Rapport du Projet Métro

Lounis Oucherif - Maxime Orlhac - Yanni Lassouani

29 avril 2022

Sujet du projet :

Dans ce projet nous avions à réaliser une application permettant de donner le plus court chemin entre deux stations du métro parisien. Pour cela nous disposions d'un fichier texte "metro.txt» contenant le nom des stations, chaque station étant lié à une station qui la succède ainsi que la temps de trajet entre ces stations.

Préparation :

La première étape obligatoire de ce projet a été la préparation. En effet, le fichier dont nous disposions était incomplet puisqu'il manquait les numéros de ligne de chaque station. Nous avons donc dû entrer à chaque stations le numéro de ligne associés sachant que certaines gares sont reliées par plusieurs lignes différentes. Pour pouvoir obtenir les données tel qu'on voulait nous avons choisis d'utiliser Excel pour récupérer toutes les données nécessaires de manière isolée, puis nous avons importé le fichier dans notre projet au format CSV. Le problème rencontré lors de cette partie était le codage en UTF 16. En fait, le problème des stations de métro parisienne est qu'il y a beaucoup de gare contenant tout type d'accent et de caractère spéciaux non reconnus par python nous avons donc passer tout notre fichier en UTF 8 pour régler le problème.

Description des Fonctions :

nom_sommet(dataframe): cette fonction nous permet de parcourir notre fichier passé en argument et renvoi la liste de tous les noms des sommets (stations). On utilise la fonction set() pour enlever les doublons présents dans le fichier.

num_sommet(dataframe) : cette fonction nous permet de parcourir notre fichier passé en argument et renvoi la liste de tous les numéros des sommets

num_stat(dataframe, nom_som):cette fonction nous permet de parcourir notre fichier passer en argument, le deuxième argument étant le nom d'un sommet et renvoi la liste de tous les numéros correspondant à ce sommet-là.

nom_stat(dataframe, num_sommet): cette fonction prend en argument notre fichier ainsi que le numéro d'un sommet et renvoi la station correspondant à ce numéro de sommet.

ligne(dataframe, num_som1, num_som2): cette fonction prend en argument notre fichier ainsi que deux numero de sommet et renvoi le numéro de ligne permettant de relier ces deux stations.

alentoure(dataframe, num_sommet): cette fonction prend en argument notre fichier ainsi que le numéro d'un sommet et renvoi une liste de liste contenant le numéro de sommet passé en argument ainsi que le numéro des sommets voisins et le temps de trajet en seconde entre les deux.

alentour_graphe(): cette fonction prend en argument notre fichier ainsi que le numéro d'un sommet et renvoi une liste de liste contenant le numéro de sommet passé en argument ainsi que le numéro des sommets voisins (ici pas le temps de trajet entre les deux sommets).

affichage(dataframe, dyco, u, longueur): cette fonction prend en argument notre fichier, dyco (dictionnaire ayant comme clé la longueur du plus court chemin et comme valeur une liste contenant le sommet de départ et le sommet d'arrivé), u (le sommet de fin) et longueur (longueur du plus court chemin). Elle renvoie un message dans le terminal grace a la fonction print() décrivant les étapes de notre trajet comme les lignes empreintés les correspondances effectués, la station de départ, la station d'arrivée, les stations de correspondance ainsi que le temps du trajet.###

graphe(dataframe): cette fonction nous permet de parcourir notre fichier passer en argument et renvoi un dictionnaire avec comme clé le numéro de sommet et comme valeur un autre dictionnaire avec comme clé le numéro de sommet atteignable depuis le premier sommet et comme valeur le temps de trajet entre les deux en seconde.

dijkstra(dataframe, dictio, s, fin): cette fonction prend en argument notre fichier, un dictionnaire implémenté par la fontion graphe(), s un sommet de départ et fin un sommet d'arrivée. Elle renvoie un dictionnaire avec comme clé la longueur du plus court chemin et comme valeur une liste contenant le sommet de départ et le sommet d'arrivé du chemin.

connexite(dico, s): cette fonction prend en argument un dictionnaire avec comme clé un sommet et comme valeur les sommets qu'on peut rejoindre a partir de ce sommet ainsi qu'un sommet s. Et renvoi un dictionnaire père décrivant le parcours en profondeur effectué par la fonction.

