoyer FAUX
  si  $e \in F$  alors
    renvoyer VRAI
  pour chaque  $x$  dans  $s(e)$ 
    si  $\text{DFS}(m, x, p + 1) = \text{VRAI}$  alors
      renvoyer VRAI
    renvoyer FAUX

```

FIGURE 2 – Parcours en profondeur (partie II), limité par une profondeur maximale  $m$ .

**Programmation.** Dans la suite, on suppose donnés un type `etat` et les valeurs suivantes pour représenter un jeu en Caml :

```

initial: etat
suivants: etat -> etat list
final: etat -> bool

```

**Question 4.** Écrire une fonction `bfs: unit -> int` qui effectue un parcours en largeur à partir de l'état initial et renvoie la profondeur de la première solution trouvée. Lorsqu'il n'y a pas de solution, le comportement de cette fonction pourra être arbitraire.

Indication : On pourra avantageusement réaliser les ensembles  $A$  et  $B$  par des listes, sans chercher à éliminer les doublons, et utiliser une fonction récursive plutôt qu'une boucle `while`.

**Question 5.** Montrer que la fonction `bfs` renvoie toujours une profondeur optimale lorsqu'une solution existe.

## Partie II. Parcours en profondeur

Comme on vient de le montrer, l'algorithme BFS permet de trouver une solution optimale mais il peut consommer un espace important pour cela, comme illustré dans le cas particulier du jeu (1) qui nécessite un espace exponentiel. On peut y remédier en utilisant plutôt un parcours en profondeur. La figure 2 contient le pseudo-code d'une fonction DFS effectuant un parcours en profondeur à partir d'un état  $e$  de profondeur  $p$ , sans dépasser une profondeur maximale  $m$  donnée.

**Question 6.** Montrer que  $\text{DFS}(m, e_0, 0)$  renvoie VRAI si et seulement si une solution de profondeur inférieure ou égale à  $m$  existe.

**Recherche itérée en profondeur.** Pour trouver une solution optimale, une idée simple consiste à effectuer un parcours en profondeur avec  $m = 0$ , puis avec  $m = 1$ , puis avec  $m = 2$ , etc., jusqu'à ce que  $\text{DFS}(m, e_0, 0)$  renvoie VRAI.

**Question 7.** Écrire une fonction `ids: unit -> int` qui effectue une recherche itérée en profondeur et renvoie la profondeur d'une solution optimale. Lorsqu'il n'y a pas de solution, cette fonction ne termine pas.

**Question 8.** Montrer que la fonction `ids` renvoie toujours une profondeur optimale lorsqu'une solution existe.

**Question 9.** Comparer les complexités en temps et en espace du parcours en largeur et de la recherche itérée en profondeur dans les deux cas particuliers suivants :

1. il y a exactement un état à chaque profondeur  $p$ ;
2. il y a exactement  $2^p$  états à la profondeur  $p$ .

On demande de justifier les complexités qui seront données.

### Partie III. Parcours en profondeur avec horizon

On peut améliorer encore la recherche d'une solution optimale en évitant de considérer successivement toutes les profondeurs possibles. L'idée consiste à introduire une fonction  $h : E \rightarrow \mathbb{N}$  qui, pour chaque état, donne un minorant du nombre de coups restant à jouer avant de trouver une solution. Lorsqu'un état ne permet pas d'atteindre une solution, cette fonction peut renvoyer n'importe quelle valeur.

Commençons par définir la notion de *distance* entre deux états. S'il existe une séquence de  $k + 1$  états  $x_0 x_1 \dots x_k$  avec  $x_{i+1} \in s(x_i)$  pour tout  $0 \leq i < k$ , on dit qu'il y a un chemin de longueur  $k$  entre  $x_0$  et  $x_k$ . Si de plus  $k$  est minimal, on dit que la distance entre  $x_0$  et  $x_k$  est  $k$ .

On dit alors que la fonction  $h$  est *admissible* si elle ne surestime jamais la distance entre un état et une solution, c'est-à-dire que pour tout état  $e$ , il n'existe pas d'état  $f \in F$  situé à une distance de  $e$  strictement inférieure à  $h(e)$ .

On procède alors comme pour la recherche itérée en profondeur, mais pour chaque état  $e$  considéré à la profondeur  $p$  on s'interrompt dès que  $p + h(e)$  dépasse la profondeur maximale  $m$  (au lieu de s'arrêter simplement lorsque  $p > m$ ). Initialement, on fixe  $m$  à  $h(e_0)$ . Après chaque parcours en profondeur infructueux, on donne à  $m$  la plus petite valeur  $p + h(e)$  qui a dépassé  $m$  pendant ce parcours, le cas échéant, pour l'ensemble des états  $e$  rencontrés dans ce parcours. La figure 3 donne le pseudo-code d'un tel algorithme, appelé IDA\*, où la variable globale `min` est utilisée pour retenir la plus petite valeur ayant dépassé  $m$ .

**Question 10.** Écrire une fonction `idastar: unit -> int` qui réalise l'algorithme IDA\* et renvoie la profondeur de la première solution rencontrée. Lorsqu'il n'y a pas de solution, le comportement de cette fonction pourra être arbitraire. Il est suggéré de décomposer le code en plusieurs fonctions. On utilisera une référence globale `min` dont on précisera le type et la valeur retenue pour représenter  $\infty$ .

**Question 11.** Proposer une fonction  $h$  admissible pour le jeu (1), non constante, en supposant que l'ensemble  $F$  est un singleton  $\{t\}$  avec  $t \in \mathbb{N}^*$ . On demande de justifier que  $h$  est admissible.

**Question 12.** Montrer que, si la fonction  $h$  est admissible, la fonction `idastar` renvoie toujours une profondeur optimale lorsqu'une solution existe.

$\text{DFS}^*(m, e, p) \stackrel{\text{def}}{=} c \leftarrow p + h(e)$ <b>si</b> $c > m$ <b>alors</b> <b>si</b> $c < \min$ <b>alors</b> $\min \leftarrow c$ <b>renvoyer</b> FAUX <b>si</b> $e \in F$ <b>alors</b> <b>renvoyer</b> VRAI <b>pour chaque</b> $x$ <b>dans</b> $s(e)$ <b>si</b> $\text{DFS}^*(m, x, p+1) = \text{VRAI}$ <b>alors</b> <b>renvoyer</b> VRAI <b>renvoyer</b> FAUX	$\text{IDA}^*() \stackrel{\text{def}}{=} m \leftarrow h(e_0)$ <b>tant que</b> $m \neq \infty$ $\min \leftarrow \infty$ <b>si</b> $\text{DFS}^*(m, e_0, 0) = \text{VRAI}$ <b>alors</b> <b>renvoyer</b> VRAI $m \leftarrow \min$ <b>renvoyer</b> FAUX
---	---

FIGURE 3 – Pseudo-code de l’algorithme IDA\*.

## Partie IV. Application au jeu du taquin

Le jeu du taquin est constitué d’une grille  $4 \times 4$  dans laquelle sont disposés les entiers de 0 à 14, une case étant laissée libre. Dans tout ce qui suit, les lignes et les colonnes sont numérotées de 0 à 3, les lignes étant numérotées du haut vers le bas et les colonnes de la gauche vers la droite. Voici un état initial possible :

	0	1	2	3
0	2	3	1	6
1	14	5	8	4
2		12	7	9
3	10	13	11	0

On obtient un nouvel état du jeu en déplaçant dans la case libre le contenu de la case située au-dessus, à gauche, en dessous ou à droite, au choix. Si on déplace par exemple le contenu de la case située à droite de la case libre, c’est-à-dire 12, on obtient le nouvel état suivant :

2	3	1	6
14	5	8	4
12		7	9
10	13	11	0

Le but du jeu du taquin est de parvenir à l’état final suivant :

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

Ici, on peut le faire à l'aide de 49 déplacements supplémentaires et il n'est pas possible de faire moins.

**Question 13.** En estimant le nombre d'états du jeu du taquin, et la place mémoire nécessaire à la représentation d'un état, expliquer pourquoi il n'est pas réaliste d'envisager le parcours en largeur de la figure 1 pour chercher une solution optimale du taquin.

**Fonction  $h$ .** On se propose d'utiliser l'algorithme IDA\* pour trouver une solution optimale du taquin et il faut donc choisir une fonction  $h$ . On repère une case de la grille par sa ligne  $i$  (avec  $0 \leq i \leq 3$ , de haut en bas) et sa colonne  $j$  (avec  $0 \leq j \leq 3$ , de gauche à droite). Si  $e$  est un état du taquin et  $v$  un entier entre 0 et 14, on note  $e_v^i$  la ligne de l'entier  $v$  dans  $e$  et  $e_v^j$  la colonne de l'entier  $v$  dans  $e$ . On définit alors une fonction  $h$  pour le taquin de la façon suivante :

$$h(e) = \sum_{v=0}^{14} |e_v^i - \lfloor v/4 \rfloor| + |e_v^j - (v \bmod 4)|.$$

**Question 14.** Montrer que cette fonction  $h$  est admissible.

**Programmation.** Pour programmer le jeu de taquin, on abandonne l'idée d'un type `etat` et d'une fonction `suivants`, au profit d'un unique état global et modifiable. On se donne pour cela une matrice `grid` de taille  $4 \times 4$  contenant l'état courant  $e$  ainsi qu'une référence `h` contenant la valeur de  $h(e)$ .

```
grid: int vect vect
h: int ref
```

La matrice `grid` est indexée d'abord par  $i$  puis par  $j$ . Ainsi `grid.(i).(j)` est l'entier situé à la ligne  $i$  et à la colonne  $j$ . Par ailleurs, la position de la case libre est maintenue par deux références :

```
li: int ref
lj: int ref
```

La valeur contenue dans `grid` à cette position est non significative.

**Question 15.** Écrire une fonction `move: int -> int -> unit` telle que `move i j` déplace l'entier situé dans la case  $(i, j)$  vers la case libre supposée être adjacente. On prendra soin de bien mettre à jour les références `h`, `li` et `lj`. On ne demande pas de vérifier que la case libre est effectivement adjacente.

**Déplacements et solution.** De cette fonction `move`, on déduit facilement quatre fonctions qui déplacent un entier vers la case libre, respectivement vers le haut, la gauche, le bas et la droite.

```
let haut    () = move (!li + 1) !lj;;
let gauche () = move !li (!lj + 1);;
let bas    () = move (!li - 1) !lj;;
let droite () = move !li (!lj - 1);;
```

Ces quatre fonctions supposent que le mouvement est possible.

Pour conserver la solution du taquin, on se donne un type `déplacement` pour représenter les quatre mouvements possibles et une référence globale `solution` contenant la liste des déplacements qui ont été faits jusqu'à présent.

```
type déplacement = Gauche | Bas | Droite | Haut
solution: déplacement list ref
```

La liste `!solution` contient les déplacement effectués dans l'ordre inverse, *i.e.*, la tête de liste est le déplacement le plus récent.

**Question 16.** Écrire une fonction `tente_gauche: unit -> bool` qui tente d'effectuer un déplacement vers la gauche si cela est possible et si le dernier déplacement effectué, le cas échéant, n'était pas un déplacement vers la droite. Le booléen renvoyé indique si le déplacement vers la gauche a été effectué. On veillera à mettre à jour `solution` lorsque c'est nécessaire.

On suppose avoir écrit de même trois autres fonctions

```
tente_bas   : unit -> bool
tente_droite: unit -> bool
tente_haut  : unit -> bool
```

**Question 17.** Écrire une fonction `dfs: int -> int -> bool` correspondant à la fonction DFS\* de la figure 3 pour le jeu du taquin. Elle prend en argument la profondeur maximale  $m$  et la profondeur courante  $p$ . (L'état  $e$  est maintenant global.)

**Question 18.** En déduire enfin une fonction `taquin: unit -> déplacement list` qui renvoie une solution optimale, lorsqu'une solution existe.

*Note : Trouver une solution au taquin n'est pas très compliqué, mais trouver une solution optimale est nettement plus difficile. Avec ce qui est proposé dans la dernière partie de ce sujet, on y parvient en moins d'une minute pour la plupart des états initiaux et en une dizaine de minutes pour les problèmes les plus difficiles.*

```
*  *
*
```

**ÉCOLES NORMALES SUPÉRIEURES****CONCOURS D'ADMISSION 2017****FILIÈRE MP – CONCOURS INFO****COMPOSITION D'INFORMATIQUE-MATHÉMATIQUES – (ULCR)**

(Durée : 4 heures)

L'utilisation des calculatrices n'est pas autorisée pour cette épreuve.

Ce sujet comprend 10 pages numérotées de 1 à 10.

\* \* \*

**Détection de carrés dans les mots**

Ce sujet traite de la recherche de répétitions dans un texte. Plus particulièrement, on s'intéresse aux répétitions appelées **carrés**.

La partie I introduit le problème de la recherche de répétitions dans un texte et donne un premier algorithme permettant de le résoudre. La partie II est consacrée à la construction d'un mot infini sans carré. Enfin, les parties III et IV sont consacrées à la mise au point d'un algorithme efficace de détection de carrés.

La **complexité**, ou le **coût**, d'un programme  $P$  est le nombre d'opérations élémentaires (addition, soustraction, affectation, test, lecture ou écriture dans un tableau, etc...) nécessaires à l'exécution de  $P$  dans le cas le pire. Lorsque cette complexité dépend d'un ou plusieurs paramètres  $n_1, \dots, n_k$ , on dira que  $P$  a une **complexité en**  $\mathcal{O}(f(n_1, \dots, n_k))$ , s'il existe une constante  $K > 0$  telle que, pour toutes les valeurs de  $n_1, \dots, n_k$  suffisamment grandes (c'est-à-dire plus grandes qu'un certain seuil), pour toute instance du problème de paramètres  $n_1, \dots, n_k$ , la complexité de  $P$  est au plus  $Kf(n_1, \dots, n_k)$ . Lorsqu'il est demandé de garantir une certaine complexité, le candidat devra justifier cette dernière.

On considère un **alphabet**  $\Sigma$ , c'est-à-dire un ensemble fini d'éléments appelés **lettres**. On appelle  $\Sigma^*$  l'ensemble de tous les mots de longueur finie utilisant les lettres de l'alphabet  $\Sigma$ . La **longueur** d'un mot  $x$ , c'est-à-dire le nombre de lettres qui le composent, est notée  $|x|$ . Le **mot vide**, de longueur 0 et noté  $\epsilon$ , appartient également à  $\Sigma^*$ . Le mot obtenu par **concaténation** de deux mots  $x$  et  $y$ , noté  $xy$ , a pour longueur  $|x| + |y|$  et est composé des lettres de  $x$ , suivies des lettres de  $y$ .

Un **carré** est un mot égal à  $xx$  où  $x$  est un mot non vide. On dit qu'un mot  $m$  **contient un carré** s'il existe un mot non vide  $x$  et deux mots (éventuellement vides)  $y$  et  $z$  tels que :

$$m = yxxz$$

On dit alors que  $xx$  est un **carré de  $m$** . La **période** d'un carré  $xx$  est la longueur du mot  $x$ . Par exemple, le mot *bonbon* est un carré de période 3, et le mot *repetition* contient le carré *titi* de période 2.

On choisit de représenter les mots par des tableaux de caractères. Les cases d'un tableau de longueur  $n$  sont numérotées de 0 à  $n - 1$ . Pour des raisons de cohérence entre les notations mathématiques et les représentations en machine, on note  $x_0, x_1, \dots, x_{|x|-1}$  les lettres qui composent le mot  $x$  de  $\Sigma^*$ . On a donc pour tout mot non vide  $x$  :

$$x = x_0 \dots x_{|x|-1}$$

On dit aussi que  $x_i$  est la lettre qui apparaît à la **position  $i$  du mot  $x$** . La numérotation des lettres d'un mot permet de parler de la **position à laquelle un carré apparaît dans un mot**. Par exemple, le mot *tintinnabuler* est représenté par le tableau décrit ci-dessous.

Numéro de la case	0	1	2	3	4	5	6	7	8	9	10	11	12
Caractère contenu dans la case	<i>t</i>	<i>i</i>	<i>n</i>	<i>t</i>	<i>i</i>	<i>n</i>	<i>n</i>	<i>a</i>	<i>b</i>	<i>u</i>	<i>l</i>	<i>e</i>	<i>r</i>

On dit que le mot (ou le tableau qui le représente) contient le carré *tintin* de période 3, à la position 0. En effet, la première lettre de *tintin* apparaît à la case 0 du tableau. Le mot contient aussi le carré *nn* de période 1, à la position 5.

Le sujet de ce devoir s'appuie sur le langage informatique Caml Light. Le candidat pourra rédiger ses réponses dans ce langage, ou dans un format pseudo-code proche. On rappelle quelques fonctions Caml Light pour manipuler les tableaux.

- **vect\_length**  $\text{m}$  renvoie le nombre de cases du tableau  $\text{m}$ .
- **sub\_vect**  $\text{m} \text{ x} \text{ y}$  renvoie le sous-tableau de  $\text{m}$  constitué des  $\text{y}$  cases à partir de la case  $\text{x}$ .
- **make\_vect**  $\text{x} \text{ y}$  renvoie un tableau de  $\text{x}$  cases qui contiennent toutes la valeur  $\text{y}$ .

```
(* Caml *)  vect_length : 'a vect -> int
(* Caml *)  sub_vect : 'a vect -> int -> int -> 'a vect
(* Caml *)  make_vect : int -> 'a -> 'a vect
```

## Partie I. Existence d'un carré

**Question 1** On considère le programme suivant :

```
let carre_pe_pos m p i =
  let j = ref 0 in
  while !j < p &&
    i + !j + p < (vect_length m) &&
    m.(i + !j + p) = m.(i + !j)
  do
    j := !j + 1
  done;
  !j;;
```

---

```
(* Caml *)  carre_pe_pos :char vect -> int -> int -> int
```

---

Donner les valeurs des appels suivants.

```
carre_pe_pos [|'c'; 'o'; 'u'; 'c'; 'o'; 'u'|] 3 0;;
carre_pe_pos [|'c'; 'o'; 'u'; 'c'; 'o'; 'o'|] 3 0;;
carre_pe_pos [|'a'; 'b'; 'a'; 'b'; 'a'; 'b'; 'a'; 'b'|] 2 4;;
carre_pe_pos [|'a'; 'b'; 'a'; 'b'; 'a'; 'b'; 'a'; 'b'|] 2 5;;
```

**Question 2** On suppose  $p > 0$  et  $i \geq 0$ . Définir précisément ce que renvoie la fonction `carre_pe_pos` en fonction de ses arguments  $m$ ,  $p$  et  $i$ . Que signifie le fait que la valeur renvoyée est égale à  $p$  ?

**Question 3** Écrire une fonction `carre_pe` qui prend en entrée un mot  $m$  et un entier  $p > 0$ , et qui renvoie `true` si le mot  $m$  contient un carré de période  $p$  et `false` sinon.

---

```
(* Caml *)  carre_pe :char vect -> int -> bool
```

---

**Question 4** Écrire une fonction `carre_naif` qui prend en entrée un mot  $m$  et renvoie `true` si le mot contient un carré de longueur quelconque, et `false` sinon.

---

```
(* Caml *)  carre_naif :char vect -> bool
```

---

**Question 5** Dans cette question, on suppose que pour l'alphabet considéré, pour tout entier  $\ell \geq 1$ , il existe des mots de longueurs  $\ell$  sans carré. Donner alors la complexité dans le pire des cas de la fonction `carre_naif` définie précédemment.

**Question 6** On considère l'alphabet  $\Sigma = \{a, b\}$ . Montrer que tout mot suffisamment long contient un carré. Donner un algorithme efficace (s'exécutant en temps constant) permettant de décider l'existence d'un carré pour les mots  $m$  construits sur l'alphabet  $\Sigma = \{a, b\}$ . On pourra supposer ici que la fonction `vect_length` s'exécute en temps constant.

## Partie II. Construction d'un mot infini sans carré

Le but de cette partie est de construire un mot sur un alphabet de quatre lettres, de longueur infinie, et sans carré. La notion de carré s'étend naturellement aux mots infinis. Un mot infini est une suite de lettres indiquées par les entiers de  $\mathbb{N}$ . Le mot obtenu par **concaténation** d'un mot fini  $x$  et d'un mot infini  $y$  est le mot infini  $xy$  composé des lettres de  $x$ , suivies des lettres de  $y$ . Dans ce contexte, un **carré** reste un mot fini de la forme  $xx$ , avec  $x$  un mot non vide. Un mot infini  $m$  **contient un carré** s'il existe un mot fini non vide  $x$ , un mot fini  $y$  (éventuellement vide) et un mot infini  $z$  tels que  $m = yxz$ . Un **bit** est un nombre entier égal à 0 ou 1. Soit  $(b_i)_{i \in \mathbb{N}}$  une suite de bits tous nuls à partir d'un certain rang (c'est-à-dire qu'il existe  $N$  tel que pour tout  $i \geq N$ ,  $b_i = 0$ ). On dit que  $(b_i)_{i \in \mathbb{N}}$  est la **représentation binaire** d'un nombre entier naturel  $n$  si

$$n = \sum_{i=0}^{+\infty} b_i 2^i.$$

On rappelle que tout nombre entier naturel possède une unique représentation binaire. On rappelle qu'en pratique, pour donner la représentation binaire d'un nombre entier dont les bits sont tous nuls à partir du rang  $N$ , on écrit simplement  $b_{N-1}b_{N-2}\dots b_0$ .

Soit  $n$  un nombre entier naturel et  $(b_i)_{i \in \mathbb{N}}$  sa représentation binaire. On note  $p(n)$  la **profondeur** de  $n$ , à savoir le plus petit entier  $i$  tel que  $b_i = 0$ . On note  $c(n) = b_{p(n)+1}$ , le  $(p(n)+1)$ -ème bit de la représentation binaire de  $n$ . On étudie le mot infini  $M$  sur l'alphabet  $\{0, 1\}$  défini par

$$M = c(0)c(1)c(2)c(3)\dots$$

**Question 7** Donner les huit premières lettres de  $M$  et vérifier que  $M$  contient des carrés de période 1 et 3.

**Question 8** Soient  $d \geq 0$  et  $k \geq 1$  deux nombres entiers. On définit  $i(d, k)$  comme l'unique nombre entier de l'ensemble  $\{d, d+1, \dots, d+2^k-1\}$  égal à  $2^{k-1}-1$  modulo  $2^k$ . Soit  $n$  un nombre entier naturel. En comparant les représentations binaires de  $n$  et de  $n$  modulo  $2^k$ , montrer que  $p(i(d, k)) = k - 1$ .

**Question 9** Montrer que pour tous entiers  $d \geq 0$  et  $k \geq 1$ ,

$$c(i(d, k) + 2^k) = c(i(d, k)) + 1 \mod 2.$$

**Question 10** Montrer que le mot  $M$  ne contient aucun carré de période  $p$  avec  $p = 2^k$  et  $k \geq 1$ .

**Question 11** Généraliser le raisonnement précédent pour en déduire que le mot  $M$  ne contient aucun carré de période paire.

**Question 12** Définir un alphabet sur quatre lettres et donner, sur cet alphabet, un mot infini, c'est-à-dire une suite de lettres indicée par les entiers de  $\mathbb{N}$ , qui ne contient aucun carré.

*Il existe également des mots infinis sur trois lettres sans carré, mais leur construction n'est pas demandée dans ce devoir. Dans la suite, on ne considère que des mots finis.*

### Partie III. Recherche efficace des carrés

Nous souhaitons mettre en place un algorithme efficace permettant la recherche de carrés dans un mot. Dans cette partie, nous nous intéressons à la détection de carrés créés lors de la concaténation de deux mots. Il s'en déduit un algorithme de test d'existence d'un carré dans un mot par la stratégie "diviser pour régner".

À titre d'exemple, considérons les mots  $u = \text{tin}$  et  $v = \text{tinnabuler}$ . Le carré *tintin* apparaissant au début du mot  $uv$  est créé lors de la concaténation des mots  $u$  et  $v$  alors que le carré *nn* apparaissant dans le mot  $uv$  n'est pas créé lors de la concaténation de  $u$  et de  $v$ . Il apparaissait déjà dans le mot  $v$ .

Un mot  $y$  est un **préfixe** d'un mot  $x$  s'il existe un mot  $z$  tel que  $x = yz$ . Un mot  $y$  est un **suffixe** d'un mot  $x$  s'il existe un mot  $z$  tel que  $x = zy$ . Noter que le mot vide est préfixe et suffixe de n'importe quel mot.

Pour tous mots  $u$  et  $v$  sur un alphabet  $\Sigma$  et tout indice  $i$  tel que  $0 \leq i \leq |v| - 1$ , on définit :

1.  $\text{lms}(u, v, i)$  est la longueur maximum d'un suffixe de  $u$  qui apparaît dans  $v$  en se terminant à la position  $i$  de  $v$ . Par exemple,  $\text{lms}(\text{lebon}, \text{dubonnet}, 4) = 3$  parce que le suffixe *bon* de *lebon*, de longueur 3, apparaît dans *dubonnet* et se termine à la position 4 de *dubonnet*.
2.  $\text{lmp}(u, v, i)$  est la longueur maximum d'un préfixe de  $u$  qui apparaît dans  $v$  en commençant à la position  $i$  de  $v$ . Par exemple,  $\text{lmp}(\text{pabon}, \text{pabonpapa}, 5) = 2$  car le préfixe *pa* de *pabon* de longueur 2 apparaît à la position 5 de *pabonpapa*.

**Question 13** On considère  $\Sigma = \{a, b, c\}$ . Soient  $u = \text{cabacbab}$  et  $v = \text{cbacbabcbab}$ . Calculer  $\text{lms}(u, v, i)$  et  $\text{lmp}(v, v, i)$  pour  $i$  compris entre 0 et 10. Faites bien attention à lire correctement ce qui est demandé : pour *lms*, on demande  $u$  et  $v$ , mais pour *lmp*, on demande  $v$  et  $v$ . Donner votre réponse en recopiant et complétant le tableau suivant.

$i$	0	1	2	3	4	5	6	7	8	9	10
$v_i$	c	b	a	c	b	a	b	c	b	a	b
$\text{lms}(u, v, i)$											
$\text{lmp}(v, v, i)$											

Soient  $u$  et  $v$  deux mots. On appelle **nouveau carré pour la concaténation de  $u$  et  $v$** , ou simplement **nouveau carré** quand il n'y a pas de risque de confusion, tout carré de  $uv$  qui est la concaténation d'un suffixe non-vide de  $u$  et d'un préfixe non-vide de  $v$  (de manière plus imagée, le carré est à cheval sur les deux mots, ou est créé par la concaténation de  $u$  et  $v$ ).

Un nouveau carré  $ww$  qui apparaît en position  $i$  sur le mot  $uv$  est un **carré centré sur  $u$**  lorsque  $i + |w| < |u|$ . Si  $i + |w| > |u|$  alors on dit qu'il est **centré sur  $v$** . Par exemple, avec les mots  $u$  et  $v$  définis ci-dessus,  $cbabcbacbabca$  est un nouveau carré de période 7 qui apparaît en position 4 sur le mot  $uv$ . Il est centré sur  $v$ . On remarquera que si  $i + |w| = |u|$ , alors  $ww$  est un nouveau carré qui n'est centré ni sur  $u$ , ni sur  $v$ .

Le schéma ci-dessous montre le cas général d'un nouveau carré obtenu lors de la concaténation de  $u$  et de  $v$ , centré sur  $v$ , de période  $p$  et se terminant à la position  $i$  de  $v$ .

$$u_0 \dots u_{|u|-2p+i} \underbrace{u_{|u|-2p+i+1} \dots u_{|u|-1}}_{\text{mot } w} v_0 \dots v_{i-p} \underbrace{v_{i-p+1} \dots v_i}_{\text{mot } w} v_{i+1} \dots v_{|v|-1}$$

**Question 14** Dans l'hypothèse où il existe dans  $uv$  un nouveau carré  $ww$  centré sur  $v$ , de période  $p$  et se terminant à la position  $i$  de  $v$ , calculer en fonction de  $p$  l'indice  $j$  tel que :

$$v_0 \dots v_{i-p} = v_{j+1} \dots v_i$$

Que peut-on dire des mots  $u_{|u|-2p+i+1} \dots u_{|u|-1}$  et  $v_{i-p+1} \dots v_j$  ?

**Question 15** Montrer qu'il existe un carré de période  $p$  dans  $uv$  se terminant à la position  $i$  de  $v$  et centré sur  $v$  si et seulement si

- (a)  $1 \leq p < |v|$ , et
- (b)  $p \leq i < 2p - 1$ , et
- (c)  $2p - lms(u, v, p - 1) - 1 \leq i \leq p + lmp(v, v, p) - 1$ .

**Question 16** Donner sans preuve une condition nécessaire et suffisante similaire à celle de la question précédente pour les carrés centrés sur  $u$ . On commencera par écrire le schéma montrant le cas général d'un nouveau carré centré sur  $u$ .

**Question 17** Utiliser les résultats des questions précédentes pour donner les nouveaux carrés de période 7 centrés sur  $v$  obtenus lors de la concaténation de  $u = cabacbab$  et  $v = cbacbabcbab$ .

Dans cette partie, on suppose donnée une implémentation des fonctions  $lms$  et  $lmp$ . En fait, pour des raisons d'efficacité du calcul, on ne dispose pas directement des fonctions  $lms$  et  $lmp$ , mais plutôt des fonctions suivantes :

```
(* Caml *)  calcul_lms : char vect -> char vect -> int vect
(* Caml *)  calcul_lmp : char vect -> char vect -> int vect
```

La fonction `calcul_lms` appliquée à  $u$  et  $v$  renvoie un tableau d'entiers de taille  $|v|$  tel que l'entier stocké à la position  $i$  soit égal à  $lms(u, v, i)$  pour  $0 \leq i < |v|$ . De façon similaire, la fonction `calcul_lmp` appliquée à  $u$  et  $v$  renvoie un tableau d'entiers de taille  $|v|$  tel que l'entier stocké à la position  $i$  soit égal à  $lmp(u, v, i)$  pour  $0 \leq i < |v|$ . On supposera que l'appel à chacune de ces fonctions sur les mots  $u$  et  $v$  a un coût  $\mathcal{O}(|u| + |v|)$ .

**Question 18** Écrire une fonction `nouveau_carre_v` qui prend en entrée deux mots  $u$  et  $v$  et renvoie `true` si le mot obtenu lors de la concaténation de  $u$  et de  $v$  fait apparaître un nouveau carré centré sur  $v$ , et `false` sinon. On garantira une complexité en  $\mathcal{O}(|u| + |v|)$ .

---

```
(* Caml *) nouveau_carre_v : char vect -> char vect -> bool
```

---

On admettra l'existence d'une fonction `nouveau_carre_u` (similaire à celle de la question précédente) de complexité  $\mathcal{O}(|u| + |v|)$  qui renvoie `true` si un nouveau carré centré sur  $u$  apparaît lors de la concaténation des mots  $u$  et  $v$  (et renvoie `false` sinon).

**Question 19** Écrire une fonction `nouveau_carre` qui prend en entrée deux mots  $u$  et  $v$  et renvoie `true` si un nouveau carré apparaît lors de la concaténation des mots  $u$  et  $v$ , et renvoie `false` sinon. On veillera à bien prendre en compte d'éventuels nouveaux carrés qui ne sont ni centrés sur  $u$  ni centrés sur  $v$ , et on garantira une complexité en  $\mathcal{O}(|u| + |v|)$ .

---

```
(* Caml *) nouveau_carre : char vect -> char vect -> bool
```

---

**Question 20** En déduire une fonction `carre` qui renvoie `true` si le mot  $m$  donné en entrée admet un carré, et `false` sinon. On garantira que le calcul se fait en  $\mathcal{O}(|m| \times \log(|m|))$ .

---

```
(* Caml *) carre : char vect -> bool
```

---

## Partie IV. Implémentation efficace des fonctions *lms* et *lmp*

On souhaite maintenant implémenter efficacement le calcul de  $lms(u, v, i)$ , et  $lmp(u, v, i)$ . Comme annoncé dans la partie précédente, on implémente la fonction `calcul_lms` (respectivement `calcul_lmp`) qui, appliquée à  $u$  et  $v$ , renvoie un tableau d'entiers de taille  $|v|$  tel que l'entier stocké dans la case  $i$  est égal à  $lms(u, v, i)$  (respectivement  $lmp(u, v, i)$ ) pour  $0 \leq i < |v|$ . On souhaite de plus garantir une complexité  $\mathcal{O}(|u| + |v|)$  lors de l'appel à `calcul_lms` u v (respectivement `calcul_lmp` u v).

Pour programmer `calcul_lmp`, on met à jour itérativement la table des préfixes d'un mot comme expliqué ci-après. Considérons un mot  $v$  et pour tout entier  $0 \leq i < |v|$ , posons

$$pref_i = lmp(v, v, i)$$

Une méthode naïve pour calculer  $pref_i$  consisterait à évaluer chaque valeur indépendamment des valeurs précédentes par comparaisons directes. Cependant, l'utilisation des valeurs déjà calculées permet d'obtenir un algorithme plus efficace.

Illustrons le procédé sur un exemple. Considérons le mot  $v = aabaabaaab$ , et supposons  $pref_i$  déjà connu pour  $i$  allant de 0 à 3 comme indiqué dans la table des préfixes donnée ci-dessous.

$i$	0	1	2	3	4	5	6	7	8	9
$v_i$	a	a	b	a	a	b	a	a	a	b
$pref_i$	10	1	0	5	?	?	?	?	?	?

TABLE 1 – Table des préfixes pour le mot  $v = aabaabaaab$ .

Posons  $u = v_3v_4v_5v_6v_7$ . Le calcul de  $pref_3$  nous dit que  $u$  est un préfixe de  $v$  (et c'est le plus long débutant à la position 3). Autrement dit,  $u = v_0v_1v_2v_3v_4$  et  $v_8 \neq v_5$ .

1. On souhaite calculer  $pref_4$ . De l'égalité précédente, on déduit  $v_4v_5v_6v_7 = v_1v_2v_3v_4$ . Dans la mesure où  $pref_1 \leq 4$ , la situation à la position 4 est semblable à celle de la position 1. On a donc  $pref_4 = pref_1 = 1$ . De même, on a  $pref_5 = pref_2 = 0$ .
2. On regarde maintenant comment calculer  $pref_6$ . Du calcul de  $pref_3$ , on a  $u = v_3v_4v_5v_6v_7$  préfixe de  $v$ , et donc les égalités suivantes :  $v_3v_4 = v_0v_1$ ,  $v_5 = v_2$ , et  $v_6v_7 = v_3v_4$ . De plus, on a vu que  $v_8 \neq v_5$ . On en déduit que  $pref_6 = 2$ .
3. Enfin, regardons comment calculer  $pref_7$ . On a  $v_7$  suffixe de  $u$  et préfixe de  $v_4 \dots v_9$  et donc de  $v$  (car  $pref_4 = 1$ ). On en déduit que  $pref_7 \geq 1$ . Pour savoir la longueur maximale du préfixe de  $v$  qui démarre à la position 7, il nous faut donc reprendre les comparaisons :  $v_8$  contre  $v_1$ ,  $v_9$  contre  $v_2$ , ... On obtient ici  $pref_7 = 3$ .

Pour faire ce raisonnement de façon systématique, on introduit, l'indice  $i \geq 2$  étant fixé, deux valeurs  $g$  et  $f$  qui constituent les éléments clés de la méthode. Elles satisfont les relations :

$$g = \max\{j + pref_j \mid 0 < j < i\} \quad f \in \{j \mid 0 < j < i \text{ et } j + pref_j = g\}$$

**Question 21** Compléter la table 1 en indiquant la valeur de  $g$  et les valeurs possibles de  $f$  pour chaque indice  $i$  compris entre 2 et  $|v| - 1 = 9$ .

**Question 22** Soient  $1 < i < |v|$ ,  $g = \max\{j + pref_j \mid 0 < j < i\}$ , et  $f \in \{j \mid 0 < j < i \text{ et } j + pref_j = g\}$ .

- a. Que peut-on dire du mot  $v_f \dots v_{g-1}$  ?
- b. Lorsque que  $g < i$ , exprimer  $g$  en fonction de  $i$ .

**Question 23** Soient  $1 < i < |v|$ ,  $g = \max\{j + pref_j \mid 0 < j < i\}$ , et  $f \in \{j \mid 0 < j < i \text{ et } j + pref_j = g\}$ . On suppose de plus que  $i < g$ . Montrer le résultat suivant :

$$pref_i = \begin{cases} pref_{i-f} & \text{si } pref_{i-f} < g - i \\ g - i & \text{si } pref_{i-f} > g - i \\ g - i + \ell & \text{sinon} \end{cases}$$

où  $\ell$  est la longueur du plus long préfixe commun à  $v_{g-i} \dots v_{m-1}$  et  $v_g \dots v_{m-1}$ . Autrement dit, on a  $\ell = lmp(v_{g-i} \dots v_{m-1}, v_g \dots v_{m-1}, 0)$ . On posera  $k = pref_{i-f}$  et  $k' = g - i$ , et on étudiera séparément les cas  $k < k'$ ,  $k > k'$ , et  $k = k'$ . Ces trois cas correspondent aux trois situations illustrées au travers des exemples donnés en début de cette partie.

**Question 24** On considère le code donné en Figure 1.

```

1. let calcul_pref v =
2.   let pref = make_vect (vect_length v) 0 in
3.   pref.(0) <- vect_length v;
4.   let g = ref 0 in
5.   let f = ref 0 in
6.   for i = 1 to vect_length v - 1 do
7.     if i < !g && pref.(i - !f) < !g - i then
8.       pref.(i) <- pref.(i - !f)
9.     else
10.      if i < !g && pref.(i - !f) > !g - i then
11.        pref.(i) <- !g - i
12.      else
13.        begin
14.          f:=i;
15.          g:= max !g i;
16.          while !g < vect_length v && v.(!g) == v.(!g - !f) do
17.            g := !g + 1;
18.            done;
19.            pref.(i) <- !g - !f;
20.          end;
21.        done;
22.      pref;;

```

FIGURE 1 – Code de la question 24.

```
(* Caml *)  calcul_pref :char vect -> int vect
```

- a. Justifier que ce code réalise bien le calcul demandé, c'est-à-dire que la fonction `calcul_pref`, appliquée à un mot  $v$ , renvoie le tableau  $\text{pref}$  défini ci-dessus.
- b. Justifier la terminaison de l'algorithme. Donner et justifier sa complexité.

Dans les trois questions suivantes, on demande des programmes courts et simples. En particulier, dans les deux questions suivantes, l'idée est d'utiliser la fonction `calcul_pref` sur des mots facilement obtenus à partir de  $u$ ,  $v$  et d'un caractère spécial '#' n'appartenant pas à l'alphabet (on pourra utiliser des concaténations et des retournements).

**Question 25** Donner une implémentation de complexité  $\mathcal{O}(|u| + |v|)$  pour la fonction `calcul_lmp`.

**Question 26** Afin d'implémenter la fonction `calcul_lms` demandée à la question suivante, on se propose tout d'abord de programmer une fonction `calcul_suff`, telle que `(calcul_suff v)` renvoie la table des suffixes du mot  $v$ , c'est-à-dire un tableau d'entiers de taille  $|v|$  tel que l'entier stocké dans la case  $i$  est égal à  $\text{lms}(v, v, i)$ . Écrire le code de la fonction `calcul_suff`  $v$  en garantissant une complexité  $\mathcal{O}(|v|)$ . Attention, ici, on ne suppose pas disposer d'une implémentation

de la fonction `calcul_lms` car cette fonction sera implémentée à la question suivante à l'aide de `calcul_suff`.

---

```
(* Caml *)  calcul_suff : char vect -> int vect
```

---

**Question 27** En utilisant la fonction `calcul_suff` de la question précédente, écrire le code de la fonction `calcul_lms`. On garantira une complexité en  $O(|u| + |v|)$ .

---

```
(* Caml *)  calcul_lms : char vect -> char vect -> int vect
```

---

Notes : Il existe de nombreuses constructions de mots infinis sans carrés. Le plus ancien est le mot de Thue. Le mot présenté dans ce sujet est original, mais très proche du mot de Dean [R.A. Dean, A sequence without repeats on  $x, x^{-1}, y, y^{-1}$ , *American Mathematical Monthly*, volume 72, pages 383–385, 1965]. L'algorithme présenté de ce sujet est dû à Main et Lorentz [M.G. Main and R.J. Lorentz, An  $O(n \log n)$  algorithm for finding all repetitions in a string, *Journal of Algorithm*, volume 5, pages 422–432, 1984].

\* \*  
\*

## Première partie : langages et automates

On s'intéresse aux langages sur l'alphabet  $\Sigma = \{a\}$  ; un tel langage est dit *unaire*. Un automate reconnaissant un langage unaire sera dit *unaire*. Lorsqu'on dessinera un automate unaire, il ne sera pas utile de faire figurer les étiquettes des transitions, toutes ces étiquettes étant l'étiquette  $a$ . C'est ce qui est fait dans cet énoncé.

Dans un automate unaire, on appelle *chemin* une suite  $q_1, \dots, q_p$  d'états telle que, pour  $i$  compris entre 1 et  $p$ , il existe une transition de  $q_{i-1}$  vers  $q_i$  ; on dit qu'il s'agit d'un chemin de  $q_1$  à  $q_p$ . On appelle *circuit* un chemin  $q_1, \dots, q_p$  tel qu'il existe une transition de  $q_p$  vers  $q_1$ .

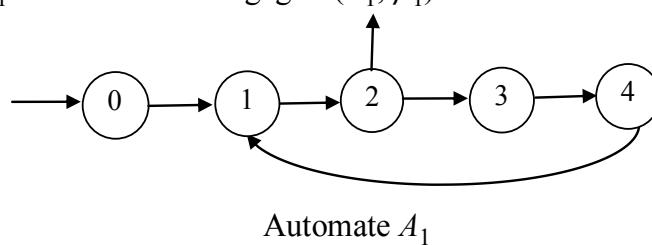
Dans cet exercice, tous les automates considérés seront finis et auront un et un seul état initial. On dit qu'un automate est *émondé* si, pour tout état  $q$ , il existe d'une part un chemin de l'état initial à  $q$  et d'autre part un chemin de  $q$  à un état final.

On rappelle qu'un langage non vide est rationnel si et seulement s'il est reconnu par un automate ou encore si et seulement s'il est reconnu par un automate déterministe émondé.

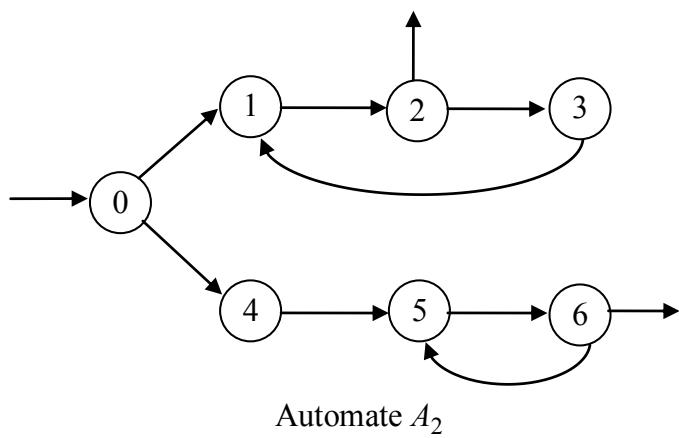
Soient  $\alpha$  et  $\beta$  deux entiers positifs ou nuls. On note  $L(\alpha, \beta)$  le langage unaire défini par :

$$L(\alpha, \beta) = \{a^{\alpha k} + \beta \mid k \text{ entier positif ou nul}\}.$$

- 1 – Donner sans justification une condition nécessaire et suffisante pour que  $L(\alpha, \beta)$  soit fini. Dans le cas où cette condition est satisfaite, donner sans justification le cardinal de  $L(\alpha, \beta)$ .
- 2 – On considère l'automate  $A_1$  ci-dessous. Indiquer sans justification deux entiers  $\alpha_1, \beta_1$  tels que  $A_1$  reconnaisse le langage  $L(\alpha_1, \beta_1)$ .



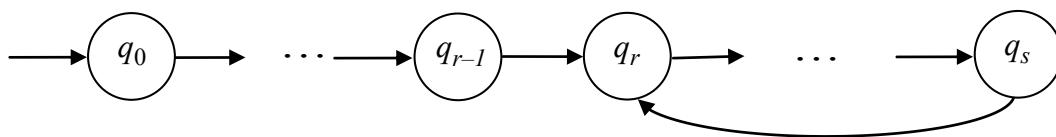
- 3 – On considère l'automate  $A_2$  ci-dessous :



On note  $L_2$  le langage reconnu par  $A_2$ . Indiquer sans justification quatre entiers  $\alpha_2, \beta_2, \alpha_3, \beta_3$  tels que  $A_2$  reconnaisse le langage  $L_2 = L(\alpha_2, \beta_2) \cup L(\alpha_3, \beta_3)$ .

- 4 – Construire un automate *déterministe émondé*  $A_3$  en appliquant la procédure de déterminisation à l'automate  $A_2$ .
- 5 – En s'appuyant sur l'automate  $A_3$ , indiquer sans justification cinq entiers  $\alpha_4, \beta_4, \beta_5, \beta_6, \beta_7$ , tels que  $A_3$  reconnaisse le langage  $L_3 = L(\alpha_4, \beta_4) \cup L(\alpha_4, \beta_5) \cup L(\alpha_4, \beta_6) \cup L(\alpha_4, \beta_7)$  (remarque : le langage  $L_3$  est égal par ailleurs au langage  $L_2$ ).

On dit ci-dessous qu'un automate est *de la forme F* si, en omettant les états finals, il peut se tracer selon le schéma ci-dessous :



Le chemin  $q_0, \dots, q_{r-1}$  peut être vide, auquel cas on a  $r = 0$ . Le circuit  $q_r, \dots, q_s$  ne doit pas être vide mais on peut avoir  $r = s$  avec une transition de l'état  $q_r$  vers lui-même (un tel circuit s'appelle aussi une *boucle*). On constate que les automates  $A_1$  et  $A_3$  sont de la forme  $F$ , mais non  $A_2$ .

- 6 – Dessiner sans justification un automate de la forme  $F$  qui reconnaît le langage  $L(1, 2)$ . On fera figurer le ou les état(s) final(s).

*ATTENTION* : on ne demande aucune justification mais uniquement de tracer un automate de la forme  $F$  en choisissant correctement les longueurs du chemin et du circuit et en ajoutant le ou les état(s) final(s).

- 7 – Dessiner un automate de la forme  $F$  qui reconnaît le langage  $L(2, 3) \cup L(5, 2)$ . On fera figurer le ou les état(s) final(s). Comme à la question précédente, on ne demande aucune justification.

- 8 – En s'inspirant de la réponse à la question précédente, décrire sans justification un automate de la forme  $F$  qui reconnaît le langage  $L(2, 3) \cap L(5, 2)$ . Indiquer deux entiers  $\alpha$  et  $\beta$  tels qu'on ait la relation :  $L(2, 3) \cap L(5, 2) = L(\alpha, \beta)$ .

- 9 – Montrer qu'un automate déterministe émondé qui reconnaît un langage unaire rationnel infini est de la forme  $F$ . Donner une condition nécessaire et suffisante portant sur les états finals pour qu'un automate de la forme  $F$  reconnaîsse un langage infini.

- 10 – Soit  $L$  un langage rationnel unaire infini. En s'appuyant sur la question précédente, montrer qu'il existe deux entiers  $\alpha \geq 1$  et  $\beta \geq 0$  tels que  $L$  contient  $L(\alpha, \beta)$ .

- 11 – On considère une suite  $(u_n)_{n \geq 0}$  de nombres entiers positifs ou nuls. On suppose que la suite  $(u_{n+1} - u_n)_{n \geq 0}$  est positive et strictement croissante. Soit  $L$  le langage défini par :  $L = \{a^{u_n} \mid n \geq 0\}$ . En utilisant la question précédente, montrer que  $L$  n'est pas rationnel.

- 12 – Montrer que le langage  $L$  défini par  $L = \{a^{n^2} \mid n \geq 0\}$  n'est pas rationnel.

## Seconde partie : algorithmique et programmation

### Préliminaire concernant la programmation

Il faudra coder des fonctions à l'aide du langage de programmation Caml, tout autre langage étant exclu. Lorsque le candidat écrira une fonction, il pourra faire appel à d'autres fonctions définies dans les questions précédentes ; il pourra aussi définir des fonctions auxiliaires. Quand l'énoncé demande de coder une fonction, il n'est pas nécessaire de justifier que celle-ci est correcte, sauf si l'énoncé le demande explicitement. Enfin, si les paramètres d'une fonction à coder sont supposés vérifier certaines hypothèses, il ne sera pas utile dans l'écriture de cette fonction de tester si les hypothèses sont bien vérifiées.

Dans les énoncés de l'exercice, un même identificateur écrit dans deux polices de caractères différentes désignera la même entité, mais du point de vue mathématique pour la police en italique (par exemple  $n$ ) et du point de vue informatique pour celle en romain (par exemple  $n$ ).

On ne se préoccupera pas d'un éventuel dépassement du plus grand entier représentable.

On pourra utiliser les fonctions suivantes.

- La fonction `empiler` ajoute une valeur en début d'une liste dont la référence est passée en paramètre ; par exemple si on a :

```
let liste = ref [1; 2; 3];;
après l'instruction : empiler liste 5;;
!liste vaut [5; 1; 2; 3].
```

- La fonction `depiler` retire la valeur du début d'une liste dont la référence est passée en paramètre et renvoie la valeur retirée ; par exemple si on a :

```
let liste = ref [1; 2; 3];;
après l'instruction : let val = depiler liste;;
!liste vaut [2; 3] et val vaut 1.
```

Cette fonction ne doit être utilisée que si la liste dont la référence est passée en paramètre n'est pas vide.

- La fonction `longueur` renvoie la longueur d'une liste passée en paramètre.

- La fonction `inverse` reçoit en paramètre une liste et renvoie une nouvelle liste qui est l'inverse de la première. Par exemple, si on a :

