PyRat : cours 3

L'équipe PyRat

Jalons

Séance 1 ...

Séance 2 ...

Séance 3 Ramasser un unique morceau de fromage dans le labyrinthe

Séance 4 ...

Séance 5 ...

Séance 6 Ramasser plusieurs morceaux de fromage dans le labyrinthe

Séance 7 Gagner contre un adversaire

Plus court chemin dans un graphe pondéré

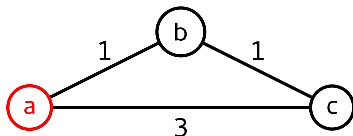
Jusqu'à présent

- Cas du graphe non pondéré
- L'algorithme de parcours en largeur donne la réponse

Maintenant

- On va considérer des graphes pondérés
- *Dans le cadre du projet, il y aura maintenant de la boue entre certaines cases, ce qui coûte plus de temps pour se déplacer*
- Le parcours en largeur ne suffit plus
- Deux algorithmes vont être vus :
  - Un algorithme 1-to-* : Dijkstra
  - Un algorithme *-to-* : Roy-Warshall

Cas du parcours en largeur



Dans cet exemple, le parcours en largeur nous donne c à distance 3 de a alors que le plus court chemin est clairement de longueur 2.

Idée : garder la philosophie du parcours en largeur mais tenir compte des poids des arêtes.

Pour cela on va remplacer la file par un **tas-min**

Tas-min

Définition

Un tas-min (aussi appelé file de priorité) est une structure de données dans laquelle les éléments stockés ont la forme de couples `(clé,valeur)` où `valeur` est une quantité munie d'un ordre total (par exemple un entier).

La structure dispose des primitives suivantes

- `ajouter_ou_remplacer(file_priorité, clé, valeur)` qui permet d'ajouter un nouveau couple à la structure de données. Si la clé existe déjà et que sa valeur stockée est supérieure, alors elle est mise à jour avec la nouvelle valeur
- `retirer(file_priorité)` qui permet de récupérer un couple `(clé,valeur)` dans la structure de données tel que `valeur` est la valeur minimale dans la structure de donnée
- `vide(file_priorité)` qui renvoit `True` si et seulement si la structure de données est vide

Algorithme de Dijkstra

Resolution du problème de plus court chemin 1-to-*

```
def dijkstra(graphe, sommet_départ):
    # initialisation
    tas_min = nouveau tas-min
    ajouter_ou_remplacer(tas_min, sommet_départ, 0)
    routage = [] * ordre(graphe)

    # boucle de l'algorithme
    tant que non(vide(tas_min)):
        sommet_courant, distance = retirer(tas_min)
        pour tout sommet i voisin de sommet_courant:
            dist_par_courant = distance + graphe[sommet_courant][i]
            ajouter_ou_remplacer(tas_min, i, dist_par_courant)
            routage[i] = sommet_courant
```

Quelques notes sur Dijkstra

Le robinet

Une vision intuitive de l'algorithme de Dijkstra est d'imaginer un robinet d'eau ouvert au sommet initial.
Les sommets sont visités par Dijkstra dans l'ordre où l'eau les atteindrait.

Condition suffisante pour correction

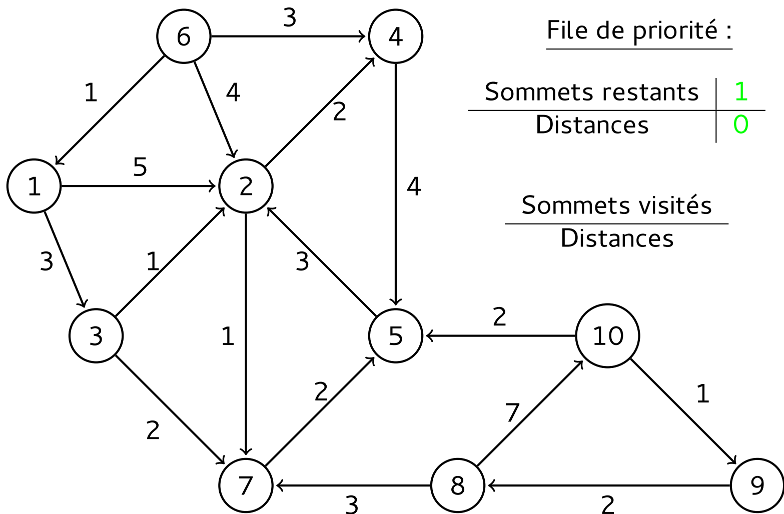
Le graphe contient des poids positifs

Complexité

La complexité dépend fortement de celle de la file de priorité.
En pratique on peut obtenir $\mathcal{O}(|E| + |V| \ln(|V|))$ assez facilement (i.e. $\mathcal{O}(|V| \ln(|V|))$ pour notre cas, car $|E| < 4|V|$)