graphe_connexe(dataframe): cette fonction prend en argument notre fichier et renvoi un dictionnaire avec comme clé un sommet s et comme valeur les sommets joignables à partir du sommet s .

getEntry_dep(départ): cette fonction récupéré le choix de l'utilisateur sur le menu déroulant départ est le stock dans la variable

getEntry_arr(arriver): cette fonction récupéré le choix de l'utilisateur sur le menu déroulant arrive est le stock dans la variable

fonc(dataframe): la fonction a donc récupérer les stations de départ et d'arriver choisi via les variable TXT_DEPART et TXT_ARRIVE et trouve leurs numéros associés les boucles "for" exécute Dijkstra ce qui permet de récupérer la longueur entre 2 sommets, et la liste "liste_q" répertorie toutes les longueurs. Ensuite la variable "cle_q" récupérer le premier élément de la liste et si l'élément et déjà dans la liste de clé du dictionnaire "dic" on ajoute la clé a laquelle on donne la valeur de "q" sinon "dic" prend la clé de "q" en clé et en valeur la valeur de la clé de "q"

affichage(dataframe, dyco, u, longueur): cette fonction retourne l'itinéraire du chemin le plus court. On utilise une variable "TXT", à laquelle on ajoute le temps du trajet, dans une boucle si deux noms qui se suivent sont les mêmes on affiche une correspondance sinon On indique seulement "vous partez de {0} jusqu'à {1} avec la ligne {2} \n". Une fois sortie de la boucle on est arrivé. On a utilisé ".format" pour retourner les stations dans le texte.

get_entry(): cette fonction récupéré la station d'arrive et de départ choisie quand on appuie sur le bouton recherche La (liste_2) est la liste des sommets trié par ordre croissant grace au ".sort" Dans la partie placement on se sert de ".grid" pour positionner les widgets

Fonctionnement général du programme :

Le déroulement de notre programme consiste en demander à l'utilisateur de choisir sur l'interface graphique, avec les deux menus déroulant, le point de départ et le point d'arrivée. Une fois les deux gares choisis il suffit de cliquer a l'aide de la souris sur le bouton "chercher". Notre programme calcul ensuite à l'aide de l'algorithme de Dijkstra l'itinéraire le plus rapide entre les deux stations et l'affiche en détail gare par gare en précisant les lignes a suivre ainsi que le temps total du trajet en minute.

Connexité :

Dans la première partie de l'énoncé nous devions tester la connexité des gares dans le fichier metro.txt pour cela nous avons décidé de programmer une fonction graphe_connexe() permettant de renvoyer un dictionnaire avec comme clé un sommet s et comme valeur tous les sommets voisins de s. Depuis ce dictionnaire nous allons faire un parcours en profondeur avec la fonction parcours_en_profondeur() nous permettant de voir s'il est possible de parcourir tout le fichier depuis un sommet quelconque puis nous testons si la longueur du dictionnaire renvoyé par parcours_en_profondeur() est bien égale à la longueur du dictionnaire en sortie de graphe_connexe().

Dijkstra :

Choisir un algorithme du plus court chemin à utiliser dans notre projet a été un moment crucial de notre projet. En effet nous avons étudié différents algorithmes tout au long du semestre et plusieurs d'entre eux pouvaient sembler pertinents. Nous avons écarté l'algorithme de Ford-Bellman car il prend en compte les poids des arêtes négatives or dans ce projet nous n'en avions pas besoin. L'algorithme de Flloyd-Warshall quant à lui était très intéressant mais après recherche nous avons trouvé que la complexité de cet algorithme était supérieure à celle de l'algorithme de Dijkstra. Le premier problème que nous avons rencontré avec l'algorithme de Dijkstra était le fait que des sommets avec des numéros différents peuvent avoir le même nom. Nous avons donc dû utiliser num_stat() pour lister les numéros de stations correspondant aux même gares pour pouvoir appliquer notre algorithme. Pour pouvoir calculer le plus court chemin nous avons dû utiliser un dictionnaire avec comme clé la longueur u de trajet entre les deux gares données et comme valeur une liste de liste avec les sommets de départ et d'arrivée.