```
let liste = [1; 2; 3];;
l'instruction inverse liste;; renvoie la liste [3; 2; 1].
```

On considère un ensemble  $U$  muni d'une loi de composition interne associative appelée *multiplication* et possédant un élément neutre pour cette loi noté  $e$ . Cette multiplication est notée avec le signe  $\times$ .

Par exemple,  $U$  peut être l'ensemble des entiers ou des réels munis de la multiplication usuelle, l'élément neutre étant 1. L'ensemble  $U$  peut aussi être l'ensemble des matrices carrées booléennes (respectivement d'entiers, de réels) d'une même dimension  $d$  avec le produit usuel comme multiplication, l'élément neutre étant la matrice identité booléenne (respectivement entière, réelle) de dimension  $d$ .

Soit  $a$  un élément de  $U$  et soit  $n$  un entier positif ou nul. On définit  $a^n$  de la façon suivante :

- $a^0 = e$ ,
- si  $n \geq 1$ ,  $a^n = a^{n-1} \times a$ .

La multiplication étant associative, si  $i$  et  $j$  sont deux entiers positifs ou nuls de somme égale à  $n$ , on a :  $a^n = a^i \times a^j$ .

Un élément  $a$  de  $U$  et un entier  $n$  supérieur ou égal à 1 étant donnés, on cherche à calculer  $a^n$  en s'intéressant au nombre de multiplications effectuées.

Dans toute la suite,  $a$  et  $n$  désignent respectivement un élément quelconque de  $U$  et un entier strictement positif.

*Exemple 1 :*  $n = 14$ . On peut calculer  $a^{14}$  en multipliant 13 fois l'élément  $a$  par lui-même. On effectue ainsi 13 multiplications.

*Exemple 2 :*  $n = 14$ . On peut calculer  $a^{14}$  en calculant  $a^2$  par  $a^2 = a \times a$ , puis  $a^3$  par  $a^3 = a^2 \times a$  puis  $a^6$  par  $a^6 = a^3 \times a^3$ , puis  $a^7$  par  $a^7 = a^6 \times a$ , puis enfin  $a^{14} = a^7 \times a^7$ . On a ainsi obtenu le résultat en effectuant 5 multiplications.

*Exemple 3 :*  $n = 14$ . On peut aussi calculer  $a^{14}$  en calculant  $a^2$  par  $a^2 = a \times a$ , puis  $a^4$  par  $a^4 = a^2 \times a^2$ , puis  $a^6$  par  $a^6 = a^2 \times a^4$  puis  $a^8$  par  $a^8 = a^4 \times a^4$ , puis  $a^{14}$  par  $a^{14} = a^6 \times a^8$ . On a ainsi obtenu le résultat en effectuant encore 5 multiplications.

L'objectif est de déterminer des algorithmes qui effectuent peu de multiplications. Soit  $x$  un nombre réel positif ; on note  $\lfloor x \rfloor$  la partie entière par défaut de  $x$  et  $\lceil x \rceil$  sa partie entière par excès.

On appelle *suite pour l'obtention de la puissance n* toute suite non vide croissante d'entiers distincts  $(n_0, n_1, \dots, n_r)$  telle que :

- $n_0 = 1$ ,
- $n_r = n$ ,
- pour tout indice  $k$  vérifiant  $1 \leq k \leq r$ , il existe deux entiers  $i$  et  $j$  distincts ou non vérifiant  $0 \leq i \leq k - 1$ ,  $0 \leq j \leq k - 1$  et  $n_k = n_i + n_j$  (la paire  $\{i, j\}$  n'est pas nécessairement unique).

À une suite pour l'obtention de la puissance  $n$  correspond une suite de multiplications conduisant au calcul de  $a^n$ . Par exemple, la suite  $(1, 2, 4, 6, 7, 12, 19)$  correspond au calcul de  $a^{19}$  en faisant les 6 multiplications suivantes :  $a^2 = a \times a$ ,  $a^4 = a^2 \times a^2$ ,  $a^6 = a^2 \times a^4$ ,  $a^7 = a \times a^6$ ,  $a^{12} = a^6 \times a^6$ ,  $a^{19} = a^7 \times a^{12}$ .

Réciproquement, considérons un calcul de  $a^n$  dans lequel on fait en sorte d'ordonner les multiplications pour que les puissances calculées soient d'exposants croissants ; on peut associer à ce calcul une suite pour l'obtention de la puissance  $n$ .

À l'exemple 1 est associée la suite  $(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14)$ , de longueur 14.

À l'exemple 2 est associée la suite  $(1, 2, 3, 6, 7, 14)$ , de longueur 6.

À l'exemple 3 est associée la suite  $(1, 2, 4, 6, 8, 14)$ , de longueur 6.

Le nombre de multiplications correspondant à une suite pour l'obtention de la puissance  $n$  est égal à la longueur de la suite diminuée de 1.

- 13 – Montrer que tout calcul de  $a^n$  qui n'utilise que des multiplications nécessite un nombre de multiplications au moins égal à  $\lceil \log_2 n \rceil$ . Donner une famille infinie de valeurs de  $n$  qui peuvent être calculées en effectuant exactement ce nombre de multiplications ; justifier la réponse.

On considère un algorithme appelé *par\_division* ayant pour objectif le calcul de  $a^n$ . Cet algorithme s'appuie sur le principe récursif suivant :

si  $n$  vaut 1, alors  $a^n$  vaut  $a$ ,  
sinon

- on calcule la partie entière par défaut, notée  $q$ , de  $n/2$ ,
- on calcule par l'algorithme *par\_division* la valeur de  $b = a^q$ ,
- si  $n$  est pair, alors  $a^n = b \times b$ ,
- sinon  $a^n = (b \times b) \times a$ .

Ainsi, pour obtenir  $a^{14}$ , l'algorithme *par\_division* fait appel au calcul de  $a^7$  qui fait appel au calcul de  $a^3$  (pour obtenir  $a^6$  en multipliant  $a^3$  par  $a^3$  puis  $a^7$  en multipliant  $a^6$  par  $a$ ) qui fait appel au calcul de  $a^1$  (pour obtenir  $a^2$  puis  $a^3$ ). Les différentes puissances calculées sont les puissances 1, 2, 3, 6, 7 et 14. On constate ainsi que la suite pour l'obtention de la puissance 14 correspondant à l'algorithme *par\_division* est la suite (1, 2, 3, 6, 7, 14), de longueur 6. De même, la suite pour l'obtention de la puissance 19 correspondant à l'algorithme *par\_division* est la suite (1, 2, 4, 8, 9, 18, 19) de longueur 7.

14 – Calculer (sans justification) la suite correspondant à l'algorithme *par\_division* successivement :

- pour l'obtention de la puissance 15 ;
- pour l'obtention de la puissance 16 ;
- pour l'obtention de la puissance 27 ;
- pour l'obtention de la puissance 125.

Dans chaque cas, indiquer la longueur de la suite obtenue.

15 – Écrire en Caml la fonction nommée *par\_division* qui calcule la suite pour l'obtention de la puissance  $n$  correspondant à l'algorithme *par\_division*. Si  $n$  est une valeur entière strictement positive, *par\_division n* renvoie une liste contenant la suite pour l'obtention de la puissance  $n$ .

16 – Montrer que l'algorithme *par\_division* pour l'obtention de la puissance  $n$  effectue au plus  $2 \times \lfloor \log_2 n \rfloor$  multiplications. Montrer que ce nombre est atteint pour un nombre infini de valeurs de  $n$ .

On considère maintenant un algorithme appelé *par\_decomposition\_binaire* dont l'objectif est aussi le calcul de  $a^n$ . Cet algorithme utilise la décomposition d'un entier suivant les puissances de 2. L'algorithme est expliqué ci-dessous à l'aide d'exemples.

- Soit  $n = 14$ . On décompose 14 selon les puissances de 2 :  $14 = 2 + 4 + 8$ . On a donc :  $a^{14} = (a^2 \times a^4) \times a^8$ , ce qui conduit à calculer les puissances de  $a$  d'exposants 2, 4, 8 mais aussi 6 et 14 ; la suite pour l'obtention de la puissance 14 correspondant à cet algorithme est la suite (1, 2, 4, 6, 8, 14).
- Soit  $n = 18$ . On a :  $18 = 2 + 16$ , ce qui implique :  $a^{18} = a^2 \times a^{16}$ . L'algorithme calcule les puissances d'exposants 2, 4, 8, 16 puis 18 ; la suite pour l'obtention de la puissance 18 correspondant à cet algorithme est : (1, 2, 4, 8, 16, 18).
- Soit  $n = 101$ . On a :  $101 = 1 + 4 + 32 + 64$ . L'algorithme calcule  $a^{101}$  en utilisant les multiplications impliquées par la formule :  $a^{101} = ((a \times a^4) \times a^{32}) \times a^{64}$  ; on calcule les puissances 2, 4, 5 (pour  $a \times a^4 = a^5$ ), 8, 16, 32, 37 (pour  $a^5 \times a^{32} = a^{37}$ ), 64 et 101 (pour  $a^{37} \times a^{64} = a^{101}$ ) ; la suite pour l'obtention de la puissance 101 correspondant à cet algorithme est : (1, 2, 4, 5, 8, 16, 32, 37, 64, 101).

De manière générale, l'algorithme procède en écrivant la décomposition unique de  $n$  comme une somme de puissances croissantes du nombre 2, et calcule la valeur cible de  $a^n$  en effectuant les produits correspondant aux sommes partielles de cette somme.

17 – Calculer (sans justification) la suite correspondant à l'algorithme *par\_decomposition\_binaire* successivement :

- pour l'obtention de la puissance 15 ;
- pour l'obtention de la puissance 16 ;
- pour l'obtention de la puissance 27 ;
- pour l'obtention de la puissance 125.

Dans chaque cas, indiquer la longueur de la suite obtenue.

On considère la décomposition de  $n$  suivant les puissances croissantes du nombre 2 :

$$n = c_0 + c_1 \times 2 + \dots + c_i \times 2^i + \dots + c_k \times 2^k,$$

où, pour  $i$  vérifiant  $0 \leq i < k$ , le coefficient  $c_i$  vaut 0 ou 1 et  $c_k$  vaut 1.

On appelle *écriture binaire inverse* de  $n$  la suite  $(c_0, c_1, \dots, c_i, \dots, c_k)$ .

Par exemple, l'écriture binaire inverse de l'entier 14 est (0, 1, 1, 1), celle de l'entier 18 est (0, 1, 0, 0, 1) et celle de l'entier 101 est (1, 0, 1, 0, 0, 1, 1).

18 – Écrire en Caml une fonction nommée *binaire\_inverse* qui calcule l'écriture binaire inverse d'un nombre donné. Si  $n$  est une valeur entière strictement positive, alors *binaire\_inverse n* renvoie une liste contenant l'écriture binaire inverse de  $n$ .

19 – Écrire en Caml la fonction *par\_decomposition\_binaire* qui calcule la suite pour l'obtention de la puissance  $n$  correspondant à l'algorithme *par\_decomposition\_binaire*. Si  $n$  est une valeur entière strictement positive, *par\_decomposition\_binaire n* renvoie une liste contenant la suite cherchée.

20 – On suppose que l'on a  $n = 3^k$ , où  $k$  est un entier positif ou nul. En utilisant la formule :  $3^k = 3^{k-1} + 2 \times 3^{k-1}$ , montrer qu'il existe une suite pour l'obtention de la puissance  $n$  de longueur  $2k + 1$ . Indiquer une telle suite correspondant à  $n = 27$  ; comparer la longueur de cette suite à la longueur de la suite correspondant à l'algorithme *par\_division*.

21 – Soit  $k$  un entier positif ou nul. Écrire en Caml une fonction *suite\_3* calculant une suite de longueur  $2k + 1$  pour l'obtention de la puissance  $3^k$ . On s'appuiera pour cela sur la question précédente. Si  $k$  est une valeur entière positive ou nulle, *suite\_3 k* renvoie une liste contenant la suite cherchée.

22 – On suppose que l'on a  $n = 5^k$ , où  $k$  est un entier positif ou nul quelconque. Montrer qu'il existe une suite pour l'obtention de la puissance  $n$  de longueur  $3k + 1$ . Indiquer la suite correspondant à  $n = 125$  ; comparer la longueur de cette suite à la longueur de la suite correspondant à l'algorithme *par\_division*.

23 – Donner une suite de longueur 6 pour l'obtention de la puissance 15. Qu'en déduire quant aux algorithmes *par\_division* et *par\_decomposition\_binaire* étudiés précédemment ?

□ 24 – On considère un tableau (ou vecteur)  $T$ , indicé à partir de 0, contenant une suite pour l'obtention d'une certaine puissance positive  $n$ . Soit  $k$  un entier compris entre 1 et la longueur de  $T$  diminuée de 1. Soit  $val$  la valeur contenue dans  $T$  à l'indice  $k$ . On sait que  $val$  est la somme de deux valeurs du tableau  $T$  situées à des indices (éventuellement confondus, éventuellement non uniques) strictement inférieurs à  $k$ . Il s'agit de programmer une fonction nommée `chercher_indice` qui détermine ces deux indices. Par exemple, si tableau  $T$  contient les valeurs 1, 2, 3, 4, 7, 14, 17, 31, la longueur du tableau vaut 8, et, si  $k$  vaut 6, alors  $val$  vaut 17 et la fonction `chercher_indice` doit renvoyer les indices 2 et 5 correspondant aux valeurs 3 et 14 du tableau ; si, avec ce même tableau,  $k$  vaut 1, la fonction doit renvoyer 0 et 0.

Écrire en Caml la fonction `chercher_indice` telle que, si :

- $T$  code un vecteur contenant une suite pour l'obtention d'une certaine puissance positive  $n$  (la valeur de  $n$  est inutile pour l'écriture de la fonction),
  - $k$  code un entier compris entre 1 et la longueur de  $T$  diminuée de 1,
- alors `chercher_indice T k` renvoie une liste de deux entiers contenant les deux indices cherchés, par ordre croissant. Indiquer (sans justification) la complexité  $C(k)$  de cette fonction.

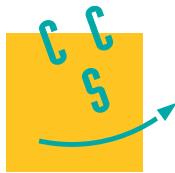
□ 25 – Dans cette question,  $U$  est l'ensemble des réels. Soit  $x$  un nombre réel quelconque et soit un tableau (ou vecteur)  $T$  contenant une suite pour l'obtention d'une certaine puissance positive  $n$ . Écrire en Caml une fonction `puissance` qui calcule la valeur de  $x^n$  en utilisant le tableau  $T$ . Si  $x$  est une valeur de type `float` et  $T$  code un vecteur contenant une suite pour l'obtention d'une certaine puissance positive  $n$ , alors `puissance x T` renvoie la valeur de  $x^n$  en effectuant des multiplications suivant la suite représentée par  $T$ .

*Indications :*

- on utilisera la fonction `chercher_indice` de la question précédente ; en appelant  $h$  la longueur de  $T$ , la complexité de la fonction `puissance` devra nécessairement être en  $O(h \times C(h))$  (il n'est pas demandé de justifier cette complexité) ;
- si  $T$  est un vecteur, `vect_length T` donne la longueur de  $T$  ;
- l'opérateur `*.` permet de faire le produit de deux valeurs de type `float`.

□ 26 – Décrire le principe d'une fonction nommée `suite_optimale` permettant d'exhiber une suite de longueur minimale pour l'obtention de la puissance  $n$ , en effectuant une énumération exhaustive des suites possibles. Cette fonction devra utiliser une fonction récursive nommée `suite_optimale_rec` dont on donnera aussi le principe.

□ 27 – Écrire en Caml les fonctions `suite_optimale_rec` et `suite_optimale` correspondant aux fonctions de la question précédente. Si  $n$  est une valeur de type `int`, alors `suite_optimale_rec n` renvoie une liste contenant une suite de longueur minimale pour l'obtention de la puissance  $n$ .



CONCOURS CENTRALE-SUPÉLEC

# Option informatique

MP

2017

4 heures

Calculatrices autorisées

## Mots synchronisants

### Notations

- Pour tout ensemble fini  $E$ , on note  $|E|$  son cardinal.
- On appelle *machine* tout triplet  $(Q, \Sigma, \delta)$  où  $Q$  est un ensemble fini non vide dont les éléments sont appelés *états*,  $\Sigma$  un ensemble fini non vide appelé *alphabet* dont les éléments sont appelés *lettres* et  $\delta$  une application de  $Q \times \Sigma$  dans  $Q$  appelée *fonction de transition*. Une machine correspond donc à un automate déterministe complet sans notion d'état initial ou d'états finaux.
- Pour un état  $q$  et une lettre  $x$ , on note  $q.x = \delta(q, x)$ .
- L'ensemble des *mots* (c'est-à-dire des concaténations de lettres) sur l'alphabet  $\Sigma$  est noté  $\Sigma^*$ .
- Le mot vide est noté  $\varepsilon$ .
- On note  $ux$  le mot obtenu par la concaténation du mot  $u$  et de la lettre  $x$ .
- On note  $\delta^*$  l'extension à  $Q \times \Sigma^*$  de la fonction de transition  $\delta$  définie par

$$\begin{cases} \forall q \in Q, \quad \delta^*(q, \varepsilon) = q \\ \forall (q, x, u) \in Q \times \Sigma \times \Sigma^*, \quad \delta^*(q, xu) = \delta^*(\delta(q, x), u) \end{cases}$$

- Pour un état  $q$  de  $Q$  et un mot  $m$  de  $\Sigma^*$ , on note encore  $q.m$  pour désigner  $\delta^*(q, m)$ .

Pour deux états  $q$  et  $q'$ ,  $q'$  est dit *accessible* depuis  $q$  s'il existe un mot  $u$  tel que  $q' = q.u$ .

On dit qu'un mot  $m$  de  $\Sigma^*$  est *synchronisant* pour une machine  $(Q, \Sigma, \delta)$  s'il existe un état  $q_0$  de  $Q$  tel que pour tout état  $q$  de  $Q$ ,  $q.m = q_0$ .

L'existence de tels mots dans certaines machines est utile car elle permet de ramener une machine dans un état particulier connu en lisant un mot donné (donc en pratique de la « réinitialiser » par une succession précise d'ordres passés à la machine réelle).

La partie I de ce problème étudie quelques considérations générales sur les mots synchronisants, la partie II est consacrée à des problèmes algorithmiques classiques, la partie III relie le problème de la satisfiabilité d'une formule logique à celui de la recherche d'un mot synchronisant de longueur donnée dans une certaine machine et enfin la partie IV s'intéresse à l'étude de l'existence d'un mot synchronisant pour une machine donnée. Les parties I, II et III peuvent être traitées indépendamment. La partie IV, plus technique, utilise la partie II.

Dans les exemples concrets de machines donnés plus loin, l'ensemble d'états peut être quelconque, de même que l'alphabet ( $\Sigma = \{0, 1\}$ ,  $\{a, b, c\}$  ...). Par contre, pour la modélisation en Caml, l'alphabet  $\Sigma$  sera toujours considéré comme étant un intervalle d'entiers  $\llbracket 0, p - 1 \rrbracket$  où  $p = |\Sigma|$ . Une lettre correspondra donc à un entier entre 0 et  $p - 1$ . Un mot de  $\Sigma^*$  sera représenté par une liste de lettres (donc d'entiers).

```
type lettre == int;;
type mot == lettre list;;
```

De même, en Caml, l'ensemble d'états  $Q$  d'une machine sera toujours considéré comme étant l'intervalle d'entiers  $\llbracket 0, n - 1 \rrbracket$  où  $n = |Q|$ .

```
type etat == int;;
```

Ainsi, la fonction de transition  $\delta$  d'une machine sera modélisée par une fonction Caml de signature `etat -> lettre -> etat`. On introduit alors le type `machine`

```
type machine = { n_etats : int ; n_lettres : int ; delta : etat -> lettre -> etat};;
```

`n_etats` correspond au cardinal de  $Q$ , `n_lettres` à celui de  $\Sigma$  et `delta` à la fonction de transition. Pour une machine nommée `M`, les syntaxes `M.n_etats`, `M.n_lettres` ou `M.delta` permettent d'accéder à ses différents paramètres. Dans le problème, on suppose que `M.delta` s'exécute toujours en temps constant.

Par exemple, on peut créer une machine `M0` à trois états sur un alphabet à deux lettres ayant comme fonction de transition la fonction `f0` donnée ci-après.

```

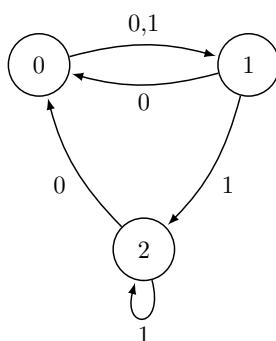
let f0 etat lettre = match etat,lettre with
| 0,0 -> 1
| 0,1 -> 1
| 1,0 -> 0
| 1,1 -> 2
| 2,0 -> 0
| 2,1 -> 2;;
;

f0 : int -> int -> int = <fun>

let M0 = { n_etats = 3 ; n_lettres = 2 ; delta = f0 };;

```

La figure 1 fournit une représentation de la machine  $M_0$ .



**Figure 1** La machine  $M_0$

On pourra observer que les mots 11 et 10 sont tous les deux synchronisants pour la machine  $M_0$ .

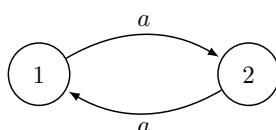
Dans tout le sujet, si une question demande la complexité d'un programme ou d'un algorithme, on attend une complexité temporelle exprimée en  $O(\dots)$ .

## I Considérations générales

**I.A** – Que dire de l'ensemble des mots synchronisants pour une machine ayant un seul état ?

Dans toute la suite du problème, on supposera que les machines ont au moins deux états.

**I.B** – On considère la machine  $M_1$  représentée figure 2. Donner un mot synchronisant pour  $M_1$  s'il en existe un. Justifier la réponse.



**Figure 2** La machine  $M_1$

**I.C** – On considère la machine  $M_2$  représentée figure 3. Donner un mot synchronisant de trois lettres pour  $M_2$ . On ne demande pas de justifier sa réponse.

**I.D** – Écrire une fonction `delta_etoile` de signature `machine -> etat -> mot -> etat` qui, prenant en entrée une machine  $M$ , un état  $q$  et un mot  $u$ , renvoie l'état atteint par la machine  $M$  en partant de l'état  $q$  et en lisant le mot  $u$ .

**I.E** – Écrire une fonction `est_synchronisant` de signature `machine -> mot -> bool` qui, prenant en entrée une machine  $M$  et un mot  $u$ , dit si le mot  $u$  est synchronisant pour  $M$ .

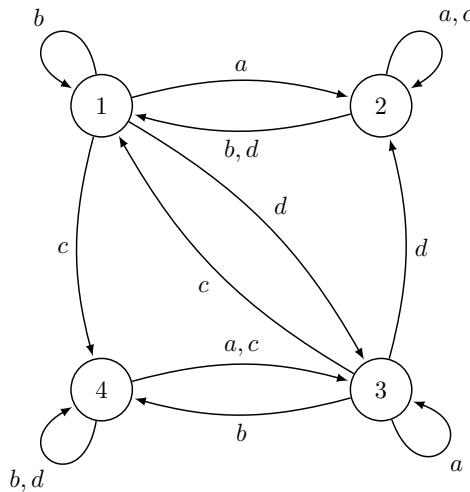
**I.F** – Montrer que pour qu'une machine ait un mot synchronisant, il faut qu'il existe une lettre  $x$  et deux états distincts de  $Q$ ,  $q$  et  $q'$ , tels que  $q.x = q'.x$ .

**I.G** – Soit  $LS(M)$  le langage des mots synchronisants d'une machine  $M = (Q, \Sigma, \delta)$ . On introduit la machine des parties  $\widehat{M} = (\widehat{Q}, \Sigma, \widehat{\delta})$  où  $\widehat{Q}$  est l'ensemble des parties de  $Q$  et où  $\widehat{\delta}$  est définie par

$$\forall P \subset Q, \forall x \in \Sigma, \quad \widehat{\delta}(P, x) = \left\{ \delta(p, x), p \in P \right\}$$

**I.G.1**) Justifier que l'existence d'un mot synchronisant pour  $M$  se ramène à un problème d'accessibilité de certain(s) état(s) depuis certain(s) état(s) dans la machine des parties.

**I.G.2**) En déduire que le langage  $LS(M)$  des mots synchronisants de la machine  $M$  est reconnaissable.

**Figure 3**  $M_2$  : une machine à 4 états

**I.G.3)** Déterminer la machine des parties associée à la machine  $M_0$  puis donner une expression régulière du langage  $LS(M_0)$ .

**I.H –** Montrer que si l'on sait résoudre le problème de l'existence d'un mot synchronisant, on sait dire, pour une machine  $M$  et un état  $q_0$  de  $M$  choisi, s'il existe un mot  $u$  tel que pour tout état  $q$  de  $Q$ , le chemin menant de  $q$  à  $q.u$  passe forcément par  $q_0$ .

## II Algorithmes classiques

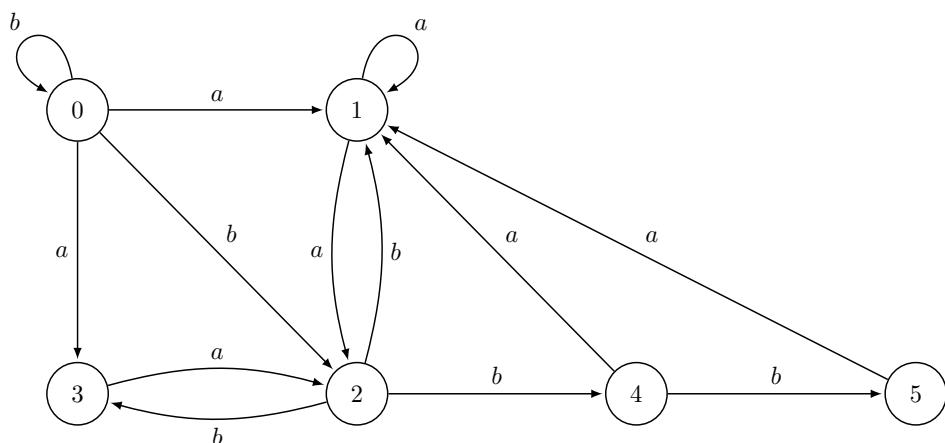
On appellera *graphe d'automate* tout couple  $(S, A)$  où  $S$  est un ensemble dont les éléments sont appelés *sommets* et  $A$  une partie de  $S \times \Sigma \times S$  dont les éléments sont appelés *arcs*. Pour un arc  $(q, x, q')$ ,  $x$  est l'*étiquette* de l'arc,  $q$  son *origine* et  $q'$  son *extrémité*. Un graphe d'automate correspond donc à un automate non déterministe sans notion d'état initial ou d'état final.

Par exemple, avec

$$\begin{aligned}\Sigma &= \{a, b\} \\ S_0 &= \{0, 1, 2, 3, 4, 5\} \\ A_0 &= \{(0, b, 0), (0, a, 3), (0, b, 2), (0, a, 1), (1, a, 1), (1, a, 2), (2, b, 1), \\ &\quad (2, b, 3), (2, b, 4), (3, a, 2), (4, a, 1), (4, b, 5), (5, a, 1)\}\end{aligned}$$

le graphe d'automate  $G_0 = (S_0, A_0)$  est représenté figure 4.

Soient  $s$  et  $s'$  deux sommets d'un graphe  $(S, A)$ . On appelle chemin de  $s$  vers  $s'$  de longueur  $\ell$  toute suite d'arcs  $(s_1, x_1, s'_1), (s_2, x_2, s'_2), \dots, (s_\ell, x_\ell, s'_\ell)$  de  $A$  telle que  $s_1 = s$ ,  $s'_\ell = s'$  et pour tout  $i$  de  $\llbracket 1, \ell - 1 \rrbracket$ ,  $s'_i = s_{i+1}$ . L'étiquette de ce chemin est alors le mot  $x_1 x_2 \dots x_\ell$  et on dit que  $s'$  est accessible depuis  $s$ . En particulier, pour tout  $s \in S$ ,  $s$  est accessible depuis  $s$  par le chemin vide d'étiquette  $\varepsilon$ .

**Figure 4** Le graphe d'automate  $G_0$

Dans les programmes à écrire, un graphe aura toujours pour ensemble de sommets un intervalle d'entiers  $\llbracket 0, n-1 \rrbracket$  et l'ensemble des arcs étiquetés par  $\Sigma$  (comme précédemment supposé être un intervalle  $\llbracket 0, p-1 \rrbracket$ ) sera codé par un vecteur de listes d'adjacence  $V$  : pour tout  $s \in S$ ,  $V.(s)$  est la liste (dans n'importe quel ordre) de tous les couples  $(s', x)$  tel que  $(s, x, s')$  soit un arc du graphe. Pour des raisons de compatibilité ultérieure, les sommets (qui sont, rappelons-le, des entiers) seront codés par le type `etat`.

Ainsi, avec l'alphabet  $\Sigma = \{a, b\}$ , la lettre  $a$  est codée 0 et la lettre  $b$  est codée 1 ; l'ensemble des arcs du graphe  $G_0$ , dont chaque sommet est codé par son numéro, admet pour représentation Caml :

```
V0 : (etat * lettre) list vect = []
  [(0,1); (3,0); (2,1); (1,0)];
  [(1,0); (2,0)];
  [(1,1); (3,1); (4,1)];
  [(2,0)];
  [(1,0); (5,1)];
  [(1,0)]
[]
```

**II.A** – On veut implémenter une file d'attente à l'aide d'un vecteur circulaire. On définit pour cela un type particulier nommé `file` par

```
type 'a file=[tab:'a vect; mutable deb: int; mutable fin: int; mutable vide: bool]
```

`deb` indique l'indice du premier élément dans la file et `fin` l'indice qui suit celui du dernier élément de la file, `vide` indiquant si la file est vide. Les éléments sont rangés depuis la case `deb` jusqu'à la case précédent `fin` en repartant à la case 0 quand on arrive au bout du vecteur (cf exemple). Ainsi, on peut très bien avoir l'indice `fin` plus petit que l'indice `deb`. Par exemple, la file figure 5 contient les éléments 4, 0, 1, 12 et 8, dans cet ordre, avec `fin=2` et `deb=9`.

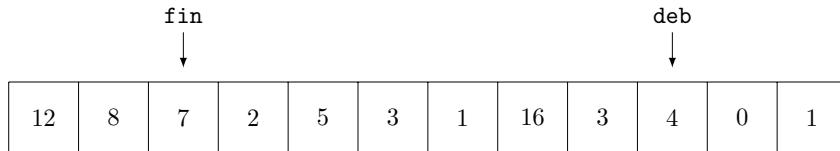


Figure 5 Un exemple de file où `fin < deb`

On rappelle qu'un champ mutable peut voir sa valeur modifiée. Par exemple, la syntaxe `f.deb <- 0` affecte la valeur 0 au champ `deb` de la file `f`.

**II.A.1)** Écrire une fonction `ajoute` de signature `'a file -> 'a -> unit` telle que `ajoute f x` ajoute `x` à la fin de la file d'attente `f`. Si c'est impossible, la fonction devra renvoyer un message d'erreur, en utilisant l'instruction `failwith "File pleine"`.

**II.A.2)** Écrire une fonction `retire` de signature `'a file -> 'a` telle que `retire f` retire l'élément en tête de la file d'attente et le renvoie. Si c'est impossible, la fonction devra renvoyer un message d'erreur.

**II.A.3)** Quelle est la complexité de ces fonctions ?

On considère l'algorithme 1 s'appliquant à un graphe d'automate  $G = (S, A)$  et à un ensemble de sommets  $E$  (on note  $n = |S|$  et  $\infty$ , `vide` et `rien` des valeurs particulières).

**II.B** – Justifier que l'algorithme 1 termine toujours.

**II.C** – Donner la complexité de cet algorithme en fonction de  $|S|$  et  $|A|$ . On justifiera sa réponse.

**II.D** – Justifier qu'au début de chaque passage dans la boucle « `tant que F n'est pas vide` », si  $F$  contient dans l'ordre les sommets  $s_1, s_2, \dots, s_r$ , alors  $D[s_1] \leq D[s_2] \leq \dots \leq D[s_r]$  et  $D[s_r] - D[s_1] \leq 1$ .

**II.E** – Pour  $s$  sommet de  $G$ , on note  $d_s$  la distance de  $E$  à  $s$  c'est-à-dire la longueur d'un plus court chemin d'un sommet de  $E$  à  $s$  (avec la convention  $d_s = \infty$  s'il n'existe pas de tel chemin).

**II.E.1)** Justifier brièvement qu'à la fin de l'algorithme, pour tout sommet  $s$ ,  $D[s] \neq \infty$  si et seulement si  $s$  est accessible depuis un sommet de  $E$  et que  $d_s \leq D[s]$ . Que désigne alors  $c$  ?

**II.E.2)** Montrer qu'en fait, à la fin, on a pour tout sommet  $s$ ,  $D[s] = d_s$ . Que vaut alors  $P[s]$  ?

**II.F** – Écrire une fonction `accessibles` de signature

```
((etat*lettre) list) vect -> etat list -> int * int vect * (etat*lettre) vect
```

prenant en entrée un graphe d'automate (sous la forme de son vecteur de listes d'adjacence  $V$ ) et un ensemble  $E$  de sommets (sous la forme d'une liste d'états) et qui renvoie le triplet  $(c, D, P)$  calculé selon l'algorithme précédent. Les constantes  $\infty$ , `vide` et `rien` seront respectivement codées dans la fonction `accessibles` par  $-1$ ,  $(-2, -1)$  et  $(-1, -1)$ .

créer une file d'attente  $F$ , vide au départ  
 créer un tableau  $D$  dont les cases sont indexées par  $S$  et initialisées à  $\infty$   
 créer un tableau  $P$  dont les cases sont indexées par  $S$  et initialisées à *vide*  
 créer une variable  $c$  initialisée à  $n$

**pour tout**  $s \in E$  faire  
     insérer  $s$  à la fin de la file d'attente  $F$   
     fixer  $D[s]$  à 0  
     fixer  $P[s]$  à *rien*  
     diminuer  $c$  de 1

**fin pour**  
**tant que**  $F$  n'est pas vide faire  
     extraire le sommet  $s$  qui est en tête de  $F$   
     **pour tout** arc  $(s, y, s') \in A$  tel que  $D[s'] = \infty$  faire  
         fixer  $D[s']$  à  $D[s] + 1$   
         fixer  $P[s']$  à  $(s, y)$   
         insérer  $s'$  à la fin de la file d'attente  $F$   
         diminuer  $c$  de 1

**fin pour**  
**fin tant que**  
 renvoyer  $(c, D, P)$

### Algorithm 1

**II.G** – Écrire une fonction **chemin** de signature **etat**  $\rightarrow$  (**etat\*lettre**) **vect**  $\rightarrow$  **mot** qui, prenant en entrée un sommet **s** et le vecteur **P** calculé à l'aide de la fonction **accessibles** sur un graphe  $G$  et un ensemble  $E$ , renvoie un mot de longueur minimale qui est l'étiquette d'un chemin d'un sommet de  $E$  à **s** (ou un message d'erreur s'il n'en existe pas).

## III Réduction SAT

On s'intéresse dans cette partie à la satisfiabilité d'une formule logique portant sur des variables propositionnelles  $x_1, \dots, x_m$ . On note classiquement  $\wedge$  le connecteur logique « et »,  $\vee$  le connecteur « ou » et  $\bar{f}$  la négation d'une formule  $f$ .

On appelle littéral une formule constituée d'une variable  $x_i$  ou de sa négation  $\bar{x}_i$ , on appelle clause une disjonction de littéraux.

Considérons une formule logique sous forme normale conjonctive c'est-à-dire sous la forme d'une conjonction de clauses. Par exemple,

$$F_1 = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_4) \wedge (x_2 \vee \bar{x}_3 \vee x_4)$$

est une formule sous forme normale conjonctive formée de trois clauses et portant sur quatre variables propositionnelles  $x_1, x_2, x_3$  et  $x_4$ .

Soit  $F$  une formule sous forme normale conjonctive, composée de  $n$  clauses et faisant intervenir  $m$  variables. On suppose les clauses numérotées  $c_1, c_2, \dots, c_n$ . On veut ramener le problème de la satisfiabilité d'une telle formule au problème de la recherche d'un mot synchronisant de longueur inférieure ou égale à  $m$  sur une certaine machine. On introduit pour cela la machine suivante associée à  $F$ :

- $Q$  est formé de  $mn + n + 1$  états, un état particulier noté  $f$  et  $n(m + 1)$  autres états qu'on notera  $q_{i,j}$  avec  $(i, j) \in \llbracket 1, n \rrbracket \times \llbracket 1, m + 1 \rrbracket$  ;
- $\Sigma = \{0, 1\}$  ;
- $\delta$  est défini par
  - $f$  est un état *puits*, c'est-à-dire  $\delta(f, 0) = \delta(f, 1) = f$ ,
  - pour tout entier  $i$  de  $\llbracket 1, n \rrbracket$ ,  $\delta(q_{i,m+1}, 0) = \delta(q_{i,m+1}, 1) = f$ ,
  - pour tout  $i$  dans  $\llbracket 1, n \rrbracket$  et  $j$  dans  $\llbracket 1, m \rrbracket$ ,

$$\delta(q_{i,j}, 1) = \begin{cases} f & \text{si le littéral } x_j \text{ apparaît dans la clause } c_i \\ q_{i,j+1} & \text{sinon} \end{cases}$$

$$\delta(q_{i,j}, 0) = \begin{cases} f & \text{si le littéral } \bar{x}_j \text{ apparaît dans la clause } c_i \\ q_{i,j+1} & \text{sinon} \end{cases}$$

**III.A** – Représenter la machine associée à la formule  $F_1$ .

**III.B** – Donner une distribution de vérité  $(v_1, v_2, v_3, v_4) \in \llbracket 0, 1 \rrbracket^4$  (la valeur  $v_i$  étant associée à la variable  $x_i$ ) satisfaisant  $F_1$ . Le mot  $v_1v_2v_3v_4$  est-il synchronisant ?

**III.C** – Montrer que tout mot  $u$  de longueur  $m + 1$  est synchronisant. À quelle condition sur les  $q_{i,1} \cdot u$  un mot  $u$  de longueur  $m$  est-il synchronisant ?

**III.D** – Montrer que si la formule  $F$  est satisfiable, toute distribution de vérité la satisfaisant donne un mot synchronisant de longueur  $m$  pour l'automate.

**III.E** – Inversement, prouver que si l'automate dispose d'un mot synchronisant de longueur inférieure ou égale à  $m$ ,  $F$  est satisfiable. Donner alors une distribution de vérité convenable.

## IV Existence

On reprend dans cette partie le problème de l'existence d'un mot synchronisant pour une machine  $M$ .

**IV.A** – Soit  $M = (Q, \Sigma, \delta)$  une machine.

Pour toute partie  $E$  de  $Q$  et tout mot  $u$  de  $\Sigma^*$ , on note  $E \cdot u = \{q \cdot u, q \in E\}$ .

**IV.A.1**) Soit  $u$  un mot synchronisant de  $M$  et  $u_0, u_1, \dots, u_r$  une suite de préfixes de  $u$  rangés dans l'ordre croissant de leur longueur et telle que  $u_r = u$ . Que peut-on dire de la suite des cardinaux  $|Q \cdot u_i|$  ?

**IV.A.2**) Montrer qu'il existe un mot synchronisant si et seulement s'il existe pour tout couple d'états  $(q, q')$  de  $Q^2$  un mot  $u_{q,q'}$  tel que  $q \cdot u_{q,q'} = q' \cdot u_{q,q'}$ .

On veut se servir du critère établi ci-dessus pour déterminer s'il existe un mot synchronisant. Pour cela, on associe à la machine  $M$  la machine  $\widetilde{M} = (\widetilde{Q}, \Sigma, \tilde{\delta})$  définie par :

- $\widetilde{Q}$  est formé des parties à un ou deux éléments de  $Q$  ;
- $\tilde{\delta}$  est définie par  $\forall (E, x) \in \widetilde{Q} \times \Sigma, \tilde{\delta}(E) = \{\delta(q, x), q \in E\}$ .

**IV.B** – Si  $n = |Q|$ , que vaut  $\tilde{n} = |\widetilde{Q}|$  ?

**IV.C** – On a dit que pour la modélisation informatique, l'ensemble d'états d'une machine doit être modélisée par un intervalle  $\llbracket 0, n - 1 \rrbracket$ .  $\widetilde{Q}$  doit donc être modélisé par l'intervalle  $\llbracket 0, \tilde{n} - 1 \rrbracket$ . Soit  $\varphi_n$  une bijection de  $\widetilde{Q}$  sur  $\llbracket 0, \tilde{n} - 1 \rrbracket$ . On suppose qu'on dispose d'une fonction `set_to_nb` de signature `int -> (etat list) -> etat` telle que `set_to_nb n ℓ` pour  $n$  élément de  $\mathbb{N}^*$  et  $\ell$  liste d'états renvoie

$$\begin{cases} \varphi_n(\{i\}) & \text{si } \ell = [i] \text{ avec } i \in \llbracket 0, n - 1 \rrbracket \\ \varphi_n(\{i, j\}) & \text{si } \ell = [i; j] \text{ avec } (i, j) \in \llbracket 0, n - 1 \rrbracket^2, i < j \end{cases}$$

On suppose qu'on dispose aussi d'une fonction réciproque `nb_to_set` de signature `int -> etat -> (etat list)` telle que `nb_to_set n q` pour  $n$  élément de  $\mathbb{N}^*$  et  $q$  élément de  $\llbracket 0, \tilde{n} - 1 \rrbracket$  renvoie une liste d'états de la forme  $[i]$  ou  $[i; j]$  (avec  $i < j$ ) correspondant à  $\varphi_n^{-1}(q)$ . Ces deux fonctions de conversion sont supposées agir en temps constant.

Enfin, pour ne pas confondre un état de  $\widetilde{Q}$  avec sa représentation informatique par un entier, on notera  $\bar{q}$  l'entier associé à l'état  $q$ .

Écrire une fonction `delta2` de signature `machine -> etat -> lettre -> etat` qui prenant en entrée une machine  $M$ , un état  $\bar{q}$  de  $\widetilde{Q}$  et une lettre  $x$ , renvoie l'état de  $\widetilde{Q}$  atteint en lisant la lettre  $x$  depuis l'état  $q$  dans  $\widetilde{M}$ .

**IV.D** – Il est clair qu'à la machine  $\widetilde{M}$ , on peut associer un graphe d'automate  $\widetilde{G}$  dont l'ensemble des sommets est  $\widetilde{Q}$  et dont l'ensemble des arcs est  $\{(q, x, \tilde{\delta}(q, x)), (q, x) \in \widetilde{Q} \times \Sigma\}$ . On associe alors à  $\widetilde{G}$  le graphe retourné  $\widetilde{G}_R$  qui a les mêmes sommets que  $\widetilde{G}$  mais dont les arcs sont retournés (i.e  $(q, x, q')$  est un arc de  $\widetilde{G}_R$  si et seulement si  $(q', x, q)$  est un arc de  $\widetilde{G}$ ).

Écrire une fonction `retourne_machine` de signature `machine -> ((etat*lettre) list) vect` qui à partir d'une machine  $M$ , calcule le vecteur  $V$  des listes d'adjacence du graphe  $\widetilde{G}_R$ .

**IV.E** – Justifier qu'il suffit d'appliquer la fonction `accessibles` de la partie II au graphe  $\widetilde{G}_R$  et à l'ensemble des sommets de  $\widetilde{G}_R$  correspondant à des singltons pour déterminer si la machine  $M$  possède un mot synchronisant.

**IV.F** – Écrire une fonction `existe_synchronisant` de signature `machine -> bool` qui dit si une machine possède un mot synchronisant.

*Jan Černý, chercheur slovaque, a conjecturé au milieu des années 60 que si une machine à  $n$  états possédait un mot synchronisant, elle en avait un de longueur inférieure ou égale à  $(n - 1)^2$ . La construction faite dans la partie III affirme que la recherche, dans une machine, d'un mot synchronisant de longueur inférieure ou égale à une valeur  $m$  fixée est au moins aussi difficile en terme de complexité que celui de la satisfiabilité d'une formule logique à  $m$  variables sous forme normale conjonctive (qu'on sait être un problème « difficile »).*

**EPREUVE SPECIFIQUE - FILIERE MP****INFORMATIQUE****Jeudi 4 mai : 14 h - 18 h**

---

*N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.*

---

**Les calculatrices sont interdites****Le sujet est composé de trois parties, toutes indépendantes.**

Ce sujet évalue les compétences acquises dans les enseignements d'*Informatique pour tous* et d'option *Informatique*. **Toutes les questions** doivent être traitées par les candidats.

Les questions demandant d'écrire une fonction en langage Python doivent exploiter le contenu des enseignements d'*Informatique pour tous*. Les questions demandant d'écrire une fonction en langage CaML doivent exploiter le contenu des enseignements d'option *Informatique*.

Les fonctions écrites en langage Python ne devront pas être récursives sauf dans la **question Q31** page 9.

Les fonctions écrites en langage CaML devront être récursives ou faire appel à des fonctions auxiliaires récursives. Elles ne devront pas utiliser d'instructions itératives (c'est-à-dire **for, while, ...**), ni de références, ni d'exceptions.

## Partie I - Logique et calcul des propositions

Imaginez-vous ethnologue. Vous étudiez une peuplade primitive qui présente un comportement manichéen extrême : lorsque plusieurs personnes participent à une même conversation sur un sujet donné, elles vont toutes avoir le même comportement manichéen tant que la conversation reste sur le même sujet, c'est-à-dire que toutes les affirmations seront soit des vérités, soit des mensonges. Par contre, si le sujet de la conversation change, la nature des affirmations, soit mensonge, soit vérité, peut changer, mais toutes les affirmations seront de la même nature tant que le sujet ne changera pas à nouveau.

Pour être autorisé à séjourner dans cette peuplade, vous devez respecter cette règle. Vous participez à une conversation avec trois de leurs membres que nous appellerons  $X$ ,  $Y$  et  $Z$ . Ceux-ci vous indiquent comment rejoindre leur village. Si vous n'arrivez pas à le rejoindre, vous ne serez pas autorisé à y séjourner.

Le premier sujet abordé est la région dans laquelle se trouve le village :

$X$  indique : « Le village se trouve dans la vallée » ;

$Z$  réplique : « Non, il ne s'y trouve pas » ;

$X$  reprend : « Ou alors dans les collines ».

Nous noterons  $V$  et  $C$  les variables propositionnelles associées à la région dans laquelle se trouve le village.

Nous noterons  $X_1$  et  $Z_1$  les formules propositionnelles correspondant aux affirmations de  $X$  et de  $Z$  sur le premier sujet.

Puis, le second sujet est abordé : le chemin qui permet de rejoindre le village dans la région concernée.

$X$  dit : « Le chemin de gauche conduit au village » ;

$Z$  répond : « Tu as raison » ;

$X$  complète : « Le chemin de droite y conduit aussi » ;

$Y$  affirme : « Si le chemin du milieu y conduit, alors celui de droite n'y conduit pas » ;

$Z$  indique : « Celui du milieu n'y conduit pas ».

Nous noterons  $G$ ,  $M$ ,  $D$  les variables propositionnelles correspondant respectivement au fait que le chemin de gauche, du milieu et de droite, conduit au village.

Nous noterons  $X_2$ ,  $Y_2$  et  $Z_2$  les formules propositionnelles correspondant aux affirmations de  $X$ , de  $Y$  et de  $Z$  sur le second sujet.

- Q1.** Représenter le comportement manichéen des interlocuteurs dans le premier sujet abordé sous la forme d'une formule du calcul des propositions dépendant des formules propositionnelles  $X_1$  et  $Z_1$ .
- Q2.** Représenter les informations données par les participants sous la forme de deux formules du calcul des propositions  $X_1$  et  $Z_1$  dépendant des variables  $V$  et  $C$ .
- Q3.** En utilisant la résolution avec les propriétés des opérateurs booléens et les formules de De Morgan en calcul des propositions, déterminer dans quelle région vous devez vous rendre pour rejoindre le village.
- Q4.** Représenter le comportement manichéen des interlocuteurs dans le second sujet abordé sous la forme d'une formule du calcul des propositions dépendant des formules propositionnelles  $X_2$ ,  $Y_2$  et  $Z_2$ .
- Q5.** Représenter les informations données par les participants sous la forme de trois formules du calcul des propositions  $X_2$ ,  $Y_2$  et  $Z_2$  dépendant des variables  $G$ ,  $M$  et  $D$ .
- Q6.** En utilisant la résolution avec les tables de vérité en calcul des propositions, déterminer quel chemin vous devez suivre pour rejoindre le village.

- Q7.** En admettant que les trois participants aient menti, pouvez-vous prendre d'autres chemins ?  
Si oui, le ou lesquels ?

## Partie II - Algorithmique et programmation (Informatique pour tous)

Cette partie étudie l'algorithme de recherche dichotomique de la position d'une valeur dans une séquence croissante d'entiers. Pour simplifier les notations et les preuves, les séquences manipulées ne contiennent qu'une seule fois chaque valeur. Les résultats obtenus se généralisent aux séquences croissantes quelconques.

**D finition II.1** (séquence) : soient  $d, f \in \mathbb{N}$  avec  $d \leq f + 1$ ,

- une séquence  $s$  de valeurs  $v_i$  avec  $i \in \mathbb{N}$  et  $d \leq i \leq f$  est notée  $s = \langle v_i \rangle_{i=d}^f$  ;
- sa taille est notée  $|s| = f - d + 1$  ;
- si  $i \in \mathbb{N}$  et  $d \leq i \leq f$ , sa  $i$ -ème valeur est notée  $s_i = v_i$  ;
- son domaine, c'est- -dire l'intervalle des indices de ses valeurs, est noté  $\text{dom}(\langle v_i \rangle_{i=d}^f) = [d, f]$  ;
- son co-domaine, c'est- -dire l'ensemble de ses valeurs, est noté  $\text{codom}(\langle v_i \rangle_{i=d}^f) = \{v_i\}_{i=d}^f$  ;
- la séquence vide de taille 0 est notée  $\langle \rangle$  ;
- si  $d \leq d'$  et  $d' \leq f' + 1$  et  $f' \leq f$ , alors  $\langle v_i \rangle_{i=d'}^{f'}$  de taille  $f' - d' + 1$  désigne la sous-séquence de  $\langle v_i \rangle_{i=d}^f$  contenant les valeurs de  $v_{d'} \dots v_{f'}$ .

Les valeurs contenues dans les séquences  $s$  manipulées par la suite sont toutes distinctes, c'est- -dire que le cardinal du co-domaine de  $s$  est égal à la taille de  $s$  ( $\text{card}(\text{codom}(s)) = |s|$ ).

Une séquence sera représentée en Python par une liste. Dans la suite de cet exercice, nous considérons la fonction `position` du programme suivant en langage Python.

```
def position(v, p):
    d = 0
    f = len(p)-1
    while (d < f):
        m = d + (f - d) // 2
        print (v, d, m, f, p[d], p[m], p[f])
        if (v < p[m]):
            f = m - 1
        elif (p[m] < v):
            d = m + 1
        else:
            f = d = m
    return d

