Projet Math-Info

LABYRINTHES

WENG Julien
BERTINE Thomas
LEMOINE Léandre

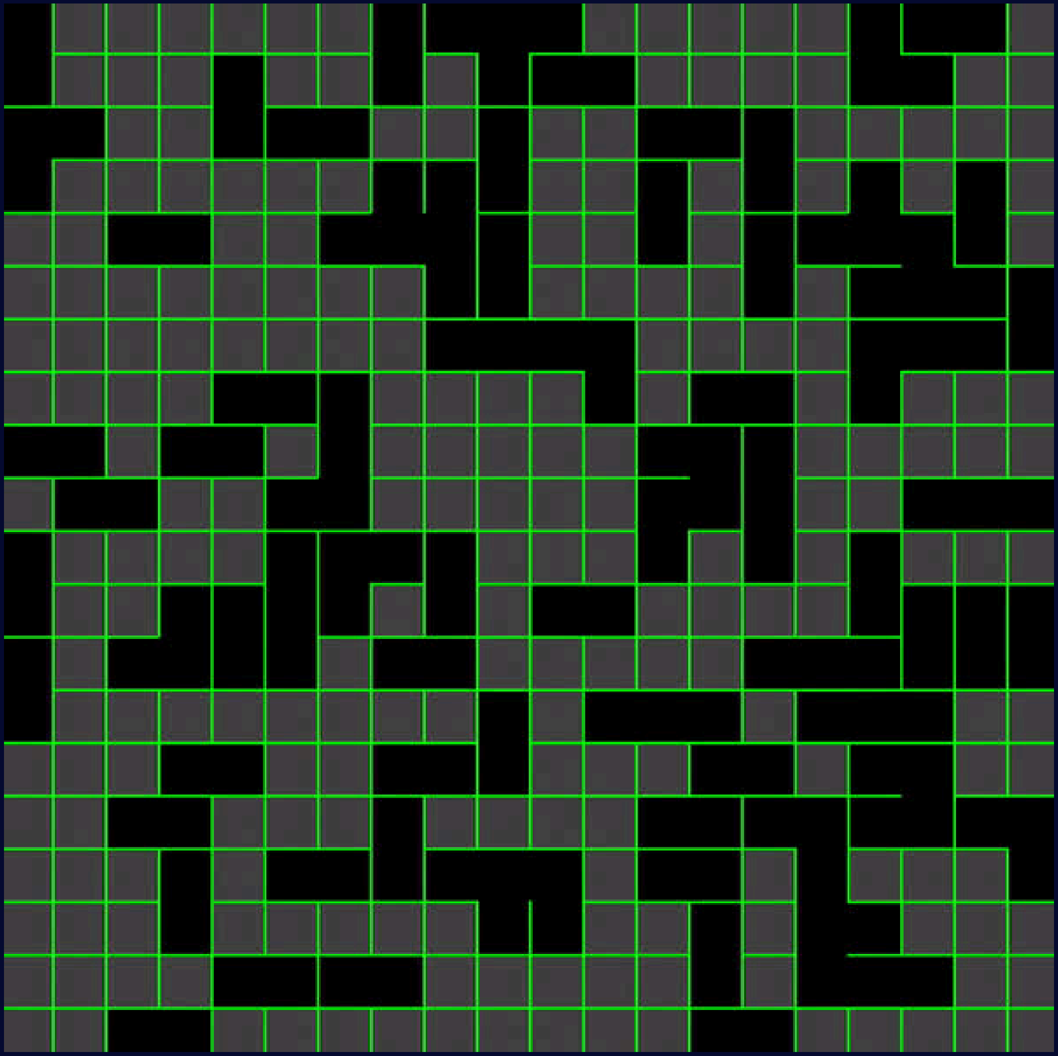
PRÉSENTATION DU SUJET :

Problèmes traités lors de ce projet :

- Algorithme testant leurs validité
- Génération de “vrais” labyrinthes
- Résolution des labyrinthes
- Affichage des labyrinthes
- Énumérations des labyrinthes
- Uniformité de génération

Génération des labyrinthes :

