Programmation et IA

Cours 6

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/prog-ia-25

Plan

- graphes
- représentation par listes de successeurs
- recherche de chemin
- graphes valués
- représentation par matrice de connexion

[texte des programmes]

Pour apprendre Python:

w3schools: tutoriel et réference

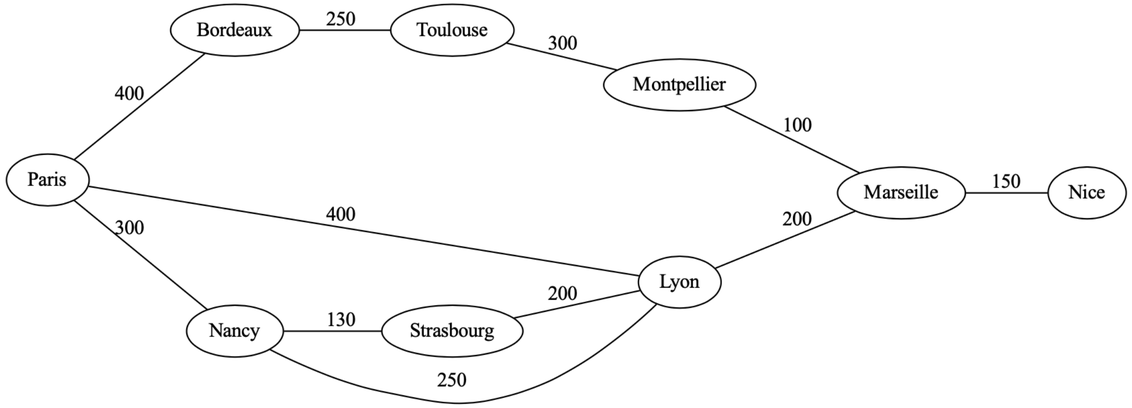
`https://www.w3schools.com/python/default.asp`

programiz: tutoriel et réference

`https://www.programiz.com/python-programming`

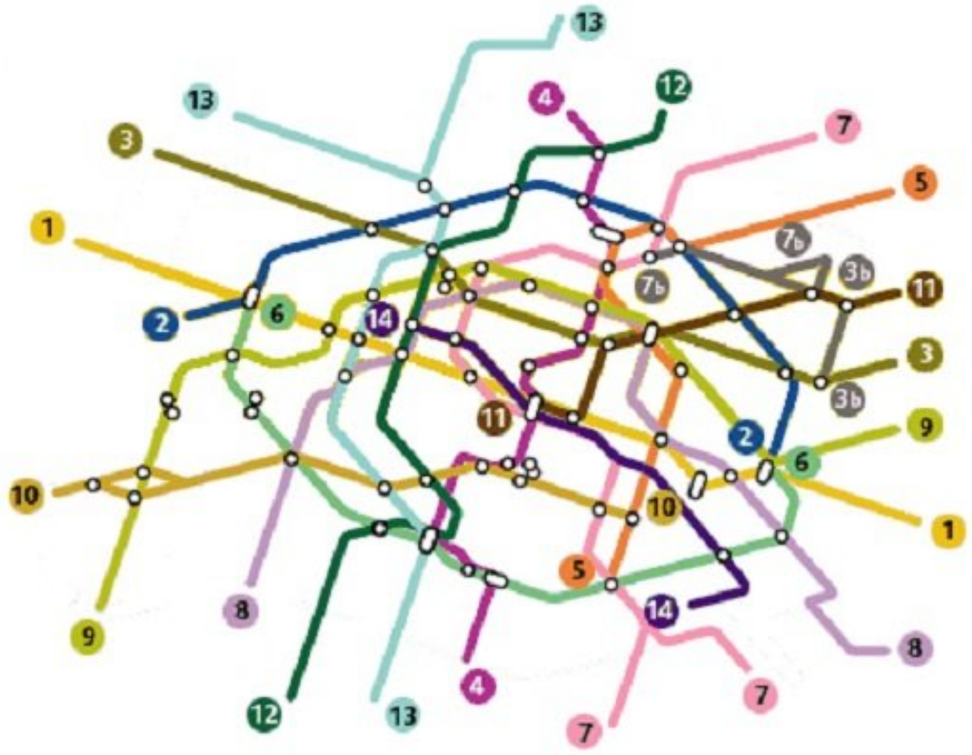
les structures de données (3)