Interface graphique :

En troisième partie de ce projet nous devions réaliser une interface graphique nous avons donc décidé d'utiliser Tkinter. Nous avons répertorié toutes les stations dans deux menu déroulant (Option Menu) afin que l'utilisateur puisse choisir la station de départ et d'arriver qu'il veut. Le bouton recherche permet de récupérer les deux stations choisies et de les envoyer dans la fonction "fonc" qui effectue Dijkstra afin de trouver le chemin le plus court.

```python
#projet_metro.py

import csv

import tkinter as tk
from tkinter import *


def nom_sommet(dataframe):
    #renvoie la liste de tous les nom des sommets

    liste_nom = []
    for i in range(len(dataframe)):
        if 'nom_sommet' in dataframe[i]:
            liste_nom.append(dataframe[i]['nom_sommet'])
        liste_nom = list(set(liste_nom))
    return liste_nom

def num_sommet(dataframe):
    #renvoie la liste de tous les numeros des sommets

    liste_num = []
    for i in range(len(dataframe)):
        if 'num_sommet' in dataframe[i]:
            liste_num.append(dataframe[i]['num_sommet'])
    return liste_num


def num_stat(dataframe, nom_som):
    #d'apres le nom de la station il sort tous les numero de la station
    liste_num_stat = []
    for i in range(len(dataframe)):
        if 'nom_sommet' in dataframe[i]:
            if dataframe[i]['nom_sommet'] == nom_som:
                liste_num_stat.append(dataframe[i]['num_sommet'])
    return liste_num_stat

def nom_stat(dataframe, num_som):
    #d'apres le numero de la station il sort le nom de la station
    if 'nom_sommet' in dataframe[num_som]:
            nom_station = dataframe[num_som]['nom_sommet']
    return nom_station

def ligne(dataframe, num_som1,num_som2):
    #renvoie la ligne a suivre pour arriver au sommet 2 a partir du sommet 1

    l = ""
```

```
    for i in range(len(dataframe)):
        if dataframe[i]['num_sommet1'] == num_som1 and dataframe[i]['num_sommet2'] == num_som2 :
            l = dataframe[num_som1]['ligne']
    if l == "":
        for i in range(len(dataframe)):
            if dataframe[i]['num_sommet1'] == num_som2 and dataframe[i]['num_sommet2'] == num_som1:
                l = dataframe[num_som2]['ligne']

    return l

def alentoure(dataframe, num_sommet):
    #renvoie une liste de liste contenant le numero du sommet ,num_sommet1 , num_sommet2 et
temps_en_s  (les detaile de l'arrete associer au sommet)

    liste_al = []
    for i in range(len(dataframe)):
        if dataframe[i]['num_sommet1'] == num_sommet or dataframe[i]['num_sommet2'] == num_sommet :
            liste_al.append([num_sommet ,dataframe[i]['num_sommet1'] ,dataframe[i]['num_sommet2']
,dataframe[i]['temps_en_s']])
    return liste_al

def alentoure_graph(dataframe, num_sommet):
    #renvoie une liste de liste contenant le numero du sommet ,num_sommet1 , num_sommet2 et
temps_en_s  (les detaile de l'arrete associer au sommet)

    liste_al = []
    for i in range(len(dataframe)):
        if dataframe[i]['num_sommet1'] == num_sommet or dataframe[i]['num_sommet2'] == num_sommet :
            liste_al.append([num_sommet ,dataframe[i]['num_sommet1'] ,dataframe[i]['num_sommet2']])
    return liste_al

def connexite(dico,s) :
    couleur=dict()
    for station in dico :couleur[station]='blanc'
    pere=dict()
    pere[s]=None
    couleur[s]='gris'
    pile=[s]
    while pile :
        u=pile[-1]
        R=[y for y in dico[u] if couleur[y]=='blanc']
        if R :
            station=R[0]
            couleur[station]='gris'
            pere[station]=u
            pile.append(station)
```

```
        else :
            pile.pop()
            couleur[u]='noir'
    if len(pere) == len(dico) :
        print (True)

def graphe_conex(dataframe):
    #renvoie un dictionaire de clé (numero de sommet) et comme valeur (un dictionaire de clé (le numero des sommets ateniable) et comme valeur( le temps entre les 2 en s))

