Informatique et Programmation

Cours 14

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

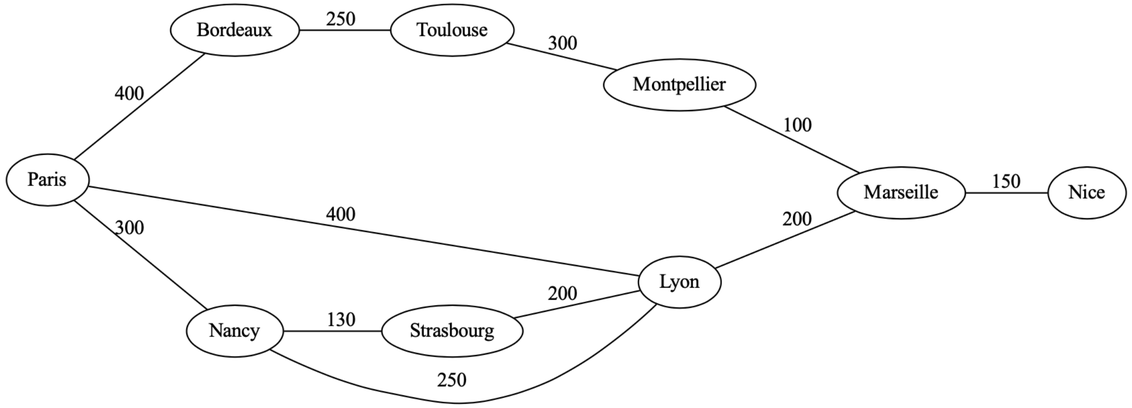
http://jeanjacqueslevy.net/prog-py

Plan

- graphes
- graphes (représentation avec matrice de connexion)
- graphes (représentation avec listes de successeurs)
- chemin entre deux sommets

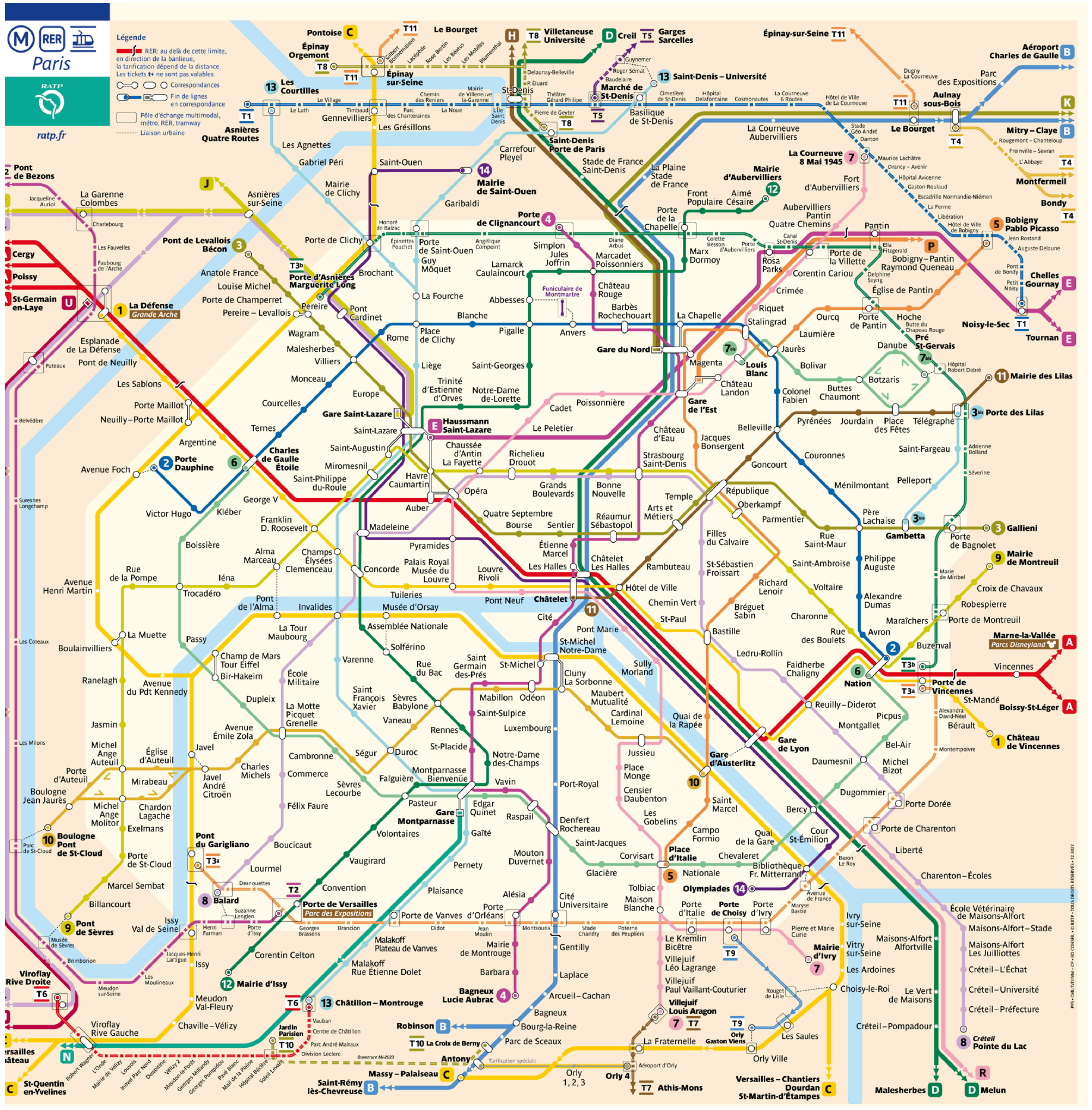
dès maintenant: télécharger Python 3 en `http://www.python.org`

