

THÉORIE DES GRAPHS

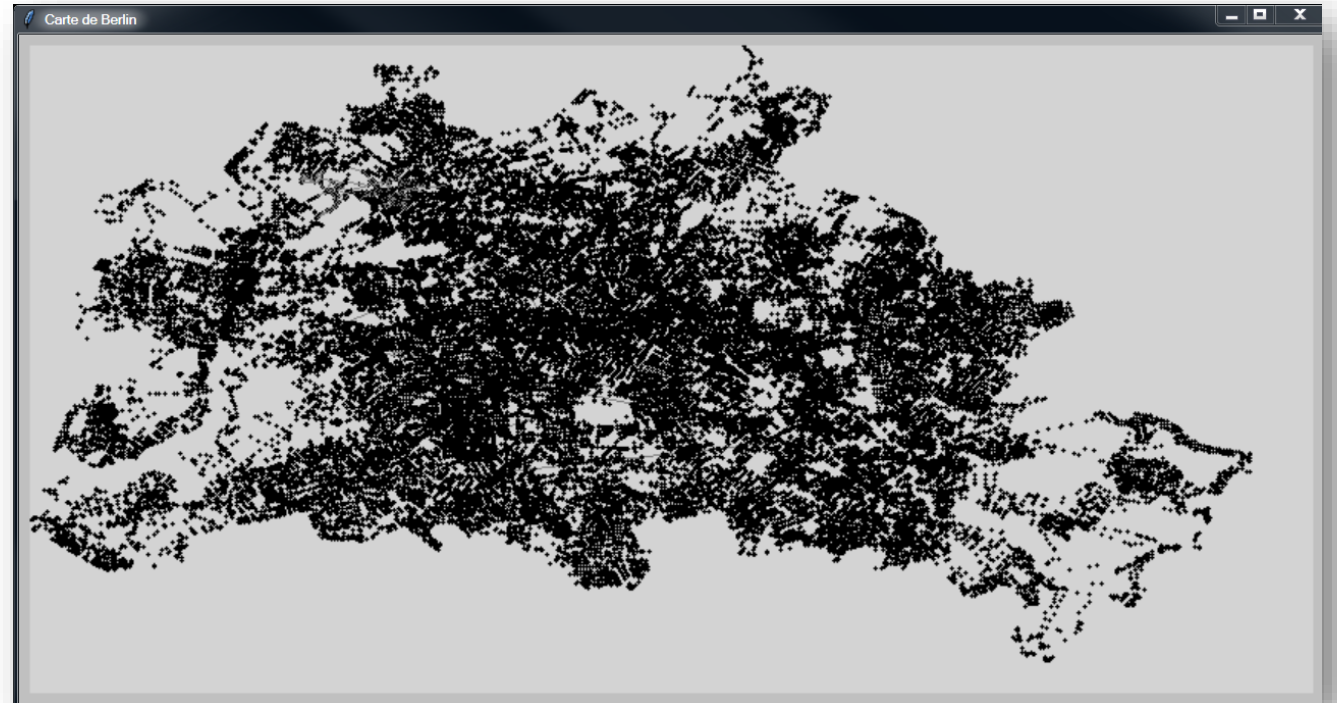
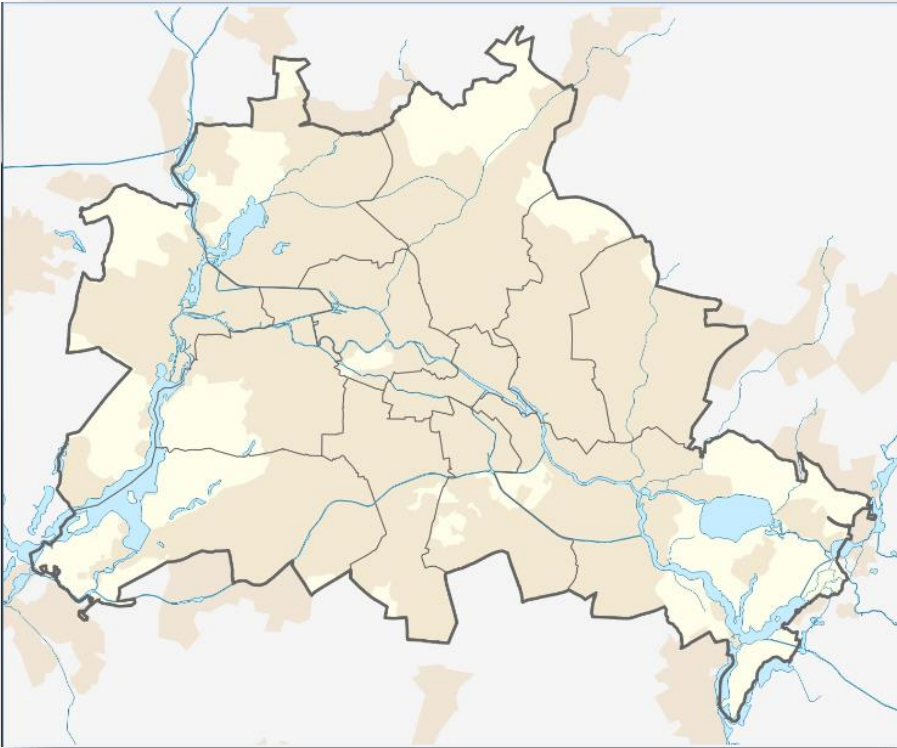
TP CHALLENGES