exemple = [ 1, 4, 7, 9, 12, 15]
resultat = position(9, exemple)
```

**Listing 1 – Recherche dichotomique en Python**

- Q8.** Quelles sont les informations affichées lors de l'exécution du programme complet du Listing 1 ? Que contient la variable `resultat` après cette exécution ? Que contient la variable `exemple` après cette exécution ?

**Q9.** Soient la séquence  $p$  et les entiers  $r$  et  $v$  tels que  $r = \text{position}(v, p)$ , avec :

- (i)  $p = \langle p_0, \dots, p_n \rangle$  avec  $n \in \mathbb{N}$ ;
- (ii)  $\forall i \in [0, n[, p_i < p_{i+1}$ ;
- (iii)  $\exists i \in [0, n], p_i = v$ .

Montrer que  $v = p_r$ .

Vous utiliserez pour cela la propriété suivante dont vous montrerez qu'il s'agit d'un invariant pour la boucle :

$$0 \leq d \leq f \leq n \wedge \exists i \in [d, f], p_i = v.$$

**Q10.** Montrer que le calcul de la fonction `position` se termine quelles que soient les valeurs de ses paramètres  $v$  et  $p$ .

**Q11.** Donner des exemples de valeurs des paramètres  $v$  et  $p$  de la fonction `position` qui correspondent au pire cas en temps d'exécution.

Montrer que la complexité en temps d'exécution dans le pire cas de la fonction `position`, en fonction de la taille  $n$  des séquences transmises comme paramètre, est de  $O(\log(n))$ .

## Partie III - Automates et langages

Cette partie étudie l'opérateur de produit synchronisé  $\parallel_S$  sur l'alphabet  $S$  de deux automates sur un même alphabet  $X$  tel que  $S \subseteq X$ . Cet opérateur est utilisé pour modéliser des activités concurrentes.

### 1 Mots et langages

**D finition III.1** (alphabet, mot, langage) : *un alphabet  $X$  est un ensemble de symboles.  $\epsilon \notin X$  est le symbole représentant le mot vide.  $X^*$  est l'ensemble contenant  $\epsilon$  et les mots composés de séquences de symboles de  $X$ . Un langage  $L$  sur  $X$  est un sous-ensemble de  $X^*$ .*

Nous étudierons deux implantations des mots et langages : d'une part en algorithmique récursive programmée en langage CaML ; d'autre part en algorithmique itérative programmée en langage Python.

#### 1.1 Algorithmique r cursive (option Informatique)

Nous utiliserons par la suite en CaML le type `char` pour représenter les symboles de l'alphabet et les listes de symboles pour représenter les mots.

**Q12.** écrire en CaML une définition pour les types `alphabet`, `mot` et `langage` qui représentent les alphabets, mots et langages sur les symboles de cet alphabet.

**Q13.** écrire en CaML une fonction `prefixer`, de type `mot -> langage -> langage`, telle que l'appel `prefixer p l` sur un mot `p` et un langage `l`, renvoie un langage contenant les mêmes mots que `l` préfixés par le mot `p`. L'algorithme utilisé ne devra parcourir qu'une seule fois le langage `l`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

**Q14.** Calculer une estimation de la complexité de la fonction `prefixer` en fonction du nombre de mots du langage `l`. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.

## 1.2 Algorithmique itérative (Informatique pour tous)

Nous utiliserons par la suite en Python des chaînes de caractère de taille 1 (type `str`) pour représenter les symboles de l'alphabet.

**Q15.** Proposer une représentation en Python des mots et langages sur cet alphabet.

**Q16.** écrire en Python une fonction `prefixer`, telle que l'appel `prefixer p,l` sur un mot `p` et un langage `l`, renvoie un langage contenant les mêmes mots que `l` préfixés par le mot `p`. L'algorithme utilisé ne devra parcourir qu'une seule fois le langage `l`. Cette fonction devra être itérative ou faire appel à des fonctions auxiliaires itératives.

**Q17.** Calculer une estimation de la complexité de la fonction `prefixer` en fonction du nombre de mots du langage `l`. Cette estimation ne prendra en compte que le nombre d'itérations effectuées.

## 2 Automate fini

### 2.1 Définition d'un automate fini

**Définition III.2** (automate fini) : un automate fini sur un alphabet  $X$  est un quintuplet  $\mathcal{A} = (Q, X, I, T, \delta)$  composé :

- d'un ensemble fini d'états :  $Q$ ;
- d'un ensemble d'états initiaux :  $I \subseteq Q$ ;
- d'un ensemble d'états terminaux :  $T \subseteq Q$ ;
- d'une relation de transition :  $\delta \subseteq Q \times X \times Q$ .

Pour une transition  $(o, e, d) \in \delta$  donnée, nous appelons  $o$  l'origine de la transition,  $e$  l'étiquette de la transition et  $d$  la destination de la transition.

Remarquons que  $\delta$  est le graphe d'une application de transition  $\delta : Q \times X \rightarrow \mathcal{P}(Q)$  dont les valeurs sont définies par :

$$\forall o \in Q, \forall e \in X, \delta(o, e) = \{d \in Q \mid (o, e, d) \in \delta\}.$$

La notation  $\delta$  est plus adaptée que  $\delta$  à la formalisation et la construction des preuves dans le cadre des automates non déterministes.

### 2.2 Langage accepté par un automate fini

**Définition III.3** (transition sur un mot) : l'extension  ${}^*$  de  $\delta : Q \times X^* \times Q$  est définie par :

- fermeture réflexive : pour tout état  $q$  de  $Q$ ,

$$(q, \epsilon, q) \in {}^*$$

- fermeture transitive : pour tout symbole  $e$  de  $X$ , pour tout mot  $m$  de  $X^*$ , pour tous états  $o, d$  de  $Q$ ,

$$(o, e.m, d) \in {}^* \Leftrightarrow \exists q \in Q, ((o, e, q) \in \delta) \wedge ((q, m, d) \in \delta^*).$$

**Définition III.4** (langage accepté par un automate) : le langage sur  $X$  accepté par un automate fini  $\mathcal{A}$  est :

$$L(\mathcal{A}) = \{m \in X^* \mid \exists i \in I, \exists d \in T, (i, m, d) \in \delta^*\}.$$

### 2.3 Représentation graphique d'un automate

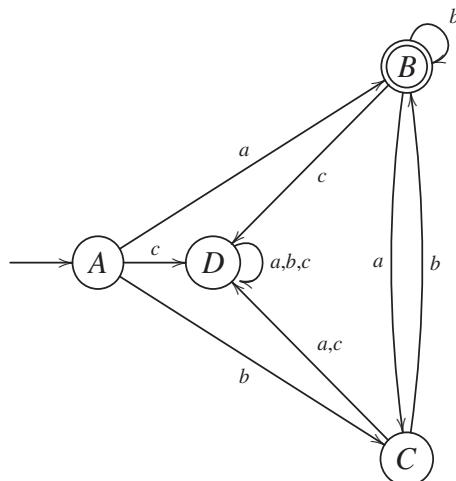
Les automates peuvent être représentés par un schéma suivant les conventions :

- les valeurs de la relation de transition sont représentées par un graphe orienté dont les nœuds encerclés sont les états et les arêtes sont les transitions ;
- tout état initial  $i \in I$  est désigné par une flèche  $\xrightarrow{} i$ ;
- tout état terminal  $t$  est entouré d'un double cercle  $(t)$ ;
- une arête étiquetée par le symbole  $e \in X$  va de l'état  $o$  à l'état  $d$  si et seulement si  $(o, e, d) \in \delta$ .

**Exemple III.1 :** l'automate  $\mathcal{E}_1 = (Q_1, X, I_1, T_1, \delta_1)$  tel que

$$\begin{aligned} Q_1 &= \{A, B, C, D\} \quad X = \{a, b, c\} \quad I_1 = \{A\} \quad T_1 = \{B\} \\ \delta_1 &= \left. \begin{array}{l} (A, a, B), \quad (A, b, C), \quad (A, c, D), \quad (B, a, C), \quad (B, b, B), \quad (B, c, D), \\ (C, a, D), \quad (C, b, B), \quad (C, c, D), \quad (D, a, D), \quad (D, b, D), \quad (D, c, D) \end{array} \right\} \end{aligned}$$

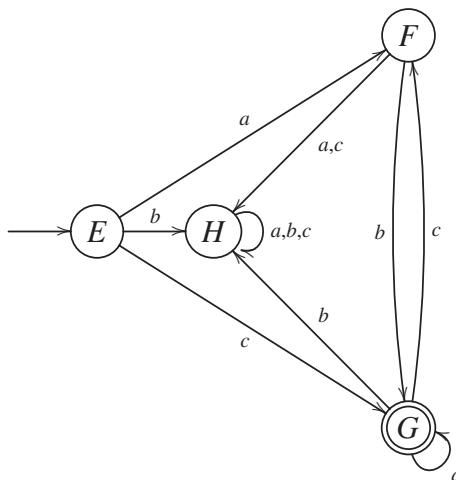
est représenté par le graphe suivant :



**Exemple III.2 :** l'automate  $\mathcal{E}_2 = (Q_2, X, I_2, T_2, \delta_2)$  tel que

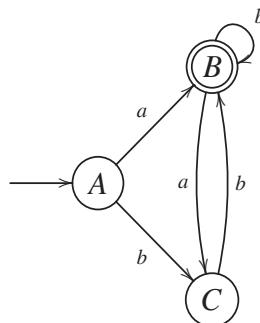
$$\begin{aligned} Q_2 &= \{E, F, G, H\} \quad X = \{a, b, c\} \quad I_2 = \{E\} \quad T_2 = \{G\} \\ \delta_2 &= \left. \begin{array}{l} (E, a, F), \quad (E, b, H), \quad (E, c, G), \quad (F, a, H), \quad (F, b, G), \quad (F, c, H), \\ (G, a, G), \quad (G, b, H), \quad (G, c, F), \quad (H, a, H), \quad (H, b, H), \quad (H, c, H) \end{array} \right\} \end{aligned}$$

est représenté par le graphe suivant :

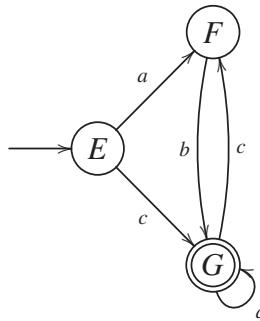


Notons que certains états et transitions ne sont pas utiles dans la description d'un langage car ils ne participent pas à la construction de mots du langage. Il s'agit, d'une part, des états qui ne figurent dans aucune séquence de transitions allant de l'état initial à un état terminal et, d'autre part, des transitions dont les états inutiles sont l'origine ou la destination. Un automate dans lequel ces états et transitions inutiles ont été éliminés est appelé automate émondé.

**Exemple III.3 :** le sous-automate de  $\mathcal{E}_1$  (**Exemple III.1** de la page 6) composé des états et transitions utiles est représenté par le graphe suivant :



**Exemple III.4 :** le sous-automate de  $\mathcal{E}_2$  (**Exemple III.2** de la page 6) composé des états et transitions utiles est représenté par le graphe suivant :



**Q18.** Donner, sans la justifier, une expression régulière ou ensembliste représentant les langages  $L(\mathcal{E}_1)$  et  $L(\mathcal{E}_2)$  sur  $X = \{a, b, c\}$  accepté par l'automate  $\mathcal{E}_1$  de l'**Exemple III.1** et  $\mathcal{E}_2$  de l'**Exemple III.2**, situés sur la page 6.

Nous étudierons deux implémentations des automates : d'une part en algorithmique récursive programmée en langage CaML ; d'autre part en algorithmique itérative programmée en langage Python.

## 2.4 Algorithmique récursive (option Informatique)

Nous utiliserons par la suite en CaML le type `int` pour représenter les états d'un automate.

**Q19.** créer en CaML une définition pour les types `etat`, `transition` et `automate` qui représentent les états et les transitions ainsi que les automates sur l'alphabet des symboles. Définir en CaML la valeur `expl_III_1` de type `automate` correspondant à l'automate de l'**Exemple III.1** de la page 6.

**Q20.** créer en CaML une fonction `valider`, de type `automate -> bool`, telle que l'appel `valider a` sur l'automate `a` renvoie la valeur `true` si l'automate `a` est valide, c'est-à-dire s'il respecte les contraintes de la **Définition III.2** de la page 5, et la valeur `false` sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois les transitions de l'automate `a`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

- Q21.** créer en CaML une fonction `accepter`, de type `mot -> automate -> bool`, telle que l'appel `accepter m a` sur un mot `m` et un automate `a`, renvoie la valeur `true` si le mot `m` appartient au langage accepté par l'automate `a` et la valeur `false` sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois le mot `m`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.
- Q22.** Calculer une estimation de la complexité de la fonction `accepter` en fonction du nombre de symboles du mot `m` et du nombre de transitions de l'automate `a`. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.

## 2.5 Algorithmique itérative (Informatique pour tous)

Nous utiliserons par la suite en Python le type `int` pour représenter les états d'un automate.

- Q23.** Proposer une représentation en Python des automates sur l'alphabet des symboles. Définir en Python la variable `expl_III_1` initialisée avec une valeur correspondant à l'automate de l'**Exemple III.1** de la page 6.
- Q24.** créer en Python une fonction `valider`, telle que l'appel `valider a` sur l'automate `a` renvoie la valeur `true` si l'automate `a` est valide, c'est-à-dire s'il respecte les contraintes de la **Définition III.2** de la page 5, et la valeur `false` sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois les transitions de l'automate `a`. Cette fonction devra être itérative ou faire appel à des fonctions auxiliaires itératives.
- Q25.** créer en Python une fonction `accepter`, telle que l'appel `accepter m, a` sur un mot `m` et un automate `a`, renvoie la valeur `true` si l'automate `a` accepte le mot `m` et la valeur `false` sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois le mot `m`. Cette fonction devra être itérative ou faire appel à des fonctions auxiliaires itératives.

## 3 Synchronisation de mots

### 3.1 Définition et Propriétés

Nous définissons d'abord l'opération de synchronisation de mots qui produit un langage.

**Définition III.5** (synchronisation de mots) : soient  $m_1$  et  $m_2$  deux mots de  $X^*$  et  $S \subseteq X$  un alphabet de synchronisation, l'opération de synchronisation de  $m_1$  et  $m_2$  sur  $S$ , notée  $m_1 \|_S m_2$ , produit un langage sur  $X$  tel que : pour tous mots  $m, m_1, m_2$  de  $X^*$ , pour tous symboles  $s, s_1, s_2$  de  $S$  avec  $s_1 \neq s_2$  et pour tous symboles  $x, x_1, x_2$  de  $X \setminus S$ , nous avons :

$$\begin{aligned} m_1 \|_S m_2 &= m_2 \|_S m_1 \\ \epsilon \|_S \epsilon &= \{\epsilon\} \\ (s.m) \|_S \epsilon &= \emptyset \\ (x.m) \|_S \epsilon &= x.(m \|_S \epsilon) \\ (s.m_1) \|_S (s.m_2) &= s.(m_1 \|_S m_2) \\ (s_1.m_1) \|_S (s_2.m_2) &= \emptyset \\ (s.m_1) \|_S (x.m_2) &= x.((s.m_1) \|_S m_2) \\ (x_1.m_1) \|_S (x_2.m_2) &= x_1.(m_1 \|_S (x_2.m_2)) \cup x_2.((x_1.m_1) \|_S m_2). \end{aligned}$$

- Q26.** Soient les alphabets  $X = \{a, b, c, d\}$  et  $S = \{b\}$ , construire le langage  $(a.b.c) \|_S (a.b.d)$  en détaillant chaque étape.

**Q27.** Montrer que pour tous mots  $m_1, m_2$  de  $X^*$ ,

$$\epsilon \in m_1 \parallel_S m_2 \Leftrightarrow ((m_1 = \epsilon) \wedge (m_2 = \epsilon)).$$

**Q28.** Montrer que pour tout symbole  $s$  de  $S$ , pour tous mots  $m, m_1, m_2$  de  $X^*$ , il existe des mots  $m'_1, m'_2$  de  $X^*$  tels que :

$$s.m \in m_1 \parallel_S m_2 \Leftrightarrow ((m_1 = s.m'_1) \wedge (m_2 = s.m'_2)) \wedge (m \in m'_1 \parallel_S m'_2).$$

**Q29.** Montrer que pour tout symbole  $x$  de  $X \setminus S$ , pour tous mots  $m, m_1, m_2$  de  $X^*$ , il existe des mots  $m'_1, m'_2$  de  $X^*$  tels que :

$$x.m \in m_1 \parallel_S m_2 \Leftrightarrow ((m_1 = x.m'_1) \wedge (m \in m'_1 \parallel_S m_2)) \vee ((m_2 = x.m'_2) \wedge (m \in m_1 \parallel_S m'_2)).$$

Nous nous limiterons à l'étude d'une implantation en algorithmique récursive que nous programme-rons à la fois en langage CaML et en langage Python.

### 3.2 Implantation r cursive en langage CaML (option Informatique)

**Q30.** crire en CaML une fonction `synchro_mot`, de type `mot -> mot -> alphabet -> langage`, telle que l'appel `synchro_mot m1 m2 s` sur les mots  $m_1$  et  $m_2$  et sur l'alphabet de synchronisation  $s$ , renvoie le langage  $m \parallel_s m_2$ . L'algorithme utilisé ne devra parcourir qu'une seule fois les mots  $m_1$  et  $m_2$ . Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

### 3.3 Implantation r cursive en langage Python (Informatique pour tous)

**Q31.** crire en Python une fonction `synchro_mot`, telle que l'appel `synchro_mot m1,m2,s` sur les mots  $m_1$  et  $m_2$  et sur l'alphabet de synchronisation  $s$ , renvoie le langage  $m \parallel_s m_2$ . L'algorithme utilisé ne devra parcourir qu'une seule fois les mots  $m_1$  et  $m_2$ . Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

## 4 Synchronisation de langages

Soit l'opération interne de synchronisation sur  $S$  des langages définie par :

**D finition III.6** (synchronisation de langages) : soient  $L_1$  et  $L_2$  deux langages sur l'alphabet  $X$  et  $S \subseteq X$  un alphabet de synchronisation, le langage  $L_1 \parallel_S L_2$  sur  $X$  résultant de la synchronisation sur  $S$  des mots de  $L_1$  et  $L_2$  est défini par :

$$L_1 \parallel_S L_2 = \bigcup_{m_1 \in L_1, m_2 \in L_2} m_1 \parallel_S m_2.$$

**Q32.**crire en CaML une fonction `synchro_lang`, de type `langage -> langage -> alphabet -> langage`, telle que l'appel `synchro_lang l1 l2 s`) sur les langages  $l_1$  et  $l_2$  et sur l'alphabet de synchronisation  $s$ , renvoie le langage  $l_1 \parallel_s l_2$ . Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

## 5 Synchronisation d'automates

### 5.1 Définition

Soit l'opération interne sur les automates finis déterministes définie par :

**D définition III.7** (synchronisation d'automates) : soient  $\mathcal{A}_1 = (Q_1, X, I_1, T_1, \delta_1)$  et  $\mathcal{A}_2 = (Q_2, X, I_2, T_2, \delta_2)$  deux automates finis déterministes sur l'alphabet  $X$  et  $S \subseteq X$  un alphabet de synchronisation, l'automate qui résulte de la synchronisation sur  $S$  de  $\mathcal{A}_1$  et  $\mathcal{A}_2$  est  $\mathcal{A} = (Q_1 \times Q_2, X, I_1 \times I_2, T_1 \times T_2, \delta_{\mathcal{A}})$  où la relation de transition  $\delta_{\mathcal{A}}$  est définie par : pour tous états  $o_1, d_1$  de  $Q_1$  et  $o_2, d_2$  de  $Q_2$ , pour tout symbole  $s$  de  $S$  et pour tout symbole  $x$  de  $X \setminus S$  :

$$((o_1, o_2), s, (d_1, d_2)) \in \delta_{\mathcal{A}} \Leftrightarrow ((o_1, s, d_1) \in \delta_1 \wedge (o_2, s, d_2) \in \delta_2)$$

$$((o_1, o_2), x, (d_1, d_2)) \in \delta_{\mathcal{A}} \Leftrightarrow ((o_1, x, d_1) \in \delta_1 \wedge o_2 = d_2) \vee (o_1 = d_1 \wedge (o_2, x, d_2) \in \delta_2).$$

**Q33.** En considérant l'**Exemple III.1** et l'**Exemple III.2** de la page 6, construire l'automate  $\mathcal{E}_1 \parallel_S \mathcal{E}_2$  avec  $S = \{b\}$ . Le résultat devra être émondé (seuls les états et les transitions utiles devront être construits).

**Q34.**crire en CaML une fonction `synchro_auto`, de type `automate -> automate -> alphabet -> automate`, telle que l'appel `synchro_auto a1 a2 s`) sur les automates  $a_1$  et  $a_2$  et sur l'alphabet de synchronisation  $s$ , renvoie l'automate  $a \parallel_s a_2$ . Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

### 5.2 Propriétés

**Q35.**Montrer que si  $\mathcal{A}_1$  et  $\mathcal{A}_2$  sont des automates finis, alors  $\mathcal{A}_1 \parallel_S \mathcal{A}_2$  est un automate fini.

**Q36.**Montrer que pour tout mot  $m$  de  $X^*$ , il existe des mots  $m_1, m_2$  de  $X^*$  tels que : pour tous états  $o_1, d_1$  de  $Q_1$  et  $o_2, d_2$  de  $Q_2$  :

$$((o_1, o_2), m, (d_1, d_2)) \in \delta_{\mathcal{A}}^* \Leftrightarrow (m \in m_1 \parallel_S m_2 \wedge (o_1, m_1, d_1) \in \delta_1^* \wedge (o_2, m_2, d_2) \in \delta_2^*).$$

**Q37.**Soient  $\mathcal{A}_1$  et  $\mathcal{A}_2$  deux automates finis, montrer que :

$$L(\mathcal{A}_1 \parallel_S \mathcal{A}_2) = L(\mathcal{A}_1) \parallel_S L(\mathcal{A}_2).$$

**FIN**

**Exercice 1.**

On suppose défini le type arbre de la manière suivante :

```
type arbre = Feuille of int | Noeud of arbre * arbre;;
```

On dit qu'un arbre est un *peigne* si tous les noeuds à l'exception éventuelle de la racine ont au moins une feuille pour fils. On dit qu'un peigne est *strict* si sa racine a au moins une feuille pour fils, ou si il est réduit à une feuille. On dit qu'un peigne est *rangé* si le fils droit d'un noeud est toujours une feuille. Un arbre réduit à une feuille est un peigne rangé.

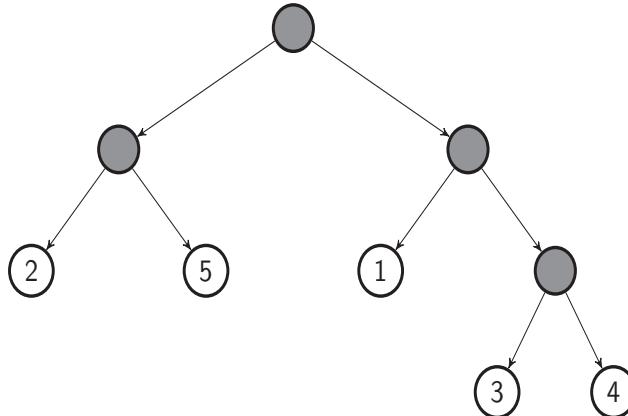


Figure 1 : Un peigne à cinq feuilles.

1. Représenter un peigne rangé à 5 feuilles.
2. La *hauteur* d'un arbre est le nombre de noeuds maximal que l'on rencontre pour aller de la racine à une feuille (la hauteur d'une feuille seule est 0). Quelle est la hauteur d'un peigne rangé à  $n$  feuilles ? On justifiera la réponse.
3. Écrire une fonction `est_range : arbre -> bool` qui renvoie `true` si l'arbre donné en argument est un peigne rangé.
4. Écrire une fonction `est_peigne_strict : arbre -> bool` qui renvoie `true` si l'arbre donné en argument est un peigne strict. En déduire une fonction `est_peigne : arbre -> bool` qui renvoie `true` si l'arbre donné en argument est un peigne.
5. On souhaite ranger un peigne donné. Supposons que le fils droit  $N$  de sa racine ne soit pas une feuille. Notons  $A_1$  le sous-arbre gauche de la racine,  $f$  l'une des feuilles du noeud  $N$  et  $A_2$  l'autre sous-arbre du noeud  $N$ . On va utiliser l'opération de *rotation* qui construit un nouveau peigne où
  - le fils droit de la racine est le sous-arbre  $A_2$  ;
  - le fils gauche de la racine est un noeud de sous-arbre gauche  $A_1$  et de sous-arbre droit la feuille  $f$ .

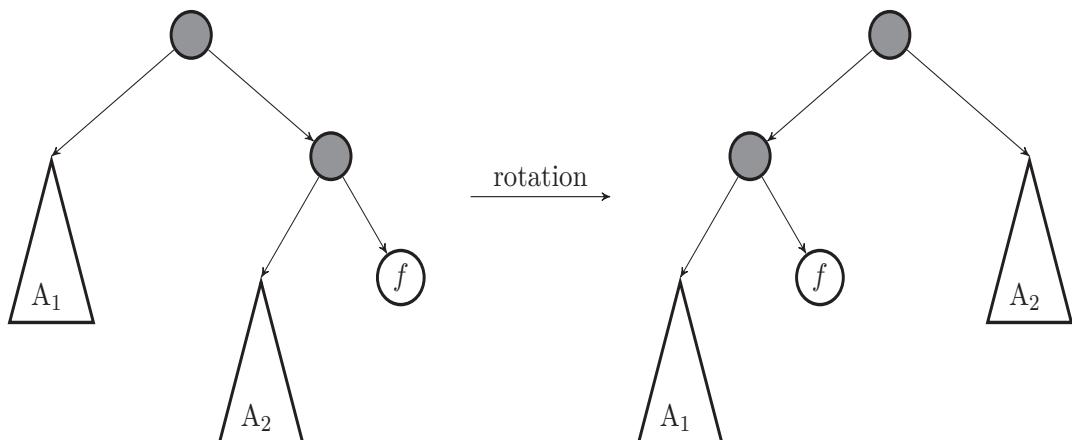


Figure 2 : Une rotation.

- (a) Donner le résultat d'une rotation sur l'arbre de la Figure 1.
- (b) Écrire une fonction `rotation : arbre -> arbre` qui effectue l'opération décrite ci-dessus. La fonction renverra l'arbre initial si une rotation n'est pas possible.
- (c) Écrire une fonction `rangement : arbre -> arbre` qui range un peigne donné en argument, c'est à dire qu'il renvoie un peigne rangé ayant les mêmes feuilles que celui donné en argument. La fonction renverra l'arbre initial si celui-ci n'est pas un peigne.

### Exercice 2.

Dans une classe de  $n$  élèves, les élèves sont numérotés de 0 à  $n - 1$ . Un professeur souhaite faire l'appel, c'est-à-dire déterminer quels élèves sont absents.

#### Partie A.

1. Écrire une fonction `mini : int list -> (int * int list)`, qui prend en argument une liste non vide d'entiers distincts, et renvoie le plus petit élément de cette liste, ainsi que la liste de départ privée de cet élément (pas forcément dans l'ordre initial).
2. En notant  $k$  la longueur de la liste donnée en argument, quelle est la complexité en nombre de comparaisons de la fonction précédente ?
3. En utilisant la fonction `mini`, écrire une fonction `absents : int list -> int -> int list` qui, étant donné une liste non vide d'entiers distincts et  $n$ , renvoie, dans un ordre quelconque, la liste des entiers de  $[0; n - 1]$  qui n'y sont pas.
4. En notant  $k$  la longueur de la liste donnée en argument, quelle est la complexité en nombre de comparaisons (en fonction de  $n$  et  $k$ ) de la fonction précédente ?

#### Partie B.

Dans cette partie, une salle de classe pour  $n$  élèves est décrite par la donnée d'un tableau à  $n$  entrées. Si `tab` est un tel tableau et  $i$  un entier de  $[0; n - 1]$ , alors `tab.(i)` donne le numéro de l'élève assis à la place  $i$  (ou  $-1$  si cette place est vide).

5. Écrire une fonction `asseoir : int list -> int -> int vect`, qui prend en argument une liste non vide d'entiers distincts, un entier  $n$ , et renvoie un tableau représentant une salle de classe pour  $n$  élèves où chaque élève de la liste a été assis à la place numérotée par son propre numéro. Les entiers supérieurs ou égaux à  $n$  seront ignorés.
6. En déduire une fonction `absent2 : int list -> int -> int list` qui étant donné une liste non vide d'entiers distincts et un entier  $n$ , renvoie la liste des entiers de  $[0; n - 1]$  qui n'y sont pas. Les entiers supérieurs ou égaux à  $n$  seront ignorés.
7. En notant  $k$  la longueur de la liste donnée en argument, quelle est la complexité en nombre de lectures et d'écritures dans un tableau (en fonction de  $k$  et  $n$ ) de la fonction précédente ?

#### Partie C.

Dans cette partie, indépendante des précédentes, les élèves sont déjà assis en classe.

8. On considère la fonction `place : int vect -> unit` ci-dessous :

```

let rec place tab i =
  if i <> -1 then
    begin
      let temp = tab.(i) in
      tab.(i) <- i;
      place tab temp
    end;;

```

On note `classe` le tableau `[-1;4;5;-1;3;0]` (on suppose  $n = 6$  dans cette question). Donner le tableau `classe` après l'exécution de `place classe 1`. On donnera l'état de la variable `classe` à chaque appel récursif de la fonction.

9. On considère la fonction `placement` : `int vect -> int -> unit` ci-dessous :

```

let placement tab n =
  for i = 0 to n - 1 do
    if tab.(i) <> -1 && tab.(i) <> i then
      begin
        let temp = tab.(i) in
        tab.(i) <- -1;
        place tab temp
      end
  done;;

```

Si `classe` est un tableau d'entiers (les entiers positifs sont distincts) représentant une classe, que fait `placement classe` ?

### Exercice 3.

On rappelle la définition de la suite de Fibonacci :

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} \text{ pour } n \geq 2 \end{cases}$$

#### Partie A. Calcul des termes de la suite.

1. On considère la fonction `fibo` : `int -> int` suivante :

```

let rec fibo = function
| 0 -> 0
| 1 -> 1
| n -> fibo (n-1) + fibo (n-2);;

```

Pourquoi est-il une mauvaise idée d'utiliser cette fonction pour calculer  $F_n$  ?

2. Écrire une fonction `fibo2` : `int -> int` qui, étant donné  $n$ , calcule  $F_n$  en effectuant un nombre linéaire en  $n$  d'additions.

### Partie B. Décomposition de Zeckendorf.

Pour  $n \in \mathbb{N}$ , on appelle décomposition de Zeckendorf de  $n$  une décomposition de  $n$  comme somme de termes distincts (d'indices supérieurs ou égal à 2) de la suite de Fibonacci, de telle manière qu'il n'y ait pas deux termes d'indices consécutifs dans la somme. Autrement dit, il existe des indices  $c_0, c_1, \dots, c_k$  tels que :

- $c_0 \geq 2$ ,
- pour tout  $i < k : c_{i+1} > c_i + 1$  (pas d'indices consécutifs),
- $n = \sum_{i=0}^k F_{c_i}$

Par exemple :  $2 + 5 = F_3 + F_5$  est une décomposition de Zeckendorf de 7 (avec  $c_0 = 3$  et  $c_1 = 5$ ), alors que  $3 + 5 = F_4 + F_5$  n'est pas une décomposition de Zeckendorf de 8, Car  $F_4$  et  $F_5$  sont deux termes consécutifs de la suite de Fibonacci.

3. Déterminer une décomposition de Zeckendorf de 20, 21 et 22.
4. Montrer que tout entier strictement positif admet une décomposition de Zeckendorf (on admet qu'elle est unique).

### Partie C. Codage de Fibonacci.

On appelle codage de Fibonacci d'un entier positif  $k$  un tableau `tab` de 0 et de 1 indiquant par des 1 les indices des termes de la suites de Fibonacci utilisés dans la décomposition de Zeckendorf de  $k$  (`tab.(0)` indique alors si  $F_2$  est utilisé dans la représentation). On remarque que par définition de la décomposition de Zeckendorf, `tab` ne peut pas contenir deux 1 consécutifs.

`[|0,1,0,1|]` est ainsi une représentation de Fibonacci de 7.

5. Écrire une fonction `decode` : `int vect -> int` qui traduit en entier une représentation de Fibonacci d'un entier.
6. (a) Décrire, sans l'implémenter, une fonction `plusun` : `int vect -> int vect`, qui à partir d'une représentation de Fibonacci d'un entier  $k$ , renvoie une représentation de Fibonacci de l'entier  $k + 1$ .
- (b) Décrire, sans l'implémenter, une fonction `moinsun` : `int vect -> int vect`, qui à partir d'une représentation de Fibonacci d'un entier  $k$  non-nul, renvoie une représentation de Fibonacci de l'entier  $k - 1$ .

### Exercice 4.

Dans cet exercice, les programmes seront écrits avec le langage python. On pourra utiliser si besoin la bibliothèque numpy :

- `zeros([n,p])` renvoie un tableau bidimensionnel (n,p) rempli de 0.
- Si `T` est un tableau bidimensionnel, `T[i,j]` accède à l'élément ligne  $i$  colonne  $j$  de ce tableau.

Le réseau du métro d'une grande ville comporte  $N$  stations réparties un certains nombre de lignes interconnectées (par exemple, pour la ville de Lyon on a  $N = 40$  avec 4 lignes).

On représente ce réseau par un graphe non orienté  $G = ([0, N - 1], A)$  défini par :

- L'ensemble des sommets est  $[0, N - 1]$ . On a numéroté ces stations de 0 à  $N - 1$ . Le sommet  $i$  du graphe  $G$  représente la station  $i$ .

- A désigne l'ensemble des arêtes de G. Une arête entre les sommets  $i$  et  $j$  n'existe que si les stations  $i$  et  $j$  sont des stations adjacentes sur une même ligne de métro, pour tous  $i, j$  dans  $\llbracket 0, N - 1 \rrbracket$ . Ce graphe est représenté par la liste Liste\_A (donnée en variable globale) de ses arêtes sous la forme  $[i, j]$  pour  $i, j$  dans  $\llbracket 0, N - 1 \rrbracket$  tels que  $i < j$  et  $(i, j) \in A$ .
1. Écrire une fonction voisin\_G qui prend en entrée un entier  $i$  dans  $\llbracket 0, N - 1 \rrbracket$  et retourne la liste des sommets adjacents au sommet  $i$  dans le graphe G.

On suppose le graphe G connexe, et on cherche à déterminer le trajet le plus rapide entre deux stations du réseau :

Soit duree la fonction définie sur A qui attribue à une arête  $(i, j)$  la durée du trajet entre la station  $i$  et la station  $j$  en minutes, arrondie sur un nombre entier.

Un trajet entre deux stations  $i$  et  $j$  correspond à un chemin dans G qui part de  $i$  et arrive en  $j$ . La durée d'un tel trajet  $i = i_0 \rightarrow i_1 \rightarrow \dots \rightarrow i_n = j$  utilisant  $n$  arêtes est la somme des durées des  $n$  arêtes :

$$duree(i = i_0 \rightarrow i_1 \rightarrow \dots \rightarrow i_n = j) = \sum_{k=0}^{n-1} duree((i_k, i_{k+1})).$$

Pour simplifier, on ne tient pas compte des durées correspondant aux changements de métro.

On dispose de la liste Duree des listes  $[i, j, duree(i, j)]$ , pour  $i, j$  dans  $\llbracket 0, N - 1 \rrbracket$  tels que  $i < j$  et  $(i, j) \in A$ . On note D la valeur de sum([U[2] for U in Duree]).

2. Soient  $i$  et  $j$  deux sommets de G. Démontrer qu'il existe au moins un trajet de durée minimale entre  $i$  et  $j$ .

Pour  $i$  et  $j$  deux sommets de G, on note  $\delta_{min}(i, j)$  la durée minimale d'un trajet entre  $i$  et  $j$ .

3. Soient  $i$  et  $j$  deux sommets de G. Soit  $i = i_0 \rightarrow i_1 \rightarrow \dots \rightarrow i_n = j$  un chemin dans G qui part de  $i$  et arrive en  $j$  en utilisant  $n$  arêtes ( $n \geq 1$ ). On suppose que ce chemin réalise un trajet de durée minimale entre  $i$  et  $j$ . Justifier que pour tout  $k$  entre 1 et  $n$ ,  $i = i_0 \rightarrow \dots \rightarrow i_k$  est un trajet de durée minimale entre  $i_0$  et  $i_k$  et  $i_k \rightarrow \dots \rightarrow i_n = j$  est un trajet de durée minimale entre  $i_k$  et  $j$ .
4. En déduire, pour  $i$  et  $j$  distincts tels que  $(i, j) \notin A$ , une expression de  $\delta_{min}(i, j)$  en fonction des valeurs de  $\delta_{min}(i, k)$  et  $\delta_{min}(k, j)$ ,  $k$  parcourant la liste des sommets du graphe G, à l'exception de  $i$  et  $j$ . Justifier votre réponse.
5. Définir Tinit le tableau bidimensionnel  $(N, N)$  tel que :

$$\forall (i, j) \in \llbracket 0, N - 1 \rrbracket^2, Tinit[i, j] = \begin{cases} 0, & \text{si } i = j, \\ duree(i, j), & \text{si } (i, j) \in A, \\ D, & \text{sinon.} \end{cases}$$

6. Définir la fonction FW qui prend en entrée un tableau bidimensionnel  $(N, N)$  T et retourne le tableau bidimensionnel  $(N, N)$  T' défini par :

$$\forall (i, j) \in \llbracket 0, N - 1 \rrbracket^2, T'[i, j] = \min(T[i, j], T[i, k] + T[k, j], k \in \llbracket 0, N - 1 \rrbracket).$$

7. Écrire un programme qui, en utilisant le tableau Tinit et la fonction FW, permet de calculer un tableau bidimensionnel  $(N, N)$  dont le  $(i, j)$ -ème coefficient vaut  $\delta_{min}(i, j)$ , pour tous  $i$  et  $j$ , sommets de G. Combien d'itérations sont-elles nécessaires pour conclure ?
8. Expliquer comment modifier le programme pour obtenir un trajet de durée minimale entre  $i$  et  $j$ , pour tous  $i$  et  $j$ , sommets de G. On ne demande pas de le programmer précisément, mais d'expliquer ce qu'il faudrait ajouter au programme précédent pour obtenir de plus ces informations.

# CONCOURS EPITA-IPSA 2017

## Consignes Python

Tout code doit être écrit dans le langage Python.

- Tout code Python non indenté ne sera pas corrigé.
- Tout ce dont vous avez besoin (fonctions, méthodes) est indiqué ci-dessous.
- Vous pouvez écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).

Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.

### Fonctions et méthodes autorisées

Vous pouvez utiliser la fonction range :

```
>>> for i in range(10):
...:     print(i, end=' ')
0 1 2 3 4 5 6 7 8 9

>>> for i in range(5, 10):
...:     print(i, end=' ')
5 6 7 8 9
```

Sur les listes, vous pouvez utiliser la méthode append et la fonction len :

```
>>> help ( list . append )
Help on method_descriptor : append (...)
L. append ( object ) -> None -- append object to end of L

>>> help (len)
Help on built-in function len in module builtins : len (...)
len ( object )
Return the number of items of a sequence or collection .
```

Les matrices sont représentées par des listes de listes comme dans l'exemple ci-dessous :

```
>>> M1 = [[1, 10, 3, 0, 3, 10, 1],
           [1, 0, 1, 8, 1, 0, 1],
           [10, 9, 4, 1, 4, 9, 10],
           [10, 3, 7, 1, 7, 3, 10],
           [7, 8, 5, 1, 5, 8, 7]]
```

# CONCOURS EPITA-IPSA 2017

Note : \_\_\_\_\_ / \_\_\_\_\_

Centre d'écrit : \_\_\_\_\_

Épreuve : ABF;A@;@8AD? 3F;CG7ŽC5?

Nom : \_\_\_\_\_

Prénom : \_\_\_\_\_

N° de candidat : \_\_\_\_\_

N° de copie : \_\_\_\_\_ sur \_\_\_\_\_

Classe actuelle : \_\_\_\_\_

Ce qcm est un peu particulier: chaque question peut comporter 0, 1, ou plusieurs bonnes réponses! Pour gagner des points, il faut répondre correctement (les mauvaises réponses feront perdre des points).

La majorité des questions suivantes sont formulées au singulier ou au pluriel, par commodité pour la grammaire française, mais sans corrélation directe avec le nombre de réponses correctes.

**?** Citez le(s) nom(s) de créateur(s) du langage C:

- Denis Richie
- Lionel Richie
- Bjarne Stroustrup
- Brian Kernighan

**?** Un octet peut représenter:

- des valeurs entre 0 et 255 inclus
- des valeurs entre entre 0 et 256 inclus
- des valeurs entre -128 et +127 inclus
- un idéogramme chinois quelconque

**?** Parmi ces sociétés, qui sont des éditeurs logiciels:

- Micromania
- Dell
- Adobe
- Accenture

**?** Lequel de ces acronymes ne correspond pas à un protocole réseau:

- FTP
- POP3
- HTML
- HTTP

❓ Qui n'est pas un service cloud:

- Netflix
- Amazon WS
- Microsoft Azure
- Apple icloud

❓ Quelles sont les base(s) de données:

- MySql
- Excel
- MongoDB
- NFS

❓ Qui n'est pas un comité de normalisation qui valide des standards informatiques:

- FNOR
- IETF
- W3C
- ISO

❓ Python est un langage dynamiquement typé

Côté héritage, on peut utiliser n'importe quelle classe qui possède les bonnes méthodes. On parle de:

- duck typing
- dog typing
- héritage strict
- cat typing

❓ Quels sont les algorithme(s) de chiffrement

- NSA
- RSA
- DGSE
- QUICK SORT

❓ Qu'affiche le code python suivant:

```
list = [3, 4]
list.append(12)
list.append(15)
print(list.pop())
```

- 3
- 12
- 15
- une erreur

?

Trouvez le(s) algorithme(s) de tri:

- dichotomie
- tri par fusion
- Bresenham
- tri par y-buffer

?

Que vaut 0xA0 (hexadécimal) en base 10:

- 160
- 100
- 144
- 175

?

Une adresse MAC, c'est:

- le numéro de série d'un ordinateur apple
- une adresse physique de carte réseau
- un synonyme pour une adresse IP
- une façon rapide de situer une carte mémoire dans un ordinateur

?

Qui n'est pas un Design Pattern

- Visiteur
- Memento
- Façade
- Composite

?

La table de vérité suivante correspond à:

a/b	0	1
0	0	0
1	0	1

- la fonction ou
- la fonction négation
- la fonction et
- la fonction ou exclusif

❓ Qu'affiche le code python suivant

```
def fact(n):
    return n * fact(n-1)

print(fact(5))
```

- 120
- 24
- 42
- une erreur

❓ Lesquels de ces acronymes désignent des systèmes de télécommunication:

- GSM
- ADSL
- FTTH
- RTC

❓ Lequel de ces termes ne désigne pas un système d'exploitation:

- Android
- SamsungOS
- WindowsCE
- MacOS

❓ Quelles sont les affirmation(s) vraie(s):

- La valeur 0,25 se représente exactement sur un flottant codé en base 2.
- Une recherche par dichotomie nécessite d'avoir un tableau trié.
- Un algorithme de recherche par clé demande d'avoir une opération de comparaisons entre clés.
- Il existe des problèmes mathématiques sans solution informatique.
- Un graphe sans cycles s'appelle un arbre.
- La valeur 0,1 se représente exactement sur un flottant codé en base 2.

❓ Trouvez les langage(s) de programmation:

- Java
- C<sub>b</sub>
- .NET
- TCP/IP
- Haskell



# CONCOURS D'ENTRÉE

## CYCLE INGENIEUR

**OPTION : INFORMATIQUE**

*MP / PC / PSI / PT / TSI*

Samedi 15 Avril 2017

**Durée : 2 Heures**

---

**Condition(s) particulière(s)**

Calculatrice interdite  
Remettre le QCM avec vos copies d'examen

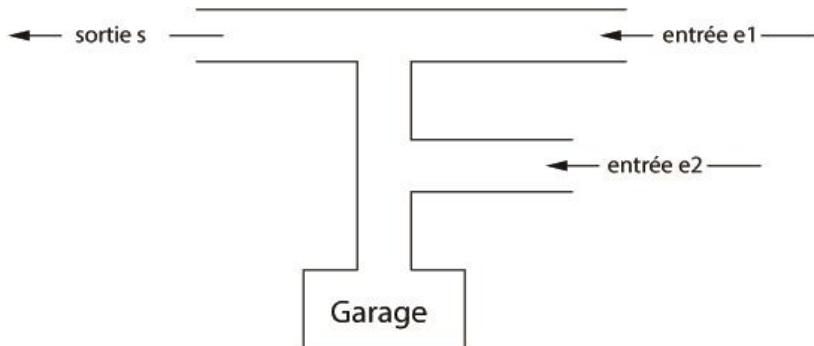
## OPTION : INFORMATIQUE

*MP / PC / PSI / PT / TSI*

Concours EPITA / IPSA 2017

### A) Piles et garages ...

Imaginons un garage possédant deux voies d'entrée et une seule de sortie (voir Figure 1). Pour garer sa voiture il existe 2 voies d'accès (*entrée e1* et *entrée e2*) et pour la ressortir une seule voie (*sortie s*).



**Figure 1 - Garage à deux entrées et une sortie**

Les mouvements possibles des voitures sont :

- une voiture peut entrer par l'*entrée e1*
- une voiture peut entrer par l'*entrée e2*
- une voiture ne peut sortir que par la *sortie s*
- une voiture ne peut ressortir que si elle est la dernière entrée (peu importe l'entrée)
- une voiture ne peut pas sauter par dessus une autre

*Remarque : Il n'y a pas de problème de voitures se retrouvant face à face.*

Le garage fonctionne comme une pile à double entrée.

Symbolisons une entrée à l'aide du couple ( $v_{\text{numéro du véhicule}}, e_{\text{numéro de l'entrée empruntée}}$ ) et la sortie simplement par la lettre *s*.

On réalise la suite d'actions suivante:

$(v_1, e_1), (v_2, e_1), (v_3, e_2), (v_4, e_1), s, s, (v_5, e_2), (v_6, e_2), s, s, s, (v_7, e_1), (v_8, e_2), (v_9, e_2), s, s, s, s$

L'ordre de sortie des voitures est donc:

$v_4, v_3, v_6, v_5, v_2, v_9, v_8, v_7, v_1$

- 1) A partir d'un garage vide, est-ce que les séquences d'entrées/sorties suivantes de véhicules sont valides ?

a)  $(v_1, e_1), (v_2, e_1), (v_3, e_1), s, s, (v_4, e_2), (v_5, e_1), s, s, s, (v_6, e_2), s$

b)  $(v_1, e_1), (v_2, e_2), s, (v_3, e_2), s, s, s, (v_4, e_1), (v_5, e_2), s, (v_6, e_1), (v_7, e_2), s, s$

Dans les cas où la séquence n'est pas valide, indiquer brièvement pourquoi.

- 2) On peut symboliser l'action "entrer une voiture" par les symboles  $E1$  ou  $E2$  (empiler) suivant l'entrée empruntée et l'action "sortir une voiture" par le symbole  $D$  (dépiler).

La suite d'actions présentée au début peut alors être symbolisée par la séquence suivante :

$E1E1E2E1DDE2E2DDDE1E2E2DDDD$

Toujours à partir d'un garage vide, donner une règle qui caractériserait les séquences admissibles.

### B) Dichotomie : « Chemin » de recherche ...

Supposons des listes d'entiers triées en ordre croissant. Si on effectuait dans celles-ci une recherche dichotomique de la valeur 66: Parmi les séquences suivantes, lesquelles ne pourraient pas correspondre à la suite des valeurs rencontrées lors de la recherche?

- a) 46 - 65 - 81 - 73 - 70 - 66
- b) 31 - 62 - 90 - 72 - 61 - 66
- c) 36 - 70 - 53 - 50 - 61 - 66
- d) 35 - 51 - 55 - 58 - 61 - 66

### C) Codage RLE simplifié ...

Le but de cet exercice est d'écrire les fonctions de compression/décompression en utilisant une version simplifiée de l'algorithme RLE (Run Length Encoding).

La compression RLE standard permet de compresser des éléments par factorisation, mais uniquement lorsque ceci permet de gagner de la place. Votre algorithme RLE simplifié effectuera cette factorisation dans tous les cas, y compris quand cela prend plus de place que l'objet non compressé.

Le flux que l'on encode est une liste d'éléments; le flux encodé est une liste de couples, chaque couple étant composé du nombre d'éléments consécutifs identiques, puis de cet élément (Voir les exemples associés aux questions suivantes).

- 1) Écrire la fonction python decodeRLE qui décomprime une liste compressée en RLE.  
*Exemples d'application:*

```
>>> decodeRLE([(6, 'grr')])  
['grr', 'grr', 'grr', 'grr', 'grr', 'grr']  
  
>>> decodeRLE([(5, 'a'), (1, 'b'), (3, 'c'), (2, 'd'), (1, 'e')])  
['a', 'a', 'a', 'a', 'a', 'b', 'c', 'c', 'c', 'd', 'd', 'e']
```

- 2) Écrire la fonction python encodeRLE ci-dessous qui compresse une liste en utilisant l'algorithme RLE.

*Exemples d'application:*

```
>>> encodeRLE(['grr', 'grr', 'grr', 'grr', 'grr', 'grr'])
[(6, 'grr')]
```

```
>>> encodeRLE(['a', 'a', 'a', 'a', 'a', 'b', 'c', 'c', 'c', 'd', 'd', 'e'])
[(5, 'a'), (1, 'b'), (3, 'c'), (2, 'd'), (1, 'e')]
```

## D) Des matrices ...

1	10	3	0	3	10	1
1	0	1	8	1	0	1
10	9	14	1	14	9	10
10	3	7	11	7	3	10
7	8	5	1	5	8	7

**Figure 2 – Mat1**

1	10	3	3	10	1
1	0	1	1	0	1
10	9	4	4	9	10
10	3	7	7	3	10
7	8	15	15	8	7

**Figure 3 – Mat2**

1	24	12	18	4
10	15	15	0	18
8	14	0	16	2
22	4	8	14	22
19	7	23	5	5

**Figure 4 – Mat3**

Pour les questions suivantes, les matrices sont supposées non vides.

- 1) Écrire la fonction python posMinimax qui cherche la valeur minimale parmi les maximums de chaque ligne d'une matrice d'entiers et retourne la position de la valeur cherchée. En cas de réponses multiples, vous conserverez la première rencontrée.

*Exemples d'applications sur les matrices des figures 2,3 et 4:*

```
>>> posMinimax(Mat1)
(1, 3)

>>> posMinimax(Mat2)
(1, 0)

>>> posMinimax(Mat3)
(2, 3)
```

- 2) Écrire la fonction python symmetric qui vérifie si une matrice est symétrique selon un axe vertical (symétrie horizontale).

*Exemples d'applications sur les matrices des figures 2, 3 et 4:*

```
>>> symmetric(Mat1)
True

>>> symmetric(Mat2)
True

>>> symmetric(Mat3)
False
```

L'usage de la calculatrice est interdit.

Les résultats des questions pourront être réutilisés ainsi que les différentes fonctions demandées non traitées au cours du problème.

Le sujet comporte deux problèmes indépendants ainsi qu'une annexe présentant entre autres un certain nombre de fonctions utiles à la rédaction des programmes en langage Python.

## Problème 1

Les systèmes de Lindenmayer, appelés aussi **L-Systèmes**, ont été imaginés par le biologiste A. Lindenmayer et modélisent le processus de développement et de prolifération de plantes. Le concept central des L-Systèmes est la représentation d'une plante par une chaîne de caractères. Cela permet de modéliser son évolution, voire sa destruction par des agents pathogènes, au moyen de règles de transformations de ces caractères.

Dans tout le problème on se place dans un plan orienté muni d'un repère (non visible). L'axe des abscisses est dirigé classiquement vers l'est (**azimut** 0 degrés) et l'axe des ordonnées vers le nord (azimut 90 degrés).

### Partie A : représentation des L-Systèmes

#### Le module turtle

Le langage Python dispose d'un module **turtle** permettant de réaliser des figures en déplaçant une « tortue » symbolisée par un triangle. Au départ, la tortue est tournée vers l'est et l'unité de longueur est le pixel. Trois instructions seront utiles ( $x$  et  $a$  pouvant être de type entier ou flottant) :

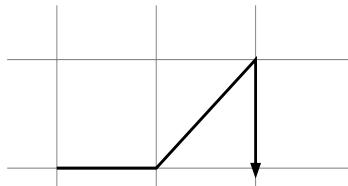
- **forward( $x$ )** : fait avancer la tortue de  $x$  pixels
- **left( $a$ )** et **right( $a$ )** : font tourner la tortue respectivement de  $a$  degrés vers sa gauche ou vers sa droite (la tortue n'avance pas).

Par exemple le code suivant :

```
from turtle import *
from math import sqrt

forward(100)
left(45)
forward(100*sqrt(2))
right(135)
forward(100)
```

trace à l'écran :



1. Écrire un programme Python permettant de réaliser la figure « maison » ci-après avec le mode `turtle`, sachant que :

$$AB = BC = CD = DE = EA = 100, (AB) \perp (AE) \text{ et } (AB) \perp (BC).$$

## Un alphabet pour coder les figures

Comme on réalise régulièrement les mêmes instructions, on se propose de **décrire** les figures selon les règles suivantes : une figure  $\mathcal{F}$  est définie par la donnée d'un triplet contenant :

- une longueur  $\ell$
- un pas de rotation  $a$ , donné en degré
- un mot, appelé **motif**, sur l'alphabet  $\{F, +, -\}$  avec comme convention :
  - $F$  : avancer de  $\ell$
  - $+$  : tourner à gauche de l'angle défini par le pas de rotation
  - $-$  : tourner à droite de l'angle défini par le pas de rotation.

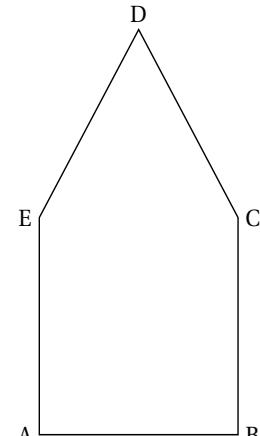


Figure « maison »

Par exemple, la figure  $\mathcal{F}(50, 90, F+F+F+F)$  représente un carré de 50 unités de côté.

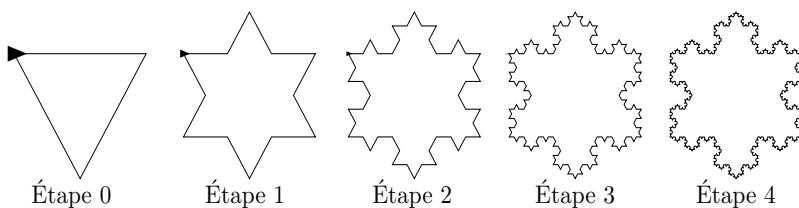
2. Décrire la figure « maison » sous la forme  $\mathcal{F}(\ell, a, m)$  en précisant les valeurs de  $\ell$ ,  $a$  et  $m$  (on commencera du point  $A$ ).

3. Écrire une fonction **dessiner** qui :

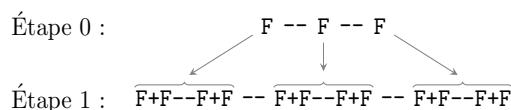
- reçoit en entrée :
  - **unite** : un nombre représentant la longueur  $\ell$
  - **angle** : un nombre représentant le pas de rotation  $a$
  - **motif** : le motif  $m$  de la figure sous forme d'une chaîne de caractères
- affiche le dessin  $\mathcal{F}(\ell, a, m)$  et retourne 0 si celui-ci a pu être réalisé intégralement et 1 sinon (par exemple si un caractère non défini est présent dans le motif).

## Les L-Systèmes

L'intérêt des L-Systèmes est de permettre de décrire simplement l'évolution d'une figure. Prenons l'exemple du flocon de Von Koch (l'échelle d'une image à l'autre a été ajustée pour une meilleure visibilité) :



En choisissant un pas de rotation de  $60^\circ$ , les motifs des deux premières figures sont :



À chaque étape, chaque lettre  $F$  dénotant un segment est remplacé par le motif  $F+F- -F+F$ . Un L-système est la donnée d'un **axiome** (motif de départ) et d'une **règle** ou d'un ensemble de règles. Dans notre exemple, l'axiome est  $F- -F- -F$  et la règle  $F \rightarrow F+F- -F+F$ .

Dans la suite, on considère un L-système avec un axiome  $F$  et une seule règle  $F \rightarrow m$  où  $m$  est une chaîne de caractères, appelée **membre droit** de la règle.

4. Écrire une fonction suivant qui :

- reçoit en entrée deux chaînes de caractères :
  - **motif** : le motif de la figure à une étape donnée
  - **regle** : le membre droit de la règle

— retourne en sortie une chaîne de caractères représentant la figure à l'étape suivante.

Par exemple :

```
>>> suivant('F--F--F', 'F+F--F+F')
'F+F--F+F--F+F--F+F--F+F'
```

5. Programmer la fonction evolution qui :

— reçoit en entrée :

- **axiome** : une chaîne de caractères représentant le motif de départ
- **regle** : une chaîne de caractères indiquant la règle de mutation
- **etape** : un entier indiquant le numéro de l'étape à calculer

— retourne en sortie une chaîne de caractères représentant la figure à l'étape demandée.

Par exemple :

```
>>> evolution('F', 'F+', 4)
'F+++'
```

6. Démontrer qu'à l'étape  $n$  ( $n \in \mathbb{N}$ ), la chaîne de caractères représentant le flocon contient  $3 \times 4^n$  caractères F et  $4^{n+1}$  caractères de rotation (+ ou -).

7. La tortue trace à la vitesse de 1000 pas par seconde et tourne à la vitesse de 800 degrés par seconde. Quel temps mettra-t-elle pour dessiner le tracé du flocon de l'étape 4 si l'on donne pour longueur d'un segment  $\ell = 2$  ?

## Nouveau mode de représentation

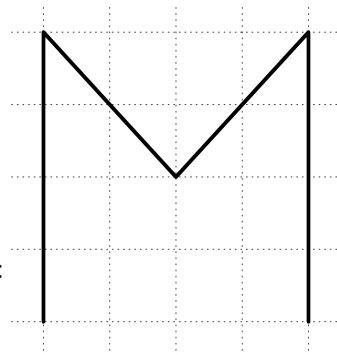
La durée de tracé étant assez longue avec le module `turtle`, on se propose d'utiliser à partir de maintenant le module `pyplot` du package `matplotlib` pour effectuer le tracé.

La fonction `plot` 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 caractère. Pour une ligne noire continue, ce style est décrit par la chaîne `k-`. Elle a pour effet de tracer la ligne brisée reliant les points de coordonnées  $(x_i, y_i)$ .

```
from matplotlib.pyplot import *
X = [0, 0, 2, 4, 4]
Y = [0, 4, 2, 4, 0]
plot(X, Y, "k-")
show()
```

Le code

affiche à l'écran :



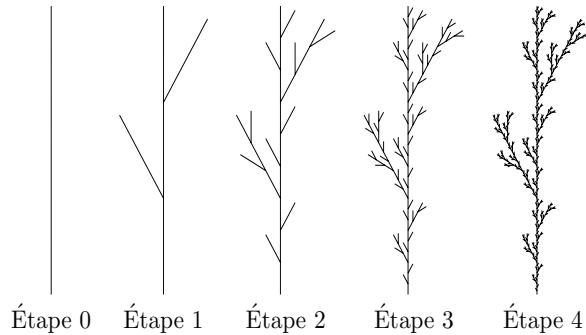
8. Écrire un programme permettant de tracer la figure « maison » à l'aide du module `pyplot`.

En se plaçant dans un repère orthonormé  $(O, \vec{i}, \vec{j})$ , on note  $(x, y)$  les coordonnées de la tortue et  $d$  son azimut en degrés. Ces 3 quantités pourront être implémentées par des flottants.

10. En considérant une longueur  $\ell$ , un pas de rotation de  $a$  degrés et un motif  $m$ , exprimer les nouvelles valeurs de  $x$ ,  $y$  et  $d$  en fonction des anciennes lorsque l'on rencontre le caractère F dans le motif. Faire de même lorsque l'on rencontre le caractère +.
11. Écrire une fonction `dessine` qui a le même effet que la fonction `dessiner` de la question 3 mais qui utilise le module `matplotlib` (et non le module `turtle`).

## Gestion des ramifications

On souhaite à présent représenter une plante à l'aide d'un L-Système. Par exemple, avec une règle préalablement choisie et partant d'une branche F, on peut obtenir les images suivantes où le pas de rotation est de  $20^\circ$  (l'échelle est ajustée d'une image à l'autre et l'azimut de départ initialisé à  $90^\circ$  pour un rendu plus réaliste) :



12. Écrire les 20 premiers caractères d'une règle de transformation de l'exemple ci-dessus.

On se propose d'ajouter deux nouveaux symboles à l'alphabet des L-Systèmes :

- [ : place l'état de la « tortue » (coordonnées et orientation) en tête d'une pile
- ] : dépile la dernière position de la « tortue » et replace la tortue à cet endroit (sans effectuer de tracé). La pile sera gérée par une liste et par les méthodes `append` et `pop`.

L'exemple pourrait alors être construit par la règle  $F \rightarrow F [+F]F[-F]F$ .

13. Quel avantage présente l'ajout de ces deux nouveaux symboles ?
14. Dessiner la figure  $\mathcal{F}(2, 90, F[-F [+F] -F]F)$  où la longueur est donnée en centimètres et le pas de rotation en degrés.
15. Réécrire la fonction `dessiner` pour qu'elle tienne compte de ces deux nouveaux symboles ; son interface devra accepter comme paramètres :
- `unité` : la longueur  $\ell$
  - `angle` : le pas de rotation  $a$  en degrés
  - `motif` : une chaîne de caractères  $m$  représentant le motif de la figure
  - `azimut` : l'azimut initial du tracé (0 par défaut afin d'assurer une cohérence avec les fonctions du mode `turtle`).

## Partie B : génération automatique de L-Systèmes

Dans cette seconde partie du problème, on cherche à générer des règles de transformation pour obtenir des L-Systèmes ayant des allures de plantes. L'axiome sera systématiquement F et la direction de départ  $90^\circ$ .

Le choix retenu ici est le recours à un algorithme (dit **génétique**) dont le principe est le suivant : partant d'une population souche de 100 individus représentant 100 règles générées de manière aléatoire, on répète les opérations suivantes (qui seront détaillées par la suite) un grand nombre de fois :

- **Sélection** : on conserve les 80 plus belles plantes obtenues,
- **Croisement** : parmi les individus restants, on en choisit 40 pour se « reproduire » deux par deux et ainsi générer 20 descendants,
- **Mutation** : certains individus subissent une légère mutation.

On ne se propose pas dans ce problème de réaliser l'intégralité de l'algorithme génétique mais on s'intéresse à certaines étapes.

## Génération de la population d'origine

On souhaite réaliser une fonction `genereRegle` qui ne reçoit aucun argument. Cette fonction renverra une chaîne composée de 15 à 30 caractères aléatoires, représentant une règle de transformation d'un L-Système.

Voici une proposition naïve de fonction :

```
from random import *

def genereRegle():
    """
    Fonction qui ne reçoit pas d'argument et retourne une règle sous
    forme d'une chaîne de caractères
    """
    alphabet = ['F', '−', '+', '[', ']']
    regle = ""
    for i in range(randint(15, 30)):
        regle = regle + choice(alphabet)
    return regle
```

- La chaîne générée ici est totalement aléatoire et peut ne pas être le motif d'une figure. Par exemple de la chaîne `] +F-F]` [--- représente une règle **invalid**e. En effet, au premier symbole `]`, la pile dont le principe a été présenté en fin de partie A est vide. Écrire une fonction `verifie` qui indique si la chaîne reçue en argument est une règle de transformation valide.
- Pour une chaîne représentant une règle valide, un certain nombre de symboles peuvent être inutiles. C'est ainsi que la chaîne : `F+-[F-F]+F[F-]-F` peut se simplifier en `F[F-F]+F[F]-F` (suppression de trois caractères inutiles car sans effet sur le dessin). Voici une proposition de fonction (comportant des erreurs) qui reçoit une règle sous forme d'une chaîne de caractères et retourne une chaîne simplifiée en se limitant à la suppression des motifs `+-`, `--`, `+`] et `-]` :

```
1 def simplifie(regle) :
2     """
3         Fonction qui reçoit une règle sous forme d'une chaîne de caractères et
4         retourne une chaîne de caractères représentant la règle simplifiée.
5     """
6     i, reponse = 0, ""
7     while i < len(regle):
8         double = regle[i] + regle[i+1]
9         if double == "++" or double == "--":
10             i = i + 1
11         elif double != "-" or double != "+]":
12             reponse = reponse + regle[i]
13             i = i + 1
14         reponse = reponse + regle[-1]
15         if len(reponse) != len(regle):
16             reponse = simplifie(reponse)
17     return reponse
```

- À quoi servent les lignes 15 et 16 de cette fonction ?
- Corriger les erreurs qui empêchent cette fonction de réaliser le travail demandé.
- Écrire une fonction `population` qui reçoit un entier `n` et renvoie une liste de `n` règles simplifiées, deux à deux distinctes et valides comportant au moins 3 symboles [ et 2 symboles de rotation.

## Mutation

La mutation d'un individu va consister ici en l'échange d'un symbole + de la règle en un symbole - ou inversement. Par exemple la règle F+F-F pourrait muter en F-F-F.

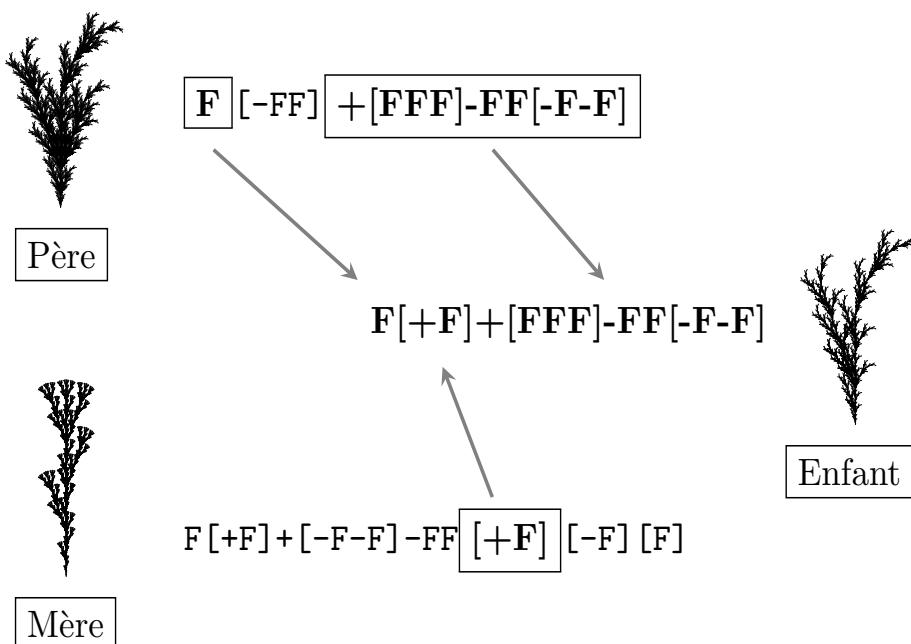
4. Réaliser une fonction `mutation` qui reçoit une chaîne de caractères représentant une règle et renvoie une nouvelle règle ayant subi une mutation quelconque : un des symboles de rotation de la chaîne, choisi aléatoirement, sera transformé en son symbole opposé.
5. La fonction `mutationPopulation` reçoit une liste de règles et un nombre  $p \in [0, 1]$  puis renvoie une nouvelle liste de règles :

```
def mutationPopulation(L, p) :
    """
    L est une liste de règles et p un nombre entre 0 et 1
    """
    for i in range(len(L)) :
        if random() < p :
            L[i] = mutation(L[i])
    return L # La liste en argument est modifiée et renournée.
```

- a. Justifier qu'à l'appel de la fonction, chaque règle de la liste subit une mutation avec une probabilité de  $p$ .
- b. Pour une population de 100 règles, combien de règles en moyenne sont modifiées (justifier la réponse).

## Croisement

On souhaite à présent réaliser le croisement de deux L-Systèmes. On appelle **branche** d'une règle tout motif situé entre deux crochets. Le principe retenu est le suivant : fabriquer une nouvelle règle en remplaçant une branche de la première règle par une branche de la seconde comme l'illustre l'exemple ci-dessous :



6. Écrire une fonction `extraitBranche` qui reçoit une règle valide donnée sous forme d'une chaîne de caractères et renvoie un triplet contenant cette chaîne coupée en trois, l'élément central étant une

branche choisie de manière aléatoire. On pourra par exemple choisir aléatoirement un symbole [ et extraire la branche associée en trouvant le caractère ] correspondant.

Par exemple :

```
>>> extractBranche('F [-FF]+[FF [-F]]-FF [-F-F]')
('F', '[-FF]', '+[FF [-F]]-FF [-F-F]')
>>> extractBranche('F [-FF]+[FF [-F]]-FF [-F-F]')
('F [-FF]+[FF', '[-F]', '] -FF [-F-F]')
```

7. Écrire alors une fonction `croise(r1,r2)` qui, recevant deux règles, renvoie (sous forme d'une chaîne de caractères), un enfant issu de ces deux règles.

## Problème 2

### Notations

- Dans ce problème,  $n$  et  $k$  désigneront des entiers naturels non nuls.
- $\llbracket 0, n \rrbracket$  désignera l'ensemble des entiers compris au sens large entre 0 et  $n$ .
- La notation  $\log_2$  désignera le logarithme de base 2, c'est-à-dire  $\forall x > 0$ ,  $\log_2(x) = \frac{\ln(x)}{\ln(2)}$  où  $\ln$  est la fonction logarithme népérien.

### Préambule

Par définition, tout ensemble fini est appelé **alphabet** et ses éléments sont appelés **lettres**. Un **mot** sur l'alphabet  $\mathcal{A}$  est une concaténation de lettres de  $\mathcal{A}$ . La concaténation des  $n$  lettres  $a_1, a_2, \dots, a_n$  est notée  $a_1a_2a_3\dots a_{n-1}a_n$ . La **longueur d'un mot** est égale au nombre de lettres composant ce mot. L'ensemble des mots de longueur  $n$  sur l'alphabet  $\mathcal{A}$  est noté  $\mathcal{A}^n$ . L'ensemble des mots sur l'alphabet  $\mathcal{A}$  est noté  $\mathcal{A}^*$ . Un mot sur l'alphabet  $\{0, 1\}$  est appelé **mot binaire**.

### Partie A

1. Énumérer les mots binaires de longueur 3.
2. La fonction `product` du module `itertools` permet de générer les éléments d'un produit cartésien. Ainsi, la commande `product(A, repeat=n)` permet d'itérer sur la liste des éléments de  $\mathcal{A}^n$ . Dans l'exemple suivant, les mots binaires de longueur 2 sont affichés. La liste `[0, 1]` correspond à l'alphabet et la longueur des mots souhaités est précisée dans `repeat`.

