Lysa Derbah – Hayette Moulhabass

Algorithmique de Graphes

le projet : Tables de routage

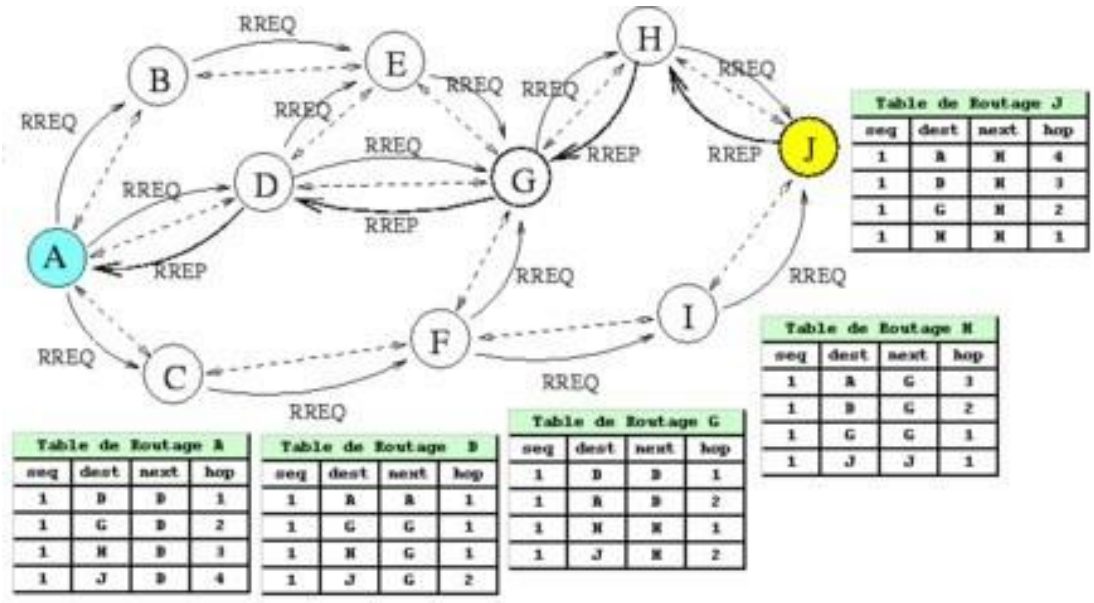


Table de Routage J

seq	dest	next	hop
1	A	H	4
1	D	H	3
1	G	H	2
1	H	H	1

Table de Routage H

seq	dest	next	hop
1	A	G	3
1	D	G	2
1	G	G	1
1	J	J	1

Table de Routage A

seq	dest	next	hop
1	D	D	1
1	G	D	2
1	H	D	3
1	J	D	4

Table de Routage D

seq	dest	next	hop
1	A	A	1
1	G	G	1
1	H	G	1
1	J	G	2

Table de Routage G

seq	dest	next	hop
1	D	D	1
1	A	D	2
1	H	H	1
1	J	H	2

Présentation :

Pour réaliser notre projet d'Algorithmique de Graphes, qui consiste en la réalisation de tables de routage, nous avons décidés, mon binôme et moi, de programmer ce dernier en langage Python, car c'est un langage facile, et qu'on a déjà vu. Pour faire cela, nous avons créer un Notebook partagé sur Collaboratory, pour pouvoir travailler chacune de chez elle, mais sur un même fichier.
Nous avons donc découpés notre programme en plusieurs fonctions, pour qu'on puisse travailler chacune de son coté. On s 'appelle deux fois par semaine pour faire le point, modifier, et améliorer ce qui ne vas pas.

Pour cela, on distingue donc bien 3 réseaux :

- ➔ Un réseau backbone (T1) de 10 nœuds
- ➔ Un réseau de transit (T2) de 20 nœuds
- ➔ Un réseau local (T3) de 70 nœuds

Les graphes des réseaux ne sont pas orientés et sont simples (un seul arc entre deux nœuds).

Les nœuds :

Les réseaux sont définis par leurs nœuds, qui sont des instances de la classe Nœud, et pour un nœud ont a les attributs suivants :

- **nom** nom : Chaque nœud a un identifiant qui est un entier. Si il y a 100 nœuds en tout, les noms vont de 1 à 100.
- **voisins** voisins : Chaque nœud a un certain nombre de *voisins*. Les arcs sont représentés par les voisins des nœuds. Un voisin est une liste [nœud_voisin, valeur].
- **tier** tier : réseau auquel appartient le nœud. "T1", "T2" ou "T3".
- **marque** marque : Cette zone est utilisée pour différents usages.

    1) Pour marquer un nœud comme étant visité (True ou False), dans l'algorithme de contrôle de connectivité.

    2) Pour recueillir la longueur d'un plus court chemin à ce nœud dans l'algorithme de recherche des plus courts chemins.
- **prédécesseur** prédécesseur : nœud prédécesseur qui a permis d'atteindre le nœud dans l'algorithme de plus court chemin.
- **table-de-routage** table−de−routage : dictionnaire clé-valeur indiquant vers quel nœud voisin se diriger pour atteindre un nœud cible. La clé est le nœud cible, la valeur est le nœud voisin. Exemple : Table de routage du nœud 5 {"1" : "3", "2": "6" ...}. A partir du nœud 5 : pour atteindre le nœud 1, aller au voisin 3, pour atteindre le nœud 2, aller au voisin 6, etc.
- **coord** coord : coordonnées du nœud pour faire un schéma. Ces cordonnées sont choisies aléatoirement.

Réseau backbone (T1) :

Il est constitué de 10 nœuds très connectés entre eux. Pour chaque lien possible entre deux nœuds, il y a 75 % de chances qu'il existe. Et les arc du réseau T1 ont une valeur tirée aléatoirement entre 5 et 10.

Réseau de transit (T2) :

Il est constitués de 20 nœuds, chacun connecté à 1 ou 2 nœuds du backbone tirés aléatoirement, et chacun connecté à 2 ou 3 nœuds du niveau 2 tirés également aléatoirement. Chacun de ces liens est évalué par une valeur comprise entre 10 et 20.
Pour sélectionner n nœuds dans une liste N de nœuds, on utilise la fonction sample(N,n) en Python, et pour choisir aléatoirement 2 ou 3 arcs on utilise la fonction choice([2,3]).