Polytech Tours

Environnement

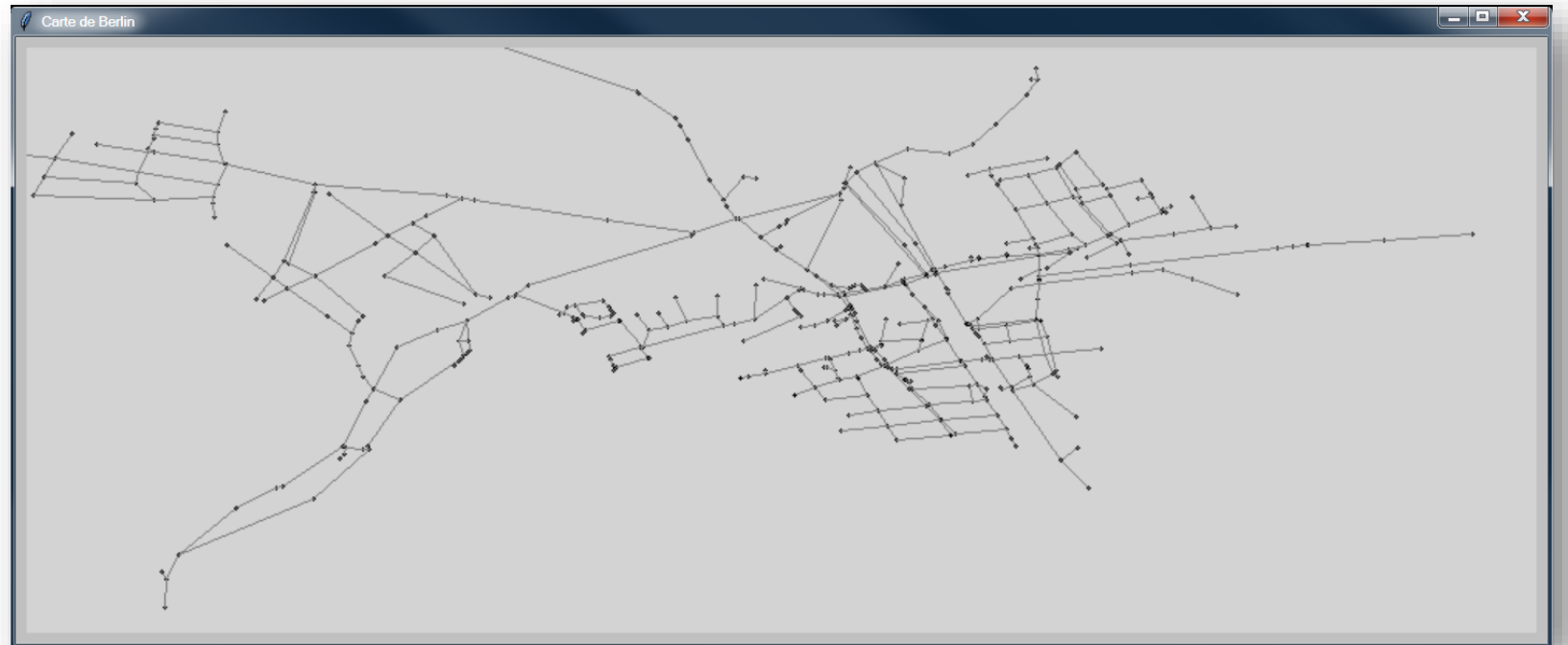
Environnement

- Tous les TP se feront sur un graphe unique.
- Ce graphe représente le plan de la ville de Berlin



Environnement

- Le graphe comporte 59673 nœuds et 145840 arcs.
- On travaillera d'abord sur un graphe partiel (« toy ») comportant seulement 444 sommets et 977 arcs, avant de tester ses algorithmes sur le graphe complet.