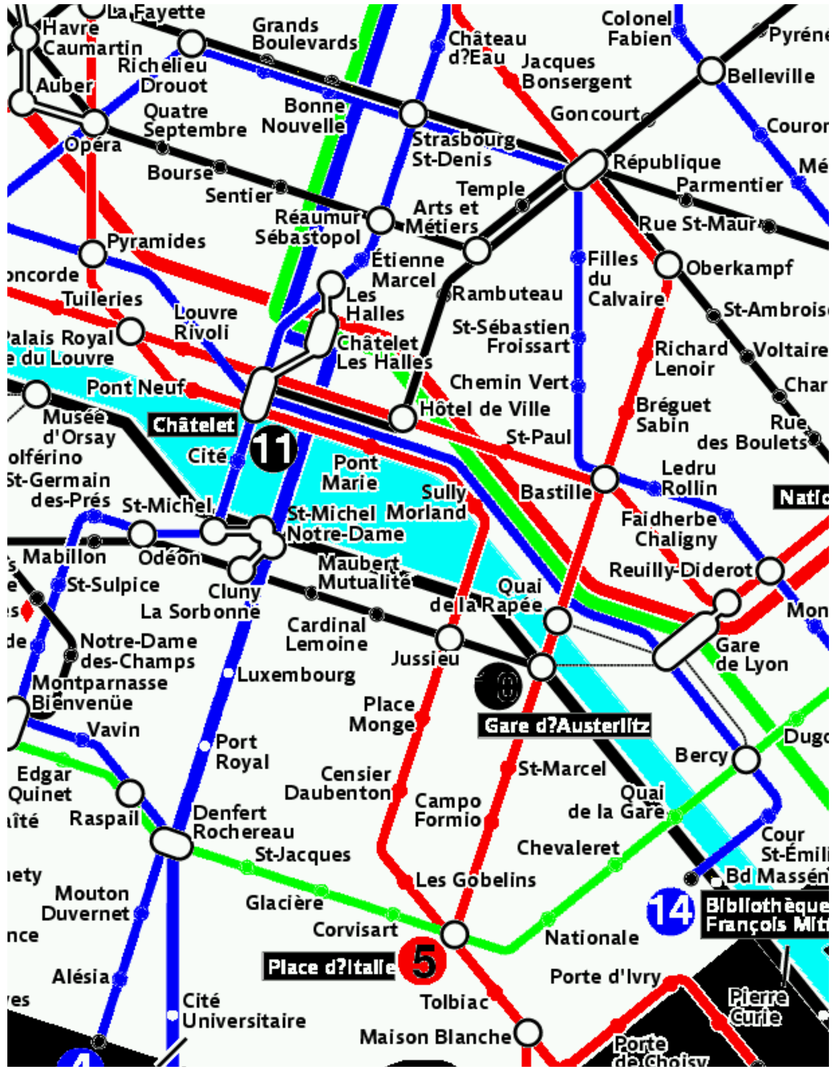
Graphes



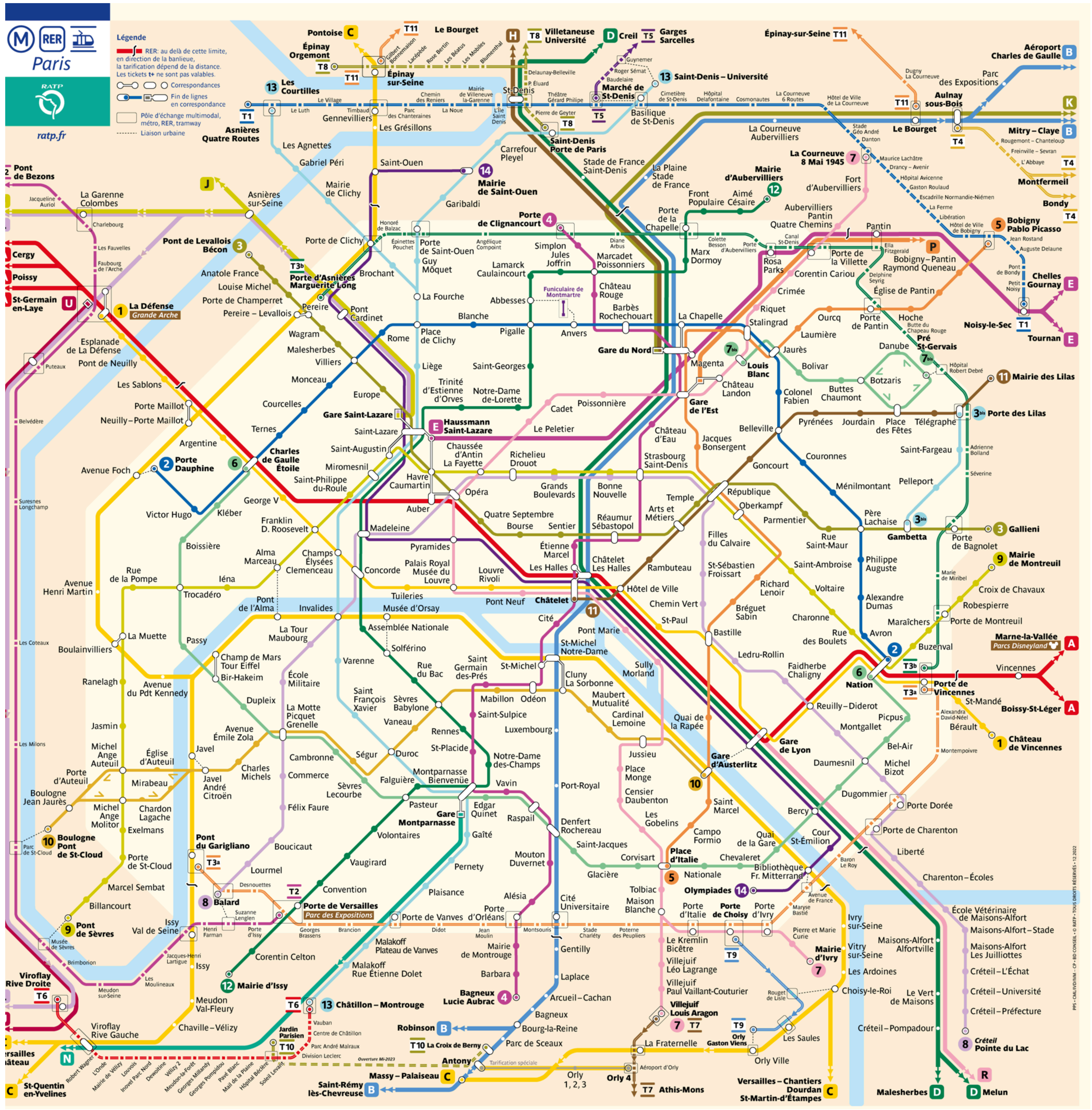
- carte routière et graphe des connexions entre villes avec la distance en km

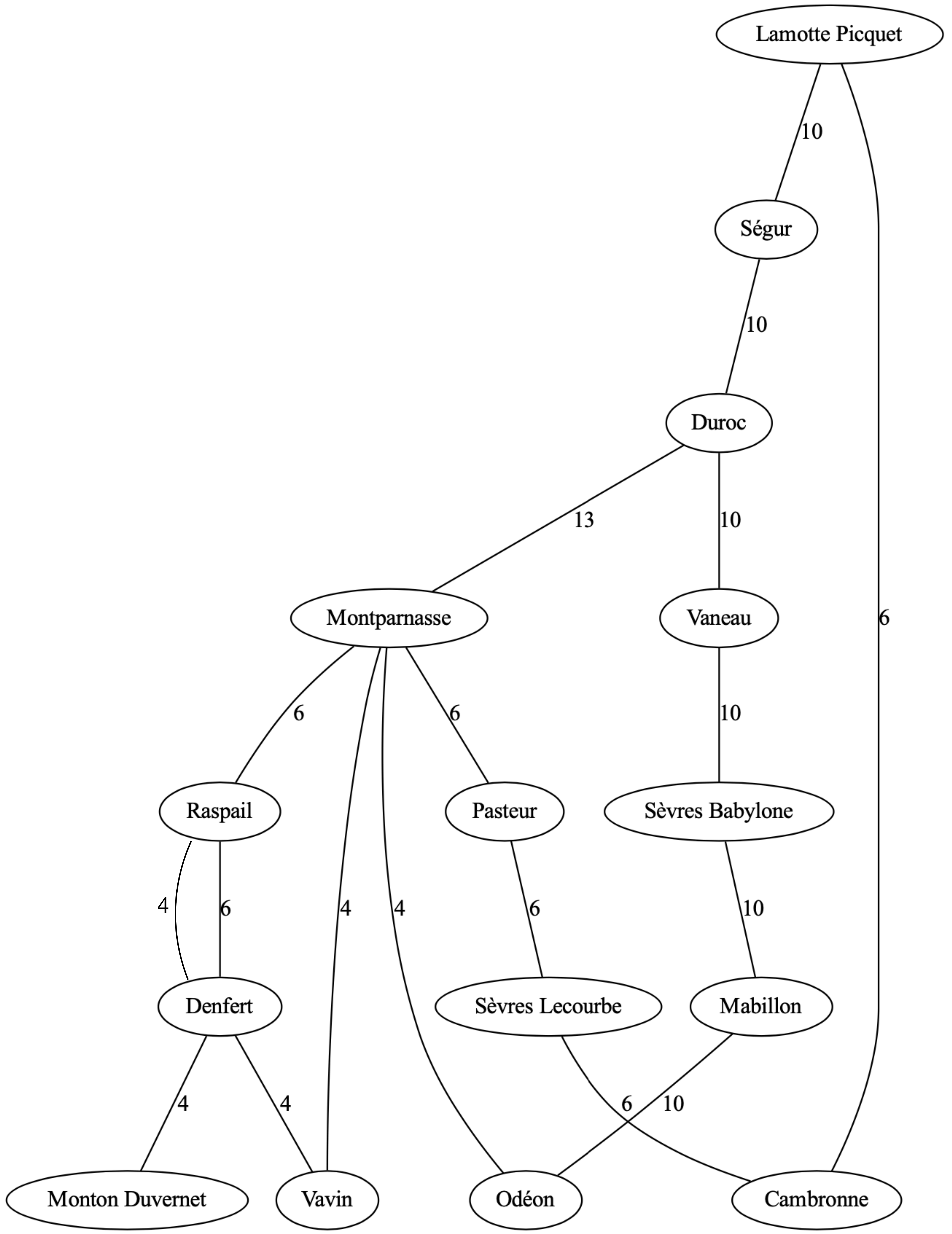
Graphes




Havre Caumartin
La Fayette
Grands Boulevards
Château d?Eau
Colonel Fabien
Auber
Richelieu Drouot
Jacques Bonsergent
Belleville
Quatre Septembre
Bonne Nouvelle
Goncourt
Opéra
Strasbourg St-Denis
Bourse
Sentier
République
Temple
Parmentier
Réaumur Sébastopol
Arts et Métiers
Rue St-Maur
Pyramides
Étienne Marcel
Filles du Calvaire
Oberkampf
Tuileries
Les Halles
Rambuteau
Louvre Rivoli
St-Sébastien Froissart
Châtelet Les Halles
Richard Lenoir
Voltaire
Pont Neuf
Chemin Vert
Hôtel de Ville
Bréguet Sabin
Châtelet
11
St-Paul
Cité
Pont Marie
Sully
Bastille
Ledru Rollin
St-Michel
St-Michel Notre-Dame
Morland
Faidherbe Chaligny
Mabillon
Odéon
Maubert Mutualité
Reuilly-Diderot
St-Sulpice
Cluny La Sorbonne
Quai de la Rapée
Cardinal Lemoine
Notre-Dame des-Champs
Jussieu
Gare de Lyon
Luxembourg
Montparnasse Bienvenüe
Place Monge
Gare d?Austerlitz
Vavin
Port Royal
St-Marcel
Bercy
Edgar Quinet
Censier Daubenton
Quai de la Gare
Raspail
Denfert Rochereau
Campo Formio
St-Jacques
Chevaleret
Les Gobelins
Mouton Duvernet
Glacière
14
Corvisart
Nationale
Place d?Italie
5
Porte d'Ivry
Alésia
Tolbiac
Cité Universitaire
Maison Blanche
Pierre curie

