



R o b o t i q u e
CRC
R o b o t i c s

CRC
**COMPÉTITION DE
PROGRAMMATION**



LIVRET DE PROBLÈMES
BLOC A

QUELQUES NOTES

- Les règles complètes sont dans la section 4 du livret des règlements
- Vous avez jusqu'au **19h59** pour remettre vos solutions
- N'hésitez pas à nous poser des questions et à communiquer avec les membres de votre équipe de programmation. Ce n'est pas un examen!
- **On vous donne des fichiers modèles faciles à utiliser pour votre code. Merci de les utiliser.**

UTILISATION DU FICHIER MODÈLE

- Tous les fichiers possèdent une fonction `solve()` dans lequel vous devez résoudre le problème. **NE CHANGEZ PAS LE NOM DE CETTE FONCTION!**
- Pour rouler vos tests, dans un terminal (nous vous conseillons l'utilisation de `vscode`) écrivez la ligne de commande: `pytest` pour rouler les tests. Si vous faites la commande `pytest` dans le répertoire contenant tous les fichiers, tous les tests seront exécutés.
- Pour rouler les tests d'un problème spécifique, naviguez dans le répertoire de la question et exécutez la commande `pytest` pour rouler les tests du problème. Vous pouvez faire la même chose pour une section entière!

STRUCTURE

Chaque problème contient une petite mise en situation comme celle-ci expliquant les fondements du problème et donnant les bases nécessaires pour résoudre celui-ci. Chaque problème préparatoire contient aussi la répartition des points pour le problème.

Spécifications d'entrée et de sortie:

Dans cette section, on retrouve les caractéristiques des entrées qui peuvent être fournies au code en question ainsi que les critères attendus pour les sorties du programme.

Exemple d'entrée et de sortie:

Dans cette section se trouve un exemple d'entrée (parfois constitué lui-même de plusieurs sous-exemples) pour que vous puissiez tester votre code. L'exemple de sortie donne donc la réponse attendue pour cette entrée.

Explication de la première sortie:

Si le problème n'est toujours pas clair après la mise en situation, l'explication de la première sortie sert parfois à démêler le tout en expliquant comment la première entrée est traitée et en montrant le chemin menant à cette réponse.

Table des matières

PROBLÈMES 2D (2D)	4
2D1: Squaresception (20 points)	4
2D2: Connecte-moé-ça (30 points)	6
2D3: Fourmis dans les jambes (45 points)	8
2D4: Mycologie 101 (75 points)	11
STRUCTURES DE DONNÉES (DS)	16
DS1: Horizon vertical (15 points)	16
DS2: Transvection (35 points)	17
DS3: Parcours d'arbre (45 points)	20
CALCULS SCIENTIFIQUES (SC)	22
SC1: Cercle carré (25 points)	22
SC2: Balancez la balance! (65 points)	23
SC3: Fractionnellement Additif (35 points)	25
TRAITEMENT DE TEXTE (TP)	27
TP1: Réglez les règles (15 points)	27
TP2: Zipper (25 points)	28
TP3: Palindrôle (30 points)	29

PROBLÈMES 2D (2D)

2D1: Squaresception (20 points)

Vous devez afficher des carrés insérés les uns dans les autres. Nous vous fournirons le nombre de carrés voulus et vous nous renverrez cette belle construction!! Vous aurez besoin des symboles suivants: (ascii 9488)'⌏', (ascii 9492)'┐', (ascii 9496)'┑', (ascii 9484)'┒', (ascii 9472)'—' et (ascii 9474)'|'.

Spécifications d'entrée:

En entrée vous recevrez:

- ***n***: le nombre de carrés à insérer

Spécifications de sortie:

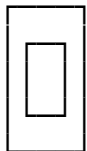
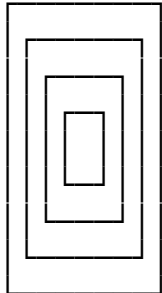
En sortie vous devrez fournir:

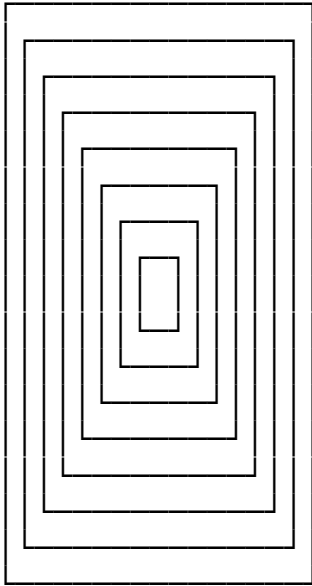
- un array de strings qui correspond aux carrés imbriqués

Exemple d'entrée:

4
2
8

Exemple de sortie:





Explication de la première sortie:

Nous avons quatre carrés imbriqués sans aucun espacement supplémentaire. Le carré du centre est composé de 4 coins, 2 barres horizontales, 2 barres verticales et d'un espace au centre.

2D2: Connecte-moé-ça (30 points)

Vous jouez aux échecs avec plusieurs dames sur le plateau. Vous souhaitez indiquer par une ligne celles d'entre elles qui sont capables de voir une des autres. Pour ce faire, vous allez indiquer par des lignes les connexions possibles entre les différentes dames. N'oubliez pas que les dames peuvent se déplacer d'autant de cases que voulu que ce soit sur une même rangée, sur une même colonne ou sur une même diagonale à partir de la position actuelle.

Les cases vides de l'échiquier seront représentées par des . et les cases contenant des dames seront remplacées par des x. Vous devrez remplir les liens entre chacune d'entre elles par les symboles - | / \ pour former une ligne qui connecte les dames qu'il est possible de connecter. Il n'y aura pas plusieurs lignes qui passent par la même case.

N.B. Certaines des dames ne peuvent avoir aucune connexion et d'autres, plusieurs connexions.