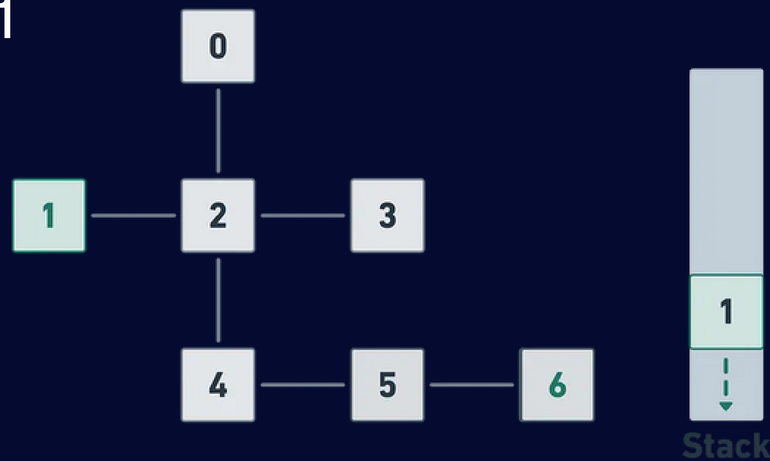
Algorithme de Kruskal:



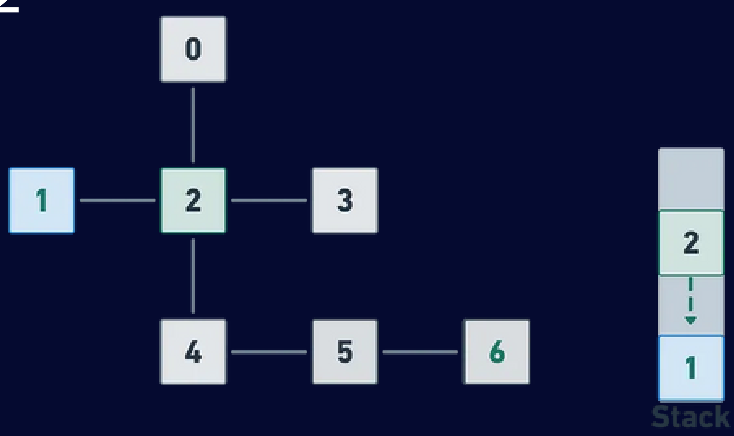
Lien vers l'animation :