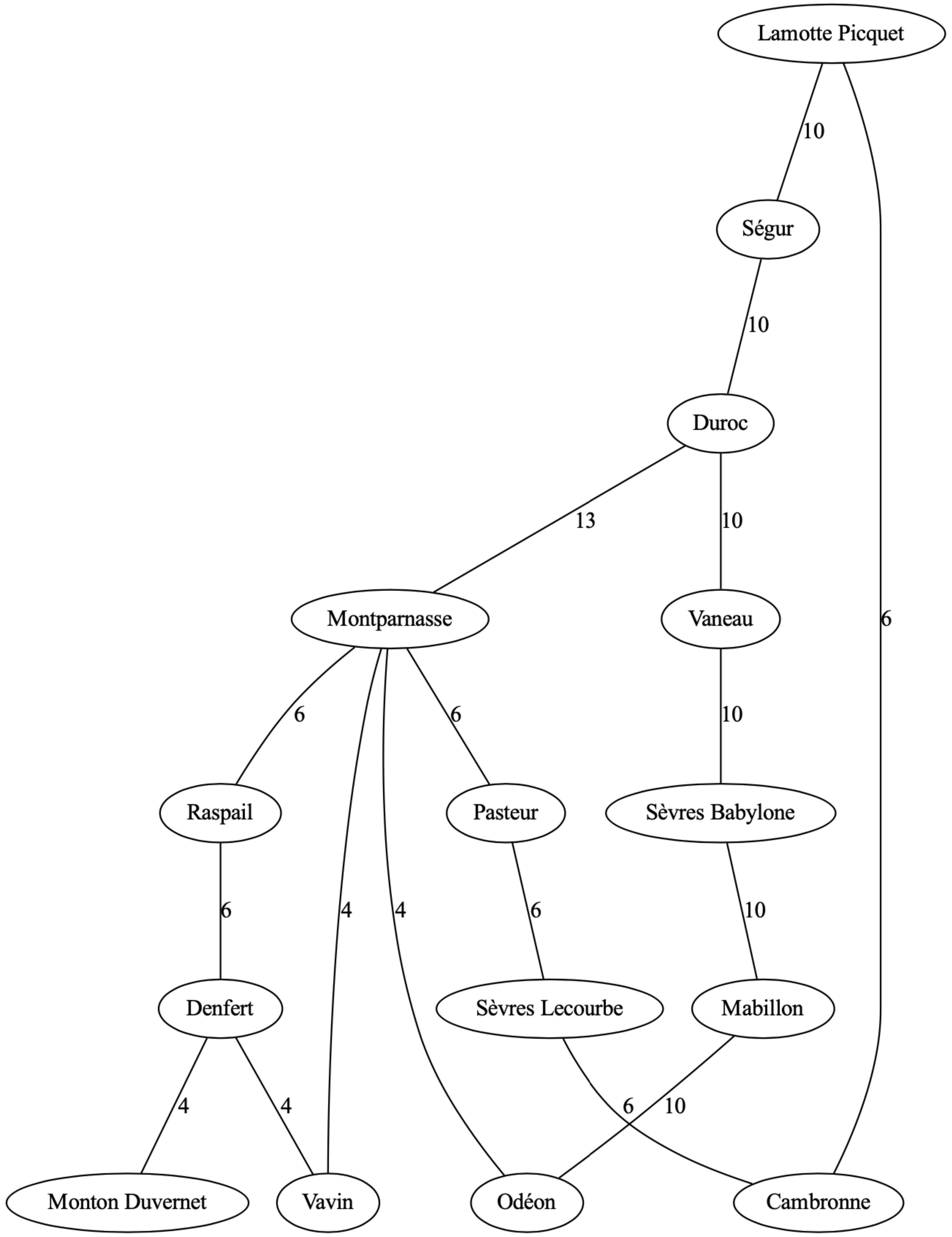
Graphes



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

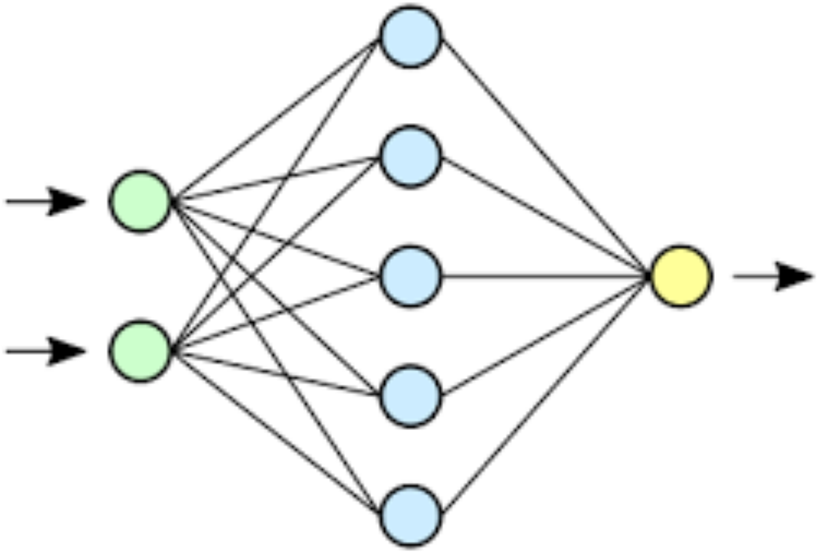
Graphes