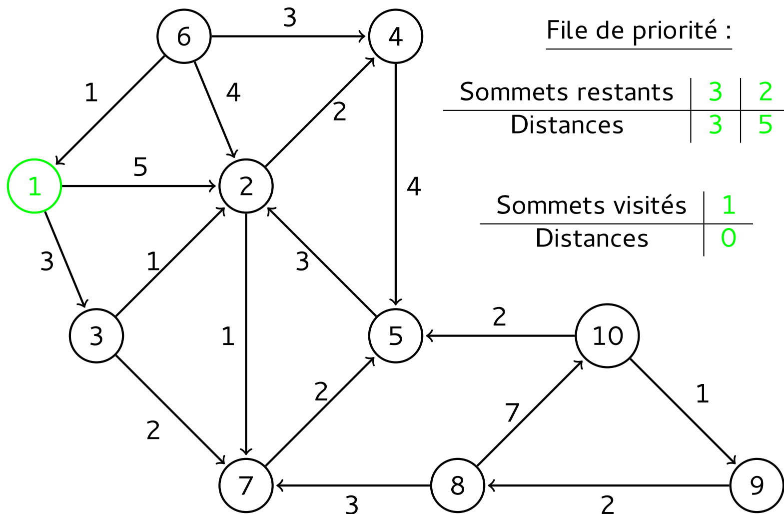
TELECOM Bretagne

Exemple

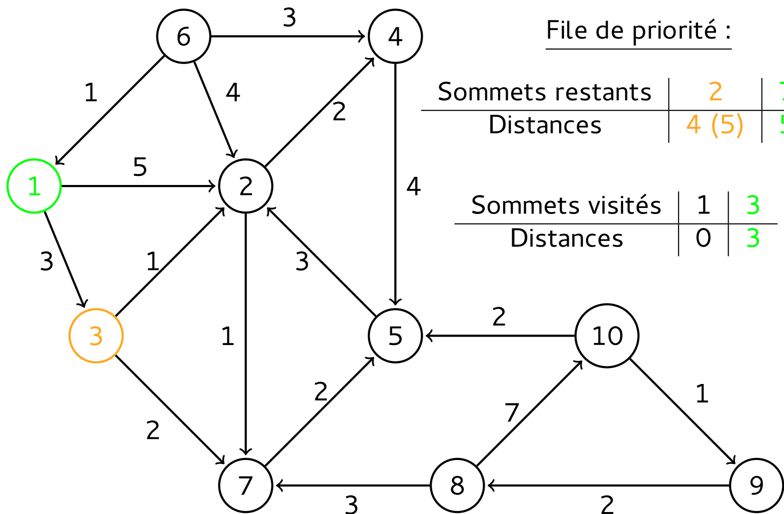


TELECOM Bretagne

Exemple

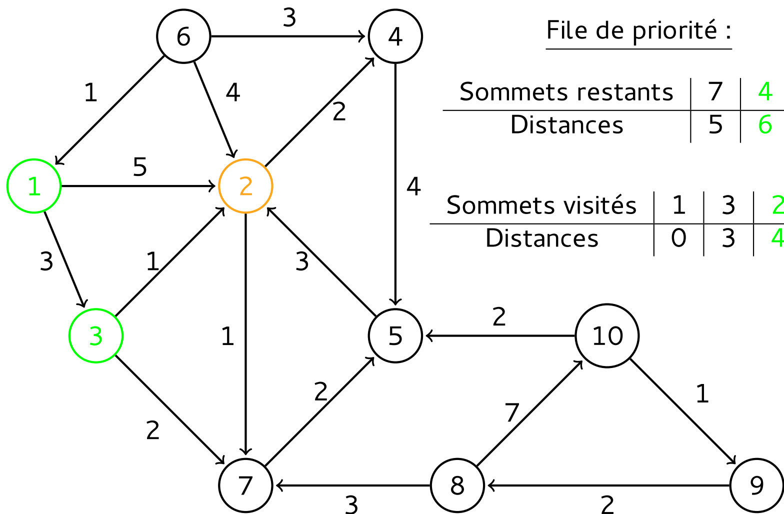


Exemple



TELECOM Bretagne

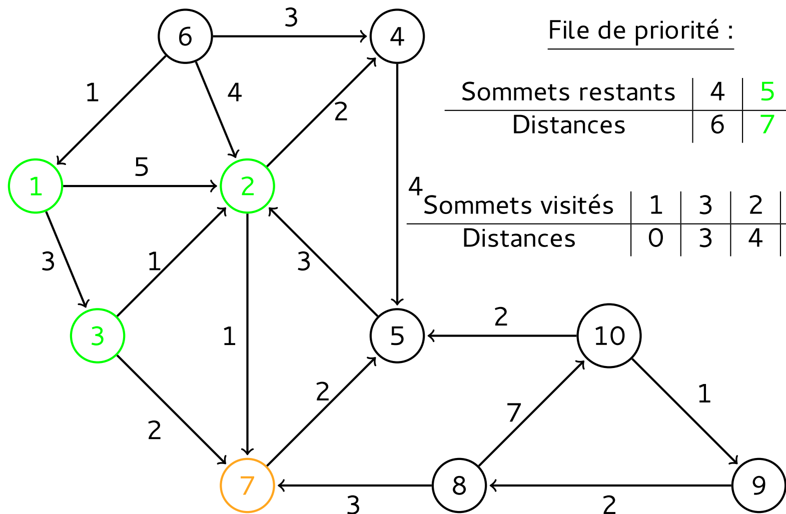
