Université Paris-Est Créteil Val-de-Marne
Faculté des sciences et technologies
M1 ISC/TSI 2019-2020

COMPTE RENDU

TP Projet Python

TP + compte rendu Réalisés par :

- **BENSEMMAR Mohamed Aimene**

PROGRAMMATION PYTHON

Calcul du Plus court chemin

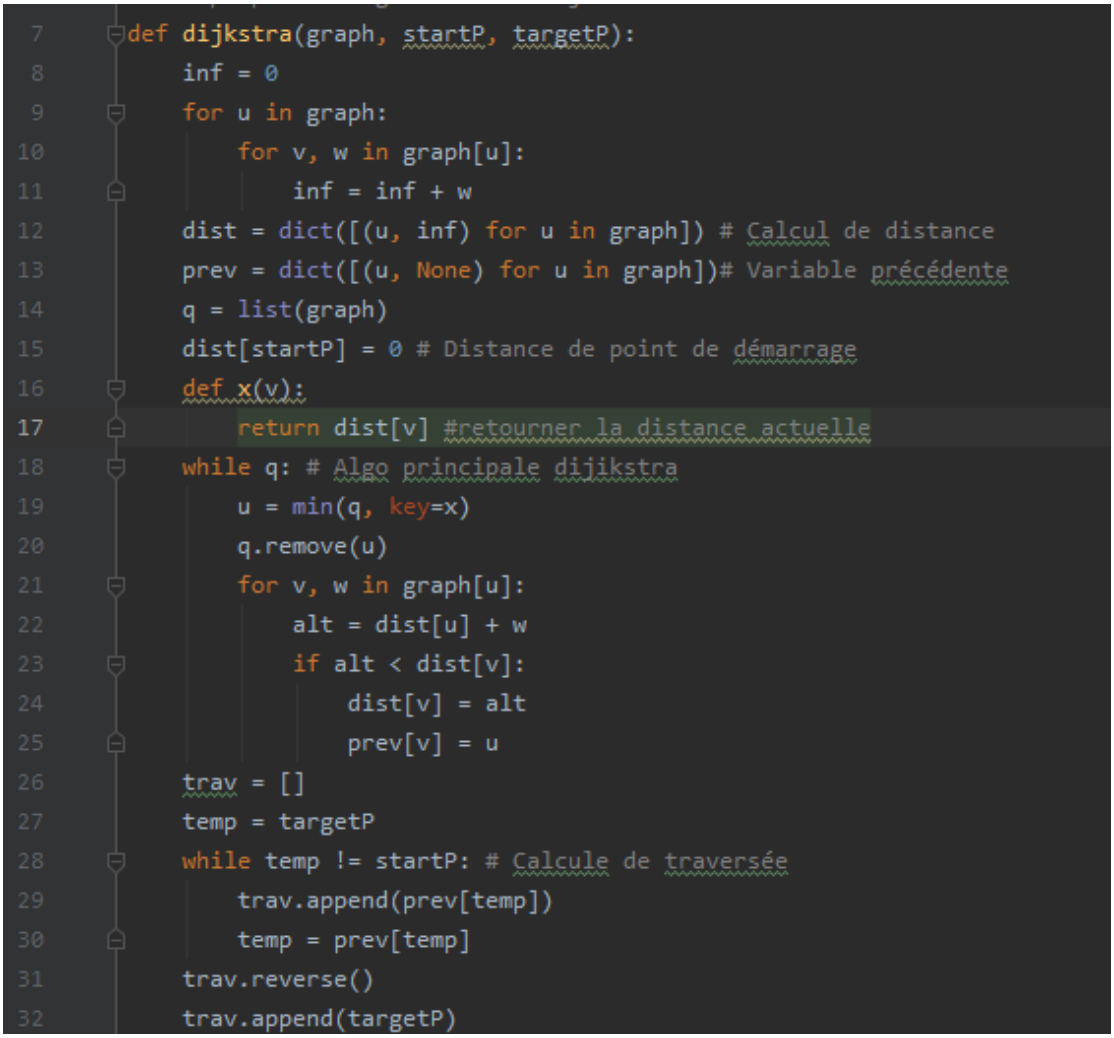
Projet Paris Metro

Objectif :

Implémentation de l'algorithme de Dijkstra étudié en cours.
L'algorithme de Dijkstra permet d'affecter à chaque arc du graphe un poids. Le chemin le plus court est déterminé par le poids des arcs.
L'implémentation Dijkstra est appliquée au réseau de métro de Paris, chaque station étant un nœud et le trajet entre deux nœuds étant un arc. L'objectif est donc de trouver le chemin le plus court entre deux stations.

