TP4

Le but de ce TP est d'appliquer la théorie des flots dans les graphes à des problèmes concrets.



Le fait de tester toutes vos créations n'est plus explicitement demandé car cela doit **encore & toujours** être fait.

Les graphes seront représentés "mathématiquement" par des listes d'adjacence, & plus précisément/concrètement par des dictionnaires de dictionnaires de dictionnaires. Par exemple :

```
CAPACITE = "capacite"
FLUX = "flux"

the_graph = {1: {2: {CAPACITE: 5, FLUX: 2}, 3:{CAPACITE: 6, FLUX: 1}},
             2: {3: {CAPACITE: 7, FLUX: 1}, 4: {CAPACITE: 8, FLUX: 1}},
             3: {5: {CAPACITE: 4, FLUX: 2}},
             4: {6: {CAPACITE: 3, FLUX: 2}},
             5: {4: {CAPACITE: 6, FLUX: 1}, 6: {CAPACITE: 8, FLUX: 1}},
             6: {}
             }
```

Dessinez le graphe ci-dessus.

Implémentez l'Algorithme de Ford & Fulkerson. On pourra utiliser les fonctions suivantes :

```
import math

INFINI = math.inf
CAPACITE = "capacite"
FLUX = "flux"


def mise_a_zero_flot(graph):
    for sommet in graph:
        for voisin in graph[sommet]:
            graph[sommet][voisin][FLUX] = 0


def construction_graphe_ecart(graphe_antisymetrique):
    graphe_ecart = {}
    for sommet in graphe_antisymetrique:
        graphe_ecart[sommet] = {}
    for sommet in graphe_antisymetrique:
        for voisin in graphe_antisymetrique[sommet]:
            capacite = graphe_antisymetrique[sommet][voisin][CAPACITE]
            flux = graphe_antisymetrique[sommet][voisin][FLUX]
            if capacite > flux:
```

```
                graphe_ecart[sommet][voisin] = capacite - flux
            if flux > 0:
                graphe_ecart[voisin][sommet] = flux
    return graphe_ecart


def construction_chemin_via_dfs(graphe_ecart, source, puits):
    sommets_marques = {}
    antecedents = {}
    pile = [source]
    for sommet in graphe_ecart:
        antecedents[sommet] = None
    while pile != []:
        sommet = pile.pop()
        if sommet not in sommets_marques:
            sommets_marques[sommet] = True
            for voisin in graphe_ecart[sommet]:
                if voisin not in sommets_marques:
                    antecedents[voisin] = sommet
                    pile.append(voisin)
        if sommet == puits:
            return construction_effective_chemin(sommet, antecedents)


def construction_effective_chemin(sommet, antecedents):
    chemin = []
    while sommet is not None:
        chemin.append(sommet)
        sommet = antecedents[sommet]
    chemin.reverse()
    return  chemin


def min_sur_chemin(chemin, graphe_ecart):
    minimum = INFINI
    for i in range(len(chemin) - 1):
        courant = chemin[i]
        suivant = chemin[i + 1]
        if graphe_ecart[courant][suivant] < minimum:
            minimum = graphe_ecart[courant][suivant]
    return minimum


def ajout_flot(valeur, chemin, graphe_antisymetrique):
    for i in range(len(chemin) - 1):
        courant = chemin[i]
        suivant = chemin[i + 1]
        if suivant in graphe_antisymetrique[courant]:
            graphe_antisymetrique[courant][suivant][FLUX] += valeur
        else:
            graphe_antisymetrique[suivant][courant][FLUX] -= valeur
```

Dans la fonction construction_graphe_ecart, le résultat graphe_ecart est-il représenté comme the_graph. Pourquoi ?