```
>>> from itertools import product
>>> liste = []
>>> for u in product([0,1], repeat = 2):
...     liste.append(u)
...
>>> liste
[(0, 0),(0, 1),(1, 0),(1, 1)]
```

Nous noterons  $M(n, k)$  l'ensemble des mots de longueur  $n$  sur l'alphabet  $\llbracket 0, k - 1 \rrbracket$ .

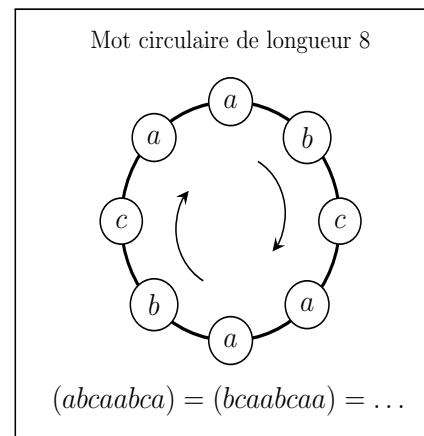
Écrire une fonction `motn` donnant les mots de  $M(n, k)$  pour  $k < 10$ . La fonction prendra comme paramètres  $n$  et  $k$  et renverra une liste contenant les mots sous forme de chaînes de caractères.

3. Combien y a-t-il d'éléments dans  $M(n, k)$  ?
4. En supposant qu'une liste puisse contenir jusqu'à 500 000 000 éléments, quelle longueur maximale peut-on donner aux mots binaires pour que la fonction `motn` s'exécute sans erreur (c'est-à-dire  $k = 2$ ) ? Vous pourrez vous aider de la courbe donnée en annexe.

## Partie B

Un **mot circulaire** est une séquence  $(a_1 a_2 \dots a_n)$  de lettres données avec un ordre circulaire, c'est-à-dire que la lettre  $a_1$  suit  $a_n$ .

Les mots  $\left\{ \begin{array}{l} a_1 a_2 \dots a_n, \\ a_2 \dots a_n a_1, \\ a_3 \dots a_1 a_2, \\ \vdots \\ a_n a_1 \dots a_{n-1} \end{array} \right.$  représentent le même mot circulaire.



La **longueur d'un mot circulaire** est égale à la longueur de n'importe lequel de ses représentants. Par exemple, les mots 101, 011 et 110 sont les représentants du même mot circulaire de longueur 3 c'est-à-dire :

$$(101) = (011) = (110).$$

1. Déterminer tous les représentants possibles du mot circulaire (1011).
2. Déterminer le nombre maximal de représentants d'un mot circulaire de  $n$  lettres. La réponse devra être justifiée.
3. Déduire de la question précédente un minorant du nombre de mots circulaires différents obtenus à partir de  $M(n, k)$ . La réponse devra être justifiée.
4. On se propose de définir une classe (au sens du langage Python) `MotCirculaire` avec :
  - un constructeur initialisant une instance à partir d'une chaîne de caractères donnant un représentant du mot circulaire,
  - deux attributs `chaine` et `longueur` stockant d'une part le représentant concaténé à lui-même et d'autre part la longueur du représentant,
  - une méthode `representant` sans paramètre et retournant le représentant initial,
  - une méthode `estEgal` permettant de déterminer si deux instances de cette classe représentent ou non le même mot circulaire. Cette méthode aura comme paramètre un mot circulaire et renverra un booléen, de sorte qu'on pourra écrire : `if mot1.estEgal(mot2):...`

Le squelette de la classe `MotCirculaire` est proposé ci-dessous. On demande de préciser le code des méthodes `representant` et `estEgal`.

```

class MotCirculaire:
    """
    La classe MotCirculaire est une implémentation naïve des mots circulaires.
    On se contente ici de listes de caractères, que l'on implémente au moyen
    de chaînes de caractères (en concaténant la chaîne avec elle-même).
    """

    def __init__(self, representant):
        """
        Initialisation de la liste circulaire à partir d'une chaîne de caractères.
        """
        self.chaine = representant * 2
        self.longueur = len(representant)

    def __len__(self):
        """
        Renvoie la longueur de la liste circulaire.
        """
        return self.longueur

    def representant(self):
        """
        Renvoie le représentant initial du mot circulaire
        """

    def estEgal(self, autreMot):
        """
        Renvoie True si autreMot est un représentant du même mot circulaire.
        """

```

On remarquera que l'attribut `chaine` permet de tester l'appartenance d'un mot en considérant la chaîne `representant` concaténée à elle-même.

5. Donner une suite d'instructions créant deux instances de la classe `MotCirculaire` à partir des chaînes 01001 et 10010 puis permettant de vérifier au moyen de la méthode `estEgal` qu'elles représentent le même mot circulaire.
  6. On dit qu'un mot circulaire  $m = (m_0 \dots m_{p-1})$  est **un mot de De Bruijn** d'ordre  $(n, k)$  lorsque chaque mot de  $M(n, k)$  apparaît exactement une fois dans  $m_0 \dots m_{p+n-2}$  (où les indices sont considérés modulo  $p$ ).
- Par exemple, (0110) est un mot de De Bruijn d'ordre  $(2, 2)$ . En effet, les mots binaires de longueur 2 : 00, 01, 10 et 11 sont présents exactement une fois dans 01100.
- Montrer que (002212011) est un mot de De Bruijn d'ordre  $(2, 3)$ .
7. Déterminer un mot de De Bruijn d'ordre  $(3, 2)$ .
  8. Montrer que la longueur de tout représentant d'un mot de De Bruijn d'ordre  $(n, k)$  est inférieure ou égale à  $n \times k^n$ .

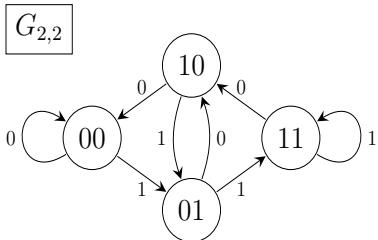
## Partie C

Nous noterons  $G_{n,k}$  le graphe orienté et étiqueté dont les sommets sont les éléments de  $M(n, k)$  et dont les arêtes sont définies comme suit : si  $u$  et  $v$  sont deux mots de  $n$  lettres sur l'alphabet  $\llbracket 0, k - 1 \rrbracket$  alors  $(u, v)$  est une arête de  $G_{n,k}$  si

$$\exists a, b \in \llbracket 0, k - 1 \rrbracket, \exists x \in M(n - 1, k) \text{ tel que } u = ax \text{ et } v = xb.$$

L'arête  $(u, v)$  aura alors l'étiquette  $b$ . Ainsi, le graphe  $G_{4,2}$  admet  $(0110, 1101)$  comme arête avec  $a = 0$ ,  $x = 110$  et  $b = 1$ .

La figure suivante donne une représentation du graphe  $G_{2,2}$  :



1. Représenter le graphe  $G_{3,2}$ .
2. La bibliothèque Python **networkx** peut être utilisée pour modéliser des graphes. Elle fournit une classe **DiGraph()** permettant de définir un graphe orienté, avec pour méthodes :
  - **add\_edge** pour ajouter une arête et ses sommets,
  - **nodes** pour retrouver les sommets,
  - **edges** pour retrouver les arêtes.

Voici un exemple en console :

```
>>> from networkx import *          # chargement du module networkx
>>> G = DiGraph()                  # définit G comme une instance de la classe DiGraph
()
>>> G.add_edge(1,2,'label='a')    # ajoute l'arête (1,2) (les sommets sont créés si
                                # besoin)
>>>                               # et donne 'a' pour étiquette à cette arête
>>> G[1][2]['label']             # retourne l'étiquette de l'arête (1,2)
'a'
>>> G.add_edge(1,'z')            # une autre arête
>>> G.nodes()                   # retourne la liste des sommets de G
[1, 2, 'z']
>>> G.edges()                   # retourne la liste des arêtes de G
[(1, 2), (1, 'z')]
```

Écrire une fonction **genGrapheDeBruijn** générant le graphe  $G_{n,k}$ . La fonction prendra en paramètres  $n$  et  $k$  et renverra une instance du graphe  $G_{n,k}$ . Notons qu'en Python, il est possible de retourner une instance  $G$  de la classe **DiGraph()** avec l'instruction **return G**.

On rappelle que dans un graphe orienté  $G$ , **un circuit eulérien** est un chemin fermé passant une fois et une seule par chaque arête de  $G$ . Un graphe orienté  $G$  possédant un circuit eulérien est appelé **graphe eulérien**.

3. Montrer que  $G_{3,2}$  est eulérien.

Dans la suite de ce problème, on admettra que  $G_{n,k}$  est **eulérien**.

4. Trouver un circuit eulérien dans  $G_{2,2}$  puis vérifier que la concaténation des étiquettes lues au fil de ce circuit donne un représentant d'un mot de De Bruijn d'ordre  $(3, 2)$ .

**On admettra que la concaténation des étiquettes lues au fil d'un circuit eulérien de  $G_{n,k}$  donne un représentant d'un mot de De Bruijn d'ordre  $(n+1, k)$  et que les mots de De Bruijn d'ordre  $(n, k)$  ont tous la même longueur.**

5. En déduire la longueur d'un mot de De Bruijn d'ordre  $(n, k)$ .

6. La fonction `eulerian_circuit` du module `networkx` permet d'obtenir les arêtes d'un circuit eulérien (lorsqu'il en existe un) à partir d'un sommet donné. Ses paramètres sont l'instance de la classe `DiGraph` dont on souhaite obtenir un circuit eulérien, ainsi que l'étiquette du sommet de départ. L'exemple suivant permet d'afficher les arcs du circuit eulérien d'un graphe ainsi que les étiquettes correspondantes.

```
>>> G = DiGraph()
>>> G.add_edge(0,1,label='a')
>>> G.add_edge(1,2,label='b')
>>> G.add_edge(2,0,label='c')
>>> liste1=[]
>>> liste2=[]
>>> for e in eulerian_circuit(G,0):
...     liste1.append(e)
...     liste2.append(G[e[0]][e[1]]['label'])
...
>>> liste1
[(0,1),(1,2),(2,0)]
>>> liste2
['a','b','c']
```

Écrire une fonction `genMotDeBruijn` qui prendra comme paramètres deux entiers  $n$  et  $k$  et renverra un représentant d'un mot de De Bruijn d'ordre  $(n, k)$  sous la forme d'une chaîne de caractères.

On suppose désormais qu'une classe `MotDeBruijn` a été écrite, ayant pour but de créer à partir de deux entiers  $n$  et  $k$  un objet représentant un mot de De Bruijn d'ordre  $(n, k)$ . Cette classe hérite de la classe `MotCirculaire` et a trois attributs qui sont `n`, `k` et `representant`. Elle contient trois méthodes sans paramètres `motn`, `genGrapheDeBruijn` et `genMotDeBruijn` adaptées des fonctions du même nom précédemment définies. L'attribut `representant` est initialisé à l'aide de la méthode `genMotDeBruijn`.

7. Écrire une fonction `estDeBruijn` permettant de déterminer si une chaîne définit un représentant pour un mot de De Bruijn d'ordre  $(n, k)$ . La fonction prendra comme paramètres une chaîne de caractères,  $n$  et  $k$ . Elle renverra un booléen (`True` si la chaîne est un représentant). Cette fonction utilisera le résultat de la question C-5 et testera la présence de toutes les sous-chaînes.

## Partie D

Dans cette partie, nous considérons une porte s'ouvrant avec un digicode à 4 chiffres. Il s'agit de trouver un nombre quelconque à 4 chiffres avec les touches 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9.

- Combien y a-t-il de combinaisons possibles ?

Il n'y a pas de validation : lorsque le code est composé, la porte s'ouvre, peu importe ce qui est tapé avant ou après. Par exemple si le code est 1256 la séquence 34125698 ouvre la porte.

- Une première méthode « naïve » est de tester tous les codes indépendamment les uns des autres. Combien de frappes de touches doit-on effectuer au maximum dans ce cas ?
- Comment utiliser les mots de De Bruijn pour ouvrir la porte plus rapidement ?
- Comparer la taille maximale des mots à écrire avec les deux méthodes.
- Sachant qu'il faut en moyenne une seconde pour appuyer sur 4 touches, combien faut-il de temps pour essayer toutes les combinaisons avec chacune des méthodes ? La réponse sera donnée en heures, minutes, secondes.
- Écrire une fonction `genCode` qui génère aléatoirement un code à 4 chiffres. La fonction ne prendra pas de paramètre et renverra une chaîne de caractères de 4 chiffres.

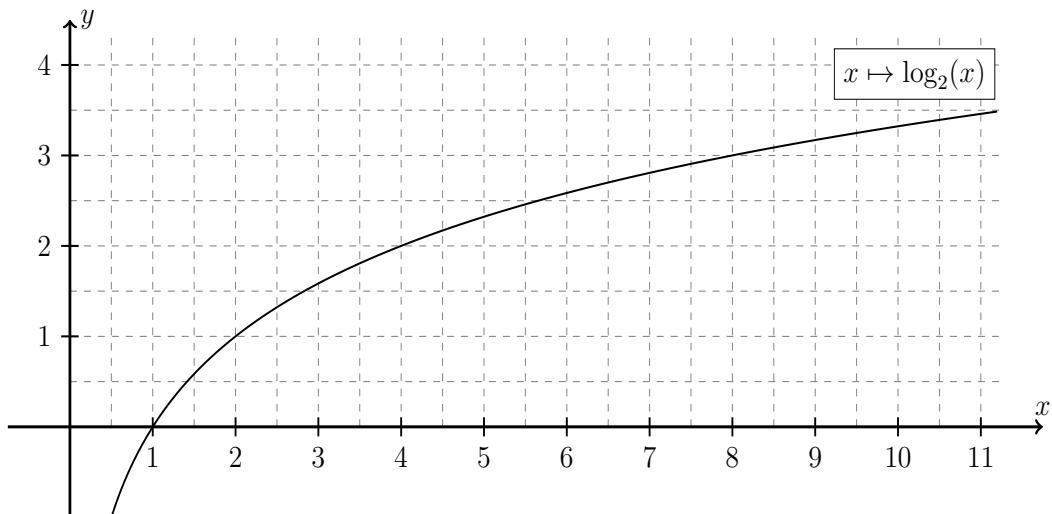
La fonction `randint` du module `random` pourra être utilisée. Sa syntaxe est décrite en annexe.

7. Écrire un programme qui :

- génère un code  $c$  aléatoirement,
- génère un mot  $m$  de De Bruijn,
- retrouve le code  $c$  généré en parcourant le mot  $m$ , puis l'affiche,
- affiche le quotient  $\frac{n_f}{n_{max}}$  où  $n_f$  représente le nombre de frappes nécessaires et  $n_{max}$  le nombre de frappes trouvé à la question 2.

# Annexe

## Courbe de la fonction logarithme de base 2



## Langage Python

### Listes

```

>>> maListe = [1,8,'e']      # définition d'un liste
>>> maListe[0]              # le premier élément d'une liste a l'indice 0
1
>>> maListe[1]
8
>>> maListe[-1]            # le dernier élément de la liste
'e'
>>> len(maListe)           # longueur d'une liste
3
>>> maListe.append(12)      # ajout d'un élément en fin de liste
>>> maListe
[1, 8, 'e', 12]
>>> maListe.remove(8)       # suppression du premier élément égale à 8
>>> maListe
[1, 'e', 12]
>>> maListe.pop(1)          # retourne l'élément d'indice 1 et le supprime de la liste
'e'
>>> maListe
[1, 12]
>>> maListe.insert(1,'a')   # insert l'élément 'a' à l'indice 1 du tableau
>>> maListe
[1, 'a', 12]

```

## Chaînes

```
>>> maChaine = 'Informatique'      # définition d'une chaîne de caractère
>>> len(maChaine)                 # longueur d'une chaîne de caractère
12
>>> maChaine[0]                  # le premier caractère de la chaîne est d'indice 0
'I'
>>> maChaine += ' et mathematiques' # concaténation de chaînes
>>> maChaine
'Informatique et mathematiques'
>>> maChaine[-1]                 # Un indice négatif permet un parcourt depuis la fin.
's'
```

## Module math

```
>>> from math import *      # chargement du module math
>>> pi                      # le nombre pi
3.141592653589793
>>> cos(pi/3)                # cosinus en radian
0.5
>>> sin(pi/2)                # sinus en radian
1.0
>>> sqrt(2)                  # racine de 2
1.4142135623730951
```

## Module random

```
>>> from random import *    # chargement du module random
>>> random()                 # renvoie aléatoirement un nombre entre 0 et 1 selon une
     loi uniforme.
0.5971130745072283
>>> randint(2,8)              # renvoie avec équiprobabilité un entier de l'intervalle [
     a,b]
3
>>> choice([1,2,4,'e',7])    # renvoie aléatoirement un élément de la liste
2
```

Cette épreuve est constituée de deux problèmes indépendants.

## Problème n° 1

### Notations.

Pour  $m$  et  $n$  deux entiers naturels,  $\llbracket m; n \rrbracket$  désigne l'ensemble des entiers  $k$  tels que  $m \leq k \leq n$ .

*La résolution d'une grille de Sudoku est une gymnastique du cerveau qui peut être assimilée à un décodage correcteur d'effacement. En effet, à partir d'une grille presque vide, il est possible (pour une grille bien faite) de la compléter d'une unique manière.*

*L'objectif de cet exercice est de mettre en œuvre deux méthodes permettant de compléter une grille de Sudoku, l'une naïve, et l'autre par backtracking.*

Une grille de Sudoku est une grille de taille  $9 \times 9$ , découpée en 9 carrés de taille  $3 \times 3$ . Le but est de la remplir avec des chiffres de  $\llbracket 1; 9 \rrbracket$ , de sorte que chaque ligne, chaque colonne et chacun des 9 carrés de taille  $3 \times 3$  contienne une et une seule fois chaque entier de  $\llbracket 1; 9 \rrbracket$ . On dira alors que la grille est complète. En pratique, certaines cases sont déjà remplies et on fera l'hypothèse que le Sudoku qui nous intéresse est bien écrit, c'est-à-dire qu'il possède une unique solution.

On représente en Python une grille de Sudoku par une liste de taille  $9 \times 9$ , c'est-à-dire une liste de 9 listes de taille 9, dans laquelle les cases non remplies sont associées au chiffre 0. Ainsi, la grille suivante est représentée par la liste ci-contre :

	6				2		5	
4		9	2	1				
	7			8			1	
				5			9	
6	4					7	3	
1		4						
3		7			6			
		1	4	6			2	
2	6				1			

```
L= [[0,6,0,0,0,0,2,0,5],[4,0,0,9,2,1,0,0,0],
[0,7,0,0,0,8,0,0,1],[0,0,0,0,0,5,0,0,9],
[6,4,0,0,0,0,7,3],[1,0,0,4,0,0,0,0,0],
[3,0,0,7,0,0,0,6,0],[0,0,0,1,4,6,0,0,2],
[2,0,6,0,0,0,0,1,0]]
```

Les 9 carrés de taille  $3 \times 3$  sont numérotés du haut à gauche jusqu'en bas à droite. Ainsi, sur cette grille, le carré 0, en haut et à gauche, contient les chiffres 6, 4 et 7 ; le carré 1, en haut et au milieu, contient les chiffres 9, 2, 1 et 8 ; le carré 8, en bas et à droite, contient les chiffres 6, 2 et 1.

On rappelle que les lignes du Sudoku sont alors les éléments de  $L$  accessibles par  $L[0], \dots, L[8]$ . L'élément de la case  $(i, j)$  est accessible par  $L[i][j]$ .

**Remarque :** on fera bien attention, dans l'ensemble de ce sujet, aux indices des tableaux. Les lignes, ainsi que les colonnes, sont indiquées de 0 à 8.

## Partie A : Généralités

### Résultats préliminaires

1. Montrer que si une grille de Sudoku est complète, alors pour chacune des lignes, chacune des colonnes et chacun des carrés de taille  $3 \times 3$ , la somme des chiffres fait 45. La réciproque est-elle vraie ?
2. Écrire une fonction `ligne_complete(L, i)` qui prend une liste Sudoku `L` et un entier `i` entre 0 et 8, et renvoie `True` si la ligne `i` du Sudoku `L` vérifie les conditions de remplissage d'un Sudoku, et `False` sinon.  
On définit de même (on ne demande pas de les écrire) les fonctions `colonne_complete(L, i)` pour la colonne `i` et `carre_complet(L, i)` pour le carré `i`.
3. Écrire une fonction `complet(L)` qui prend une liste Sudoku `L` comme argument, et qui renvoie `True` si la grille est complète, `False` sinon.

### Fonctions annexes

4. Compléter la fonction suivante `ligne(L, i)`, qui renvoie la liste des nombres compris entre 1 et 9 qui apparaissent sur la ligne d'indice `i`.

```
def ligne(L,i):
    chiffre = []
    for j in ....:
        if(...):
            chiffre.append(L[i][j])
    return chiffre
```

Ainsi, avec la grille donnée dans l'énoncé, on doit obtenir :

```
>>> ligne(L,0)
[6, 2, 5]
```

On définit alors, de la même manière, la fonction `colonne(L, j)` qui renvoie la liste des nombres compris entre 1 et 9 qui apparaissent dans la colonne `j` (on ne demande pas d'écrire son code).

5. On se donne une case  $(i, j)$ , avec  $(i, j) \in \llbracket 0; 8 \rrbracket^2$ . Montrer que la case en haut à gauche du carré  $3 \times 3$  auquel appartient la case  $(i, j)$  a pour coordonnées  $\left(3 \times \left\lfloor \frac{i}{3} \right\rfloor, 3 \times \left\lfloor \frac{j}{3} \right\rfloor\right)$ , où  $[x]$  représente la partie entière de  $x$ .
6. Compléter alors la fonction `carre(L, i, j)`, qui renvoie la liste des nombres compris entre 1 et 9 qui apparaissent dans le carré  $3 \times 3$  auquel appartient la case  $(i, j)$ .

```
def carre(L,i,j):
    icoin = 3*(i//3)
    jcoin = 3*(j//3)
    chiffre = []
    for i in range(....):
        for j in range(....):
            if(...):
                chiffre.append(L[i][j])
    return chiffre
```

On rappelle que si  $x$  et  $y$  sont des entiers,  $x//y$  renvoie le quotient de la division euclidienne de  $x$  par  $y$ . Ainsi, avec la grille donnée dans l'énoncé, on doit obtenir :

```
>>> carre(L,4,6)
[9, 7, 3]
>>>carre(L,4,5)
[5, 4]
```

7. Déduire des questions précédentes une fonction `conflit(L,i,j)` renvoyant la liste des chiffres que l'on ne peut pas écrire en case  $(i,j)$  sans contredire les règles du jeu. La liste renvoyée peut très bien comporter des redondances. On ne prendra pas en compte la valeur de  $L[i][j]$ .
8. Compléter enfin la fonction `chiffres_ok(L,i,j)` qui renvoie la liste des chiffres que l'on peut écrire en case  $(i,j)$ .

```
def chiffres_ok(L,i,j):
    ok = []
    conflit = conflit(L,i,j)
    for k in ....:
        if ....:
            ok.append(k)
    return ok
```

Par exemple, avec la grille initiale :

```
>>> chiffres_ok(L,4,2)
[2, 5, 8, 9]
```

On pourra, dans la suite du sujet, utiliser les fonctions annexes définies précédemment.

## Partie B : Algorithme naïf

Naïvement, on commence par compléter les cases n'ayant qu'une seule possibilité.  
Nous prendrons dans la suite comme Sudoku :

```
M= [[2, 0, 0, 0, 9, 0, 3, 0, 0], [0, 1, 9, 0, 8, 0, 0, 7, 4],
[0, 0, 8, 4, 0, 0, 6, 2, 0], [5, 9, 0, 6, 2, 1, 0, 0, 0],
[0, 2, 7, 0, 0, 0, 1, 6, 0], [0, 0, 0, 5, 7, 4, 0, 9, 3],
[0, 8, 5, 0, 0, 9, 7, 0, 0], [9, 3, 0, 0, 5, 0, 8, 4, 0],
[0, 0, 2, 0, 6, 0, 0, 0, 1]]
```

9. A partir des fonctions annexes, écrire une fonction `nb_possible(L,i,j)`, indiquant le nombre de chiffres possibles à la case  $(i,j)$ .
10. On souhaite disposer de la fonction `un_tour(L)` qui parcourt l'ensemble des cases du Sudoku et qui complète les cases dans le cas où il n'y a qu'un chiffre possible, et renvoie `True` s'il y a eu un changement, et `False` sinon. La liste  $L$  est alors modifiée par effet de bords.

Par exemple, en partant de la grille initiale  $M$  :

```
>>> un_tour(M)
True
>>> M
[[2, 0, 0, 0, 9, 0, 3, 0, 0], [0, 1, 9, 0, 8, 0, 5, 7, 4],
[0, 0, 8, 4, 0, 0, 6, 2, 9], [5, 9, 0, 6, 2, 1, 4, 8, 7],
[0, 2, 7, 0, 3, 8, 1, 6, 5], [0, 6, 1, 5, 7, 4, 2, 9, 3],
[0, 8, 5, 0, 0, 9, 7, 3, 0], [9, 3, 6, 0, 5, 0, 8, 4, 2],
[0, 0, 2, 0, 6, 0, 9, 5, 1]]
```

On propose la fonction suivante :

```
def un_tour(L):
    changement = False
    for i in range(1,9):
        for j in range(1,9):
            if (L[i][j] == 0):
                if (nb_possible(L,i,j) == 1):
                    L[i][j] = chiffres_ok(L,i,j)[1]
    return changement
```

Recopier ce code en en corrigeant les erreurs.

11. Écrire une fonction `complete(L)` qui exécute la fonction `un_tour` tant qu'elle modifie la liste, et renvoie `True` si la grille est complétée, et `False` sinon.

## Partie C : Backtracking

La deuxième idée est de résoudre la grille par « Backtracking » ou « retour-arrière ». L'objectif est d'essayer de compléter la grille de Sudoku en testant les combinaisons, en commençant par la première case, et jusqu'à la dernière. Si on obtient un conflit avec les règles, on est obligé de revenir en arrière.

On va compléter la grille en utilisant l'ordre lexicographique, c'est-à-dire les cases  $(0, 0), \dots, (0, 8)$  puis  $(1, 0), (1, 1), \dots, (1, 8), (2, 0), \dots$

12. Écrire une fonction `case_suivante(pos)` qui prend une liste `pos` du couple des coordonnées de la case, et renvoie la liste du couple d'indices de la case suivante en utilisant l'ordre lexicographique, et qui renvoie `[9, 0]` si `pos=[8, 8]`. Par exemple :

```
>>> case_suivante([1,3])
[1, 4]
>>> case_suivante([8,8])
[9, 0]
```

La fonction principale va avoir la structure suivante :

```
def solution_sudoku(L):
    return backtracking(L,[0,0])
```

où `backtracking(L, pos)` est une fonction récursive qui doit renvoyer `True` s'il est possible de compléter la grille à partir des hypothèses faites sur les cases qui précèdent la case `pos`, et `False` dans le cas contraire. Ainsi :

- Si `pos` est la liste `[9, 0]`, la grille est complétée, et on renvoie `True` (cas d'arrêt).
- Si la case est déjà remplie (donnée initiale du Sudoku), on passe à la case suivante via un appel récursif.
- Sinon, on affecte un des chiffres possibles à la case, et on passe à la case suivante par un appel récursif.

13. Compléter le squelette de la fonction `backtracking(L, pos)` selon les règles précédentes.

```
def backtracking(L, pos):
    """
    pos est une liste désignant une case du sudoku,
    [0,0] pour le coin en haut à gauche.
    """
    if (pos==[9, 0]):
        .....
        i,j = pos[0],pos[1]
        if L[i][j] != 0:
            return .....
        for k in ....:
            L[i][j] = .....
            if ....:
                return .....
        L[i][j] = .....
    return .....
```

14. On suppose qu'au départ, il y a  $p$  cases déjà remplies.

Montrer qu'au maximum, la fonction `backtracking` est appelée  $9^{81-p}$  fois.

15. Que renvoie la fonction `solution_sudoku(L)` si le sudoku  $L$  admet plusieurs solutions ? Et si  $L$  est le sudoku rempli de 0 ?

16. Dans l'algorithme précédent, on parcourt l'ensemble des cas dans l'ordre lexicographique. Comment améliorer celui-ci pour limiter le nombre d'appels à la variable `pos` ?

## Problème n° 2

### Notations.

Pour  $m$  et  $n$  deux entiers naturels,  $\llbracket m; n \rrbracket$  désigne l'ensemble des entiers  $k$  tels que  $m \leq k \leq n$ .

Ce problème a pour but d'étudier certains aspects de la géométrie algorithmique qui sont fortement utilisés en ingénierie et surtout en imagerie numérique. L'objectif est d'étudier deux algorithmes déterminant l'enveloppe convexe d'un ensemble de  $n$  points. Le premier, dit parcours de Jarvis, s'exécute en temps  $O(nN)$  où  $N$  est le nombre de sommets de l'enveloppe convexe. Le second, dit balayage de Graham, s'exécute en temps  $O(n \ln n)$ .

Dans tout le problème, on se place dans un plan euclidien orienté muni d'un repère orthonormé direct (non visible sur les figures de ce problème).

On pourra utiliser les fonctions de la bibliothèque `math` supposée déjà importée.

On rappelle que la complexité (temporelle) d'un algorithme est le nombre d'opérations élémentaires (affections, comparaisons, opérations arithmétiques) effectuées par un algorithme. Ce nombre s'exprime en fonction de la taille des données.

Un point  $P$  du plan muni d'un repère est représenté en Python par une liste de deux valeurs  $P=[x,y]$  où  $x$  et  $y$  sont deux nombres flottants correspondants aux coordonnées cartésiennes du point.

Un nuage de points est un ensemble  $L = \{P_0, \dots, P_{n-1}\}$  fini et non vide de points du plan. On le représente en Python par une liste  $L$  de longueur  $n$ , où pour tout entier  $i$  dans  $\llbracket 0; n-1 \rrbracket$ ,  $L[i]$  représente le point  $P_i$ .

## Partie A : Préliminaires

### Calcul de la distance entre deux points du plan

La distance euclidienne entre deux points du plan  $P$  et  $Q$  de coordonnées respectives  $(x_P, y_P)$  et  $(x_Q, y_Q)$  est donnée par

$$PQ = \sqrt{(x_P - x_Q)^2 + (y_P - y_Q)^2}.$$

1. Écrire en Python une fonction `distance` prenant en arguments deux points  $P$  et  $Q$  du plan et renvoyant la valeur de la distance euclidienne entre ces deux points.
2. Un élève a écrit une fonction qui prend en argument un nuage de points de taille supérieure à 2 et qui détermine la distance minimale entre deux points de ce nuage. Voici le code de son programme en Python :

```
def distance_minimale(L):
    n=len(L)
    minimum=distance(L[0],L[1])
    i,j=0,0
    while i < n:
        while j < n:
            a=distance(L[i],L[j])
            if a<minimum:
                minimum=a
            j+=1
        i+=1
    return minimum
```

- 2.a Combien d'appels à la fonction `distance` sont effectués par cette fonction ?
- 2.b Quelle est la valeur renvoyée par cette fonction `distance_minimale` ?
- 2.c Corriger le programme de l'élève afin que la fonction `distance_minimale` soit correcte.
3. Écrire une fonction `distance_maximale` qui prend en argument un nuage de points  $L$  et qui renvoie la distance maximale entre deux points du nuage  $L$  en effectuant exactement  $\frac{n(n-1)}{2}$  appels à la fonction `distance`. La fonction `distance_maximale` renverra également les indices d'un couple de points réalisant le maximum voulu. Par exemple :

```
>>> distance_maximale(L)
(0.9243140331952826, 1, 8)
```

Dans le nuage de points  $L$  donné en argument, la distance  $P_1P_8$  est égale à 0.9243140331952826 qui est la distance maximale obtenue.

### Recherche du point d'abscisse minimale

4. Écrire une fonction `point_abs_min` qui prend en paramètre un nuage de points  $L$  et qui renvoie l'indice du point de plus petite abscisse parmi les points du nuage de  $L$ . Si plusieurs points ont une abscisse minimale alors la fonction renverra parmi ces points, l'indice du point d'ordonnée minimale. Préciser la complexité temporelle de votre fonction lorsque le nuage est composé de  $n$  points.

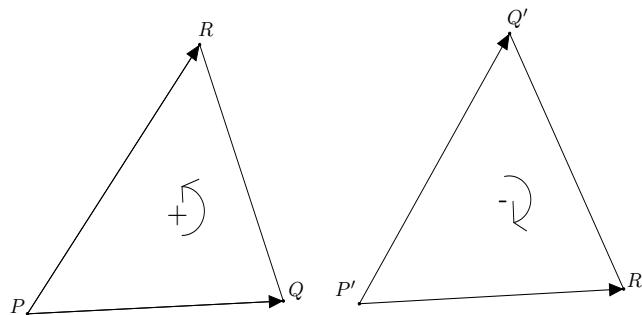
## Détermination de l'orientation de trois points du plan

**Définition.** Soient  $P$ ,  $Q$  et  $R$  trois points du plan. On considère les vecteurs  $\overrightarrow{PQ}$  et  $\overrightarrow{PR}$  de coordonnées respectives  $\begin{pmatrix} a \\ b \end{pmatrix}$  et  $\begin{pmatrix} c \\ d \end{pmatrix}$ . On note  $M$  la matrice  $\begin{pmatrix} a & c \\ b & d \end{pmatrix}$ .

On dit que l'orientation du triplet  $(P, Q, R)$

- est en sens direct si le déterminant de la matrice  $M$  est strictement positif.
- est en sens indirect si le déterminant de la matrice  $M$  est strictement négatif.
- est un alignement si est seulement si le déterminant de la matrice  $M$  est nul.

Sur la figure ci-dessous, le triplet  $(P, Q, R)$  est en sens direct et le triplet  $(P', Q', R')$  est en sens indirect :



5. On suppose qu'un triplet  $(P, Q, R)$  est en sens direct. Quelle est l'orientation des triplets  $(Q, R, P)$  et  $(P, R, Q)$ ? Justifier votre réponse.
6. Écrire une fonction `orientation` qui prend en arguments 3 points du plan  $P$ ,  $Q$  et  $R$  et qui renvoie 1 si le triplet  $(P, Q, R)$  est en sens direct, 0 si le triplet  $(P, Q, R)$  est un alignement, et -1 si le triplet  $(P, Q, R)$  est en sens indirect.

## Étude de deux algorithmes de tri

7. la fonction `tri_bulle` ci-dessous prend en argument une liste  $L$  de nombres flottants et en effectue le tri en ordre croissant :

```
def tri_bulle(L):
    n=len(L)
    for i in range(n):
        for j in range(n-1,i,-1):
            if L[j]<L[j-1]:
                L[j],L[j-1]=L[j-1],L[j]           #échange d'éléments
```

- 7.a Lors de l'appel `tri_bulle(L)` où  $L$  est la liste  $[5, 2, 3, 1, 4]$ , donner le contenu de la liste  $L$  à la fin de chaque itération de la boucle `for i in range(n):`.
- 7.b On suppose que  $L$  est une liste non vide de nombres flottants. Montrer, pour tout  $k \in \llbracket 0; n \rrbracket$ , la propriété  $\mathcal{P}_k$  :  
« après  $k$  itérations de la première boucle, les  $k$  premiers éléments de la liste sont triés par ordre croissant et sont tous inférieurs aux  $n - k$  éléments restants ».  
En déduire que `tri_bulle(L)` trie bien la liste  $L$  en ordre croissant.
- 7.c Donner la complexité dans le meilleur des cas et dans le pire des cas de la fonction `tri_bulle`.

8 Soit la fonction `tri_fusion` suivante :

```
def tri_fusion(L):
    """ Fonction qui prend en argument une liste L de nombres flottants
    et qui trie cette liste
    """
    n=len(L)
    if n<=1:
        return(L)
    else:
        m=n//2
        return (fusion(tri_fusion(L[0:m]),tri_fusion(L[m:n])))
```

8.a Écrire une fonction `fusion` qui prend en arguments deux listes triées  $L_1$  et  $L_2$  et qui renvoie une seule liste triée contenant les éléments de  $L_1$  et  $L_2$ .

La fonction `fusion` devra avoir une complexité en  $O(n_1 + n_2)$  où  $n_1$  et  $n_2$  sont les tailles respectives des listes  $L_1$  et de  $L_2$ .

Par exemple l'appel `fusion([2,4,7], [3,5,6,9])` renverra la liste `[2,3,4,5,6,7,9]`.

8.b On admet que la fonction `fusion` de la question précédente se termine. Montrer que la fonction `tri_fusion` se termine également.

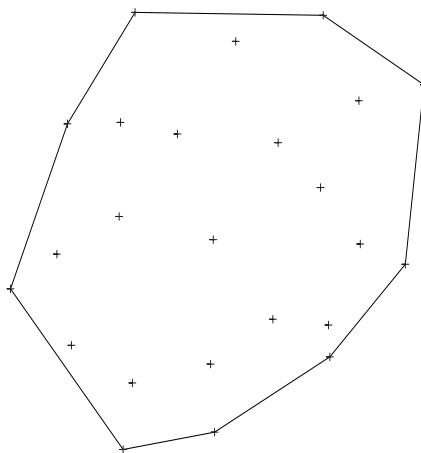
8.c On suppose que la longueur de la liste  $L$  est  $n = 2^p$ , où  $p$  est un entier naturel. Quelles sont alors les complexités dans le meilleur des cas et dans le pire des cas de `tri_fusion` ?

## Partie B : Enveloppe convexe d'un nuage de points

**Définition.** Un ensemble  $S$  est **convexe** si pour tout couple de points  $(P, Q)$  de  $S$ , le segment  $[PQ]$  est contenu dans  $S$ . L'**enveloppe convexe** d'un ensemble  $S$  est le plus petit ensemble convexe contenant  $S$ .

Dans cette partie, l'objet est d'étudier deux algorithmes permettant d'obtenir l'enveloppe convexe d'un ensemble fini de points du plan.

**Théorème** (admis). Soit  $S$  un ensemble de  $n$  points du plan, avec  $n > 1$ . l'enveloppe convexe de  $S$  est constituée par un polygone  $P$ , sous-ensemble de  $S$ , des segments unissant les points successifs de  $P$ , et de tous les segments unissant deux points des segments précédents, comme l'illustre la figure ci-dessous.



Dans la suite du problème, on supposera que les nuages de points considérés ne contiennent pas 3 points distincts alignés. Cette hypothèse permettra de simplifier les algorithmes.

## L'algorithme de Jarvis

La « marche de Jarvis » est un des algorithmes les plus naturels. Conçu en 1973, il consiste à reprendre l'image de l'emballage d'un cadeau.

La première idée de Jarvis pour déterminer l'enveloppe convexe est de chercher tout d'abord les segments qui forment ses côtés à l'aide de la propriété admise suivante :

**Propriété** : soit  $L$  un nuage de points. Pour tous points distincts  $P$  et  $Q$  de  $L$ , le segment  $[PQ]$  est un côté de l'enveloppe convexe de  $L$  si les triplets  $(P, Q, R)$ , où  $R$  est un point de  $L$  distinct de  $P$  et  $Q$ , ont tous la même orientation.

9. Voici une fonction « naïve » qui permet de déterminer les sommets de l'enveloppe convexe d'un nuage de points :

```

1 def jarvis(L):
2     """
3         Fonction qui reçoit en argument un nuage de points et qui renvoie
4         une liste contenant les indices des sommets de l'enveloppe
5         convexe de ce nuage
6     """
7     n=len(L)
8     EnvConvexe=[]
9     for i in range(n):
10         for j in range(n):
11             Listeorientation=[]
12             if i!=j:
13                 for k in range(n):
14                     if k!=i and k!=j:
15                         Listeorientation.append(orientation(L[i],L[j],L[k]))
16                     a=Listeorientation[0]
17                     sommet=True
18                     for v in Listeorientation:
19                         if (v!=a):
20                             sommet=False
21                     if sommet and (i not in EnvConvexe) :
22                         EnvConvexe.append(i)
23                     if sommet and (j not in EnvConvexe):
24                         EnvConvexe.append(j)
25     return EnvConvexe

```

9.a Expliquer à quoi servent les lignes 16 à 20 de cette fonction.

9.b Si on considère un nuage de  $n$  points avec  $n \geq 3$ , quelle est la complexité temporelle de cette fonction ? Justifier.

9.c Le script suivant trace-t-il le polygone formé par les sommets de l'enveloppe convexe du nuage  $L$ ? Justifier.

```

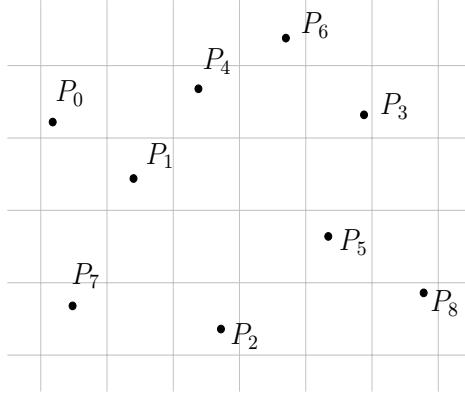
1 >>> import matplotlib.pyplot as plt
2 >>> Enveloppe=jarvis(L)
3 >>> plt.plot([L[i][0]  for i in Enveloppe],[L[i][1]  for i in Enveloppe])
4 >>> plt.show()

```

Jarvis proposa ensuite une version plus efficace de son algorithme en voulant « ranger » les points du nuage autour d'un point de l'enveloppe. Il utilise une relation d'ordre sur les points du nuage.

**Définition.** Soit  $P$  un sommet de l'enveloppe convexe du nuage. Le point suivant sur l'enveloppe convexe est le plus petit pour la relation d'ordre  $\preccurlyeq_P$  définie sur l'ensemble des sommets du nuage différents de  $P$  par :

$$Q \preccurlyeq_P R \Leftrightarrow \text{le triplet } (P, Q, R) \text{ est en sens direct ou } Q = R.$$



Par exemple, dans le nuage points ci-dessus, on obtient le classement des points suivants pour la relation d'ordre  $\preccurlyeq_{P_2}$  :

$$P_8 \preccurlyeq_{P_2} P_5 \preccurlyeq_{P_2} P_3 \preccurlyeq_{P_2} P_6 \preccurlyeq_{P_2} P_4 \preccurlyeq_{P_2} P_1 \preccurlyeq_{P_2} P_0 \preccurlyeq_{P_2} P_7.$$

Le plus petit point pour la relation  $\preccurlyeq_{P_2}$  est le point  $P_8$ .

**10.** Donner le classement des points du nuage de la figure ci-dessus pour la relation d'ordre  $\preccurlyeq_{P_6}$ .

Ainsi si on connaît un sommet  $P$  de l'enveloppe convexe d'un nuage de points alors le prochain point de l'enveloppe convexe à déterminer est celui qui est minimal pour la relation  $\preccurlyeq_P$ .

**11.** Écrire une fonction `prochain_sommet` qui prend en arguments un nuage de points  $L$  et l'indice d'un sommet  $P$  de ce nuage de points qui est un sommet de l'enveloppe convexe et qui renvoie l'indice du prochain sommet de l'enveloppe convexe. Cette fonction devra avoir une complexité en  $O(n)$  où  $n$  est le nombre de points du nuage.

Comme le point du nuage d'abscisse minimale (et d'ordonnée minimale s'il y a plusieurs points d'abscisse minimale) fait partie de l'enveloppe convexe on peut construire un algorithme qui détermine au fur et à mesure tous les sommets de l'enveloppe convexe. On arrête l'algorithme quand la fonction `prochain_sommet` renvoie l'indice du sommet de départ.

**12.** Recopier et compléter la fonction suivante afin qu'elle renvoie l'enveloppe convexe d'un nuage de points  $L$ .

