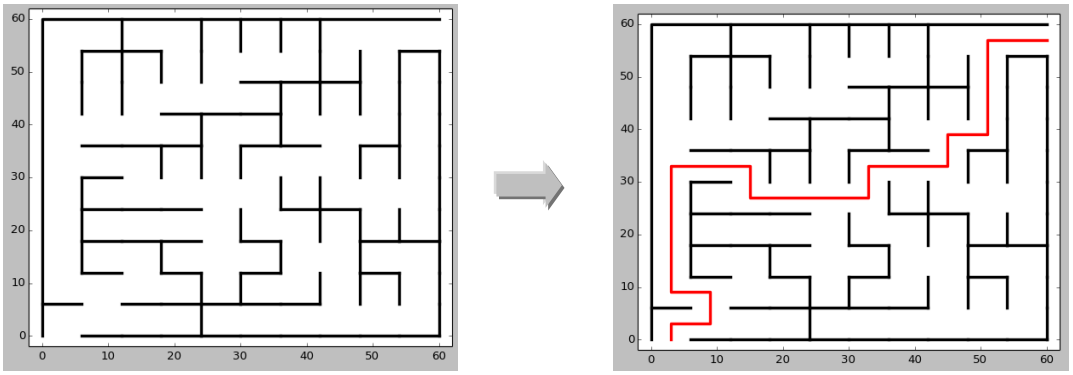
MINI PROJET LIBRE
LABYRINTHE

Ceci est **un devoir libre**, vous pouvez **si souhaitez que je regarde votre travail ou que je corrige certaines choses, déposez vos fichiers sur le site www.prepabellevue.org dans votre rubrique Informatique/dépôt de fichier**

Pensez à mettre votre nom sur vos fichiers si vous les déposez.

Je vous conseille enfin de travailler en équipe (2/3)

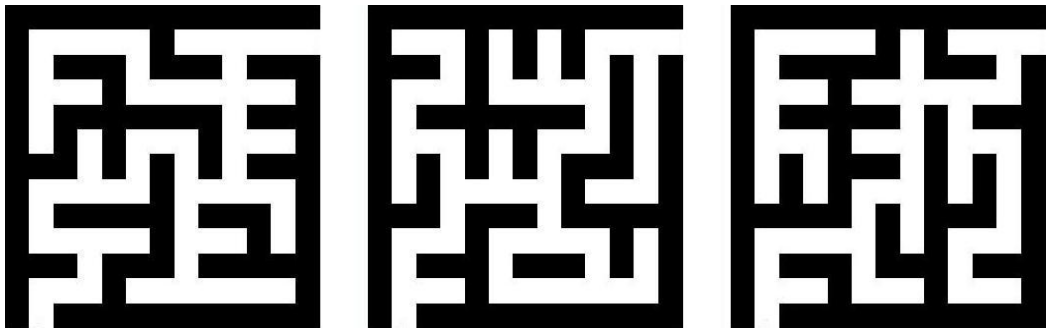
On souhaite ici représenter un labyrinthe parfait et la solution qui permet d'aller de la sortie à l'entrée …



Nous considérerons qu'un labyrinthe est un tableau en deux dimensions, dont les cases contiguës sont séparées ou non par un mur (appelé aussi porte). De plus, deux cases particulières du tableau sont choisies, et appelées entrée et sortie. Il s'agit en général de deux coins opposés.

Un labyrinthe est dit parfait s'il a la propriété suivante : étant données deux de ses cases, il existe un et un seul chemin reliant ces deux cases. En d'autres termes, toutes les cases peuvent être atteintes à partir de l'entrée, et ceci par un unique chemin.

Q1.Parmi les exemples de labyrinthes ci-dessous, lesquels sont parfaits ? Pourquoi ?