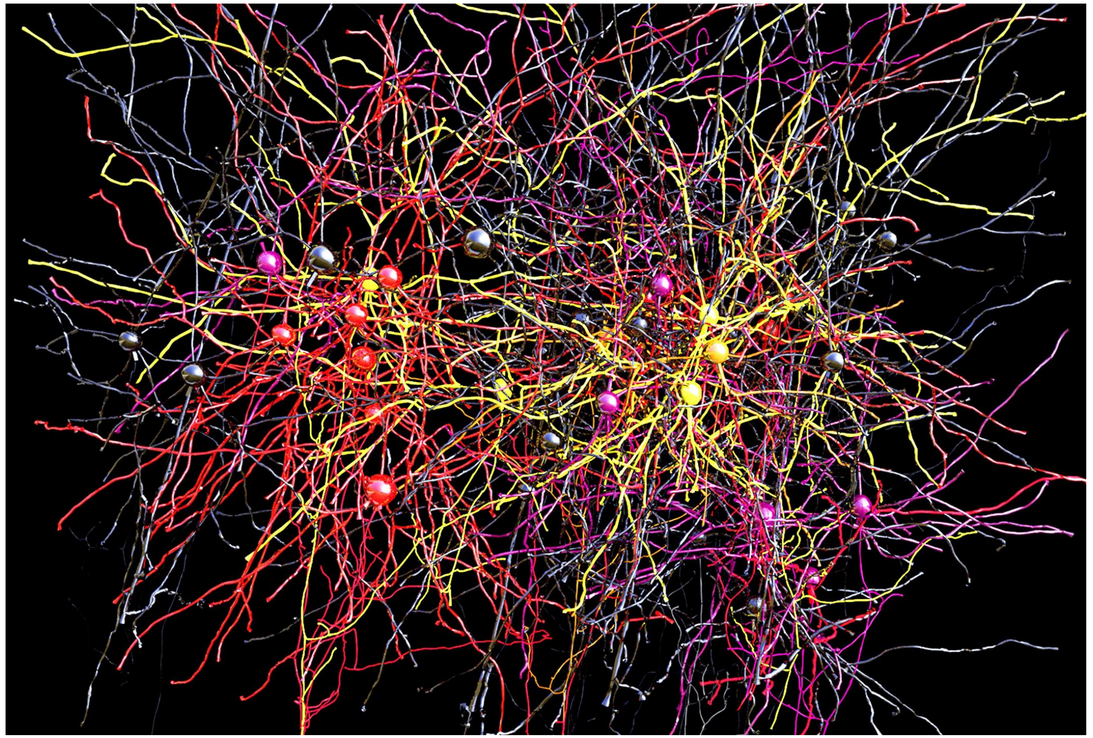




- plan du métro et le graphe qui relie les différentes stations

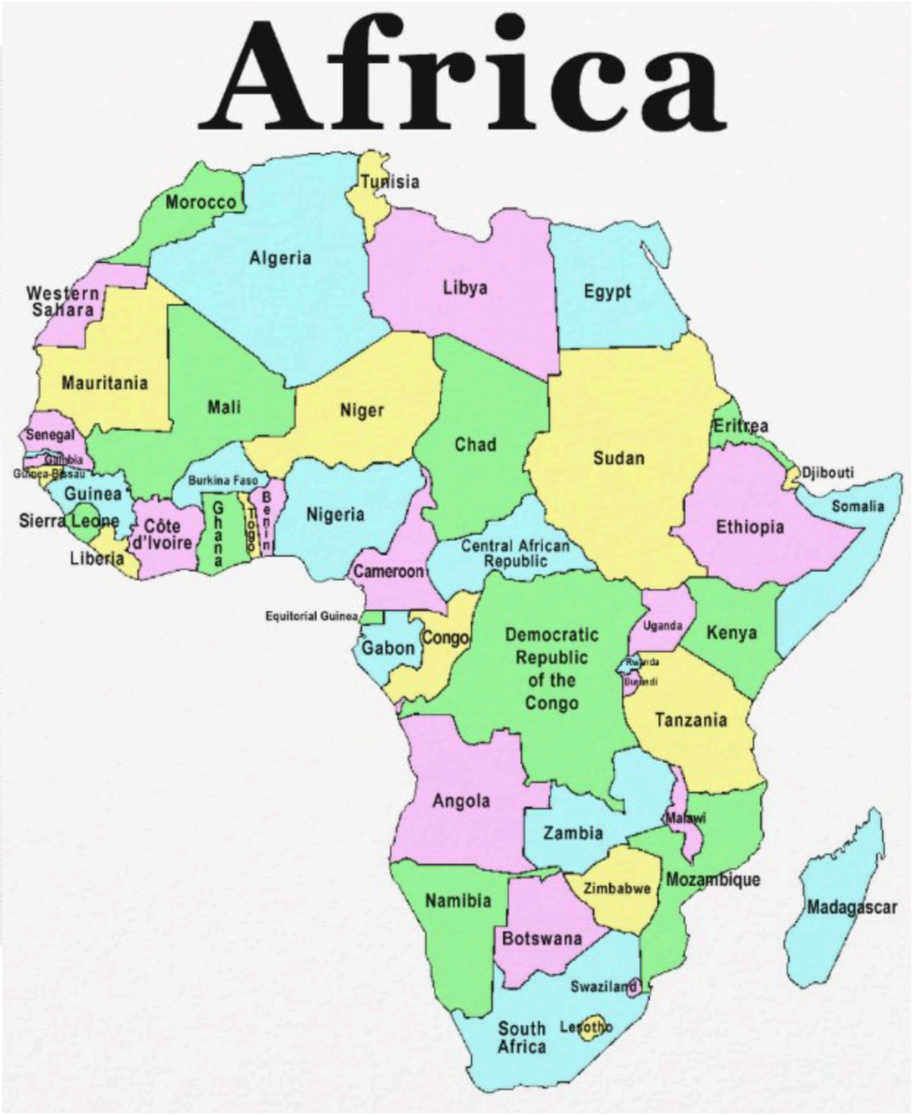