Environnement

Affichage du graphe

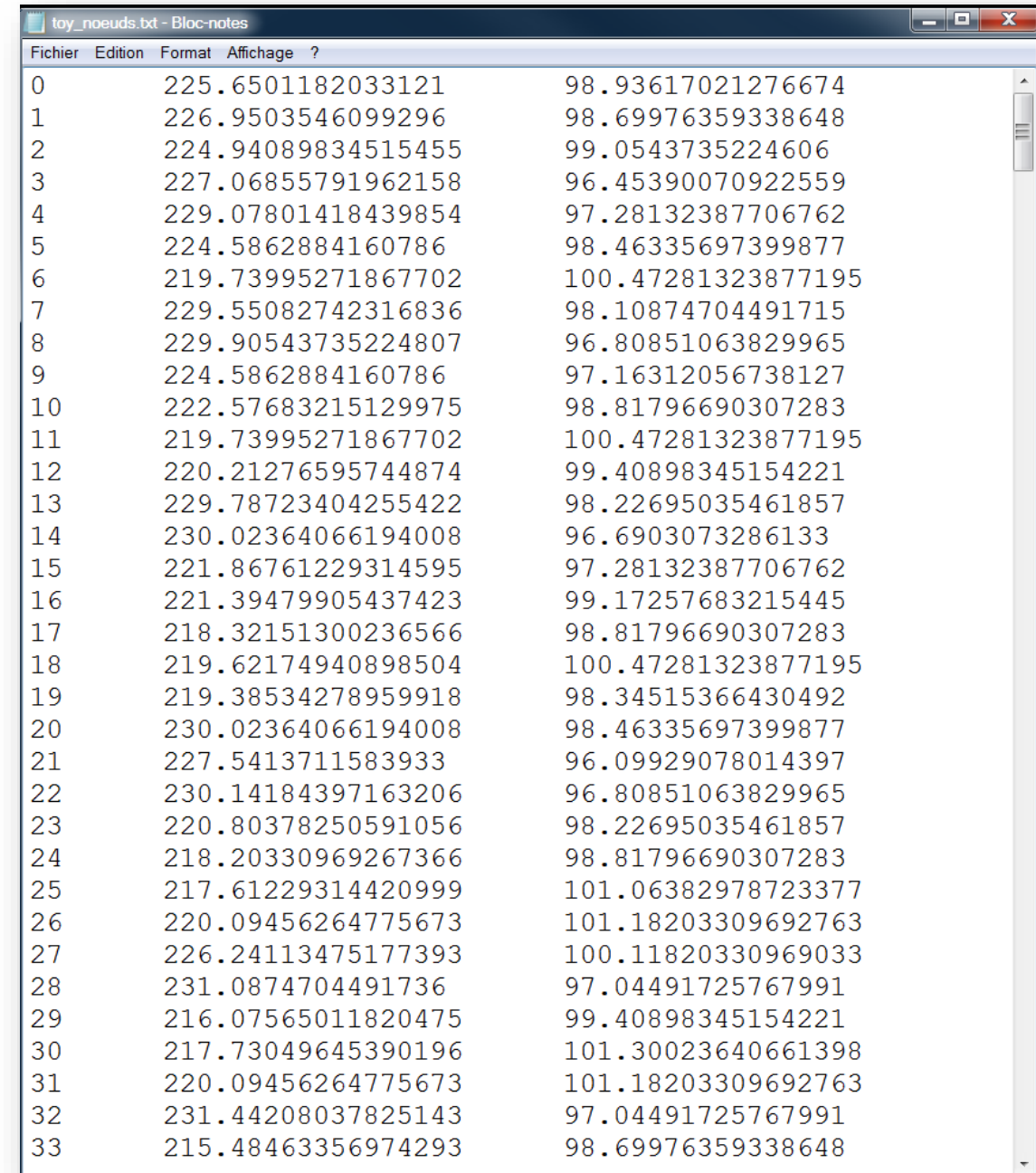
- Le booléen « graphe_toy » s'il est à 1 fera ouvrir le fichier d'exercice, s'il est à 0 fera ouvrir le graphe complet.

```
# #####  
# Lecture des fichiers  
# #####  
  
graphe_toy = 1  
  
graphe_berlin = 1-graphe_toy  
  
if graphe_toy:  
    fichier_noeuds = 'toy_noeuds.txt' # 'berlin_noeuds.txt'  
    fichier_arcs = 'toy_arcs.txt' # 'berlin_arcs.txt'  
    zoom = 10  
else:  
    fichier_noeuds = 'berlin_noeuds.txt' # 'toy_noeuds.txt' # 'berlin_noeuds.txt'  
    fichier_arcs = 'berlin_arcs.txt' # 'toy_arcs.txt' # 'berlin_arcs.txt'  
    zoom = 1
```