Réseau local (Tier 3) :

Il est constituer de 70 nœuds, reliés à 2 nœuds du T2, et reliés 1 nœud du T3. (Ce qui implique que le nombre de nœuds du T3 soit pair). Chacun de ces liens est évalué par une valeur comprise entre 15 et 20. En Python pour faire une boucle avançant d'un pas de 2, on fait: for i in range(0,69,2):
print("Arc" : i,i+1)

Vérification de la connectivité du réseau :

On fait une recherche en "profondeur d'abord" pour vérifier que tous le nœuds peuvent être atteints d'un nœud de départ quelconque. Les nœuds disposent d'une marque, True (marqués) ou False (non marqués).
On utilise une fonction récursive suivante :

- on marque le nœud

- on se positionne sur son premier voisin non marqué.

L'algorithme s'arrête quand on est placé sur un nœud dont tous les voisins sont déjà marqués.

Avec cet algorithme, on vérifie qu'à partir d'un nœud donné N0 on peut atteindre tous les autres nœuds.
Si cela est le cas, on peut en déduire que le réseau est connecté, c'est-à-dire qu'il existe au moins un chemin de n'importe quel nœud N1 et n'importe quel nœud N2. Ce chemin est : $N1-N0-N2$

Détermination de la table de routage de chaque nœud :

A partir d'un nœud de départ Nd, on doit indiquer pour chaque destination possible (les 99 autres nœuds) à quel voisin il convient de router un paquet compte tenu de la destination finale.
Pour cela on calcule les plus courts chemins de No à tous les autres nœuds avec l'algorithme de Dijkstra.
Toutes les valeurs des arêtes sont positives.
On utilise les marques de nœud déjà utilisées.
Ici la *marque* d'un nœud N représente la plus courte distance calculée $Nd-N$.
On initialise toutes les marques des nœuds à l'infini, sauf la marque de Nd qui est initialisée à 0.
Chaque nœud a un champ *prédécesseur* qui indique le nœud prédécesseur qui a permis de lui attribuer sa marque.
A chaque étape, on choisit le nœud de marque la plus faible parmi les nœuds à visiter.
Pour chaque voisin du nœud choisi, on met à jour sa marque si celle-ci est supérieure à la marque du nœud choisi plus la valeur de l'arc menant du nœud choisi à son voisin.
Ceci fait, on retire le nœud choisi de la liste des nœuds à visiter.
Les étapes s'arrêtent lorsqu'il n'y a plus de nœuds à visiter.
Lorsque l'algorithme est fini, chaque nœud comporte un champ *prédécesseur* qui indique le nœud qui permet de l'atteindre sur le plus court chemin entre Nd et lui.

table_de_routage est un attribut d'instance de Nœud qui prend la forme d'un dictionnaire Python.
Ce dictionnaire comprend des couples clé-valeur, où la clé est le nœud cible et la valeur le nœud voisin à emprunter.

Le processus est le suivant :
Pour chacun des nœuds N du graphe :

- on calcule les plus courts de N vers tous les autres nœuds.
- on met à jour les tables de routage pour tous les plus courts chemins

Reconstitution d'un chemin entre deux nœuds :

Reconstitution d'un chemin deux nœuds en utilisant les tables de routage.

code source :

```
In [ ]: from random import * # pour faire les tirages aléatoires
        from math import *
        import matplotlib.pyplot as plt # pour dessiner le réseau
```

On commence par inclure les bibliothèques, dont on aura besoin pour notre projet.

```
In [ ]: class Noeud:
          """Chaque noeud a un certain nombre de voisins.
             Un voisin est un liste [noeud, valeur].
             Chaque noeud a un identifiant nom qui est un entier."""
          # méthode de construction d'une instance de noeud
          def __init__(self, nom, réseau):
            self.nom=nom
            self.tier=réseau
            self.voisins=[]
            self.marque = False # marque du noeud initialisée à False
            self.prédécesseur = self
            self.table_routage={} # Table de routage. Dictionnaire
            # calcul d'un couple de coordonnées aléatoires
            # La coordonnée est x est choisie entre 0 et 100
            # la coordonnée y est choisie en fonction du réseau d'appartenance
            y = None
            if réseau == "T1":
              y=uniform(90,100)
            elif réseau== "T2":
              y=uniform(70,90)
            else:
              y=uniform(0,70)
            self.coord=[uniform(0,100),y]

          # méthode permettant d'imprimer un noeud
          def __str__(self):
            return "{}-{}".format(self.tier,self.nom)
```

Dans cette fonction on défini la méthode de construction d'une instance du nœud, et et la méthode pour l'imprime par la suite.

```
In [ ]:
        nb_noeuds_T1 = 10
        nb_noeuds_T2 = 20

        nb_noeuds_T3 = 70
```

ici, on choisit le nombre de nœuds de chacun des réseaux.

In [ ]:
```python
# Fonctions d'affichage et de dessin
# Ces fonctions sont aussi utilisées pour le débuggage

def affiche_noeud(n, voisins=False, marque=False, pred=False, table=False):
  print("Noeud {}-{}".format(n.tier, n.nom))
  if voisins:
    print("  Voisins : {}".format([x[0].nom for x in n.voisins]))
  if marque:
    print("  Marque : {:.0f}".format(n.marque))
  if pred:
    print("  Prédécésseur : {}".format(n.prédécesseur))
  if table:
    affiche_table(n.table_routage)

# affichage d'une table de routage
def affiche_table(table):
  for clé, valeur in table.items():
    print("   Pour aller à {} : {}".format(clé, valeur))

def affichage_réseau(réseau, voisins=False, marque=False, pred=False, table=False):
  for n in réseau:
    affiche_noeud(n, voisins, marque, pred, table)

def couleur_noeud(n):
  if n.tier=="T1":
    couleur = 'r'
  elif n.tier=="T2":
    couleur = 'b'
  else:
    couleur ='y'
  return couleur

def couleur_arc(n1, n2):
  couleur='silver'
  if n1.tier=="T1" and n2.tier=="T1":
    couleur = 'r'
  if n1.tier=="T2" and n2.tier=="T2":
    couleur = 'b'
  if n1.tier=="T3" and n2.tier=="T3":
    couleur = 'y'
  return couleur


def dessine_réseau(graphe):
  for n in graphe:
    plt.scatter(n.coord[0], n.coord[1], color=couleur_noeud(n), s=30)
    for voisin in n.voisins:
      if voisin[0] in graphe:
        if voisin[0].nom > n.nom:
          plt.plot([n.coord[0], voisin[0].coord[0]], [n.coord[1], voisin[0].coord[1]], couleur_arc(n, voisin[0]))
  plt.show()
```

ces fonctions permettent d'afficher les nœuds, tables de routages, réseau, et aussi de dessiner les graphes en choisissant la couleur des nœuds, et des arcs.