Définition des fonctions :

Fonction dijikstra pour l'algorithme de Dijkstra : (Calcule de traversée ou chemin + la distance parcourue), on a comme paramètre un graphe, le point de départ et le point d'arrivée), le retour est la traversée + la distance.



```python
def dijkstra(graph, startP, targetP):
    inf = 0
    for u in graph:
        for v, w in graph[u]:
            inf = inf + w
    dist = dict([(u, inf) for u in graph]) # Calcul de distance
    prev = dict([(u, None) for u in graph])# Variable précédente
    q = list(graph)
    dist[startP] = 0 # Distance de point de démarrage
    def x(v):
        return dist[v] #retourner la distance actuelle
    while q: # Algo principale dijikstra
        u = min(q, key=x)
        q.remove(u)
        for v, w in graph[u]:
            alt = dist[u] + w
            if alt < dist[v]:
                dist[v] = alt
                prev[v] = u
    trav = []
    temp = targetP
    while temp != startP: # Calcule de traversée
        trav.append(prev[temp])
        temp = prev[temp]
    trav.reverse()
    trav.append(targetP)
```

Fonction dijAlgo qui donne le format d'un graphe à partir d'une liste de sources :

```
36    # Liste vers Graphe pour l'algorithme de Dijikstra
37    def dijAlgo(lists_sources_lists):
38        for k in xrange(len(list_keys)):
39            for unit in list_source_destination:
40                if unit[0] == k:
41                    lists_sources_lists.append((unit[1], unit[2]))
42                else:
43                    pass
44            if len(lists_sources_lists) != 0:
45                graph_dijkstra[k] = lists_sources_lists
46                lists_sources_lists = []
47
```

Fonction short_path qui permet de retourner une liste contient les numéros des stations avec leurs noms :

```
49    # Joindre le chemin le plus court avec le dictionnaire (n° de chaque station)
50    def short_path(l, dict_stations, flag=None):
51        list_station = []
52        index = 0
53        for unit in l:
54            index = index + 1
55            list_station.append(("[{0}] Station N°{1}: {2}".format(index, unit, dict_stations[unit])))
56        if flag:
57            return ''.join(list_station)
58        else:
59            return list_station
60
```

Fonction dictionary permet la création de dictionnaire à partir des données recueillis déjà de notre fichier texte :

```
62    def dictionary():
63        # Création de dictionnaire en chargeant le jeu de données (dataset) (.txt)
64        for i in xrange(len(line_dataset) - 1):
65            # xrange() Génèration des nombres à la demande
66            # range() pré-calcule tous les nombres et les enregistre en mémoire, ce qui provoque l'erreur.
67            if not re.match(r'([\d]+)', line_dataset[i]):
68                line_temp = i
69
70        for i in xrange(line_temp + 1, len(line_dataset) - 1):
71            list_source_destination.append(
72                (int(line_dataset[i].strip().split(" ")[0]), int(line_dataset[i].strip().split(" ")[1]),
73                 float(line_dataset[i].strip().split(" ")[2])))
74            list_keys.append(int(line_dataset[i].strip().split(" ")[0]))
75
```

Fonction dataset_with_dict permet de construire un dictionnaire (n° station – station) à partir de fichier texte metro_paris.txt : principe est simple, on lit le fichier ligne par ligne jusqu'on trouve la ligne [Edges] : et on retourne comme résultat un dictionnaire :

```python
# dataset avec dictionnaire
def dataset_with_dict():
    # Variable locale pour l'ensemble de données (dataset)
    f = open('metro_paris.txt', 'r') # Ouverture de fichier
    stations_with_dictinary = []
    my_stations_with_dict = {}
    for unit in f.readlines()[1:]:
        if unit.strip() != '[Edges]': #Tant que la ligne représente un nom d'une station
            stations_with_dictinary.append(unit[5:])
        else:
            break
    for i in xrange(len(stations_with_dictinary)):# Convertir la liste en dictionnaire
        my_stations_with_dict[i] = stations_with_dictinary[i]
    return my_stations_with_dict

