Graphes et Complexité
TP3 Dijkstra v1

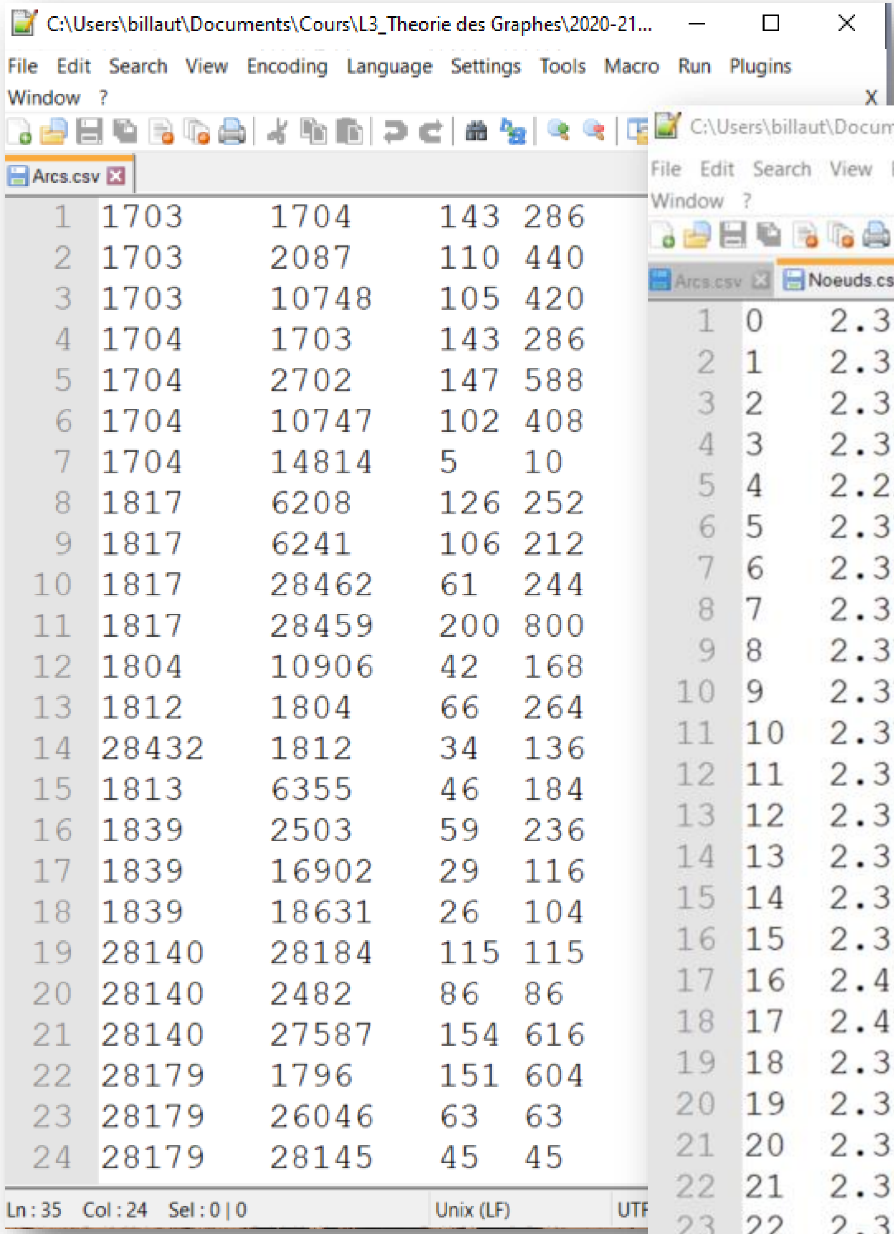
UFR Sciences et Techniques
2020-2021

TP3 DISJKSTRA

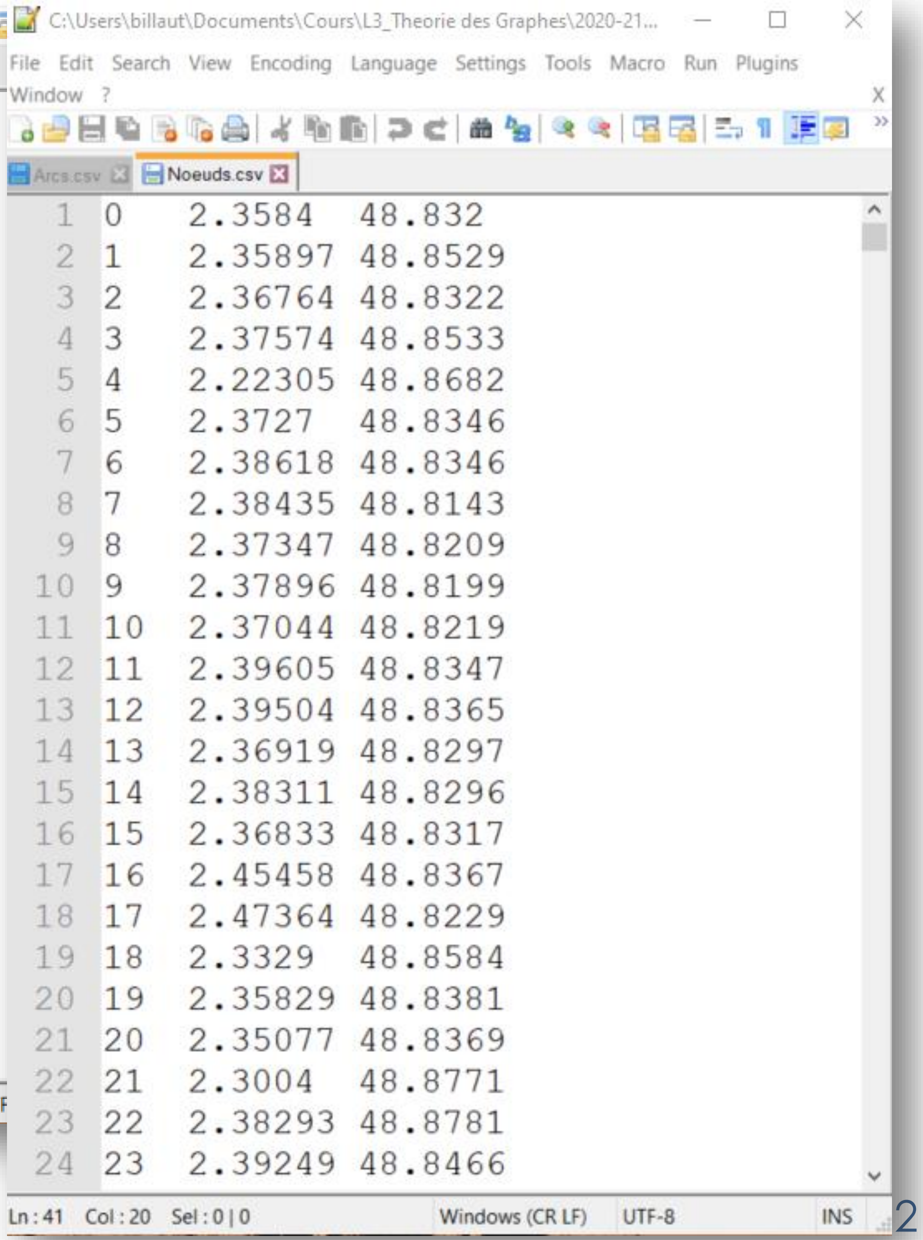
Préambule

- On dispose de deux fichiers :
  - Arcs.csv
  - Nœuds.csv

- Arcs.csv contient pour chaque arc :
  - Sommet origine
  - Sommet destination
  - Longueur
  - « dangerosité »
- Nœuds.csv contient pour chaque nœud
  - Numéro du nœud
  - Longitude
  - Latitude



1703	1704	143	286
1703	2087	110	440
1703	10748	105	420
1704	1703	143	286
1704	2702	147	588
1704	10747	102	408
1704	14814	5	10
1817	6208	126	252
1817	6241	106	212
1817	28462	61	244
1817	28459	200	800
1804	10906	42	168
1812	1804	66	264
28432	1812	34	136
1813	6355	46	184
1839	2503	59	236
1839	16902	29	116
1839	18631	26	104
28140	28184	115	115
28140	2482	86	86
28140	27587	154	616
28179	1796	151	604
28179	26046	63	63
28179	28145	45	45