cette fonction permet de créer le réseau T1, qui est le réseau backbone, de l'afficher, et de dessiner le graphe qui lui correspond. Quand on compile on obtiens de résultat suivant :

Création du réseau T1 (backbone) : 10 nœuds

Nœud T1-1

```
réseau_T1= []

def création_réseau_T1():
  for i in range (1, nb_noeuds_T1 +1):
    réseau_T1.append(Noeud(i, "T1"))
  print("Création du réseau T1 (backbone) : {} noeuds".format(len(réseau_T1)))

création_réseau_T1()
affichage_réseau(réseau_T1)
dessine_réseau(réseau_T1)
```

Nœud T1-2

Nœud T1-3

Nœud T1-4

Nœud T1-5

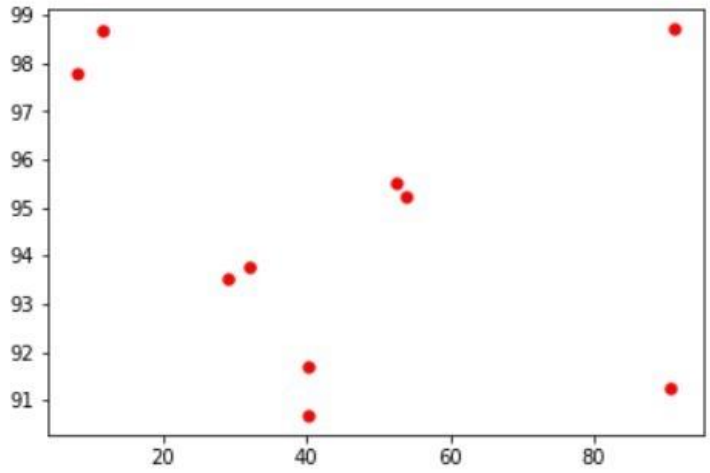
Nœud T1-6

Nœud T1-7

Nœud T1-8

Nœud T1-9

Nœud T1-10



In [ ]:
```
# Maillage du réseau T1 intra-backbone

# on initialise à vide les voisins
for n in réseau_T1:
  n.voisins=[]
```

```
proba_arc_interne_T1 = 0.45

# Effet : le maillage met à jour l'attribut voisins de chaque noeud
def maillage_intra_T1():
  print("Maillage de chaque noeud T1 vers {:.0%} des autres noeuds T1.".format(proba_arc_interne_T1))
  nb_arcs=0
  for n1 in réseau_T1:
    for n2 in réseau_T1:
        if n2.nom > n1.nom:
          # un arc est crée dans 75 % des cas
          if random() < proba_arc_interne_T1:
            valeur=uniform(5 , 10)
            # on crée un voisin pour n1 et pour n2 (arc dans les deux sens)
            n1.voisins.append([n2, valeur])
            n2.voisins.append([n1, valeur])
            nb_arcs=nb_arcs+1
  print("Nombre d'arcs intra-T1 créés : {}".format(nb_arcs))

maillage_intra_T1()
affichage_réseau(réseau_T1, voisins=True)
dessine_réseau(réseau_T1)
```

cette fonction permet de faire le maillage du réseau T1. D'abord on initialise à vide les voisins, puis on met la valeur de probabilité d'occurrence d'un arc interne au réseau T1 à 0,45 pour obtenir un résultat réaliste (car dans l'énoncer cette valeur est de 0,75). Ensuite, on fait le maillage du réseau T1, ce dernier met a jour l'attribut voisins de chaque nœud. Enfin, on affiche et on dessine le réseau. Quand on compile on obtiens le résultat suivant :

Maillage de chaque nœud T1 vers 45% des autres nœuds T1.
Nombre d'arcs intra-T1 créés : 17
Noeud T1-1
Voisins : [6, 9]
Noeud T1-2
Voisins : [3, 6, 9, 10]
Noeud T1-3
Voisins : [2, 6, 9]
Nœud T1-4
Voisins : [6, 8, 9]
Nœud T1-5
Voisins : [6, 7, 8]
Nœud T1-6
Voisins : [1, 2, 3, 4, 5, 10]
Nœud T1-7

Voisins : [5, 8]

Noeud T1-8

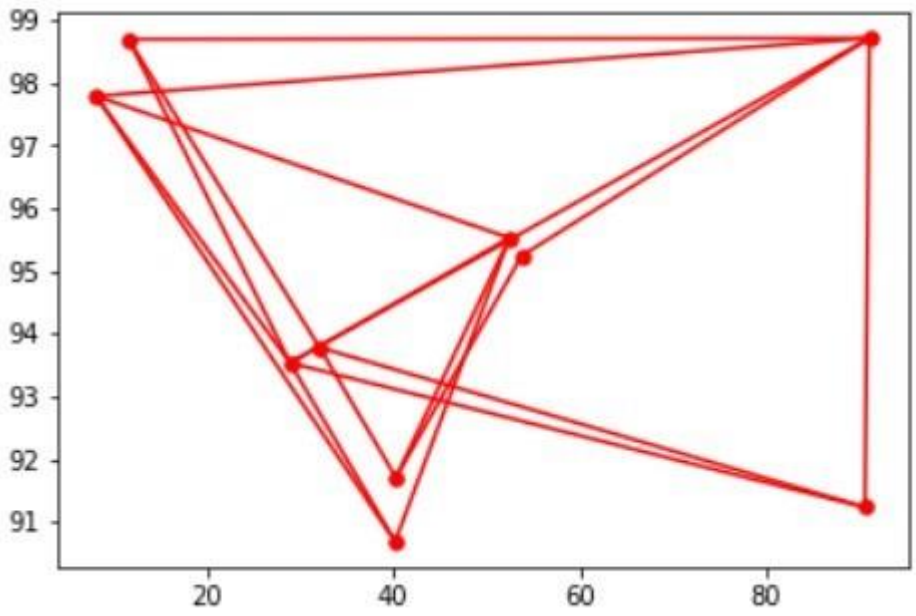
Voisins : [4, 5, 7, 9]