```

Fonction main d'où le programme principal se déroule : le fonctionnement est expliqué avec les commentaires :

```python
if __name__ == "__main__":
    # Ouverture de fichier metro_paris.txt + déclaration des structures nécéssaires à l'implémentation
    dataset = open("metro_paris.txt", "r")
    list_source_destination = []
    list_keys = []
    graph_dijkstra = {}
    lists_sources_lists = []
    line_dataset = dataset.readlines()
    dictionary()
    # Déclarer une variable locale pour la fonction dataset_with_dict()
    dict_my_stations = dataset_with_dict()
    dijAlgo(lists_sources_lists)
    print('------------Algorithme de Dijkstra le plus court chemin------------')
    startPoint = int(input("\nEntrer N° de point de départ (-1 pour afficher la liste des stations disponibles): "))
    if startPoint==-1:
        print("\nStations disponibles :\n")
        print(short_path(dict_my_stations, dict_my_stations, "p"))
        startPoint = int(input("Entrer N° de point de départ: "))
    targetPoint = int(input("Entrer N° de point d'arrivée: "))
    # Début chronomètre pour calculer le temps d'exécution
    f0 = time()
    # Déclarer une variable locale pour calculer le chemin et sa distance
    traverse = dijkstra(graph_dijkstra, startPoint, targetPoint)[0]
    distance = dijkstra(graph_dijkstra, startPoint, targetPoint)[1]
    # Afficher la traversée de chemin
    print()
    print("Le chemin le plus court entre", startPoint, "et", targetPoint, " est:\n", traverse, "\n")
    # Afficher les stations parcourues dans le chemin en utilisant le dictionnaire des stations
    print("Stations parcourues :\n")
    print(short_path(traverse, dict_my_stations, "p"))
    # Afficher la distance parcourue
    print("Distance parcourue:",
          ("%.0f" % distance), '\n')
    # Temps d'exécution : Arret de chronomètre
    f1 = time()
    print("Temps d'exécution d'algorithme : %f" % (f1 - f0), "secondes")
```

Résultat d'exécution :

L'utilisateur entre les deux points d'entrée (départ) et d'arrivée, il peut aussi consulter la liste des stations en tapant -1 comme illustré ci-dessous :

```
------------Algorithme de Dijkstra le plus court chemin-------------

Entrer N° de point de départ (-1 pour afficher la liste des stations disponibles): -1

Stations disponibles :

[1] Station N°0: Abbesses
[2] Station N°1: Alexandre Dumas
[3] Station N°2: Alma Marceau
[4] Station N°3: Alésia
[5] Station N°4: Anatole France
[6] Station N°5: Anvers
[7] Station N°6: Argentine
[8] Station N°7: Arts et Métiers
[9] Station N°8: Arts et Métiers
[10] Station N°9: Assemblée Nationale
[11] Station N°10: Aubervilliers-Pantin, Quatre Chemins
[12] Station N°11: Avenue ƀmile Zola
[13] Station N°12: Avron
[14] Station N°13: Barbès Rochechouart
[15] Station N°14: Barbès Rochechouart
[16] Station N°15: Basilique de Saint-Denis
[17] Station N°16: Bastille
[18] Station N°17: Bastille
[19] Station N°18: Bastille
[20] Station N°19: Bel Air
[21] Station N°20: Belleville
```

6: TODO Terminal Python Console

Après la sélection des deux points, l'algorithme calcule le chemin le plus court, et le résultat d'affichage est sous forme d'une liste des stations parcourues, de distance de chemin et le temps d'exécution d'algorithme :

```
Entrer N° de point de départ: 48
Entrer N° de point d'arrivée: 356

Le chemin le plus court entre 48 et 356  est:
 [48, 182, 126, 272, 131, 153, 245, 168, 326, 202, 51, 50, 77, 356]

Stations parcourues :

[1] Station N°48: Carrefour Pleyel
[2] Station N°182: Mairie de Saint-Ouen
[3] Station N°126: Garibaldi
[4] Station N°272: Porte de Saint-Ouen
[5] Station N°131: Guy Môquet
[6] Station N°153: La Fourche
[7] Station N°245: Place de Clichy
[8] Station N°168: Liège
[9] Station N°326: Saint-Lazare
[10] Station N°202: Miromesnil
[11] Station N°51: Champs ƀlysées, Clémenceau
[12] Station N°50: Champs ƀlysées, Clémenceau
[13] Station N°77: Concorde
[14] Station N°356: Tuileries

Distance parcourue: 833

Temps d'exécution d'algorithme : 0.098006 secondes

Process finished with exit code 0
```

4: Run 6: TODO Terminal Python Console

Référence aidant à la réalisation de ce petit projet (droits d'auteur) :

https://github.com/BTajini/Paris-Metro-Project