Spécifications d'entrée:

En entrée vous recevrez:

- **board**: un array de strings correspondant au **board** 2D avec des . pour représenter les cases vides et des x pour représenter les dames

Spécifications de sortie:

En sortie vous devrez fournir:

- un array de strings avec des . pour représenter les cases vides, des x pour représenter les dames et les symboles - | / \ pour représenter les connexions

Exemple d'entrée:

```
[ "x.....x",
  ". . . . x . .",
  ". . . . . .",
  ". . . . . .",
  ". . . . . .",
  ". . . . . .",
  ". . . . . .",
  ". . . . . .",
  ". . . . . x" ]
[ ". x . . . x .",
  ". . . . . .",
  ". x . x . . .",
  ". . . . . .",
  ". . . x . . .",
  ". . . . . .",
  ". . . . . .",
  ". . . . . ." ]
```

```
[ "...x.....",
  "...x...",
  ".....x",
  "x.....",
  "...x.....",
  ".....x.",
  ".....x...",
  ".x....."]
```

Exemple de sortie:

```
[ "x-----x",
  ". \ . . x . |",
  ". . \ . . . |",
  ". . . \ . . . |",
  ". . . . \ . . |",
  ". . . . . \ . |",
  ". . . . . \ |",
  ". . . . . x"]
[ ".x---x..",
  ". | \ . / . .",
  ".x-x....",
  ". . \ | . . .",
  ". . .x....",
  ". . . . .",
  ". . . . .",
  ". . . . ."]
[ "...x.....",
  "...x...",
  ".....x",
  "x.....",
  "...x.....",
  ".....x.",
  ".....x...",
  ".x....."]
```

Explication de la première sortie:

La première dame en haut à gauche voit celle en haut à droite comme elles sont sur la même ligne, ainsi que celle en bas à droite, car elles sont sur la même diagonale. La dame en haut à droite voit aussi celle en bas à droite parce qu'elles sont sur la même colonne, mais la dernière dame ne voit aucune des autres donc n'a aucune connexion.

2D3: Fourmis dans les jambes (45 points)

Une colonie de fourmis vient d'aménager dans une nouvelle fourmilière. Les fourmis ouvrières se mettent alors à la recherche de nourriture dans leur nouvel environnement pour permettre de maintenir la colonie en vie. Le nouvel environnement sera représenté par un carré de côté N dans le premier cadran d'un plan cartésien 2D qui partage son centre avec la fourmilière.

Pour créer un réseau de chemins, elles vont commencer par trouver la cache de nourriture la plus proche de la colonie et construire le chemin le plus court possible entre ces deux points. Le réseau comprend maintenant 2 points (la fourmilière et la première cache de nourriture) et 1 chemin. Ensuite, elles iront trouver la cache de nourriture avec la distance la plus petite avec n'importe quel point du réseau existant et créer un chemin entre les deux. Elles continueront ainsi jusqu'à ce que tous les caches de nourriture dans le nouvel environnement fassent partie du réseau de chemins.

Vous recevrez en entrée la longueur N d'un côté du carré qui forme l'aire du nouvel environnement ainsi qu'une liste avec la position des caches de nourriture sous forme de listes de int. Vous devrez donner en sortie une liste des indices (classés par ordre croissant, "f" est en dernier) des points dans L qui sont reliés par un chemin au point qui porte le même indice i dans L , en terminant par les connexions avec la fourmilière, signifiées par le fait que le premier élément de sa liste de connexions est un "f".

Spécifications d'entrée:

En entrée vous recevrez:

- n : un entier
- L : une liste de coordonnées (un tableau 2D d'entiers)

Spécifications de sortie:

En sortie vous devrez fournir:

- une liste des caches connectées à chaque point (un tableau 2D de strings)

Exemple d'entrée:

n : 10

L : $[[1, 2], [3, 4], [5, 6], [7, 8], [9, 1], [2, 3]]$

n : 8

L : $[[4, 5], [5, 6], [6, 7], [2, 3]]$

n : 6

L : $[[1, 4], [5, 3], [5, 2], [2, 1], [4, 4], [3, 1]]$

Exemple de sortie:

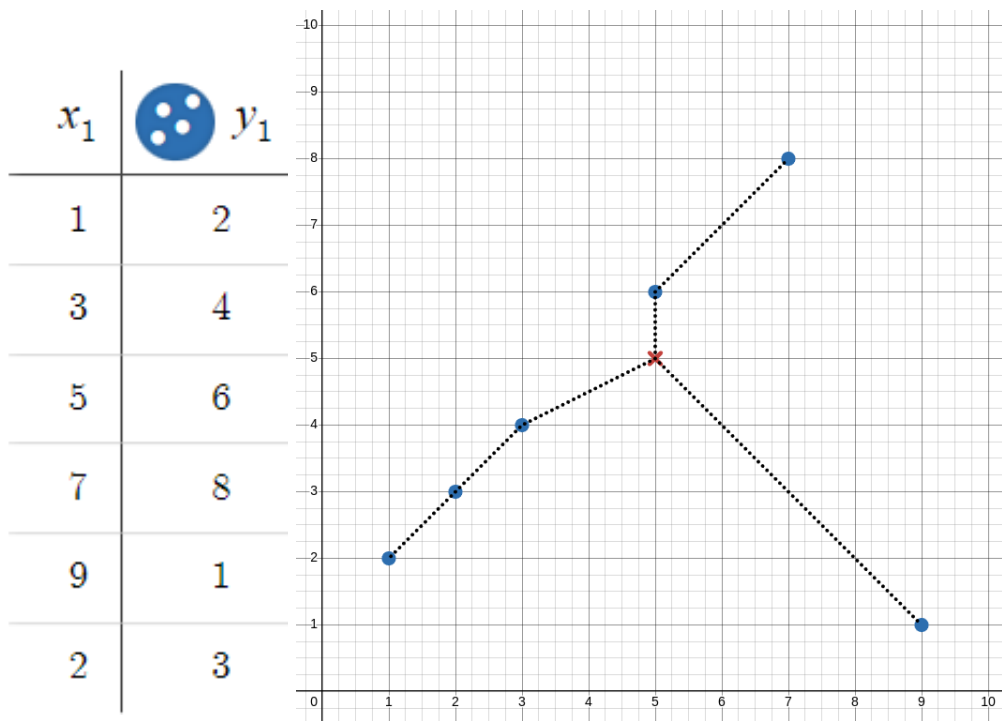
```
[ [ \5' ],  
  [ \5' , \f' , ],  
  [ \3' , \f' ],  
  [ \2' ],  
  [ \f' ],  
  [ \0' , \1' ],  
  [ \f:' , \1' , \2' , \4' ] ]
```

```
[ [ \1' , \f' ],  
  [ \0' , \2' ],  
  [ \1' ],  
  [ \f' ],  
  [ \f:' , \0' , \3' ] ]
```

```
[ [ \f' ],  
  [ \2' , \4' ],  
  [ \1' ],  
  [ \5' ],  
  [ \1' , \f' ],  
  [ \3' , \f' ],  
  [ \f:' , \0' , \4' , \5' ] ]
```

Explication de la première sortie:

Le point (5,6) est le plus proche de la fourmilière, donc on relie les deux. Le point (3,4) est le plus proche par la suite du réseau de la fourmilière donc on fait la connexion la plus courte possible avec, ce qui le relie aussi à la fourmilière. On continue ainsi de suite avec (dans l'ordre) les points (2,3), (1,2), (7,8) et, finalement, (9,1). On doit maintenant donner les connexions aux points dans l'ordre que ceux-ci nous ont été données. Le premier point dans la liste est (1,2), qui est connecté seulement au point (2,3), soit l'indice 5. En répétant la même chose pour le point (3,4), on obtient les connexions à 5 et à la fourmilière ("f"). On continue ainsi de suite jusqu'à la fin de la liste, où on donne les connexions de la fourmilière, précédées par un "f:" dans la liste. (voir l'image ci-dessous pour une visualisation du réseau)