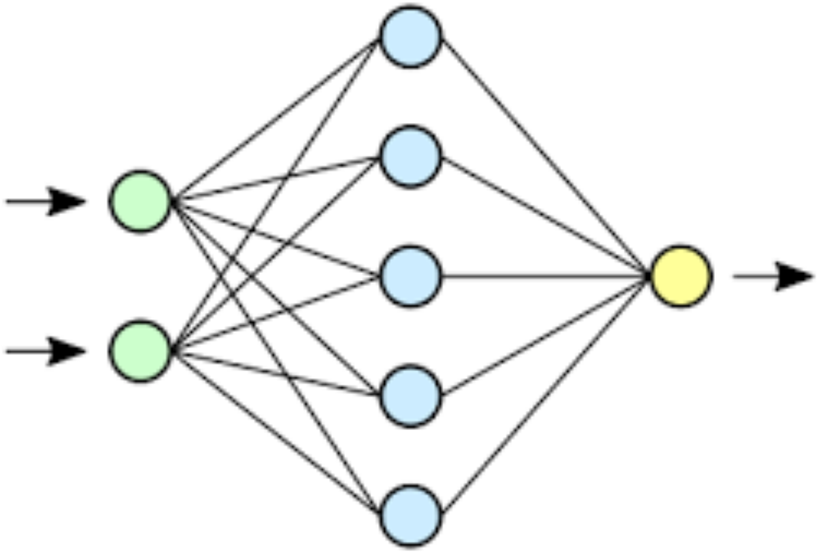
Graphes

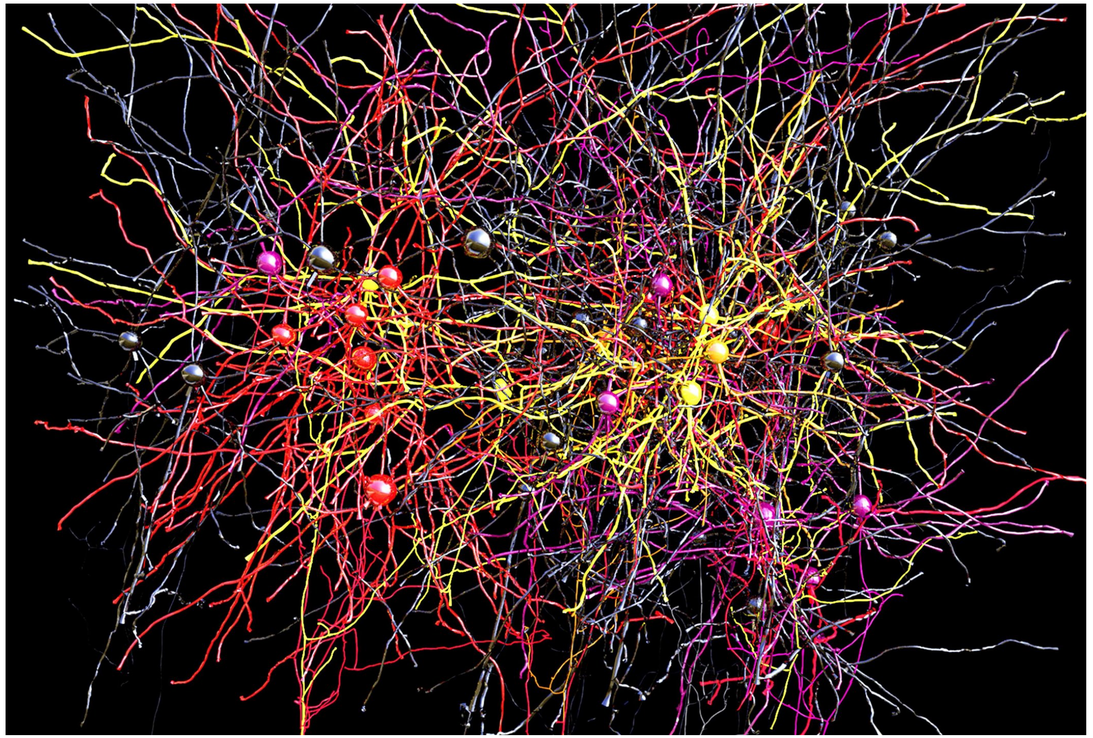




- plan du métro et le graphe qui relie les différentes stations

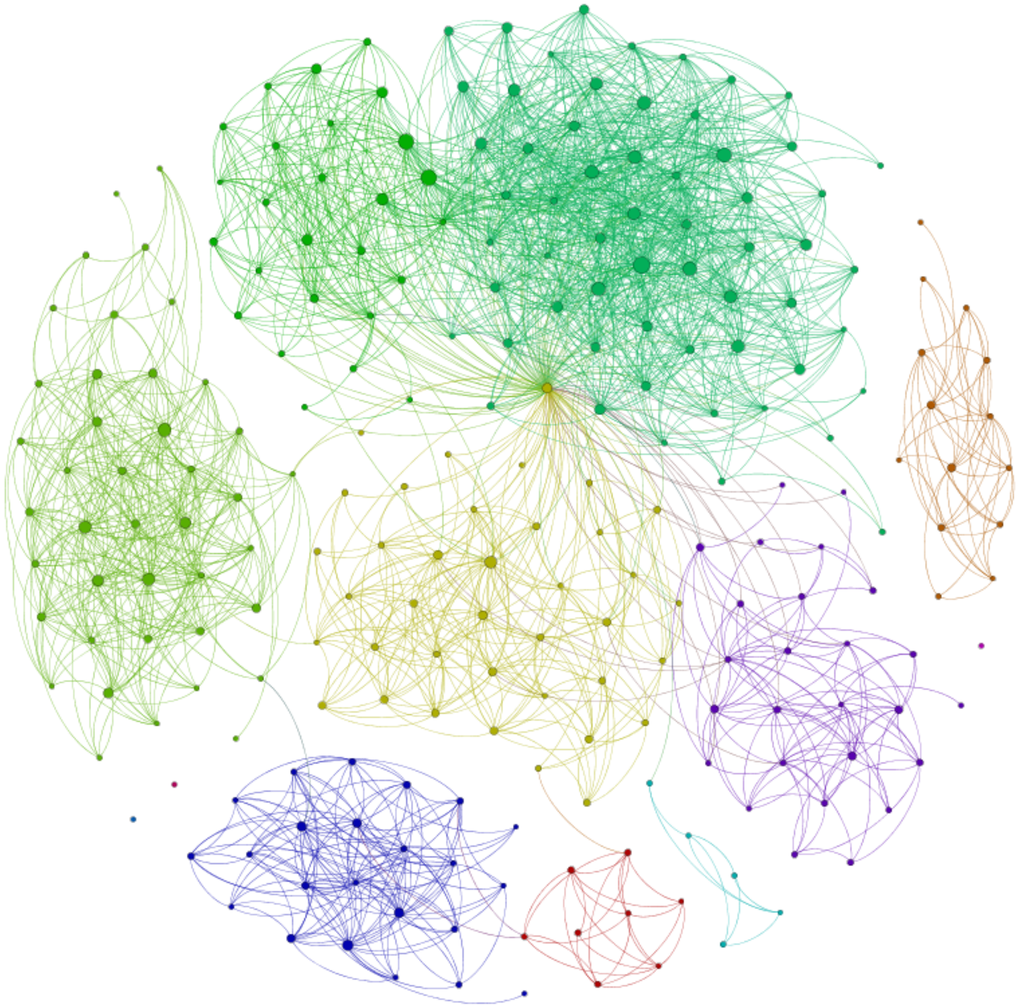
Graphes





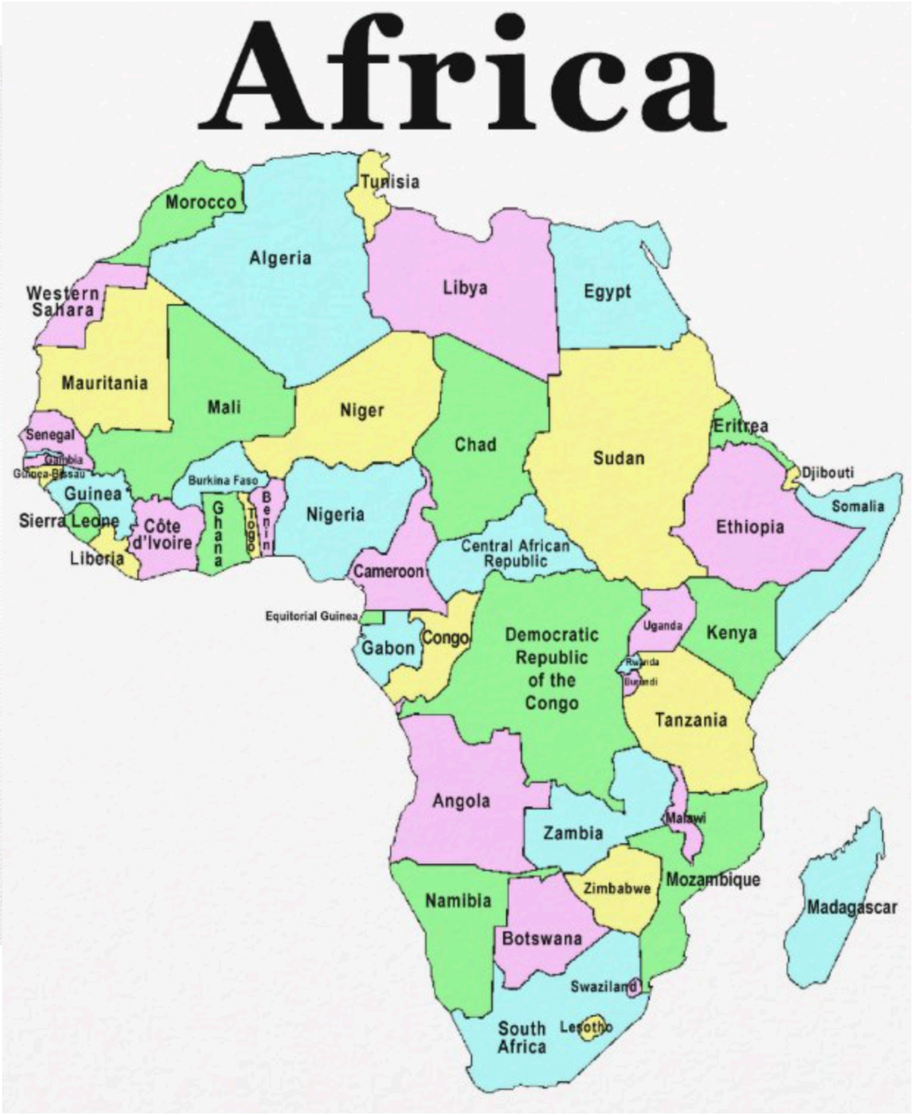
- Réseaux de neurone 2 couches et vrais réseaux de neurones biologiques