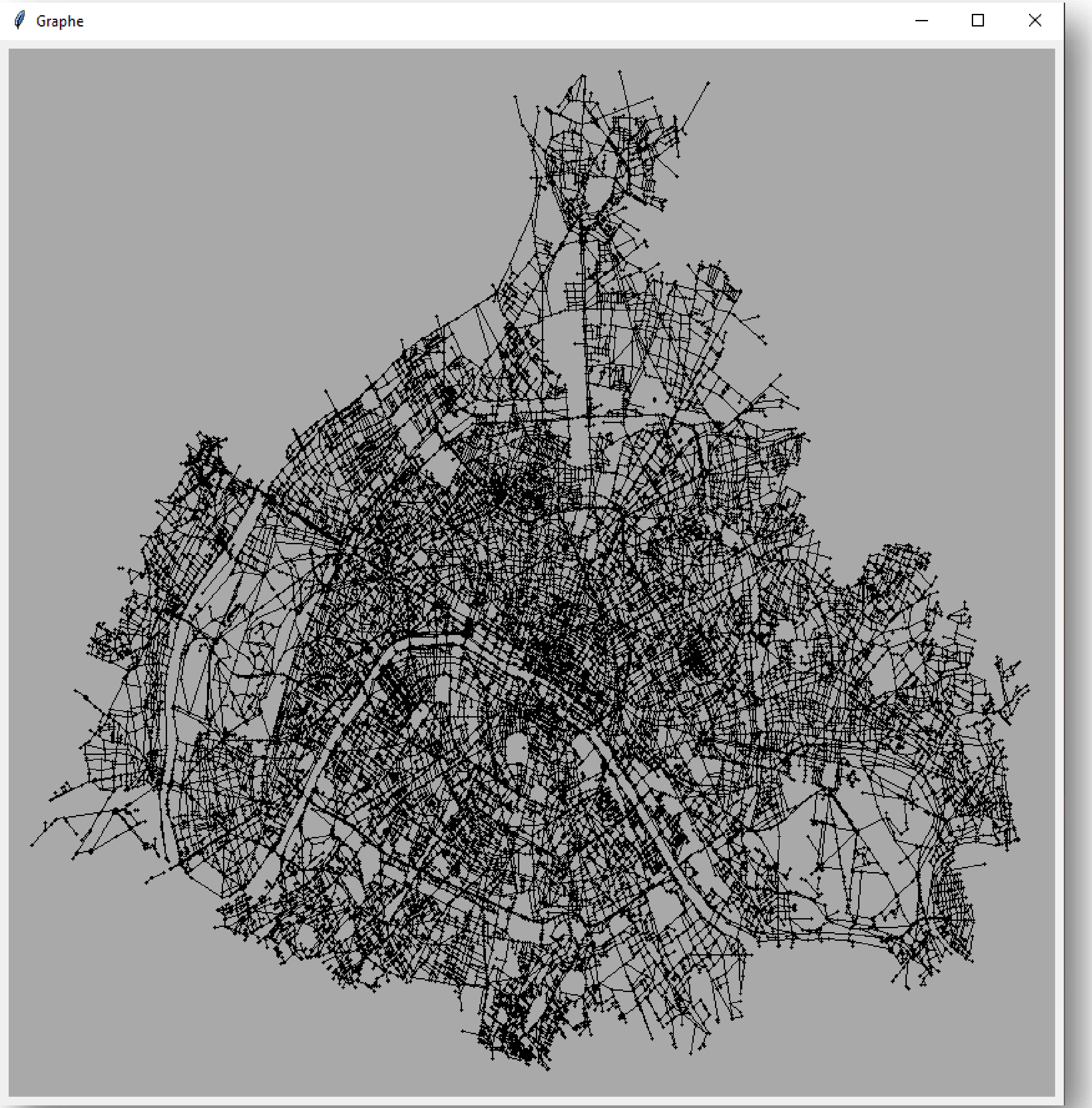
0	2.3584	48.832
1	2.35897	48.8529
2	2.36764	48.8322
3	2.37574	48.8533
4	2.22305	48.8682
5	2.3727	48.8346
6	2.38618	48.8346
7	2.38435	48.8143
8	2.37347	48.8209
9	2.37896	48.8199
10	2.37044	48.8219
11	2.39605	48.8347
12	2.39504	48.8365
13	2.36919	48.8297
14	2.38311	48.8296
15	2.36833	48.8317
16	2.45458	48.8367
17	2.47364	48.8229
18	2.3329	48.8584
19	2.35829	48.8381
20	2.35077	48.8369
21	2.3004	48.8771
22	2.38293	48.8781
23	2.39249	48.8466

TP3 DISJKSTRA

Préambule

- Ce graphe représente la ville de Paris.
- Il comporte
  - 29086 sommets
  - 64538 arcs