Graphes





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

Graphes

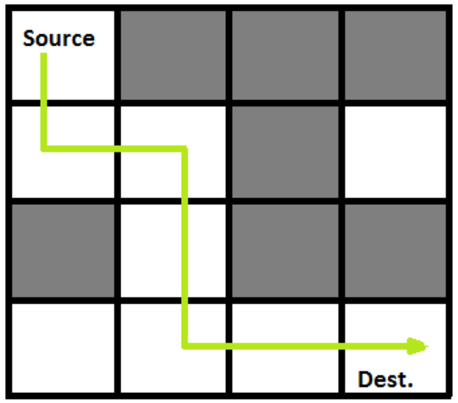


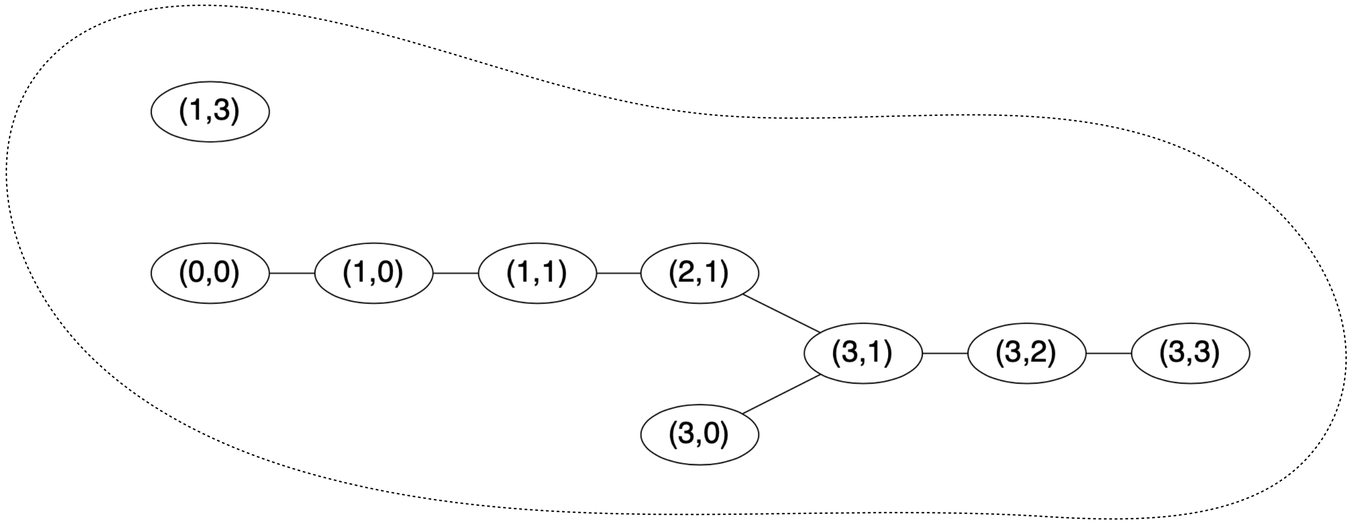