Dans un deuxième temps, on l'appliquera au problème du mariage avec les données suivantes :

```
CLEOPATRE = "Cleopatre"
IPHIGENIE = "Iphigenie"
JULIETTE = "Juliette"
FANNY = "Fanny"
CHIMENE = "Chimene"

ACHILLE = "Achille"
CESAR = "Cesar"
RODRIGUE = "Rodrigue"
ROMEO = "Romeo"
MARIUS = "Marius"

LES_COUPLES = [(CLEOPATRE, ACHILLE), (CLEOPATRE, CESAR), (CLEOPATRE, ROMEO),
               (IPHIGENIE, ACHILLE), (JULIETTE, CESAR), (JULIETTE,
RODRIGUE), (JULIETTE, ROMEO),
               (FANNY, CESAR), (FANNY, MARIUS), (CHIMENE, RODRIGUE),
(CHIMENE, ROMEO)]
```

On résoudra après le problème de "la bataille de la Marne", en supposant que,

- Au temps 0, on a autant de taxis que nécessaire dans la ville 14 ("Paris").
- Au temps 50, il faut qu'il y en ait le plus possible dans la ville 0 ("La Marne").
- Chaque ville (intermédiaire) a 10 places de parking (Paris en a autant que nécessaire).
- Les routes sont données dans le tableau suivant :

```
LES_ROUTES = [{'depart': 0, 'capacite': 2, 'arrivee': 1, 'longueur': 5},
{'depart': 0, 'capacite': 3, 'arrivee': 2, 'longueur': 3},
{'depart': 0, 'capacite': 6, 'arrivee': 1, 'longueur': 6},
{'depart': 1, 'capacite': 3, 'arrivee': 2, 'longueur': 6},
{'depart': 1, 'capacite': 7, 'arrivee': 4, 'longueur': 7},
{'depart': 1, 'capacite': 6, 'arrivee': 2, 'longueur': 7},
{'depart': 2, 'capacite': 7, 'arrivee': 4, 'longueur': 4},
{'depart': 2, 'capacite': 6, 'arrivee': 4, 'longueur': 8},
{'depart': 2, 'capacite': 2, 'arrivee': 5, 'longueur': 5},
{'depart': 3, 'capacite': 7, 'arrivee': 5, 'longueur': 5},
{'depart': 3, 'capacite': 3, 'arrivee': 5, 'longueur': 4},
{'depart': 3, 'capacite': 7, 'arrivee': 6, 'longueur': 4},
{'depart': 4, 'capacite': 7, 'arrivee': 5, 'longueur': 8},
{'depart': 4, 'capacite': 8, 'arrivee': 7, 'longueur': 3},
{'depart': 4, 'capacite': 4, 'arrivee': 7, 'longueur': 3},
{'depart': 5, 'capacite': 8, 'arrivee': 6, 'longueur': 8},
{'depart': 5, 'capacite': 4, 'arrivee': 7, 'longueur': 7},
{'depart': 5, 'capacite': 4, 'arrivee': 6, 'longueur': 5},
{'depart': 6, 'capacite': 5, 'arrivee': 9, 'longueur': 3},
{'depart': 6, 'capacite': 5, 'arrivee': 7, 'longueur': 3},
{'depart': 6, 'capacite': 7, 'arrivee': 8, 'longueur': 3},
```

```
{'depart': 7, 'capacite': 2, 'arrivee': 8, 'longueur': 8},
{'depart': 7, 'capacite': 2, 'arrivee': 10, 'longueur': 5},
{'depart': 7, 'capacite': 7, 'arrivee': 10, 'longueur': 4},
{'depart': 8, 'capacite': 3, 'arrivee': 11, 'longueur': 4},
{'depart': 8, 'capacite': 5, 'arrivee': 9, 'longueur': 3},
{'depart': 8, 'capacite': 8, 'arrivee': 10, 'longueur': 3},
{'depart': 9, 'capacite': 1, 'arrivee': 11, 'longueur': 8},
{'depart': 9, 'capacite': 4, 'arrivee': 12, 'longueur': 3},
{'depart': 9, 'capacite': 1, 'arrivee': 11, 'longueur': 7},
{'depart': 10, 'capacite': 5, 'arrivee': 12, 'longueur': 7},
{'depart': 10, 'capacite': 3, 'arrivee': 13, 'longueur': 3},
{'depart': 10, 'capacite': 7, 'arrivee': 11, 'longueur': 4},
{'depart': 11, 'capacite': 4, 'arrivee': 14, 'longueur': 8},
{'depart': 11, 'capacite': 2, 'arrivee': 12, 'longueur': 3},
{'depart': 11, 'capacite': 6, 'arrivee': 14, 'longueur': 6},
{'depart': 12, 'capacite': 5, 'arrivee': 13, 'longueur': 6},
{'depart': 12, 'capacite': 2, 'arrivee': 14, 'longueur': 7},
{'depart': 12, 'capacite': 2, 'arrivee': 14, 'longueur': 6},
{'depart': 13, 'capacite': 2, 'arrivee': 14, 'longueur': 5},
{'depart': 13, 'capacite': 2, 'arrivee': 14, 'longueur': 3},
{'depart': 13, 'capacite': 8, 'arrivee': 14, 'longueur': 6}]
```



