

## TP : Graphes et labyrinthes

---

L'objectif de ce TP est de faire le lien entre les labyrinthes avancés des TD précédents, et la notion de graphe vue plus récemment. Cela permettra notamment de calculer le plus court chemin entre deux points d'un labyrinthe (ou de déterminer qu'un tel chemin n'existe pas).

### I. Cases accessibles

#### a. En un coup

**Question 1.** Ecrire une fonction qui prend en argument un labyrinthe et les coordonnées  $(i, j)$  d'une case de ce labyrinthe, et qui renvoie la liste des cases accessibles dans ce labyrinthe depuis la case  $(i, j)$  en un coup.

**Remarque :** Cette fonction doit donc renvoyer une liste de couples de coordonnées.

#### b. En au plus deux coups

**Question 2.** Ecrire une fonction qui prend en argument un labyrinthe et les coordonnées  $(i, j)$  d'une case de ce labyrinthe, et qui renvoie la liste des cases accessibles dans ce labyrinthe depuis la case  $(i, j)$  en au plus deux coups.

**Indice :** La méthode `extend` permet d'ajouter le contenu d'une liste dans une deuxième liste.

#### c. En exactement deux coups

**Question 3.** Ecrire une fonction qui prend en argument un labyrinthe et les coordonnées  $(i, j)$  d'une case de ce labyrinthe, et qui renvoie la liste des cases accessibles dans ce labyrinthe depuis la case  $(i, j)$  en exactement deux coups.

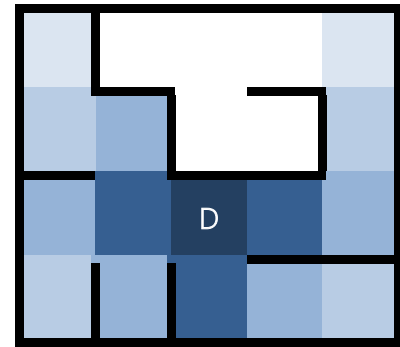
#### d. En autant de coups que nécessaire

**Question 4.** Ecrire une fonction qui prend en argument un labyrinthe et les coordonnées  $(i, j)$  d'une case de ce labyrinthe, et qui renvoie la liste des cases accessibles dans ce labyrinthe depuis la case  $(i, j)$  (sans contrainte sur le nombre de coups nécessaires).

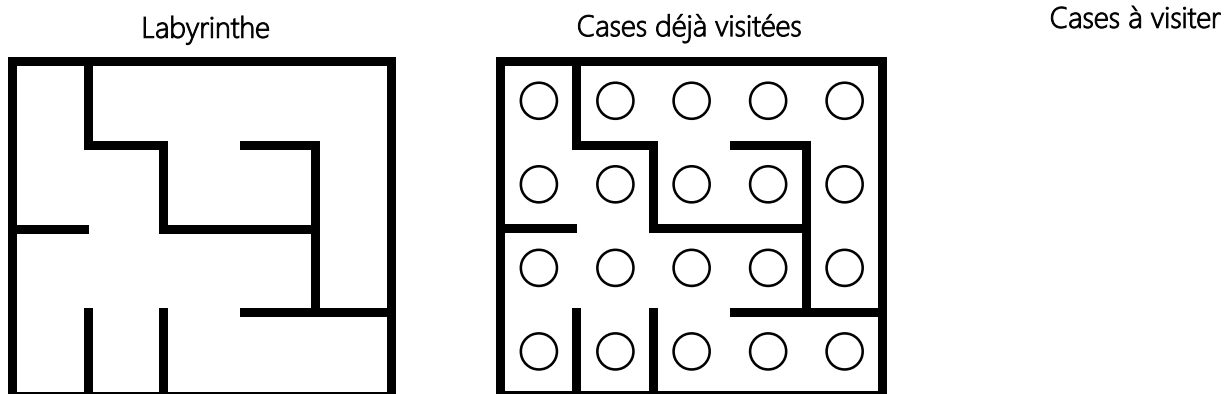
**Indice 1 :** Il pourra être pratique de gérer :

- une liste des cases à visiter
- une liste des cases déjà visitées (sous la forme d'un tableau à deux dimensions contenant des booléens)

**Indice 2 :** Inspirez-vous du dégradé du labyrinthe ci-contre pour l'ordre dans lequel vous devez traiter les cases (la lettre "D" marquant la case de départ).



**Indice 3 :** Vous pouvez également essayer d'essayer de deviner ce que devra faire votre algorithme en vous entraînant sur l'exemple ci-dessous :



### e. Test de faisabilité

**Question 5.** Ecrire une fonction qui prend en argument un labyrinthe et les coordonnées  $(i1, j1)$  et  $(i2, j2)$  de deux cases de ce labyrinthe, et qui renvoie `True` si la case  $(i2, j2)$  est accessible dans ce labyrinthe en partant de la case  $(i1, j1)$ , et `False` sinon.

### f. Nombre de composantes connexes

**Question 6.** Ecrire une fonction qui prend en argument un labyrinthe, et qui renvoie le nombre de composantes connexes dans ce labyrinthe, c'est-à-dire le nombre de "sous-labyrinthes" isolés les uns des autres de sorte qu'il n'est pas possible de passer de l'un à l'autre.

**Indice :** Les cases accessibles depuis une certaine case forment une composante connexe : si un des cases du labyrinthe n'a pas été atteinte, il suffit de recommencer l'exploration à partir de cette nouvelle case.

## II. Distances

Dans cette partie, on se concentrera dans un premier temps sur le cas des labyrinthes parfaits : il existe donc un seul et unique chemin entre tout couple de cases.