Noeud T1-9

Voisins : [1, 2, 3, 4, 8]

Noeud T1-10

Voisins : [2, 6]



cette fonction permet de créer le réseau T2, qui est le réseau transit, de l'afficher, et de dessiner le graphe qui lui correspond. Quand on compile on obtiens de résultat suivant :

Création du réseau T2 (transit) : 20 nœuds

Noeud T2-11

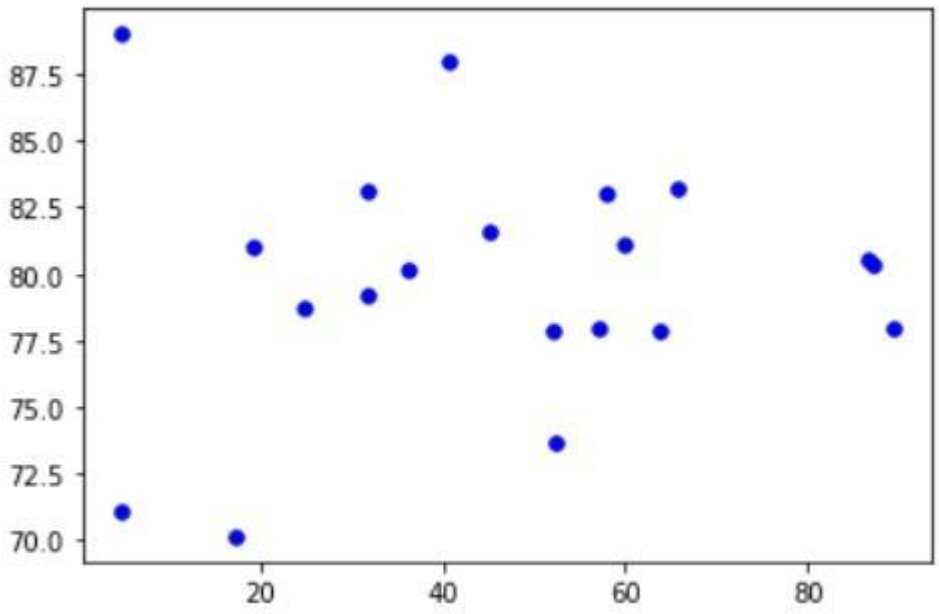
Noeud T2-12

Noeud T2-13

Noeud T2-14
Noeud T2-15
Noeud T2-16
Noeud T2-17
Noeud T2-18
Noeud T2-19

In [ ]:
```
réseau_T2 = []


def création_réseau_T2():
  # on crée les noeuds de T2 en leur donnant des noms entiers à la suite de ceux du T1.
  # Typiquement, si on crée 20 noeuds, les noms vont de 11 à 30.
  for i in range (len(réseau_T1)+1, len(réseau_T1)+nb_noeuds_T2+1):
    réseau_T2.append(Noeud(i,"T2"))
  print("Création du réseau T2 (transit) : {} noeuds".format(len(réseau_T2)))

création_réseau_T2()
affichage_réseau(réseau_T2)
dessine_réseau(réseau_T2)
```

Noeud T2-20
Noeud T2-21
Noeud T2-22
Noeud T2-23
Noeud T2-24
Noeud T2-25
Noeud T2-26
Noeud T2-27
Noeud T2-28
Noeud T2-29
Noeud T2-30


87.5
85.0
82.5
80.0
77.5
75.0
72.5
70.0
20
40
60
80

In [ ]:
```
# Maillage du réseau de transit

def maillage_T2():

  # on initialise à vide les voisins
  for n in réseau_T2:
    n.voisins=[]

  print("Maillage de chaque noeud T2 vers un ou deux noeuds T1.")
  nb_arcs_T2_T1 = 0
  for n2 in réseau_T2:
    # on sélectionne aléatoirement un ou deux noeuds du T1
    noeuds_T1_choisis=sample(réseau_T1, choice([1, 2]))
    for n1 in noeuds_T1_choisis:
      valeur=uniform(10,20)
      n2.voisins.append([n1, valeur])
      n1.voisins.append([n2, valeur])
      nb_arcs_T2_T1 = nb_arcs_T2_T1 + 1
  print("Nombre d'arcs T2-T1 créés : {}".format(nb_arcs_T2_T1))

  print("Maillage de chaque noeud T2 à 2 ou 3 autres noeuds de T2.")
  nb_arcs_T2_T2 = 0
  for n2 in réseau_T2:
    # on selectionne aléatoirement 2 ou 3 autres noeuds du réseau T2
    # On crée une copie des noeuds du réseau de transit
    L=réseau_T2.copy()
    # on enlève le noeud n2 de cette copie
    L.remove(n2)
    noeuds_T2_choisis=sample(L, choice([2, 3]))
    for nc in noeuds_T2_choisis:
      if nc.nom > n2.nom:
        # On crée un arc entre le noeud n2 et nc après avoir calculé sa valeur :
        valeur=uniform(10,20)
        n2.voisins.append([nc, valeur])
        nc.voisins.append([n2, valeur])
        nb_arcs_T2_T2 = nb_arcs_T2_T2 +1
  print("Nombre d'arcs T2-T2 créés : {}".format(nb_arcs_T2_T2))

maillage_T2()
affichage_réseau(réseau_T1, voisins=True)
affichage_réseau(réseau_T2, voisins=True)
dessine_réseau(réseau_T1+réseau_T2)
```