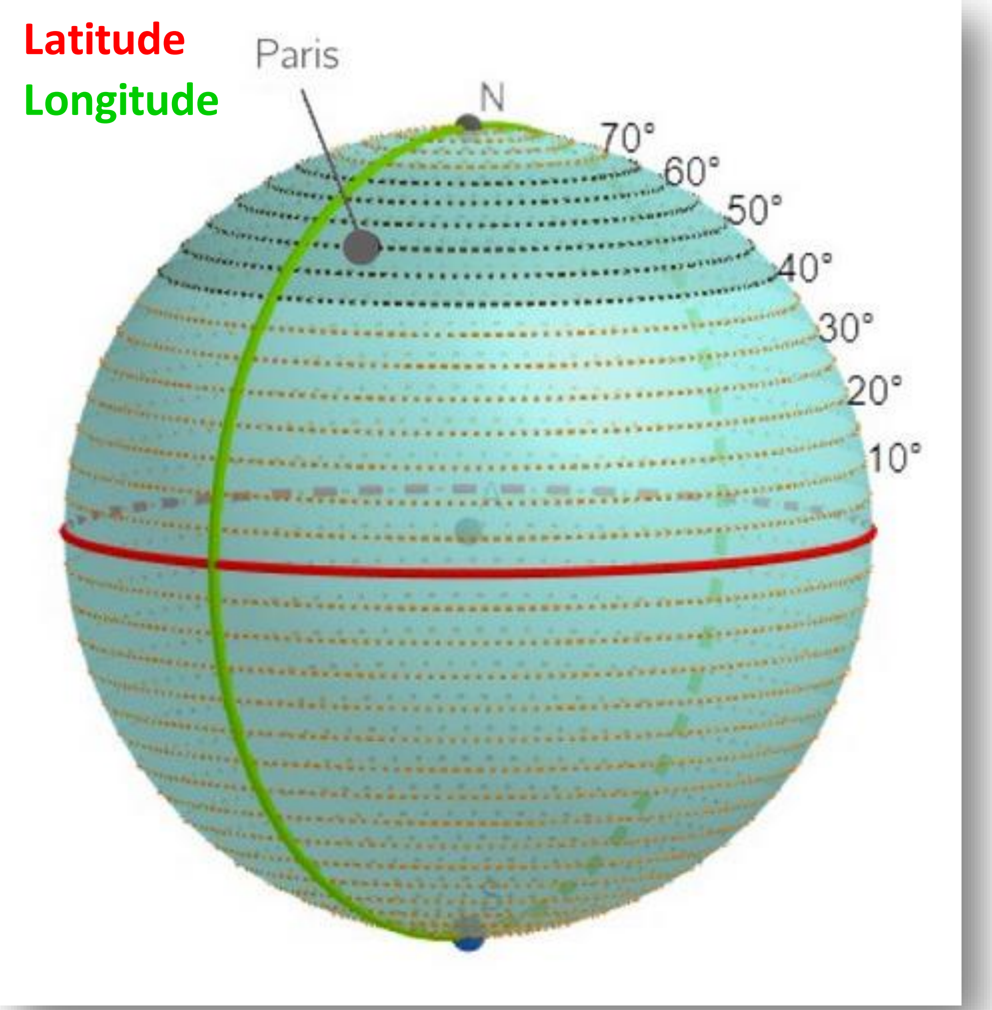
- Nous allons coder 3 algorithmes pour trouver le plus court chemin entre 2 sommets dans ce graphe :
  - Disjkstra v1 non optimisée
  - Disjkstra v2 optimisée
  - A* (« A-star »)



TP3 DISJKSTRA

Lecture des données

1. Ouvrir le fichier « Nœuds.csv »
2. Créer deux listes vides :
   - `Longitude`
   - `Latitude`
3. Pour chaque sommet lire :
   - Le numéro de l'arc (on ne le gardera pas, ce sera l'indice dans les listes)
   - La longitude
   - La latitude
4. Avant d'être stockées, les longitudes et latitudes sont converties en radians en multipliant la valeur lue par $\pi/180$.
   - π s'obtient par `math.pi` du package `math`
5. Stocker dans `minLat`, `maxLat`, `minLong`, `maxLong` les plus petites et plus grandes valeurs des latitudes et des longitudes.