```
def jarvis2(L):
    i=point_abs_min(L)
    suivant=prochain_sommet(L,i)
    Enveloppe=[i,suivant] #initialisation de la liste des indices des sommets de
    l'enveloppe convexe
    while (.....) :
        .....
        .....
    return Enveloppe
```

**13.** Soit  $L$  un nuage de  $n$  points dont on sait que l'enveloppe convexe est un polygone à  $N$  sommets. Montrer que l'algorithme décrit par la fonction `jarvis2` possède une complexité en  $O(n \times N)$ .

## L'algorithme de Graham - Andrew

En 1972, Graham et Andrew proposèrent une méthode pour déterminer l'enveloppe convexe d'un nuage de points. Leur algorithme est basé sur une méthode de tri des points.

La première étape de l'algorithme de Graham et Andrew est de trier les  $n$  points du nuage  $L$  par ordre croissant d'abscisses (si deux points ont la même abscisse on classera ces points suivant leurs ordonnées). On supposera donnée une fonction `tri_nuage` prenant en entrée un nuage de points  $L$  et réalisant cette opération en complexité  $O(n \log n)$  où  $n$  est le nombre de points du nuage. ( $n \geq 3$ ).

L'idée de l'algorithme est de balayer le nuage de points dans l'ordre croissant de leurs abscisses tout en mettant à jour l'enveloppe convexe des points. L'enveloppe convexe a été scindée en deux listes `EnvSup` et `EnvInf`.

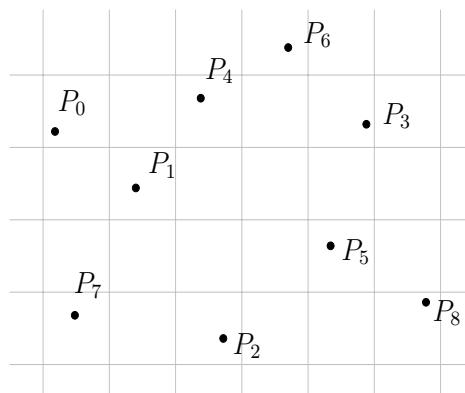
Dans ces listes `EnvSup` et `EnvInf`, on maintient l'enveloppe convexe des points déjà traités. Chaque nouvel indice du point  $P$  du nuage est ajouté à `EnvSup` et `EnvInf`, puis tant que l'avant dernier-point de `EnvSup` rend la séquence non-convexe, il est enlevé (de même pour `EnvInf`).

**14.** Voici en Python la fonction qui permet d'obtenir l'enveloppe convexe :

```
def graham_andrew(L):
    L=tri_nuage(L)
    EnvSup=[]
    EnvInf=[]
    for i in range(len(L)):
        while len(EnvSup)>=2 and orientation(L[i],L[EnvSup[-1]],L[EnvSup[-2]])<=0:
            EnvSup.pop()
            EnvSup.append(i)
        while len(EnvInf)>=2 and orientation(L[EnvInf[-2]],L[EnvInf[-1]],L[i])<=0:
            EnvInf.pop()
            EnvInf.append(i)
    return EnvInf[:-1]+EnvSup[::-1]
```

**14.a** A partir du nuage de points représentés ci-dessous, donner le contenu de la liste `EnvSup` à chaque itération de la boucle `for`.

Donner également le résultat de `orientation(P,EnvSup[-1],EnvSup[-2])<=0`.



**14.b** Montrer la terminaison de la fonction `graham_andrew`.

**14.c** Montrer que la complexité de `graham_andrew` est en  $O(n \log n)$  où  $n$  est la taille du nuage de points.

# Annexe

## Langage Python

### Listes

```

>>> maListe = [1,8,'e']      # définition d'un liste
>>> maListe[0]              # le premier élément d'une liste a l'indice 0
1
>>> maListe[1]
8
>>> maListe[-1]            # le dernier élément de la liste
'e'
>>> maListe[-2]            # l'avant dernier élément de la liste
8
>>> len(maListe)           # longueur d'une liste
3
>>> maListe.append(12)      # ajout d'un élément en fin de liste
>>> maListe
[1, 8, 'e', 12]
>>> maListe.remove(8)       # suppression du premier élément égale à 8
>>> maListe
[1, 'e', 12]
>>> maListe.pop()          # retourne le dernier élément et le supprime de la liste
'e'
>>> maListe
[1, 12]
>>> maListe.insert(1,'a')   # insert l'élément 'a' à l'indice 1 du tableau
>>> maListe
[1, 'a', 12]
>>>range(len(maListe)) :   # parcours des indices de la liste
>>> range(8)               # parcours des indices entiers de 0 à 8 exclus
0 1 2 3 4 5 6 7
>>>range(3,8)              # parcours des indices entiers de 3 inclus à 8 exclus
3 4 5 6 7
>>> range(3,8,2)           # parcours des indices entiers de 3 inclus
3 5 7                   # à 8 exclus avec un pas de 2
>>>A= [1,'e',8]+[2,'tu']  # concaténation de deux listes
[1,'e',8,2,'tu']
>>>A[1:4:2]                # renvoie la liste des éléments de A d'indice 1 inclus
['e',2]
>>>A[:-1]                  # renvoie la liste des éléments de A à l'exclusion
[1,'e',8,2]
>>>A[::-1]                  # renvoie la liste des éléments de A, du dernier au premier
['tu',2,8,'e',1]

```

Complexité de certaines procédures en langage Python

<code>Liste.pop()</code>	$O(1)$
<code>Liste.append(element)</code>	$O(1)$
<code>element in Liste</code>	$O(\text{len(Liste)})$
<code>element not in Liste</code>	$O(\text{len(Liste)})$

## Module math

```
>>> from math import *    # chargement du module math
>>> sqrt(2)                # racine de 2
1.4142135623730951
```

**Sélection internationale  
École Normale Supérieure  
Épreuve de culture scientifique - Informatique**

Session 2016

Paris

Durée : 3 hours

**Pour les candidats ayant choisi l'informatique comme discipline principale**

Si vous ne parvenez pas à répondre à une question, vous pouvez cependant l'utiliser comme hypothèse pour les questions suivantes.

Calculatrices interdites.

**Exercice 1.** Supposons donné un tableau  $A[0 \dots n + 1]$  avec les valeurs extrêmes  $A[0] = A[n + 1] = -\infty$ . Un élément  $A[x]$  est un *maximum local* si il est supérieur ou égal à ses voisins, ou de façon plus formelle si  $A[x - 1] \leq A[x]$  et  $A[x] \geq A[x + 1]$ .

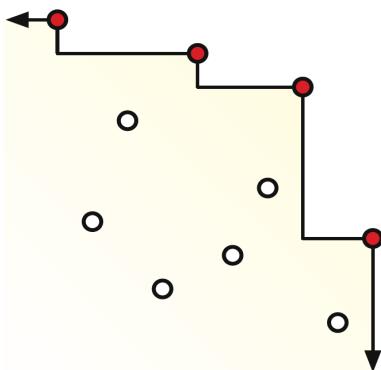
Nous pouvons de façon évidente trouver un maximum local en temps  $O(n)$  en parcourant le tableau. Décrire et analyser un algorithme qui retourne l'indice d'un maximum local en temps  $O(\log n)$ .

**Indication:** Avec les conditions aux bords, le tableau doit contenir un maximum local. Pourquoi ?

**Exercice 2.** Nous allons partitionner l'ensemble des sommets d'un graphe  $G$  en deux ensembles  $S$  et  $T$ . L'algorithme tire un bit uniformément au hasard pour chaque sommet et si le bit vaut 0, l'algorithme place le sommet dans  $S$  et si le bit vaut 1, il le place dans  $T$ .

1. Montrer que le nombre espéré d'arêtes du graphe avec une extrémité dans  $S$  et une extrémité dans  $T$  est égal à la moitié du nombre d'arêtes de  $G$ .
2. Supposons désormais que les arêtes sont pondérées. Que peut-on dire sur la somme des poids des arêtes du graphe avec une extrémité dans  $S$  et une extrémité dans  $T$  ?

**Exercice 3.** Supposons donné un ensemble  $P$  de  $n$  points dans le plan. Un point  $p \in P$  est *maximal* dans  $P$  si aucun autre point de  $P$  n'est à la fois au-dessus et à droite de  $p$ . Intuitivement, les points maximaux définissent un “escalier” avec tous les autres points de  $P$  en dessous.



Décrire et analyser un algorithme qui calcule le nombre de points maximaux de  $P$  en temps  $O(n \log n)$ . Étant donné les dix points de l'exemple ci-dessus, l'algorithme doit retourner l'entier 4.

**Exercice 4.**

1. Un *tableau extensible* est une structure de données qui stocke une suite d'éléments et permet les opérations suivantes:

- $\text{ADDTOFRONT}(x)$  ajoute  $x$  au début de la suite.
- $\text{ADDTOEND}(x)$  ajoute  $x$  à la fin de la suite.
- $\text{LOOKUP}(k)$  retourne le  $k$ -ième élément de la suite ou  $\text{NULL}$  si la longueur de la suite ne dépasse pas  $k$ .

Décrire une structure de données simple qui implante un tableau extensible. Les algorithmes  $\text{ADDTOFRONT}$  et  $\text{ADDTOEND}$  doivent s'exécuter en temps amorti  $O(1)$ . L'algorithme  $\text{LOOKUP}$  doit s'exécuter en temps  $O(1)$  dans le pire des cas. La structure de données doit utiliser un espace  $O(n)$ , où  $n$  est la longueur courante de la suite.

2. Une pile est une structure de données dernier-entré premier-sorti. Elle permet les opérations  $\text{PUSH}$  et  $\text{POP}$ .  $\text{PUSH}$  ajoute un élément au sommet de la pile et  $\text{POP}$  retire et retourne l'élément du sommet de la pile (l'élément ajouté le plus récemment). Une *pile ordonnée* est une structure de donnée qui stocke une suite d'éléments et permet les opérations suivantes:

- $\text{ORDEREDPUSH}(x)$  retire tous les éléments inférieurs à  $x$  à partir du sommet de la pile et ajoute  $x$  au sommet de la pile.
- $\text{POP}$  retire et retourne l'élément du sommet de la pile (ou  $\text{NULL}$  si la pile est vide).

Supposons que nous implantons une pile ordonnée avec une liste simplement chaînée en utilisant les algorithmes  $\text{ORDEREDPUSH}$  et  $\text{POP}$  évidents. Montrer que si l'on démarre d'une structure de donnée vide, le coût amorti des opérations  $\text{ORDEREDPUSH}$  et  $\text{POP}$  est  $O(1)$ .

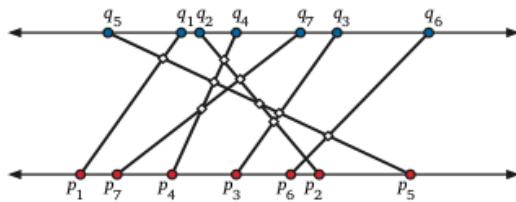
3. Une file est une structure de données dernier-entré dernier-sorti. Elle permet les opérations  $\text{PUSH}$  et  $\text{POP}$ .  $\text{PUSH}$  ajoute un élément au début de la file et  $\text{POP}$  retire et retourne l'élément à la fin de la file. Montrer comment simuler une file avec deux piles. Une séquence de  $\text{PUSH}$  et  $\text{POP}$  devra s'exécuter en temps amorti constant.

**Exercice 5.**

1. Une inversion dans un tableau  $A[1 \dots n]$  est un couple d'indice  $(i, j)$  tel que  $i < j$  et  $A[i] > A[j]$ . Le nombre d'inversions d'un tableau de longueur  $n$  est entre 0 (si le tableau est trié) et  $\binom{n}{2}$  (si le tableau est trié dans le sens inverse). Décrire et analyser un algorithme de type “diviser-pour-régner” qui retourne le nombre d'inversions dans un tableau de longueur  $n$  en temps  $O(n \log n)$ . Nous supposons que tous les éléments du tableau d'entrée sont différents.
2. Étant donnés deux ensembles de  $n$  points, un ensemble  $\{p_1, p_2, \dots, p_n\}$  sur la droite  $y = 0$  et un autre ensemble  $\{q_1, q_2, \dots, q_n\}$  sur la droite  $y = 1$ , nous associons un ensemble de  $n$  segments de droites en reliant chaque point  $p_i$  au point correspondant  $q_i$ . Décrire et analyser un algorithme de type “diviser-pour-régner” qui retourne, en temps  $O(n \log n)$ , le nombre de couples de segments qui s'intersectent.

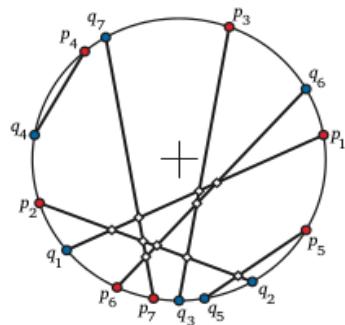
**Indication:** Utiliser la solution de la question 1.

Nous supposons une représentation “raisonnable” des points de l'entrée et nous supposons que les coordonnées  $x$  de ces points sont deux à deux distinctes. Par exemple, sur l'entrée ci-dessous, l'algorithme doit retourner le nombre 9.



3. Étant donnés deux ensembles de  $n$  points  $\{p_1, p_2, \dots, p_n\}$  et  $\{q_1, q_2, \dots, q_n\}$  sur le cercle unité, nous associons un ensemble de  $n$  segments de droites en reliant chaque point  $p_i$  au point correspondant  $q_i$ . Décrire et analyser un algorithme de type “diviser-pour-régner” qui retourne, en temps  $O(n \log^2 n)$ , le nombre de couples de segments qui s’intersectent.
- Indication:** Utiliser la solution de la question 2.

Nous supposons une représentation “raisonnable” des points de l’entrée et nous supposons que les points sont deux à deux distincts. Par exemple, sur l’entrée ci-dessous, l’algorithme doit retourner le nombre 10.



4. (\*) Proposer un algorithme améliorer en temps  $O(n \log n)$ .

**Sélection internationale**  
**École Normale Supérieure**  
**Épreuve de culture scientifique - Informatique**

Session 2016  
 Paris  
 Durée : 2 hours

Pour les candidats qui ont choisi **Informatique** comme **discipline secondaire**.

Si vous ne pouvez pas répondre à une question, vous pouvez l'utilisez comme hypothèse dans les questions suivantes.

Calculatrices non autorisées.

**Exercice 1.**

Une sous-séquence est obtenue à partir d'une séquence en supprimant un sous-ensemble de ces éléments ; les éléments de la sous-séquence ne doivent pas être nécessairement être consécutifs dans la séquence originale. Par exemple, les chaînes L, ELECTION, EEIOI, NTRNATNL et SELECTIONINTERNATIONALE sont des sous-séquences de la chaîne SELECTIONINTERNATIONALE.

Dans toutes les questions suivantes, décrivez tout d'abord un algorithme récursif et transformez le seulement ensuite en un algorithme itératif.

1. Supposons donnés deux tableaux  $A[1 \dots \ell]$  et  $B[1 \dots m]$ . Une sous-séquence commune de  $A$  et  $B$  est une sous-séquence de  $A$  qui est également une sous-séquence de  $B$ . Par exemple, ETION est une sous-séquence commune de SELECTION et INTERNATIONALE. Décrire et analyser un algorithme efficace qui calcule la longueur de la plus longue sous-séquence commune de deux tableaux  $A[1 \dots \ell]$  et  $B[1 \dots m]$ .
2. Décrire et analyser un algorithme efficace qui calcule la longueur de la plus longue sous-séquence commune de trois tableaux  $A[1 \dots \ell]$ ,  $B[1 \dots m]$  et  $C[1 \dots n]$ .
3. Un mélange de deux chaînes  $X$  et  $Y$  est formé par la dispersion des caractères dans une nouvelle chaîne de sorte que les caractères de  $X$  et  $Y$  restent dans le même ordre. Par exemple, étant données les chaînes SELECTION et INTERNATIONALE, la chaîne SELINETRNCTAITIONONALE est un mélange valide:

SEL<sup>IN</sup>E<sup>TERN</sup>CT<sup>A</sup>I<sup>TION</sup>ON<sup>AL</sup>E

Étant données trois chaînes  $A[1 \dots \ell]$ ,  $B[1 \dots m]$  et  $C[1 \dots \ell + m]$ , décrire un algorithme qui détermine si  $C$  est un mélange de  $A$  et  $B$ .

**Exercice 2.** Considérons l'algorithme suivant où  $\text{Random}(1, n)$  retourne un entier tiré uniformément au hasard entre 1 et  $n$  (inclus) en temps constant.

```

SHUFFLE( $A[1 \dots n]$ )
for  $i \leftarrow 1$  to  $n$ 
     $B[i] \leftarrow \text{null}$ 
for  $i \leftarrow 1$  to  $n$ 
     $j \leftarrow \text{Random}(1, n)$ 
    while  $B[j] \neq \text{null}$ 
         $j \leftarrow \text{Random}(1, n)$ 
     $B[j] \leftarrow A[i]$ 
for  $i \leftarrow 1$  to  $n$ 
     $A[i] \leftarrow B[i]$ 

```

1. Montrer que SHUFFLE permute le tableau en entrée dans un ordre aléatoire où chaque permutation est équiprobable.

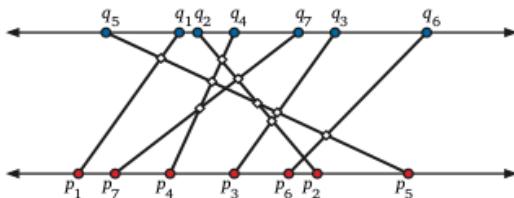
2. Quel est le temps d'exécution moyen de cet algorithme ? Justifier votre réponse et donner une borne asymptotique fine.
3. Décrire un algorithme qui prend en entrée un tableau de longueur  $n$  et retourne le tableau permuté dans un ordre aléatoire (de sorte que chaque permutation soit équiprobable) en temps  $O(n)$ .

### Exercice 3.

1. Une inversion dans un tableau  $A[1 \dots n]$  est un couple d'indice  $(i, j)$  tel que  $i < j$  et  $A[i] > A[j]$ . Le nombre d'inversions d'un tableau de longeur  $n$  est entre 0 (si le tableau est trié) et  $\binom{n}{2}$  (si le tableau est trié dans le sens inverse). Décrire et analyser un algorithme de type "diviser-pour-régner" qui retourne le nombre d'inversions dans un tableau de longueur  $n$  en temps  $O(n \log n)$ . Nous supposons que tous les éléments du tableau d'entrée sont différents.
2. Étant donnés deux ensembles de  $n$  points, un ensemble  $\{p_1, p_2, \dots, p_n\}$  sur la droite  $y = 0$  et un autre ensemble  $\{q_1, q_2, \dots, q_n\}$  sur la droite  $y = 1$ , nous associons un ensemble de  $n$  segments de droites en reliant chaque point  $p_i$  au point correspondant  $q_i$ . Décrire et analyser un algorithme de type "diviser-pour-régner" qui retourne, en temps  $O(n \log n)$ , le nombre de couples de segments qui s'intersectent.

**Indication:** Utiliser la solution de la question 1.

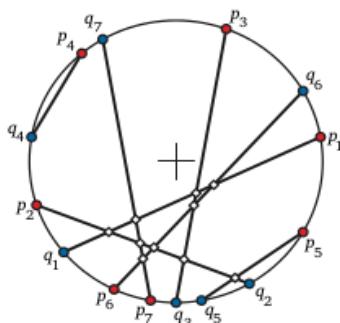
Nous supposons une représentation "raisonnable" des points de l'entrée et nous supposons que les coordonnées  $x$  de ces points sont deux à deux distinctes. Par exemple, sur l'entrée ci-dessous, l'algorithme doit retourner le nombre 9.



3. Étant donnés deux ensembles de  $n$  points  $\{p_1, p_2, \dots, p_n\}$  et  $\{q_1, q_2, \dots, q_n\}$  sur le cercle unité, nous associons un ensemble de  $n$  segments de droites en reliant chaque point  $p_i$  au point correspondant  $q_i$ . Décrire et analyser un algorithme de type "diviser-pour-régner" qui retourne, en temps  $O(n \log^2 n)$ , le nombre de couples de segments qui s'intersectent.

**Indication:** Utiliser la solution de la question 2.

Nous supposons une représentation "raisonnable" des points de l'entrée et nous supposons que les points sont deux à deux distincts. Par exemple, sur l'entrée ci-dessous, l'algorithme doit retourner le nombre 10.



4. (\*) Proposer un algorithme améliorer en temps  $O(n \log n)$ .

**ÉCOLE POLYTECHNIQUE — ÉCOLES NORMALES SUPÉRIEURES  
ÉCOLE SUPÉRIEURE DE PHYSIQUE ET DE CHIMIE INDUSTRIELLES**

CONCOURS D'ADMISSION 2017

FILIÈRES **MP** HORS SPECIALITÉ INFO,  
**PC** ET **PSI**

**COMPOSITION D'INFORMATIQUE – B – (XELCR)**

(Durée : 2 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.  
Le langage de programmation sera **obligatoirement PYTHON**.

\*  
\* \*

## Intersection de deux ensembles de points

Soit  $n$  un entier naturel. On note  $D_n$  l'ensemble des entiers naturels compris entre 0 et  $2^n - 1$ . On appelle « point de  $D_n \times D_n$  » tout couple d'entiers  $(x, y) \in D_n \times D_n$ . Soient  $P$  et  $Q$  deux parties de  $D_n \times D_n$ . On cherche à calculer efficacement l'intersection des ensembles de points  $P$  et  $Q$ . La résolution de ce problème a des applications en simulation numérique, en robotique ou encore dans l'implémentation d'interfaces utilisateurs.

Ce sujet est découpé en cinq parties. La partie I porte sur une solution naïve en PYTHON, la partie II sur une solution naïve en SQL. Les parties III, IV et V conduisent à la réalisation d'une solution efficace en PYTHON. La partie I et la partie II sont totalement indépendantes l'une de l'autre et du reste du sujet. Les parties suivantes ne sont pas indépendantes.

### Remarques préliminaires

**Complexité.** La complexité, ou le temps d'exécution, d'un programme  $P$  (fonction ou procédure) est le nombre d'opérations élémentaires (addition, multiplication, affectation, test, etc.) nécessaires à l'exécution de  $P$ . Lorsque cette complexité dépend de plusieurs paramètres  $n$  et  $m$ , on dira que  $P$  a une complexité en  $O(\phi(n, m))$ , lorsqu'il existe trois constantes  $A$ ,  $n_0$  et  $m_0$  telles que la complexité de  $P$  soit inférieure ou égale à  $A \times \phi(n, m)$ , pour tout  $n > n_0$  et  $m > m_0$ . Lorsqu'il est demandé de garantir une certaine complexité, le candidat devra justifier cette dernière en raisonnant sur la structure du code.

**Rappels de PYTHON.** Si `a` est une liste alors `a[i]` désigne le  $i$ -ième élément de cette liste où l'entier  $i$  est supérieur ou égal à 0 et strictement plus petit que la longueur `len(a)` de la liste. La commande `a[i] = e` affecte la valeur de l'expression `e` au  $i$ -ième élément de la liste `a`. L'expression `[]` construit une liste vide. L'expression `n * [k]` construit une liste de longueur `n` contenant `n` occurrences de `k`. La commande `a = list(b)` construit une copie de la liste `b` et l'affecte à la variable `a`. La commande `a.append(x)` modifie la liste `a` en lui rajoutant un nouvel élément final contenant la valeur de `x`. **Important :** Seules les opérations sur les listes apparaissant dans ce paragraphe sont autorisées dans les réponses. Si une fonction PYTHON standard est nécessaire, elle devra être réécrite.

*Nous attacherons la plus grande importance à la lisibilité du code produit par les candidats ; aussi, nous encourageons les candidats à utiliser des commentaires et à introduire des procédures ou des fonctions intermédiaires pour faciliter la compréhension du code.*

## Partie I. Une solution naïve en PYTHON

Pour commencer, un point de coordonnées  $(x, y) \in D_n \times D_n$  est représenté en PYTHON par une liste de deux entiers naturels `[x, y]`. Un ensemble de points est représenté par une liste de points sans répétition, donc comme une liste de listes d'entiers naturels de longueur 2.

**Question 1.** Écrire une fonction `membre(p, q)` qui renvoie `True` si le point `p` est dans l'ensemble représenté par la liste `q` et qui renvoie `False` dans le cas contraire.

**Question 2.** Écrire une fonction `intersection(p, q)` qui renvoie une liste représentant l'intersection des ensembles représentés par `p` et `q`. On implémentera l'algorithme qui consiste à itérer sur tous les points de `p` et à insérer dans le résultat seulement ceux qui sont aussi dans `q`.

**Question 3.** Si la comparaison entre entiers naturels est prise comme opération élémentaire, quelle est la complexité de l'algorithme de la question précédente exprimée en fonction de la longueur de `p` et `q` ?

## Partie II. Une solution naïve en SQL

On suppose maintenant que l'on représente les points du problème à l'aide d'une base de données. Cette base de données comporte deux tables. La table `POINTS` contient trois colonnes :

- `id` (clé primaire) qui est un entier naturel unique représentant le point ;
- `x` qui est un entier naturel représentant son abscisse ;
- `y` qui est un entier naturel représentant son ordonnée.

On suppose qu'il n'existe pas deux points d'identifiants distincts et de mêmes coordonnées.

La relation d'appartenance d'un point à un ensemble de points est représentée par la table `MEMBRE` à deux colonnes :

- `idpoint`, un entier naturel qui identifie un point ;
- `idensemble`, un entier naturel qui identifie un ensemble de points.

**Question 4.** Écrire une requête SQL qui renvoie les identifiants des ensembles auxquels appartient le point de coordonnées  $(a, b)$ .

**Question 5.** Écrire une requête SQL qui renvoie les coordonnées des points qui appartiennent à l'intersection des ensembles d'identifiants  $i$  et  $j$ .

**Question 6.** Écrire une requête SQL qui renvoie les identifiants des points appartenant à au moins un des ensembles auxquels appartient le point de coordonnées  $(a, b)$ .

### Partie III. Codage de Lebesgue

On souhaite implémenter en PYTHON une solution efficace au problème du calcul de l'intersection entre deux ensembles de points. La solution proposée s'appuie sur une structure de données appelée AQL. Cette structure de données suppose que les coordonnées des points sont représentées par leur codage de Lebesgue.

Le codage de Lebesgue d'un point de coordonnées  $(x, y) \in D_n \times D_n$  s'obtient par entrelacement des bits des représentations binaires de  $x$  et  $y$  en commençant par les bits de  $x$ . On suppose que les bits de poids forts sont situés à gauche dans les représentations binaires des entiers naturels.

Par exemple, si  $n$  vaut 3, si  $x$  vaut 6 (donc  $\overline{110}^2$  en binaire), et si  $y$  vaut 3 (donc  $\overline{011}^2$  en binaire) alors le codage de Lebesgue du point de coordonnées  $(x, y)$  est  $\overline{101101}^2$ , c'est-à-dire 45.

Le codage de Lebesgue d'un point peut être vu comme un nombre écrit dans la base formée des chiffres  $\overline{00}^2$ ,  $\overline{01}^2$ ,  $\overline{10}^2$  et  $\overline{11}^2$ . Ainsi, si  $n = 3$ , le point  $(6, 3) = (\overline{110}^2, \overline{011}^2)$  est codé par le nombre  $\overline{10}^2\overline{11}^2\overline{01}^2$ .

De plus, on utilisera la notation décimale 0, 1, 2 et 3 pour représenter les chiffres  $\overline{00}^2$ ,  $\overline{01}^2$ ,  $\overline{10}^2$  et  $\overline{11}^2$ , et on notera  $\overline{c_{n-1} \dots c_0}^\ell$  la représentation en base 4 du codage de Lebesgue d'un point de  $D_n \times D_n$ . Par exemple, pour  $n = 3$ , le codage de Lebesgue du point  $(6, 3)$  sera écrit  $\overline{231}^\ell$ .

En PYTHON, la séquence des chiffres d'un codage de Lebesgue d'un point de  $D_n \times D_n$  est stockée dans une liste de longueur  $n$  triée par poids décroissants : le chiffre de poids le plus fort se trouve en première position, le chiffre de poids le plus faible en dernière position. Ainsi, si  $n = 3$ , le codage de Lebesgue du point  $(6, 3)$  est représenté en PYTHON par la liste  $[2, 3, 1]$ .

**Question 7.** Soit  $n = 3$ , quelle liste PYTHON représente le codage de Lebesgue du point  $(1, 6)$  ?

**Question 8.** On suppose que l'on dispose d'une fonction `bits(x, k)`, qui prend en arguments deux entiers naturels  $x$  et  $k$ , et qui renvoie la valeur du bit de coefficient  $2^k$  dans la représentation binaire de  $x$ .

Écrire une fonction `code(n,p)` qui prend en arguments un entier strictement positif  $n$  et un point  $p$  représenté par une liste de longueur 2 dont les deux coordonnées sont prises dans  $D_n$ . Cette fonction renvoie le codage de Lebesgue de  $p$  représenté sous la forme d'une liste PYTHON.

## Partie IV. Représentation d'un ensemble de points

On utilise l'ordre lexicographique (autrement dit, l'ordre du dictionnaire) pour trier les codages. Soient  $c = \overline{c_{n-1} \dots c_0}^\ell$  et  $d = \overline{d_{n-1} \dots d_0}^\ell$  deux codages de Lebesgue de points de  $D_n \times D_n$ . On note  $c < d$  pour «  $c$  est strictement plus petit que  $d$  » si

$$\exists i, 0 \leq i < n \text{ tel que } \begin{cases} \forall j, j > i \Rightarrow c_j = d_j \\ c_i <_{\mathbb{N}} d_i \end{cases}$$

où  $<_{\mathbb{N}}$  est l'ordre usuel sur les entiers naturels.

**Question 9.** Trier les codages suivants par ordre croissant pour l'ordre lexicographique :  $\{\overline{311}^\ell, \overline{000}^\ell, \overline{012}^\ell, \overline{101}^\ell, \overline{233}^\ell\}$ .

**Question 10.** Écrire une fonction `compare_pcodes(n,c1,c2)`, qui prend en arguments deux codages de Lebesgue de  $D_n \times D_n$  et renvoie 0 s'ils sont égaux, et qui renvoie 1 si  $c2$  est plus grand par l'ordre lexicographique que  $c1$  et renvoie  $-1$  sinon.

Nous allons maintenant représenter un ensemble  $P$  de points de  $D_n \times D_n$  sous la forme d'une liste triée pour l'ordre lexicographique des codages de Lebesgue des points de  $P$ . En guise d'exemple, nous allons coder les points  $S_0 = \{(0,0), (1,0), (1,1), (2,2), (3,0), (0,1)\}$  de  $D_2 \times D_2$ . Ces points sont représentés en noir dans la figure 1 ci-dessous, dont l'origine est en bas à gauche, les abscisses croissent de gauche à droite et les ordonnées du bas vers le haut.

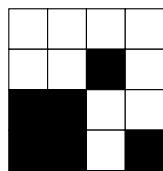


FIGURE 1 – L'ensemble de points  $S_0$

On considère d'abord les représentations binaires des coordonnées des points de  $S_0$  :

$$\overline{S_0}^2 = \{(\overline{00}^2, \overline{00}^2), (\overline{01}^2, \overline{00}^2), (\overline{01}^2, \overline{01}^2), (\overline{10}^2, \overline{10}^2), (\overline{11}^2, \overline{00}^2), (\overline{00}^2, \overline{01}^2)\}$$

à partir desquelles on calcule le codage de Lebesgue de chaque point

$$\overline{S_0}^\ell = \{\overline{00}^2\overline{00}^2, \overline{00}^2\overline{10}^2, \overline{00}^2\overline{11}^2, \overline{11}^2\overline{00}^2, \overline{10}^2\overline{10}^2, \overline{00}^2\overline{01}^2\} = \{\overline{00}^\ell, \overline{02}^\ell, \overline{03}^\ell, \overline{30}^\ell, \overline{22}^\ell, \overline{01}^\ell\}.$$

Cet ensemble, une fois trié pour l'ordre lexicographique, s'écrit

$$\{\overline{00}^\ell, \overline{01}^\ell, \overline{02}^\ell, \overline{03}^\ell, \overline{22}^\ell, \overline{30}^\ell\}$$

ce que l'on représente en PYTHON par la liste :

$[[0, 0], [0, 1], [0, 2], [0, 3], [2, 2], [3, 0]] .$

Remarquons que nous venons d'effectuer un changement de système de coordonnées. Le codage de Lebesgue du point  $(x, y)$  représente le chemin à emprunter pour atteindre  $(x, y)$  dans l'espace récursivement subdivisé en quadrants. En effet, en suivant la numérotation des quadrants donnée dans la figure 2 ci-dessous, on s'aperçoit par exemple que le point de coordonnées  $(1, 1)$  dans le système de coordonnées usuel est situé dans le quadrant 0 de  $D_2 \times D_2$  et qu'à l'intérieur de ce quadrant 0 subdivisé à son tour, le point  $(1, 1)$  est dans le quadrant 3.

1	3
0	2

FIGURE 2 – Numérotation des quadrants.

On peut vérifier que ces coordonnées  $[0, 3]$  dans ce nouveau système correspondent bien au codage de Lebesgue du point de coordonnées  $(1, 1)$  du système usuel.

Formellement, pour  $k < n$ , le quadrant atteint dans  $D_n \times D_n$  par le chemin  $c = d_1 \cdot d_2 \cdots d_k$  (avec  $d_i \in \{0, 1, 2, 3\}$ ) est défini comme suit :

- Si  $k$  vaut 0 alors le chemin  $c$  est vide et le quadrant atteint dans  $D_n \times D_n$  par  $c$  est l'ensemble des points de  $D_n \times D_n$ .
- Si  $k > 0$  alors le chemin est de la forme  $d_1 \cdot d_2 \cdots d_k$ . Dans ce cas, le quadrant atteint par  $d_1 \cdot d_2 \cdots d_k$  dans  $D_n \times D_n$  est l'ensemble des points de  $D_n \times D_n$  dont le codage de Lebesgue est de la forme  $\overline{d_1 d_2 \dots d_k c_{n-k-1} \dots c_0}^\ell$  pour  $c_{n-k-1}, \dots, c_0 \in \{0, 1, 2, 3\}$ .

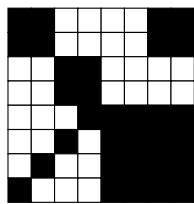
**Question 11.** On pose pour cette question  $n = 2$ . Donner la représentation sous forme de codage de Lebesgue compacté trié par ordre lexicographique de l'ensemble de points  $S_1$  valant  $\{(0, 0), (3, 3), (3, 2), (1, 1), (1, 2), (2, 2), (2, 3)\}$ .

## Partie V. Calcul efficace de l'intersection d'ensembles de points

**Compaction par codage des quadrants.** Soit  $S$  un ensemble de points de  $D_n \times D_n$  représenté par la liste  $L$  triée par ordre lexicographique des codages de Lebesgue de ses points (comme dans la partie précédente). En se dotant d'un symbole supplémentaire, notons-le 4, on compacte la liste  $L$  en représentant chaque sous-séquence (maximale)  $L'$  correspondant à un quadrant de chemin  $d_1 \cdots d_k$  par l'unique mot  $\overline{d_1 \cdots d_k \cdot 4 \cdots 4}^\ell$  (dans lequel on a rajouté  $(n - k)$  fois le symbole 4).

Notons qu'un codage de Lebesgue compacté représente *un ensemble de points* et non un unique point comme c'est le cas avec les codages de Lebesgue non compactés. Notons aussi que la liste des codages reste triée pour l'ordre lexicographique.

Par exemple, l'ensemble de points  $S_0$  est compactable en  $[[0, 4], [2, 2], [3, 0]]$ . Le codage  $[0, 4]$  représente le quadrant 0 situé en bas à gauche de la figure 1. Enfin, pour illustrer un cas où  $n > 2$ , la figure 3 décrit le codage compacté d'un ensemble de points de  $D_3 \times D_3$ .



est compacté en

$$\{\overline{000}^\ell, \overline{003}^\ell, \overline{030}^\ell, \overline{033}^\ell, \overline{114}^\ell, \overline{124}^\ell, \overline{244}^\ell, \overline{334}^\ell\}$$

FIGURE 3 – Un ensemble de points de  $D_3 \times D_3$  et sa représentation compactée.

**Structure de données d’AQL.** On appelle « AQL de l’ensemble de points  $S$  » la liste triée et compactée des codages de Lebesgue des points de l’ensemble  $S$ .

**Question 12.** Donner l’AQL de l’ensemble  $S_1$  de la question 11.

**Question 13.** Écrire une fonction `ksuffixe(n, k, q)` qui prend en arguments un entier  $n$  strictement positif, une liste  $q$  représentant le codage de Lebesgue compacté d’un quadrant de  $D_n \times D_n$  et un entier naturel  $k$  inférieur strictement à  $n$ . Si les  $k$  derniers chiffres de la liste  $q$  ont pour valeur 4, cette fonction renvoie une nouvelle liste semblable à la liste  $q$  mais dont les  $k + 1$  derniers chiffres valent 4. Sinon, cette fonction renvoie  $q$  inchangée.

Ainsi, `ksuffixe(4, 2, [0,1,4,4])` renvoie `[0,4,4,4]`, et `ksuffixe(4, 2, [0,1,2,4])` renvoie `[0,1,2,4]`.

**Question 14.** L’algorithme de compaction d’une liste triée de codages de Lebesgue consiste à parcourir  $n$  fois la liste représentant l’ensemble de points. L’itération  $k$  vise à remplacer quatre codages successifs formant un quadrant complet de côté  $2^{k+1}$  par la représentation compactée de ce quadrant.

Écrire une fonction `compacte(n,s)` qui prend en arguments un entier strictement positif  $n$  et un ensemble de points  $P$  de  $D_n \times D_n$  représenté par la liste triée  $s$  des codages de Lebesgue de ses points. Cette fonction renvoie l’AQL de l’ensemble de points  $P$ .

**Question 15.** On remarque que l’ordre lexicographique  $<$  défini plus haut s’adapte sans changement aux codages de Lebesgue compactés. Cependant, on souhaite comparer deux codages de Lebesgue compactés en termes des relations d’inclusion et d’exclusion des ensembles de points qu’ils représentent.

Écrire une fonction `compare_ccodes(n,p,q)` qui prend en arguments un entier strictement positif  $n$ , une liste  $p$  contenant le codage de Lebesgue compacté d’un quadrant  $P$  de  $D_n \times D_n$  et une liste  $q$  contenant le codage de Lebesgue compacté d’un quadrant  $Q$  de  $D_n \times D_n$ . Cinq valeurs de retour sont possibles :

- l’entier 0 si les quadrants sont égaux ;
- l’entier 1 si les quadrants  $P$  et  $Q$  sont disjoints et  $p < q$  ;
- l’entier  $-1$  si les quadrants  $P$  et  $Q$  sont disjoints et  $q < p$  ;
- l’entier 2 si  $P \subset Q$  ;
- l’entier  $-2$  si  $Q \subset P$ .

Par exemple, `compare_ccodes(3, [1,4,4], [2,4,4])` renvoie 1 et `compare_ccodes(3, [1,2,4], [1,4,4])` renvoie 2.

**Question 16.** Pour calculer efficacement l'intersection de deux ensembles de points représentés par leur AQL respectif, on *fusionne* les deux listes triées qui leur correspondent.

En utilisant `compare_ccodes`, écrire une fonction `intersection2(n,p,q)` qui prend en arguments un entier strictement positif  $n$  ainsi que deux AQL  $p$  et  $q$  représentant respectivement deux ensembles  $P$  et  $Q$  de points de  $D_n \times D_n$ . Cette fonction renvoie un AQL représentant  $P \cap Q$ . Le nombre d'appels à la fonction `compare_ccode` effectués par `intersection2(n,p,q)` doit être en  $O(\text{len}(p) + \text{len}(q))$ .

\* \*  
\*

## ÉTUDE DE TRAFIC ROUTIER

---

Ce sujet concerne la conception d'un logiciel d'étude de trafic routier. On modélise le déplacement d'un ensemble de voitures sur des files à sens unique (voir Figure 1(a) et 1(b)). C'est un schéma simple qui peut permettre de comprendre l'apparition d'embouteillages et de concevoir des solutions pour fluidifier le trafic.

Le sujet comporte des questions de programmation. Le langage à utiliser est Python.

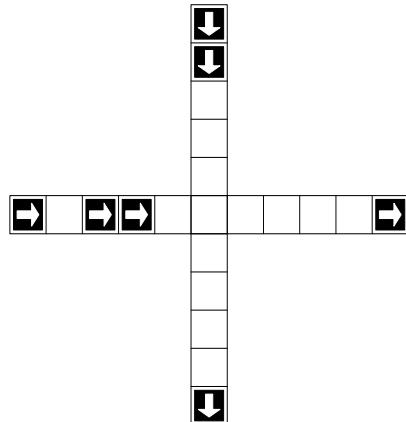
### Notations

Soit  $L$  une liste,

- on note  $\text{len}(L)$  sa longueur ;
- pour  $i$  entier,  $0 \leq i < \text{len}(L)$ , l'élément de la liste d'indice  $i$  est noté  $L[i]$  ;
- pour  $i$  et  $j$  entiers,  $0 \leq i < j \leq \text{len}(L)$ ,  $L[i : j]$  est la sous-liste composée des éléments  $L[i], \dots, L[j - 1]$  ;
- $p * L$ , avec  $p$  entier, est la liste obtenue en concaténant  $p$  copies de  $L$ . Par exemple,  $3 * [0]$  est la liste  $[0, 0, 0]$ .



(a) Représentation d'une file de longueur onze comprenant quatre voitures, situées respectivement sur les cases d'indices 0, 2, 3 et 10.



(b) Configuration représentant deux files de circulation à sens unique se croisant en une case. Les voitures sont représentées par un carré noir.

FIGURE 1 – Files de circulation

### Partie I. Préliminaires

Dans un premier temps, on considère le cas d'une seule file, illustré par la Figure 1(a). Une file de longueur  $n$  est représentée par  $n$  cases. Une case peut contenir au plus une voiture. Les voitures présentes dans une file circulent toutes dans la même direction (sens des indices croissants, désigné par les flèches sur la Figure 1(a)) et sont indifférencierées.

- Q1** – Expliquer comment représenter une file de voitures à l'aide d'une liste de booléens.
- Q2** – Donner une ou plusieurs instructions Python permettant de définir une liste  $A$  représentant la file de voitures illustrée par la Figure 1(a).
- Q3** – Soit  $L$  une liste représentant une file de longueur  $n$  et  $i$  un entier tel que  $0 \leq i < n$ . Définir en Python la fonction  $\text{occupe}(L, i)$  qui renvoie `True` lorsque la case d'indice  $i$  de la file est occupée par une voiture et `False` sinon.
- Q4** – Combien existe-t-il de files différentes de longueur  $n$ ? Justifier votre réponse.

**Q5** – Écrire une fonction `egal(L1, L2)` retournant un booléen permettant de savoir si deux listes  $L1$  et  $L2$  sont égales.

**Q6** – Que peut-on dire de la complexité de cette fonction ?

**Q7** – Préciser le type de retour de cette fonction.

## Partie II. Déplacement de voitures dans la file

On identifie désormais une file de voitures à une liste. On considère les schémas de la Figure 2 représentant des exemples de files. Une *étape de simulation pour une file* consiste à déplacer les voitures de la file, à tour de rôle, en commençant par la voiture la plus à droite, d'après les règles suivantes :

- une voiture se trouvant sur la case la plus à droite de la file sort de la file ;
- une voiture peut avancer d'une case vers la droite si elle arrive sur une case inoccupée ;
- une case libérée par une voiture devient inoccupée ;
- la case la plus à gauche peut devenir occupée ou non, selon le cas considéré.

On suppose avoir écrit en Python la fonction `avancer` prenant en paramètres une liste de départ, un booléen indiquant si la case la plus à gauche doit devenir occupée lors de l'étape de simulation, et renvoyant la liste obtenue par une étape de simulation.

Par exemple, l'application de cette fonction à la liste illustrée par la Figure 2(a) permet d'obtenir soit la liste illustrée par la Figure 2(b) lorsque l'on considère qu'aucune voiture nouvelle n'est introduite, soit la liste illustrée par la Figure 2(c) lorsque l'on considère qu'une voiture nouvelle est introduite.

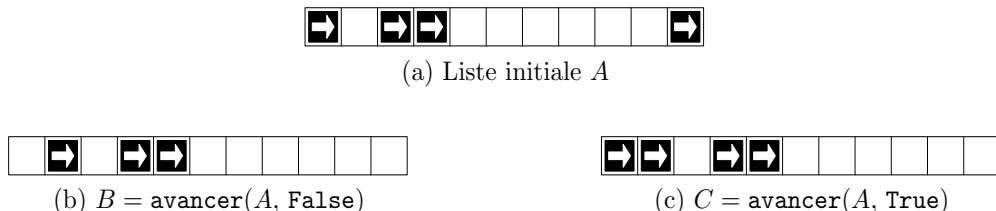


FIGURE 2 – Étape de simulation

**Q8** – Étant donnée  $A$  la liste définie à la question 2, que renvoie `avancer(avancer(A, False), True)` ?

**Q9** – On considère  $L$  une liste et  $m$  l'indice d'une case de cette liste ( $0 \leq m < \text{len}(L)$ ). On s'intéresse à une *étape partielle* où seules les voitures situées sur la case d'indice  $m$  ou à droite de cette case peuvent avancer normalement, les autres voitures ne se déplaçant pas.

Par exemple, la file devient .

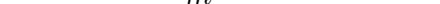
Définir en Python la fonction `avancer_fin(L, m)` qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste sans modifier  $L$ .

**Q10** – Soient  $L$  une liste,  $b$  un booléen et  $m$  l'indice d'une case *inoccupée* de cette liste. On considère une étape partielle où seules les voitures situées à gauche de la case d'indice  $m$  se déplacent, les autres voitures ne se déplacent pas. Le booléen  $b$  indique si une nouvelle voiture est introduite sur la case la plus à gauche.

Par exemple, la file devient lorsque aucune nouvelle voiture n'est introduite.

Définir en Python la fonction `avancer_debut(L, b, m)` qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste sans modifier  $L$ .

□ **Q11** – On considère une liste  $L$  dont la case d'indice  $m > 0$  est temporairement inaccessible et bloque l'avancée des voitures. Une voiture située immédiatement à gauche de la case d'indice  $m$  ne peut pas avancer. Les voitures situées sur les cases plus à gauche peuvent avancer, à moins d'être bloquées par une case occupée, les autres voitures ne se déplacent pas. Un booléen  $b$  indique si une nouvelle voiture est introduite lorsque cela est possible.

Par exemple, la file  devient  lorsque aucune nouvelle voiture n'est introduite.

Définir en Python la fonction `avancer_debut_bloc(L, b, m)` qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste.

On considère dorénavant deux files  $L1$  et  $L2$  de même longueur impaire se croisant en leur milieu ; on note  $m$  l'indice de la case du milieu. La file  $L1$  est toujours prioritaire sur la file  $L2$ . Les voitures ne peuvent pas quitter leur file et la case de croisement ne peut être occupée que par une seule voiture. Les voitures de la file  $L2$  ne peuvent accéder au croisement que si une voiture de la file  $L1$  ne s'apprête pas à y accéder. Une *étape de simulation à deux files* se déroule en deux temps. Dans un premier temps, on déplace toutes les voitures situées sur le croisement ou après. Dans un second temps, les voitures situées avant le croisement sont déplacées en respectant la priorité. Par exemple, partant d'une configuration donnée par la Figure 3(a), les configurations successives sont données par les Figures 3(b), 3(c), 3(d), 3(e) et 3(f) en considérant qu'*aucune nouvelle voiture n'est introduite*.

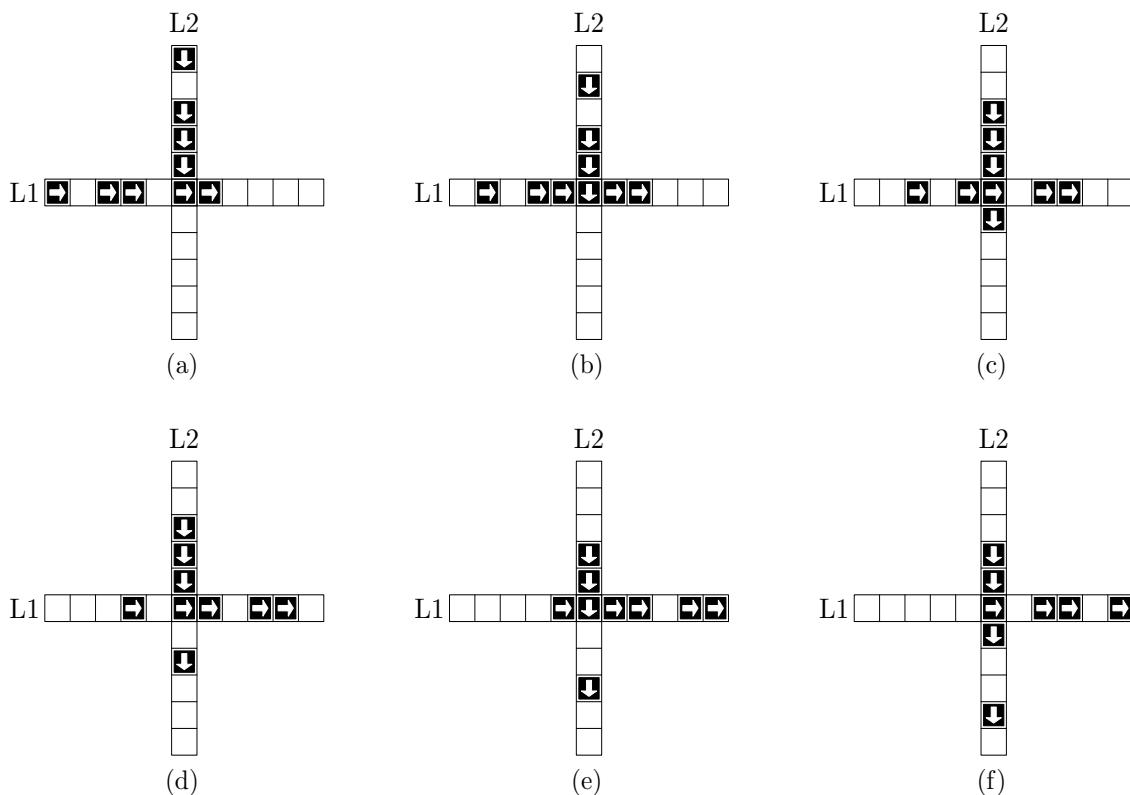


FIGURE 3 – Étapes de simulation à deux files

### Partie III. Une étape de simulation à deux files

L'objectif de cette partie est de définir en Python l'algorithme permettant d'effectuer une étape de simulation pour ce système à deux files.

**Q12** – En utilisant le langage Python, définir la fonction `avancer_files(L1, b1, L2, b2)` qui renvoie le résultat d'une étape de simulation sous la forme d'une liste de deux éléments notée  $[R1, R2]$  sans changer les listes  $L1$  et  $L2$ . Les booléens  $b1$  et  $b2$  indiquent respectivement si une nouvelle voiture est introduite dans les files  $L1$  et  $L2$ . Les listes  $R1$  et  $R2$  correspondent aux listes après déplacement.

**Q13** – On considère les listes

$$D = [\text{False}, \text{True}, \text{False}, \text{True}, \text{False}], \quad E = [\text{False}, \text{True}, \text{True}, \text{False}, \text{False}]$$

Que renvoie l'appel `avancer_files(D, False, E, False)` ?

#### Partie IV. Transitions

**Q14** – En considérant que de nouvelles voitures peuvent être introduites sur les premières cases des files lors d'une étape de simulation, décrire une situation où une voiture de la file  $L2$  serait indéfiniment bloquée.

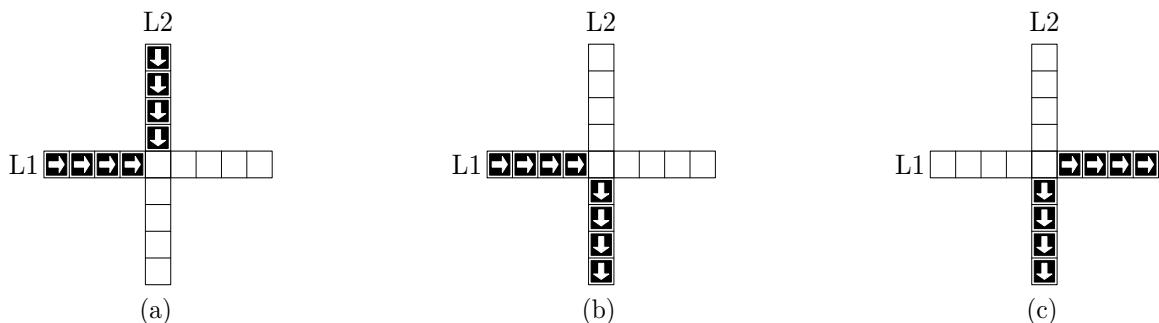


FIGURE 4 – Étude de configurations

**Q15** – Étant données les configurations illustrées par la Figure 4, combien d'étapes sont nécessaires (on demande le nombre minimum) pour passer de la configuration 4(a) à la configuration 4(b) ? Justifier votre réponse.

**Q16** – Peut-on passer de la configuration 4(a) à la configuration 4(c) ? Justifier votre réponse.

#### Partie V. Atteignabilité

Certaines configurations peuvent être néfastes pour la fluidité du trafic. Une fois ces configurations identifiées, il est intéressant de savoir si elles peuvent apparaître. Lorsque c'est le cas, on dit qu'une telle configuration est *atteignable*.

Pour savoir si une configuration est atteignable à partir d'une configuration initiale, on a écrit le code *incomplet* donné en annexe.

Le langage Python sait comparer deux listes de booléens à l'aide de l'opérateur usuel `<`, on peut ainsi utiliser la méthode `sort` pour trier une liste de listes de booléens.

**Q17** – Écrire en langage Python une fonction `elim_double(L)` non récursive, de complexité linéaire en la taille de  $L$ , qui élimine les éléments apparaissant plusieurs fois dans une liste triée  $L$  et renvoie la liste triée obtenue. Par exemple `elim_double([1, 1, 3, 3, 3, 7])` doit renvoyer la liste `[1, 3, 7]`.

On dispose de la fonction suivante :

```

1 def doublons(liste):
2     if len(liste)>1:
3         if liste[0] != liste[1]:
4             return [liste[0]] + doublons(liste[1:])
5         del liste[1]
6         return doublons(liste)
7     else:
8         return liste

```

**□ Q18** – Que retourne l'appel suivant ?

doublons([1, 1, 2, 2, 3, 3, 3, 5])

**□ Q19** – Cette fonction est-elle utilisable pour éliminer les éléments apparaissant plusieurs fois dans une liste *non* triée ? Justifier.

**□ Q20** – La fonction **recherche** donnée en annexe permet d'établir si la configuration correspondant à **but** est atteignable en partant de l'état **init**. Préciser le type de retour de la fonction **recherche**, le type des variables **but** et **espace**, ainsi que le type de retour de la fonction **successeurs**.

**□ Q21** – Afin d'améliorer l'efficacité du test **if but in espace**, ligne 10 de l'annexe, on propose de le remplacer par **if in1(but, espace)** ou bien par **if in2(but, espace)**, avec **in1** et **in2** deux fonctions définies ci-dessous. On considère que le paramètre **liste** est une liste triée par ordre croissant.

Quel est le meilleur choix ? Justifier.

```

1 def in1(element,liste):
2     a = 0
3     b = len(liste)-1
4     while a <= b and element >= liste[a]:
5         if element == liste[a]:
6             return True
7         else:
8             a = a + 1
9     return False
10
11
12 def in2(element,liste):
13     a = 0
14     b = len(liste)-1
15     while a < b:
16         pivot = (a+b) // 2 # l'opérateur // est la division entière
17         if liste[pivot] < element:
18             a = pivot + 1
19         else:
20             b = pivot
21     if element == liste[a]:
22         return True
23     else:
24         return False

```

**□ Q22** – Afin de comparer plus efficacement les files représentées par des listes de booléens on remarque que ces listes représentent un codage binaire où **True** correspond à 1 et **False** à 0. Écrire la fonction **versEntier(L)** prenant une liste de booléens en paramètre et renvoyant l'entier correspondant. Par exemple, l'appel **versEntier([True, False, False])** renverra 4.

**□ Q23** – On veut écrire la fonction inverse de `versEntier`, transformant un entier en une liste de booléens. Que doit être au minimum la valeur de `taille` pour que le codage obtenu soit satisfaisant ? On suppose que la valeur de `taille` est suffisante. Quelle condition booléenne faut-il écrire en ligne 4 du code ci-dessous ?

```

1 | def versFile(n, taille):
2 |     res = taille * [False]
3 |     i = taille - 1
4 |     while ...:
5 |         if (n % 2) != 0: # % est le reste de la division entière
6 |             res[i] = True
7 |         n = n // 2 # // est la division entière
8 |         i = i - 1
9 |     return res

```

**□ Q24** – Montrer qu'un appel à la fonction `recherche` de l'annexe se termine toujours.

**□ Q25** – Compléter la fonction `recherche` pour qu'elle indique le nombre minimum d'étapes à faire pour passer de `init` à `but` lorsque cela est possible. Justifier la réponse.

## Partie VI. Base de données

On modélise ici un réseau routier par un ensemble de *croisements* et de *voies* reliant ces croisements. Les voies partent d'un croisement et arrivent à un autre croisement. Ainsi, pour modéliser une route à double sens, on utilise deux voies circulant en sens opposés.

La base de données du réseau routier est constituée des relations suivantes :

- Croisement(id, longitude, latitude)
- Voie(id, longueur, id\_croisement\_debut, id\_croisement\_fin)

Dans la suite on considère  $c$  l'identifiant (`id`) d'un croisement donné.

**□ Q26** – Écrire la requête SQL qui renvoie les identifiants des croisements atteignables en utilisant une seule voie à partir du croisement ayant l'identifiant  $c$ .

**□ Q27** – Écrire la requête SQL qui renvoie les longitudes et latitudes des croisements atteignables en utilisant une seule voie, à partir du croisement  $c$ .

**□ Q28** – Que renvoie la requête SQL suivante ?