    liste_num1 = []
    liste_stat = []
    dic = {}

    liste_nom = nom_sommet(dataframe)
    liste_nom = list(set(liste_nom))
    for nom in liste_nom:
        liste_num = num_stat(dataframe, nom)
        for num in liste_num :
            liste_num1.append(num)
    for num in liste_num1:
        liste = alentoure_graph(dataframe, num)
        for stat in liste :
            liste_stat.append(stat)
    for i in range(len(liste_stat)):
        Valeur_base = []
        key = liste_stat[i][0]

        if liste_stat[i][1] == key:
            Valeur = liste_stat[i][2]
        elif liste_stat[i][2] == key:
            Valeur = liste_stat[i][1]

        Valeur_base.append(Valeur)
        nouveau = {key:Valeur_base}

        if key in dic.keys():
            dic[key].append(Valeur)
        else:
            dic.update(nouveau)

    connexite(dic,1)

def graphe(dataframe):
    #renvoie un dictionaire de clé (numero de sommet) et comme valeur (un dictionaire de clé (le numero des sommets ateniable) et comme valeur( le temps entre les 2 en s))
```

```
    liste_num1 = []
    liste_stat = []
    dic = {}

    liste_nom = nom_sommet(dataframe)
    liste_nom = list(set(liste_nom))
    for nom in liste_nom:
        liste_num = num_stat(dataframe, nom)
        for num in liste_num :
            liste_num1.append(num)
    for num in liste_num1:
        liste = alentoure(dataframe, num)
        for stat in liste :
            liste_stat.append(stat)
    for i in range(len(liste_stat)):
        Valeur_base = {}
        key = liste_stat[i][0]

        if liste_stat[i][1] == key:
            Valeur = {liste_stat[i][2] : liste_stat[i][3]}
        elif liste_stat[i][2] == key:
            Valeur = {liste_stat[i][1] : liste_stat[i][3]}

        Valeur_base.update(Valeur)
        nouveau = {key:Valeur_base}

        if key in dic.keys():
            dic[key].update(Valeur)
        else:
            dic.update(nouveau)


    return dic


def dijsktra(dataframe ,dictio , s, fin):

    # trouve le plus court chemin entre 2 sommets
    # s:sommet de debut , fin:le sommet de arriver
    # et renvoie dic: un dictionaire de clé (la longueur du plus court chemin) et de valeur (une liste dans la
quelle y a le sommet de depart et le sommet d'arriver)
    # s_connu : la liste des sommet traverser
    # u : le sommet d'arriver

    # initialisation
    dic = {}
    infini = 10000000000000
```

```
    s_connu = {s :[0 , [s]]}
    s_inconnu = {k : [infini , ''] for k in dictio if k != s}
    for suivant in dictio[s]:
        s_inconnu[suivant] = [dictio[s][suivant],s]
    
    # recherche

    while s_inconnu and any(s_inconnu[k][0] < infini for k in s_inconnu):
        u = min(s_inconnu,key = s_inconnu.get)
        longueur_u, precedent_u = s_inconnu[u]
        for v in dictio[u]:
            if v in s_inconnu:
                d = longueur_u + dictio[u][v]
                if d < s_inconnu[v][0]:
                    s_inconnu[v] = [d, u]
        s_connu[u] = [longueur_u, s_connu[precedent_u][1] + [u]]
        del s_inconnu[u]
        
        if u == fin :
            dic = {longueur_u : [s_connu[u][1][0],u]}
            break
        
    return dic , s_connu, u


process_list = []
_path = "./metro1.csv"
with open(_path, newline='') as csv_file:
    reader = csv.reader(csv_file, delimiter=';')
    for row in reader:
        num_somm = lambda a : int(a) if (a != '') else None
        av ={
            'sommet': row[0],
            'num_sommet': num_somm(row[1]),
            'nom_sommet': row[2],
            'ligne': row[3],
            'arete': row[4],
            'num_sommet1': int(row[5]),
            'num_sommet2': int(row[6]),
            'temps_en_s' : int(row[7]),
        }
        ap = {'arete': row[4],'num_sommet1': int(row[5]),'num_sommet2': int(row[6]),'temps_en_s' :