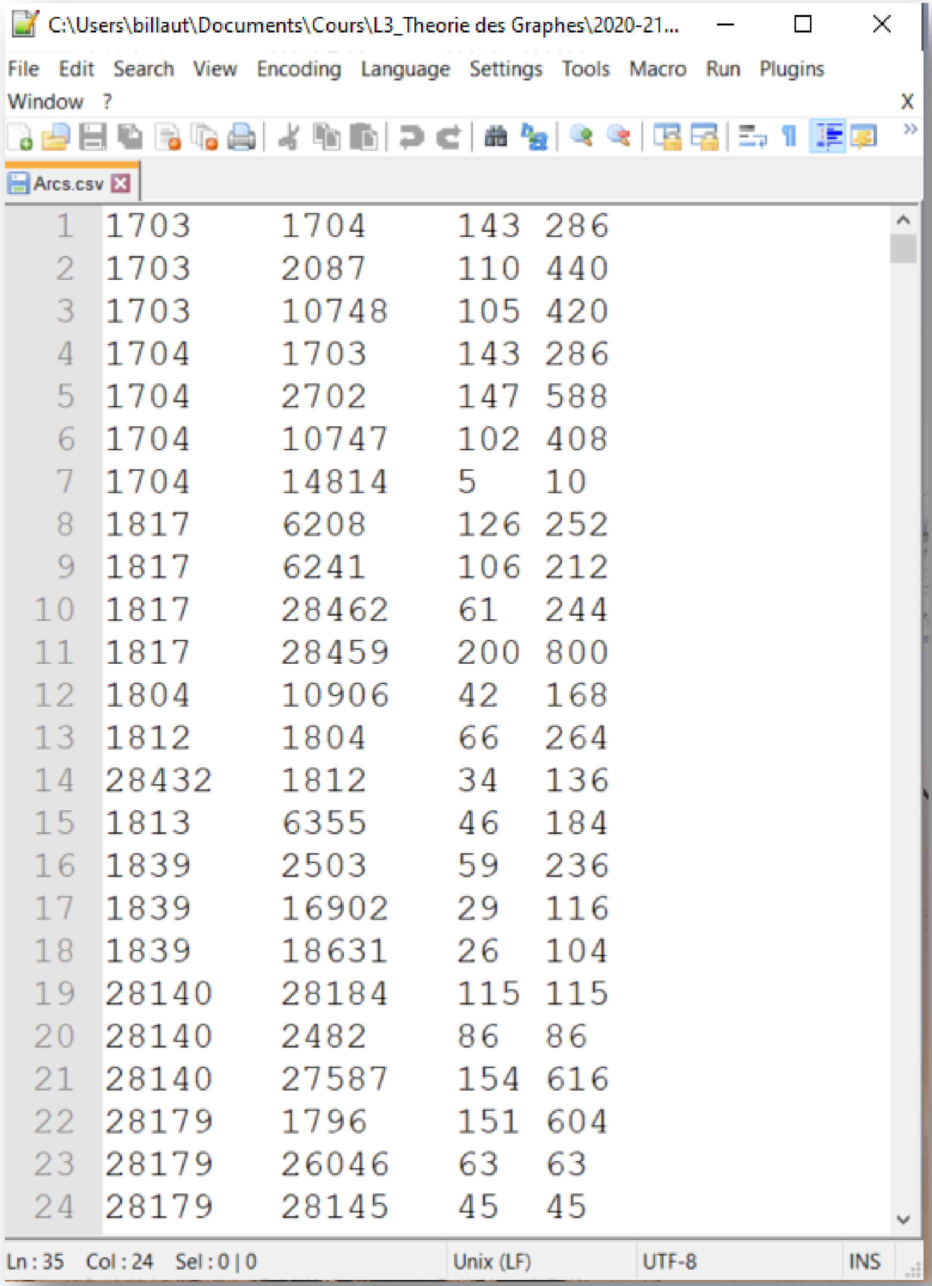
TP3 DISJKSTRA

Lecture des données

1. Ouvrir le fichier « Arcs.csv »

2. Créer 4 listes vides :
   - `Origine`
   - `Destination`
   - `Longueur`
   - `Dangerosite`

3. Pour chaque arc lire :
   - L'origine
   - La destination
   - La longueur
   - La dangerosité



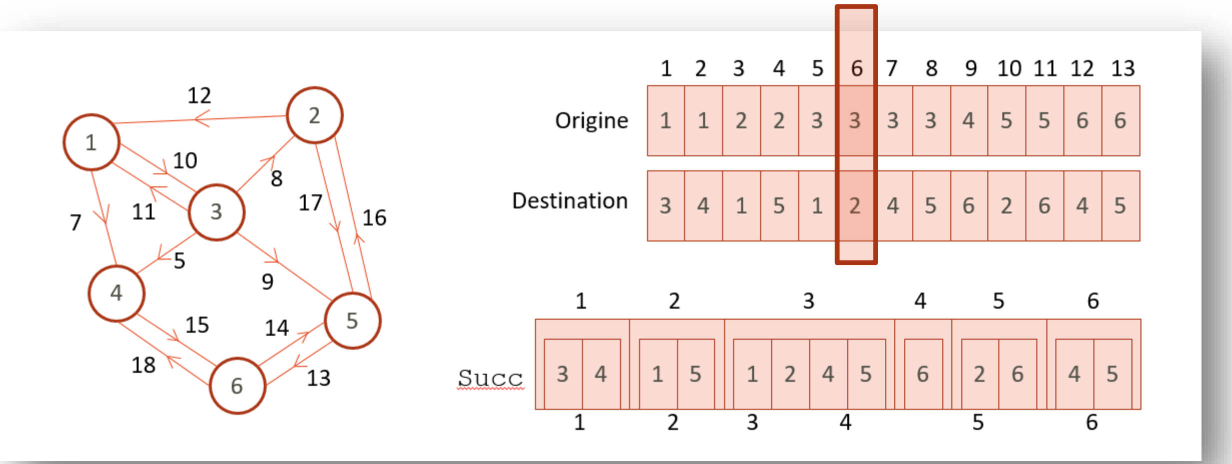
1703	1704	143	286
1703	2087	110	440
1703	10748	105	420
1704	1703	143	286
1704	2702	147	588
1704	10747	102	408
1704	14814	5	10
1817	6208	126	252
1817	6241	106	212
1817	28462	61	244
1817	28459	200	800
1804	10906	42	168
1812	1804	66	264
28432	1812	34	136
1813	6355	46	184
1839	2503	59	236
1839	16902	29	116
1839	18631	26	104
28140	28184	115	115
28140	2482	86	86
28140	27587	154	616
28179	1796	151	604
28179	26046	63	63
28179	28145	45	45

TP3 DISJKSTRA

Structure de données

1. Créer la liste des listes de successeurs : `Succ`

i=3, j=2 ⇒ Arc(i,j)=6



2. Ecrire une routine `Arc(i,j)` qui renvoie le numéro de l'arc qui commence par `i` et qui termine par `j` (en explorant tous les arcs mais en retournant le numéro dès qu'il est trouvé).