Exemple



Sommets restants	7	4
Distances	5	6

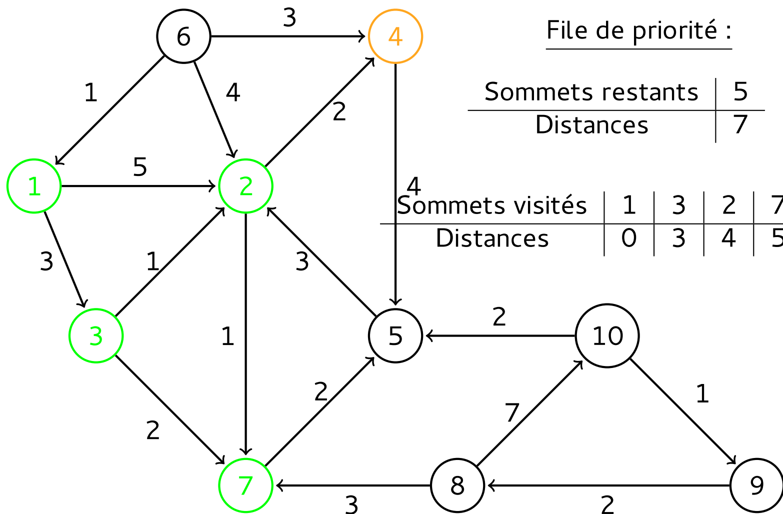
Sommets visités	1	3	2
Distances	0	3	4

Exemple



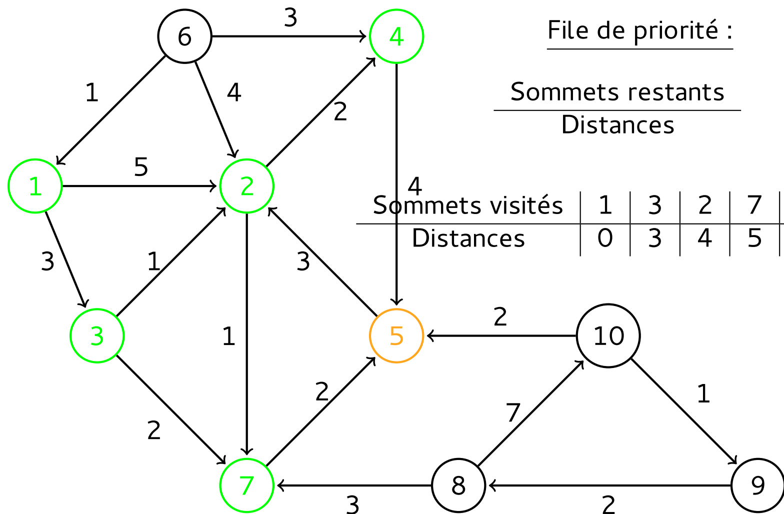
TELECOM Bretagne

Exemple



TELECOM Bretagne

Exemple



File de priorité :