Résolution du labyrinthe à l’aide de l’algorithme DFS

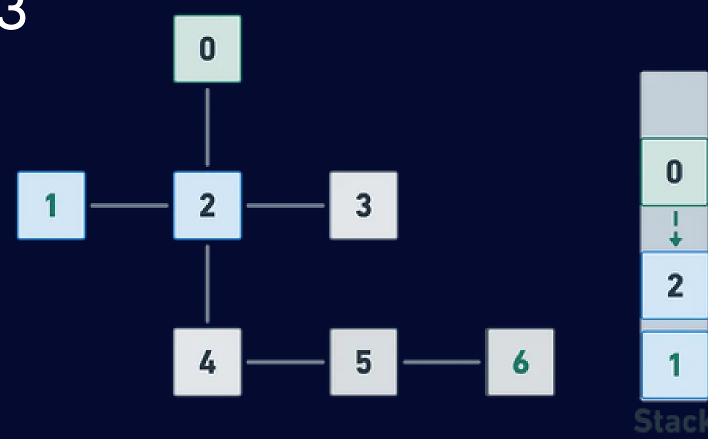
1



2

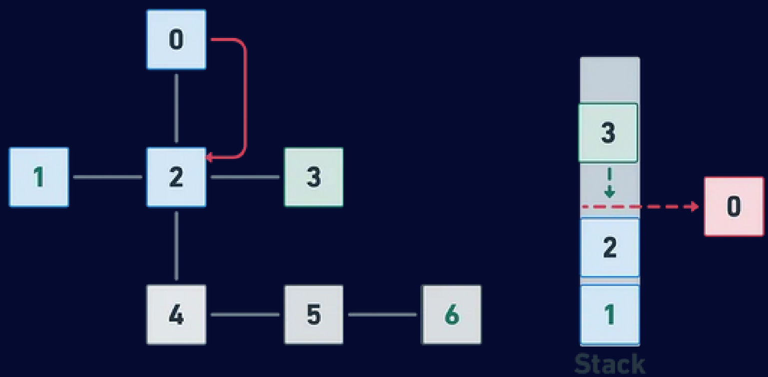


3

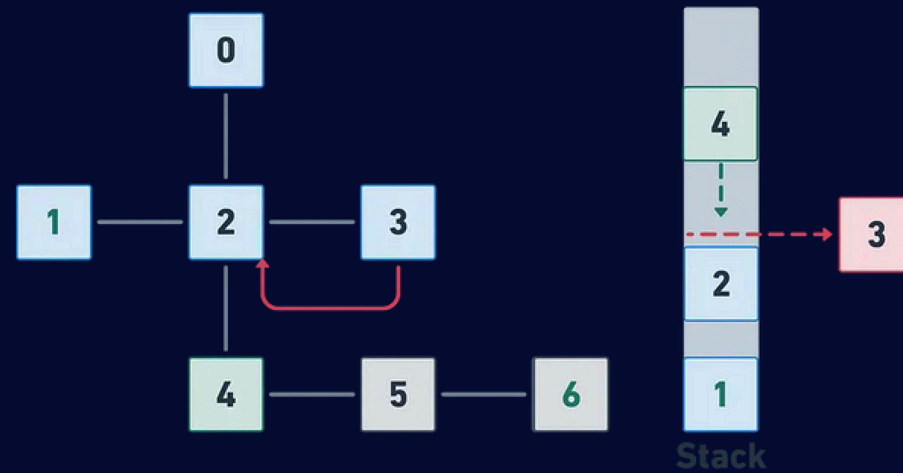


source : https://medium.com/swlh/solving-mazes-with-depth-first-search-e315771317ae