Environnement

Lire les nœuds du graphe

- Ouvrir le fichier `fichier_noeuds`
- Lire toutes les lignes `readlines()`
- Fermer le fichier
- Initialiser 2 listes à vide : `X`, `Y`
 - Remplir ces listes (convertir les strings en float)
- On appellera `NbNoeuds` le nombre de noeuds



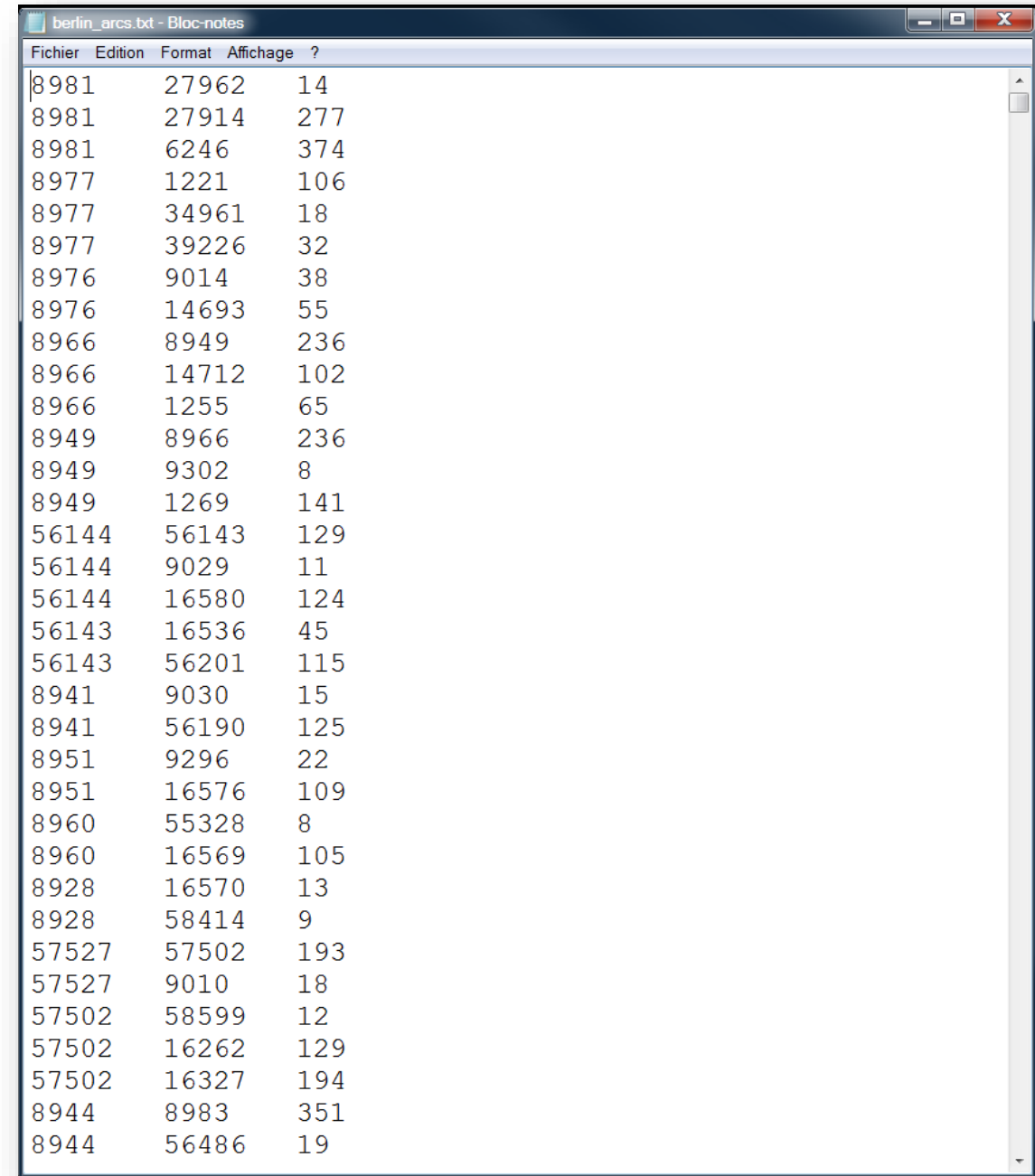
0	225.6501182033121	98.93617021276674
1	226.9503546099296	98.69976359338648
2	224.94089834515455	99.0543735224606
3	227.06855791962158	96.45390070922559
4	229.07801418439854	97.28132387706762
5	224.5862884160786	98.46335697399877
6	219.73995271867702	100.47281323877195
7	229.55082742316836	98.10874704491715
8	229.90543735224807	96.80851063829965
9	224.5862884160786	97.16312056738127
10	222.57683215129975	98.81796690307283
11	219.73995271867702	100.47281323877195
12	220.21276595744874	99.40898345154221
13	229.78723404255422	98.22695035461857
14	230.02364066194008	96.6903073286133
15	221.86761229314595	97.28132387706762
16	221.39479905437423	99.17257683215445
17	218.32151300236566	98.81796690307283
18	219.62174940898504	100.47281323877195
19	219.38534278959918	98.34515366430492
20	230.02364066194008	98.46335697399877
21	227.5413711583933	96.09929078014397
22	230.14184397163206	96.80851063829965
23	220.80378250591056	98.22695035461857
24	218.20330969267366	98.81796690307283
25	217.61229314420999	101.06382978723377
26	220.09456264775673	101.18203309692763
27	226.24113475177393	100.11820330969033
28	231.0874704491736	97.04491725767991
29	216.07565011820475	99.40898345154221
30	217.73049645390196	101.30023640661398
31	220.09456264775673	101.18203309692763
32	231.44208037825143	97.04491725767991
33	215.48463356974293	98.69976359338648

