

Dernière mise à jour	Informatique	Denis DEFAUCHY
01/06/2022	11 – Bases des graphes	TD 11-3 – A star

Informatique

11

Bases des graphes

TD 11-3

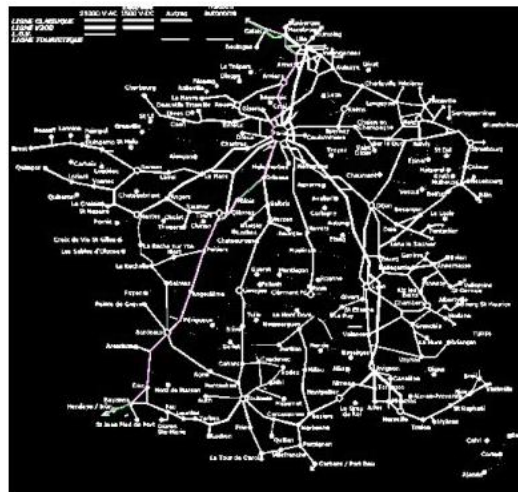
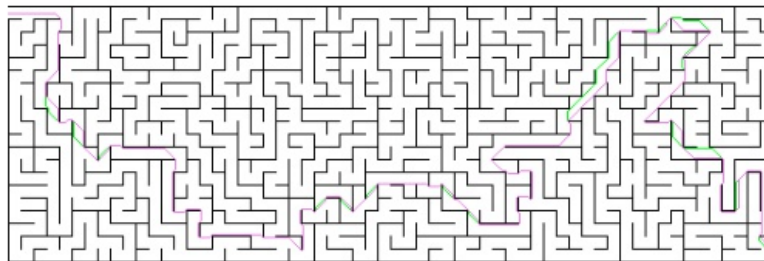
A star

Dernière mise à jour	Informatique	Denis DEFAUCHY
01/06/2022	11 – Bases des graphes	TD 11-3 – A star

Exercice 1: A star

Contexte

On souhaite trouver le plus court chemin à réaliser sur une image en noir et blanc entre un départ et une arrivée. Les pixels noirs seront des obstacles, les blancs des cases accessibles. Voici des exemples de chemins que vous pourrez trouver dans la dernière partie de ce TD :



Nous allons donc :

- Importer des fonctions d'affichage prédéfinies
- Réaliser la grille/image de départ
- Créer un dictionnaire représentant le graphe du domaine d'étude
- Trouver le plus court chemin à l'aide de l'algorithme de Dijkstra
- Faire de même avec l'algorithme A-star
- Comparer performances de ces algorithmes

Pour rappel :

- L'algorithme de Dijkstra permet de trouver le plus court chemin dans un graphe en « avançant » par « arcs de cercles » depuis le départ, jusqu'à atteindre l'arrivée.
- L'algorithme A-star privilégie dans ses choix, les cases dont l'heuristique (distance depuis le début + distance à vol d'oiseau jusqu'à l'arrivée) est la plus faible. Dans ce TD, il cherchera donc toujours à se diriger vers l'arrivée.

Dernière mise à jour	Informatique	Denis DEFAUCHY
01/06/2022	11 – Bases des graphes	TD 11-3 – A star

Prise en main du code élèves

Afin d'assurer un fonctionnement rapide sur tous les ordinateurs, je vous mets à disposition un dossier à télécharger COMPLETEMENT, soit le dossier contenant tous les fichiers et images, et non les fichiers pris séparément.

Sans ouvrir le dossier, faite juste « Télécharger – Téléchargement direct » puis mettez ce dossier dans votre répertoire personnel.