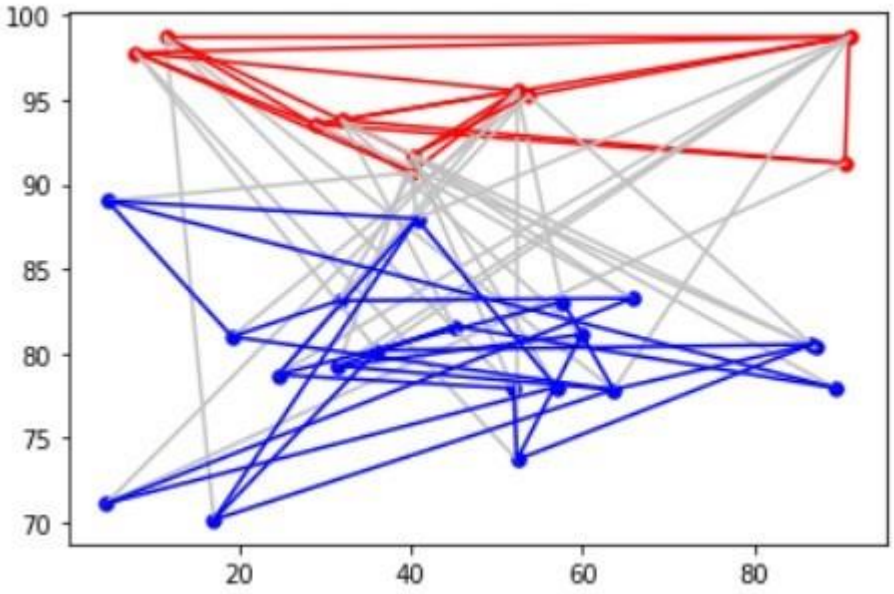
cette fonction permet de faire le maillage du réseau T2. D'abord on initialise à vide les voisins, on sélection un ou deux nœuds du réseau T1, puis on sélection aléatoirement deux ou trois autres nœuds du réseau T2, on crée une copie des nœuds du réseau T2, on lui enlève le nœud n2, puis on crée un arc entre le nœud n2 et nc après avoir calculer sa valeur. Enfin, on affiche et on dessine le réseau. Quand on compile on obtiens le résultat suivant :

Maillage de chaque nœud T2 vers un ou deux nœuds T1.

Nombre d'arcs T2-T1 créés : 31
Maillage de chaque nœud T2 à 2 ou 3 autres nœuds de T2.
Nombre d'arcs T2-T2 créés : 26

Noeud T1-1
  Voisins : [2, 4, 8, 16, 23, 28]
Noeud T1-2
  Voisins : [1, 3, 7, 8, 9, 10, 30]
Noeud T1-3
  Voisins : [2, 6, 9, 10, 15, 20, 25, 26]
Noeud T1-4
  Voisins : [1, 6, 7, 14, 19, 30]
Noeud T1-5
  Voisins : [6, 8, 13, 18, 27]
Noeud T1-6
  Voisins : [3, 4, 5, 11, 17, 21, 24, 28]
Noeud T1-7
  Voisins : [2, 4, 8, 25]
Noeud T1-8
  Voisins : [1, 2, 5, 7, 10, 12, 17, 22, 23, 26]
Noeud T1-9
  Voisins : [2, 3, 10, 18, 21, 22, 29]
Noeud T1-10
  Voisins : [2, 3, 8, 9, 12, 20]
Noeud T2-11
  Voisins : [6, 29, 19]
Noeud T2-12
  Voisins : [10, 8, 13, 19, 16]
Noeud T2-13
  Voisins : [5, 12, 23, 20]
Noeud T2-14
  Voisins : [4, 17, 20]
Noeud T2-15

Voisins : [3, 26, 23]
Noeud T2-16
Voisins : [1, 12, 23, 22]
Noeud T2-17
Voisins : [6, 8, 14, 23, 19]
Noeud T2-18
Voisins : [5, 9, 27, 21]
Noeud T2-19
Voisins : [4, 11, 12, 17]
Noeud T2-20
Voisins : [3, 10, 13, 14, 28]
Noeud T2-21
Voisins : [9, 6, 18, 25]
Noeud T2-22
Voisins : [9, 8, 16, 29, 26, 24]
Noeud T2-23
Voisins : [1, 8, 13, 15, 16, 17, 27]
Noeud T2-24
Voisins : [6, 22, 25]
Noeud T2-25
Voisins : [3, 7, 21, 24]
Noeud T2-26
Voisins : [8, 3, 15, 22, 28]
Noeud T2-27
Voisins : [5, 18, 23, 29]
Noeud T2-28
Voisins : [6, 1, 20, 26]
Noeud T2-29
Voisins : [9, 11, 22, 27]
Noeud T2-30
Voisins : [4, 2]



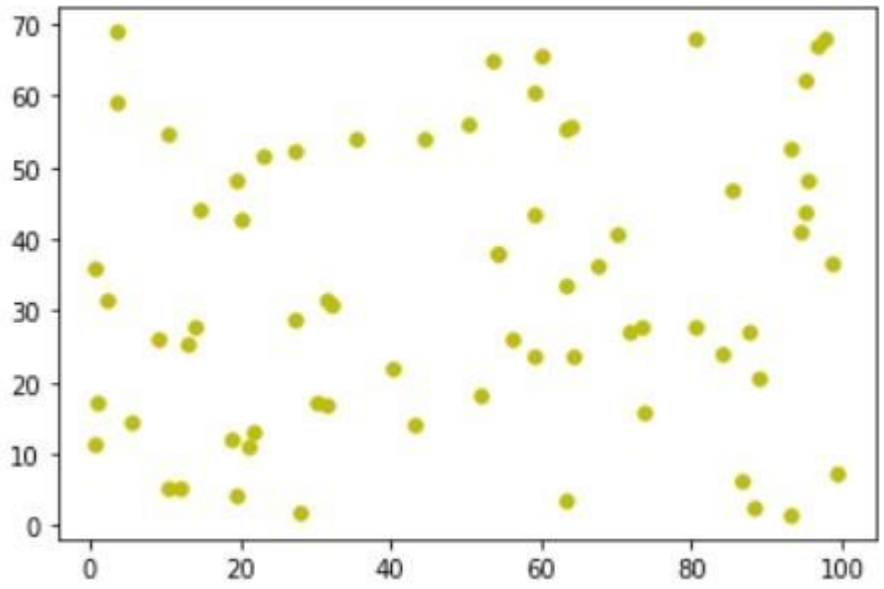

```
In [ ]: réseau_T3 = []

def création_réseau_T3():
  for i in range (len(réseau_T1)+len(réseau_T2)+1, len(réseau_T1)+len(réseau_T2)+nb_noeuds_T3+1):
    réseau_T3.append(Noeud(i,"T3"))
  print("Création du réseau T3 (local) : {} noeuds".format(len(réseau_T3)))

création_réseau_T3()
affichage_réseau(réseau_T3)
dessine_réseau(réseau_T3)
```

Cette fonction permet de créer le réseau T3, qui est le réseau local, de l'afficher, et de dessiner le graphe qui lui correspond. Quand on compile on obtiens de résultat suivant :