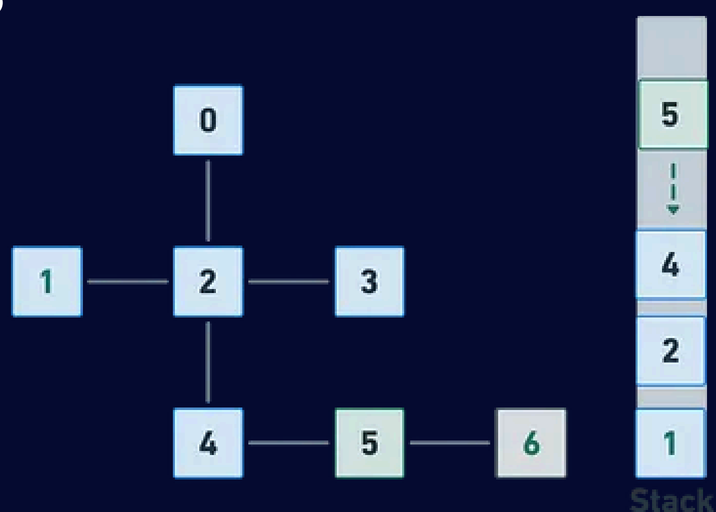
4



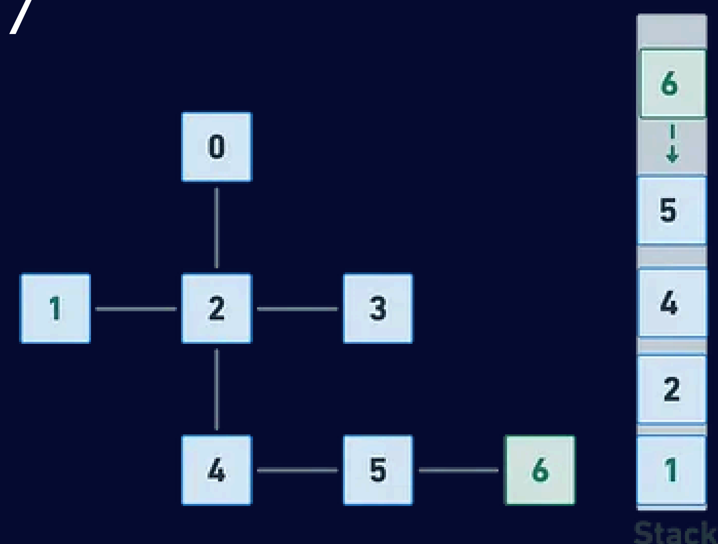
5



6

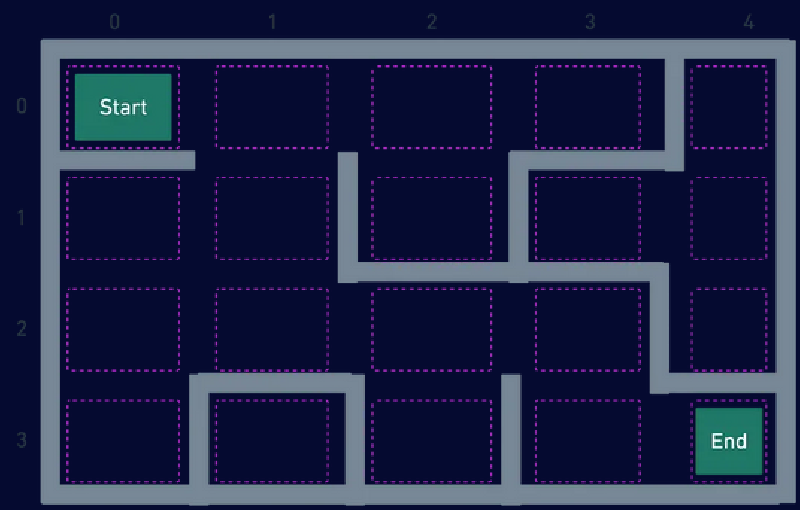


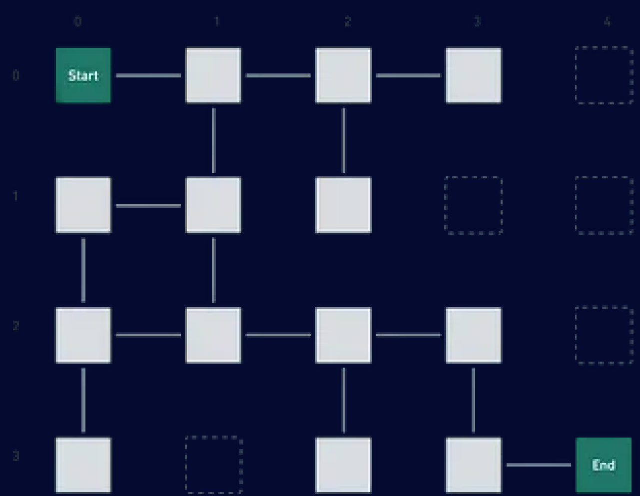
7

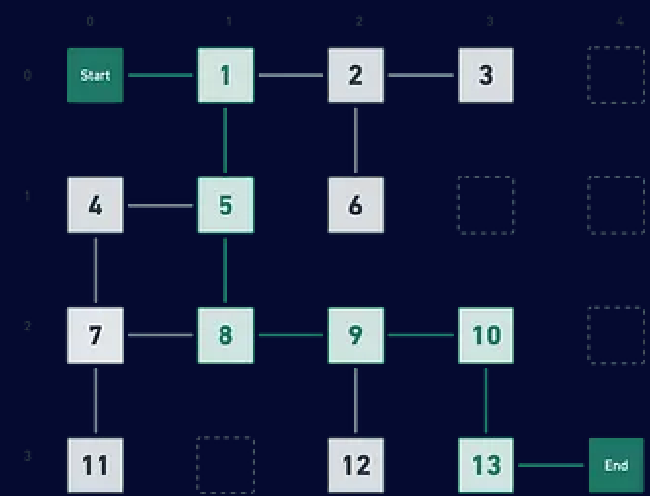


source : https://medium.com/swlh/solving-mazes-with-depth-first-search-e315771317ae