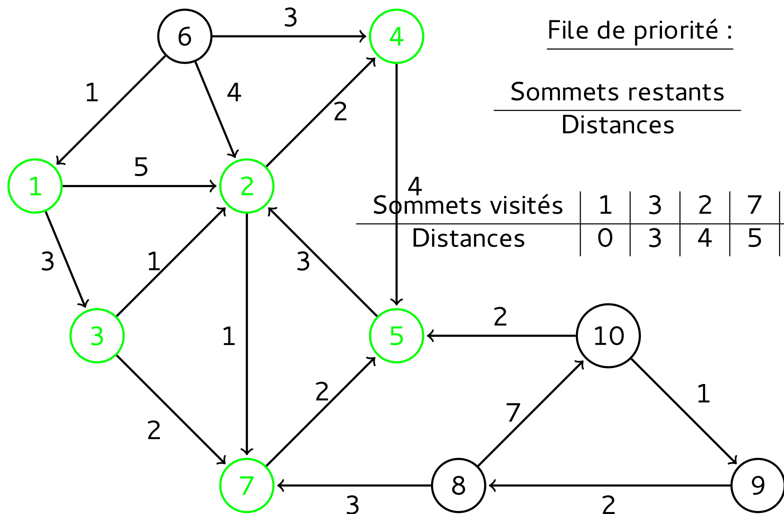
Sommets restants

Distances

Sommets visités	1	3	2	7	4	5
Distances	0	3	4	5	6	7

TELECOM Bretagne

Exemple



File de priorité :

Sommets restants
Distances

Sommets visités	1	3	2	7	4	5
Distances	0	3	4	5	6	7

TELECOM Bretagne

Roy-Warshall (ou Floyd-Warshall)

Roy-Warshall permet de calculer directement *-to-*

Algorithme

```
pour k dans range(|V|):
    pour i dans range(|V|):
        pour j dans range(|V|):
            alt_distance = distances[i][k] + distances[k][j]
            si alt_distance < distances[i][j]:
                distances[i][j] = alt_distance
                routage[i][j] = routage[k][j]
```

- `distances` est une matrice $|V| \times |V|$ initialisée à $+\infty$ partout sauf la diagonale, initialisée à 0, et pour les cases `[i][j]` où i est relié par une arête à j dans le graphe auquel cas la case contient le poids de l'arête correspondante
- `routage` se lit comme : "pour aller de i à j, il faut passer par le sommet dans la case `routage[i][j]`", et s'initialise à une valeur quelconque

TELECOM Bretagne

Quelques notes sur Roy–Warshall

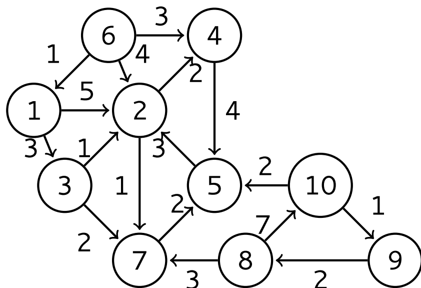
Condition nécessaire et suffisante pour la correction

Roy-Warshall fonctionne pour tout graphe pondéré (même avec des valuations négatives), s'il n'y a pas de cycle absorbant
On appelle cycle absorbant un cycle dont la longueur est négative

Complexité

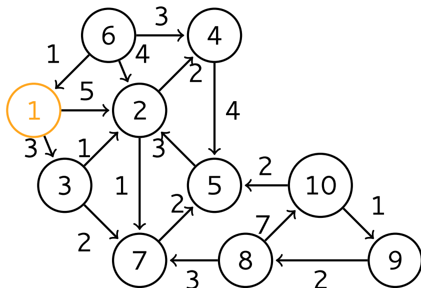
Dans sa version canonique donnée dans le cours, elle est en $\Theta(|V|^3)$

Exemple



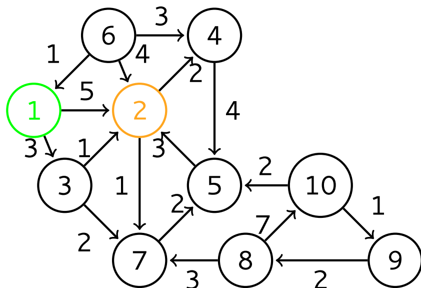
k=0	1	2	3	4	5	6	7	8	9	10
1		5	3							
2				2			1			
3		1					2			
4					4					
5		3								
6	1	4		3						
7					2					
8							3			7
9								2		
10					2				1	