- Graphes planaires coloriables avec 4 couleurs

Graphes

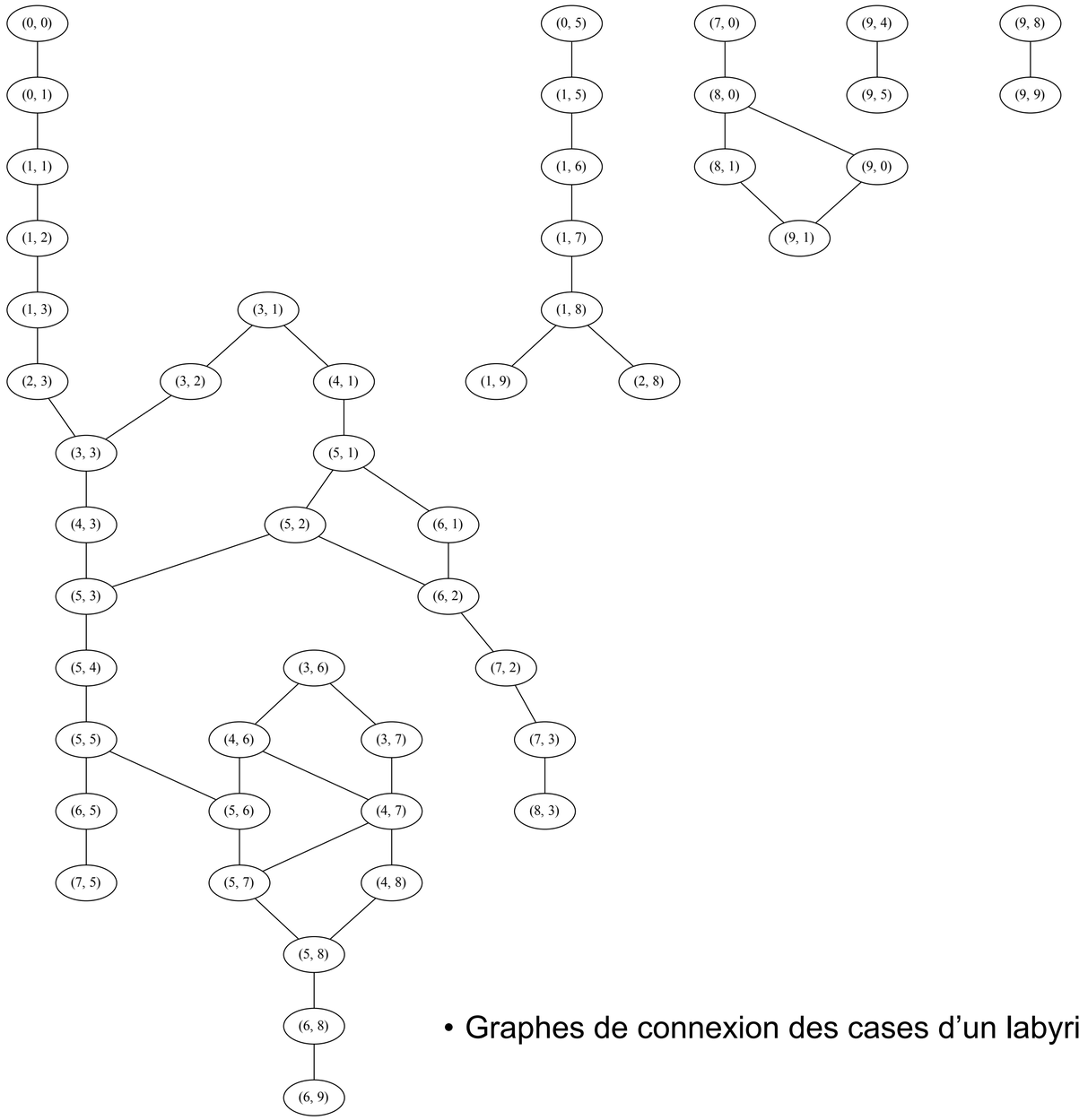
- Graphes de connexion des cases d'un labyrinthe