Graphes



- Réseaux d’amis sur Facebook

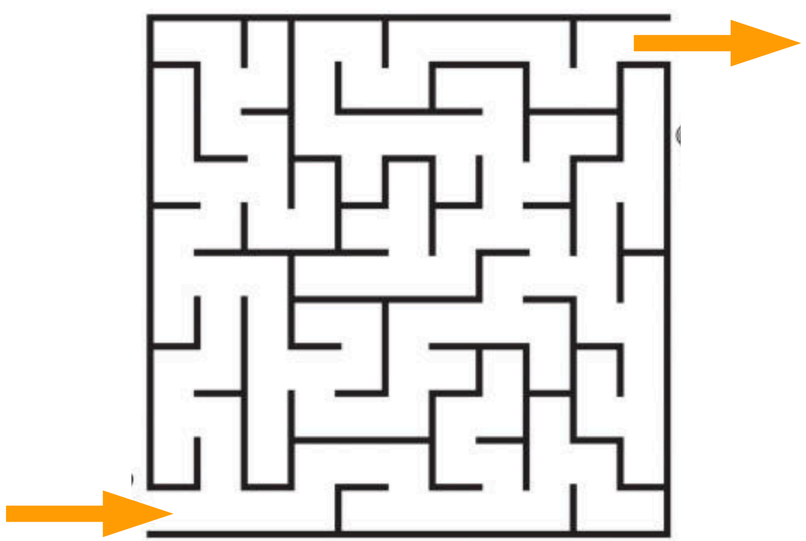
Graphes

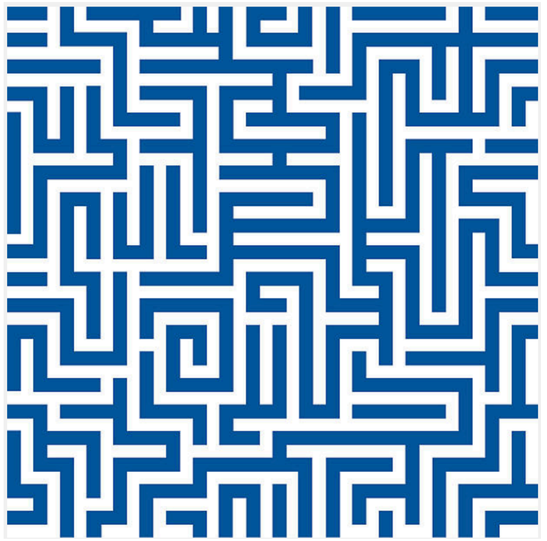


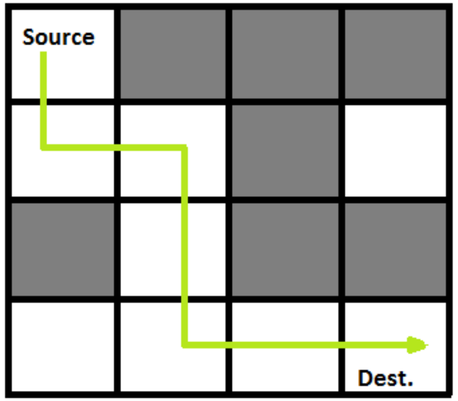
- Graphes planaires coloriables avec 4 couleurs

Labyrinthes

- trouver le chemin vers la sortie !