Environnement

Lire les arcs du graphe

- Ouvrir le fichier `fichier_arcs`
- Lire toutes les lignes `readlines()`
- Fermer le fichier
- Initialiser 3 listes à vide : `Origine`, `Destination`, `Longueur`
- Remplir ces listes (convertir les strings en int)
- On appellera `NbArcs` le nombre d'arcs
- Ajouter (pour affichage)

```
minX = min(X)
minY = min(Y)
for j in range(NbNoeuds):
    X[j] = (X[j]-minX)*zoom
    Y[j] = (Y[j]-minY)*zoom
```



8981	27962	14
8981	27914	277
8981	6246	374
8977	1221	106
8977	34961	18
8977	39226	32
8976	9014	38
8976	14693	55
8966	8949	236
8966	14712	102
8966	1255	65
8949	8966	236
8949	9302	8
8949	1269	141
56144	56143	129
56144	9029	11
56144	16580	124
56143	16536	45
56143	56201	115
8941	9030	15
8941	56190	125
8951	9296	22
8951	16576	109
8960	55328	8
8960	16569	105
8928	16570	13
8928	58414	9
57527	57502	193
57527	9010	18
57502	58599	12
57502	16262	129
57502	16327	194
8944	8983	351
8944	56486	19

Environnement

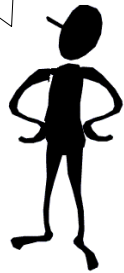
Dessiner le graphe

- Saisir les fonctions ci-contre et la définition du canvas

- Terminer par :

```
can.pack()  
fen.mainloop()
```