L’exemple concret avec un pseudo-labyrinthe

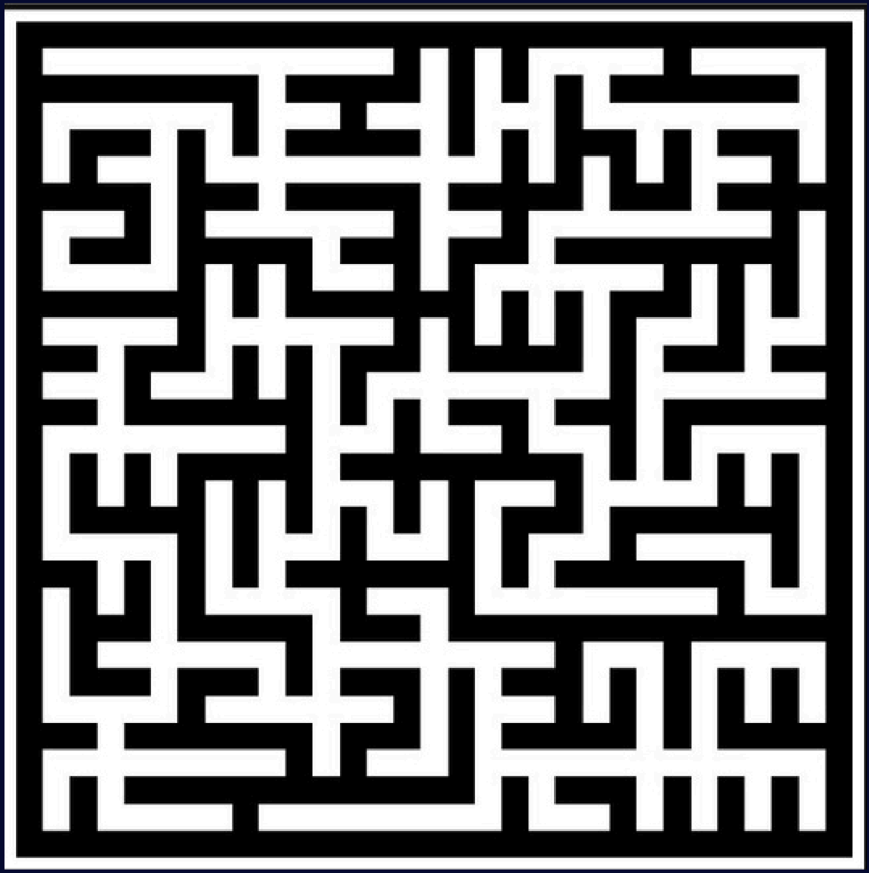


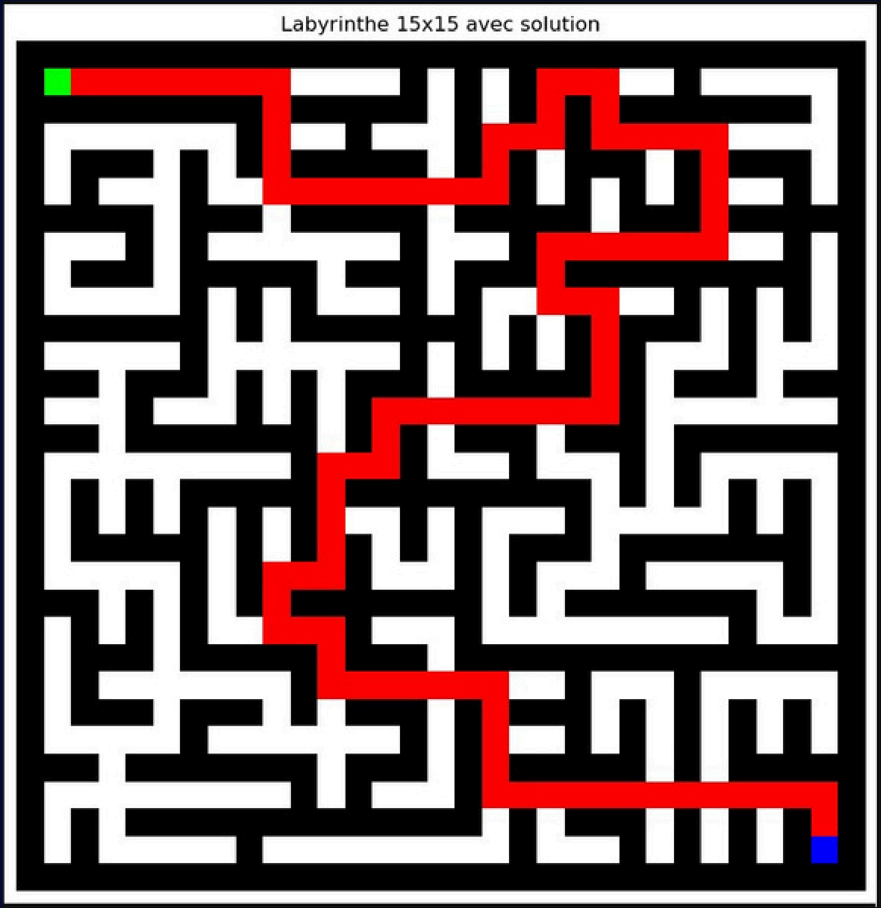




source : https://medium.com/swlh/solving-mazes-with-depth-first-search-e315771317ae

Résolution et affichage des labyrinthes :