[LIEN](#)

Dossier_Partagé

Enregistrer dans Dropbox Télécharger

Nom

11-2 - TD - A star

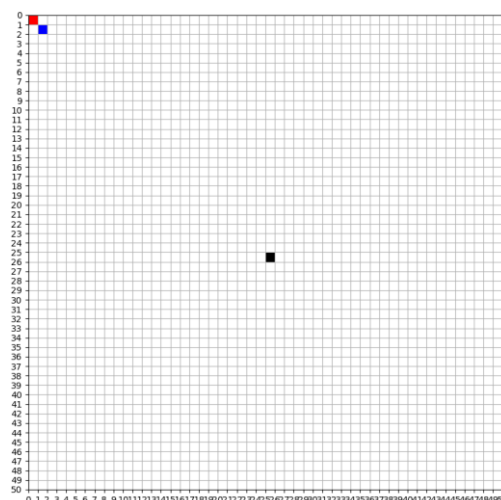
Si le téléchargement est sous forme de Rar, Zip... Pensez à dézipper l'archive afin d'avoir le dossier voulu !

Vous avez à disposition 5 fichiers Python :

11-3 - 1 - Affichage	Import des librairies numpy et matplotlib Création de 3 fonctions d'affichage : Affiche(fig,im,grille) : Affiche l'image d'étude Affiche_Save(fig,im,grille,chemin) : Enregistre l'image pour créer des animations (sans affichage sinon bug de redimensionnement) Affiche_Degrade(Fig,Tab) : Affiche avec un dégradé les distances de chaque pixel depuis le départ sur une nouvelle image
11-3 - 2 - Grille - Elèves	Code prérempli à compléter pour créer la grille de départ
11-3 - 3 - Dico - Elèves	A compléter
11-3 - 4 - Dijkstra – Elèves	A compléter
11-3 - 4 - A star - Elèves	A compléter

Question 1: Télécharger et exécuter les fichiers Affichage et Grille dans l'ordre

A ce stade, vous devriez voir apparaître le domaine d'étude sous la forme suivante :



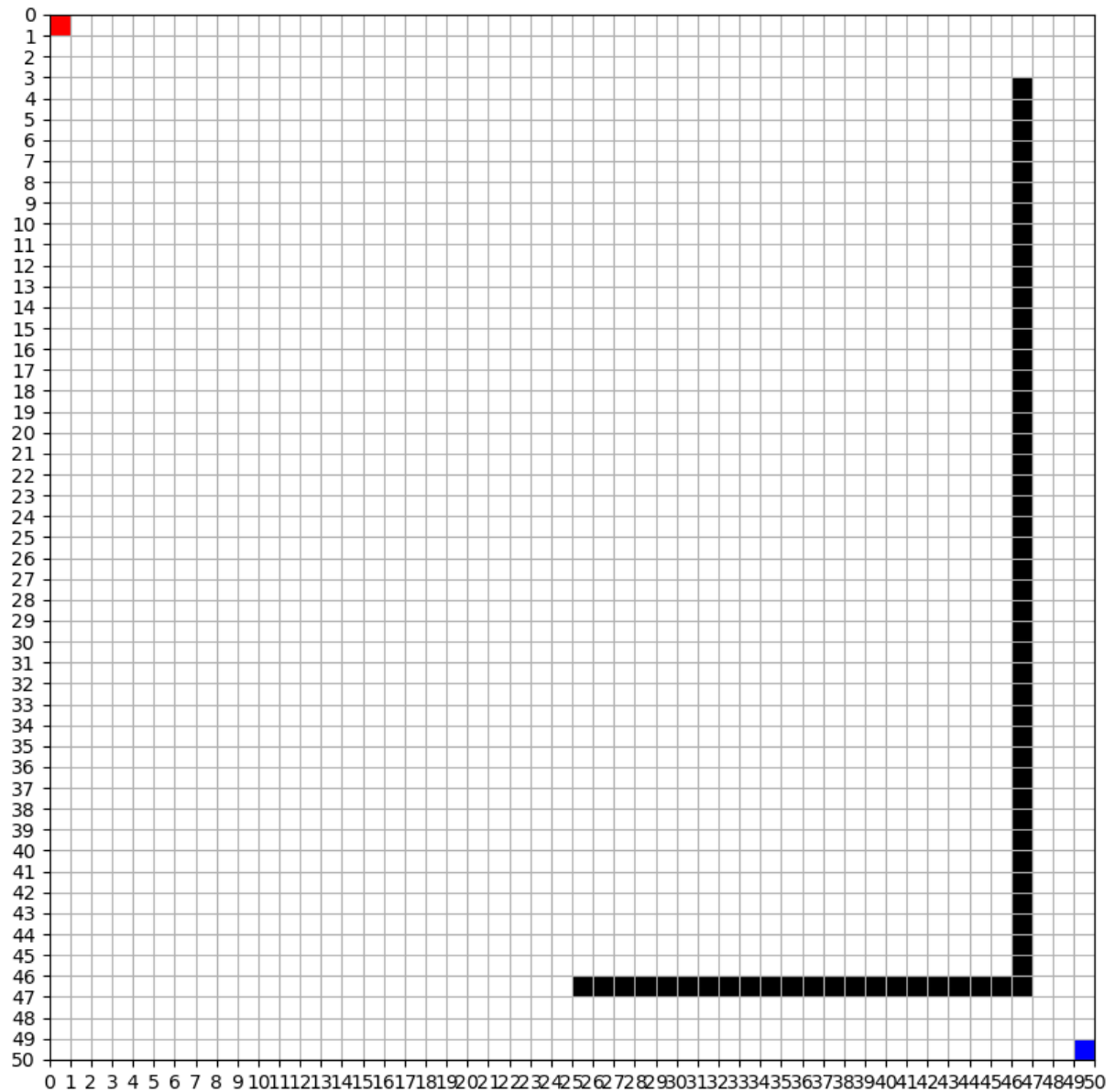
La grille est remplie de cases blanches (accessibles), on voit apparaître le point de départ en rouge, le point d'arrivée en bleu et une case obstacle en noir.