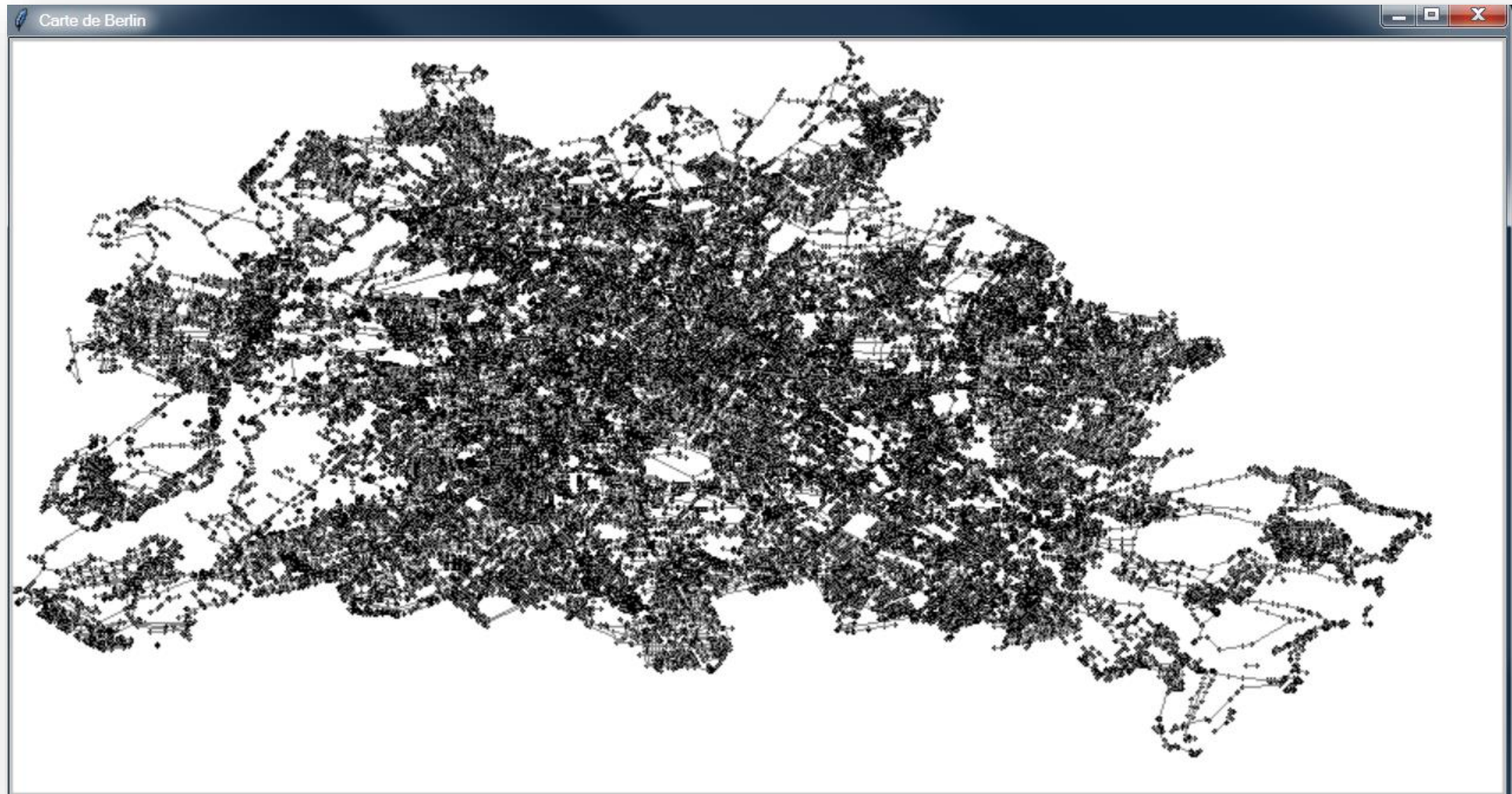
*Ces 2 lignes sont
toujours les
dernières de votre
programme*



- On obtient les cartes souhaitées.

```
# #####  
# Dessin du graphe  
# #####  
print('*** Dessin du graphe ***')  
  
def TraceCercle(j,couleur,rayon,ep):  
    can.create_oval(X[j]-rayon, Y[j]-rayon, X[j]+rayon, Y[j]+rayon, \  
                    outline = couleur, fill = couleur, width=ep)  
  
def TraceArc(j1,j2,couleur,ep):  
    can.create_line(X[j1],Y[j1],X[j2],Y[j2],fill = couleur,width=ep)  
  
fen = Tk()  
fen.title('Carte de Berlin')  
coul_fond = "lightgrey"  
#['purple','cyan','maroon','green','red','blue','orange','yellow']  
coul_noeud = "black"  
  
border = 20          # taille en px des bords  
  
Delta_X = max(X)-min(X)  
Delta_Y = max(Y)-min(Y)  
winWidth = Delta_X+2*border  
winHeight = winWidth * Delta_Y / Delta_X  
  
can = Canvas(fen, width = winWidth, height = winHeight, bg =coul_fond)  
  
# Affichage des noeuds et des arcs  
rayon_noeud = 1      # rayon pour dessin des points  
for i in range(NbNoeuds):  
    TraceCercle(i,coul_noeud,rayon_noeud,1)  
    for j in Succ[i]: TraceArc(i,j,'grey',1)
```