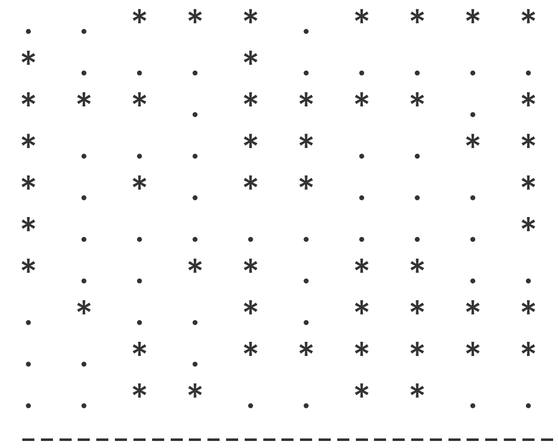
```
maze = [[0, 1, 1, 1],
        [0, 0, 1, 0],
        [1, 0, 1, 1],
        [0, 0, 0, 0]]
```





Graphes





```
m1 = [[0, 0, 1, 1, 1, 0, 1, 1, 1, 1],
      [1, 0, 0, 0, 1, 0, 0, 0, 0, 0],
      [1, 1, 1, 0, 1, 1, 1, 1, 0, 1],
      [1, 0, 0, 0, 1, 1, 0, 0, 1, 1],
      [1, 0, 1, 0, 1, 1, 0, 0, 0, 1],
      [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
      [1, 0, 0, 1, 1, 0, 1, 1, 0, 0],
      [0, 1, 0, 0, 1, 0, 1, 1, 1, 1],
      [0, 0, 1, 0, 1, 1, 1, 1, 1, 1],
      [0, 0, 1, 1, 0, 0, 1, 1, 0, 0]]
```

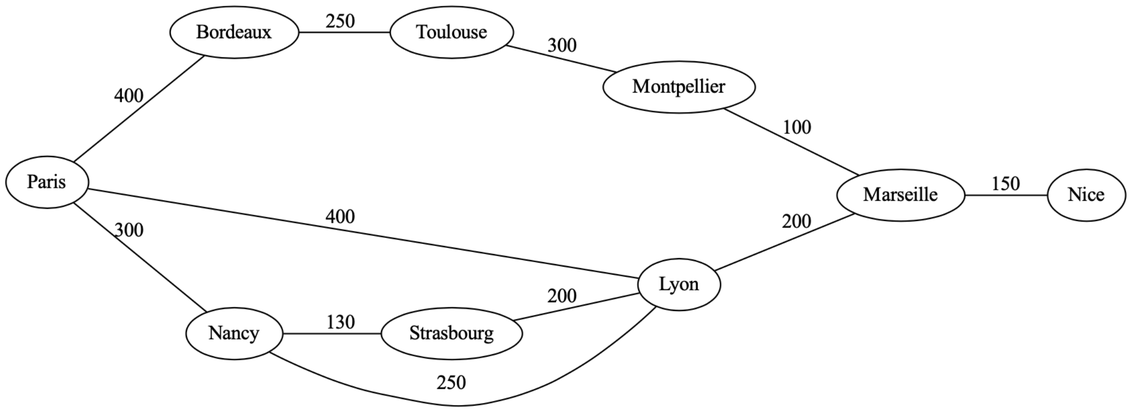
- Graphes de connexion des cases d’un labyrinthe

Graphes (représentation 1)

- Représentation par matrice d'adjacence

```
villes = [ 'Paris', 'Bordeaux', 'Toulouse', 'Montpellier',
     'Marseille', 'Nancy', 'Strasbourg', 'Lyon', 'Nice']
nVilles = len (villes)


graphe =  new_matrix (nVilles, nVilles, None)
```




```
     0   1   2   3   4   5   6   7   8
-------------------------------------------
0:       400                 300       400
1:  400      250
2:       250       300
3:             300       100
4:                   100                 200 150
5:  300                                130 250
6:                              130       200
7:  400                   200 250 200
8:                        150
```

Graphes (représentation 1)