```

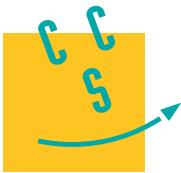
1 | SELECT V2.id_croisement_fin
2 | FROM   Voie as V1
3 | JOIN   Voie as V2
4 | ON     V1.id_croisement_fin = V2.id_croisement_debut
5 | WHERE  V1.id_croisement_debut = c

```

## Annexe

```
1  def recherche(but, init):
2      espace = [init]
3      stop = False
4      while not stop:
5          ancien = espace
6          espace = espace + successeurs(espace)
7          espace.sort() # permet de trier espace par ordre croissant
8          espace = elim_double(espace)
9          stop = egal(ancien,espace) # fonction définie à la question 5
10         if but in espace:
11             return True
12     return False
13
14
15 def successeurs(L):
16     res = []
17     for x in L:
18         L1 = x[0]
19         L2 = x[1]
20         res.append( avancer_files(L1, False, L2, False) )
21         res.append( avancer_files(L1, False, L2, True) )
22         res.append( avancer_files(L1, True, L2, False) )
23         res.append( avancer_files(L1, True, L2, True) )
24     return res
25
26 # dans une liste triée, elim_double enlève les éléments apparaissant plus d'une fois
27 # exemple : elim_double([1, 1, 2, 3, 3]) renvoie [1, 2, 3]
28 def elim_double(L):
29     # code à compléter
30
31 # exemple d'utilisation
32 # debut et fin sont des listes composées de deux files de même longueur impaire,
33 # la première étant prioritaire par rapport à la seconde
34 debut = [5*[False], 5*[False]]
35 fin = [3*[False]+2*[True], 3*[False]+2*[True]]
36 print(recherche(fin,debut))
```

**Fin de l'épreuve.**



CONCOURS CENTRALE-SUPÉLEC

# Informatique

**MP, PC, PSI, TSI**

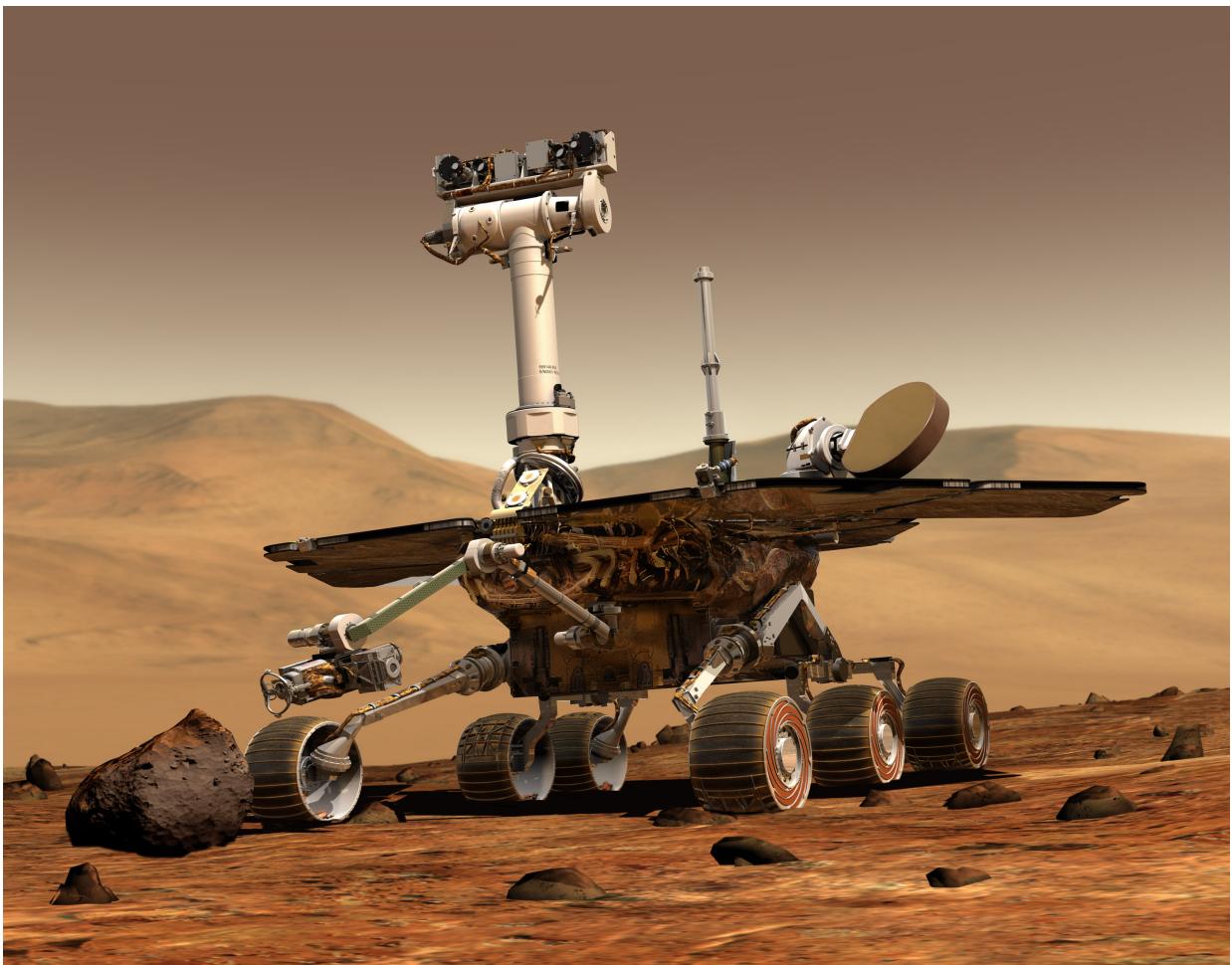
3 heures

Calculatrices autorisées

**2017**

## *Mars Exploration Rovers Mission d'exploration martienne*

*Mars Exploration Rovers* (MER) est une mission de la NASA qui cherche à étudier le rôle joué par l'eau dans l'histoire de la planète Mars. Deux robots géologues, Spirit et Opportunity (figure 1), se sont posés sur cette planète, sur deux sites opposés, en janvier 2004. Leur mission est de rechercher et d'analyser différents types de roches et de sols qui peuvent contenir des indices sur la présence d'eau. Ils sont équipés de six roues et d'une suspension spécialement conçue pour leur permettre de se déplacer quelle que soit la nature du terrain rencontré. Leur cahier des charges prévoyait une durée de vie de 90 jours martiens (le jour martien est environ 40 minutes plus long que le jour terrestre). Spirit a cessé d'émettre le 22 mars 2010, soit 2210 jours martiens après son arrivée sur la planète. Début 2017, Opportunity est toujours en activité et il a parcouru plus de 44 km sur Mars.



**Figure 1** Vue d'artiste d'un robot géologue de la mission *Mars Exploration Rovers* (NASA/JPL – Caltech/Cornell)

Chaque robot est équipé de plusieurs instruments d'analyse (caméra, microscope, spectromètres) et d'un bras qui permet d'amener les instruments au plus près des roches et sols dignes d'intérêt. À partir de photographies de la surface de la planète, prises à plusieurs longueurs d'ondes par différents satellites et par le robot lui-même, les scientifiques de la NASA définissent une liste d'emplacements (*points d'intérêt* ou PI) où effectuer des analyses. Cette liste est transmise au robot qui doit se rendre à chaque emplacement indiqué et y effectuer les analyses prévues. Chaque robot est capable d'effectuer un certain nombre de types d'analyses géologiques correspondant aux différents instruments dont il dispose. Une fois tous les points d'intérêts visités et les résultats des analyses

transmis à la Terre, le robot reçoit une nouvelle liste de points d'intérêts et démarre une nouvelle *exploration*. Compte-tenu des contraintes de transmission entre la Terre et les robots (latence, périodes d'ombre, faible débit, etc.) il est prévu que les robots travaillent en autonomie pour planifier le parcours de chaque exploration. Ainsi, une fois la liste des points d'intérêt reçue, le robot analyse le terrain afin de détecter d'éventuels obstacles et détermine le meilleur chemin lui permettant de visiter l'ensemble de ces points en dépensant le moins d'énergie possible.

Après s'être intéressé à l'enregistrement des explorations, des points d'intérêts correspondants et des analyses à y mener, ce sujet aborde trois algorithmes qui peuvent être utilisés par le robot pour déterminer le meilleur parcours lui permettant de visiter chaque point d'intérêt une et une seule fois. Pour cela nous faisons quelques hypothèses simplificatrices.

- La zone d'exploration est dépourvue d'obstacle : le robot peut rejoindre directement en ligne droite n'importe quel point d'intérêt.
- Le sol est horizontal et de nature constante : l'énergie utilisé pour se déplacer entre deux points ne dépend que de leur distance, autrement dit le meilleur chemin est le plus court.
- La courbure de la planète est négligée compte tenu de la dimension réduite de la zone d'exploration : nous travaillerons en géométrie euclidienne et les points d'intérêts seront repérés par leurs coordonnées cartésiennes à l'intérieur de la zone d'exploration.

Les seuls langages de programmation autorisés dans cette épreuve sont Python et SQL. Toutes les questions sont indépendantes. Néanmoins, il est possible de faire appel à des fonctions ou procédures créées dans d'autres questions. Dans tout le sujet on suppose que les bibliothèques `math`, `numpy` et `random` ont été importées grâce aux instructions

```
import math
import numpy as np
import random
```

Si les candidats font appel à des fonctions d'autres bibliothèques ils doivent préciser les instructions d'importation correspondantes.

Ce sujet utilise la syntaxe des annotations pour préciser le types des arguments et du résultat des fonctions à écrire. Ainsi

```
def maFonction(n:int, x:float, d:str) -> np.ndarray:
```

signifie que la fonction `maFonction` prend trois arguments, le premier est un entier, le deuxième un nombre à virgule flottante et le troisième une chaîne de caractères et qu'elle renvoie un tableau `numpy`.

Une liste de fonctions utiles est donnée à la fin du sujet.

## I Création d'une exploration et gestion des points d'intérêt

Une exploration est un ensemble de points d'intérêt à l'intérieur d'une zone géographique limitée, une série d'analyses étant associée à chaque point d'intérêt. Chaque type d'analyse que le robot peut effectuer est codifié et référencé par un nombre entier. Un point d'intérêt est repéré par deux entiers, positifs ou nuls, correspondant à ses coordonnées cartésiennes en millimètres à l'intérieur de la zone d'exploration. L'ensemble des points d'intérêt d'une exploration qui en contient  $n$  est représenté par un objet de type `numpy.ndarray`, à éléments entiers, à 2 colonnes et  $n$  lignes, l'élément d'indice  $i, 0$  correspondant à l'abscisse du point d'intérêt  $i$  et l'élément d'indice  $i, 1$  à son ordonnée.

	$x$	$y$
0	345	635
1	1076	415
2	38	859
3	121	582

**Figure 2** Exemple d'exploration avec quatre points d'intérêt

### I.A – Génération d'une exploration d'essai

#### I.A.1) Choix de points au hasard

a) Afin de disposer de données pour tester les différents algorithmes de calcul de chemin qui seront développés plus tard, écrire une fonction qui construit une exploration au hasard. Cette fonction d'entête

```
def générer_PI(n:int, cmax:int) -> np.ndarray:
```

prend en paramètres le nombre de points d'intérêts à générer et la largeur de la zone d'exploration (supposée carrée) et renvoie un objet de type `numpy.ndarray` contenant les coordonnées de  $n$  points **deux à deux distincts** choisis au hasard dans la zone d'exploration (figure 2).

b) Quelles contraintes doivent vérifier les arguments de la fonction `générer_PI` ?

#### I.A.2) Calcul des distances

On dispose de la fonction d'entête

```
def position_robot() -> tuple:
```

qui renvoie un couple donnant les coordonnées actuelles du robot dans le système de coordonnées de l'exploration à planifier. Ainsi l'instruction `x, y = position_robot()` permet de récupérer les coordonnées courantes du robot.

Afin de faciliter l'application des différents algorithmes de recherche de chemin, on souhaite construire un tableau des distances entre les différents points d'intérêt d'une exploration et entre ceux-ci et la position courante du robot au moment du calcul. Écrire une fonction d'entête

```
def calculer_distances(PI:np.ndarray) -> np.ndarray:
```

qui prend en paramètre un tableau de  $n$  points d'intérêt tel que décrit précédemment et renvoie un tableau de nombres flottants, de dimension  $(n + 1) \times (n + 1)$ , tel que l'élément d'indice  $i, j$  fournit la distance entre les points d'intérêt  $i$  et  $j$ , l'indice  $n$  désignant le point de départ du robot.

#### I.B – Traitement d'image

On dispose de photographies d'une zone d'exploration effectuées à différentes longueur d'onde. Chaque photographie a été mises à l'échelle de la zone d'exploration puis stockée dans un tableau numpy (`np.ndarray`) à deux dimensions. Les dimensions correspondent aux coordonnées géographiques du point photographié, chaque élément est un entier, compris entre 0 et 255, donnant l'intensité du point considéré sur l'image. Ainsi l'élément d'indice `x, y` contient un entier, compris entre 0 et 255, correspondant à l'intensité sur la photographie considérée du point de coordonnées  $(x, y)$ . À partir de ces photographies, les géologues déterminent les endroits à analyser en filtrant ceux qui ont un profil d'émission caractéristique de certaines roches intéressantes.

#### I.B.1) Analyse d'une image

La fonction `F1` ci-dessous prend en paramètre une photographie représentée comme décrit plus haut. Expliquer ce que fait cette fonction et décrire son résultat.

```
1 def F1(photo:np.ndarray) -> np.ndarray:
2     n = photo.min()
3     b = photo.max()
4     h = np.zeros(b - n + 1, np.int64)
5     for p in photo.flat:
6         h[p - n] += 1
7     return h
```

#### I.B.2) Sélection de points d'intérêts

Écrire une fonction d'entête

```
def sélectionner_PI(photo:np.ndarray, imin:int, imax:int) -> np.ndarray:
```

où `photo` est un tableau représentant une photographie. Le résultat de la fonction `sélectionner_PI` est un tableau à deux dimensions, de structure similaire à celui décrit figure 2, contenant les coordonnées des points dont l'intensité sur la photographie est comprise entre `imin` et `imax`.

#### I.C – Base de données

Afin d'assurer son autonomie opérationnelle, le robot dispose localement des informations nécessaires à son fonctionnement quotidien. Ainsi, il enregistre la durée d'utilisation de ses différents instruments embarqués. Il connaît également les différents types d'analyses qu'il peut effectuer et, pour chacun de ces types, les instruments à utiliser. Il enregistre la prochaine exploration, c'est-à-dire les différents points d'intérêt qu'il doit visiter et pour chacun la ou les analyses qu'il doit effectuer. D'autre part, il conserve les résultats d'analyses effectuées lors de ses explorations passées. Ces résultats ne sont effacés qu'après confirmation de leur bonne transmission sur Terre.

Ces différentes informations sont stockées dans une base de données relationnelle dont le modèle physique est schématisé figure 3.

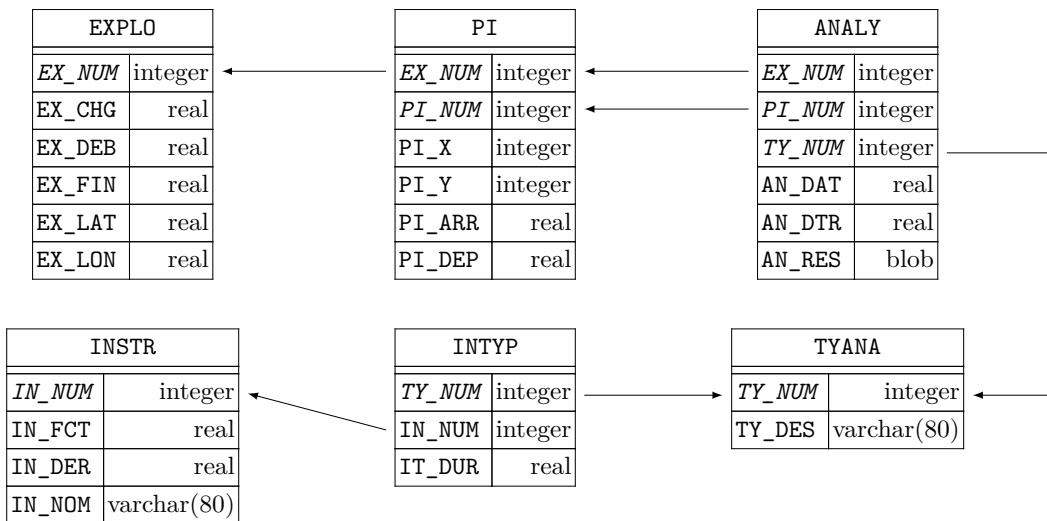


Figure 3 Structure physique de la base de données d'un robot

Cette base comporte les six tables suivantes :

- la table EXPLO des explorations, avec les colonnes
  - *EX\_NUM* numéro (entier) de l'exploration (clé primaire)
  - *EX\_CHG* date de transmission des points d'intérêts de cette exploration
  - *EX\_DEB* date de début de l'exploration (NULL si l'exploration n'est pas encore commencée)
  - *EX\_FIN* date de fin de l'exploration (NULL si l'exploration n'est pas encore terminée)
  - *EX\_LAT* latitude (en degrés décimaux) du point de coordonnées (0,0) de la zone d'exploration
  - *EX\_LON* longitude (en degrés décimaux) du point de coordonnées (0,0) de la zone d'exploration
- la table PI des points d'intérêts, de clé primaire (*EX\_NUM*, *PI\_NUM*), avec les colonnes
  - *EX\_NUM* numéro de l'exploration à laquelle appartient le point d'intérêt
  - *PI\_NUM* numéro du point d'intérêt dans l'exploration (au sein d'une exploration les PI sont numérotés en séquence en commençant à 0, ce numéro n'a pas de rapport avec l'ordre dans lequel les PI sont explorés par le robot)
  - *PI\_X* l'abscisse du point d'intérêt dans la zone d'exploration (entier positif en millimètres)
  - *PI\_Y* l'ordonnée du point d'intérêt dans la zone d'exploration (entier positif en millimètres)
  - *PI\_ARR* date d'arrivée du robot au point d'intérêt (NULL si ce point n'a pas encore été visité)
  - *PI\_DEP* date à laquelle le robot a quitté le point d'intérêt (NULL si ce point n'a pas encore été exploré ou si la visite est en cours)
- la table INSTR des instruments embarqués, avec les colonnes
  - *IN\_NUM* le numéro (entier) de l'instrument (clé primaire)
  - *IN\_FCT* la durée pendant lequel l'instrument a déjà été utilisé depuis l'arrivée sur la planète (nombre décimal : fraction de jour martien)
  - *IN\_DER* la date de la dernière utilisation de l'instrument
  - *IN\_NOM* nom de l'instrument
- la table TYANA des types d'analyses à effectuer, avec les colonnes
  - *TY\_NUM* le numéro de référence (entier) du type d'analyse (clé primaire)
  - *TY\_des* le nom du type d'analyse
- la table INTYP des instruments utilisés pour un type d'analyse, de clé primaire (*TY\_NUM*, *IN\_NUM*), avec les colonnes
  - *TY\_NUM* le numéro de référence (entier) du type d'analyse
  - *IN\_NUM* le numéro (entier) de l'instrument
  - *IT\_DUR* la durée standard d'utilisation de l'instrument dans ce type d'analyse (nombre décimal : fraction de jour martien)
- la table ANALY indiquant pour chaque point d'intérêt les types d'analyses à effectuer ou effectuées, de clé primaire (*EX\_NUM*, *PI\_NUM*, *TY\_NUM*) et avec les colonnes
  - *EX\_NUM* numéro de l'exploration à laquelle appartient le point d'intérêt
  - *PI\_NUM* numéro du point d'intérêt dans l'exploration
  - *TY\_NUM* type de l'analyse

- AN\_DAT date de l'analyse (NULL si l'analyse n'a pas été effectuée)
- AN\_DTR date de transmission sur Terre des résultats de l'analyse (NULL si l'analyse n'a pas été transmise)
- AN\_RES résultat de l'analyse (donnée opaque dont la signification dépend du type d'analyse)

Toutes les dates sont stockées sous forme d'un nombre décimal correspondant au nombre de jours martiens depuis l'arrivée du robot sur la planète.

**I.C.1)** Écrire une requête SQL qui donne le numéro de l'exploration en cours, s'il y en a une.

**I.C.2)** Écrire une requête SQL qui donne, pour une exploration dont on connaît le numéro, la liste des points d'intérêts de cette exploration avec leurs coordonnées.

**I.C.3)** Écrire une requête SQL qui donne la surface, en mètres carrés, de chaque zone déjà explorée par le robot. La zone d'exploration est définie comme le plus petit rectangle qui englobe l'ensemble des points d'intérêts de l'exploration et dont les bords sont parallèles aux axes de référence (axes des abscisses et des ordonnées).

**I.C.4)** Quelle est la surface maximale d'une zone d'exploration que peut stocker cette base de données ?

**I.C.5)** Écrire une requête SQL qui donne, pour l'exploration en cours, le nombre de fois où chaque instrument doit être utilisé et sa durée d'utilisation théorique (en jours martiens) pour la totalité de l'exploration.

## II Planification d'une exploration : première approche

Avant de démarrer une nouvelle exploration, le robot doit déterminer un chemin qui lui permet de passer par tous les points d'intérêts une et une seule fois. L'enjeu de l'opération est de trouver le chemin le plus court possible afin de limiter la dépense d'énergie et de limiter l'usure du robot.

Chaque point d'intérêt sera repéré par un entier positif ou nul correspondant à son indice dans le tableau des points d'intérêt. Un chemin d'exploration sera représenté par un objet de type `list` donnant les indices des points d'intérêt dans l'ordre de leur parcours.

### II.A – Quelques fonctions utilitaires

#### II.A.1) Longueur d'un chemin

Écrire une fonction d'entête

```
def longueur_chemin(chemin:list, d:np.ndarray) -> float:
```

qui prend en paramètre un chemin à parcourir et la matrice des distances entre points d'intérêt (telle que renvoyée par la fonction `calculer_distances`) et renvoie la distance que doit effectuer le robot pour suivre ce chemin en partant de sa position courante (correspondant à la dernière ligne/colonne du tableau `d`) et en visitant tous les points d'intérêt dans l'ordre indiqué.

#### II.A.2) Normalisation d'un chemin

Écrire une fonction d'entête

```
def normaliser_chemin(chemin:list, n:int) -> list:
```

qui prend en paramètre une liste d'entiers et renvoie une liste correspondant à un chemin valide, c'est-à-dire contenant une seule fois tous les entiers entre 0 et `n` (exclu). Pour cela cette fonction commence par supprimer les éventuels doublons (en ne conservant que la première occurrence) et les valeurs supérieures ou égales à `n`, sans modifier l'ordre relatif des éléments conservés, puis ajoute à la fin les éventuels éléments manquants en ordre croissant de numéros.

### II.B – Force brute

Pour rechercher le plus court chemin, on peut imaginer de considérer tous les chemins possibles et de calculer leur longueur. On obtiendra ainsi à coup sûr le chemin le plus court.

**II.B.1)** Déterminer en fonction de `n`, nombre de points à visiter, le nombre de chemins possibles passant exactement une fois par chacun des points.

**II.B.2)** Cet algorithme est-il utilisable pour une zone d'exploration contenant 20 points d'intérêts ? Justifier.

### II.C – Algorithme du plus proche voisin

Une idée simple pour obtenir un algorithme utilisable est de construire un chemin en choisissant systématiquement le point, non encore visité, le plus proche de la position courante.

**II.C.1)** Écrire une fonction d'entête

```
def plus_proche_voisin(d:np.ndarray) -> list:
```

qui prend en paramètre le tableau des distances résultant de la fonction `calculer_distances` (question I.A.2) et fournit un chemin d'exploration en appliquant l'algorithme du plus proche voisin.

**II.C.2)** Quelle est la complexité temporelle de l'algorithme du plus proche voisin en considérant que cet algorithme est constitué des deux fonctions `calculer_distances` et `plus_proche_voisin` ?

**II.C.3)** En considérant les trois points de coordonnées (0, 0), (0, 3000), (0, 7000) et en choisissant un point de départ adéquat pour le robot, montrer que l'algorithme du plus proche voisin ne fournit pas nécessairement le plus court chemin.

Dans la pratique, on constate que, dès que le nombre de points d'intérêt devient important, l'algorithme du plus proche voisin fournit un chemin qui peut être 50% plus long que le plus court chemin.

### III Deuxième approche : algorithme génétique

Les algorithmes génétiques s'inspirent de la théorie de l'évolution en simulant l'évolution d'une population. Ils font intervenir cinq traitements.

#### 1. Initialisation

Il s'agit de créer une population d'origine composée de  $m$  individus (ici des chemins pour l'exploration à planifier). Généralement la population de départ est produite aléatoirement.

#### 2. Évaluation

Cette étape consiste à attribuer à chaque individu de la population courante une note correspondant à sa capacité à répondre au problème posé. Ici la note sera simplement la longueur du chemin.

#### 3. Sélection

Une fois tous les individus évalués, l'algorithme ne conserve que les « meilleurs » individus. Plusieurs méthodes de sélection sont possibles : choix aléatoire, ceux qui ont obtenu la meilleure note, élimination par tournoi, etc.

#### 4. Croisement

Les individus sélectionnés sont croisés deux à deux pour produire de nouveaux individus et donc une nouvelle population. La fonction de croisement (ou reproduction) dépend de la nature des individus.

#### 5. Mutation

Une proportion d'individus est choisie (généralement aléatoirement) pour subir une mutation, c'est-à-dire une transformation aléatoire. Cette étape permet d'éviter à l'algorithme de rester bloqué sur un optimum local.

En répétant les étapes de sélection, croisement et mutation, l'algorithme fait ainsi évoluer la population, jusqu'à trouver un individu qui réponde au problème initial. Cependant dans les cas pratiques d'utilisation des algorithmes génétiques, il n'est pas possible de savoir simplement si le problème est résolu (le plus court chemin figure-t-il dans ma population ?). On utilise donc des conditions d'arrêt heuristiques basées sur un critère arbitraire.

Le but de cette partie est de construire un algorithme génétique pour rechercher un meilleur chemin d'exploration que celui obtenu par l'algorithme du plus proche voisin.

#### III.A – Initialisation et évaluation

Une population est représentée par une liste d'individus, chaque individu étant représenté par un couple (*longueur, chemin*) dans lequel

- *chemin* désigne un chemin représenté comme précédemment par une liste d'entiers correspondant aux indices des points d'intérêt dans le tableau des distances produit par la fonction `calculer_distances` ;
- *longueur* est un entier correspondant à la longueur du chemin, en tenant compte de la position de départ du robot.

Écrire une fonction d'entête

```
def créer_population(m:int, d:np.ndarray) -> list:
```

qui crée une population de  $m$  individus aléatoires. Cette fonction prend en paramètre le nombre d'individus à engendrer et le tableau des distances entre points d'intérêt (et la position courante du robot) tel que produit par la fonction `calculer_distances`. Elle renvoie une liste d'individus, c'est-à-dire de couples (*longueur, chemin*).

#### III.B – Sélection

Écrire une fonction d'entête

```
def réduire(p:list) -> None:
```

qui réduit une population de moitié en ne conservant que les individus correspondant aux chemins les plus courts. On rappelle que *p* est une liste de couples (*longueur, chemin*). La fonction `réduire` ne renvoie pas de résultat mais modifie la liste passée en paramètre.

***III.C – Mutation*****III.C.1)** Écrire une fonction d'entête

```
def muter_chemin(c:list) -> None:
```

qui prend en paramètre un chemin et le transforme en inversant aléatoirement deux de ses éléments.

**III.C.2)** Écrire une fonction d'entête

```
def muter_population(p:list, proba:float, d:np.ndarray) -> None:
```

qui prend en paramètre une population dont elle fait muter un certain nombre d'individus. Le paramètre `proba` (compris entre 0 et 1) désigne la probabilité de mutation d'un individu. Le paramètre `d` est la matrice des distances entre points d'intérêt.

***III.D – Croisement*****III.D.1)** Écrire une fonction d'entête

```
def croiser(c1:list, c2:list) -> list:
```

qui crée un nouveau chemin à partir de deux chemins passés en paramètre. Ce nouveau chemin sera produit en prenant la première moitié du premier chemin suivi de la deuxième moitié du deuxième puis en « normalisant » le chemin ainsi obtenu.

**III.D.2)** Écrire une fonction d'entête

```
def nouvelle_génération(p:list, d:np.ndarray) -> None:
```

qui fait grossir une population en croisant ses membres pour en doubler l'effectif. Pour cela, la fonction fait se reproduire tous les couples d'individus qui se suivent dans la population (`p[i]`, `p[i+1]`) et (`p[m-1]`, `p[0]`) de façon à produire  $m$  nouveaux individus qui s'ajoutent aux  $m$  individus de la population de départ.

***III.E – Algorithme complet*****III.E.1)** Écrire une fonction d'entête

```
def algo_génétique(PI:np.ndarray, m:int, proba:float, g:int) -> float, list:
```

qui prend en paramètre un tableau de points d'intérêts (figure 2), la taille  $m$  de la population, la probabilité de mutation `proba` et le nombre de générations  $g$ . Cette fonction implante un algorithme génétique à l'aide des différentes fonctions écrites jusqu'à présent et renvoie la longueur du plus court chemin d'exploration et le chemin lui-même obtenus au bout de  $g$  générations.

**III.E.2)** Est-il possible avec l'implantation réalisée, qu'une itération de l'algorithme dégrade le résultat : le meilleur chemin obtenu à la génération  $n + 1$  est plus long que celui de la génération  $n$  ?

Dans l'affirmative, comment modifier le programme pour que cette situation ne puisse plus arriver ?

**III.E.3)** Quelles autres conditions d'arrêt peut-on imaginer ? Établir un comparatif présentant les avantages et inconvénients de chaque condition d'arrêt envisagée.

## Opérations et fonctions Python disponibles

***Fonctions***

- `range(n)` renvoie la séquence des  $n$  premiers entiers ( $0 \rightarrow n - 1$ )
- `list(range(n))` renvoie une liste contenant les  $n$  premiers entiers dans l'ordre croissant :  
`list(range(5)) → [0, 1, 2, 3, 4]`
- `random.randrange(a, b)` renvoie un entier aléatoire compris entre `a` et `b-1` inclus (`a` et `b` entiers)
- `random.random()` renvoie un nombre flottant tiré aléatoirement dans  $[0, 1[$  suivant une distribution uniforme
- `random.shuffle(u)` permute aléatoirement les éléments de la liste `u` (modifie `u`)
- `random.sample(u, n)` renvoie une liste de  $n$  éléments distincts de la liste `u` choisis aléatoirement, si  $n > \text{len}(u)$ , déclenche l'exception `ValueError`
- `math.sqrt(x)` calcule la racine carrée du nombre  $x$
- `math.ceil(x)` renvoie le plus petit entier supérieur ou égal à  $x$

- `math.floor(x)` renvoie le plus grand entier inférieur ou égal à `x`
- `sorted(u)` renvoie une nouvelle liste contenant les éléments de la liste `u` triés dans l'ordre « naturel » de ses éléments (si les éléments de `u` sont des listes ou des tuples, l'ordre utilisé est l'ordre lexicographique)

#### ***Opérations sur les listes***

- `len(u)` donne le nombre d'éléments de la liste `u` :  
`len([1, 2, 3]) → 3 ; len([[1,2], [3,4]]) → 2`
- `u + v` construit une liste constituée de la concaténation des listes `u` et `v` :  
`[1, 2] + [3, 4, 5] → [1, 2, 3, 4, 5]`
- `n * u` construit une liste constitué de la liste `u` concaténée `n` fois avec elle-même :  
`3 * [1, 2] → [1, 2, 1, 2, 1, 2]`
- `e in u` et `e not in u` déterminent si l'objet `e` figure dans la liste `u`, cette opération a une complexité temporelle en  $O(\text{len}(u))$   
`2 in [1, 2, 3] → True ; 2 not in [1, 2, 3] → False`
- `u.append(e)` ajoute l'élément `e` à la fin de la liste `u` (similaire à `u = u + [e]`)
- `del u[i]` supprime de la liste `u` son élément d'indice `i`
- `del u[i:j]` supprime de la liste `u` tous ses éléments dont les indices sont compris dans l'intervalle `[i, j[`
- `u.remove(e)` supprime de la liste `u` le premier élément qui a pour valeur `e`, déclenche l'exception `ValueError` si `e` ne figure pas dans `u`, cette opération a une complexité temporelle en  $O(\text{len}(u))$
- `u.insert(i, e)` insère l'élément `e` à la position d'indice `i` dans la liste `u` (en décalant les éléments suivants) ; si `i >= len(u)`, `e` est ajouté en fin de liste
- `u[i], u[j] = u[j], u[i]` permute les éléments d'indice `i` et `j` dans la liste `u`
- `u.sort()` trie la liste `u` en place, dans l'ordre « naturel » de ses éléments (si les éléments de `u` sont des listes ou des tuples, l'ordre utilisé est l'ordre lexicographique)

#### ***Opérations sur les tableaux (np.ndarray)***

- `np.array(u)` crée un nouveau tableau contenant les éléments de la liste `u`. La taille et le type des éléments de ce tableau sont déduits du contenu de `u`
- `np.empty(n, dtype)`, `np.empty((n, m), dtype)` crée respectivement un vecteur à `n` éléments ou une matrice à `n` lignes et `m` colonnes dont les éléments, de valeurs indéterminées, sont de type `dtype` qui peut être un type standard (`bool`, `int`, `float`, ...) ou un type spécifique numpy (`np.int16`, `np.float32`, ...). Si le paramètre `dtype` n'est pas précisé, il prend la valeur `float` par défaut
- `np.zeros(n, dtype)`, `np.zeros((n, m), dtype)` fonctionne comme `np.empty` en initialisant chaque élément à la valeur zéro pour les types numériques ou `False` pour les types booléens
- `a.ndim` nombre de dimensions du tableau `a` (1 pour un vecteur, 2 pour une matrice, etc.)
- `a.shape` tuple donnant la taille du tableau `a` pour chacune de ses dimensions
- `len(a)` taille du tableau `a` dans sa première dimension (nombre d'éléments d'un vecteur, nombre de lignes d'une matrice, etc.) équivalent à `a.shape[0]`
- `a.size` nombre total d'éléments du tableau `a`
- `a.flat` itérateur sur tous les éléments du tableau `a`
- `a.min()`, `a.max()` renvoie la valeur du plus petit (respectivement plus grand) élément du tableau `a` ; ces opérations ont une complexité temporelle en  $O(a.size)$
- `b in a` détermine si `b` est un élément du tableau `a` ; si `b` est un scalaire, vérifie si `b` est un élément de `a` ; si `b` est un vecteur ou une liste et `a` une matrice, détermine si `b` est une ligne de `a`
- `np.concatenate((a1, a2))` construit un nouveau tableau en concaténant deux tableaux ; `a1` et `a2` doivent avoir le même nombre de dimensions et la même taille à l'exception de leur taille dans la première dimension (deux matrices doivent avoir le même nombre de colonnes pour pouvoir être concaténées)

**EPRÉUVE SPECIFIQUE - FILIERE PSI**

---

**INFORMATIQUE****Vendredi 5 mai : 8 h - 11 h**

---

*N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.*

---

**Les calculatrices sont interdites**

Le sujet comporte 15 pages dont :

- 13 pages de texte de présentation et énoncé du sujet ;
- 2 pages d'annexes.

Toute documentation autre que celle fournie est interdite.

---

**REMARQUES PRÉLIMINAIRES**

---

L'épreuve doit être traitée en langage Python. Les syntaxes sont rappelées en **annexe 2**.

Les différents algorithmes doivent être rendus dans leur forme définitive sur la copie à rendre (les brouillons ne seront pas acceptés).

Il est demandé au candidat de bien vouloir rédiger ses réponses en **précisant bien le numéro de la question traitée et, si possible, dans l'ordre des questions**. La réponse ne doit pas se cantonner à la rédaction de l'algorithme sans explication ; les programmes doivent être expliqués et commentés.

# Étude de la capacité et de la congestion de l'autoroute A7

## I Objectifs et démarche d'étude

Il existe plusieurs formes de congestions routières, selon leur cause : la congestion récurrente, la congestion « prévisible » (travaux, manifestations, météo) et la congestion due aux incidents et accidents, par définition imprévisibles. On s'intéresse ici au niveau de congestion récurrente qui peut être défini comme le surplus de demande qui amène la congestion.

Cette étude se focalise sur la congestion routière de l'autoroute A7 en France. La section étudiée ne comporte ni entrée ni sortie. La longueur de l'axe est de 8,5 km et comporte 3 voies sauf au niveau de la dernière station (non étudiée ici). Un seul sens de circulation est étudié.

La cartographie présentée sur la **figure 1** donne l'implantation des différentes stations de mesure, de la station *M8A* à *M8P*, et des contrôles de sanction automatique (CSA). Ces stations de mesure font partie du système de recueil automatique des données (RAD) présent sur les autoroutes.

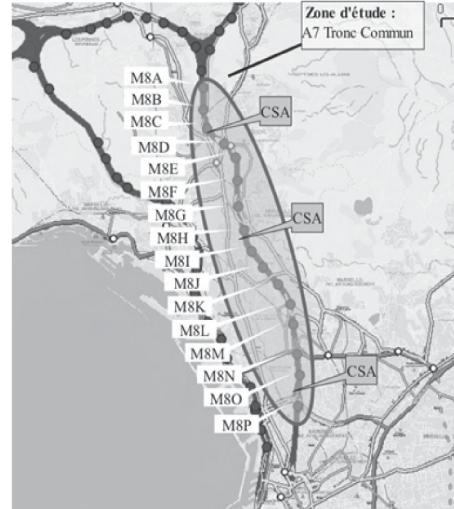
Pour réaliser les mesures, la chaussée est équipée de boucles électromagnétiques connectées aux stations qui remontent l'information vers un système central. Ces boucles permettent de compter le nombre de véhicules qui passent sur les routes et, dans le cadre de cette étude, il s'agit de boucles doubles qui permettent aussi d'estimer les vitesses.

La connaissance et le suivi du niveau de congestion récurrente permettent ensuite d'envisager des actions de régulation et d'aménagement de voirie. La simulation numérique est alors employée pour étudier différentes solutions d'aménagement ou proposer des solutions de régulation dynamique du trafic.

### Objectifs

*Le premier objectif est d'établir le diagramme fondamental (tracé du débit en fonction de la concentration) caractéristique du tronçon étudié à partir de l'historique de données de comptage.*

*Le deuxième objectif est de mettre en place une simulation numérique adaptée au tronçon étudié à partir de deux modèles afin de tester deux approches numériques différentes.*



**Figure 1 – Cartographie de la zone d'étude**

## II Traitement des données expérimentales

La méthodologie choisie ici pour estimer la capacité d'une route est celle utilisée notamment par les centres d'études techniques de l'équipement. Elle consiste à représenter un diagramme fondamental, c'est-à-dire le débit  $q$ , nombre de véhicules par unité de temps, en fonction de la concentration  $c$ , nombre de véhicules par unité de longueur. Ce diagramme permet alors de déterminer les paramètres caractéristiques de la route.

## II.1 Exploitation des données de mesure

### II.1.1 Sélection des mesures

L'ensemble des données produites par le réseau est archivé dans une base de données. Les variables d'intérêt moyennées par tranche de temps de 6 min pour chaque point de mesure sont le débit  $q\_exp$ , représentatif du nombre de véhicules par unité de temps et la vitesse moyenne  $v\_exp$  des véhicules en ce point. On peut également accéder à la concentration,  $c\_exp$ , par calcul étant donné que  $c\_exp = q\_exp/v\_exp$ .

Une version simplifiée de cette base de données est réduite à deux tables.

La table **STATIONS** répertorie les stations de mesures ; elle contient les attributs :

- $id\_station$  (clé primaire), entier identifiant chaque station ;
- $nom$ , chaîne de caractères désignant le nom de la station ;
- $nombre\_voies$ , entier donnant le nombre de voies de la section d'autoroute.

STATIONS	COMPTAGES
$id\_station$ $nom$ $nombre\_voies$	$id\_comptage$ $id\_station$ $date$ $voie$ $q\_exp$ $v\_exp$

La table **COMPTAGES** répertorie les différents enregistrements de données réalisés au cours du temps par les stations de comptage. Elle contient les attributs :

- $id\_comptage$ , entier identifiant chaque comptage ;
- $id\_station$ , entier identifiant la station concernée ;
- $date$ , entier datant la mesure ;
- $voie$ , entier numérotant la voie sur laquelle a été effectuée la mesure ;
- $q\_exp$ , flottant donnant le débit mesuré pendant 6 minutes ;
- $v\_exp$ , flottant donnant la vitesse moyenne mesurée pendant 6 minutes.

**Q1.** L'étude se focalise uniquement sur les mesures de l'une des stations, la *M8B*. Écrire une requête SQL qui renvoie les données de comptage ( $id\_comptage$ ,  $date$ ,  $voie$ ,  $q\_exp$ ,  $v\_exp$ ) mesurées à la station de comptage de nom *M8B*.

Le résultat de la requête précédente est stocké dans une nouvelle table **COMPTAGES\_M8B** à cinq colonnes ( $id\_comptage$ ,  $date$ ,  $voie$ ,  $q\_exp$ ,  $v\_exp$ ).

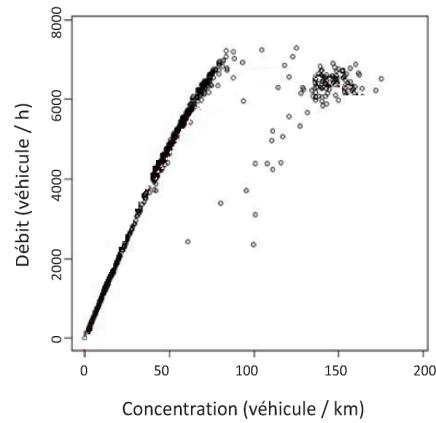
On fait l'hypothèse que les mesures sur les différentes voies d'une même station sont enregistrées de façon synchronisée. Lors d'un enregistrement pour une station à trois voies, on écrit donc trois lignes dans la table **COMPTAGES** avec trois dates identiques. Pour chacun des enregistrements de la station *M8B*, trois lignes avec trois dates identiques sont donc présentes dans la nouvelle table **COMPTAGES\_M8B**. Pour la suite de l'étude, les résultats expérimentaux de chacune des trois voies doivent être agrégés pour se ramener à une voie unique.

**Q2.** Écrire une requête SQL qui renvoie, pour chaque date des données de **COMPTAGES\_M8B**, le débit correspondant à la somme des débits de chaque voie.

De la même façon, une requête SQL permet d'obtenir la moyenne des vitesses sur l'ensemble des trois voies pour chaque date des données de **COMPTAGES\_M8B**. Il n'est pas demandé d'écrire cette requête. Ainsi, dans la suite de l'étude, la portion d'autoroute sera simplifiée en ne considérant qu'une seule voie.

### II.1.2 Diagramme fondamental

On veut tracer le diagramme fondamental du tronçon d'autoroute étudié (**figure 2**). Suite au traitement de la base de données, on dispose à présent du tableau à une dimension (aussi appelé vecteur) des débits  $q_{exp}$  (en véhicules par heure) et du vecteur des vitesses  $v_{exp}$  (en kilomètres par heure). Ces deux vecteurs possèdent  $nb_{mesures}$  composantes avec  $nb_{mesures}$  le nombre de points de mesure à tracer. Pour chaque composante  $i$ , la relation  $c_{exp}[i] = q_{exp}[i]/v_{exp}[i]$  permet d'obtenir la concentration. L'utilisation des tableaux à une dimension (ou vecteurs) est rappelée en **annexe 2**.



**Figure 2** – Points de mesure M8B : diagramme fondamental ( $c_{exp}, q_{exp}$ )

- Q3.** Écrire une fonction  $trace(q_{exp}, v_{exp})$  qui prend en arguments  $q_{exp}$  et  $v_{exp}$  et qui permet d'afficher le nuage de points du diagramme fondamental. On considérera que les bibliothèques sont importées et on pourra utiliser la fonction « `plot` » donnée en **annexe 2**.

## II.2 Estimation de l'état de congestion

Au niveau d'une station de mesure, la situation est dite congestionnée lorsque les vitesses prises par les véhicules restent inférieures à 40 km/h et la situation est dite fluide lorsque les vitesses restent supérieures à 80 km/h.

- Q4.** La fonction  $congestion(v_{exp})$ , qui prend en argument  $v_{exp}$  et renvoie la valeur médiane du tableau de valeurs  $v_{exp}$  est définie ci-dessous. La recherche de la médiane est basée sur un algorithme de tri. Choisir une des 4 propositions données pour compléter les 2 lignes manquantes (indiquées par « **ligne à compléter** »). Donner le nom, puis la complexité de l'algorithme de tri employé, dans le meilleur et le pire des cas. Analyser la pertinence de ce choix.