1. DIMENSIONS

Q2.Ecrire d'abord les lignes qui permettent à l'opérateur de renseigner la taille du labyrinthe :

*On nommera **m** sa largeur et **n** sa longueur*

*Q3.Créer la fonction **Verif(m,n)** qui vérifie si l'opérateur a bien renseigné une valeur à m et n et sinon, qui attribue à m la valeur 8 et à n la valeur 6*

2. CREATION

Importer tout d'abord toutes les bibliothèques qui seront utiles :

```
from numpy import *
from random import randint
from time import clock,time
from matplotlib.pyplot import *
```

Voici l'algorithme de *fusion aléatoire de chemins,* que vous allez programmer.

1. Au début toutes les portes sont fermées.
2. On numérote toutes les cases du labyrinthe de chiffres différents (de 0 à m*n)
3. On répète la procédure suivante :
   (a) On choisit une case au hasard, que nous appelons ici *A.*
   (b) On localise toutes les cases contiguës qui ne portent pas le même numéro que la case *A.* On les notes $B_1 \dots B_k$, où *k* est un entier compris entre 0 et 4.
   (c) On choisit au hasard une case parmi les cases $B_1 \dots B_k$, que nous appelons *B.*
   (d) On ouvre la porte entre *A et B.*
   (e) On affecte à *A et B* le même numéro *N*. Ce numéro sera le plus petit numéro entre celui de *A* et celui de *B*. Toutes les cases portant l'un de ces numéros recevra elle aussi le numéro *N*.

   On répète cette procédure jusqu'à ce que toutes les cases soient numérotées 1. Alors elles sont toutes reliées, et donc le labyrinthe est créé, il est bien parfait.

Le labyrinthe sera modélisé par une matrice de taille (m,n). La case de coordonnées (0,0) sera la case en haut à gauche, et donc la case de coordonnées (m,n) sera la case en bas à droite.

Partie 1 :

Q4.Recopier les lignes suivantes et observer ce qui est contenu dans MS et ME. Quelle est leur taille ?

```
MS=reshape(m*n*[True],[n,m]) # murs sud
ME=reshape(m*n*[True],[n,m]) # murs est
MS[n-1,0]=False # entrée
ME[0,m-1]=False # sortie
```

Q5.Pourquoi ne pas créer les murs Nord et Ouest ?

Il faudra veiller à ce que les portes sud des cases de coordonnées (m, j), c'est-à-dire les cases les plus au sud, soient toujours fermées, afin que le tour du labyrinthe soit toujours fermé. De même les portes est des cases de coordonnées (i, n), c'est-à-dire les cases les plus à droite, devront rester fermées.

Par contre on ouvrira la porte sud de la case (m, 1) et la porte est de la case (1, n), qui seront l'entrée et la sortie du labyrinthe.

Partie 2 :

*Q6.Créer le tableau auxiliaire de taille m*n que vous nommerez T qui contient les valeurs allant de 0 à m*n+1.*

Par exemple pour m=8 et n=6 :

```
T=array([[ 0, 1, 2, 3, 4, 5, 6, 7],
[ 8, 9, 10, 11, 12, 13, 14, 15],
[16, 17, 18, 19, 20, 21, 22, 23],
[24, 25, 26, 27, 28, 29, 30, 31],
[32, 33, 34, 35, 36, 37, 38, 39],
[40, 41, 42, 43, 44, 45, 46, 47]])
```

Partie 3 :

*Q7. Créer une liste de portes ouvrables, que vous appellerez **LPO**, contiguës à une case tirée au hasard.*

*Q8. Tirer au sort la porte qui va être créée dans cette liste LPO tout en gardant la valeur de T correspondant à cette porte ouverte (que l'on appellera **a**)*

*Q9. Modifier alors la liste T en donnant le plus petit numéro **N** entre celui de **a** et la case tirée au hasard. Toutes les cases portant le numéro max entre **a** et la case tirée au hasard recevra elle aussi le numéro **N**.*

*Q10. Réitérer la procédure à l'aide d'une boucle **while** sur la valeur max de T*