Dernière mise à jour	Informatique	Denis DEFAUCHY
01/06/2022	11 – Bases des graphes	TD 11-3 – A star

Création de la grille de départ

Vous travaillerez dans le fichier nommé « 11-3 - 2 - Grille - Elèves.py ».

On souhaite réaliser la grille de départ suivante :



Question 2: Compléter/Modifier le code afin de réaliser cette image de départ

Dernière mise à jour	Informatique	Denis DEFAUCHY
01/06/2022	11 – Bases des graphes	TD 11-3 – A star

Réalisation du dictionnaire des voisins

Les mouvements possibles pour se déplacer d'une case à l'autre sont les 8 directions :

- $\uparrow \rightarrow \downarrow \leftarrow$ de distance 1
- $\nearrow \searrow \swarrow \nwarrow$ de distance $\sqrt{2}$

On souhaite réaliser un dictionnaire nommé « Dico_Voisins » représentant le graphe du système tel que :

- Les clés sont des tuples de type (ligne,colonne) de chaque pixel pour chaque case/pixel accessible (pas noire). Exemple d'une portion des clés du dictionnaire :

```
>>> Dico_Voisins.keys()
dict_keys([(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (0, 9), (0, 10), (0, 11), (0, 12), (0, 13), (0, 14), (0, 15), (0, 16), (0, 17), (0, 18), (0, 19), (0, 20), (0, 21), (0, 22), (0, 23), (0, 24), (0, 25), (0, 26), (0, 27), (0, 28), (0, 29), (0, 30), (0, 31), (0, 32), (0, 33), (0, 34), (0, 35), (0, 36), (0, 37), (0, 38), (0, 39), (0, 40), (0, 41), (0, 42), (0, 43), (0,
```

- Les valeurs associées à chaque clé/case sont des listes contenant des sous listes (une par case voisine accessible) de deux éléments du type [clé case,distance]. Exemple :

```
>>> Dico_Voisins[(0,0)]
[[ (0, 1), 1.0], [(1, 0), 1.0], [(1, 1), 1.4142135623730951]]
```

Vous travaillerez dans le fichier nommé « 11-3 - 3 - Dico - Elèves.py ».