Environnement



Environnement

Lecture des fichiers
Création du canvas
Affichage du contour et des sommets

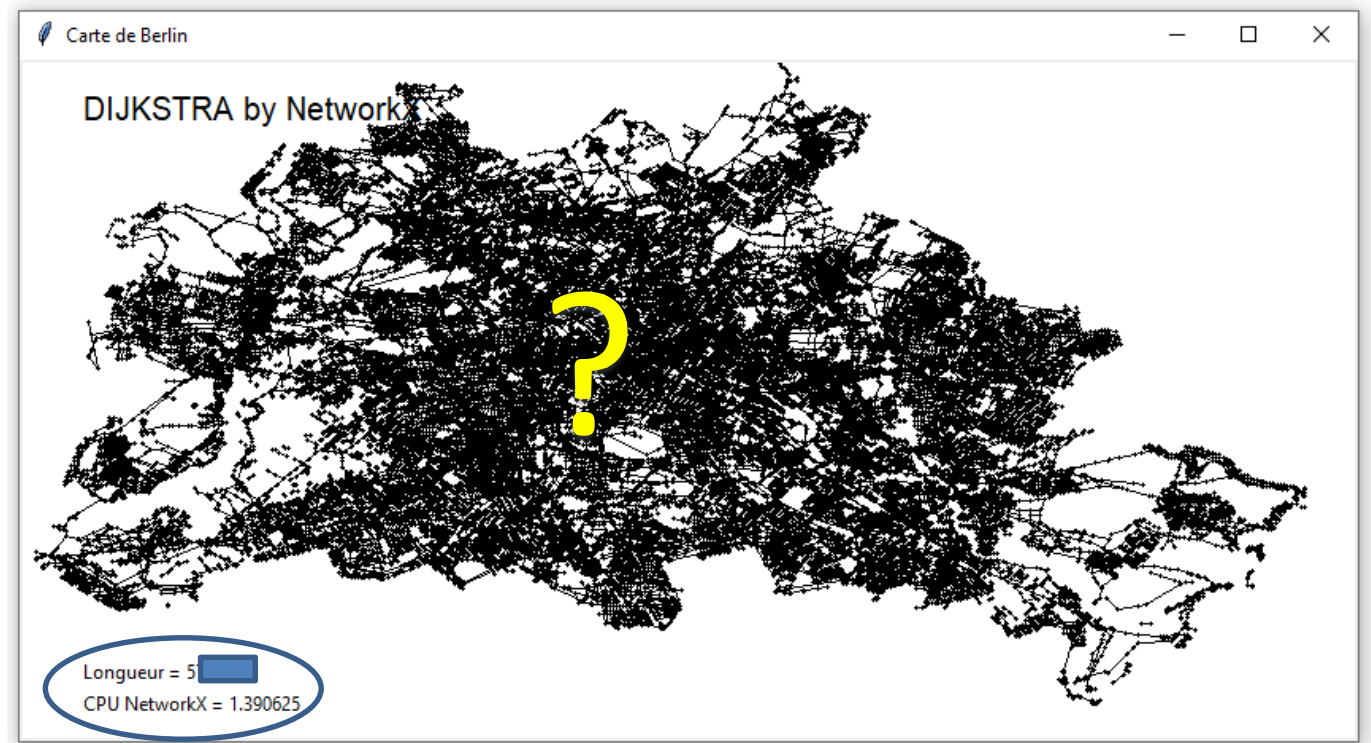
Chaque TP

`can.pack()`
`fen.mainloop()`

TP Challenges

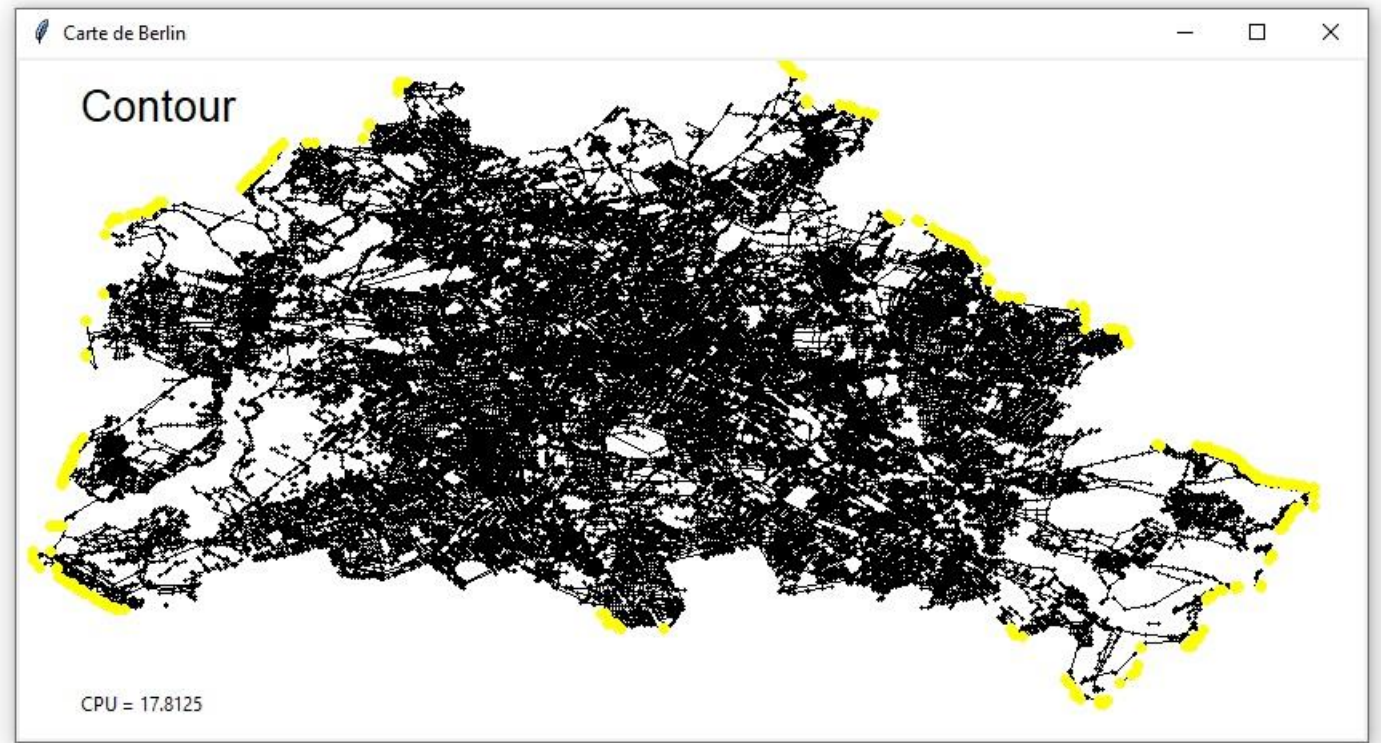
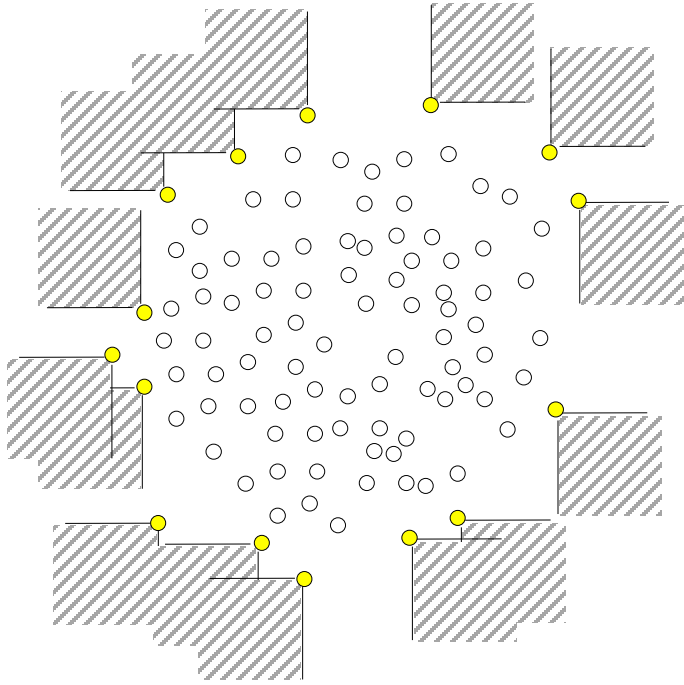
TP Challenges

55000



- Trouver deux points dans la ville pour lesquels le plus court chemin est supérieur ou égal à 55000.
- Tous les moyens sont bons, dès lors que c'est un algorithme.
- *Find two points in the city for which the shortest path is greater than or equal to 55000.*
- *All means are good, as long as it is an algorithm.*

TP Challenges



- Afficher en jaune l'ensemble des sommets « non dominés » du graphe.
- Tous les moyens sont bons, dès lors qu'il y a un algorithme.
- *Draw in yellow the set of non dominated vertices in the graph.*
- *All means are good, as long as there is an algorithm.*