3. AFFICHAGE DU LABYRINTHE

On choisira dans un premier temps les variables de couleurs et d'épaisseurs suivantes :

```
em=3 # épaisseur des murs
ec=3 # épaisseur de couloirs, doit être impaire
et=em+ec
cm='k' # couleur des murs (noire)
cc='w' # couleur des couloirs (blanc)
```

*Q11. Créer la fonction **MurS(i,j)** qui trace le mur sud (la droite horizontale) entre les points i et j avec l'épaisseur de trait em.*

On considère que m et n sont connus mais vous pouvez les mettre comme arguments supplémentaires à votre fonction

*Q12. Créer la fonction **MurE(i,j)** qui trace le mur est (la droite verticale) entre les points i et j avec l'épaisseur de trait em.*

On considère que m et n sont connus mais vous pouvez les mettre comme arguments supplémentaires à votre fonction

*Q13. Créer la fonction **Affichage(S,E)** qui affiche le labyrinthe modélisé par ses murs Sud et Est*

On considère que m et n sont connus mais vous pouvez les mettre comme arguments supplémentaires à votre fonction
Pour rendre l'affichage plus esthétique, je vous conseille d'utiliser
*axis([-ec//2,m*et+ec//2+1,-ec//2,n*et+ec//2+1],'scaled','off')*
et de tracer une bordure extérieure nord et ouest.

4. RESOLUTION

Deux algorithmes sont possibles. Le premier est l'algorithme intuitif qu'utiliserait un personnage cherchant la sortie du labyrinthe, il est plus difficile à programmer.

Le second est impossible à faire pour un personnage cherchant la sortie dans le labyrinthe, mais beaucoup plus facile à programmer.

4.1. Algorithme pratique

Pour sortir d'un labyrinthe parfait, il suffit de toujours tourner à droite (ou, au choix, toujours tourner à gauche). En d'autres termes le personnage mets sa main contre le mur de droite et il ne le quitte jamais.

Q14. Pourquoi cet algorithme peut-il ne pas convenir si labyrinthe n'est pas parfait ?

4.2. Algorithme théorique

On affecte le numéro 1 à l'entrée du labyrinthe, et 0 à toutes les autres cases. A toutes les cases atteignables depuis la case 1 en un mouvement on affecte le numéro 2. Puis on affecte le numéro 3 à toutes les cases portant le numéro 0 et atteignables depuis une case 2 en un mouvement. Et ainsi de suite, **jusqu'à ce qu'il n'y ait plus de numéro 0**. Alors on part de la sortie, et on rejoint l'entrée en passant par les cases atteignables dont le numéro descend d'une unité à chaque mouvement. On a ainsi le trajet de l'entrée à la sortie.

Par exemple si la sortie porte le numéro 37, alors on cherche la case atteignable en un mouvement portant le numéro 36, puis la case atteignable en un mouvement portant le numéro 35, et ainsi de suite.

*Q15. Créer le tableau auxiliaire de taille m*n que vous nommerez S qui contient des 0. Attention à bien le retourner …*

Q16. Remplir alors ce tableau S en suivant l'algorithme 2 présenté ci-dessus et en fonction des rencontres des murs Est et Sud.

5. AFFICHAGE DE LA SOLUTION

*Q17. Créer une fonction **AffichageSol(S)** qui affiche la solution !*

Travailler avec une liste X et une liste Y qui enregistre les positions x et y validées à partir des autres valeurs de S et des murs voisins.

Rappel :
em=3 # épaisseur des murs
ec=3 # épaisseur de couloirs, doit être impaire
et=em+ec #on utilisera donc et//2 pour se placer au milieu du couloir

Q18. Afficher enfin le labyrinthe puis la solution en rouge …

6. EVALUER LE TEMPS

Q19. Ajouter une évaluation du temps de création de votre quadrillage : TC

Et du temps de résolution :TS
Utiliser la fonction clock() ou la fonction time()