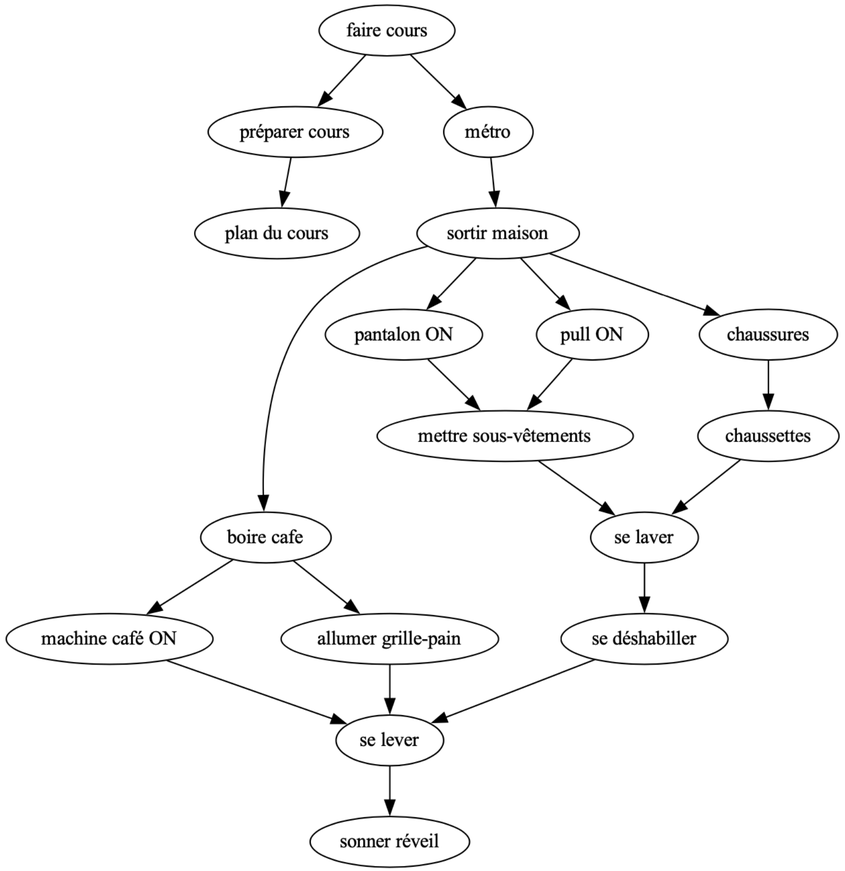


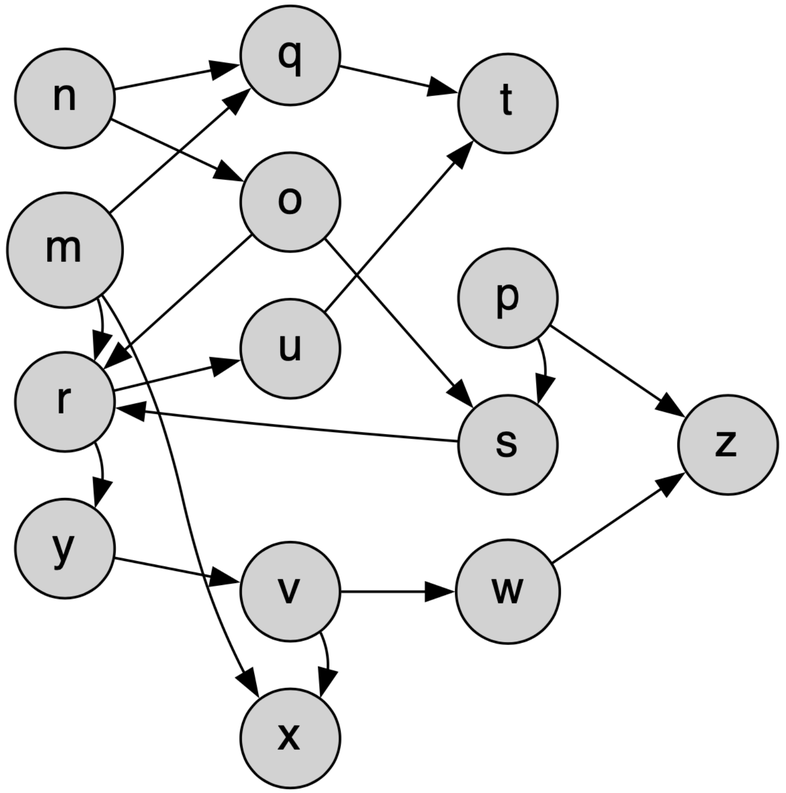




Graphe acyclique

- exemples sans cycles





Exercice Ecrire un programme qui teste si un graphe est sans cycle

Graphes

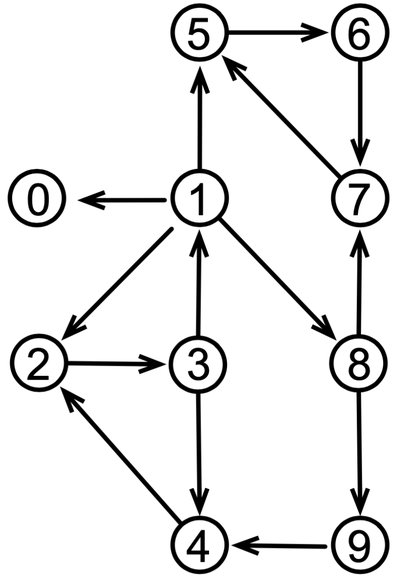
- un graphe est formé par un ensemble de **sommets** (*vertices*) et d'**arrêtes** (*edges*)
- une arrête relie 2 sommets (**origine** et **extrémité**)
- un graphe peut être **non-orienté** s'il existe une arrête (w, v) pour toute arrête (v, w)
- parfois on parle aussi d'**arcs** pour les arrêtes et de **noeuds** pour les sommets

Remarque: un arbre est un cas particulier d'un graphe (sommets = noeuds, arrêtes = branches)

Représentation des graphes

- on définit une classe pour les graphes

```
class Graph :
    def __init__ (self, vs, edges) :
        self.vertices = vs
        self.succ = {v : {v2 for (v1, v2) in edges if v1 == v}
                     for v in vs}
        self.pred = {v : {v1 for (v1, v2) in edges if v2 == v}
                     for v in vs}

    def __str__ (self) :
        return 'sommets = {}\nsucc = {}\npred = {}'.format (
                self.vertices, self.succ, self.pred)
```



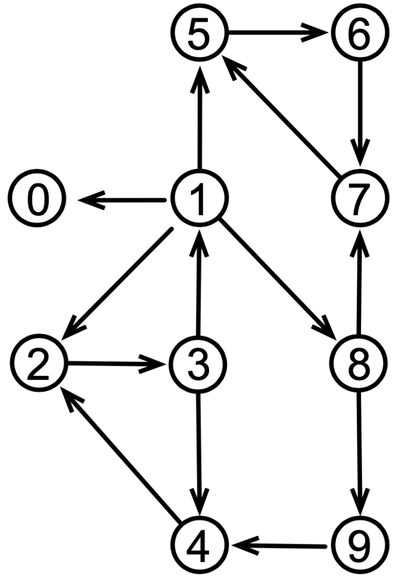
- et on construit un graphe

```
g1 = Graph ({i for i in range(10)},
 {(1,0), (1,2), (1,5), (1,8), (2,3),
  (3,1), (3,4), (4,2), (5,6), (6,7),
  (7,5), (8,7), (8,9), (9,4)})

print (g1)
```

Représentation des graphes

- on rajoute 2 méthodes plus explicites pour la suite

```
class Graph :
    def __init__ (self, vs, edges) :
        self.vertices = vs
        self.succ = {v : {v2 for (v1, v2) in edges if v1 == v}
                     for v in vs}
        self.pred = {v : {v1 for (v1, v2) in edges if v2 == v}
                     for v in vs}

    def __str__ (self) :
        return 'sommets = {}\nsucc = {}\npred = {}'.format (
                self.vertices, self.succ, self.pred)

    def successors (self, v) :
        return self.succ[v]

    def predecessors (self, v) :
        return self.pred[v]
```




```
print (g1.successors (0))
set()
print (g1.successors (1))
{8, 0, 2, 5}
```

[texte des programmes]

Représentation des graphes

- on construit le graphe du métro parisien

```
stations = {'lamotte-picquet', 'segur', 'duroc', 'montparnasse', 'raspail', 'pasteur',
            'sevres-lecourbe', 'denfert-rochereau', 'mouton-duvernet', 'vavin',
            'cambronne', 'vaneau', 'sevres-babylone', 'mabillon', 'odeon'}

lignes = { ('lamotte-picquet', 'segur'), ('lamotte-picquet', 'cambronne'),
           ('cambronne', 'sevres-lecourbe'), ('sevres-lecourbe', 'pasteur'),
           ('pasteur', 'montparnasse'), ('montparnasse', 'raspail'),
           ('raspail', 'denfert-rochereau'), ('denfert-rochereau', 'mouton-duvernet'),
           ('denfert-rochereau', 'vavin'), ('vavin', 'montparnasse'),
           ('segur', 'duroc'), ('duroc', 'montparnasse'), ('duroc', 'vaneau'),
           ('vaneau', 'sevres-babylone'), ('sevres-babylone', 'mabillon'),
           ('mabillon', 'odeon'), ('odeon', 'montparnasse') }
```