```
def congestion(v_exp):
    nbmesures=len(v_exp)
    for i in range(nbmesures):
        v=v_exp[i]
        j = i
        while 0 < j and v < v_exp[j-1]:
            ligne à compléter
            ligne à compléter
            v_exp[j] = v
    return v_exp[nbmesures//2]
```

Propositions pour les lignes manquantes :

1.  $v_{exp}[j-1] = v_{exp}[j]$   
 $j = j-1$
2.  $v_{exp}[j] = v_{exp}[j-1]$   
 $j = j-1$
3.  $v_{exp}[j+1] = v_{exp}[j]$   
 $j = j+1$
4.  $v_{exp}[j] = v_{exp}[j+1]$   
 $j = j+1$

- Q5.** À partir de la base de données de la station *M8B*, on obtient le vecteur  $v_{exp}$ . On exécute la fonction  $congestion(v_{exp})$ , la valeur renournée par la fonction étant 30, quelle conclusion peut-on tirer de ce résultat ?

### III Élaboration d'une première simulation du trafic routier par la mécanique des fluides

Il est rappelé ici que dans toute la suite de l'étude, l'autoroute est assimilée à une seule voie. Le modèle continu revient à négliger le caractère discret de la matière. Pour le modèle routier, cela revient donc à regarder l'évolution du trafic sur des distances grandes devant la taille des véhicules, notée  $L_0$ . On appelle  $c(t,x)$ , en véhicules par mètre, la concentration de véhicules par unité de longueur de route à l'instant  $t$  et à la position  $x$ . Sur une longueur  $L_0$ , il y a, au plus, un seul véhicule, soit  $c \leq c_{max} = \frac{1}{L_0}$ . On appelle  $q(t,x)$ , en véhicules par seconde, le débit de véhicules, c'est-à-dire le nombre de véhicules par unité de temps traversant la section de la route située à la position  $x$ . La vitesse  $v(t,x)$ , en mètres par seconde, ne représente pas la vitesse de chacun des véhicules mais la vitesse moyenne du trafic à la position  $x$ . On considère que les véhicules se déplacent selon l'axe  $x$  dans le sens des  $x$  croissants.

On rappelle la relation  $q(t,x) = c(t,x) \times v(t,x)$ . Désormais,  $q$ ,  $v$  et  $c$  représentent les grandeurs simulées et non plus des données expérimentales. On travaille dans la suite avec les unités du système international.

En considérant une portion d'autoroute  $dx$  pendant une durée  $dt$  et en supposant qu'il n'y a ni perte, ni création de véhicule, il est possible de montrer que  $q(t,x)$  et  $c(t,x)$  vérifient l'équation aux dérivées partielles suivante :

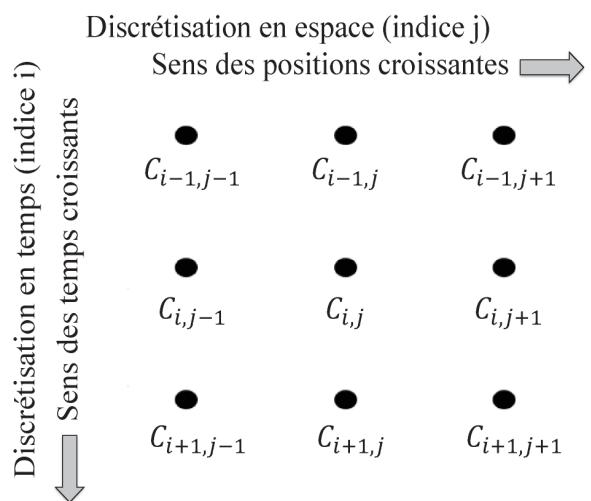
$$\frac{\partial q(t,x)}{\partial x} + \frac{\partial c(t,x)}{\partial t} = 0. \quad (1)$$

Pour comprendre comment évoluent la concentration, la vitesse moyenne ou le débit de véhicules au cours du temps le long de l'autoroute, il convient donc de résoudre cette équation aux dérivées partielles à partir de la situation initiale.

#### III.1 Discréétisation

Afin de résoudre numériquement cette équation aux dérivées partielles, nous avons besoin de la discréétiser. On choisit les paramètres suivants :

- longueur de l'autoroute :  $La$
- durée de simulation :  $Temps$
- pas d'espace (en mètres) :  $dx$
- pas de temps (en secondes) :  $dt$



**Figure 3 – Représentation de la discréétisation**

- Q6.** Soit  $C$  le tableau de valeurs contenant les concentrations en tous les points  $x$  et à tous les instants  $t$  discréétisés, comme représenté sur la **figure 3**. L'approximation numérique de la concentration au temps  $t_i$  et à la position  $x_j$  sera notée  $C_{i,j}$  avec  $i$  désignant l'indice de temps et  $j$  désignant l'indice d'espace. Quelles sont les dimensions de  $C$  ?

### III.2 Un modèle de diagramme fondamental

L'équation (1) possède deux inconnues. Il faut donc ajouter une deuxième équation pour pouvoir la résoudre. On propose tout d'abord de relier la vitesse et la concentration par le modèle de Greenshield établi à partir des analyses suivantes :

- lorsque la concentration en véhicules tend vers 0, les conducteurs peuvent rouler à la vitesse maximale autorisée,  $v\_max$  en mètres par seconde ;
- lorsque les véhicules sont pare-choc contre pare-choc, la concentration est égale à  $c\_max$  en véhicules par mètre : ils n'avancent plus.

Une relation linéaire entre vitesse et concentration est choisie dans le modèle de Greenshield, qui est ainsi défini par la relation suivante :

$$v(t,x) = v\_max(1 - c(t,x)/c\_max). \quad (2)$$

On souhaite concevoir une fonction *diagramme* permettant de réaliser le tracé du diagramme fondamental pour un instant donné  $t_i$  (soit pour une ligne de  $C$ ). Cette fonction fait appel à une fonction *debit* permettant de calculer les valeurs de débit à un instant  $t_i$  en utilisant la relation de Greenshield (2). Ces valeurs sont stockées dans un vecteur  $Q$ . Le tracé du diagramme fondamental est réalisé et le tableau  $Q$  est retourné.

- Q7.** Écrire une fonction *debit*( $v\_max$ ,  $c\_max$ ,  $C\_ligne$ ) qui prend en arguments la vitesse maximale ( $v\_max$ ), la concentration maximale ( $c\_max$ ) et un tableau contenant les concentrations à un instant donné (soit les éléments d'une ligne du tableau  $C$ ) nommé ici  $C\_ligne$  et qui renvoie un tableau de valeurs contenant les débits (en véhicules par seconde) aux différentes positions à ce même instant.
- Q8.** Spécifier les arguments d'entrée (et leur type) de la fonction *diagramme*. L'écriture du code de la fonction n'est pas demandée. Préciser les unités des différents termes. Tracer l'allure du diagramme fondamental obtenu. L'allure du diagramme dépend-elle du temps  $t_i$  auquel on se place (soit du choix de la ligne de  $C$ ) ?

### III.3 Résolution de l'équation

#### III.3.1 Situation initiale

On considère une situation de départ ( $t = 0$ ), comme indiqué sur la **figure 4**. On se place dans une configuration où l'on a un profil de concentration dont on veut étudier l'évolution. La concentration  $c_1$  la plus faible passe à  $c_2$ , plus forte, à la distance  $d_1$  et revient à  $c_1$  en  $d_2$ . Chacune des distances sera discrétisée. L'approximation numérique d'une distance correspondra à la division entière de la distance par le pas.

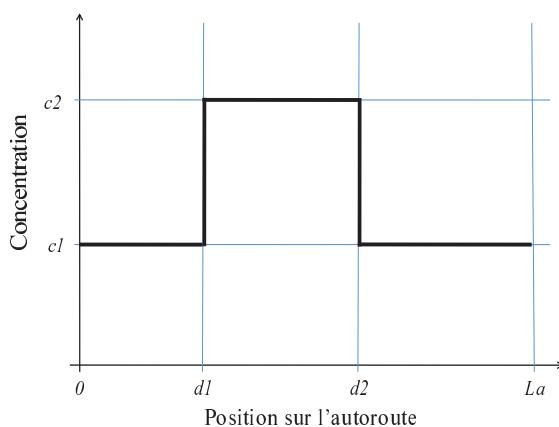


Figure 4 – Configuration initiale

- Q9.** Concevoir une fonction *C\_depart* qui permet d'initialiser la première ligne du tableau  $C$  (correspondant à  $t = 0$ ). L'écriture du code correspondant n'est pas demandée ; en revanche, on précisera toutes les valeurs de  $j$  pour lesquelles  $C_{0,j}$  sera égale à  $c_1$  et toutes les valeurs pour lesquelles  $C_{0,j}$  sera égale à  $c_2$ . L'en-tête ou spécification de la fonction devra être précisé(e) (et comporter les arguments d'entrée et leur type), ainsi que le résultat renvoyé par la fonction.

### III.3.2 Résolution

Le tableau  $C$  contient des zéros, exceptée la première ligne qui a été remplie de valeurs grâce à la mise en œuvre de la fonction  $C\_depart$ . On souhaite à présent écrire la fonction  $resolution(C, dt, dx, c\_max, v\_max)$  permettant de résoudre l'équation et de remplir complètement le tableau  $C$ .

Connaissant pour tout indice  $j$  les valeurs de  $C_{i,j}$ , on cherche à déterminer  $C_{i+1,j}$ .

Dans le schéma d'Euler « avant », la dérivée d'une fonction  $f$  par rapport à la variable  $x$ , au point  $x_j$ ,  $\frac{df}{dx}(x_j)$ , est approximée (en utilisant ce point et le point situé « devant » lui) par  $\frac{f_{j+1} - f_j}{dx}$ .

$Q$  est un vecteur contenant les valeurs de débits  $Q_j$  aux différentes positions  $x_j$  et à l'instant  $t_i$  (l'approximation du débit au temps  $t_i$  et à la position  $x_j$  sera donc notée  $Q_{j, \cdot}$ ). À chaque instant  $t_i$ ,  $Q$  devra être recalculé.

**Q10.** À partir de l'équation (1) et en utilisant des schémas d'Euler « avant » pour l'écriture des dérivées, montrer que la relation de récurrence donnant  $C_{i+1,j}$  en fonction de  $C_{i,j}$ ,  $Q_{j+1}$ ,  $Q_j$ ,  $dx$  et  $dt$  est donnée par l'une des propositions ci-dessous. Le numéro de la réponse correcte sera clairement indiqué sur la copie.

1.  $C_{i+1,j} = C_{i,j} - \frac{Q_j - Q_{j+1}}{dx}.dt$
2.  $C_{i+1,j} = C_{i,j} - \frac{Q_{j+1} - Q_j}{dx}.dt$
3.  $C_{i+1,j} = C_{i,j} - \frac{Q_{j+1} - Q_j}{dt}.dx$
4.  $C_{i+1,j} = C_{i,j} - \frac{Q_j - Q_{j-1}}{dx}.dt$

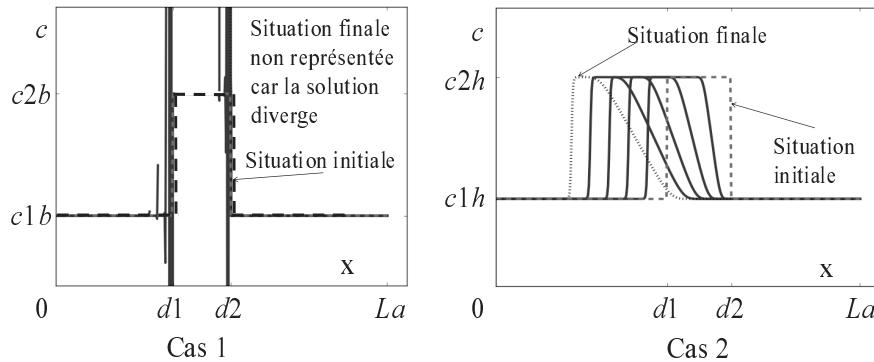
Pour que le nombre de voitures soit constant sur la longueur de la route, il faut se fixer des conditions aux limites périodiques. Ainsi, quand un véhicule arrive en bout d'autoroute, il est replacé au début de celle-ci. On considère donc que le véhicule situé en  $x = La$ , se déplaçant vers la droite, a pour voisin de droite le véhicule situé en  $x = 0$ .

**Q11.** L'initialisation a été effectuée avec la fonction  $C\_depart(dx, d1, d2, c1, c2, C)$ . Écrire une fonction  $resolution(C, dt, dx, c\_max, v\_max)$  qui prend en arguments le tableau  $C$ , les pas  $dt$  et  $dx$ , la concentration maximale  $c\_max$  et la valeur de la vitesse maximale  $v\_max$  et qui renvoie le tableau  $C$  rempli au cours de la résolution. On pourra faire appel à la fonction  $debit(v\_max, c\_max, C\_ligne)$  définie à la question **Q7**.

### III.4 Étude des solutions trouvées et modification du schéma

On étudie deux situations de départ basées sur le même profil que celui proposé à la situation initiale (**figure 4**). Dans le cas 1, on choisit  $c1$  et  $c2$ , respectivement notées  $c1b$  et  $c2b$ , correspondant à des concentrations faibles. On peut montrer que, pour ces concentrations, le créneau se déplace vers la droite (dans le sens des  $x$  croissants) au cours du temps. Cette démonstration n'est pas demandée. Dans le cas 2, on choisit  $c1$  et  $c2$ , respectivement notées  $c1h$  et  $c2h$ , correspondant à des concentrations fortes. On peut montrer que, pour ces concentrations, le créneau se déplace vers la gauche (dans le sens des  $x$  décroissants) au cours du temps. Cette démonstration n'est pas demandée.

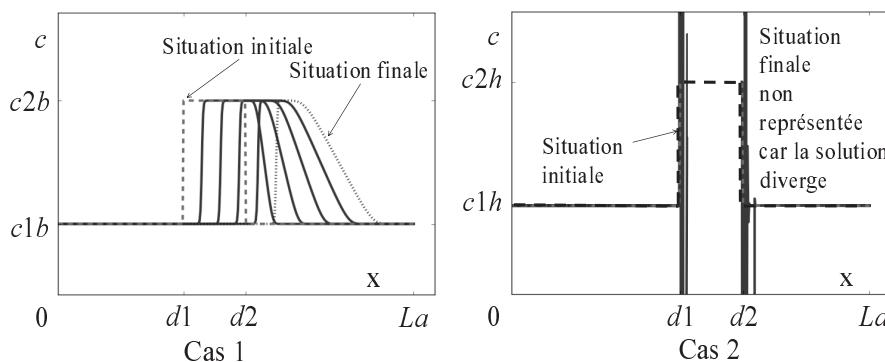
On applique la fonction *resolution* à ces deux situations de départ et on représente la concentration en fonction de la position à différents instants (**figure 5**). La situation initiale est en traits interrompus (- -), les situations intermédiaires en traits continus (-) et la situation finale en pointillés (:).



**Figure 5** – Résultats pour le cas 1 et pour le cas 2 avec Euler « avant » pour l'espace et « avant » pour le temps

On remarque qu'avec la situation de départ du cas 1, la solution diverge. Avec la situation de départ du cas 2, le créneau initial semble se déplacer vers la gauche.

Une modification est alors apportée à la fonction *resolution*. Dans la discréétisation en espace (c'est-à-dire lors de l'écriture des dérivées par rapport à la variable d'espace), le schéma d'Euler « avant » en espace est remplacé par un schéma d'Euler « arrière » en espace. Dans ce schéma, la dérivée d'une fonction  $f$  par rapport à la variable  $x$  au point  $x_j$ ,  $\frac{df}{dx}(x_j)$  est approximée (en utilisant ce point et le point situé « derrière » lui) par  $\frac{f_j - f_{j-1}}{dx}$ . On obtient alors les résultats présentés **figure 6**.



**Figure 6** – Résultats pour le cas 1 et pour le cas 2 avec Euler « arrière » pour l'espace et « avant » pour le temps

Cette fois-ci, avec la situation de départ du cas 1, le créneau initial se déplace vers la droite et avec la situation de départ du cas 2, la solution diverge.

**Q12.** On se place à l'itération  $i + 1$ ; les calculs des itérations précédentes ont déjà été réalisés.

Le calcul des termes de  $Q$  (vecteur des débits à l'instant  $t_i$ ) est fait par la fonction *debit*,  $Q_j$  étant déterminé à partir de la valeur de  $C_{i,j}$ . Sur la grille de discréétisation donnée **figure 3** (qui sera reproduite sur la copie), tracer des flèches partant des points déjà calculés aux itérations précédentes et allant vers le point à calculer  $C_{i+1,j}$  (au pas d'espace  $j$  et à l'itération  $i + 1$ ) dans le cas du schéma d'Euler « avant » pour la discréétisation en espace. Procéder de même dans le cas du schéma d'Euler « arrière » pour la discréétisation en espace.

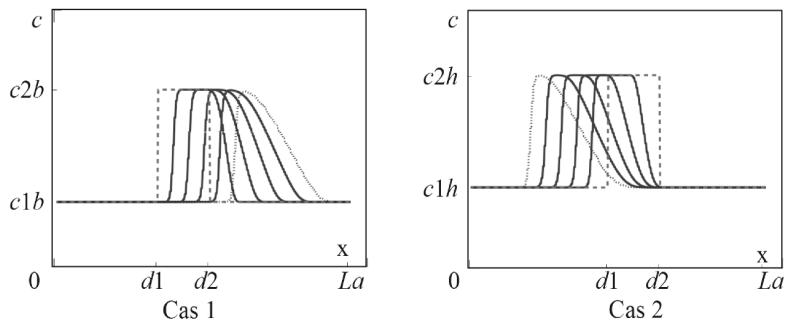
**Q13.** En déduire un argument permettant de choisir le schéma d'Euler adapté à la situation de départ.

On cherche maintenant un schéma fonctionnel pour les deux situations. On propose celui de Lax-Friedrichs qui :

- utilise un schéma centré pour l'approximation des dérivées par rapport à la variable d'espace.  
Dans ce schéma, la dérivée d'une fonction  $f$  par rapport à la variable  $x$  au point  $x_j$ ,  $\frac{df}{dx}(x_j)$  est approximée (à partir du point précédent et du point suivant) par  $\frac{f_{j+1} - f_{j-1}}{2dx}$ ;
- approxime les dérivées par rapport à la variable de temps en remplaçant la valeur  $C_{i,j}$  par la moyenne de la valeur prise au point précédent  $C_{i,j-1}$  et de la valeur prise au point suivant  $C_{i,j+1}$ .

**Q14.** Proposer les modifications de la fonction *resolution* (définie à la question **Q11**) nécessaires pour utiliser le schéma de Lax-Friedrichs.

La mise en œuvre de la fonction *resolution* permet, ensuite, d'effectuer le tracé des solutions obtenues dont le résultat est donné **figure 7**. Les instructions permettant de réaliser le tracé ne sont pas demandées.



**Figure 7 –** Les deux situations de départ résolues avec le schéma de Lax-Friedrichs

### III.5 Amélioration du programme : retour sur le choix du diagramme fondamental

Afin de s'assurer de la stabilité de la solution trouvée, il convient, une fois le schéma d'approximation déterminé, de définir les valeurs des paramètres tels que les pas de temps et d'espace. On suppose ici que ce travail a été réalisé. Cependant, cela ne signifie pas que l'on peut simuler fidèlement des phénomènes réels. En effet, le diagramme fondamental utilisé et tracé à la question **Q8** est assez différent du nuage de points expérimentaux représenté **figure 2**.

La nouvelle approximation choisie pour obtenir les paramètres caractéristiques du diagramme fondamental  $q$  en fonction de  $c$  est une régression d'ordre 3 :  $q = a_3 * c^3 + a_2 * c^2 + a_1 * c + a_0$ ,  $a_i$  étant les constantes d'ajustement de la courbe sur le nuage de  $n$  points. La fonction *regression(q\_exp, c\_exp)*, qui prend en arguments les vecteurs *q\_exp* et *c\_exp* obtenus expérimentalement (ayant servi à tracer le nuage de points de la **figure 2**) et qui renvoie les coefficients  $a_0, a_1, a_2, a_3$ , a été réalisée (on ne demande pas d'écrire cette fonction).

**Q15.** Expliquer en quelques phrases ce qu'il faut modifier dans la fonction *resolution* définie à la question **Q11** pour résoudre l'équation (1) en prenant en compte ce nouveau diagramme fondamental basé sur l'expérimentation.

La simulation que nous avons mise en place permet ainsi de déterminer les caractéristiques d'évolution d'un embouteillage. On peut notamment en déduire la vitesse de propagation de l'embouteillage, ou sa vitesse de résorption. Cependant, on aimerait à présent mettre en place une simulation permettant de modéliser les comportements individuels des conducteurs.

## IV Deuxième simulation du trafic routier : simulation de Nagel et Schreckenberg (NaSch)

L'objectif est de simuler la formation d'embouteillages dits « embouteillages fantômes ». Ils sont le résultat d'une perturbation qui apparaît localement sur la voie et s'amplifie peu à peu jusqu'à former un embouteillage.

### IV.1 Initialisation

Afin de modéliser le comportement de chacun des véhicules, l'espace, le temps ainsi que la vitesse des véhicules sont discrétilisés. La dynamique de chaque élément est modélisée de façon très simple, l'objectif étant d'obtenir un bon comportement à l'échelle macroscopique. On étudie, comme dans les parties précédentes, une autoroute de longueur  $La = 8\ 500$  (en mètres) pour laquelle on ne considère qu'un seul sens et qu'une seule voie. La vitesse est limitée à 130 km/h et on considère une durée totale d'étude  $Temps$ .

Soit  $X_n$  la position du véhicule  $n$ ,  $v_n \in \llbracket 0, 1, \dots, v\_max \rrbracket$  sa vitesse entière et  $d_n$  la distance inter-véhiculaire (par rapport au véhicule précédent). On choisit de découper la route en  $nb\_cellules$  cellules de tailles identiques valant  $pas\_x = 7,5$  (en mètres). La durée d'étude est découpée en  $nb\_temps$  pas de temps valant  $pas\_t = 1,2$  (en secondes) qui peut s'interpréter comme le temps de réaction du conducteur. Une probabilité donnée par le flottant  $p$  est utilisée dans l'algorithme.

**Q16.** Justifier le choix de la valeur du pas d'espace. En déduire ce que vaut la vitesse  $v\_max$  imposée de 130 km/h en cellules par pas de temps. On arrondira au nombre entier supérieur.

Comme précédemment, la portion d'autoroute considérée est sans entrée ni sortie. Les conditions limites seront périodiques, c'est-à-dire qu'un véhicule sortant de l'autoroute se retrouve instantanément à l'entrée de celle-ci (avec la même vitesse).

On considère que les données du problème  $pas\_x$ ,  $pas\_t$ ,  $Temps$ ,  $La$ ,  $v\_max$ ,  $p$  sont, à présent, renseignées dans le programme. On cherche, pour une densité donnée, à déterminer les vitesses à chaque position au cours du temps, ainsi que l'occupation ou non de chacune des cellules. On met en place une structure de stockage constituée :

- d'un tableau *Route* de dimension  $nb\_temps \times nb\_cellules$  qui contient des nombres binaires. Si, au pas de temps 10, la cellule 23 est occupée par un véhicule, alors  $Route[10,23] = 1$ , sinon  $Route[10,23] = 0$  ;
- d'un tableau *Vitesses* de dimension  $nb\_temps \times nb\_cellules$  qui contient des entiers. Si, au pas de temps 10, la cellule 23 est occupée par un véhicule et que sa vitesse est de 3 cellules par pas de temps, alors  $Vitesses[10,23] = 3$ . Si la cellule n'est pas occupée, alors  $Vitesses[10,23] = 0$  ;
- d'un tableau *Vitesses\_suivantes* de dimension  $1 \times nb\_cellules$  qui contient des entiers. Il permettra de stocker les vitesses au temps  $i + 1$  avant de déplacer les véhicules.

On peut désormais définir la situation initiale. On considère une route où les écarts entre tous les véhicules sont identiques. On crée ainsi une route avec une répartition homogène des véhicules. La route a été initialisée en plaçant des 1 dans les cellules possédant un véhicule dans la première ligne de *Route* pour une concentration fixée  $c_0$  en véhicules par kilomètre. Toutes les autres cellules de *Route* sont initialisées avec la valeur 0. Les cellules de *Vitesses* sont également initialisées avec des zéros. Cette étape d'initialisation est considérée comme déjà réalisée dans la suite.

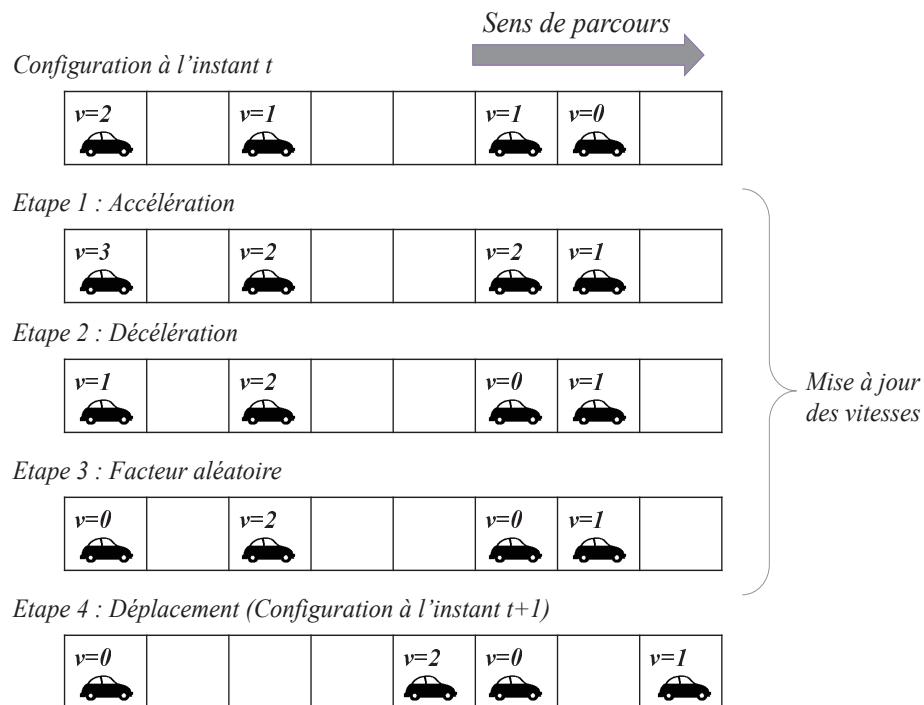
## IV.2 Mise en œuvre de l'algorithme

Le modèle NaSch, dont le pseudo-code est donné en **annexe 1**, est le pionnier des modèles cellulaires permettant de simuler un trafic routier. L'algorithme est constitué de quatre étapes qui sont toutes réalisées l'une après l'autre à chaque pas de temps.

Les étapes 1, 2 et 3 correspondent au calcul des futures vitesses. L'étape 4 permet d'inscrire chaque vitesse au lieu où se trouve le véhicule au pas suivant. Le comportement correspondant est illustré **figure 8**. Dans cet algorithme, on effectue la mise à jour des positions de tous les véhicules de façon simultanée.

Au niveau des étapes 2 et 4, des comparaisons et calculs sont effectués entre  $d_n$  et  $v_n$  ou entre  $X_n$  et  $v_n$ . En effet, les vitesses sont exprimées en cellules par pas de temps. Sur un pas de temps,  $v_n$  correspond donc à une distance parcourue en nombre de cellules.

On cherche à écrire la fonction *maj* qui permet d'appliquer les étapes 1, 2 et 3 de l'algorithme de NaSch pour toutes les valeurs de *Vitesses*[*i*, :]. On utilisera la fonction *distance*(*Route*, *i*, *j*) qui prend en arguments le tableau *Route*, l'indice de temps *i* et l'indice de position *j* du véhicule *n* et qui renvoie la distance  $d_n$  entre les véhicules *n* + 1 et *n*, à l'instant *i*. Seules les cellules comprenant un véhicule doivent être traitées ; les autres auront une vitesse nulle.



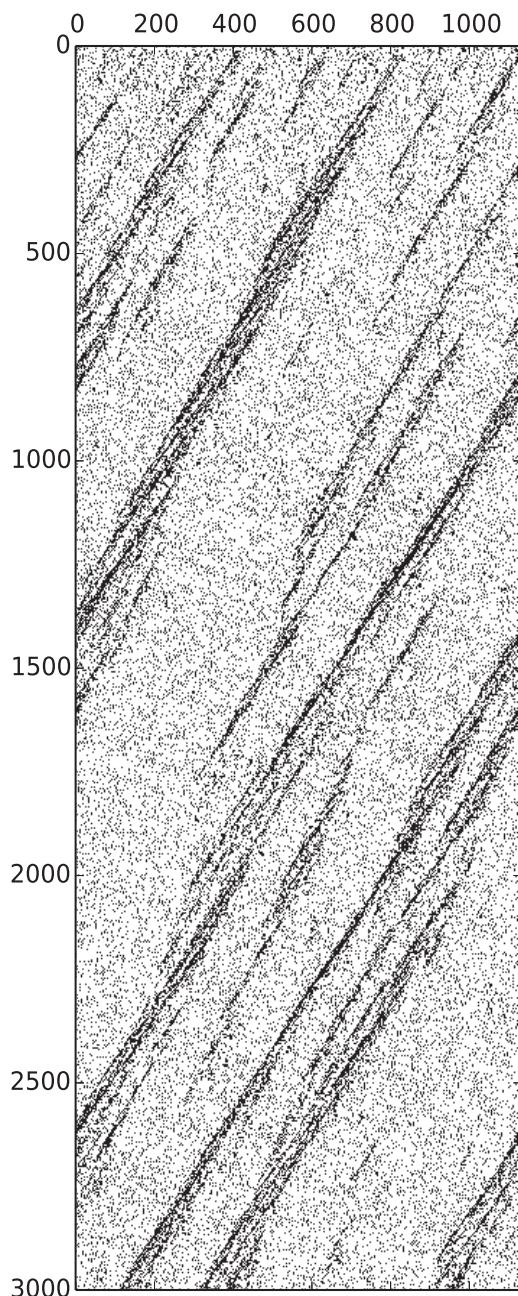
**Figure 8 – Illustration du schéma sur un exemple**

- Q17.** En utilisant l'**annexe 1**, écrire une fonction *maj*(*Route*, *Vitesses*, *p*, *vmax*, *i*) qui prend en arguments les tableaux *Route* et *Vitesses*, le paramètre aléatoire *p*, la vitesse maximale *v\_max* (en cellules par pas de temps) et l'indice de temps *i* et qui renvoie le tableau *Vitesses\_suivantes*.

- Q18.** Écrire une fonction  $deplacement(Vitesses, Route, Vitesses\_suivantes, i)$  qui permet de déterminer les valeurs de  $Vitesses[i + 1, :]$  et de  $Route[i + 1, :]$ . La fonction  $deplacement$  renvoie les tableaux  $Route$  et  $Vitesses$  mis à jour avec la ligne  $i + 1$  complétée. Les vitesses calculées doivent être placées dans les cellules où se trouvent les voitures correspondantes une fois déplacées. Penser à intégrer la prise en compte des conditions aux limites.

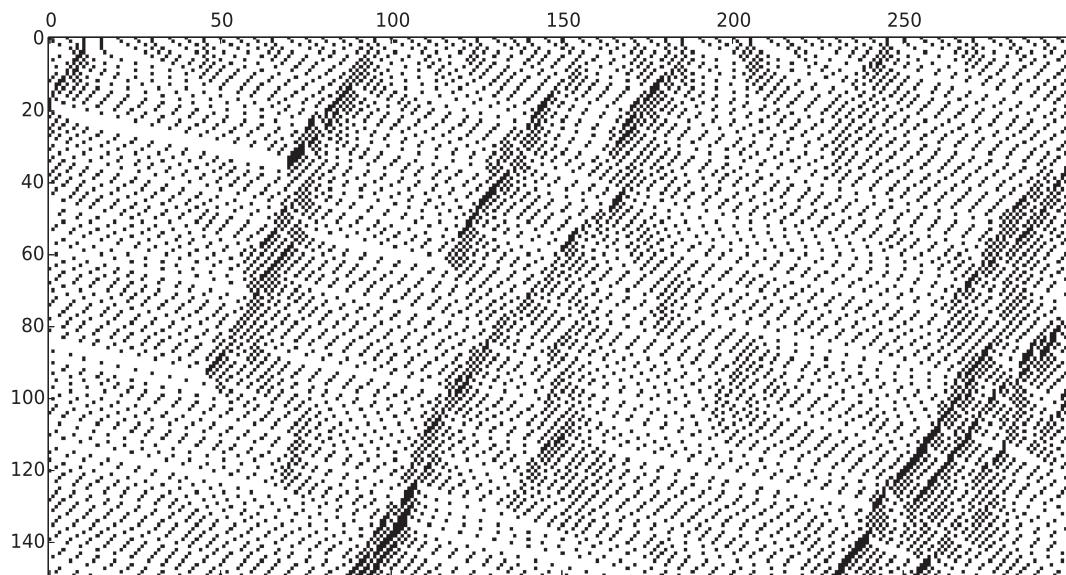
### IV.3 Simulation et analyse des résultats

Après  $nb\_temps$  itérations sur le temps, on obtient l'évolution de l'état de  $Route$  au cours du temps représentée **figure 9**. Les points noirs correspondent aux valeurs 1 du tableau (le reste étant à 0).



**Figure 9 – Affichage de  $Route$  (indice i de temps en ordonnée, indice j des positions en abscisse)**

Un zoom est réalisé **figure 10** sur les premières valeurs en temps et en espace pour mieux visualiser la formation d'embouteillages.



**Figure 10** – Zoom sur l'affichage de *Route* (indice i des temps en ordonnée, indice j des positions en abscisse)

- Q19.** Expliquer en quelques phrases en quoi les figures présentées montrent que la formation d'un embouteillage a été simulée. Sur quels paramètres peut-on agir pour que le résultat de la simulation se rapproche des résultats expérimentaux ?

## Annexes

### Annexe 1 - Algorithme de NaSch

Les opérations suivantes sont réalisées à chaque pas de temps.

- **Étape 1 - Accélération**

Le véhicule  $n$  accélère d'une unité s'il n'a pas encore atteint la vitesse maximum.

$$v_n(t+1) \rightarrow \min(v_n(t) + 1, v_{max})$$

- **Étape 2 - Décélération**

Le véhicule  $n$  décélère si la distance  $d_n = X_{n+1} - X_n$  par rapport au véhicule précédent ne permet pas de maintenir la vitesse  $v_n$ .

$$v_n(t+1) \rightarrow \min(v_n(t+1), d_n - 1)$$

- **Étape 3 - Facteur aléatoire**

Pour caractériser le comportement aléatoire de chacun des conducteurs, le véhicule  $n$  décélère avec une probabilité  $p$ , mais la vitesse n'est pas modifiée si  $v_n(t+1) = 0$  (pas de marche arrière).

On utilise pour cela la fonction `rand()` qui renvoie une valeur aléatoire  $\in [0..1]$ .

$$\text{Si } rand() < p \text{ alors } v_n(t+1) \rightarrow \max(v_n(t+1) - 1, 0)$$

- **Étape 4 - Déplacement**

Une fois les vitesses déterminées, les positions des véhicules au pas de temps suivant sont déterminées de façon simultanée.

$$X_n(t+1) \rightarrow X_n(t) + v_n(t+1)$$

## Annexe 2 - Rappels des syntaxes en Python

**Remarque :** sous Python, l'import du module numpy permet de réaliser des opérations pratiques sur les tableaux : `from numpy import *`. Les indices de ces tableaux commencent à 0.

	Python
tableau à une dimension	<code>L=[1, 2, 3]</code> (liste) <code>v=array([1, 2, 3])</code> (vecteur)
accéder à un élément	<code>v[0]</code> renvoie 1
ajouter un élément	<code>L.append(5)</code> uniquement sur les listes
tableau à deux dimensions (matrice)	<code>M=array(([1, 2, 3], [3, 4, 5]))</code>
accéder à un élément	<code>M[1, 2]</code> ou <code>M[1][2]</code> donne 5
extraire une portion de tableau (2 premières colonnes)	<code>M[:, 0:2]</code>
tableau de 0 (2 lignes, 3 colonnes)	<code>zeros((2, 3))</code>
dimension d'un tableau T de taille $(i, j)$	<code>T.shape</code> donne $[i, j]$
séquence équirépartie quelconque de 0 à 10.1 (exclus) par pas de 0.1	<code>arange(0, 10.1, 0.1)</code>
définir une chaîne de caractères	<code>mot="Python"</code>
taille d'une chaîne	<code>len(mot)</code>
extraire des caractères	<code>mot[2:7]</code>
boucle For	<code>for i in range(10):     print ( i )</code>
condition If	<code>if (i&gt;3):     print (i) else:     print ("hello")</code>
définir une fonction qui possède un argument et renvoie 2 résultats	<code>def fonction(param):     res1=param     res2=param*param     return res1, res2</code>
tracé de points (o) de coordonnées (x,y)	<code>plot(x, y, "o")</code>

**FIN**

**CONCOURS COMMUNS  
POLYTECHNIQUES****EPREUVE SPECIFIQUE - FILIERE TSI**

---

**INFORMATIQUE****Jeudi 4 mai : 14 h - 17 h**

---

*N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.*

---

**Les calculatrices sont interdites**

L'épreuve est composée de deux dossiers :

- un premier dossier de 16 pages contenant le sujet (pages numérotées de 1/16 à 14/16) et les annexes (pages numérotées de 15/16 à 16/16) ;
- un second dossier de 8 pages constituant le document réponse (DR) à rendre en fin d'épreuve (pages numérotées de DR – 1/8 à DR – 8/8).

Les consignes permettant de compléter le document réponse (DR) sont données à la page suivante.

**Seul le document réponse est à rendre.  
Chaque partie ou sous-partie de ce sujet est indépendante.**

## Remarques générales

Les réponses aux questions sont à rédiger sur le document réponse (DR) et ne doivent pas dépasser les dimensions des cadres proposés.

Si la réponse attendue est spécifique à un langage, la réponse doit être proposée en langage Python.

Les structures algorithmiques doivent être clairement identifiables par des indentations visibles ou par des barres droites entre le début et la fin de la structure comme proposé ci-dessous :

---

```
Si (Condition)
  Alors
    | Instructions
  Sinon
    | Instructions
Fin Si
```

---

# CONCEPTION D'UNE APPLICATION SPORTIVE

## “RUGBY MANAGER”

L'application « Rugby Manager » est un jeu destiné aux smartphones et aux tablettes. Il permet de créer une équipe, de l'entraîner et de jouer avec elle. Il est possible de jouer seul (contre l'intelligence artificielle du logiciel) ou en mode multijoueurs. Enfin, l'application est utilisable en ligne ou en Bluetooth.

L'objectif est de programmer un certain nombre de fonctions qui seront utilisées pour créer cette application.

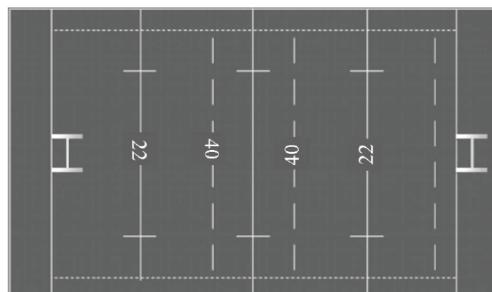
### I Étude des modes jeu et statistiques

#### I.1 Partie graphique du mode jeu

Dans cette sous-partie, nous souhaitons tout d'abord représenter la partie graphique du mode jeu, c'est-à-dire :

- un terrain ;
- deux équipes ;
- un ballon.

On souhaite afficher l'image du terrain de rugby suivant, enregistrée dans le répertoire “C:/CCP” sous le nom *stade.bmp*.



**Figure 1** - Image du terrain de rugby

#### Objectif

Les compétences évaluées dans cette partie portent sur le traitement des images. Il s'agit d'utiliser la bibliothèque d'images décrite en **Annexe 1** pour compléter ou modifier des fonctions d'affichage de l'image du terrain et des pixels des maillots des joueurs. Ensuite, il s'agit de mettre en place un “floutage” d'une image pour la mettre en arrière-plan dans le mode statistique.

**Q1.** En utilisant l'**Annexe 1**, page 15, compléter le programme du document réponse qui permet :

- de se placer dans le dossier où se trouve l'image ;
- d'ouvrir l'image et de la stocker dans une variable *image\_terrain* ;
- d'afficher l'image en arrière-plan.

Pour la suite et notamment pour l'affichage des joueurs à l'échelle du terrain, il est nécessaire de connaître les dimensions de l'image.

- Q2.** Donner les instructions qui permettent de faire cette opération et de stocker le résultat dans deux variables *dim\_long* et *dim\_larg* respectivement dimension en longueur et en largeur. Afficher ensuite le résultat sous la forme ("longueur × largeur").

On propose tout d'abord de représenter les joueurs par 4 pixels disposés en carré ayant la couleur du maillot de l'équipe. Le choix des couleurs des maillots doit être laissé libre à l'utilisateur. Cependant, le vert correspondant à la couleur du terrain et le blanc correspondant à la couleur du ballon et des lignes ne pourront pas être utilisés.

Précisons que les lignes font 2 pixels de large, sur une image qui en fait 620. Il est donc facile de se positionner au centre du terrain. Les lignes numérotées 40, en pointillés, sont situées à 50 pixels de la ligne centrale.

- Q3.** Créer une fonction *coul()* qui permet de connaître la couleur exacte du terrain et de la stocker dans une variable *coul\_ter*. L'argument d'entrée de la fonction est la variable *image*, la sortie est la variable *coul\_ter*. Attention : on choisira comme pixel de référence un des pixels le plus proche du centre du terrain.
- Q4.** Quel est le type de la variable *coul\_ter* ?  
Sur combien de bits mémoire est codé un pixel ?
- Q5.** Créer une fonction *maillot()* qui utilise la fonction précédente et qui renvoie la liste des couleurs interdites pour les maillots.

## I.2 Partie graphique du mode statistique

### Objectif

Dans cette sous-partie, nous nous intéressons au traitement d'image évolué : la mise en place du "floutage" d'une image pour l'arrière-plan du mode statistique.

On s'intéresse donc ici à un autre mode du jeu vidéo : le mode statistique. Dans ce mode, une image de jeu floue va être placée en arrière-plan. Nous allons étudier les fonctions permettant d'effectuer ce floutage.

Pour réaliser un floutage par moyenne simple sur la matrice de pixels, il faut lui appliquer un filtre, que l'on appelle également un masque. Afin de comprendre ce principe, nous proposons d'étudier un exemple de filtrage. On considère les matrices

$$A = \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$

et

$$B = \begin{pmatrix} 5 & 6 & 7 & 8 & 9 & 10 \\ -5 & -6 & -7 & -8 & -9 & -10 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 3 & 3 & 4 & 4 \\ 0 & 0 & 1 & 3 & 3 & 3 \end{pmatrix}.$$

Pour chaque élément  $b_{ij}$  de  $B$ , on considère la matrice carrée de taille 3, notée  $B_{ij}$  qui l'entoure et on calcule le produit de  $B_{ij}$  par  $A$  (multiplication classique  $A^*B_{ij}$ ).  $A$  est appelé noyau. On note  $c_{ij}$  la somme des coefficients de la matrice produit obtenue (on pourra utiliser la fonction `np.sum` sur une liste).

Si  $b_{ij}$  est un élément en bordure de  $B$ , on posera  $c_{ij} = b_{ij}$ . On forme ainsi une nouvelle matrice  $C$  dont les éléments intérieurs sont les  $c_{ij}$  (et les éléments au bord sont les  $b_{ij}$ ) :

- $c_{ij}$  = élément se trouvant ligne i et colonne j;
- $B_{ij}$  la matrice carrée dont l'élément central est  $b_{ij}$ .

Ainsi, par exemple

$$\begin{aligned} c_{22} &= \text{sum}(B_{22} * A) = \text{sum}\left(\text{dot}\left(\begin{pmatrix} 5 & 6 & 7 \\ -5 & -6 & -7 \\ 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}\right)\right) \\ &= \text{sum}\left(\begin{pmatrix} 2 & 2 & 2 \\ -2 & -2 & -2 \\ 1/3 & 1/3 & 1/3 \end{pmatrix}\right) = 1, \end{aligned}$$

`dot` et `sum` étant des fonctions du module numpy, avec `dot(B,A)` qui effectue le produit matriciel  $B * A$  (s'écrit aussi  $B \cdot \text{dot}(A)$ ) et `sum` qui somme tous les termes d'une matrice.

On dit que l'on a filtré la matrice  $B$  par la matrice  $A$ , ou que l'on a appliqué le masque (filtre)  $A$  sur l'image  $B$ .

- Q6.** Compléter la fonction `filtrer1(filtreA,matB)` qui prend en argument une matrice carrée `filtreA` de dimension `taille × taille` et une matrice quelconque `matB` de dimensions supérieures et qui renvoie la matrice  $C$ . Vous pouvez utiliser les fonctions du module numpy que vous jugez judicieuses sans préciser l'importation.

On souhaite maintenant appliquer le `filtreA` à une matrice de pixels  $B$ , c'est-à-dire aux 3 tableaux  $B[:, :, 0]$ ,  $B[:, :, 1]$ ,  $B[:, :, 2]$  et enregistrer le résultat dans une matrice  $C$  de même format que  $B$ .

- Q7.** Créer une fonction `filtrer(filtreA,matB)` qui prend en argument une matrice carrée `filtreA` de dimension `taille × taille` et un tableau numpy `matB` de dimensions  $n \times p \times 3$  et qui renvoie le tableau  $C$  de dimensions identiques. Vous pourrez calculer successivement  $C[:, :, 0]$ ,  $C[:, :, 1]$ ,  $C[:, :, 2]$  à l'aide d'une boucle.

Ces matrices réalisent un floutage par moyenne simple (coefficients tous égaux, de somme 1). Plus la taille du filtre est grande, plus le flou sera fort. On peut améliorer cette technique en utilisant un flou gaussien. Son principe est de calculer une moyenne pondérée en accordant plus de poids au pixel central et en diminuant le poids des pixels périphériques.

La matrice servant de filtre est calculée selon le modèle d'une courbe de Gauss (courbe en cloche) à 2 dimensions.

La fonction de Laplace-Gauss à une dimension est  $G(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$  et à deux dimensions est  $G(x,y) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{x^2+y^2}{2\sigma^2}}$ .

Plus l'écart-type  $\sigma$  est grand, plus l'image sera floutée. En pratique,  $\sigma$  et la taille (impaire) du filtre étant fixés, on construit la matrice filtre terme à terme, chaque élément de la matrice ayant

pour expression  $k \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$ , où  $x$  et  $y$  sont les nombres de lignes et de colonnes qui séparent cet élément du centre et  $k$  un coefficient constant, tel que la somme de tous les éléments (de type float) ainsi calculés soit égale à 1.

Par exemple, pour  $\sigma = 0,9$  et  $taille = 5$ , le filtre est proche de :

$$\frac{1}{1\,002} \begin{pmatrix} 1 & 9 & 17 & 9 & 1 \\ 9 & 58 & 107 & 58 & 9 \\ 17 & 107 & 198 & 107 & 17 \\ 9 & 58 & 107 & 58 & 9 \\ 1 & 9 & 17 & 9 & 1 \end{pmatrix}.$$

- Q8.** Compléter la fonction *matriceFlouGaussien(taille,sigma)* qui prend en argument la *taille* (impaire) de la matrice de floutage, *sigma* l'écart-type de déviation standard et qui retourne la matrice filtre correspondant au niveau gaussien. Encore une fois, vous pouvez utiliser les fonctions du module numpy que vous jugez judicieuses sans préciser l'importation.
  
- Q9.** Écrire ensuite la fonction *FloutageGaussien(tabPix,taille,sigma)* qui utilise la fonction *matriceFlouGaussien(taille,sigma)* et *filtrer(filtreA,matB)* et qui renvoie la matrice *tabPix* qui a été floutée grâce au filtre défini dans la fonction *matriceFlouGaussien*.

### I.3 Fonctionnalités du mode statistique

#### Objectif

Dans cette sous-partie, nous nous intéressons à certaines fonctions du mode statistique. Nous proposons tout d'abord de mettre en œuvre les fonctions permettant de tracer les statistiques de chaque joueur à l'aide d'histogrammes, puis d'écrire la fonction donnant les performances moyennes, minimales et maximales.

Nous allons ici étudier un des tableaux qui sera utilisé dans ce mode. La matrice *resultat* permet de stocker l'ensemble des données sur chaque joueur de l'équipe. Les lignes correspondent aux numéros des joueurs et les colonnes sont respectivement : le nombre d'essais, le nombre de passes réussies, le nombre de plaquages réussis, le nombre de plaquages manqués, le nombre de franchissements de ligne, le nombre de kilomètres parcourus.

Numéro du joueur	nombre d'essais	nombre de passes réussies	nombre de plaquages réussis	nombre de plaquages manqués	nombre de franchissements de ligne	nombre de kilomètres parcourus
1						
2						
...						

On souhaite représenter un histogramme de l'équipe pour chacune des colonnes de *resultat*. Il s'agit d'une représentation graphique présentant en abscisse le numéro de maillot du joueur et en ordonnée la donnée en question.

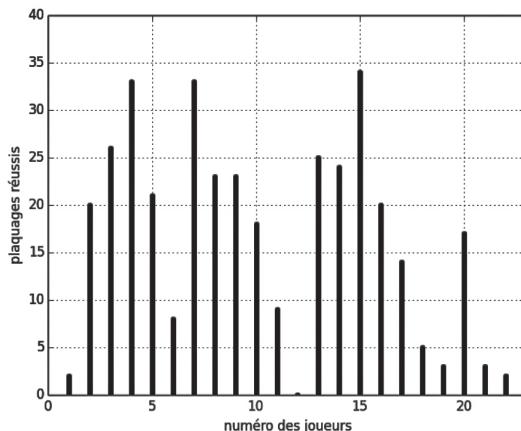
Nous allons étudier ici l'histogramme pour le nombre de plaquages réussis, soit la 4<sup>e</sup> colonne de *resultat*. L'affichage de l'histogramme ne sera pas étudié ici.

**Q10.** Écrire les instructions qui, à partir de la matrice *resultat*, construisent la liste des abscisses *x* de l'histogramme et la liste des ordonnées *y\_plaR*.

On souhaite maintenant créer l'histogramme. Pour cela, il faut créer simultanément deux listes à partir de *x* et *y\_plaR*:

- une liste *barre*, qui sera constituée, pour  $i \in [0, \text{len}(x)[$ , des valeurs allant de 0 à la valeur *y\_plaR[i]* incluse avec des pas de 0,1 ;
- une liste *abscisse*, qui, pour  $i \in [0, \text{len}(x)[$ , sera composée d'autant d'éléments que *barre*, tous égaux à *x[i]*.

L'instruction `plot(abscisse,barre,'k.',markersize=10)` donnera alors l'histogramme présenté sur la **figure 2**:



**Figure 2 - Histogramme**

**Q11.** Écrire les instructions qui, à partir de la matrice des listes `x` et `y_plaR`, construisent les listes `barre` et `abscisse`.

D'autres statistiques disponibles permettent de connaître le minimum, le maximum et la moyenne par match sur l'ensemble de la saison d'un joueur, sur la donnée qui nous intéresse. L'obtention de la liste de valeurs concernant une donnée en particulier n'est pas étudiée ici, seule la fonction qui renvoie le minimum, le maximum et la moyenne d'une liste de valeurs (float) est étudiée. Par exemple, pour 6 matchs, la liste des valeurs des plaquages réussis pour le joueur 15 est : `valeurs = [20,10,15,22,24]`.

**Q12.** Écrire une fonction `minMaxMoy(valeurs)`, qui prend en entrée une liste de valeurs flottantes et qui renvoie la liste contenant le minimum des valeurs de la liste, le maximum et la moyenne.

## II Traitement du transfert des joueurs

### Objectif

Cette partie du sujet s'intéresse au traitement de la base de données des joueurs. Dans le but de créer un jeu en ligne, il est nécessaire de prendre en compte les transferts de joueurs pour la création de sa propre équipe. Pour cela, on souhaite utiliser une base de données. L'objectif est d'analyser la structure de la base de données proposée et de réaliser les requêtes permettant de créer sa propre équipe.

On possède une base de données `rugby.db3` avec 2 tables : `Joueurs` et `Clubs`.

On souhaite modéliser les transferts des joueurs de club en club comme les transferts réels. L'application donne un certain budget à son utilisateur et en fonction de ses victoires, il pourra acheter des joueurs de plus en plus chers.

Nous allons étudier comment les tables de joueurs et leurs caractéristiques sont associées aux clubs et au salaire des joueurs. Le *Salaire* annuel sera composé de sommes en euros, alors que celle des *Budget* des Clubs sera en millions d'euros. Ces attributs représentent leur valeur financière.

<b>Joueurs</b>							
Nom	Poste	Age (an)	Poids (kg)	Taille (cm)	VMA (km/h)	Id_Club	Salaire euros
Murel Thibault	Pilier Gauche	18	105	187	13,6	102	16 000
Nkang Jesus	Pilier Droit	27	115	176	12,9	10	35 000
Yala Olive	Troisième ligne Aile	17	90	182	14,5	45	12 000
Ramis Thomas	Arrière	20	80	178	18,5	31	40 000

<b>Clubs</b>		
Id_Club	Nom	Budget total
1	Stade Français	50
2	Ulster	165
31	Stade Toulousain	150
45	Racing Club de Toulon	540

**Q13.** Donner la requête qui permet de trouver les joueurs de plus de 23 ans qui ont une Vitesse Maximale Aérobie (notée VMA) supérieure à 13 km/h.

**Q14.** Donner la requête qui permet de connaître les clubs dont les joueurs ont un salaire supérieur à 30 000 euros.

Puis, déterminer la requête qui permet de connaître le nombre de joueurs étant au Stade Toulousain et jouant au poste de Talonneur.

**Q15.** Donner la requête qui permet de calculer le rapport entre la masse salariale des joueurs d'un club (somme des salaires de tous les joueurs) et le budget total du club. Le résultat sera donné en pourcentage du budget total du club.

### III Évaluation des performances de l'équipe

#### Objectif

L'étape suivante permet de construire son équipe à partir de la liste des joueurs. Pour cela, il est important de pouvoir classer ses joueurs suivant ses préférences : le but est de trier la liste des joueurs issus de notre base de données de la manière la plus efficace pour l'utilisateur.

Dans cette partie, on s'intéresse donc au tri de la liste des joueurs de notre équipe en fonction de leurs performances.

Suite au transfert des joueurs dans notre équipe, nous avons créé une table *Equipe* avec les joueurs suivants :

Equipe					
Joueur	Poste	Age (an)	Poids (kg)	Taille (cm)	VMA (km/h)
Morellon Thibault	Pilier Gauche	18	105	187	13,6
Nkeno Jules	Pilier Droit	27	115	176	12,9
Yolozo Olivier	Troisième ligne Aile	17	90	182	14,5
Ramis Thomas	Arrière	20	80	178	18,5
Mechand Julien	Talonneur	20	105	181	13,6
Cazo Cyril	Deuxième ligne	20	115	197	12,9
Blue Richie	Deuxième ligne	26	112	206	15,5
Diverge Martin	Troisième ligne Aile	20	108	193	18,5
Dusaut Thierry	Troisième ligne Centre	33	100	188	13,6
Michal Pierre	Demi de mêlée/Demi d'ouverture	27	115	176	12,9
Dupont Antoine	Demi de mêlée	18	70	174	14,5
Choux Jonathan	Demi d'ouverture	30	92	188	18,5
Robot Jamie	Trois quart Centre	25	104	193	14,9
Totana Wesley	Trois quart Centre	27	93	182	18,5
Facke Gael	Trois quart Centre	21	90	190	17,5
Dupond Alex	Trois quart Aile	25	104	198	18,5
Wagon Anthony	Trois quart Aile	21	92	188	18,5
Cours Vincent	Trois quart Aile	34	80	178	18,5

Les instructions suivantes permettent d'afficher le tableau de l'équipe sous Python et de le stocker dans la variable *monequipe*.

```

1 import sqlite3 # Import des commandes permettant de manipuler la base de données
2 basesql = u"rugby.s3db" # Base de données initiale
3 cnx = sqlite3.connect(basesql )
4 curseur = cnx.cursor ()
5 requete = "SELECT * FROM Equipe"
6 curseur.execute(requete)
7 monequipe = curseur.fetchall()
8 print(monequipe)

```

**Q16.** Préciser le type de la variable *monequipe*.

**Q17.** Corriger les 2 erreurs de l'algorithme de tri suivant ainsi que l'appel de la fonction afin d'afficher les joueurs dans l'ordre de leur rapidité (rayer la ligne qui vous semble erronée et corriger à coté). La liste en entrée de *tri\_1* est une liste de la forme [[joueur1, poste1, age1, poids1, taille1, VMA1], [joueur2, poste2, age2, poids2, taille2, VMA2], ...] et le paramètre critère correspond à l'indice du critère qui doit être trié.

```

1 def echange(l,i,j):
2     """ echange 2 valeurs d'une liste """
3     l[i],l[j] = l[j], l[i]
4
5 def tri_1(liste,critere):
6     """ tri la liste en fonction du critere choisi """
7     for i in range(liste):
8         mini=i
9         for j in range(i+1,len(liste)):
10             if liste[j][critere] > liste[mini][critere]:
11                 mini=j
12             echange(liste,i,mini)
13     return liste
14
15 tri_1(monequipe,5)
16
17 print(monequipe)

```

**Q18.** En appelant *n* la taille de la *liste*, donner la complexité de la fonction *tri\_1( )* en fonction de *n* dans le pire des cas. Critiquer le résultat.

**Q19.** Commenter et expliquer chaque ligne d'instruction de la fonction *tri\_2( )* suivante (on ne commenterà pas *segmente*).

Quelle est la particularité de cette fonction *tri\_2( )*?

Modifier la fonction *tri\_2( )* afin qu'elle retourne comme résultat le nombre d'appels récursifs de la fonction *tri\_2( )*.

```

1 def segmente(T,val,i,j):
2     g=i+1
3     d=j
4     p=T[i][val]
5     while g<=d:
6         while d>=0 and T[d][val]>p:
7             d=d-1
8         while g<=j and T[g][val]<=p:
9             g=g+1
10        if g<d:
11            echange(T,g,d)
12            d=d-1
13            g=g+1
14        k=d
15    echange(T,i,d)
16    return k
17
18 def tri_2(L,val, i, j):
19     if i<j:
20         k=segmente(L,val,i,j)
21         tri_2(L,val,i,k-1)
22         tri_2(L,val,k+1,j)
23     return L

```

**Q20.** Donner alors l'instruction qui utilise cette fonction *tri\_2()* pour trier l'équipe en fonction du poids des joueurs.

Pour augmenter l'interactivité du jeu, il sera possible de rentrer ou de modifier des données de chaque joueur afin qu'elles correspondent à la réalité. Pour cela nous avons besoin de pouvoir insérer dans l'ordinateur des nombres à virgule flottante et de les convertir du décimal au binaire. Afin de ne pas gérer des formats de nombres trop grands, on choisit la norme IEE754 simple précision pour stocker les nombres à virgule flottante sur 32 bits (même si ceux-ci ne sont pas implémentés dans le langage Python).

Il n'est nul besoin d'avoir des connaissances de la norme IEE754 pour répondre aux différentes questions. Toutes les informations nécessaires pour répondre à une question seront explicitement données dans son énoncé.

Prenons un exemple pour bien comprendre le format de stockage.

**Q21.** Convertir le nombre décimal 74,25 en binaire.

**Q22.** Exprimer ce nombre au format IEE754 simple précision. On indique que les 32 bits du format IEE754 simple précision s'organisent comme suit : le bit de signe, puis l'exposant biaisé sur 8 bits (donc un biais de 127), puis la mantisse sur les 23 derniers bits.

La base de données contient 3 000 joueurs ayant chacun des caractéristiques de poids, taille et vitesse, qui peuvent être des nombres flottants.

**Q23.** Estimer la quantité de mémoire que représentent uniquement les données numériques.

Donner le résultat en *kilo octets* (on approximera  $1 \text{ ko} = 1\,000\text{o}$ ).

Quels sont les avantages et inconvénients du format simple précision pour la gestion des nombres flottants par rapport au format double précision ?

#### IV Entraînement à la passe vissée au rugby

##### Objectif

L'objectif de cette partie est de modéliser, puis de déterminer la trajectoire d'une balle pour l'insérer dans le mode entraînement du jeu.

Une des spécificités de cette application est qu'elle permet aussi un mode d'entraînement pour améliorer les passes. Afin de réaliser des passes réalistes, dites « vissées » au rugby, il faut modéliser l'ensemble des actions qui s'exercent sur le ballon et étudier sa dynamique.

L'un des phénomènes à modéliser est la résistance de l'air. Le ballon se déplace à sa vitesse maximale quand il vient d'être lâché par les mains, puis la résistance de l'air le ralentit immédiatement. Sa vitesse diminue jusqu'à ce qu'il atteigne sa hauteur maximale. Le ballon reprend ensuite de la vitesse quand il redescend.

Un autre phénomène physique intervient dans le comportement du ballon lors des passes : les effets donnés au ballon par les joueurs en utilisant différentes techniques de prise de ballon et de lâché. Le mouvement du ballon est alors modifié par l'effet Magnus, provoqué par la naissance, dans le sillage du ballon, de tourbillons capricieux appelés vortex.

On cherche à prendre en compte ces deux phénomènes dans l'application.

La modélisation du mouvement du ballon conduit à l'équation

$$\tau \frac{d v(t)}{dt} + v(t) = K_c u(t)$$

avec :

- $\tau$ , constante de temps ;
- $K_c$ , gain statique ;
- $v$ , vitesse du ballon ;
- $u$ , sollicitation.

L'objectif est d'obtenir la vitesse du ballon puis de déterminer sa trajectoire ultérieurement. Cette vitesse est obtenue pour une réponse temporelle du système à une entrée échelon (entrée constante) :  $u(t) = U_0$ .

**Q24.** Écrire une fonction *liste\_temps(pas,tmax)* renvoyant une liste des temps à partir du pas (intervalle entre deux instants ; cet intervalle sera pris constant) et de *tmax* (borne supérieure des temps).

- Q25.** Donner la solution analytique de l'équation différentielle précédente, en considérant des conditions initiales nulles (cela sera le cas dans toute la suite du sujet).  
Écrire une fonction *vitesse(k,tau,u,temp)* renvoyant une liste des valeurs des vitesses.
- Q26.** Programmer, en utilisant la méthode d'Euler d'ordre 1, une fonction *ordre1\_euler(K\_c, tau, U\_0, temps)*. Cette fonction prendra comme arguments d'entrée les coefficients de l'équation différentielle, la valeur de l'échelon d'entrée et la liste des temps. Cette fonction retournera une liste.
- Q27.** Nous voulons tester le programme pour différents pas de discrétisation, de [0,2; 0,4; 0,6].  
Écrire la boucle qui permet de résoudre l'équation différentielle pour les différents pas en utilisant la fonction programmée à la question **Q26**.

La bibliothèque matplotlib est détaillée en **Annexe 2** (page 16).

- Q28.** À partir de la question précédente, réécrire le script qui permet d'afficher les résultats de la résolution numérique par la méthode d'Euler pour les trois pas de temps considérés.

## Annexe 1 – Traitement d’images

### Afficher une image

```
1 import os // Import des commandes permettant de gerer les repertoires de travail
2 os.chdir('Repertoire de travail') // on se place dans le repertoire ou se trouve la base de donnees
3
4 import scipy.misc as scm // bibliotheque image
5
6 picture=scm.imread("image.bmp") // stocker l'image dans une variable
7 imshow(picture) // afficher l'image
8
9 print (picture) // donne le nombre de pixels de l'image
```

*picture* est alors un tableau aux dimensions du nombre de pixels et contenant un triplet défini pour chaque pixel utilisant le format RGB (Red, Green, Blue) en français RVB (Rouge, Vert, Bleu). La couleur “Rouge” est codée entre 00 et FF en notation hexadécimale et entre 0 et 255 en notation décimale (idem pour les couleurs “Vert” et “Bleu”).

```
1 picture = array ([[138 194 138]
2 [138 194 138]
3 [138 195 138]
4 ...,
5 [138 195 138]
6 [138 195 138]
7 [138 194 138]])
8 ...,
9 [ 49 151 49]
10 [ 49 151 49]
11 [ 49 151 49]])
```

```
1 xmax= picture.shape[1] #taille du tableau
2 ymax= picture.shape[0] #taille du tableau
3 print ("taille de l'image : %d x %d" % (xmax,ymax))
```

Pour obtenir la couleur du pixel, il suffit donc de choisir le ou les bons indices du tableau.

## Annexe 2 – Bibliothèque MATPLOTLIB.PY PLOT de Python

Cette bibliothèque permet de tracer des graphiques.

Dans les exemples ci-dessous, la bibliothèque matplotlib.pyplot a préalablement été importée à l'aide de la commande :

```
1 import matplotlib.pyplot as plt
```

On peut alors utiliser les fonctions de la bibliothèque, dont voici quelques exemples :

```
1 plt.plot(x,y)
```

- Arguments d'entrée : un vecteur d'abscisses  $x$  (tableau de dimension  $n$ ) et un vecteur d'ordonnées  $y$  (tableau de dimension  $n$ )
- Description : fonction permettant de tracer sur un graphique  $n$  points dont les abscisses sont contenues dans le vecteur  $x$  et les ordonnées dans le vecteur  $y$ . Cette fonction doit être suivie de la fonction **plt.show()** pour que le graphique soit affiché.

```
1 x= np.linspace(3,25,5)
2 y=sin(x)
3 plt.plot(x,y)
4 plt.xlabel('x')
5 plt.ylabel('y')
6 plt.show()
```

```
1 plt.xlabel(nom)
```

- Argument d'entrée : une chaîne de caractères.
- Description : fonction permettant d'afficher le contenu de nom en abscisse d'un graphique.

```
1 plt.ylabel(nom)
```

- Argument d'entrée : une chaîne de caractères.
- Description : fonction permettant d'afficher le contenu de nom en ordonnée d'un graphique.