Nous verrons ensuite comment gérer également le cas des labyrinthes presque parfaits.

### a. Objectif

5	8	7	6	5
3	2	8	9	4
2	1	0	1	2
3	2	1	2	3

L'objectif de cette partie est de calculer une carte des distances, c'est-à-dire une grille donnant, pour chacune des cases d'un labyrinthe, la distance qui la sépare d'une case de "départ".

Observez l'exemple ci-contre : que remarquez-vous ?

Comment pourriez-vous modifier le code imaginé dans la partie précédente pour créer cette carte des distances ?

### b. Implémentation

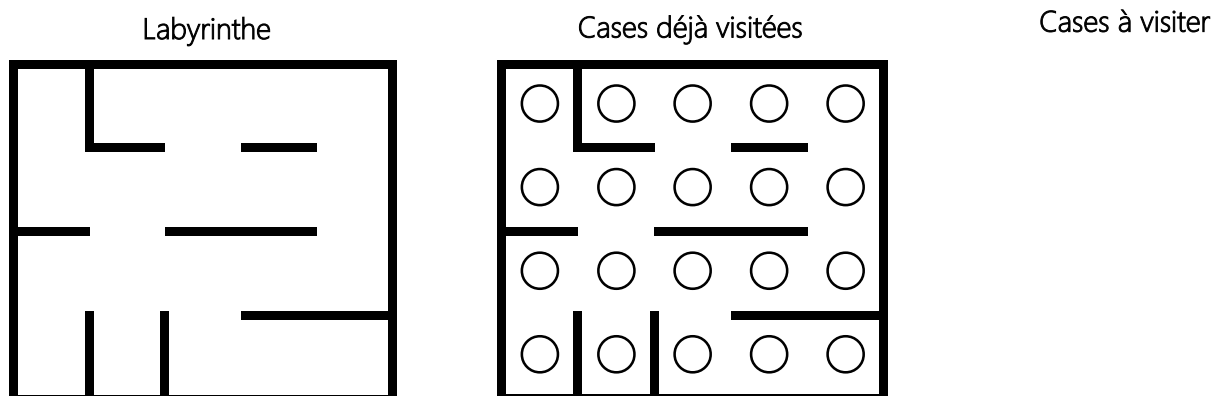
**Question 7.** Ecrire une fonction qui prend en argument un labyrinthe et les coordonnées  $(i, j)$  d'une case de ce labyrinthe, et qui renvoie la "carte des distances" à partir de la case  $(i, j)$  dans ce labyrinthe.

**Indice :** Nous avons déjà vu plus tôt dans l'année comment créer une telle structure à deux dimensions.

### c. Cas des labyrinthes presque parfaits

**Question 8.** La fonction écrite au point précédent fonctionne-t-elle également pour les labyrinthes presque parfaits ? Peut-on traiter les cases à visiter dans n'importe quel ordre ?

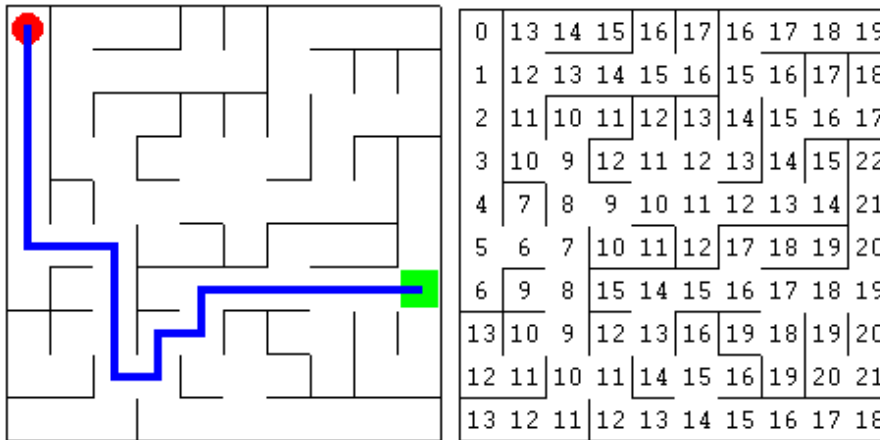
Pour justifier votre réponse, vous pourrez vous inspirer de l'exemple ci-dessous :



**Question 9.** Si besoin est, modifier la fonction de la question précédente pour gérer également le cas des labyrinthes presque parfaits.

### g. Plus court chemin entre deux points

**Question 10.** Observez la carte des distances et le plus court chemin ci-dessous. Que remarquez-vous ?



Comment pourrait-on modifier l'algorithme de calcul de la carte des distances pour obtenir le trajet le plus court entre deux points dans un labyrinthe ?

**Question 11.** Ecrire une fonction qui prend en argument un labyrinthe et les coordonnées  $(i_1, j_1)$  et  $(i_2, j_2)$  de deux cases de ce labyrinthe, et qui renvoie la liste des cases à parcourir pour aller de la case  $(i_1, j_1)$  à la case  $(i_2, j_2)$  en un minimum de coups.

**Indice 1 :** Stocker pour chaque point  $(i, j)$  les coordonnées de son "prédécesseur"  $(i_P, j_P)$ , c'est-à-dire le point  $(i_P, j_P)$  à partir duquel on a atteint le point  $(i, j)$  pendant la phase d'exploration du labyrinthe.

**Indice 2 :** Une fois qu'on a obtenu les coordonnées de tous les prédécesseurs, il suffit de partir de l'arrivée et de remonter la "trace" que constituent ces prédécesseurs pour revenir jusqu'au départ.