Création du réseau T3 (local) : 70 noeuds
Noeud T3-31
Noeud T3-32
Noeud T3-33
Noeud T3-34
Noeud T3-35
Noeud T3-36
Noeud T3-37
Noeud T3-38
Noeud T3-39
Noeud T3-40

Noeud T3-41
Noeud T3-42
Noeud T3-43
Noeud T3-44
Noeud T3-45
Noeud T3-46
Noeud T3-47
Noeud T3-48
Noeud T3-49
Noeud T3-50
Noeud T3-51
Noeud T3-52
Noeud T3-53
Noeud T3-54
Noeud T3-55
Noeud T3-56
Noeud T3-57
Noeud T3-58
Noeud T3-59
Noeud T3-60
Noeud T3-61
Noeud T3-62
Noeud T3-63
Noeud T3-64
Noeud T3-65
Noeud T3-66
Noeud T3-67
Noeud T3-68
Noeud T3-69
Noeud T3-70
Noeud T3-71 Noeud T3-72
Noeud T3-73
Noeud T3-74

Noeud T3-75
Noeud T3-76
Noeud T3-77
Noeud T3-78
Noeud T3-79
Noeud T3-80
Noeud T3-81
Noeud T3-82
Noeud T3-83
Noeud T3-84
Noeud T3-85
Noeud T3-86
Noeud T3-87
Noeud T3-88
Noeud T3-89
Noeud T3-90
Noeud T3-91
Noeud T3-92
Noeud T3-93
Noeud T3-94
Noeud T3-95
Noeud T3-96
Noeud T3-97
Noeud T3-98
Noeud T3-99
Noeud T3-100


70
60
50
40
30
20
10
0
0
20
40
60
80
100

```
In [ ]: # maillage du réseau T3
        # Chaque noeud du T3 est relié à 2 noeuds du T2
        # Chaque noeud du T3 est relié à 1 autre noeud du T3
        # Les liens sont valués par une valeur comprise entre 15 et 20

        def maillage_réseau_T3():
          print("Maillage de chaque noeud T3 vers 2 noeuds du T2.")
          nb_noeuds_T3_T2 = 0
          for n3 in réseau_T3:
            # on sélectionne aléatoirement 2 noeuds du T2
            noeuds_T2_choisis=sample(réseau_T2, 2)
            for n2 in noeuds_T2_choisis:
              valeur=uniform(15, 20)
              n3.voisins.append([n2, valeur])
              n2.voisins.append([n3, valeur])
              nb_noeuds_T3_T2 = nb_noeuds_T3_T2 +1
          print("Nombre d'arcs T3-T2 : {}".format(nb_noeuds_T3_T2))

          print("Maillage de chaque noeud T3 vers 1 autre noeud T3.")
          nb_noeuds_T3_T3 = 0
          #Pour relier les noeuds du T3 2 à 2,
          # On fait une boucle avançant d'un pas de 2
          for i in range (0, nb_noeuds_T3-1, 2):
            # création d'un arc reliant 2 noeuds du T3
            valeur=uniform(15, 20)
            réseau_T3[i].voisins.append([réseau_T3[i+1], valeur])
            réseau_T3[i+1].voisins.append([réseau_T3[i], valeur])
            nb_noeuds_T3_T3 = nb_noeuds_T3_T3 +1
          print("Nombre d'arcs T3-T3 : {}".format(nb_noeuds_T3_T3))

        maillage_réseau_T3()
        affichage_réseau(réseau_T2, voisins=True)
        affichage_réseau(réseau_T3, voisins=True)
        dessine_réseau(réseau_T1+réseau_T2+réseau_T3)
```

cette fonction permet de faire le maillage du réseau T3. D'abord on sélection aléatoirement deux nœuds du réseau T2, puis pour relier les nœuds du T3 deux à deux, on fait une boucle avançant d'un pas de 2, ensuite, on crée un arc reliant 2 nœuds du T3. Enfin, on affiche et on dessine le réseau. Quand on compile on obtiens le résultat suivant :

Maillage de chaque noeud T3 vers 2 noeuds du T2.

Nombre d'arcs T3-T2 : 140

Maillage de chaque noeud T3 vers 1 autre noeud T3.

Nombre d'arcs T3-T3 : 35

Noeud T2-11

Voisins : [6, 29, 19, 36, 52, 53, 55, 62, 73, 79, 96]
Noeud T2-12
Voisins : [10, 8, 13, 19, 16, 31, 59]
Noeud T2-13
Voisins : [5, 12, 23, 20, 33, 40, 48, 51, 70, 80, 84, 94, 98]
Noeud T2-14
Voisins : [4, 17, 20, 37, 42, 52, 54, 60, 61, 76, 91]
Noeud T2-15
Voisins : [3, 26, 23, 44, 49, 65, 66, 70, 76, 83, 90, 94, 95]
Noeud T2-16
Voisins : [1, 12, 23, 22, 35, 46, 50, 67, 83]
Noeud T2-17
Voisins : [6, 8, 14, 23, 19, 35, 46, 79, 82, 87, 88, 99, 100]
Noeud T2-18
Voisins : [5, 9, 27, 21, 48, 53, 56, 72, 74, 77, 84, 89]
Noeud T2-19
Voisins : [4, 11, 12, 17, 33, 41, 42, 47, 50, 57, 68, 71, 97, 98]
Noeud T2-20
Voisins : [3, 10, 13, 14, 28, 37, 43, 54, 56, 69, 75, 82, 93]
Noeud T2-21
Voisins : [9, 6, 18, 25, 34, 44, 64, 69, 71, 86]
Noeud T2-22
Voisins : [9, 8, 16, 29, 26, 24, 32, 34, 87, 90, 93]
Noeud T2-23
Voisins : [1, 8, 13, 15, 16, 17, 27, 32, 38, 45, 68, 72, 74]
Noeud T2-24
Voisins : [6, 22, 25, 39, 41, 63, 75, 99]
Noeud T2-25
Voisins : [3, 7, 21, 24, 47, 55, 63, 77, 80, 81, 96]
Noeud T2-26
Voisins : [8, 3, 15, 22, 28, 31, 36, 43, 57, 62, 67, 95]
Noeud T2-27
Voisins : [5, 18, 23, 29, 38, 49, 66, 73, 78, 81, 85, 97]