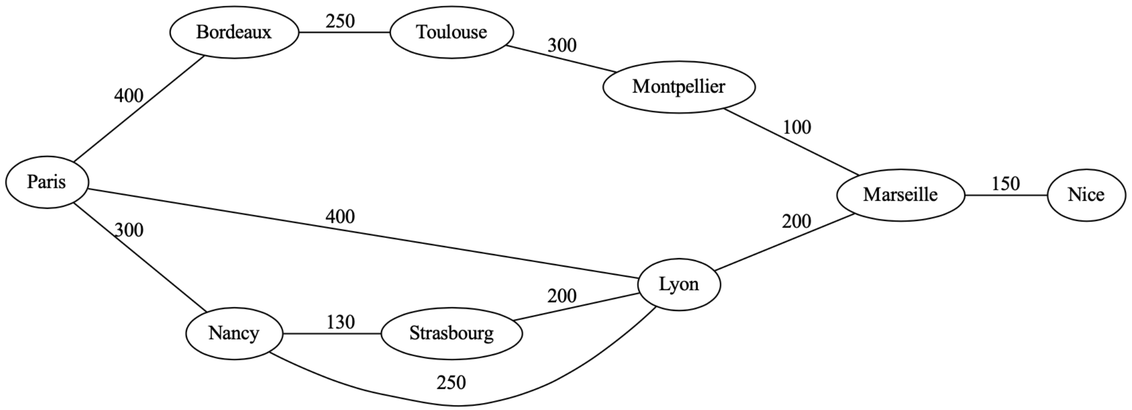
- Représentation par matrice d'adjacence

```
villes = [ 'Paris', 'Bordeaux', 'Toulouse', 'Montpellier',
     'Marseille', 'Nancy', 'Strasbourg', 'Lyon', 'Nice']
nVilles = len (villes)


graphe =  new_matrix (nVilles, nVilles, None)


def arc (g, v1, v2, d) :
    i = villes.index (v1)
    j = villes.index (v2)
    g[i][j] = g[j][i] = d


arc (graphe, 'Paris', 'Bordeaux', 400)
arc (graphe, 'Bordeaux', 'Toulouse', 250)
arc (graphe, 'Toulouse', 'Montpellier', 300)
arc (graphe, 'Montpellier', 'Marseille', 100)
arc (graphe, 'Paris', 'Lyon', 400)
arc (graphe, 'Paris', 'Nancy', 300)
arc (graphe, 'Nancy', 'Strasbourg', 130)
arc (graphe, 'Strasbourg', 'Lyon', 200)
arc (graphe, 'Nancy', 'Lyon', 250)
arc (graphe, 'Lyon', 'Marseille', 200)
arc (graphe, 'Marseille', 'Nice', 150)
```



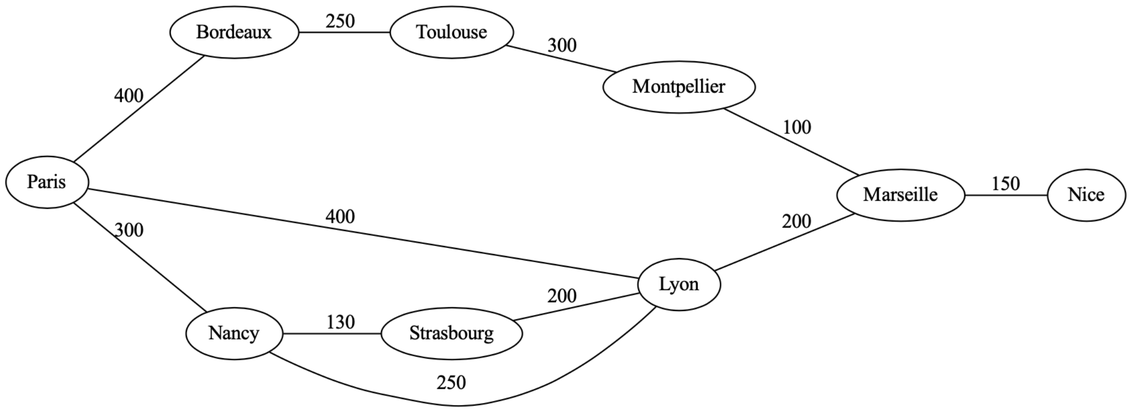

```
     0   1   2   3   4   5   6   7   8
0:       400                 300       400
1:  400      250
2:       250       300
3:           300       100
4:                 100                 200 150
5:  300                             130 250
6:                             130       200
7:  400                   200 250 200
8:                        150
```

Graphes (représentation 2)