Exemple



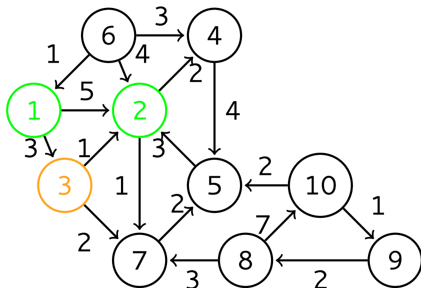
k=1	1	2	3	4	5	6	7	8	9	10
1		5	3							
2				2			1			
3		1					2			
4					4					
5		3								
6	1	4	4	3						
7					2					
8							3			7
9								2		
10					2				1	

Exemple



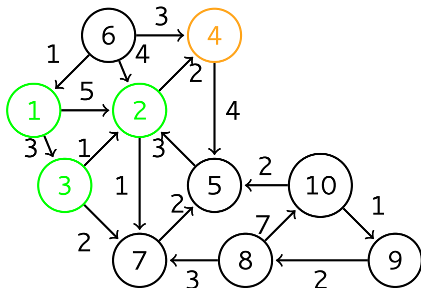
k=2	1	2	3	4	5	6	7	8	9	10
1		5	3	7			6			
2				2			1			
3		1		3			2			
4					4					
5		3		5			4			
6	1	4	4	3			5			
7					2					
8							3			7
9								2		
10					2				1	

Exemple



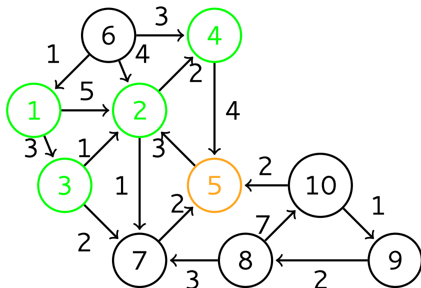
k=3	1	2	3	4	5	6	7	8	9	10
1		4(5)	3	6(7)			5(6)			
2				2			1			
3		1		3			2			
4					4					
5		3		5			4			
6	1	4	4	3			5			
7					2					
8							3			7
9								2		
10					2				1	

Exemple



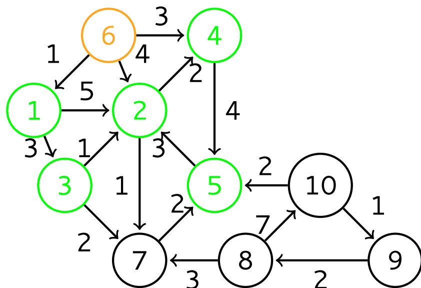
k=4	1	2	3	4	5	6	7	8	9	10
1		4	3	6	10		5			
2				2	6		1			
3		1		3	7		2			
4					4					
5		3		5	9		4			
6	1	4	4	3	7		5			
7					2					
8							3			7
9								2		
10					2				1	

Exemple



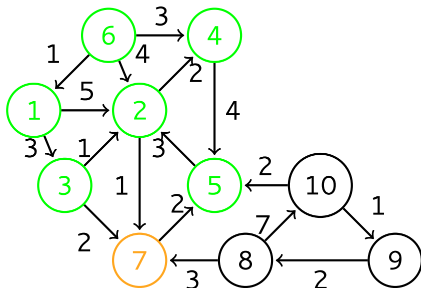
k=5	1	2	3	4	5	6	7	8	9	10
1		4	3	6	10		5			
2		9		2	6		1			
3		1		3	7		2			
4		7		9	4		8			
5		3		5	9		4			
6	1	4	4	3	7		5			
7		5		7	2		6			
8							3			7
9								2		
10		5		7	2		6		1	

Exemple



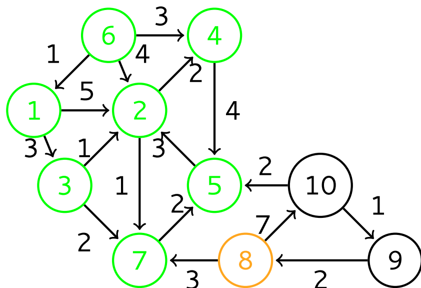
k=6	1	2	3	4	5	6	7	8	9	10
1		4	3	6	10		5			
2		9		2	6		1			
3		1		3	7		2			
4		7		9	4		8			
5		3		5	9		4			
6	1	4	4	3	7		5			
7		5		7	2		6			
8							3			7
9								2		
10		5		7	2		6		1	