Vérification des labyrinthes

```
def verifLaby(lab) :
    l = list(lab.keys())
    dim = l[len(l) - 1]
    dimx = dim[0]
    dimy = dim[1]

    spacetest = espace(dimx, dimy)
    occ = 0
    for el in spacetest:
        #print(spacetest[el])
        occ += len(spacetest[el])
    occ_cut = 0
    for el in lab:
        occ_cut += len(lab[el])
    truth_murs = (dimx*(dimy-1) - dimy + 1 == (occ - occ_cut)/2)
    return truth_murs and connexiTest(lab)
```

Énumération des labyrinthes :

On utilise ici le théorème de Kirchoff :

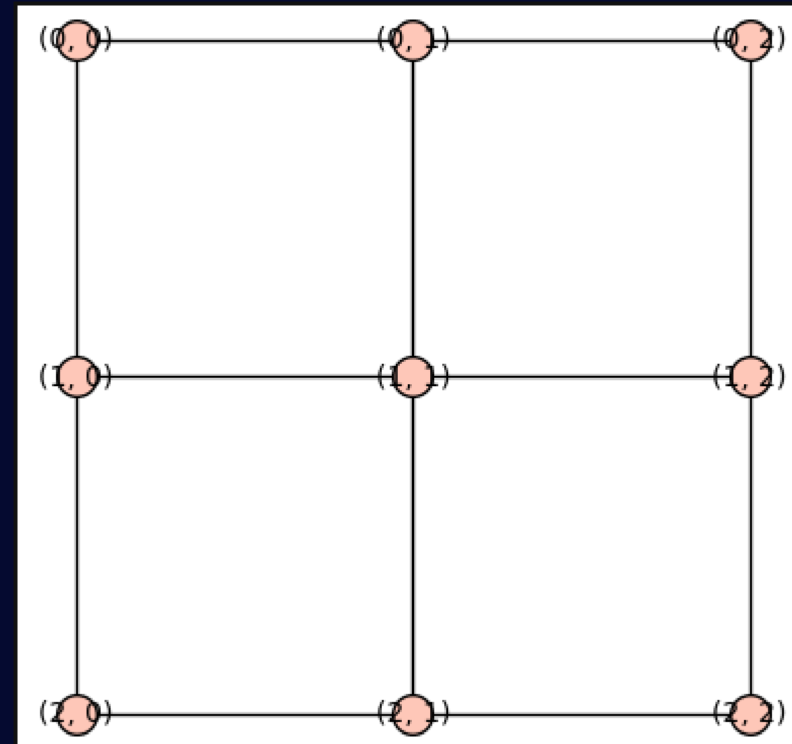
```
def nb_labyrinthes(n, m):
    """
    Selon le théorême de Kirchoff en théorie des graphes, le nombre d'arbres couvrants dans un graphe
    est égal à n'importe quel cofacteur de la matrice laplacienne du graphe.
    """
    # On crée un graphe de dimension n * m
    G = graphs.GridGraph([n, m])

    # On en extrait sa matrice Laplacienne L où l(i,j) = deg[s(i)] si i = j
    #                                                  = -1 si i != j and s(i) est adjacent à s(j)
    #                                                  = 0 sinon
    # Où l(i,j) désigne le coefficient ligne i colonne j de la matrice et s(i)/s(j)
    # représente le i-ème/j-ème sommet du graphe G avec i appartenant à [1, n]
    L = G.laplacian_matrix()

    # On supprime la première ligne et colonne de la matrice
    L_minor = L.delete_rows([0]).delete_columns([0])

    # On calcule son déterminant = un cofacteur de la matrice laplacienne
    return L_minor.determinant()
```

Graphe de dimension 3x3 :



Matrice laplacienne du graphe :

$$\begin{pmatrix} 2 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 3 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 3 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 2 \end{pmatrix}$$

Performances de l'algorithme et résultats obtenus:

Calcul du nombre de labyrinthes d'une taille donnée :

```
%%time
nb_labyrinthes(10, 10)
```

```
CPU times: user 16.1 ms, sys: 4 µs, total: 16.1 ms
Wall time: 15.3 ms
```

```
%%time
nb_labyrinthes(25, 25)
```

```
CPU times: user 1.5 s, sys: 4.12 ms, total: 1.5 s
Wall time: 1.65 s
```

Génération de tous ces labyrinthes (en utilisant une table de hachage) :

```
%%time
t = gen_laby(3, 3)
print(len(t))
```

```
192
CPU times: user 73.5 ms, sys: 64 µs, total: 73.5 ms
Wall time: 71.6 ms
```

```
%%time
t = gen_laby(3, 4)
print(len(t))
```

```
2415
CPU times: user 1.7 s, sys: 3.8 ms, total: 1.71 s
Wall time: 1.79 s
```