- Représentation par tableau de listes d'adjacence

```
villes = [ 'Paris', 'Bordeaux', 'Toulouse', 'Montpellier',
     'Marseille', 'Nancy', 'Strasbourg', 'Lyon', 'Nice']
nVilles = len (villes)


graphe = [
    [(5, 300), (7, 400), (1, 400)]
    [(2, 250), (0, 400)],
    [(3, 300), (1, 250)],
    [(4, 100), (2, 300)],
    [(8, 150), (7, 200), (3, 100)],
    [(7, 250), (6, 130), (0, 300)],
    [(7, 200), (5, 130)],
    [(4, 200), (5, 250), (6, 200), (0, 400)],
    [(4, 150)]]
```



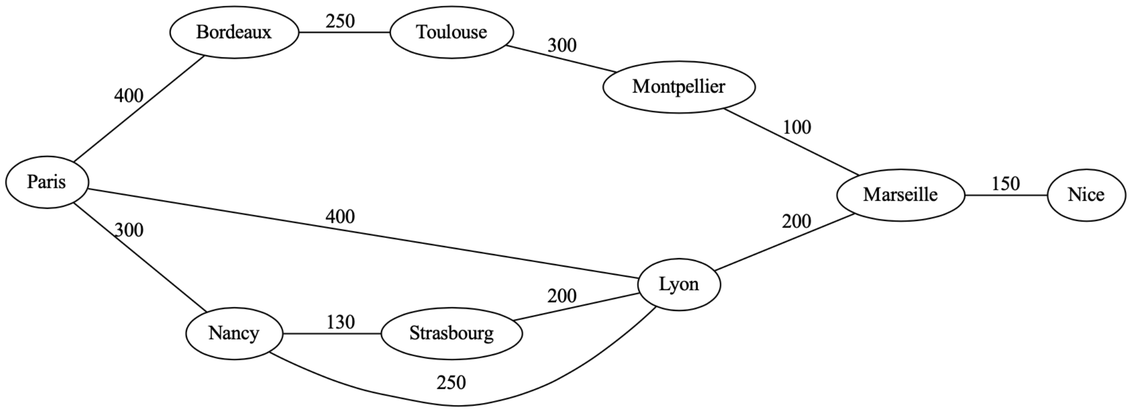
- Représentation compacte de la matrice de connexion

Graphes (représentation 2)

- Représentation par tableau de sommets et listes d'adjacence

```
class Sommet :
    def __init__ (self, s, l) :
        self.nom = s
        self.voisins = l


graphe = [
     Sommet ('Paris', [(5, 300), (7, 400), (1, 400)]),
     Sommet ('Bordeaux', [(2, 250), (0, 400)]),
     Sommet ('Toulouse', [(3, 300), (1, 250)]),
     Sommet ('Montpellier', [(4, 100), (2, 300)]),
     Sommet ('Marseille', [(8, 150), (7, 200), (3, 100)]),
     Sommet ('Nancy', [(7, 250), (6, 130), (0, 300)]),
     Sommet ('Strasbourg', [(7, 200), (5, 130)]),
     Sommet ('Lyon', [(4, 200), (5, 250), (6, 200), (0, 400)]),
     Sommet ('Nice', [(4, 150)])
     ]
```



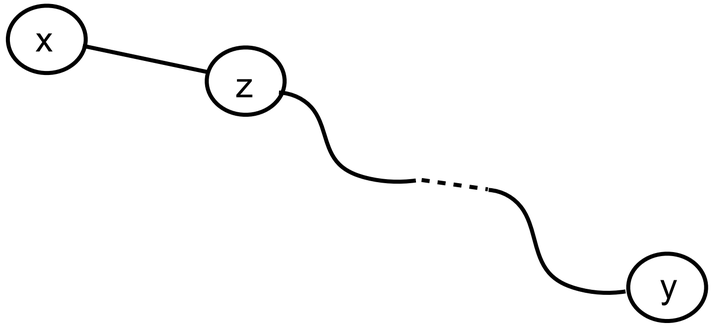
Graphes

- cakculer un chemin possible pour aller d'une ville à une autre

```
def chemin (g, x, y, dejaVu) :
    dejaVu [x] = True
    if x == y :
        return [x]
    for p in g[x].voisins :
        z = p[0] #d = p[1]
        if not dejaVu [z] :
            ch = chemin (g, z, y, dejaVu)
            if ch != [] :
               return [z] + ch
    return []
```

```
def uneSolution (g, x, y) :
    n = len (g)
    dejaVu = n*[False]
    ch = chemin (g, x, y, dejaVu)
    if ch != [] :
        return ([x] + ch)[:-1]
    return []
```

- même programme que pour sortie de labyrinthe (cours 10)



à faire

- graphes dirigés
- analyses lexicale et syntaxique
- modularité et programmation objet
- programmation graphique
- algorithmes géométriques
- calculs flottants et méthodes numériques
- programmation de plusieurs fils de calcul
- assertions et logique des programmes
- introduction à l'informatique théorique
- etc

vive l’informatique

et

la programmation !