Noeud T2-28
  Voisins : [6, 1, 20, 26, 39, 58, 59, 60, 64, 65, 78, 88, 92]
Noeud T2-29
  Voisins : [9, 11, 22, 27, 40, 51, 58, 61, 85, 91, 100]
Noeud T2-30
  Voisins : [4, 2, 45, 86, 89, 92]
Noeud T3-31
  Voisins : [26, 12, 32]
Noeud T3-32
  Voisins : [23, 22, 31]
Noeud T3-33
  Voisins : [13, 19, 34]
Noeud T3-34
  Voisins : [22, 21, 33]
Noeud T3-35
  Voisins : [17, 16, 36]
Noeud T3-36
  Voisins : [26, 11, 35]
Noeud T3-37
  Voisins : [14, 20, 38]
Noeud T3-38
  Voisins : [23, 27, 37]
Noeud T3-39
  Voisins : [28, 24, 40]
Noeud T3-40
  Voisins : [29, 13, 39]
Noeud T3-41
  Voisins : [19, 24, 42]
Noeud T3-42
  Voisins : [14, 19, 41]
Noeud T3-43
  Voisins : [20, 26, 44]
Noeud T3-44

Voisins : [15, 21, 43]
Noeud T3-45
Voisins : [30, 23, 46]
Noeud T3-46
Voisins : [17, 16, 45]
Noeud T3-47
Voisins : [19, 25, 48]
Noeud T3-48
Voisins : [13, 18, 47]
Noeud T3-49
Voisins : [27, 15, 50]
Noeud T3-50
Voisins : [16, 19, 49]
Noeud T3-51
Voisins : [29, 13, 52]
Noeud T3-52
Voisins : [14, 11, 51]
Noeud T3-53
Voisins : [11, 18, 54]
Noeud T3-54
Voisins : [14, 20, 53]
Noeud T3-55
Voisins : [25, 11, 56]
Noeud T3-56
Voisins : [18, 20, 55]
Noeud T3-57
Voisins : [26, 19, 58]
Noeud T3-58
Voisins : [28, 29, 57]
Noeud T3-59
Voisins : [12, 28, 60]
Noeud T3-60

Voisins : [28, 14, 59]
Noeud T3-61
Voisins : [29, 14, 62]
Noeud T3-62
Voisins : [26, 11, 61]
Noeud T3-63
Voisins : [25, 24, 64]
Noeud T3-64
Voisins : [28, 21, 63]
Noeud T3-65
Voisins : [15, 28, 66]
Noeud T3-66
Voisins : [15, 27, 65]
Noeud T3-67
Voisins : [16, 26, 68]
Noeud T3-68
Voisins : [23, 19, 67]
Noeud T3-69
Voisins : [20, 21, 70]
Noeud T3-70
Voisins : [15, 13, 69]
Noeud T3-71
Voisins : [19, 21, 72]
Noeud T3-72
Voisins : [23, 18, 71]
Noeud T3-73
Voisins : [11, 27, 74]
Noeud T3-74
Voisins : [18, 23, 73]
Noeud T3-75
Voisins : [24, 20, 76]
Noeud T3-76
Voisins : [14, 15, 75] Noeud T3-77

Voisins : [18, 25, 78]
Noeud T3-78
Voisins : [27, 28, 77]
Noeud T3-79
Voisins : [17, 11, 80]
Noeud T3-80
Voisins : [25, 13, 79]
Noeud T3-81
Voisins : [27, 25, 82]
Noeud T3-82
Voisins : [20, 17, 81]
Noeud T3-83
Voisins : [16, 15, 84]
Noeud T3-84
Voisins : [18, 13, 83]
Noeud T3-85
Voisins : [29, 27, 86]
Noeud T3-86
Voisins : [30, 21, 85]
Noeud T3-87
Voisins : [22, 17, 88]
Noeud T3-88
Voisins : [28, 17, 87]
Noeud T3-89
Voisins : [18, 30, 90]
Noeud T3-90
Voisins : [22, 15, 89]
Noeud T3-91
Voisins : [14, 29, 92]
Noeud T3-92
Voisins : [28, 30, 91]
Noeud T3-93

Voisins : [20, 22, 94]

Noeud T3-94

Voisins : [13, 15, 93]

Noeud T3-95

Voisins : [26, 15, 96]

Noeud T3-96

Voisins : [25, 11, 95]

Noeud T3-97

Voisins : [27, 19, 98]

Noeud T3-98

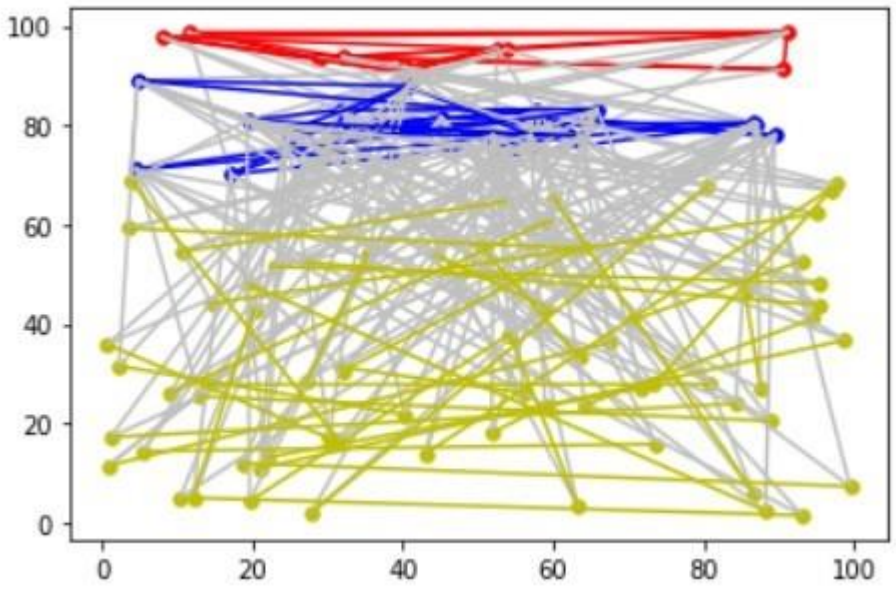
Voisins : [13, 19, 97]

Noeud T3-99

Voisins : [24, 17, 100]

Noeud T3-100

Voisins : [29, 17, 99]