Question 3: Créer une fonction Test_Pix(P1,P2) qui renvoie le booléen répondant à la question « Le pixel P1 est égal au pixel P2 »

Vérifier :

```
>>> Test_Pix([255,255,255],[0,0,0])
False

>>> Test_Pix([0,0,0],[0,0,0])
True
```

Question 4: Mettre en place le code permettant de créer le dictionnaire Dico_Voisins contenant pour valeurs de chaque case accessible des listes vides

Remarque : Penser qu'une case accessible est « non noire », et non... blanche.

Vous devriez voir :

```
>>> len(Dico_Voisins)
2435

>>> Dico_Voisins[(0,0)]
[]
```

Dernière mise à jour	Informatique	Denis DEFAUCHY
01/06/2022	11 – Bases des graphes	TD 11-3 – A star

Question 5: Mettre en place la fonction `Couples_Voisins(l,c)` qui, pour la case à la ligne `l` et la colonne `c`, ajoute dans sa liste dans `Dico_Voisins`, les couples `[Case_i,Di]` des cases voisins accessibles (max 8 cases possibles)

Remarque :

- On évitera le test « in `Dico_Voisins` » pour vérifier qu'une case est accessible, qui coûte bien plus cher que de vérifier la couleur non noire d'un pixel
- Penser que sur tous les bords de l'image, il n'y a plus 8 voisins... Cela se traduira par des conditions sur `li` et `ci` des cases voisines « `Case_i` »

Vérifier :

```
>>> Couples_Voisins(0,0)

>>> Dico_Voisins[(0,0)]
[[ (0, 1), 1.0], [(1, 0), 1.0], [(1, 1), 1.4142135623730951]]
```

Question 6: Mettre en place les lignes de code permettant de remplir `Dico_Voisins`

Vérifier :

```
>>> Dico_Voisins[(0,0)]
[[ (0, 1), 1.0], [(1, 0), 1.0], [(1, 1), 1.4142135623730951]]

>>> Dico_Voisins[(49,49)]
[[ (48, 48), 1.4142135623730951], [(48, 49), 1.0], [(49, 48), 1.0]]

>>> Dico_Voisins[(43,43)]
[[ (42, 42), 1.4142135623730951], [(42, 43), 1.0], [(42, 44), 1.4142135623730951], [(43, 42), 1.0],
[(43, 44), 1.0], [(44, 42), 1.4142135623730951], [(44, 43), 1.0], [(44, 44), 1.4142135623730951]]

>>> Dico_Voisins[(42,42)]
[[ (41, 41), 1.4142135623730951], [(41, 42), 1.0], [(41, 43), 1.4142135623730951], [(42, 41), 1.0],
[(42, 43), 1.0], [(43, 41), 1.4142135623730951], [(43, 42), 1.0], [(43, 43), 1.4142135623730951]]
```

Dernière mise à jour	Informatique	Denis DEFAUCHY
01/06/2022	11 – Bases des graphes	TD 11-3 – A star

Mise en place de l'algorithme de Dijkstra

Vous travaillerez dans le fichier nommé « 11-3 - 4 - Dijkstra - Elèves.py ».

Ce fichier contient un paragraphe d'initialisation :

Distances	Array aux mêmes dimensions que l'image traitée, tel que Distances[l,c] est la distance du chemin menant à la case concernée depuis le départ Il est initialisé avec des valeurs infinies
Reste	Copie du dictionnaire Dico_Voisins. On enlèvera les cases traitées de ce dictionnaire et l'algorithme se terminera au plus tard, quand ce dictionnaire est vide
Provenances	Dictionnaire des provenances de chaque case. Initialisé vide, il contiendra à la fin de la réalisation de l'algorithme des clés et valeurs qui seront respectivement le nom de la case concernée et la case qui a permis d'y accéder lors de l'exécution de l'algorithme. Par exemple, si Provenances[(1,1)]=(0,0), c'est que pour atteindre la case (1,1) avec un chemin depuis le départ de longueur Distances[1,1], la case précédente de ce chemin est (0,0)

Pour rappel, départ et arrivée ont été définies précédemment (ld,cd,la,ca).