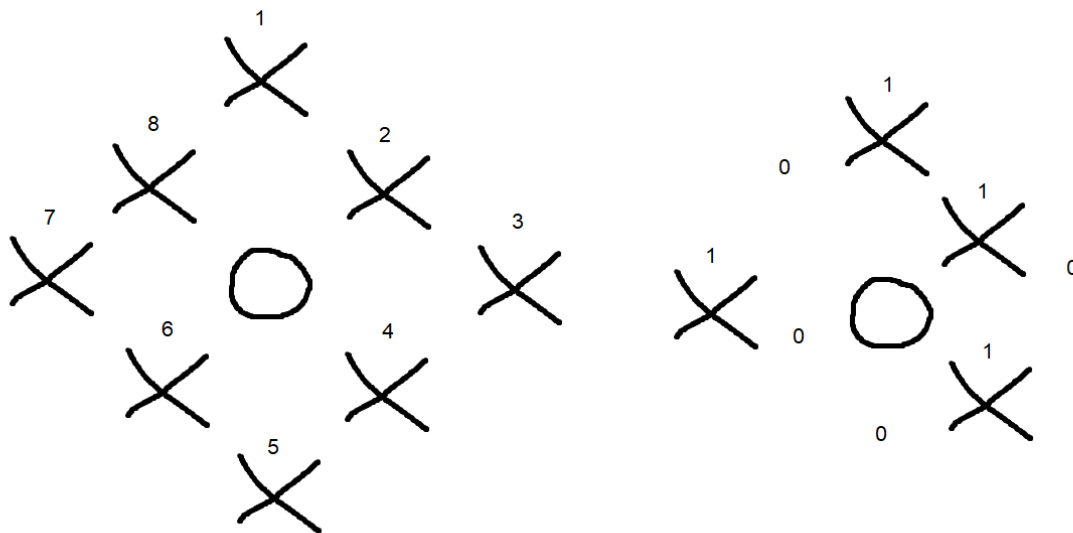


<https://www.desmos.com/calculator/saatdbz9pu?lang=en>

2D4: Mycologie 101 (75 points)

En analysant la reproduction des champignons par libération de spores, on remarque que ces spores respectent un patron régulier. Dans votre laboratoire de recherche, vous tenterez de simuler la reproduction de ces champignons avec un modélisateur de fractales. Cependant, comme l'ordinateur de laboratoire que vous utilisez est très vieux, les paramètres seront soumis sous la forme d'octets pour prendre le moins de mémoire possible. Deux octets seront transmis à votre simulation.

Premièrement, nous assumerons ici que les champignons sont identiques et ne diffèrent que par leur génération, ce qui fait que les directions de libération de spores seront les mêmes pour tous les champignons. Cependant, leur orientation peut évidemment changer; ils ne seront pas tous alignés. **Le premier octet constitue à déterminer si un champignon libère des spores ou non pour chacune des 8 directions radiales d'une "roue":**

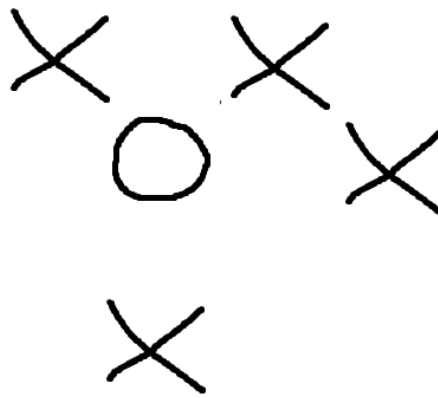


Ici, la première indique les directions possibles de libération de spores ainsi que leur position respective dans l'octet de libération. À droite, on retrouve un exemple d'octet avec son patron de libération. Dans cet exemple, l'octet en entrée serait "11010010". Cet exemple de libération est pour une distance de libération de 2.

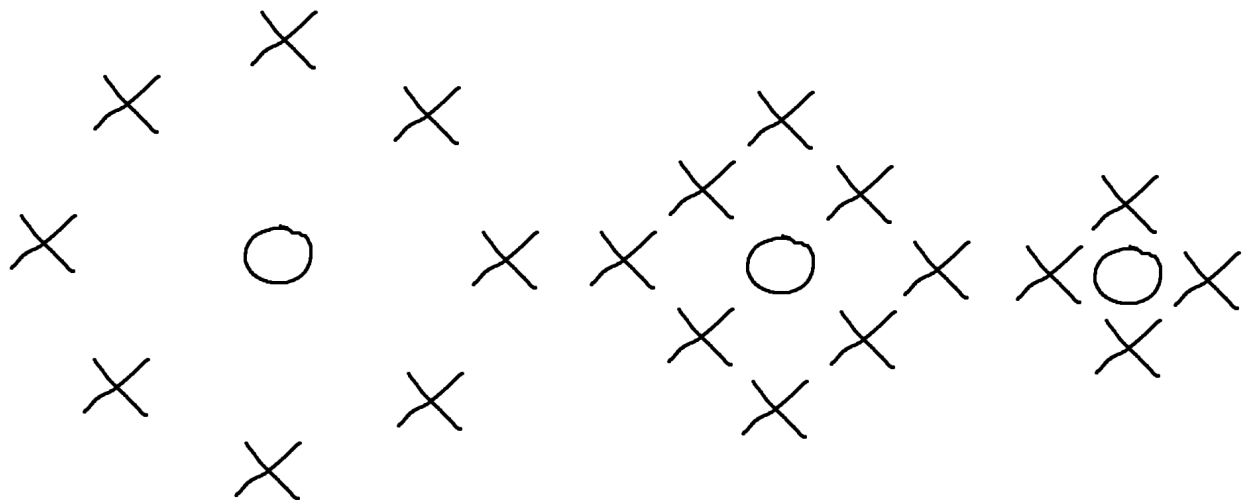
Le deuxième octet, quant à lui, contient tous les autres paramètres quant aux spores de chaque génération:

1	0	0	0	1	0	1	0
Rotation	Sens	Valeur de Rotation		Distance de Libération			

Le bit de rotation correspond à indiquer s'il y a rotation ou non du patron de libération entre les générations. Si la valeur de ce bit est de 0, il faut ignorer les paramètres de sens et de valeur de rotation comme il ne faudra pas faire de rotation. **Le bit de sens** correspond à établir le sens de la rotation du patron. Si la valeur du bit est 0, la rotation se fera dans le sens horaire; si elle est de 1, elle se fera dans le sens antihoraire. **Les trois prochains bits forment un nombre à trois chiffres en binaire, pouvant aller de 0 à 7 en base 10, correspondant aux nombres de rotations de 45 degrés entre les générations dans le sens de rotation donné.** C'est par multiples de 45 degrés comme c'est tous les 45 degrés qu'on retrouve une possible libération de spore selon les règles de notre simulation. Voici le patron ci-dessus tourné après une génération selon les valeurs du bit ci-dessus (rotation dans le sens horaire, de 45 degrés):



Les trois derniers bits sont un autre nombre à trois chiffres en binaire correspondant à la distance de libération initiale des spores. Il est important de considérer que les spores en diagonale sont libérés à une distance de (distanceActuelle-1) dans les deux coordonnées au lieu de simplement être libérés à une distance de distanceActuelle dans une seule coordonnée. Voici les distances de libération 3, 2 et 1 comme exemple, respectivement:



Il ne reste plus que trois dernières spécifications. Pour que le patron se termine, **la distance de libération des spores diminue de 1 à chaque génération jusqu'à atteindre 0** où il n'y a plus de spores. (Oui, ceci veut aussi dire que qu'il n'y a plus de spores en diagonale à une distance de libération de 1). **Tous les champignons présents dans la simulation libèrent des spores à chaque génération, peu importe le moment de leur naissance. Finalement, il faudra que le résultat de votre simulation soit un rectangle parfaitement centré sur le champignon initial (qui sera symbolisé par un "o" au lieu d'un "x", ceci veut dire que les deux dimensions peuvent être différentes, mais logiquement nécessairement impaires avec un "o" en plein centre)** pour effectivement montrer comment les spores se sont éloignés du point initial, soit le centre. Toutes les cases laissées sans champignon à la fin de la simulation seront désignées par un espace vide.

Spécifications d'entrée:

En entrée vous recevrez:

- ***spreadingByte***: une string correspondant à l'octet de libération des spores
- ***parametersByte***: une string contenant tous les autres paramètres nécessaires à l'évolution de la simulation

Spécifications de sortie:

En sortie vous devrez fournir:

- une liste de strings, chaque string correspondant à une rangée de l'affichage final de la simulation, dans le bon ordre évidemment

Exemple d'entrée:

```
spreadingByte: "01010100", parametersByte: "10101011"
spreadingByte: "01010101", parametersByte: "11001010"
spreadingByte: "10001000", parametersByte: "10111011"
spreadingByte: "00100101", parametersByte: "01101101"
```

Exemple de sortie:

```
[ "  x  x  "
  "      "
  "x x x x  "
  "      "
  "  x o x  "
  "      "
  "    x x  "
  "      "
  "      x  "]
[ " x x  "
  "xxxxx"
  " xox  "
```

```

"xxxxxx"
"  x x  "]
["xxx  "
"  xxx  "
"   xxx"
"xxx  "
"  xox  "
"   xxx"
"xxx  "
"  xxx  "
"   xxx"]
["      xx
"      xxxx
"      xxxx  xx
"      xxxxxxxx
"      xxxxxxxxxxxxxx
"      xxxxxxxxxxxxxxxx
"      xx  xxxxxxxxxxxxxxxx
"      xxxxxxxxxxxxxxxxxxxxxx
"      xxxxxxxxxxxxxxxxxxxxxxxx
"      xxxxxxxxxxxxxxxxxxxxxxxx  xx
"      xxxxxxxx  xoxxxxxxxxxxxxxxxxxxxx
"      xxxxxxxxxxxxxxxxxxxxxxxx  xx
"      xxxxxxxxxxxxxxxxxxxxxxxx
"      xxxxxxxxxxxxxxxxxxxxxxxx
"      xx  xxxxxxxxxxxxxxxx
"      xxxxxxxxxxxxxxxx
"      xxxxxxxxxxxxxxxx
"      xxxxx  xx
"      xxxxx
"      xx

```

Explication de la première sortie:

De l'octet de libération on déduit que des spores seront répandus au Nord-Ouest, Nord-Est et Sud-Est de notre champignon initial. L'octet des paramètres nous indique qu'il y a une rotation en sens horaire de 5 fois 45 degrés et que la distance de libération initiale est de 3 cases. Assumant que le champignon initial est à l'origine de notre plan cartésien, la première génération émet des spores en (-2,2), (2,2) et (2,-2). Ensuite, en raison de la rotation, le Nord-Ouest devient le Sud; le Nord-Est, l'Ouest et le Sud-Est, le Nord. Comme la distance diminue à 2 mais que cette fois-ci, les spores sont orthogonaux au lieu de diagonaux, de nouveaux champignons apparaissent à deux cases de distance au Nord, à l'Ouest et au Sud de

tous les champignons existants. À la prochaine et dernière génération, le Nord devient le Sud-Ouest; l'Ouest, le Sud-Est et le Sud, le Nord-Est. Comme tous les points sont diagonaux et que nous sommes à une distance de 1, aucun nouveau spore n'est créé à cette génération. Il ne suffit plus que de s'assurer que le résultat est centré sur le champignon initial en ajoutant deux colonnes vides à droite et le tour est joué!

Note en bas de page:

Amusez-vous avec certaines des valeurs dans les exemples ci-dessus! Vous verrez que le résultat change drastiquement selon les données initiales.

STRUCTURES DE DONNÉES (DS)

DS1: Horizon vertical (15 points)

Vous vous êtes trompé entre l'horizontal et la vertical dans votre tableau deux dimensions. Vous devez, SANS L'AIDE D'UNE LIBRAIRIE, inverser les lignes et les colonnes du tableau et le renvoyer.

Spécifications d'entrée:

En entrée vous recevrez:

- **array**: le tableau à changer les lignes et les colonnes

Spécifications de sortie:

En sortie vous devrez fournir:

- le tableau dont les colonnes et lignes ont été inversées

Exemple d'entrée:

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]  
[[1, 2], [3, 4], [5, 6]]  
[[12, 46, 213, 4123]]  
[["ah", "bé", "cé", "dé"], ["euh", "èf", "jé", "ache"]]
```

Exemple de sortie:

```
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]  
[[1, 3, 5], [2, 4, 6]]  
[[12], [46], [213], [4123]]  
[["ah", "euh"], ["bé", "èf"], ["cé", "jé"], ["dé", "ache"]]
```

Explication de la première sortie:

La première ligne 1, 2, 3 se fait transposer en colonne et devient donc le premier élément de chacune des lignes pour former la même colonne. On peut mettre l'entrée comme:

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

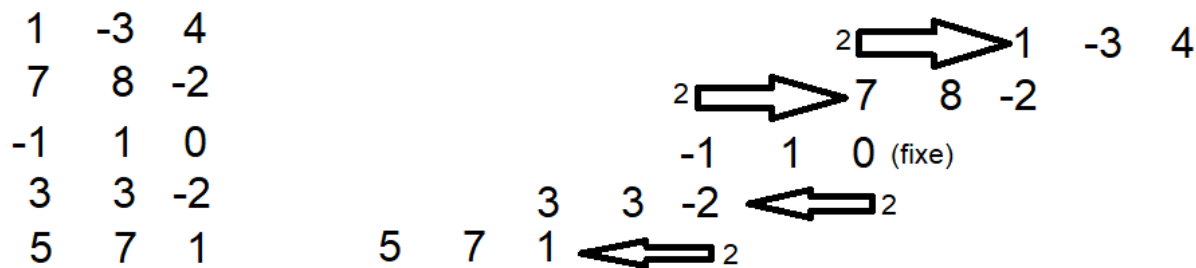
Puis la sortie comme:

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


DS2: Transvection (35 points)


Une transvection est une transformation géométrique qui permet d'étirer et de décaler un objet géométrique comme une image. La grande utilité est lorsqu'on traite de choses discrètes comme des pixels et qu'on veut s'assurer de n'en perdre aucun lors d'une rotation: on peut effectuer 3 transvections pour avoir une rotation sans perte de pixels!

Ici, on ne vous demandera pas de transvecter (peut-être pas un vrai mot) des pixels, mais de transvecter un tableau en deux dimensions afin de calculer une valeur que nous appellerons le "transvecteur". Première étape: la transformation! La transformation consiste à former un parallélogramme à partir d'un rectangle en quelque sorte. Nous prendrons la rangée centrale comme étant fixe. Si la transformation est vers la droite, les rangées en haut se décaleront vers la droite en fonction de leur hauteur et du module de transvection, et les rangées en bas se décaleront vers la gauche en fonction de leur hauteur et du module transvection aussi. Si la transformation est vers la gauche, ce sera l'inverse, évidemment. **Le décalage entre deux rangées est toujours égal au module de transvection.** Voici un exemple avec un module "m" de transvection de 2 vers la droite sur un tableau quelconque:



Chaque rangée est belle et bien décalée de deux par rapport à ses voisins. Comme la rangée centrale est fixe, nous vous donnerons toujours des tableaux avec une hauteur impaire pour garantir l'existence d'une rangée centrale. Il suffit maintenant de calculer le "transvecteur". Nous le définirons comme étant le nombre formé par les sommes des colonnes une fois le tableau 2D transvecté. Voici ce que donnerait le transvecteur toujours en utilisant le même exemple:

								1	-3	4
						7	8	-2		
				-1	1	0				
		3	3	-2						
+	5	7	1							
	5	7	4	3	-3	1	7	8	-1	-3
										4


=57437178974

Il est important de remarquer que si la somme d'une colonne est négative, nous remplacerons les chiffres de la somme par leur inverse et retirerons le négatif par la suite, comme nous avons fait ici. L'inverse "x" d'un chiffre "c" sera défini ici comme le chiffre pour lequel $c + x = 10$. Ainsi, l'inverse de 3 est 7 et l'inverse de 1 est 9, conformément à l'exemple ci-dessus. Il est important aussi de mentionner que même si ce n'est pas le cas ici, des sommes peuvent résulter en des nombres à plusieurs chiffres qui seront concaténés dans le transvecteur tout comme de simples chiffres. Si le nombre à plusieurs chiffres est aussi négatif, il faudra remplacer chacun des chiffres par leur inverse. Par exemple, "-583" devient "527". La direction de la transvection (indiquée par une lettre "D" ou "G") et son module seront joints dans le même paramètre sous la forme d'une string. Il n'y aura jamais de colonnes vides après transformation.

Spécifications d'entrée:

En entrée vous recevrez:

- **tableau**: un tableau en deux dimensions contenant des entiers aléatoires
- **mode**: une string contenant le module de transvection (un entier de 0 à 9) et la direction de la transvection ("D" ou "G")

Spécifications de sortie:

En sortie vous devrez fournir:

- un entier correspondant au transvecteur décrit plus haut

Exemple d'entrée:

tableau: `[[1,-3,4],[7,8,-2],[-1,1,0],[3,3,-2],[5,7,1]]`, **mode**: "2D"

tableau: `[[1,-3,4],[7,8,-2],[-1,1,0],[3,3,-2],[5,7,1]]`, **mode**: "2G"

tableau: `[[37,-43,12,8,126],[42,-24,37,53,75],[122,7,0,-35,-88]]`,

mode: "3G"

```
tableau: [[1,-1],[0,0],[-1,1],[0,0],[1,-1],[2,-2],[3,-3],[4,-4],[  
5,-5]], mode: "1D"  
tableau: [[1,2,3],[4,5,6],[7,8,9]], mode: "0D"
```

Exemple de sortie:

```
57437178974  
171187133371  
37671250102371758207522  
5999999119  
121518
```

Explication de la première sortie:

Voir description du problème pour une explication détaillée de la première sortie.

Note en bas de page:

Allez voir la vidéo de Matt Parker sur le sujet:

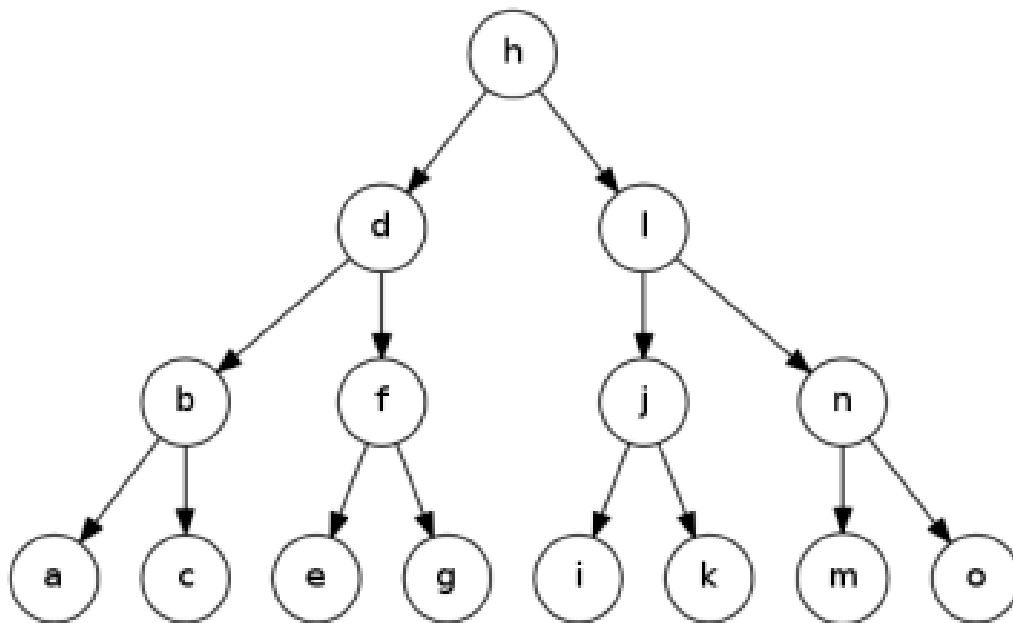
<https://www.youtube.com/watch?v=1LCEiVDHJmc> !

(Elle risque de très peu vous aider sur ce problème au cas où vous cherchiez de l'aide)

DS3: Parcours d'arbre (45 points)

Les arbres sont une structure de données importante et trop souvent oubliée en programmation. Le concept d'un arbre est d'installer les données dans une hiérarchie pour aller les explorer plus rapidement sans trop demander d'espace mémoire. Dans un arbre, nous avons des noeuds, qui ont des noeuds enfants. Dans les arbres que nous allons utiliser, le nombre de noeuds enfants maximum sera toujours de 2, et ce seront des arbres parfaits. Ceci veut dire que tous les niveaux contenant des enfants auront exactement 2 enfants pour chaque nœud du niveau. Vous devrez parcourir l'arbre comme vu de gauche à droite. Allez voir l'explication de la première sortie pour en savoir plus. Vous aurez l'arbre donné étage par étage. Ainsi le premier array ne contiendra qu'un seul élément, le second array en contiendra 2, l'étage suivant aura 4 éléments puis 8 et ainsi de suite.

Vous devrez parcourir l'arbre en ordre infixe et retourner la concaténation de ces éléments dans une string. Le parcours en ordre infixe est de prendre l'élément le plus à gauche, suivi de son parent et par la suite l'élément de droite.



Spécifications d'entrée:

En entrée vous recevrez:

- **depth:** un int représentant le nombre de niveaux de l'arbre
- **tree:** un array ayant un nombre variable de string à chaque étage (2x plus que l'étage précédent)

Spécifications de sortie:

En sortie vous devrez fournir:

- une string de tous les éléments comme vu de gauche à droite

Exemple d'entrée:

```
[['h'],  
['d', 'l'],  
['b', 'f', 'j', 'n'],  
['a', 'c', 'e', 'g', 'i', 'k', 'm', 'o']]
```

```
[['t'],  
['R', 's'],  
['C', 'C', 'e', 't']]
```

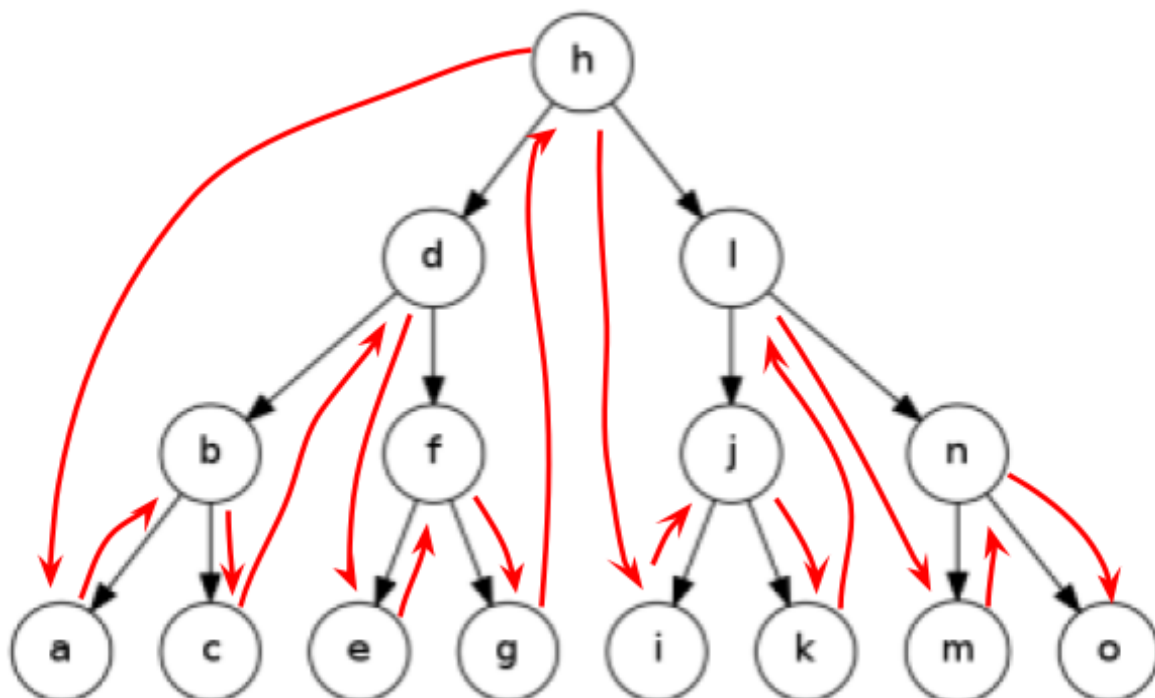
```
[['t'],  
[' ', 'b'],  
['R', 's', 'e', 's'],  
['C', 'C', 'i', ' ', 'h', ' ', 'e', 't']]
```

Exemple de sortie:

```
abcdefghijklmno  
CRCtest  
CRC is the best
```

Explication de la première sortie:

Pour afficher un arbre en ordre infixe, on va chercher le nœud le plus à gauche puis son parent et après le nœud à sa droite. Lorsqu'on atteint un nœud, on commence par regarder s'il a des enfants et on explore la portion gauche en premier. On commence ce parcours à la racine de l'arbre, à l'élément h. Vous pouvez visualiser le chemin à suivre dans un parcours infixe à l'aide des flèches rouges.



CALCULS SCIENTIFIQUES (SC)

SC1: Cercle carré (25 points)

Vous devez calculer le rapport entre l'aire d'un cercle et l'aire d'un carré. Par contre, le cercle peut être plus grand ou plus petit que le carré. Les deux seront centrés au même endroit. Comme vous ne voulez pas faire des maths pures, vous utilisez une approche statistique. Vous devrez générer des points aléatoirement distribués dans le carré et voir combien d'entre eux sont aussi dans le cercle.

Le nombre de chiffres n'est pas limité ; essayez d'être aussi proche que possible de la réponse, bien qu'il soit peu probable que vous soyez dans le mille.

Spécifications d'entrée:

En entrée vous recevrez:

- **rayon**: le rayon du cercle
- **L**: la longueur d'un côté du carré
- **nbrPoints**: le nombre de réalisations que vous allez devoir générer pour trouver le ratio de l'aire du carré couverte par le cercle

Spécifications de sortie:

En sortie vous devrez fournir:

- un float de la fraction du carré couvert par le cercle

Exemple d'entrée:

```
rayon: 2      L: 5      nbrPoints: 10 000
rayon: 3      L: 4      nbrPoints: 100 000
rayon: 2      L: 4      nbrPoints: 10 000
```

Exemple de sortie:

```
0.503061
1.0
0.785223
```

Explication de la première sortie:

Pour le premier exemple on a un cercle de rayon 2 centré en (0, 0) et un carré 5x5 centré en (0, 0) aussi donc les coins sont à (-2.5, 2.5) (-2.5, -2.5) (2.5, -2.5) (2.5, 2.5) On génère 10 000 fois un point qui se situe dans le carré et on compte le nombre de fois où le point généré se trouve aussi dans le cercle. Dans notre cas nous avons 0.503561 comme pourcentage du carré couvert par le cercle. En effet, ceci s'approche de la valeur théorique calculée par la formule suivante:

$$\frac{\text{rayon}^2 * \pi}{L^2} = \frac{2^2 * \pi}{5^2} = \frac{4\pi}{25} \approx 0.502654825$$

SC2: Balancez la balance! (65 points)

On n'a pas toujours eu des balances numériques hyper précises: avant, on déterminait le poids des objets qu'on voulait peser en les comparant à des poids qu'on connaissait déjà. Ici, un marchand ne possède qu'une très vieille balance avec une série de petits poids et il veut déterminer le poids de certains objets qu'il vend selon le kilogramme. On veut savoir de combien de façons il est possible pour lui de se rendre au contrepoids désiré.

Par exemple, considérons que l'objet que le marchand essaie de peser totalise 12 kilos et que le marchand ne possède que des poids de 4 et 6 kilos. Le marchand aura pesé l'objet lorsqu'il sera arrivé à avoir mis 3 poids de 4 kilos sur la balance ou 2 poids de 6 kilos sur la balance.

Vous devez donner toutes les combinaisons possibles pour y arriver. Les combinaisons seront classées par ordre décroissant de poids et les poids seront classés par ordre décroissant au sein d'une même combinaison. (Exemple: la combinaison [6, 4, 2] se placerait devant la combinaison [4, 4, 4], mais derrière la combinaison [6, 6]. [2, 4, 6] est la même combinaison que [6, 4, 2], même s'il y a plusieurs d'un certain type de poids et sera ainsi omise.)

Spécifications d'entrée:

En entrée vous recevrez:

- **poids**: un array d'entiers
- **contrepoids**: un entier

Spécifications de sortie:

En sortie vous devrez fournir:

- un array composé d'arrays de int

Exemple d'entrée:

poids: [1, 2, 5, 6, 3, 12, 24, 2, 2, 4, 9, 10] contrepoids: 22

poids: [1, 3, 5, 7, 9] contrepoids: 25

poids: [14, 31, 17, 50, 7, 9, 11] contrepoids: 40

poids: [7, 8, 9, 12, 13, 13, 5, 6] contrepoids: 26

Exemple de sortie:

```
[[12, 10],[12, 9, 1],[12, 6, 4],[12, 6, 3, 1],[12, 6, 2, 2],[12, 5, 4, 1],[12, 5, 3, 2],[12, 5, 2, 2, 1],[12, 4, 3, 2, 1],[12, 4, 2, 2, 2],[12, 3, 2, 2, 2, 1],[10, 9, 3],[10, 9, 2, 1],[10, 6, 5, 1],[10, 6, 4, 2],[10, 6, 3, 2, 1],[10, 6, 2, 2, 2],[10, 5, 4, 3],[10, 5, 4, 2, 1],[10, 5, 3, 2, 2],[10, 5, 2, 2, 2, 1],[10, 4, 3, 2, 2, 1],[9, 6, 5, 2],[9, 6, 4, 3],[9, 6, 4, 2, 1],[9, 6, 3, 2, 2],[9, 6, 2, 2, 2, 1],[9, 5, 4, 3, 1],[9, 5, 4, 2, 2],[9, 5,
```