In [ ]:
```python
# fonction appelée récursivement
# - on marque le noeud n
# - on recherche son premier voisin non marqué
# - on appelle la fonction avec ce vosin non marqué
def explorer_a_partir_du_noeud(n):
  # on marque le noeud n
  n.marque = True
  #print("Noeud exploré : {}".format(n))
  # On recherche le premier voisin du noeud non marqué
  for v in n.voisins:
    if v[0].marque == False:
      # on applique la même fonction à ce noeud
      explorer_a_partir_du_noeud(v[0])

def vérification_connectivité(graphe, nd):
  # graphe est une liste de noeuds
  # nd est le noeud de départ
  # On met toutes les marques à False
  for n in graphe:
    n.marque=False

  explorer_a_partir_du_noeud(nd)

  # On vérifie s'il reste des neoeuds non marqués
  nb_noeuds_isolés = 0
  for n in graphe:
    if n.marque == False:
      nb_noeuds_isolés = nb_noeuds_isolés +1
      print("Noeud isolé : {}".format(n))
  print("Nombre de noeuds isolés : {}".format(nb_noeuds_isolés))

vérification_connectivité(réseau_T1 + réseau_T2 + réseau_T3, réseau_T3[0])
#vérification_connectivité(réseau_T3, réseau_T3[0])
```

```
Nombre de noeuds isolés : 0
```

In [ ]:

ici, pour vérifier la connectivité du réseau, on va définir une fonction qui va êtres appelée récursivement, cette fonction permet donc de marqué le nœud n, chercher son premier voisin non marqué, et on applique la même fonction a ce nœud. Puis on crée une fonction pour vérifier la connectivité, qui a comme paramètres, une liste de nœuds (graphe), et le nœud de départ (nd), on met donc toutes les marques a false, et on applique la première fonction qui permet de faire une recherche en profondeur. Et enfin, on vérifie s'il reste des nœuds non marqué.

pour créer la table de routage de chaque nœud, on doit d'abord créer une fonction qui permet de calculer le plus courts chemins de nd à tous les autres nœuds, grâce à Dijkstra.

In [ ]:

```
# Entrées :
#    graphe est une liste des noeuds du graphe. ex : réseau_T1 + réseau_T2+ réseau_T3
#    nd est le noeud de départ
# Effet :
#    Calcule les plus courts chemins (PCCH) de nd à tous les autres noeuds.
#    Inscrit pour chaque noeud n:
#    marque : valeur du PCCH de nd à n.
#    prédecesseur : noeud qui permet d'atteindre n dans le PPCH
def plus_courts_chemins(nd, graphe):
  # On initialise toutes les marques des noeuds à l'infini (inf) sauf la marque de nd qui est initailisée à 0.
  for n in graphe:
    n.marque=inf
  nd.marque=0

  # liste des noeuds à visiter
  # Elle est initialisée par la copie des noeuds du graphe.
  noeuds_à_visiter = graphe.copy()

  # Tant qu'il reste des noeuds à visiter
  while len(noeuds_à_visiter)!=0:
    # On cherche le noeud à visiter ayant la marque minimum.
    # On utilise ici la fonction min avec le paramètre key indiquant avec une fonction lambda la valeur à tester.
    noeud_minimum = min(noeuds_à_visiter, key = lambda x: x.marque)
    #print("noeud minimum : ", noeud_minimum)

    # on explore les voisins du noeud minimum
    for voisin in noeud_minimum.voisins:
      if noeud_minimum.marque + voisin[1] < voisin[0].marque:
        voisin[0].marque = noeud_minimum.marque + voisin[1]
        voisin[0].prédécesseur = noeud_minimum
    # On retire le noeud_minimum de la liste des noeuds à visiter
    noeuds_à_visiter.remove(noeud_minimum)

#plus_courts_chemins(réseau_T1[0], réseau_T1 + réseau_T2+ réseau_T3)
#plus_courts_chemins(réseau_T1[0], réseau_T1)
#affichage_réseau(réseau_T1 + réseau_T2+ réseau_T3, voisins=True, marque=True, pred=True)
```

In [ ]:
```
# Entrées :
#    nd : noeud de départ
#    graphe : liste de noeuds de la classe Noeud.
# Contexte : on a auparavant calculé les plus courts chemins du noeud nd à tous les autres
# Fonction :
def fabrication_table_routage_pour_noeud_cible(nd, graphe):
  #print("Noeud nd : {}".format(nd))
  # on parcourt tous les noeuds n du graphe (sauf nd)
  for n in graphe:
    if n != nd:
      # on va remonter les prédécesseurs de n jusqu'à ce que l'on tombe sur nd
      nx = n
      pred=None
      while nx != nd:
        pred=nx.prédécesseur
        # on met à jour la table de routage de pred
        # La table dit : Quand on est sur pred, pour atteindre n, pendre le voisin nx
        pred.table_routage[n]=nx
        nx=pred

#fabrication_table_routage_pour_noeud_cible(réseau_T1[0], réseau_T1 + réseau_T2+ réseau_T3)


def fabrications_toutes_tables_routage(graphe):
  for n in graphe:
    n.table_routage = {}
  for n in graphe:
    plus_courts_chemins(n, graphe)
    fabrication_table_routage_pour_noeud_cible(n, graphe)

fabrications_toutes_tables_routage(réseau_T1 + réseau_T2+ réseau_T3)
```

puis on parcours tous les nœuds n du graphe sauf le nœud de départ, et on va remonter les prédécesseurs de n jusqu'à ce que l'on tombe sur nd, on met à jour la table de routage de pred.

In [ ]:
```
def affiche_chemin(ne, nd):
  # ne : nord émetteur
  print("Plus court chemin de {} à {}".format(ne, nd))
  n=ne
  plt.scatter(nd.coord[0], nd.coord[1], color=couleur_noeud(nd), s=30)
  while n != nd:
    print("{}-{}".format(n, n.table_routage[nd]))
    plt.plot([n.coord[0], n.table_routage[nd].coord[0]], [n.coord[1], n.table_routage[nd].coord[1]], couleur_arc(n, n.table_r
outage[nd]))
    plt.scatter(n.coord[0], n.coord[1], color=couleur_noeud(n), s=30)
    n=n.table_routage[nd]
plt.show()

affiche_chemin(choice(réseau_T1), choice(réseau_T3))
```

et pour construire un de chemin entre deux nœuds, on va utiliser les tables de routages. Quand on compile, on obtiens le résultat suivant :

Plus court chemin de T1-5 à T3-45
T1-5-T1-8

T1-8-T2-23

T2-23-T3-45