Nous allons maintenant programmer l'algorithme de Dijkstra (rappelez-vous le TD 11-1 😊), dont nous allons rappeler les grandes étapes.

Tant que **S** n'est pas l'arrivée, qu'il reste des stations dans **Reste** et que Distance[IS,cS] est différente de l'infini (cette condition permet de gagner du temps si **Arrivée** n'est pas accessible depuis **Départ**)

- **S** ← Station parmi **Reste** ayant la distance minimum dans **Distances**
- Mise à jour de **Reste** (retirer S)
- **Voisins** ← Dictionnaire des stations de **Reste** voisines de **S**
- Traitement des voisins : Pour chaque voisin V, si la distance $\text{Depart} \rightarrow S \rightarrow V < \text{Depart} \rightarrow V$ actuellement stockée dans **Distances** :
 - Mise à jour de **Distances** avec cette nouvelle distance
 - Mise à jour de **Provenances** afin d'indiquer que S est le prédécesseur de V

Question 7: Mettre en place cet algorithme

On remarquera que si la case arrivée n'est pas accessible, Reste sera vide, mais en plus, Distances[la,ca] sera toujours infinie.

Dernière mise à jour	Informatique	Denis DEFAUCHY
01/06/2022	11 – Bases des graphes	TD 11-3 – A star

Question 8: Mettre en place le code permettant soit d'afficher que le chemin n'existe pas, soit de remonter le chemin menant à l'arrivée, en affichant dans la console les cases empruntées, la distance parcourue, et le nombre d'itérations réalisées.

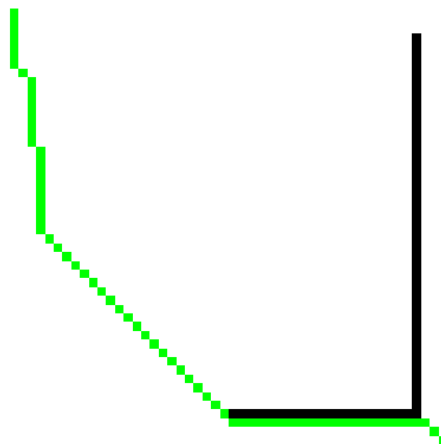
Vérifier :

Distance: 89.79898987322329

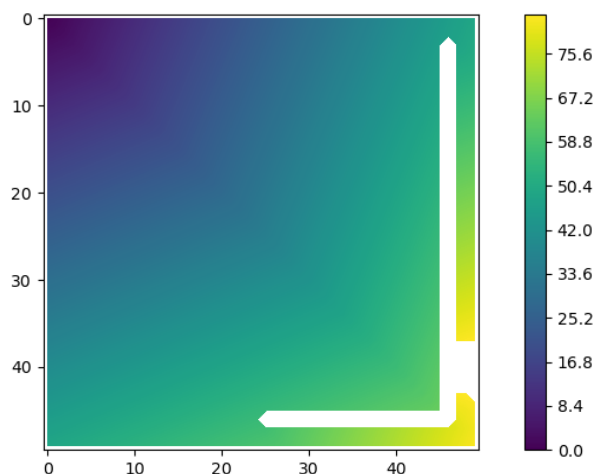
Itérations: 2426

```
['0-0', '1-0', '2-0', '3-0', '4-0', '5-0', '6-0', '7-0', '8-0', '9-0', '10-0', '11-0',
', '12-0', '13-0', '14-0', '15-0', '16-0', '17-0', '18-0', '19-0', '20-0', '21-0', '
22-0', '23-0', '24-0', '25-0', '26-0', '27-0', '28-0', '29-0', '30-0', '31-0', '32-0
', '33-0', '34-0', '35-0', '36-1', '37-2', '38-3', '39-4', '40-5', '41-6', '42-7', '
43-8', '44-9', '45-10', '45-11', '45-12', '45-13', '45-14', '45-15', '45-16', '45-17
', '45-18', '45-19', '45-20', '45-21', '45-22', '45-23', '45-24', '45-25', '45-26',
'45-27', '45-28', '45-29', '45-30', '45-31', '45-32', '45-33', '45-34', '45-35', '45
-36', '45-37', '45-38', '45-39', '45-40', '45-41', '45-42', '45-43', '45-44', '45-45
', '46-46', '47-47', '48-48', '49-49']
```