Nombre de labyrinthes en fonction de n et m :

	2	3	4	5	6	7
2	4	15	56	209	780	2911
3	15	192	2415	30305	380160	4768673
4	56	2415	100352	4140081	170537640	7022359583
5	209	30305	4140081	557568000	74795194705	10021992194369
6	780	380160	170537640	74795194705	32565539635200	14143261515284447
7	2911	4768673	7022359583	10021992194369	14143261515284447	19872369301840986112

formules de calcul des labyrinthes en fonction de n :

Pour les labyrinthes n*2 :

$$a(0) = 0$$

$$a(1) = 1$$

$$a(n) = 4*a(n-1) - a(n-2)$$

Pour les labyrinthes n*n :

$$a(n) = \frac{2^{n^2-1}}{n^2} \times \prod_{\substack{n_1=0 \\ n_2=0 \\ (n_1,n_2)\neq(0,0)}}^{n-1} \left(2 - \cos\left(\frac{\pi n_1}{n}\right) - \cos\left(\frac{\pi n_2}{n}\right)\right)$$

Source : formule cas n*2

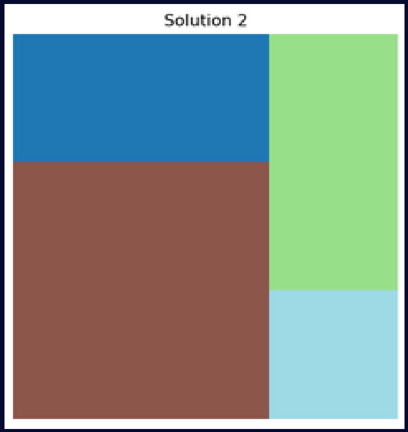
formule cas n*n

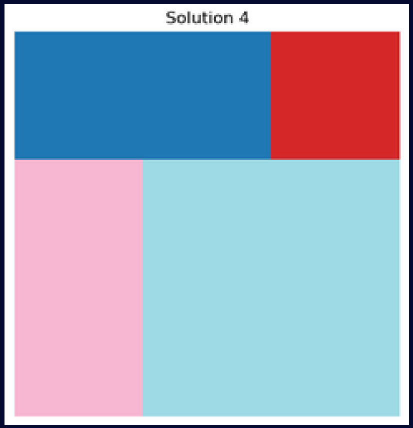
Conjectures/Idée: Chemins à une branche à l'aide d'un pavage

Exemple avec un 3 x 3 :

Quelques pavages du 3x3,
il y en a 18...

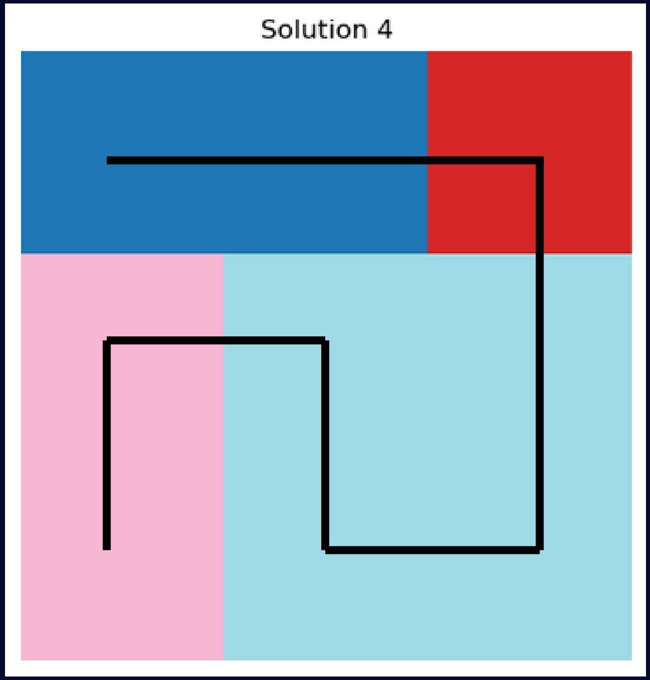
En repérant des carrés 2*2 on peut réduire ce nombre à 6 :





En utilisant les “solutions” des tuiles colorées on peut obtenir une solutions de la grille

Exemple :



Conclusion et retour sur le projet