Exemple



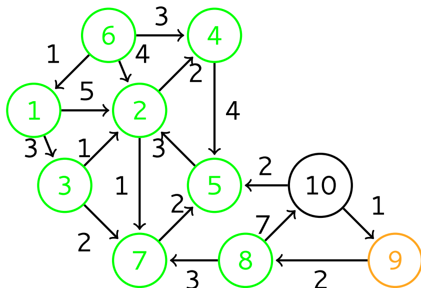
k=7	1	2	3	4	5	6	7	8	9	10
1		4	3	6	7(10)		5			
2		6(9)		2	3(6)		1			
3		1		3	4(7)		2			
4		7		9	4		8			
5		3		5	6(9)		4			
6	1	4	4	3	7		5			
7		5		7	2		6			
8		8		10	5		3			7
9								2		
10		5		7	2		6		1	

Exemple



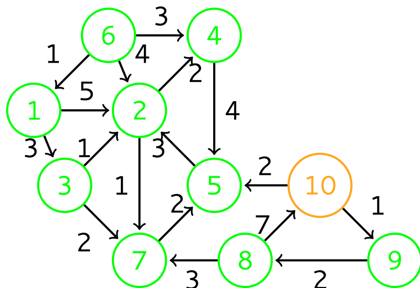
k=8	1	2	3	4	5	6	7	8	9	10
1		4	3	6	7		5			
2		6		2	3		1			
3		1		3	4		2			
4		7		9	4		8			
5		3		5	6		4			
6	1	4	4	3	7		5			
7		5		7	2		6			
8		8		10	5		3			7
9		10		12	7		5	2		9
10		5		7	2		6	3	1	

Exemple



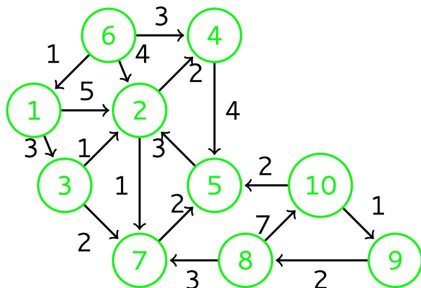
k=9	1	2	3	4	5	6	7	8	9	10
1		4	3	6	7		5			
2		6		2	3		1			
3		1		3	4		2			
4		7		9	4		8			
5		3		5	6		4			
6	1	4	4	3	7		5			
7		5		7	2		6			
8		8		10	5		3			7
9		10		12	7		5	2		9
10		5		7	2		6	3	1	10

Exemple



k=10	1	2	3	4	5	6	7	8	9	10
1		4	3	6	7		5			
2		6		2	3		1			
3		1		3	4		2			
4		7		9	4		8			
5		3		5	6		4			
6	1	4	4	3	7		5			
7		5		7	2		6			
8		8		10	5		3	10	8	7
9		10		12	7		5	2	10	9
10		5		7	2		6	3	1	10

Exemple



fini !	1	2	3	4	5	6	7	8	9	10
1		4	3	6	7		5			
2		6		2	3		1			
3		1		3	4		2			
4		7		9	4		8			
5		3		5	6		4			
6	1	4	4	3	7		5			
7		5		7	2		6			
8		8		10	5		3	10	8	7
9		10		12	7		5	2	10	9
10		5		7	2		6	3	1	10

Modularité du code

La modularité

En programmation, on appelle modularité le fait de factoriser un code en sous-fonctions simples, lesquelles peuvent être développées indépendemment, et réutilisées ensuite

Les algorithmes vus aujourd'hui

- Gardez en tête que les algorithmes vus aujourd'hui vous seront utiles dans les étapes à venir
- Il faut donc réfléchir à rendre votre code réutilisable (donc structuré et documenté)

La structure du graphe

Il sera valorisé d'utiliser des graphes en argument de vos fonctions qui sont des objets complexes, pouvant être implémentés de différentes façons, et de tester l'influence de ces implémentations sur les performances de vos programmes

C’est à vous

Idées pour le projet

- Comparer le temps de calcul de Roy-Warshall et Dijkstra
- Regarder l’influence des paramètres du labyrinthe sur la taille des chemins trouvés ou le temps de calcul
- Regarder l’impact des choix d’implémentation (e.g. tas-min) sur le temps de calcul
- Comparer à d’autres algorithmes de recherche des plus courts chemins