Question 9: Mettre en place le code permettant d'afficher le chemin trouvé en vert sur l'image



Question 10: En utilisant la fonction Affiche_Degrade, afficher les distances des pixels depuis le départ et les pixels traités par l'algorithme



Dernière mise à jour	Informatique	Denis DEFAUCHY
01/06/2022	11 – Bases des graphes	TD 11-3 – A star

Mise en place de l'algorithme de A-star

Vous travaillerez dans le fichier nommé « 11-3 - 5 - A star - Elèves.py ».

Dijkstra choisissait s comme la station la plus proche du départ. On propose de prendre en compte l'heuristique « Distances à vol d'oiseau » entre s et l'arrivée.

$$f(s) = d(s) + d'(s, s_{fin})$$

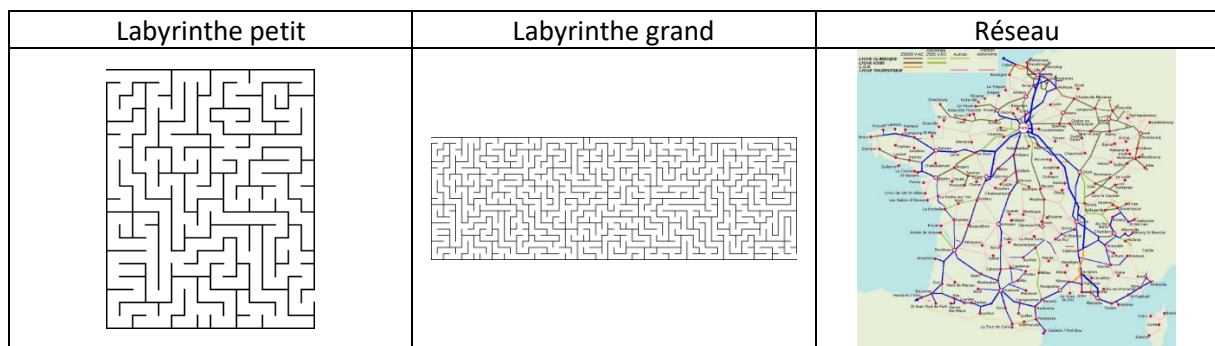
Question 11: Créer la fonction $f(\text{Case})$ renvoyant le calcul attendu

Question 12: Copier/coller le code Dijkstra et modifier ce qu'il faut afin de réaliser l'algorithme A star

Question 13: Comparer les algorithmes Dijkstra et A star

Pour aller plus loin

Vous avez à disposition trois images :



Vous trouverez dans le dossier télécharger en début de TD les 3 images ci-dessus, partagées sous les formats BMP (ouverture avec matplotlib) et array (ouverture avec numpy en cas de bug matplotlib).

Pour chacun de ces 3 cas :

Question 14: Créer un nouveau fichier « 11-2 – 2 – Nom_à_choisir.py »

Question 15: Sous Python, ouvrir l'image et l'afficher

Question 16: Créer une fonction $f_NB(im)$ qui renvoie une nouvelle image en noir et blanc $([0,0,0]$ ou $[255,255,255])$ à partir de l'image im

Attention : les labyrinthes seront aussi transformés en noir et blanc (je n'ai pas vérifié chaque pixel pour être certain que les triplets sont bien composés uniquement de 0 ou de 255).

Question 17: Transformer l'image en noir et blanc et l'afficher

Question 18: Choisir et ajouter les points de départ et d'arrivée sur l'image

Question 19: Faire tourner les algorithmes Dijkstra et A-star

Question 20: Apprécier les résultats et comparer l'efficacité des algorithmes