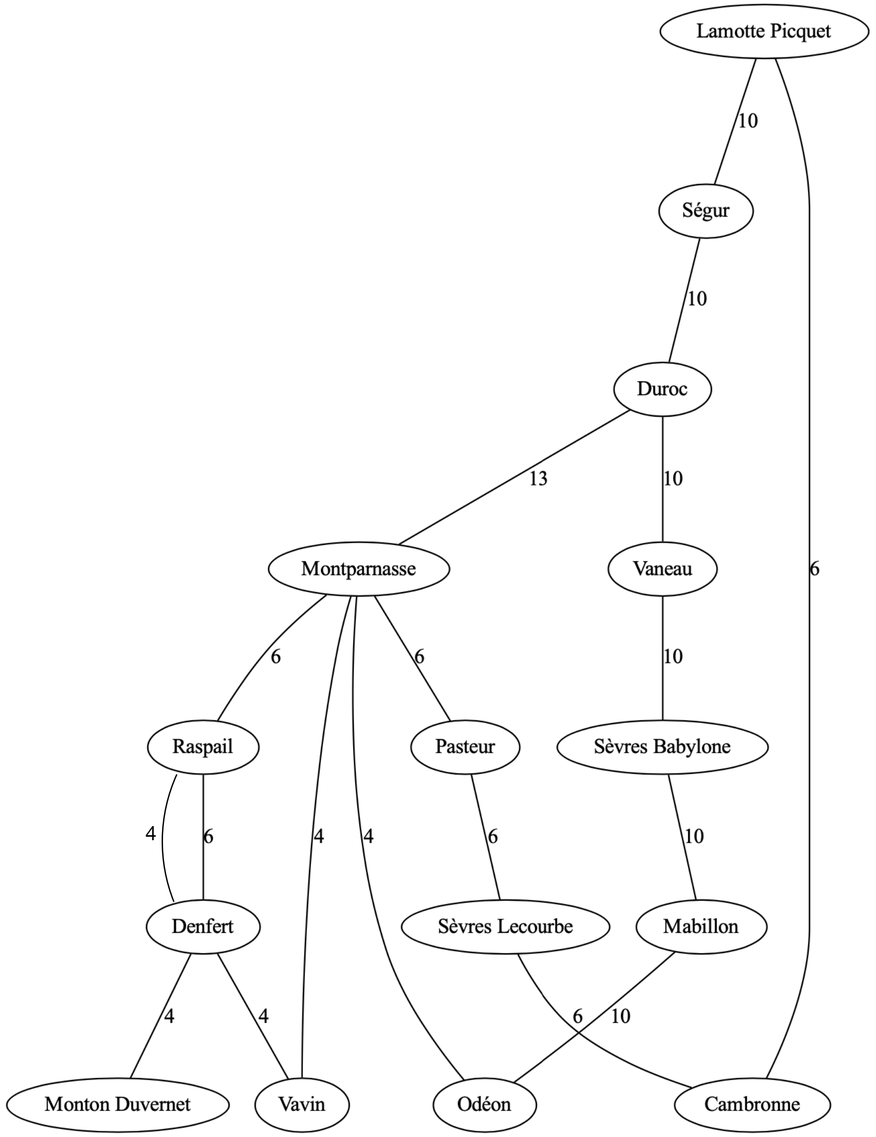
- on le transforme en un graphe non-orienté

```
def mk_symetric (edges) :
    r = set()
    for (v1, v2) in edges :
        r.add ((v1,v2))
        r.add ((v2, v1))
    return r

lignes = mk_symetric (lignes)

metro = Graph (stations, lignes)
```

- et si on veut tout le réseau parisien . . . (il faudra dé-symétriser les bouts de lignes 10 et 7)

METRO-PARIS

fichier texte qui contient les lignes de métro parisien

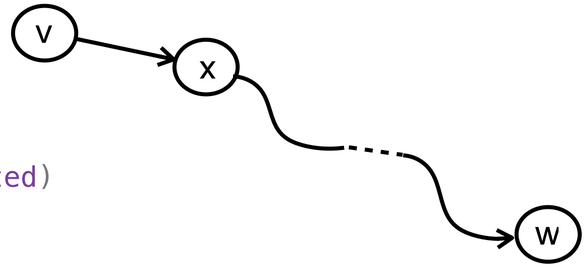


[texte des programmes]

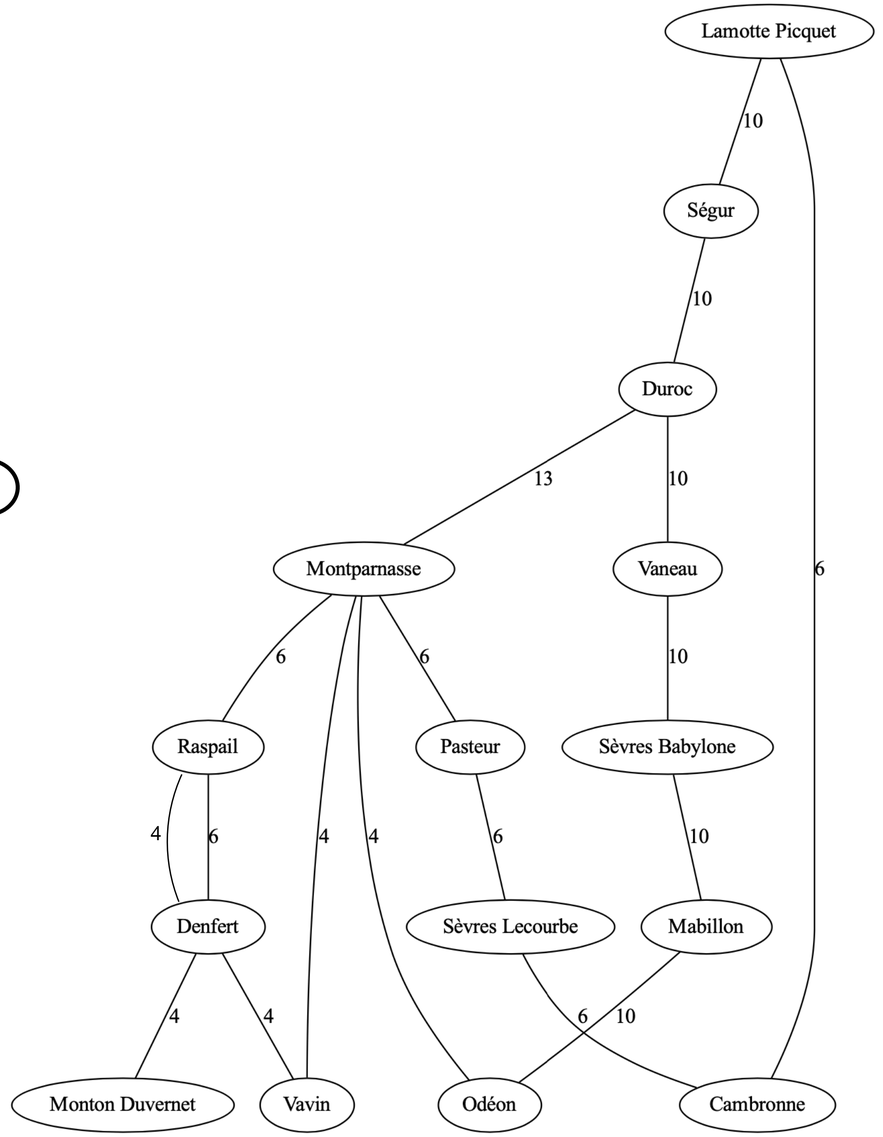
Chemins

- recherche d'un chemin entre 2 sommets

```
def chemin1 (g, v, w, visited) :
    visited [v] = True
    if v == w :
        return [v]
    else :
        for x in g.successors(v) :
            if not visited[x] :
                ch = chemin1 (g, x, w, visited)
                if ch != [ ] :
                    return [v] + ch
    return [ ]

def chemin (g, v, w) :
    return chemin1 (g, v, w,
        {v:False for v in g.vertices})
```




```
print (chemin (metro, 'segur', 'mouton-duvernet'))
```