TP3 DISJKSTRA

Dessin du graphe

1. Ajouter "`from tkinter import Tk, Canvas`" en tête de votre code
2. Recopier le code fourni dans le fichier « dessin_du_graphe.txt »
3. Créer une routine `TraceSegment(i,j,couleur)` qui trace un segment entre les points `i` et `j` en utilisant `create_line()`
4. Ajouter le tracé des arcs au dessin
5. L'affichage se fera en fin de code par l'instruction "`fen.mainloop()`", mais il peut être déjà testé à cette étape.

```python
# ##########################################################
# Dessin du graphe
# ##########################################################

print('****************************************')
print('* Dessin du graphe                     *')
print('****************************************')

def cercle(x,y,r,couleur):
    can.create_oval(x-r, y-r, x+r, y+r, outline = couleur, fill = couleur)

def TraceCercle(j,couleur,rayon):
    x=(Longitude[j]-minLong)*ratioWidth + border
    y=((Latitude[j]-minLat)*ratioHeight) + border
    y=winHeight-y
    cercle(x,y,rayon,couleur)

fen = Tk()
fen.title('Graphe')
coul = "dark green"   #['purple','cyan','maroon','green','red','blue','orange','yellow']

Delta_Long = maxLong-minLong
Delta_Lat = maxLat-minLat
border = 20         # taille en px des bords
winWidth_int = 800
winWidth = winWidth_int+2*border     # largeur de la fenetre
winHeight_int = 800
winHeight = winHeight_int+2*border     # hauteur de la fenetre : recalculee en fonction de la taille du graphe
#ratio= 1.0          # rapport taille graphe / taille fenetre
ratioWidth = winWidth_int/(maxLong-minLong)       #  rapport largeur graphe/ largeur de la fenetre
ratioHeight = winHeight_int/(maxLat-minLat)       #  rapport hauteur du graphe hauteur de la fenetre

can = Canvas(fen, width = winWidth, height = winHeight, bg ='dark grey')
can.pack(padx=5,pady=5)

#  cercles
rayon = 1              # rayon pour dessin des sommets
rayon_od = 5          # rayon pour sommet origine et destination
# Affichage de tous les sommets
for i in range(0,NbSommets):
    TraceCercle(i,'black',rayon)
```