Les routes sont à double sens (il ne faut pas se fier aux appellations "départ" & "arrivée").

On pourra commencer par une instance plus petite, par exemple en ne considérant que les villes jusqu'à 9, avec un borne max pour le temps de 15, c'est à dire avec :

```
LES_ROUTES = [{'depart': 0, 'capacite': 2, 'arrivee': 1, 'longueur': 5},
{'depart': 0, 'capacite': 3, 'arrivee': 2, 'longueur': 3},
{'depart': 0, 'capacite': 6, 'arrivee': 1, 'longueur': 6},
{'depart': 1, 'capacite': 3, 'arrivee': 2, 'longueur': 6},
{'depart': 1, 'capacite': 7, 'arrivee': 4, 'longueur': 7},
{'depart': 1, 'capacite': 6, 'arrivee': 2, 'longueur': 7},
{'depart': 2, 'capacite': 7, 'arrivee': 4, 'longueur': 4},
{'depart': 2, 'capacite': 6, 'arrivee': 4, 'longueur': 8},
{'depart': 2, 'capacite': 2, 'arrivee': 5, 'longueur': 5},
{'depart': 3, 'capacite': 7, 'arrivee': 5, 'longueur': 5},
{'depart': 3, 'capacite': 3, 'arrivee': 5, 'longueur': 4},
{'depart': 3, 'capacite': 7, 'arrivee': 6, 'longueur': 4},
{'depart': 4, 'capacite': 7, 'arrivee': 5, 'longueur': 8},
{'depart': 4, 'capacite': 8, 'arrivee': 7, 'longueur': 3},
{'depart': 4, 'capacite': 4, 'arrivee': 7, 'longueur': 3},
{'depart': 5, 'capacite': 8, 'arrivee': 6, 'longueur': 8},
{'depart': 5, 'capacite': 4, 'arrivee': 7, 'longueur': 7},
{'depart': 5, 'capacite': 4, 'arrivee': 6, 'longueur': 5},
{'depart': 6, 'capacite': 5, 'arrivee': 9, 'longueur': 3},
{'depart': 6, 'capacite': 5, 'arrivee': 7, 'longueur': 3},
{'depart': 6, 'capacite': 7, 'arrivee': 8, 'longueur': 3},
```

```
{'depart': 7, 'capacite': 2, 'arrivee': 8, 'longueur': 8},
{'depart': 7, 'capacite': 2, 'arrivee': 10, 'longueur': 5},
{'depart': 7, 'capacite': 7, 'arrivee': 10, 'longueur': 4},
{'depart': 8, 'capacite': 3, 'arrivee': 11, 'longueur': 4},
{'depart': 8, 'capacite': 5, 'arrivee': 9, 'longueur': 3}]
```

voire même plus petit.

From:
https://wiki.centrale-med.fr/informatique/ - **WiKi informatique**

Permanent link:
https://wiki.centrale-med.fr/informatique/tc_info:tp6

Last update: **2019/09/28 16:34**