- c'est la méthode du fil d'Ariane (avec les cailloux du petit Poucet)

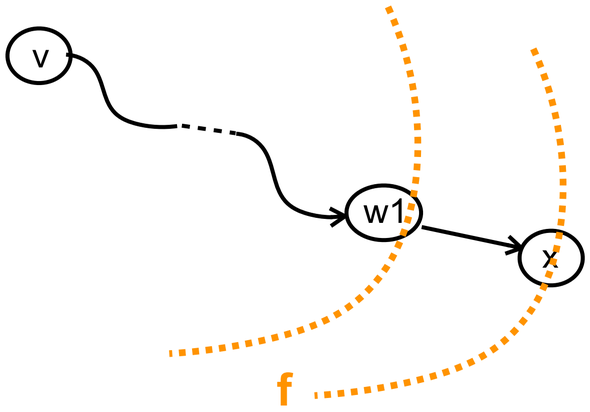
- méthode d'exploration de graphe par profondeur d'abord
(depth first search)

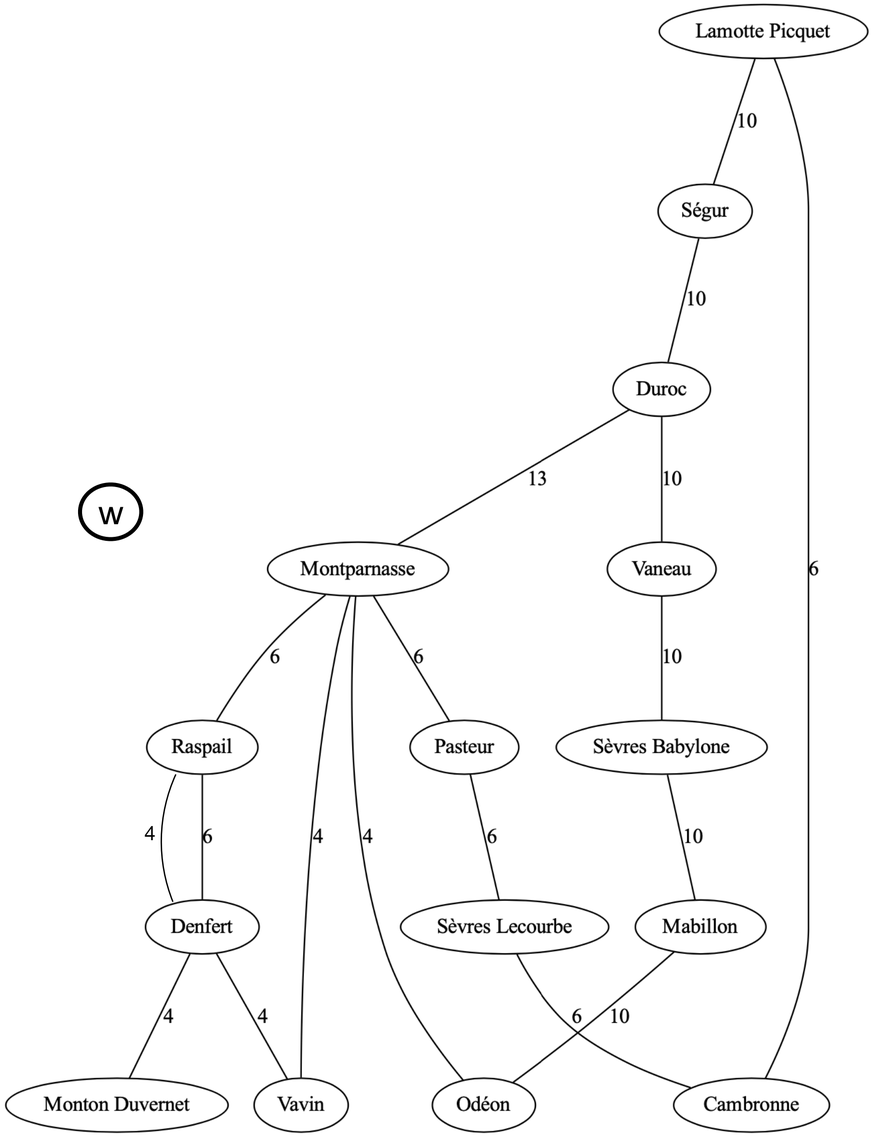
[texte des programmes]

Chemins

- recherche d'un chemin de longueur minimale entre 2 sommets

```
def chemin_min1 (g, v, f, w, visited) :
    ch = {x: [] for x in g.vertices }
    ch[v] = [v]
    while not f.is_empty() :
        w1 = f.deq()
        if w1 == w :
            return ch[w]
        else :
            for x in g.successors(w1) :
                if not visited[x] :
                  f.enq(x); visited[x] = True
                  ch[x] = ch[w1] + [x]

    return [ ]

def chemin_min (g, v, w) :
    f = Queue(); visited = {x: False for x in g.vertices}
    f.enq(v); visited[v] = True
    return chemin_min1 (g, v, f, w, visited)


print (chemin_min (metro, 'segur', 'mouton-duvernet'))
```





- méthode d'exploration de graphe par largeur d'abord
(breadth first search)

[texte des programmes]

Chemins

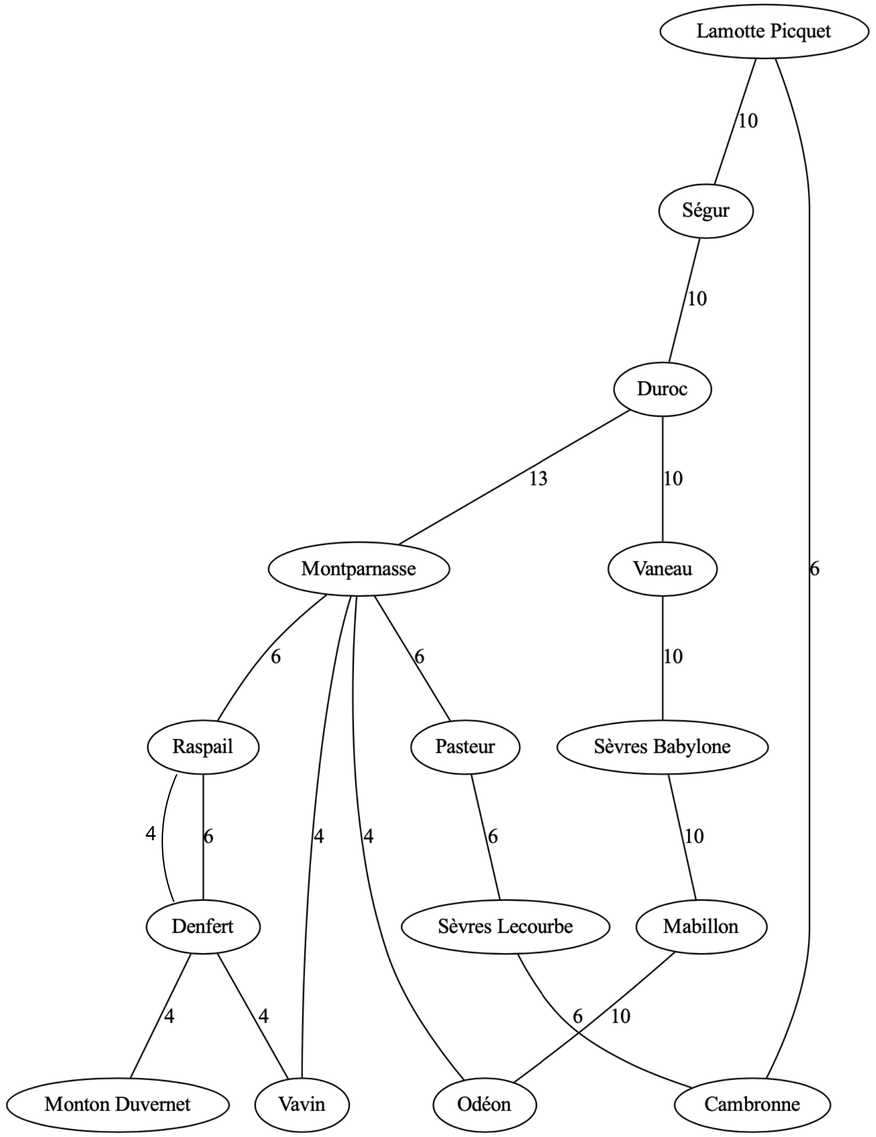
- on construit le graphe du métro parisien en ajoutant le nom des lignes

```
stations = {'lamotte-picquet', 'segur', 'duroc', 'montparnasse', 'raspail', 'pasteur',
            'sevres-lecourbe', 'denfert-rochereau', 'mouton-duvernet', 'vavin',
            'cambronne', 'vaneau', 'sevres-babylone', 'mabillon', 'odeon'}

lignesV = { ('lamotte-picquet', 10, 'segur'), ('lamotte-picquet', 6, 'cambronne'),
          ('cambronne', 6, 'sevres-lecourbe'), ('sevres-lecourbe', 6, 'pasteur'),
          ('pasteur', 6, 'montparnasse'), ('montparnasse', 6, 'raspail'),
          ('raspail', 4, 'denfert-rochereau'), ('denfert-rochereau', 4, 'mouton-duvernet'),
          ('denfert-rochereau', 4, 'vavin'), ('vavin', 4, 'montparnasse'),
          ('segur', 10, 'duroc'), ('duroc', 13, 'montparnasse'), ('duroc', 10, 'vaneau'),
          ('vaneau', 10, 'sevres-babylone'), ('sevres-babylone', 10, 'mabillon'),
          ('mabillon', 10, 'odeon'), ('odeon', 4, 'montparnasse'),
          ('raspail', 6, 'denfert-rochereau')}


def mk_symetricV (edges) :
    r = set()
    for (v1, x, v2) in edges :
        r.add ((v1, x, v2))
        r.add ((v2, x, v1))
    return r


lignesV = mk_symetricV (lignesV)

metroV = GraphV (stations, lignesV)
```