TP3 DISJKSTRA

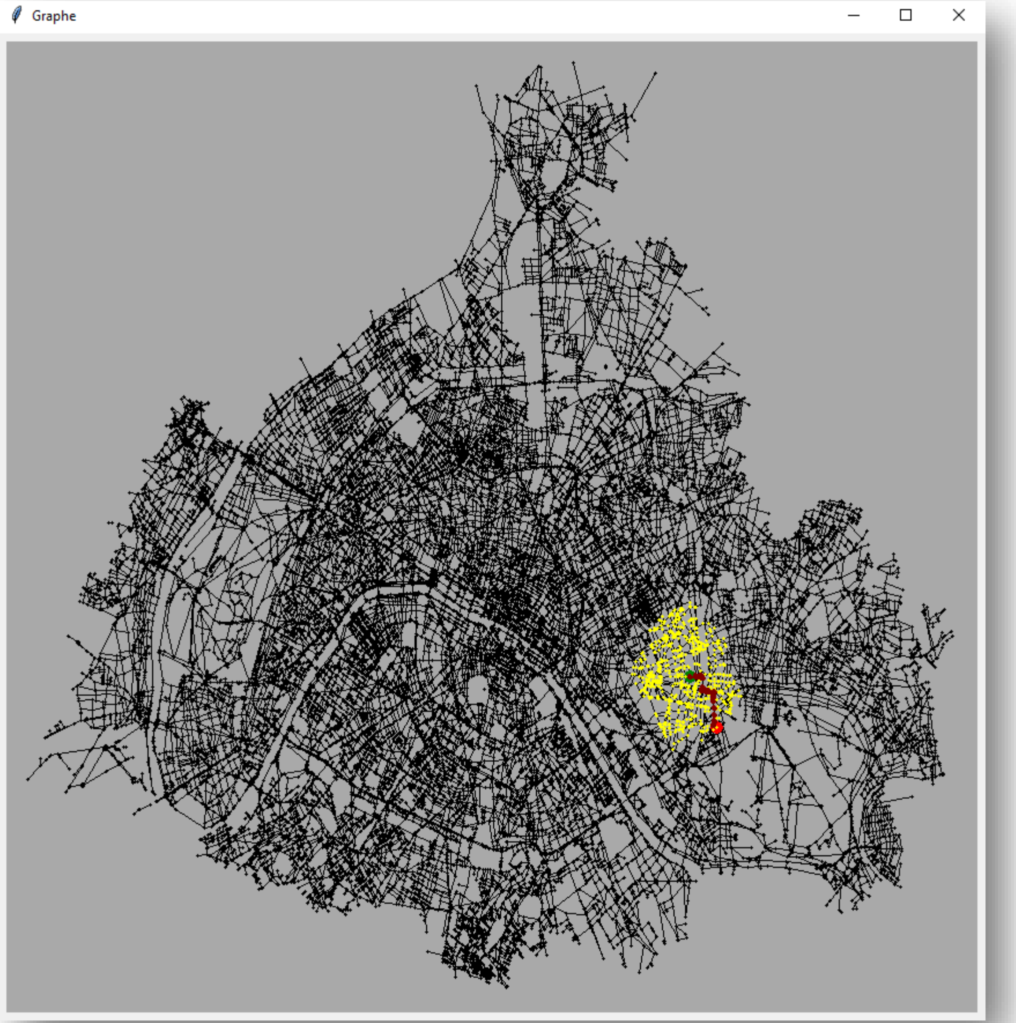
Algorithme de Disjkstra

1. On partira du `sommet_depart = 3000`

2. On cherchera un chemin jusqu'au `sommet_destination = 11342` (puis on testera jusqu'au sommet `22279`).
3. On initialise une variable `time_start` à `time.clock()` (ou `time.process_time()` selon les versions de Python) pour prendre le temps de calcul (inclure le package `time`)

4. Afficher le sommet départ en appelant `TraceCercle(sommet_depart,'green',rayon_od)`

5. Afficher le sommet destination en appelant `TraceCercle(sommet_destination,'red',rayon_od)`



TP3 DISJKSTRA

Algorithme de Disjkstra

1. Initialiser
   - La liste des potentiels `Pi` à l'infini pour tous les sommets
   - La liste des potentiels définitifs `Piprime` à l'infini pour tous les sommets
   - La liste des pères `LePere` à `-1` pour tous les sommets
   - La liste des marques `marque` à `0` pour tous les sommets.
2. Initialiser
   - Le potentiel `Pi` de `sommet_depart` à `0`
   - Le potentiel `Piprime` de `sommet_depart` à `0`
   - **Pour** chaque successeur `j` de `sommet_depart`, **Faire**
     - initialiser le potentiel `Pi` de `j` à la longueur de l'arc (obtenue grâce à `Longueur[Arc(sommet_depart,j)]`)
     - Initialiser le `Père` de `j` à `sommet_depart`
   - **Finpour**

Dijkstra

- Initialisation :
  - $\pi(s) \leftarrow 0, \bar{\pi}(s) \leftarrow 0$
  - $\bar{S} \leftarrow \{1,2,\dots,N\} \backslash \{s\}$
  - $\pi(j) \leftarrow \begin{cases} l_{s,j} & \text{si } (s,j) \in U \\ +\infty & \text{sinon} \end{cases}$
  - $P(j) \leftarrow \begin{cases} s & \text{si } (s,j) \in U \\ -1 & \text{sinon} \end{cases}$
- Tantque $\bar{S} \neq \emptyset$ Faire
  - Sélectionner $j \in \bar{S}, \pi(j) = \min_{i \in \bar{S}} \pi(i)$
  - $\bar{S} \leftarrow \bar{S} \backslash \{j\}$
  - $\bar{\pi}(j) \leftarrow \pi(j)$
  - Pour tout $(j,k) \in U, k \in \bar{S}$ Faire
    - Si $\pi(k) \geq \pi(j) + l_{j,k}$ Alors
      - $\pi(k) \leftarrow \pi(j) + l_{j,k}$
      - $P(k) \leftarrow j$
  - FinPour
- FinTantque

TP3 Disjkstra

Algorithme de Disjkstra

1. Initialiser un compteur `nb_sommets_explores` à `0`
2. Initialiser un booléen `fini` à `Faux`
3. **Tant que** `nb_sommets_explores < NbSommets` et `pas fini` **Faire**

// on mettra fini à `Vrai` quand on aura trouvé `sommet_destination`

// mais on boucle aussi tant que le nombre de sommets explorés est inférieur à `NbSommets` au cas où le graphe ne serait pas connexe… (si pas de chemin entre `sommet_origine` et `sommet_destination`)

   1. Parcourir tous les sommets et parmi ceux qui n'ont pas encore un potentiel définitif, trouver celui qui a le plus petit potentiel, noté `sommet_retenu`.
   2. Marquer ce sommet
   3. Mettre à jour `Piprime` pour ce sommet
   4. Afficher ce sommet en jaune par `TraceCercle(sommet_retenu,'yellow',1)`

   …(suite au dos)