```
3, 2, 2, 1],[9, 4, 3, 2, 2, 2],[6, 5, 4, 3, 2, 2],[6, 5, 4, 2,
2, 2, 1]]
[[9, 7, 5, 3, 1]]
[[31, 9],[17, 14, 9]]
[[13, 13],[13, 8, 5],[13, 7, 6],[12, 9, 5],[12, 8, 6],[8, 7, 6,
5]]
```

Explication de la première sortie:

En commençant par le poids le plus lourd à notre disposition, soit 24, on voit bien qu'il est trop lourd pour balancer le contrepoids. Ensuite, on continue à 12, qui peut se combiner à 10 pour donner 22, mais aussi à de nombreuses autres combinaisons. La liste des possibilités s'ensuit par ordre décroissant. Le dernier point d'intérêt à remarquer est que la présence de trois poids de 2 kilos dans ce problème nous permet d'effectuer des combinaisons où on répète le poids de 2 kilos jusqu'à 3 fois justement.

SC3: Fractionnellement Additif (35 points)

L'addition de fractions est une opération pas toujours simplement réalisable. C'est plus difficile à imaginer pour nos cerveaux qu'une simple addition d'entiers. Par exemple, si on veut connaître toutes les sommes d'entiers possible menant à 14, c'est simple: il suffit de commencer au milieu et de s'éloigner de par bons de 1 pour obtenir plus de possibilités. Ainsi, les sommes possibles seraient entre autres 7+7, 6+8, 5+9, etc.

On ne peut pas en dire de même pour la somme de deux fractions par contre. Prenons $\frac{2}{85}$, une somme évidente à deux termes est $\frac{1}{85} + \frac{1}{85}$, mais $\frac{1}{84} + \frac{1}{86}$ ne fonctionne pas nécessairement, même que $\frac{1}{84} + \frac{1}{86} = \frac{85}{3612}$. Comme c'est difficile de calculer de telles paires de fractions mentalement, utiliser un ordinateur est fort utile ici. On veut donc calculer toutes les solutions possibles de la forme:

$$\frac{1}{x} + \frac{1}{y} = \frac{2}{z}$$

où x est différent de y pour exclure la solution $x=y=z$. Pour un z donné, vous devrez donc calculer toutes les sommes possibles et retourner la somme $x+y$ associée à chaque paire trouvée.

Spécifications d'entrée:

En entrée vous recevrez:

- z: un entier correspondant au dénominateur de la fraction qu'on cherche à résoudre

Spécifications de sortie:

En sortie vous devrez fournir:

- un array d'entiers $x+y$ correspondant à toutes les solutions possibles au problème, classées par ordre décroissant

Exemple d'entrée:

9
37
179
4279
888

Exemple de sortie:

[50, 24]
[722]
[16200]
[9159200, 836550, 80000, 28008]
[198025, 99458, 66603, 50176, 33750, 25538, 22801, 17328, 13225, 11858,
9126, 6400, 6253, 5043, 3698, 3626, 2775, 2401, 2368, 1998, 1850, 1813]

Explication de la première sortie:

Pour le premier dénominateur 9, les deux résultats trouvés sont $1/5 + 1/45 = 10/45 = 2/9$ et $1/6 + 1/18 = 4/18 = 2/9$. Si on additionne les paires de dénominateurs, on obtient 50 et 24, respectivement. En s'assurant qu'ils sont en ordre décroissant, on obtient ainsi la première sortie du problème.

Note en bas de page:

Le nombre de solutions possibles pour un entier z semble relativement relié au nombre de facteurs que cet entier possède. Plus spécifiquement encore, les nombres premiers ne semblent posséder qu'une seule solution avec une très grande fraction et une bien plus petite fraction. Pouvez-vous prouver ou réfuter ceci?

TRAITEMENT DE TEXTE (TP)

TP1: Réglez les règles (15 points)

Vous avez lu les règlements de la compétition de programmation de la CRC mais vous ne comprenez pas parce qu'il y a beaucoup trop d'erreurs qui dérangent votre lecture. Vous devez enlever les majuscules qui ne sont pas en début de mot et en ajouter au début de chaque phrase. Vous devez aussi vous assurer que le nom CRC est toujours écrit avec les trois lettres en majuscule! Comme ce sont des règles très strictes, toutes les phrases doivent se compléter avec un point d'exclamation.

Spécifications d'entrée:

En entrée vous recevrez:

- **text**: le texte que vous devrez corriger

Spécifications de sortie:

En sortie vous devrez fournir:

- une string du texte corrigé selon les règles de correction

Exemple d'entrée:

pour la compétiTion de la cRc de ceTTe année voUs devez écrire
en pYthon. Vous aLLez voir c'eST FAcile comme lanGage?
FaUTe de GRAND-MÈRE.
cRc.
TroP. De? pOINts! OUI Peut-ÊTRE.

Exemple de sortie:

Pour la compétition de la CRC de cette année vous devez écrire
en python! Vous allez voir c'est Facile comme langage!
Faute de Grand-mère!
CRC!
Trop! De! Points! Oui Peut-être!

Explication de la première sortie:

Pour la première sortie, la première correction faite est sur la majuscule de début de phrase qui est absente. Dans le mot compétition les lettres sont mises en minuscule. Pour CRC, toutes les lettres sont mises en majuscule. Par la suite, la prochaine correction est de remplacer le point de fin de phrase par un point d'exclamation. Finalement, la dernière nuance à préciser est que le mot facile garde son F majuscule, car elle se trouve au début du mot. Le point d'interrogation devra aussi être remplacé par un point d'exclamation.

TP2: Zipper (25 points)

Vous recevrez un message crypté par la méthode de cryptage de zipper. Le cryptage en zipper est de séparer le message caractère par caractère en deux strings. Vous recevrez donc deux strings sans savoir laquelle des deux est la première et laquelle est la seconde. Vous devez recomposer le message dans le bon ordre.

Vous devez trouver une logique pour être toujours capable de savoir dans quel ordre organiser les strings automatiquement. Les messages que vous recevrez seront toujours juste en minuscule.

Spécifications d'entrée:

En entrée vous recevrez:

- **s1**: une des deux portions du zipper sans ordre spécifique
- **s2**: une des deux portions du zipper sans ordre spécifique

Spécifications de sortie:

En sortie vous devrez fournir:

- la string recomposée dans le bon ordre

Exemple d'entrée:

```
s1: h sti optto okytk  
s2: wyi hscmeiins rpi?
```

```
s1: l optto epormaind accetvamn u!  
s2: acmeiind rgamto el r s rietfn
```

```
s1: ial iprmrei o hthr!  
s2: fnlyzpe eg snтта ad
```

Exemple de sortie:

```
why is this competition so kryptik?  
la competition de programmation de la crc est vraiment fun!  
finally zipper merge is not that hard!
```

Explication de la première sortie:

Pour la première sortie, nous avons une des deux string qui est plus longue que l'autre (s2). On peut donc deviner qu'on commence et termine le zipper par la string s2. Ainsi on prend le premier caractère de la string s2 w puis celui de s1 h et s2 y qui donnent un après l'autre le mot why. On continue l'alternance jusqu'à la fin de la des deux strings. Ceci nous donne le message: why is this competition so kryptik?

TP3: Palindrôle (30 points)

En essayant de créer un nouveau format de compression de fichiers, Mr. C se penche longuement sur un type de mot avec des propriétés intéressantes: les [palindromes](#). Il remarque que certains mots, comme ‘kayak’, comprennent une symétrie et que d’autres contiennent une récursion de symétrie, comme ‘kayak-à-kayak’. Il qualifie la quantité de récursion à l’intérieur d’un mot comme étant l’ordre du palindrome. Un mot qui ne contient pas de symétrie est un palindrome d’ordre 0 (ex.: ‘soulier’) et un mot qui n’a qu’une seule symétrie est un palindrome d’ordre 1 (ex.: ‘radar’). Pour savoir si un mot est un palindrome d’ordre supérieur à 1, nous nous retrouvons avec 2 manières de déconstruire le mot en fonction du nombre de lettres qu’il contient. Si le mot contient un nombre pair de lettres, on le sépare simplement en deux en son milieu et on évalue les deux nouveaux mots résultants. Si le mot contient un nombre impair de lettres, on le sépare encore en deux mots résultants, mais on n’inclut pas la lettre au milieu du mot. Par exemple, ‘miaim’ devient ‘mi’ et ‘im’. Pour que le mot initial soit d’ordre 2, les deux mots résultants doivent être les mêmes et être des palindromes eux-mêmes.

Vous recevrez en entrée une liste de strings. Comme sortie, vous donnerez une liste de int avec l’ordre de palindrome des mots dans la liste obtenue en entrée.

Spécifications d’entrée:

En entrée vous recevrez:

- *L*: une liste de strings

Spécifications de sortie:

En sortie vous devrez fournir:

- un entier

Exemple d’entrée:

```
laval
cook
wwwwwwwww
oborobo
```

Exemple de sortie:

```
1
0
3
2
```

Explication de la première sortie:

“laval” est un palindrome de niveau 1 comme ses deux moitiés sont symétriques quand on le sépare en deux par le milieu. Cependant, les moitiés restantes (“al” et “la”) ne sont pas des palindromes en eux-mêmes par la suite. Ce n’est donc pas un palindrome de niveau 2.