int(row[7])}
        if row[0] != '' :
            process_list.append(av)
        else:
            process_list.append(ap)
```

```
graphe_conex(process_list)

### interface

TXT_DEPART = None
TXT_ARRIVE = None

root = tk.Tk()
root.title(" metro itineraire ")
canvas = tk.Canvas(root, bg="white", width=1000, height=800)

label_depars = tk.Label(root, text="Gare de départ :", fg="Black", font=("Helvetica", 20))
label_arrive = tk.Label(root, text="Gare d'arrivé :", fg="Black", font=("Helvetica", 20))

def getEntry_dep(depart):
    global TXT_DEPART
    depart = v_1.get()
    TXT_DEPART = depart

def getEntry_arr(arriver):
    global TXT_ARRIVE
    arriver = v_2.get()
    TXT_ARRIVE = arriver

def fonc(dataframe ):
    global TXT_DEPART , TXT_ARRIVE

    dep = TXT_DEPART
    arr =TXT_ARRIVE
    if dep == arr :
        pass
    else:
        dic = {}
        debut = num_stat(process_list, dep)
        arriver = num_stat(process_list, arr)
        dico = graphe(process_list)
        for i in debut:
            for j in arriver:
                dic_temp, sans_interer_1, sans_interer_2 = dijsktra(process_list,dico , i, j)
                liste_dic_temp = list(dic_temp.keys())
                cle_dic_temp = liste_dic_temp[0]
                if cle_dic_temp in list(dic.keys()):
                    dic[cle_dic_temp].append(dic_temp.get(cle_dic_temp))

                else:
```

```
                dic.update(dic_temp)

        liste_18 = list(dic.keys())

        temps_min = min(liste_18)
        dysc ,dic_min ,u = dijsktra(process_list,dico , dic[temps_min][0], dic[temps_min][1])
    return affichage(process_list ,dic_min , u ,temps_min)


def affichage(dataframe ,dyco, u, longueur):
    # elle affiche le trajet pour l'utilisateur
    # en entrer :
    # dyco : un dictionaire de clé (la longueur du plus court chemin) et de valeur (une liste dans la quelle y a
le sommet de depart et le sommet d'arriver)
    # u : sommet de fin
    # longueur : la longueur du plus cour chemin

    TXT =""
    TXT += "le temps de trajet est de : {} min \n \n ".format(round(longueur/60, 2))
    for w in range(len(dyco[u][1])-1):
        lign = ligne(dataframe, dyco[u][1][w],dyco[u][1][w+1])
        if nom_stat(dataframe,dyco[u][1][w]) == nom_stat(dataframe,dyco[u][1][w+1]):
            TXT +="\n corespendance de la ligne {0}  avec la ligne {1} \n
\n".format(lign,dataframe[dyco[u][1][w+1]]['ligne'])
        else:
            TXT +="vous partez de {0} jusqu'a {1} avec la ligne {2} \n".format(nom_stat(dataframe,
dyco[u][1][w]), nom_stat(dataframe,dyco[u][1][w+1]), lign)
    TXT += "\n vous etes ariver a {} \n".format(nom_stat(dataframe,u))
    label_resultat.config(text = TXT)


def get_entry():
    global TXT_DEPART, TXT_ARRIVE
    TXT_DEPART = v_1.get()
    TXT_ARRIVE = v_2.get()
    fonc(process_list)



liste_2 = nom_sommet(process_list)
liste_2.sort(key=lambda x: x[0])

boutton_recherche = tk.Button(root, height=1, width=10, text="Chercher", command=get_entry)
boutton_recherche.grid(row=7, column=60)
label_resultat = tk.Label(root, height=10, width=10, text= 'itinéraire')


#
v_1 = StringVar()
v_1.set("Gare de depart")
```

```
deroule_dep = OptionMenu(root, v_1, *liste_2, command=getEntry_dep)
#
v_2 = StringVar()
v_2.set("Gare d'arrivée")
deroule_arr = OptionMenu(root, v_2, *liste_2, command=getEntry_arr)
#

################### PLACEMENT ########################
label_resultat.grid(row= 15, rowspan=100, column=0, columnspan=200)
label_resultat.place(x=100, y=200, width=800, height=750)
deroule_dep.grid(row = 5, column = 20)
deroule_arr.grid(row = 7, column = 20)

label_depars.grid(row=5, column=2)
label_arrive.grid(row=7, column=2)
boutton_recherche.grid(row=6, column=60)

canvas.grid(row=0, rowspan=200, column=0, columnspan=200)

root.mainloop()
```