Dijkstra

- Initialisation :
  - $\pi(s) \leftarrow 0, \bar{\pi}(s) \leftarrow 0$
  - $\bar{S} \leftarrow \{1,2,\dots,N\} \backslash \{s\}$
  - $\pi(j) \leftarrow \begin{cases} l_{s,j} & \text{si } (s,j) \in U \\ +\infty & \text{sinon} \end{cases}$
  - $P(j) \leftarrow \begin{cases} s & \text{si } (s,j) \in U \\ -1 & \text{sinon} \end{cases}$
- Tantque $\bar{S} \neq \emptyset$ Faire
  - Sélectionner $j \in \bar{S}, \pi(j) = \min_{i \in \bar{S}} \pi(i)$
  - $\bar{S} \leftarrow \bar{S} \backslash \{j\}$
  - $\bar{\pi}(j) \leftarrow \pi(j)$
  - Pour tout $(j,k) \in U, k \in \bar{S}$ Faire
    - Si $\pi(k) \geq \pi(j) + l_{j,k}$ Alors
      - $\pi(k) \leftarrow \pi(j) + l_{j,k}$
      - $P(k) \leftarrow j$
  - FinPour
- FinTantque

TP3 DISJKSTRA

Algorithme de Disjkstra

1. Initialiser un compteur `nb_sommets_explores` à `0`
2. Initialiser un booléen `fini` à `Faux`
3. Tant que `nb_sommets_explores < NbSommets` et `pas fini` Faire

// on mettra fini à `Vrai` quand on aura trouvé `sommet_destination`

// mais on boucle aussi tant que le nombre de sommets explorés est inférieur à `NbSommets` au cas où le graphe n'est pas connexe… (si pas de chemin entre `sommet_origine` et `sommet_destination`)

   1. Parcourir tous les sommets et parmi ceux qui n'ont pas encore un potentiel définitif, trouver celui qui a le plus petit potentiel, noté `sommet_retenu`.
   2. Marquer ce sommet
   3. Mettre à jour `Piprime` pour ce sommet
   4. Afficher ce sommet en jaune par `TraceCercle(sommet_retenu,'yellow',1)`
   5. **Si** le `sommet_retenu` est le `sommet_destination` **Alors** mettre fini à `Vrai`
   6. **Pour** tout successeur `k` de `sommet_retenu` , $k \in \bar{S}$ **Faire**
      1. Mettre à jour le potentiel de `k`
      2. Mémoriser le `Père` si besoin
   7. **Finpour**
8. **FinTantQue**

Dijkstra

- Initialisation :
  - $\pi(s) \leftarrow 0, \bar{\pi}(s) \leftarrow 0$
  - $\bar{S} \leftarrow \{1,2,\dots,N\} \backslash \{s\}$
  - $\pi(j) \leftarrow \begin{cases} l_{s,j} & \text{si } (s,j) \in U \\ +\infty & \text{sinon} \end{cases}$
  - $P(j) \leftarrow \begin{cases} s & \text{si } (s,j) \in U \\ -1 & \text{sinon} \end{cases}$
- Tantque $\bar{S} \neq \emptyset$ Faire
  - Sélectionner $j \in \bar{S}, \pi(j) = \min_{i \in \bar{S}} \pi(i)$
  - $\bar{S} \leftarrow \bar{S} \backslash \{j\}$
  - $\bar{\pi}(j) \leftarrow \pi(j)$
  - Pour tout $(j,k) \in U, k \in \bar{S}$ Faire
    - Si $\pi(k) \geq \pi(j) + l_{j,k}$ Alors
      - $\pi(k) \leftarrow \pi(j) + l_{j,k}$
      - $P(k) \leftarrow j$
  - FinPour
- FinTantque

TP3 DISJKSTRA

Algorithme de Disjkstra

1. Identifier l'ensemble des arcs qui constituent le chemin trouvé et calculer la longueur du chemin
2. Afficher chaque sommet qui se trouve sur le chemin en utilisant `TraceCercle(j,'maroon',2)`
3. On initialise une variable `time_end` à `time.clock()` pour prendre le temps de calcul.
4. Afficher le temps de calcul de la routine.
5. Terminer par `fen.mainloop()`

Dijkstra

- Initialisation :
  - $\pi(s) \leftarrow 0, \bar{\pi}(s) \leftarrow 0$
  - $\bar{S} \leftarrow \{1,2,\ldots,N\} \backslash \{s\}$
  - $\pi(j) \leftarrow \begin{cases} l_{s,j} & \text{si } (s,j) \in U \\ +\infty & \text{sinon} \end{cases}$
  - $P(j) \leftarrow \begin{cases} s & \text{si } (s,j) \in U \\ -1 & \text{sinon} \end{cases}$
- Tantque $\bar{S} \neq \emptyset$ Faire
  - Sélectionner $j \in \bar{S}, \pi(j) = \min_{i \in \bar{S}} \pi(i)$
  - $\bar{S} \leftarrow \bar{S} \backslash \{j\}$
  - $\bar{\pi}(j) \leftarrow \pi(j)$
  - Pour tout $(j,k) \in U, k \in \bar{S}$ Faire
    - Si $\pi(k) \geq \pi(j) + l_{j,k}$ Alors
      - $\pi(k) \leftarrow \pi(j) + l_{j,k}$
      - $P(k) \leftarrow j$
  - FinPour
- FinTantque