```
1 plt.show()
```

- Description : fonction réalisant l'affichage d'un graphe préalablement créé par la commande **plt.plot(x,y)**. Elle doit être appelée après la fonction **plt.plot** et après les fonctions **plt.xlabel** et **plt.ylabel**.

**FIN**

**Repère de l'épreuve : Informatique**

**Épreuve/sous-épreuve :**

(Préciser, s'il y a lieu, le sujet choisi)

et placez les  
intercalaires dans le  
bon sens.

Note :

20

*Appréciation du correcteur\* :*

\* Uniquement s'il s'agit d'un examen.

TSII

## CONCEPTION D'UNE APPLICATION SPORTIVE "RUGBY MANAGER"

### DOCUMENT RÉPONSE

#### Question 1

```

1 import os
2 ...
3
4 """affichage stade"""
5 import ...
6 ...
7 ...

```

#### Question 2

```

1 ...
2 ...
3
4 print( ...

```

#### Question 3

```

1 def coul(
2 ...
3
4
5
6
7
8 ...

```

**Question 4**

.....  
.....  
.....

**Question 5**

```
1 def maillot(  
2  
3  
4  
5  
6  
7     ...  
.....
```

**Question 6**

```
1 def filtrer1(filtreA,matB):  
2     nA=filtreA.shape[0] #matrice carree, donc shape[0]=shape[1]  
3     nb_ligneB=...  
4     nb_colonneB=...  
5     C=matB.copy()  
6     bordure =...  
7     for i ...  
8         for j ...  
9             Bij=...  
10             C[i,j]=...  
11     return ...  
.....
```

**Question 7**

```
1 def filtrer( ...  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12     ...  
.....
```

**Question 8**

```

1 def matriceFlouGaussien (taille , sigma):
2     """
3     taille : taille de la matrice ( impaire )
4     sigma : écart type (déviation standard )
5     retourne un niveau gaussien """
6     mat = zeros ([ taille , taille ])
7     taille = ...
8     for x in range (- taille , taille +1):
9         for y in range (- taille , taille +1):
10            ...
11            ...
12    return ...

```

**Question 9**

```

1 def FloutageGaussien ( ...
2
3
4
5
6
7
8
9
10
11
12
13
14        ...

```

**Question 10**

```

1 #x
2 ...
3
4
5
6 #y_plaR
7 ...

```

**Question 11**

```

1 # listes pour l'histogramme
2 ...
3 ...
4 ...
5 ...
6 ...
7 ...

```

**Question 12**

```
1 def minMaxMoy(...  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19     ...
```

**Question 13**

Requete1=

**Question 14**

Requete2=

Requete3=

**Question 15**

Requete4=

**Question 16**

```
type(monequipe):
```

**Question 17**

```
1 def echange(l,i,j):
2     """ echange 2 valeurs d'une liste """
3     l[i],l[j] = l[j], l[i]
4
5 def tri_1(liste,criterie):
6     """ tri la liste en fonction du critère choisi """
7     for i in range(liste):
8         mini=i
9         for j in range(i+1,len(liste)):
10             if liste[j][criterie] > liste[mini][criterie]:
11                 mini=j
12         echange(liste,i,mini)
13     return liste
14
15 tri_1(monequipe,5)
16
17 print(monequipe)
```

**Question 18**

```
.....  
.....  
.....  
.....  
.....
```

**Question 19**

```

1  def segmente(T,val,i,j):
2      g=i+1
3      d=j
4      p=T[i][val]
5      while g<=d :
6          while d>=0 and T[d][val]>p:
7              d=d-1
8          while g<=j and T[g][val]<=p:
9              g=g+1
10         if g<d:
11             echange(T,g,d)
12             d=d-1
13             g=g+1
14         k=d
15     echange(T,i,d)
16     return k
17
18 def tri_2(L,val, i, j): # ...
19
20     if i<j: # ...
21
22     k=segmente(L,val,i,j) # ...
23
24     tri_2(L,val,i,k-1) # ...
25
26     tri_2(L,val,k+1,j) # ...
27
28     return L # ...
29
30 #modifier la fonction tri_2
31 def tri_2(L,val, i, j):
32
33
34
35
36     ...

```

**Question 20**

Instruction : .....

**Question 21**

$74,25_{10} = \dots \dots \dots$

.....

**Question 22**

Ecriture au format IEE754 simple précision : .....

.....  
.....  
.....

**Question 23**

Espace de stockage : .....

.....

Avantages et inconvénients du format simple ? .....

.....

**Question 24**

```
1 def liste_temps(pas,tmax):
2 ...
3 ...
4 ...
5 ...
6 ...
```

**Question 25**

.....  
.....  
.....  
.....  
.....

**Question 26**

```
1 def ordre1_euler(Kc, tau, U0, temps):  
2 ...  
3 ...  
4 ...  
5 ...  
6 ...
```

**Question 27**

```
1 ...  
2 ...  
3 ...  
4 ...  
5 ...
```

**Question 28**

```
1 #importation du module graphique de python  
2 ...  
3 for  
4 ...  
5 ...  
6 ...  
7 ...
```

**PREMIERE PARTIE – MODELISATION**  
**MOTEURS A ALLUMAGE COMMANDE**

Après le déclenchement de l'étincelle par la bougie, la combustion du mélange air-carburant se propage dans le cylindre. Pour obtenir un rendement optimal, il faut que le front de flamme atteigne le piston au moment où celui-ci est au point mort haut. L'étincelle doit être déclenchée légèrement en avance, afin de compenser le temps nécessaire à la propagation. Ce déclenchement précoce est appelé « avance à l'allumage ». Pour le quantifier, il est nécessaire de déterminer la « vitesse de flamme » dans le cylindre.

On modélise tout d'abord les **moteurs thermiques** en considérant une combustion instantanée du mélange (partie A) et on étudie la **combustion dans le cylindre** (partie B). On s'intéresse ensuite à la structure du front de flamme, avec l'**équation de la chaleur** (partie C) et un **bilan massique** (partie D), pour déterminer la **vitesse de flamme** (partie E).

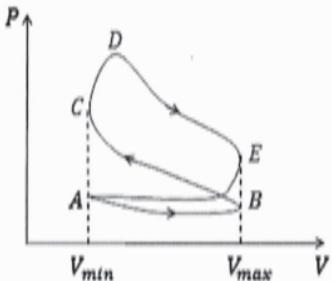
### A / MOTEURS THERMIQUES

*Le moteur est composé de plusieurs cylindres. Un piston mobile est lié à un système bielle-manivelle, ce qui transforme les allers-retours du piston en mouvement de rotation.*

*Deux soupapes permettent de faire entrer ou sortir un mélange air-essence. L'explosion entraîne une détente du mélange en repoussant le piston, lui permettant de céder du travail mécanique à l'arbre sur lequel sont fixées les roues.*

*Plus quantitativement, on distingue quatre temps :*

- admission AB du mélange par la soupape d'admission ;
- compression BC par le retour du piston ;
- combustion CD (étincelle produite par la bougie) puis détente DE ;
- échappement EA par la soupape d'échappement.

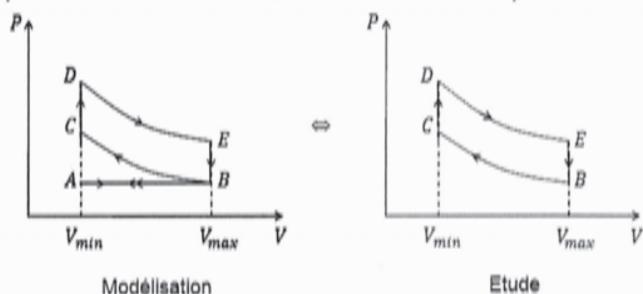


*En pratique, un moteur comporte souvent quatre pistons dont chacun est dans un temps différent de fonctionnement. Cela permet d'homogénéiser le fonctionnement du moteur et de lisser l'action motrice.*

*La modélisation du système nécessite un certain nombre d'hypothèses simplificatrices :*

- le mélange des fluides présents (air et carburant) est assimilé à un gaz parfait, caractérisé par un rapport  $\gamma = \frac{c_p}{c_v} = 1,4$  ;
- l'étape EA est assimilée à une étape EB suivie d'une étape BA ;
- les étapes CD et EB sont suffisamment rapides pour qu'elles puissent être considérées isochores ;
- les étapes BC et DE sont supposées adiabatiques.

*Le cycle BCDE obtenu peut être tracé dans le diagramme de Clapeyron et se nomme cycle de Beau de Rochas, composé de deux isochores et de deux adiabatiques.*



*On cherche à évaluer le rendement du cycle modèle, en fonction de  $\alpha = \frac{V_{max}}{V_{min}}$ , nommé rapport volumétrique ou de compression. Les adiabatiques sont supposées réversibles, et les isochores mécaniquement réversibles. La quantité de matière dans le cylindre est notée n.*

- Q1.** Identifier les phases de contact avec les sources chaude et froide. Pourquoi ces phases ne sont-elles pas réversibles ?
- Q2.** Evaluer le transfert thermique avec la source chaude  $Q_c$  en fonction de la quantité n, de la capacité thermique à volume constant  $C_{V,m}$  et des températures atteintes aux points C et D.
- Q3.** Faire de même avec  $Q_f$ .
- Q4.** Définir le rendement  $\eta$ , puis déterminer son expression en fonction des températures  $T_B$ ,  $T_C$ ,  $T_D$  et  $T_E$ .
- Q5.** Ecrire la loi de Laplace (après avoir rappelé les hypothèses nécessaires) pour les étapes BC et DE. Exprimer ensuite le rendement en fonction uniquement du rapport de compression  $\alpha$  et du rapport  $\gamma$  du gaz.
- Q6.** Comment se comporte  $\eta$  en fonction de  $\alpha$  ? Quel problème technique empêche de se rapprocher autant qu'on le désire d'un rendement unité ?

## B / COMBUSTION DANS LE CYLINDRE

*On modélise la combustion du carburant dans le cylindre avec les hypothèses suivantes :*

- *le mélange stœchiométrique combustible – comburant (dioxygène de l'air) est préalablement effectué dans le carburateur ;*
- *les réactifs et les produits sont en phase gazeuse ;*
- *la quantité de matière est conservée lors de la combustion ;*
- *les gaz frais sont à température uniforme et au repos dans le référentiel terrestre avant la réaction.*

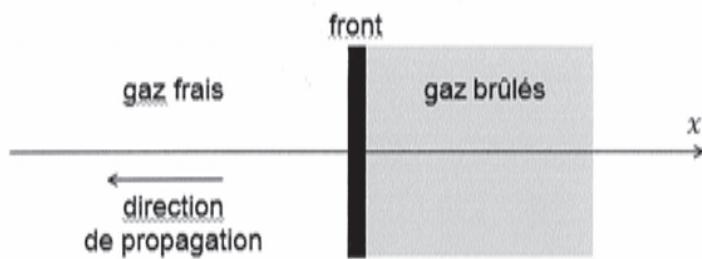
*On note symboliquement la réaction :  $C + O \rightarrow P + Q$ , où C désigne le combustible, O le comburant, P et Q les produits de la réaction.*

- Q7.** Ecrire l'équation de la combustion entre l'isoctane  $C_8H_{18}$  et le dioxygène.
- Q8.** Justifier, en s'appuyant sur la composition de l'air, l'hypothèse simplificatrice selon laquelle la quantité de gaz ne change pas.

## C / EQUATION DE LA CHALEUR

*On se place dans une géométrie simple : la réaction a lieu dans un tube calorifugé de section S : le problème est unidimensionnel selon l'axe des abscisses x.*

*On considère que la pression P, la capacité thermique massique  $c_p$  et la conductivité thermique  $\lambda$  sont constantes, uniformes et indépendantes de la température.*



On note  $T$  la température,  $\mu$  la masse volumique totale,  $\mu_C$  celle de l'espèce  $C$  et  $v$  la vitesse du gaz dans le référentiel qui se déplace avec la flamme.

L'équation locale qui traduit un bilan d'énergie s'écrit :  $\operatorname{div} \vec{j}_{th} + \mu c_p \frac{\partial T}{\partial t} = 0$ , où  $\vec{j}_{th}$  est la densité de courant thermique.

**Q9.** Simplifier l'expression dans le cadre d'une situation unidimensionnelle selon l'axe des abscisses.

**Q10.** Vérifier que les différents termes de cette équation correspondent à des puissances volumiques

Cette équation ne permet pas de décrire correctement la diffusion thermique dans le cylindre puisque la réaction de combustion dégage de la chaleur.

On note  $Q$  la chaleur dégagée par la combustion d'une mole de combustible  $C$  (enthalpie molaire).

**Q11.** Montrer que la puissance volumique dégagée par la combustion s'écrit  $P_{vol} = Qv_r$ , où  $v_r = -\frac{1}{V} \frac{dn_c}{dt}$  est la vitesse volumique de la réaction de combustion.

On ajoute cette « source » au bilan d'énergie pour écrire :  $\frac{\partial j_{th}}{\partial x} + \mu c_p \frac{\partial T}{\partial t} = Qv_r$ .

On s'intéresse maintenant à la densité de courant thermique qui comprend un terme diffusif et un terme convectif :  $\vec{j}_{th} = \vec{j}_{th}^d + \vec{j}_{th}^c$ .

**Q12.** Rappeler l'expression de la loi de Fourier pour exprimer  $j_{th}^d$ , en projection sur l'axe des abscisses. Rappeler qualitativement le sens physique du signe « - ».

**Q13.** Par analogie avec le vecteur densité de courant de masse  $\vec{j} = \mu \vec{v}$  qui définit le transport convectif, exprimer le vecteur  $\vec{j}_{th}^c$  en fonction de  $\mu$ ,  $\vec{v}$ ,  $c_p$  et  $T$ .

La conservation de la masse permet de montrer que la quantité  $\mu v$  est uniforme. On pose  $\mu v = \mu_f U$ , où  $\mu_f$  est la masse volumique des gaz frais loin de la flamme et  $U$  représente la vitesse des gaz frais par rapport à la flamme (soit en valeur absolue la vitesse de la flamme dans le référentiel terrestre).

On introduit la fraction massique  $w_C = \frac{\mu_C}{\mu}$ .

**Q14.** Réécrire l'équation différentielle vérifiée par la température. Simplifier en considérant la structure de flamme stationnaire :  $T = T(x)$ , pour obtenir une relation en fonction des paramètres  $c_p$ ,  $\lambda$ ,  $\mu_f$ ,  $U$ ,  $Q$  et  $v_r$ .

## D / BILAN MASSIQUE

Un bilan de masse sur le combustible C de masse molaire  $M_C$ , prenant en compte les aspects diffusifs et convectifs, permet d'obtenir l'équation :  $U \frac{dw_C}{dx} - \frac{\mu_D}{\mu_f} \frac{d^2 w_C}{dx^2} = -\frac{M_C v_r}{\mu_f}$ , où D est un coefficient de diffusion.

**Q15.** Déterminer la dimension du coefficient D.

On définit la concentration massique réduite  $\chi$  et la température réduite  $\theta$  par :  $\chi = \frac{w_C}{w_{C_f}}$  et  $\theta = \frac{T - T_f}{T_b - T_f}$ . Ces deux grandeurs sans dimension sont liées par  $\chi = 1 - \theta$ .

**Q16.** Déterminer les valeurs que peut prendre  $\theta$ .

**Q17.** Indiquer ce que représente  $\chi$ .

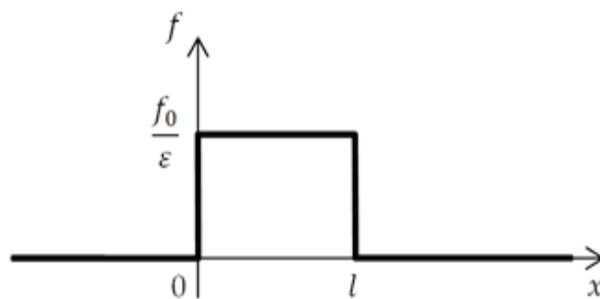
L'équation issue du bilan de masse peut être réécrite :  $U \frac{d\theta}{dx} - D_f \frac{d^2 \theta}{dx^2} = f$ , où  $D_f$  est une constante et  $f$  une fonction de  $\theta$ .

## E / VITESSE DE FLAMME

Pour résoudre l'équation, on modélise la fonction  $f$  par un créneau :

- pour  $\theta \in [0, 1 - \varepsilon[$ ,  $f(\theta) = 0$  ;
- pour  $\theta \in [1 - \varepsilon, 1[$ ,  $f(\theta) = \frac{f_0}{\varepsilon}$  ;
- pour  $\theta = 1$ ,  $f(\theta) = 0$ .

Dans la suite, on fixe l'origine 0 de l'axe des abscisses au point où  $\theta = 1 - \varepsilon$ , et on note l'abscisse du point où  $\theta$  devient égal à 1. Si on considère alors  $f$  comme une fonction de  $x$ , on admet que  $f(x)$  a l'allure suivante :



**Q18.** Montrer que dans chacun des trois domaines (domaine 1 :  $x < 0$ , domaine 2 :  $0 < x < l$ , domaine 3 :  $x > l$ ), la fonction  $\theta(x)$  est de la forme :  $\theta(x) = Ae^{\frac{x}{d}} + Bx + C$ , où A, B et C sont des constantes différentes dans chacun des domaines.

Les conditions aux limites permettent de déterminer les expressions dans chacun des domaines :

$$\begin{cases} \theta_1(x) = (1 - \varepsilon)e^{\frac{x}{d}} \\ \theta_2(x) = 1 + \frac{f_0}{\varepsilon U}(x - l) + \frac{f_0 d}{\varepsilon U} \left(1 - e^{\frac{x-l}{d}}\right) \\ \theta_3(x) = 1 \end{cases}$$

**Q19.** Justifier la continuité des fonctions  $\theta(x)$  et  $\frac{d\theta}{dx}$ .

**Q20.** Utiliser ces continuités en  $x = 0$  pour montrer que  $l = \frac{\varepsilon U}{f_0}$  et pour établir une relation entre  $\varepsilon$ ,  $f_0$ ,  $d$  et  $U$ .

*En pratique,  $\varepsilon \ll 1$ , et on rappelle que pour  $|\alpha| \ll 1$ ,  $e^\alpha = 1 + \alpha + \frac{\alpha^2}{2} + o(\alpha^2)$*

**Q21.** Dans cette approximation, montrer que  $\frac{U}{f_0} = 2d$ .

**Q22.** Justifier l'appellation « épaisseur de flamme » pour  $d$ .

**Q23.** Dans des conditions usuelles ( $T_b \approx 2000$  K),  $f_0$  est de l'ordre de  $10^4$  Hz et  $D_f$  de l'ordre de  $10^{-5} \text{ m}^2 \cdot \text{s}^{-1}$ . Calculer la vitesse de flamme  $U$  et l'épaisseur de flamme  $d$ .

## DEUXIEME PARTIE – INFORMATIQUE

### MODELISATION ET SIMULATION DE LA COMMANDE D'INJECTION D'UN MOTEUR A ALLUMAGE COMMANDE

Les algorithmes demandés seront réalisés dans le langage python. Vous porterez une attention particulière au respect de la syntaxe : indentation, définition des fonctions etc... On supposera que tout module nécessaire à l'utilisation des fonctions usuelles a été importé.

#### A / PRESENTATION DE LA PROBLEMATIQUE

Depuis l'intégration des calculateurs dans l'automobile, les principaux paramètres de commande du moteur à allumage commandé tels que l'ouverture du papillon, la durée d'injection, l'avance à l'allumage, etc... sont contrôlés numériquement par des cartographies et/ou par des boucles d'asservissement.

L'écriture du programme de gestion du moteur fait de plus en plus appel à des langages de programmation évolués qui permettent un développement rapide et fiable. Son agencement est fait par couches :

- la couche la plus basse gère les fonctions directement liées à la configuration du "hardware",
- la couche moyenne gère les signaux d'entrée et de sortie ,
- la couche supérieure supporte la stratégie de contrôle du moteur.

Le premier niveau des fonctionnalités à gérer par le calculateur comprend :

- le respect de la consigne du conducteur,
- le respect des normes d'émission à l'échappement,
- le respect des normes d'émission de vapeur,
- la minimisation de la consommation,
- la protection du moteur contre le cliquetis.

Le conducteur d'un véhicule envoie une consigne par l'intermédiaire de la pédale d'accélérateur. Cette consigne équivaut pour le calculateur à une consigne de couple moteur. Le couple moteur est directement lié à la quantité de mélange air-essence aspirée par le moteur. La quantité d'air aspirée dépend de la pression à l'admission et donc de l'ouverture du papillon des gaz. Le calculateur va réguler l'ouverture du papillon des gaz en fonction de la consigne fournie à la pédale d'accélérateur. Le calculateur doit ensuite déterminer l'avance à l'allumage et la quantité de carburant à injecter dans chacun des cylindres.

Nous allons nous intéresser dans la suite de ce sujet à la génération de la consigne d'injection de carburant à partir de la lecture des cartographies en boucle ouverte dans un premier temps. Nous envisagerons ensuite une stratégie en boucle fermée permettant d'améliorer les performances du moteur.

#### B / REPERAGE ANGULAIRE DU MOTEUR

Pour les commandes des allumages et des injections notamment, il est nécessaire de connaître précisément la position angulaire du moteur et sa vitesse de rotation.

Une cible solidaire de l'arbre moteur dispose de soixante dents. Les dents sont numérotées de 1 à 60. L'épaisseur d'une dent est égale à l'espace entre deux dents. Un capteur fixe par rapport au carter moteur permet de détecter la présence ou l'absence de dent sur la cible.

L'information issue du capteur est traitée pour être ensuite représentée par la variable booléenne *cible\_mot*. Cette variable prend la valeur vraie lorsque le capteur se trouve face à une

dent et la valeur fausse sinon. La figure 1 représente l'évolution de la variable *cible\_mot* au cours du temps pour une vitesse de rotation constante.

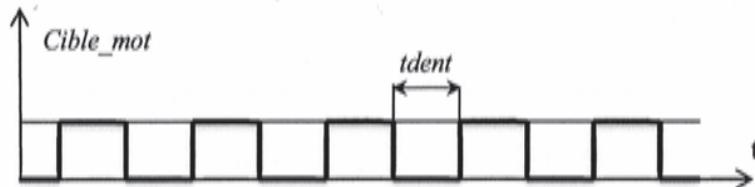


figure 1 : évolution de la variable *cible\_mot*.

La lecture de l'état des variables associées aux principaux capteurs est réalisée par le logiciel de gestion du moteur avec une période d'échantillonnage de 2 ms au mieux. L'arbre moteur tourne jusqu'à 7000 tours/min.

- Q1.** Quelle est la valeur minimale du temps écoulé entre deux changements d'état de la variable *cible\_mot* ? La période d'échantillonnage est-elle suffisante pour détecter le changement d'état entre deux fronts de la variable *cible\_mot* ?

La mesure de la rotation moteur fait donc l'objet d'un traitement numérique différencié avec une fréquence d'échantillonnage de 400 kHz. Une position angulaire quelconque est repérée par le numéro de la dent qui la précède et par la mesure de la durée écoulée entre les deux dents précédentes (durée écoulée entre deux fronts successifs de la variable *cible\_mot*, voir figure 1). Cette durée sera stockée dans la variable *tdent* en secondes.

- Q2.** Ecrire une fonction *vitesse\_moteur(tdent)* qui calcule et retourne la vitesse de rotation du moteur en tours par minute en fonction de la variable *tdent*.

## C / CONTROLE DU DOSAGE AIR-ESSENCE

### C.1 Principe du dosage air-essence

Lorsque le mélange air-essence est stoechiométrique et parfaitement homogène, le moteur à allumage commandé ne rejette que de l'eau et du dioxyde de carbone.

Le durcissement des normes antipollution a conduit les constructeurs d'automobiles à agir à différents niveaux en amont et en aval de la combustion. La première action consiste à contrôler de façon très fine le dosage du mélange en imposant lorsque les conditions d'utilisation le permettent un mélange stoechiométrique.

La quantité de carburant injecté est directement corrélée à la durée d'ouverture des injecteurs que nous noterons dans la suite du sujet **durée d'injection**.

### C.2 Contrôle du dosage air-essence par cartographies en boucle ouverte

Dans les conditions de fonctionnement stabilisé, le dosage de base est le résultat d'une interpolation cartographique calculée à partir de la vitesse et de la charge du moteur.

Toutes les cartographies et programmes du moteur sont stockés sous forme de fichiers dans la mémoire morte du calculateur (ROM) qui dispose de 32 ko d'espace. Au démarrage du véhicule, le programme de gestion du moteur et certaines données seront chargés dans la mémoire vive (RAM) qui dispose de 3 ko d'espace.

- Q3.** Expliquer les différences entre les mémoires de type ROM ou RAM. Pourquoi stocker le programme de gestion dans la RAM ?
- Q4.** Exprimer en bits l'espace mémoire disponible pour la mémoire RAM et la mémoire ROM sous la forme :  $(n\text{bits})_{10} = a \cdot b^i$ . Préciser les valeurs numériques de  $a$ ,  $b$  et  $i$  sans réaliser l'application numérique de  $(n\text{bits})_{10}$ .

Pour accéder à la mémoire, le processeur dispose d'un bus d'adresse de 24 bits. Cela signifie que pour communiquer en lecture ou en écriture avec un emplacement mémoire particulier, le processeur envoie un mot binaire codé sur 24 bits via le bus d'adresse. On appelle espace adressable le nombre d'adresses différentes accessibles par le bus d'adresse.

- Q5.** Quel est alors l'espace adressable en décimal ? Donner l'adresse maximale en binaire (base 2).

Pour représenter les adresses, le programme de gestion du moteur utilise le code hexadécimal (base 16). On donne ci-dessous la table de conversion entre les bases 10 et 16.

base 10 : $d_i =$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
base 16 : $h_i =$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

On a alors pour un nombre  $n$  écrit en base 16  $(n)_{16} = h_k \dots h_1 h_0$  la conversion en base 10 :

$$(n)_{10} = \sum_{i=0}^k d_i \cdot 16^i$$

- Q6.** Ecrire en binaire puis en hexadécimal l'adresse suivante :  $(320)_{10}$ .

- Q7.** Quel est l'intérêt d'utiliser le code hexadécimal pour la programmation ?

### C.3 Cartographie des durées d'injection

Nous allons nous intéresser au traitement de la cartographie qui permet de déterminer la durée d'injection. La figure 2 représente l'affichage d'une cartographie des durées d'injection pour un moteur 4 temps.

		Pression au collecteur P en bars						
		0,295	0,39	0,48	0,565	0,645	0,72	0,79
ligne i Rotation moteur Nm en tour/min	600	2,42	3,454	4,408	5,44	6,484	7,522	8,548
	900	2,702	3,776	4,852	5,9	6,962	8,004	9,036
	1300	3,064	4,162	5,248	6,33	7,418	8,432	9,434
	1700	3,27	4,412	5,552	6,644	7,734	8,774	9,766
	2200	3,432	4,63	5,804	6,952	8,062	9,126	10,154
	2700	3,498	4,71	5,916	7,09	8,23	9,334	10,39
	3200	3,56	4,778	6,022	7,214	3,388	9,552	10,614
	3800	3,658	4,878	6,168	7,358	8,558	9,742	10,844
	4400	3,74	5,008	6,324	7,51	8,738	9,962	11,066
	5000	3,816	5,134	6,48	7,708	8,962	10,204	11,32
	5600	3,908	5,29	6,622	7,89	9,174	10,408	11,582
	6300	3,99	5,39	6,754	8,076	9,372	10,612	11,79
	7000	4,03	5,436	6,808	8,152	9,452	10,7	11,898

colonne j

figure 2 : cartographie des durées d'injection en ms.

La cartographie est alors chargée en mémoire sous la forme suivante (voir figure 3) :

- La première ligne qui contient des valeurs de pression à l'admission en bar est stockée dans une liste de valeurs :  $P = [P_j]_{0 \leq j \leq M-1}$ .
- La première colonne qui contient des valeurs de la vitesse de rotation moteur en tour/min est stockée dans une liste de valeurs :  $Nm = [Nm_i]_{0 \leq i \leq N-1}$ .
- Les durées d'injection sont stockées sous forme d'un tableau  $T$  à deux dimensions de taille  $N \times M$ . On peut lire pour chaque couple de valeurs de pression et de rotation moteur la valeur de la durée d'injection correspondante  $T[i, j]$  en ms.

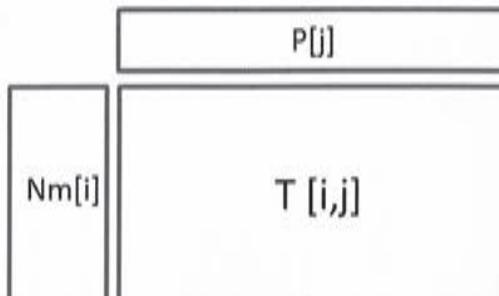


figure 3 : structure des données de cartographie en mémoire.

Le capteur de pression au collecteur d'admission mesure une valeur de pression notée  $P_{col}$ . La vitesse de rotation du moteur mesurée est notée  $Nmot$ . Pour un couple de valeurs pression-vitesse de rotation ( $P_{col}, Nmot$ ) quelconque le calculateur doit alors déterminer les valeurs de durée d'injection en réalisant une interpolation à partir des valeurs du tableau  $T$ .

#### Etape 1 :

Le calculateur doit déterminer les indices  $i$  et  $j$  tels que  $P[j] \leq P_{col} < P[j + 1]$  et  $Nm[i] \leq Nmot < Nm[i + 1]$ .

- Q8.** Ecrire une fonction *indice(A, val)* qui prend pour argument une liste notée  $A$  et un réel noté  $val$ . La liste  $A$  est triée par ordre croissant. Votre fonction doit retourner un entier  $id$  tel que :  $A[id] \leq val < A[id + 1]$ . On supposera que  $id$  existe toujours.

Le calculateur doit ensuite lire les durées d'injection dans le tableau  $T$  correspondant aux indices  $i$  et  $j$  précédemment déterminés.

Pour la suite du sujet, un tableau pourra être représenté :

- soit par un tableau de type `numpy.array` ;
- soit par une liste de listes.

- Q9.** Ecrire une fonction *extraire(T, P, Nm, i, j)* qui prend pour argument le tableau de dimension  $N \times M$  noté  $T$ , les listes  $P$  et  $Nm$ , et deux entiers  $i$  et  $j$ . Votre fonction doit retourner un tableau  $ST$  de dimension 2X4 :

$$ST = \begin{bmatrix} T[i, j] & T[i, j + 1] & P[j] & Nm[i] \\ T[i + 1, j] & T[i + 1, j + 1] & P[j + 1] & Nm[i + 1] \end{bmatrix}$$

Pour la suite, les deux premières colonnes du tableau  $ST$  correspondent aux durées d'injection de la table. La troisième colonne correspond aux valeurs de pression encadrant  $P_{col}$ . La quatrième colonne correspond aux valeurs de rotation moteur encadrant  $Nmot$ .

- Q10.** Ecrire la suite d'instructions qui à partir des variables  $Nmot$ ,  $P_{col}$ , des listes  $P$  et  $Nm$  et du tableau  $T$  permet de déterminer le sous tableau  $ST$  tel que défini à la question Q9. Le résultat pourra être affecté à une variable nommée  $ST$ .

Etape 2 :

Une fois les quatre valeurs de durée d'injection déterminées, il est nécessaire de calculer une durée d'injection  $t$  à partir d'une interpolation bilinéaire.

L'interpolation bilinéaire de la fonction de deux variables  $t(x, y)$  s'écrit comme suit :

$$t(x, y) = b_0 + b_1 \cdot x + b_2 \cdot y + b_3 \cdot x \cdot y ; x \in [0,1] \text{ et } y \in [0,1]$$

où  $b_0, b_1, b_2$  et  $b_3$  sont les inconnues du problème.

La détermination des coefficients  $b_{i,i \in [0,3]}$  est un problème linéaire qui doit être résolu à chaque pas de temps et pour chaque cartographie sous la forme :

$$A \cdot b = c \text{ et } b = A^{-1} \cdot c$$

On envisage dans un premier temps l'utilisation de l'algorithme du pivot de Gauss pour résoudre le système linéaire.

**Q11.** Ecrire en pseudo-code dans le cas d'une matrice carrée de dimension  $n \times n$  l'algorithme du pivot de Gauss permettant de résoudre le système  $A \cdot b = c$ . On admettra que les termes diagonaux sont tous non nuls.

**Q12.** Quelle est la complexité du pivot de Gauss en fonction de la dimension de la matrice ? Justifier votre réponse.

On posera pour la suite :

$$x = \frac{Pcol - P[j]}{P[j+1] - P[j]} ; \text{ et } y = \frac{Nmot - Nm[i]}{Nm[i+1] - Nm[i]}$$

On connaît les valeurs de  $t(x, y)$  pour quatre couples de valeurs  $(x, y)$  par lecture de la cartographie :

$$\begin{pmatrix} t(x=0, y=0) = ST[0,0] & t(x=1, y=0) = ST[0,1] \\ t(x=0, y=1) = ST[1,0] & t(x=1, y=1) = ST[1,1] \end{pmatrix}$$

Le problème à résoudre devient alors :

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} ST[0,0] \\ ST[0,1] \\ ST[1,0] \\ ST[1,1] \end{bmatrix}$$

On peut donc résoudre le système de façon systématique par substitution de la manière suivante :

$$\begin{aligned} b_0 &= ST[0,0] \\ b_1 &= ST[0,1] - ST[0,0] \\ b_2 &= ST[1,0] - ST[0,0] \\ b_3 &= ST[1,1] - ST[0,1] - ST[1,0] - ST[0,0] \end{aligned}$$

**Q13.** Comparer la complexité de cet algorithme à celui présenté à la question Q11. Conclure.

**Q14.** Ecrire une fonction *interp(ST,Pcol,Nmot)* qui retourne une valeur de durée d'injection obtenue par interpolation bilinéaire des éléments de la table.

## D / AMELIORATION DES PERFORMANCES D'INJECTION : BOUCLE FERMEE

La durée d'injection calculée à partir de l'interpolation étudiée dans la partie précédente sera corrigée par un ou plusieurs coefficients issus d'autres cartographies prenant en compte la température du moteur, la consigne du conducteur etc... Mais la principale correction consiste en un fonctionnement en boucle fermée tel que représenté sur la figure 4 (source Bosch).

Objectif : nous allons modéliser le comportement de cette boucle fermée afin de simuler son comportement.

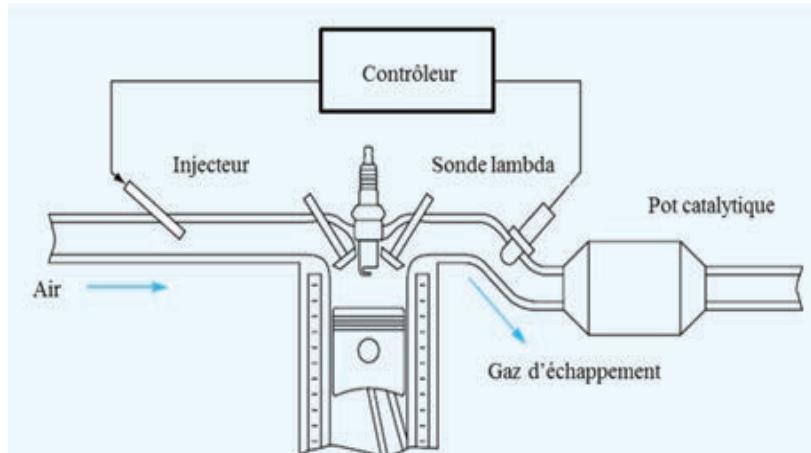


figure 4 : principe du contrôle en boucle fermée.

La sonde Lambda est placée au niveau du pot catalytique, elle mesure le taux résiduel d'oxygène qui dépend directement de la richesse du mélange. Pour définir le mélange air-essence on calcule le rapport  $\frac{A}{F} = \frac{\text{masse d'air}}{\text{masse d'essence}}$ . Dans les conditions d'un mélange stœchiométrique on a  $\frac{A}{F} = 14,6$ . On définit alors le rapport lambda par :  $\lambda = \frac{(A/F)_{réel}}{14,6}$ .

La tension délivrée par la sonde ne conduit qu'à deux états stables en fonction du taux d'oxygène dans les gaz brûlés.

On aura alors :

- $\lambda \geq 1$  : mélange pauvre,  $U_{sonde} = 0,1 V$  ;
- $\lambda < 1$  : mélange riche,  $U_{sonde} = 0,9 V$  ;
- **la richesse** du mélange est définie par  $r(t) = \frac{1}{\lambda(t)}$ .

**Q15.** Ecrire une fonction  $sonde(r)$  qui retourne la valeur de la tension de la sonde en fonction de la valeur de la richesse  $r$  en accord avec le fonctionnement décrit ci-dessus.

Le principe du contrôle en boucle fermée consiste à comparer la tension délivrée par la sonde à la valeur moyenne de 500 mV. Le contrôleur utilisé est du type proportionnel et intégral.

La consigne de durée d'injection déterminée à partir des différentes cartographies est notée  $tinjc0$ .

On aura alors la loi de commande suivante :

- $tinj(t) = \frac{tinjc0 \cdot Kpp + Ki \cdot \int_0^t tinjc0 \cdot dt}{terme\ proportionnel\ terme\ intégral}$  pour  $Usonde \leq 0,5 V$ , phase croissante ;
- $tinj(t) = \frac{tinjc0 \cdot Kpn - Ki \cdot \int_0^t tinjc0 \cdot dt}{terme\ proportionnel\ terme\ intégral}$  pour  $Usonde > 0,5 V$ , phase décroissante.

Avec :  $Kpp > 1$ ,  $Kpn < 1$  et  $Ki > 0$ .

Le terme intégral de gain  $\pm Ki$  croît ou décroît linéairement en fonction du temps suivant que la sonde indique un mélange pauvre ou riche.

À chaque détection de transition du signal de la sonde (franchissement de la valeur seuil de 0,5 V), on applique le terme proportionnel. Le terme intégral est quant à lui réinitialisé à 0. La borne inférieure  $t = 0$  du terme intégral correspond donc à l'instant initial d'une phase croissante ou décroissante.

On introduit les consignes de programmation suivantes :

- On note le pas de temps entre deux évaluations aux instants  $t_l$  et  $t_{l+1}$  :  $dt = t_{l+1} - t_l$ .
- On note :
  - la valeur de la durée d'injection à l'instant  $t_l$  :  $tinj(t_l) = tinj_l$  ;
  - la valeur du terme intégral à l'instant  $t_l$  :  $integ(t_l) = integ_l$  ;
- Le terme intégral sera évalué par la méthode des rectangles ( $tinjc0$  est constant).

Remarque : La détection de transition du signal de la sonde peut être réalisée en fonction du signe de  $integ_l$  et de la valeur de  $Usonde$ .

**Q16.** Ecrire une fonction  $duree_injection(Usonde, Kpp, Kpn, Ki, tinjc0, integi, dt)$  qui retourne la commande d'injection  $tinj(t_{l+1})$  et la valeur de  $integ(t_{l+1})$  à l'instant  $t_{l+1}$  conformément à la description donnée ci-dessus.

On donne ci-dessous la forme canonique d'une équation différentielle du premier ordre :

$$\tau \frac{ds(t)}{dt} + s(t) = K \cdot e(t)$$

**Q17.** Ecrire la relation de récurrence qui permet de calculer  $s(t_{l+1})$  à partir des valeurs  $\tau$ ,  $K$ ,  $dt = t_{l+1} - t_l$ ,  $s(t_l)$  et  $e(t_l)$  par la méthode d'Euler explicite.

On note à l'instant  $t_l$  :  $s(t_l) = si$ ,  $e(t_l) = ei$ .  $\tau$  sera noté  $tau$  dans vos programmes.

**Q18.** Ecrire une fonction  $Euler(tau, K, dt, si, ei)$  qui détermine et retourne la valeur de  $s(t_{l+1})$  par la méthode d'Euler explicite.

La réponse en richesse mesurée par la sonde peut être en première approximation assimilée à deux retards successifs dus à la dynamique du moteur d'une part et à la capacité d'absorption d'oxygène du pot catalytique d'autre part. L'évolution de la richesse notée  $r(t)$  en fonction de la consigne de durée d'injection est donc régie par l'équation :

$$tau1 \cdot tau2 \frac{d^2r(t)}{dt^2} + (tau1 + tau2) \frac{dr(t)}{dt} + r(t) = K \cdot tinj(t)$$

Cette équation différentielle du second ordre peut s'écrire sous la forme du système de deux équations différentielles suivantes :

$$\tau_{au1} \frac{dw(t)}{dt} + w(t) = K \cdot tinj(t)$$

$$\tau_{au2} \frac{dr(t)}{dt} + r(t) = w(t)$$

On note à l'instant  $t_i$  :  $r(t_i) = ri$ ,  $tinj(t_i) = tinji$ ,  $w(t_i) = wi$ .

**Q19.** Ecrire une fonction *richesse*( $\tau_{au1}, \tau_{au2}, K, dt, ri, wi, tinji$ ) qui détermine et retourne la valeur de  $r_{i+1}$  et  $w_{i+1}$  en utilisant la fonction Euler définie à la question Q18.

On souhaite réaliser une simulation pour un moteur tournant à 4000 tours/min et présentant 2 cycles moteur par tour.

**Q20.** Quelle est la durée d'un cycle moteur noté *Tcycle* ?

L'ensemble des variables définies ci-dessous seront considérées comme préalablement déclarées dans le programme. On suppose les conditions initiales suivantes :

- $\tau_{au1}=10.Tcycle$
- $\tau_{au2}=20.Tcycle$
- $Kpp=1,03$
- $Kpn=0,95$
- $Ki=0,8$
- $tinjc0=12,3\text{ ms}$
- $t0=0\text{ s}$
- $tn=100.Tcycle$
- $dt=2.10^{-6}\text{ s}$
- $w0=0,9$
- $K=1/tinjc0$
- $integ=0$

La durée de simulation sera de 100 cycles moteur.

**Q21.** Ecrire la suite d'instructions qui permet de créer la liste temps =  $[t_0, \dots, t_n = 100.Tcycle]$  avec un pas de temps régulier dt.

On initialise les trois listes suivantes :

- *liste\_richesse=[0.9]* ; qui contiendra la valeur de la richesse calculée à chaque pas de temps.
- *liste\_tinj=[tinjc0]* ; qui contiendra la valeur de durée d'injection calculée à chaque pas de temps.
- *liste\_U=[sonde(liste\_richesse[0])]* ; qui contiendra la valeur de la tension Usonde de la sonde Lambda calculée à chaque pas de temps.

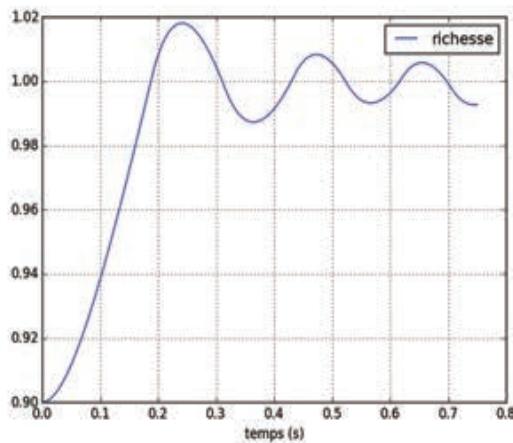
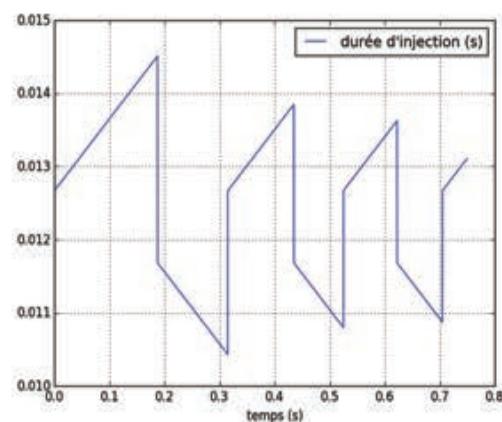
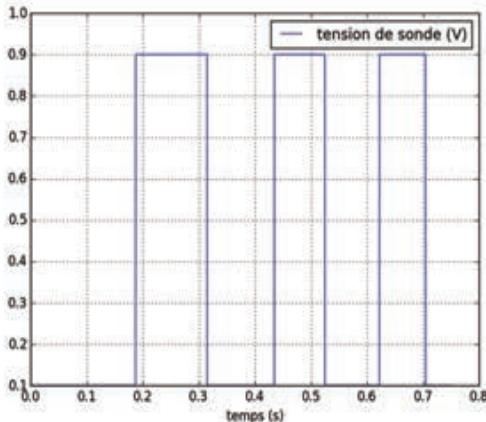
### Principe de la simulation :

Nous allons calculer à chaque pas de temps les valeurs de richesse et de tension *Usonde* afin de déterminer la commande de durée d'injection.

**Consigne de programmation :** Afin de représenter le comportement réel, la valeur de la tension *Usonde* est lue par le calculateur avec une période égale à *Tcycle* et reste donc constante sur cette période au moins avant d'être lue à nouveau.

**Q22.** Ecrire la suite d'instructions qui détermine l'évolution de la richesse en boucle fermée à partir de la liste *temps* et des fonctions *richesse()*, *duree\_injection()* et *sonde()*. Votre programme devra compléter trois listes de valeurs : *liste\_tinj*, *liste\_U* et *liste\_richesse*. Vous veillerez à respecter la consigne de programmation.

On obtient après simulation les courbes suivantes :



**Q23.** Ecrire la suite d'instructions qui permet d'obtenir le tracé de la richesse ci-dessus.

## E / AMELIORATION DES PERFORMANCES D'INJECTION : AUTO-APPRENTISSAGE

La détermination des cartographies est réalisée sur banc d'essai pour différentes situations de charges du moteur. La cartographie ainsi réalisée ne permettra pas en général d'obtenir un mélange stœchiométrique dans toutes les situations rencontrées. Grâce à la sonde Lambda, le calculateur parvient à connaître l'écart de richesse par rapport à un objectif de mélange stœchiométrique, il pourra alors modifier et mémoriser les coefficients des cartographies ainsi corrigés au cours des cycles de fonctionnement du moteur. Cet auto-apprentissage est le plus souvent réalisé sur circuit en faisant varier les conditions d'utilisation du moteur. Le calculateur est alors relié à un ordinateur externe.

Afin de pouvoir revenir à une configuration de cartographie antérieure aux dernières modifications réalisées par l'auto-apprentissage, les différentes cartographies sont stockées dans une base de données sur la mémoire de l'ordinateur externe. Cette mémorisation permet également de fusionner certaines cartographies en ne gardant que les parties les plus performantes. Une description partielle de la table *table\_injection* est donnée pour la gestion des injecteurs :

- Id : numéro d'identifiant constituant la clé primaire,
- année : année d'enregistrement du paramètre de cartographie,
- mois : mois d'enregistrement du paramètre de cartographie,
- jour : date d'enregistrement du paramètre de cartographie,
- heure : heure d'enregistrement du paramètre de cartographie,
- minute : minute d'enregistrement du paramètre de cartographie,
- Pression : pression au collecteur d'admission,
- Rotation : vitesse de rotation du moteur,
- Température : température du liquide de refroidissement,
- Commande : commande transmise à la pédale d'accélérateur en %,
- ...
- Durée\_injection : durée d'ouverture des injecteurs (en ms),
- Commande\_ouverture : instant d'ouverture des injecteurs (en ° moteur),
- Qualité : qualité de la cartographie en % par rapport à l'objectif de richesse.

On souhaite dans un premier temps ne conserver en mémoire que les enregistrements présentant une qualité supérieure à 95 %.

**Q24.** Ecrire une requête SQL permettant d'extraire la cartographie des durées d'injection présentant une qualité supérieure à 95 %. Tous les champs seront extraits et enregistrés dans une nouvelle table nommée table\_injection\_95.

Afin de reconstruire une cartographie optimale, il faut sélectionner les meilleurs enregistrements, nous allons ébaucher cette étape.

**Q25.** Ecrire une requête SQL permettant d'extraire de la table créée question Q24 les champs Pression, Rotation et Durée\_injection présentant la qualité maximale pour les pressions comprises entre 0,3 et 0,4 bar et les vitesses de rotation comprises entre 1300 et 1700 tours par minute.

**FIN DE L'EPREUVE**

**Question 1** Parmi les composants suivants d'un ordinateur, lesquels sont des périphériques d'entrée ?

- A) Le clavier.
- B) Le processeur.
- C) La mémoire vive.
- D) Un microphone.

**Question 2** Pourquoi vaut-il mieux éviter de comparer un nombre flottant à zéro ?

- A) Parce que zéro ne peut pas être représenté en flottant.
- B) Parce que le temps de calcul nécessaire pour cette comparaison est trop long par rapport à d'autres comparaisons entre flottants.
- C) Il n'est pas problématique de comparer un flottant à zéro, à condition de coder les flottants sur 64 bits.
- D) Parce que cela provoque un dépassement de capacité ("overflow").

**Question 3** On considère le problème du calcul approché d'une intégrale sur un segment d'une fonction non constante de classe  $\mathcal{C}^2$ . On utilise pour cela une subdivision régulière du segment en  $n$  sous-intervalles de même longueur.

- A) La méthode des rectangles à droite est plus efficace en terme de complexité temporelle que celle des rectangles à gauche.
- B) La méthode des rectangles à gauche est plus efficace en terme de complexité temporelle que celle des rectangles à droite.
- C) Pour  $n$  suffisamment grand, l'erreur commise en utilisant la méthode des rectangles à droite est beaucoup plus petite que l'erreur commise en utilisant celle des trapèzes.
- D) Pour  $n$  suffisamment grand, l'erreur commise en utilisant la méthode des trapèzes est beaucoup plus petite que l'erreur commise en utilisant celle des rectangles à droite.

**Question 4** L'affectation d'une variable consiste toujours à :

- A) incrémenter la variable en question.
- B) renommer la variable en question.
- C) comparer la valeur de cette variable à une autre valeur.
- D) associer une valeur à une variable.

**Question 5** Après avoir exécuté le code suivant, quelles sont les valeurs des variables `a` et `b` ?

```
a=13
b=4
b=a
a=b
```

- A) `a=13` et `b=13`.
- B) `a=4` et `b=4`.
- C) `a=4` et `b=13`.
- D) `a=13` et `b=4`.

**Question 6** Quand on compare deux nombres entiers en Python, le résultat est :

- A) un flottant.
- B) un entier.
- C) un booléen.
- D) une chaîne de caractères.

**Question 7** On considère le script Python suivant (on suppose que les variables `b` et `n` ont déjà été définies auparavant, et qu'elles sont à valeurs entières strictement positives) :

```
x=b
k=n
z=1
while k>0:
    if k%2==1:
        z=z*x
    x=x*x
    k=k//2
```

- A) “ $b^n = z \times x^k$ ” est un invariant pour la boucle “`while`”.
- B) “ $k \geq 0$ ” est un invariant pour la boucle “`while`”.
- C) La boucle “`while`” ne possède pas d'invariant de boucle.
- D) “ $k = n$ ” est un invariant pour la boucle “`while`”.

**Question 8** Peu importe les valeurs entières strictement positives que l'on puisse donner aux variables `b` et `n`, à la fin de l'exécution du script de la question précédente, on aura forcément :

- A)  $b^n = x$ .
- B)  $b^n = z$ .
- C)  $x^n = b$ .
- D)  $z^n = b$ .

**Question 9** On considère la chaîne de caractères `C="CONTROLEUR"`.

- A) `C[7]` vaut "L".
- B) `C[7]` vaut "E".
- C) `C[:7]` vaut "CONTROL".
- D) `C[:7]` vaut "CONTROLE".

**Question 10** On applique la méthode de Newton pour résoudre numériquement l'équation

$\sqrt{|x|} = 0$ , en prenant  $x_0 = 100$ . On note  $(x_n)_{n \in \mathbb{N}}$  la suite des approximations obtenues.

- A)  $\forall n \in \mathbb{N}, x_{n+1} = -\frac{x_n}{2}$ .
- B)  $\forall n \in \mathbb{N}, x_{n+1} = -x_n$ .
- C) La suite  $(x_n)_{n \in \mathbb{N}}$  converge vers 0.
- D) Pour que la suite  $(x_n)_{n \in \mathbb{N}}$  converge vers 0, il faudrait choisir un  $x_0 > 0$  beaucoup plus proche de 0.

**Question 11** On considère l'équation différentielle  $y' = ay$  (où  $a \in \mathbb{R}$  est une constante donnée), avec la condition initiale  $y(0) = 1$ . On note  $y_0, \dots, y_N$  ( $N \geq 1$ ) la suite obtenue en appliquant la méthode d'Euler explicite avec un pas de discrétisation  $h > 0$  sur l'intervalle  $[0, T]$ .

- A) Pour  $0 \leq n \leq N - 1$ , on a :  $y_{n+1} = (1 + ah)y_n$ .
- B) Pour  $0 \leq n \leq N - 1$ , on a :  $y_{n+1} = (1 - ah)y_n$ .
- C)  $h = T \times N$ .
- D) Plus  $N$  augmente, plus  $y_N$  se rapproche de  $e^{aT}$ .

**Question 12** De manière générale, concernant la méthode d'Euler explicite pour résoudre numériquement une équation différentielle, si l'on diminue le pas de discréétisation :

- A) on peut améliorer la qualité de l'approximation obtenue.
- B) on détériore la qualité de l'approximation obtenue.
- C) on augmente le temps de calcul.
- D) on diminue le temps de calcul.

**Question 13** On considère la fonction Python suivante, qui s'applique à des listes numériques :

```
def rem(L):
    a=L.pop()
    n=len(L)
    for i in range(n):
        b=L.pop()
        L.append(a)
        a=b
```

On définit  $L=[1, 2, 3, 4, 5, 6]$ .

- A) L'instruction `rem(L)` renvoie  $[1, 2, 3, 4, 5]$ .
- B) L'instruction `rem(L)` renvoie  $[2, 3, 4, 5, 6]$ .
- C) L'instruction `rem(L)` ne renvoie rien.
- D) L'instruction `rem(L)` provoque un message d'erreur.

**Question 14** Dans la fonction de la question précédente :

- A) `a` est une variable globale.
- B) `a` est une variable locale.
- C) L'instruction `rem(L)` modifiera une liste `L` auparavant définie et non vide.
- D) L'instruction `rem(L)` ne modifiera pas une liste `L` auparavant définie et non vide.

**Question 15** On considère une table `CANDIDATS(id,nom,prenom,age,sexe,adresse,tel)` appartenant à la base de données des candidats au concours ICNA. Le premier champ indique le numéro d'inscription des candidats, et les autres désignent de façon évidente les autres renseignements. Quel champ peut servir de clé primaire ?

- A) Aucun.
- B) `nom`
- C) `id`
- D) `adresse`

**Question 16** Quelle opération permet, dans une base de données, de manipuler des informations provenant de plusieurs tables différentes ?

- A) Un tri.
- B) Un renommage.
- C) Un schéma de relation.
- D) Une jointure.

**Question 17** On considère une liste L remplie de 1 et de taille  $n$ , où  $n$  est un entier naturel non nul donné. On applique l'algorithme de tri par insertion (par ordre croissant) sur la liste L. Combien de comparaisons entre éléments de la liste L seront effectuées dans le pire des cas ? On donnera un ordre de grandeur.

- A) Un  $O(\ln n)$ .
- B) Un  $O(n)$ , mais pas un  $O(\ln n)$ .
- C) Un  $O(n^2)$ , mais pas un  $O(n)$ .
- D) Un  $O(n \ln(n))$ , mais pas un  $O(n)$ .

**Question 18** On applique l'algorithme de tri par insertion (par ordre croissant) à la liste  $L=[15, 20, 10, 18]$ . Quelles sont les évolutions possibles de la liste L au cours de ce tri ?

- A)  $[15, 20, 10, 18]$  (initialisation)  $\rightarrow [15, 10, 20, 18] \rightarrow [10, 15, 20, 18] \rightarrow [10, 15, 18, 20]$
- B)  $[15, 20, 10, 18]$  (initialisation)  $\rightarrow [15, 10, 20, 18] \rightarrow [15, 10, 18, 20] \rightarrow [10, 15, 18, 20]$
- C)  $[15, 20, 10, 18]$  (initialisation)  $\rightarrow [10, 20, 15, 18] \rightarrow [10, 15, 20, 18] \rightarrow [10, 15, 18, 20]$
- D)  $[15, 20, 10, 18]$  (initialisation)  $\rightarrow [20, 10, 15, 18] \rightarrow [10, 15, 18, 20]$

**Question 19** Lequel de ces algorithmes de tri a la meilleure complexité dans le pire cas ?

- A) Le tri par insertion.
- B) Le tri rapide.
- C) Le tri fusion.
- D) Ils ont tous la même complexité dans le pire cas.

**Question 20** La complexité d'un algorithme (appliqué par exemple à une liste de taille  $n$ ) vérifie la formule  $C(n) = C\left(\frac{n}{2}\right) + n$ , et  $C(1) = 1$ . Que peut-on dire de  $C(n)$  ?

- A) C'est un  $O(\ln n)$ , mais pas un  $O(n)$ .
- B) C'est un  $O(\ln n)$ , et donc un  $O(n^k)$  pour tout entier  $k \in \mathbb{N}^*$ .
- C) C'est un  $O(n \ln n)$ , mais pas un  $O(\ln n)$ .
- D) C'est un  $O(n^2)$ , mais pas un  $O(n)$ .



# Option informatique

---

## Banque X-E.N.S. (2017)

<b>Informatique A - XULCR (4 h) [i173m1e]</b> . . . . .	3
Jeux à un joueur et solutions optimales	
<b>Informatique-Mathématiques (4 h) [m173mie]</b> . . . . .	10
Détection de carrés dans les mots	

## Concours Mines-Ponts (2017)

<b>Informatique (3 h) [i17mmoe]</b> . . . . .	20
Automates et langages unaires. Chaîne d'addition et exponentiation	

## Concours Centrale-SupÉlec (2017)

<b>Informatique (4 h) [i17cmoe]</b> . . . . .	27
Mots synchronisants	

## Concours Communs Polytechniques (2017)

<b>Informatique (4 h) [i17pmoe]</b> . . . . .	33
Logique et calcul de propositions	
Algorithmique et programmation (informatique commune) : algorithme de recherche dichotomique	
Automates et langages : algorithme du produit synchronisé	

## Concours E3A (2017)

<b>Informatique (3 h) [i17rmoe]</b> . . . . .	43
Quatre exercices d'algorithmique et programmation	

## EPITA/IPSA (2017)

<b>Informatique (2 h) [i17wmoe]</b> . . . . .	48
Un QCM et quatre exercices	

## CAPES externe de Mathématiques, option Informatique, sujet zéro (2017)

<b>Sujet zéro (5 h) [i17c0ze]</b> . . . . .	57
Problème 1 : L-systèmes	
Problème 2 : Mots circulaires, mots de DE BRUIJN	

## CAPES externe de Mathématiques, option Informatique (2017)

**Sujet zéro (5 h) [i17c3ze]** . . . . . 71

Problème 1 : Résolution de grilles de sudokus

Problème 2 : Recherche de l'enveloppe convexe d'un ensemble fini de points

## E.N.S. Paris – Sélection internationale (2016)

**Informatique (3 h) [i16uxue]** *Informatique, spécialité principale* . . . . . 84

Cinq exercices

**Informatique (2 h) [i16uxve]** *Informatique, spécialité secondaire* . . . . . 87

Trois exercices

# Épreuves d'informatique

---

## Banque X-E.N.S. (2017)

<b>Informatique B - XEC (2 h), filières MP-PC-PSI [i173m2e]</b> . . . . .	89
Intersection de deux ensembles de points	

## Concours Mines-Ponts (2017)

<b>Informatique (1 h 30), toutes filières [i17mice]</b> . . . . .	96
Étude de trafic routier	

## Concours Centrale-SupÉlec (2017)

<b>Informatique (3 h), toutes filières [i17cice]</b> . . . . .	103
Mission d'exploration martienne	

## Concours Communs Polytechniques (2017)

<b>Informatique (3 h), filière PSI [i17psue]</b> . . . . .	111
Étude de la capacité et de la congestion de l'autoroute A7	
<b>Informatique (3 h), filière TSI [i17piue]</b> . . . . .	126
Conception d'une application sportive "Rugby Manager"	

## Banque PT (2017)

<b>Informatique et Modélisation de systèmes physiques (4 h) [i17dtue]</b> 150	
Modélisation et simulation de la commande d'injection d'un moteur à allumage commandé	

## I.C.N.A. (2017)

<b>Épreuve optionnelle facultative (QCM : 1 h) [i17b2ue]</b> . . . . .	165
--	-----