- et si on veut tout le réseau parisien . . . (il faudra dé-symétriser les bouts de lignes 10 et 7)

  METRO-PARIS

  fichier texte qui contient les lignes de métro parisien



[texte des programmes]

Graphes valués

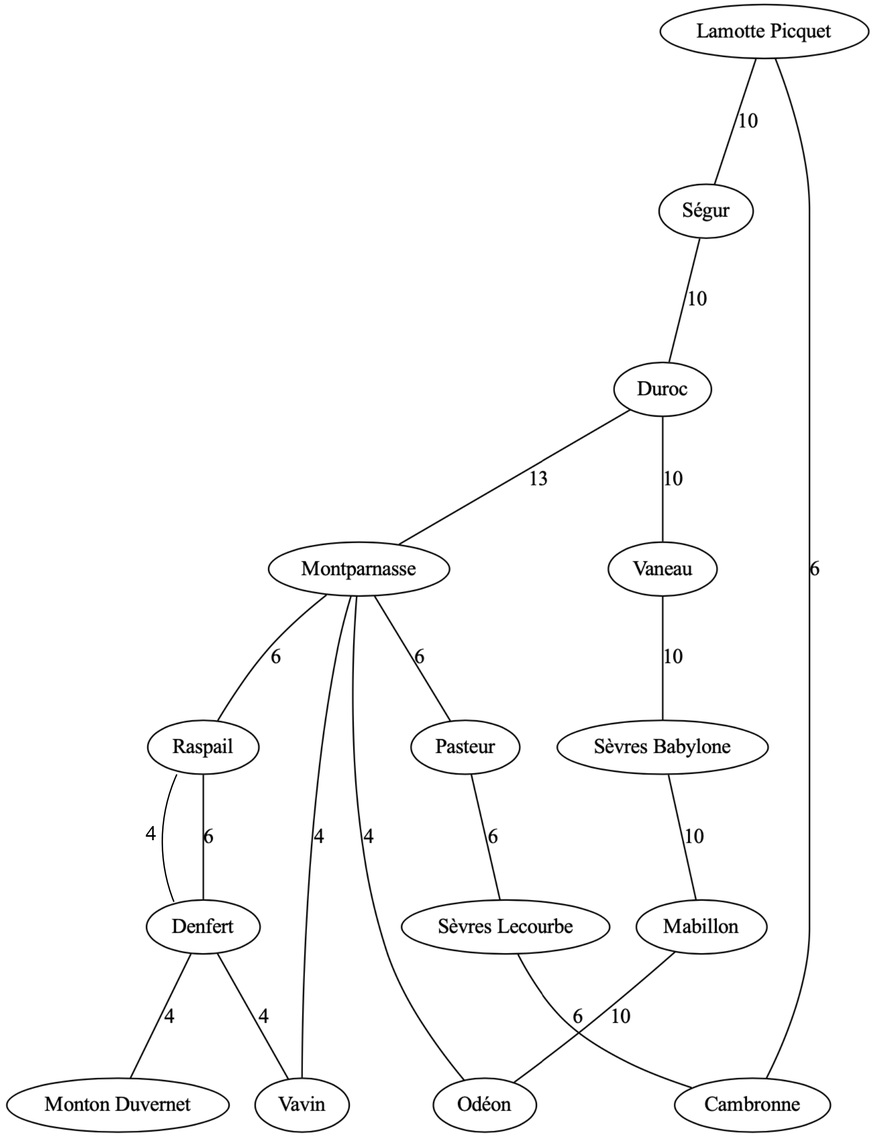
- on rajoute 2 méthodes plus explicites pour la suite

```
class GraphV (Graph):
    def __init__ (self, vs, edges) :
        self.vertices = vs
        self.succ = {v : {v2 for (v1, x, v2) in edges if v1 == v}
                     for v in vs}
        self.pred = {v : {v1 for (v1, x, v2) in edges if v2 == v}
                     for v in vs}
        self.weight = {(v1, v2): {x for (w1, x, w2) in edges
                                  if w1 == v1 and w2 == v2}
                       for (v1, x, v2) in edges}
```

- les autres méthodes et fonctions sont inchangées (grâce à l'héritage)

```
print (metroV.successors ('segur'))
{'duroc', 'lamotte-picquet'}
print (metroV.successors ('raspail'))
{'denfert-rochereau', 'montparnasse'}
print (metroV.weight ['raspail', 'denfert-rochereau'])
{4, 6}
print (metroV.weight ['segur', 'duroc'])
{10}
```

```
ch = chemin_min (metroV, 'segur', 'odeon')
for i in range (len(ch)-1) :
    v1 = ch[i]; v2 = ch[i+1]; x = metroV.weight[v1, v2]
    print (f'de {v1} à {v2} via la ligne {x}')
```



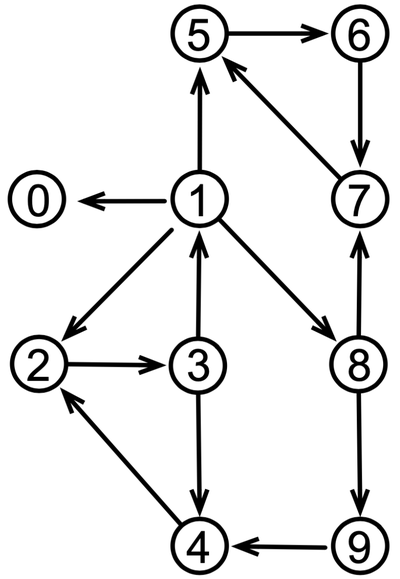
[texte des programmes]

Représentation par matrices d'adjacence

- matrice d'adjacence

```
   0  1  2  3  4  5  6  7  8  9
0:
1: *     *        *           *
2:          *
3:    *        *
4:       *
5:                *
6:                   *
7:             *
8:                   *        *
9:             *
```

$M_{i,j} =$ `Vrai` ssi il existe une arrête de v_i vers v_j



- les graphes non-orientés ont une matrice d'adjacence symétrique

- cette représentation est couteuse en place mémoire si peu d'arrêtes

[texte des programmes]

Prochain cours

- calcul flottant
- norme IEEE
- recherche de minimum
- régression linéaire