Présentation Générale

Ce TP a été préparé/testé en Juin-Juillet 2018 par Dan ZHU lors de son stage de deuxième année à l'ECM.

Contexte : De nombreux services (Google Map, Via Michelin, Mappy,...) vous donnent, en quelques secondes, le plus court chemin pour aller d'un point A à un point B au sein d'une carte contenant des millions de points. De plus, cet exploit est souvent réalisé sur des "machines" pesant à peine une centaine de grammes.

But : Le but de ce TP est de réaliser que ceci est infaisable par les méthodes "classiques" (i.e. vues en cours), & d'étudier des pistes que cela soit réalisable avec ces contraintes de temps & de puissance.

Travail à réaliser : Pour cela, on résoudra ce problème sur différentes cartes (contenant 100, 1000,... 30000 points), & on extrapolera les résultats aux cartes réelles.

Objectif : À la fin de la séance, l'élève aura une vision concrète des problèmes posés par la manipulation & le traitement de données de grande taille.

Travail à Réaliser



Il est "demandé" d'utiliser la pep8. En particulier, les noms (de variables, fonctions, ...) doivent être **explicites**. De même, on mettra des espaces autour des opérateurs : on écrira

```
i = j + 1
```

& non pas

```
i=j+1
```

ni

```
i = j+1
```

ni

```
i =i+1
```

ni ...

Les données à traiter sont là. Les fichiers `base_x.csv` contiennent, au format csv, les *x* plus grandes villes de France avec (entre autres) latitudes & longitudes. Par exemple, la ligne

3503,14515,Port-en-Bessin-Huppain,49.350 ,-0.750 ,2308

indique que, avec 2308 habitants, Port-en-Bessin-Huppain (code postal 14515) est la 3503me ville de France & que ses coordonnées sont : 49.350°Nord & 0.750°Ouest

Le code

```
NAME = 'name'
LATITUDE = 'latitude'
LONGITUDE = 'longitude'
POPULATION = 'population'

def file_to_node_list(file_name, r):
    node_list = []
    flowfile = open(file_name)
    for line in flowfile:
        the_data = line.split(',')
        node_list.append({NAME: the_data[2],
                          LATITUDE: float(the_data[3]),
                          LONGITUDE: float(the_data[4]),
                          POPULATION: int(the_data[5])})
    flowfile.close()
    return node_list
```

transforme ces fichiers en une liste de dictionnaires, qui correspondent chacun à une ville.

Première chose à faire

À partir de ces fichiers, construire des graphes comme ceux représentés dans ce répertoire. Le fichier `Villes-x_R-d.pdf` correspond au graphe dont les sommets sont les *x* plus grandes villes de France, deux villes étant reliées (par une arête) si elles sont à moins de *d* kilomètres l'une de l'autre.



Il est **fortement conseillé** d'utiliser le même format que pour ce graphe, construit à partir de quelques villes parmi les 500 plus grandes & représenté ici.



On remarquera que ce graphe est représenté par une liste d'adjacence, elle-même implémentée avec un dictionnaire, dont les clés sont des chaines de caractères (les noms des villes). Pour des raisons de cohérence, il est **conseillé**, pour les autres structures de données que l'on implémentera, d'utiliser le même type de dictionnaires.



Il n'est pas obligatoire d'utiliser la norme L2 (mais c'est mieux).



Dès cette étape (& pour toutes les autres), on représentera graphiquement les résultats obtenus. Cela permettra de "vérifier" que les résultats en question sont corrects.



Le dessin est juste une représentation. On ne peut rien faire à partir de lui. On séparera bien cette partie graphique du reste du programme, & celle-ci s'exécutera à la fin du programme.

Pour cette représentation graphique, on utilisera la librairie `matplotlib`.

par exemple, si `latitudes` & `longitudes` sont des listes (de même taille) de nombres (latitudes = [x1, ..., xn] longitudes = [y1, ... yn]), le code

```
import matplotlib
from matplotlib import pyplot

pyplot.scatter(latitudes, longitudes, s = 40, c = 'red', m = 'o')
pyplot.show()
```

trace tous les points de coordonnées `(xi, yi)`, chaque point étant représenté par un disque rouge de "taille" 40.

Pour rajouter un trait noir entre deux points de coordonnées [x1, y1] & [x2, y2], on utilisera

```
pyplot.plot([x1, x2], [y1, y2])
```



Attention, c'est

```
pyplot.plot([x1, x2], [y1, y2])
```

& non pas

```
pyplot.plot([x1, y1], [x2, y2])
```

Deuxième chose à faire

Programmez les algorithme de Dijkstra & de Roy-Floyd-Warshall & essayez-les sur les différents graphes construits.



Il est recommandé de conserver les résultats obtenus par l'algorithme de Roy-Floyd-Warshall, surtout pour les graphes de grande taille.



Pour ceux qui sont pressés, voici le code de l'algorithme de Dijkstra & de la première étape de l'algorithme de Roy-Floyd_Warshall pour des graphes donnés dans le format conseillé plus haut.

Déterminez si ces algorithmes sont utilisables pour un vrai google-map.

Pour mesurer le temps d'exécution d'un programme, on utilisera :

```
import time

beginning = time.clock()

code_à_tester

the_end = time.clock()
print(the_end - beginning)
```

Troisième chose à faire

Le but est maintenant d'écrire un programme capable de traiter, en un temps & espace mémoire raisonnable, des données de très grande taille.

L'idée de base est la même que pour (la deuxième étape de) l'algorithme de Roy-Floyd-Warshall : le plus court chemin pour aller d'une ville *x* à une ville *y* est d'abord donné par un intermédiaire *i* (appelé *hub*), puis on cherche un plus court chemin de *x* à i_0, ...

Dans l'algorithme de Roy-Floyd-Warshall, si on a *n* villes, l'ensemble des hubs est une matrice *n* x *n*. Ce que l'on va faire maintenant, c'est construire un ensemble de hubs de taille (au total) *O(n)*.

Sur un graphe de 'petite taille' (parmi ceux qui sont donnés), pour un ensemble "significatif" de villes, tracer (simultanément) tous les chemins de Marseille (par exemple) à ces villes. Qu'observez-vous ?

Tracer ensuite (toujours simultanément) les deux premiers tiers de ces chemins. Qu'observez-vous ?



Spoiler : Vous pouvez regarder ici

Est-ce que cela s'observe aussi pour les graphes de grande taille (parmi ceux qui sont donnés) ? Pour les vrais réseaux routiers ? Pourquoi ?

On 'rapprochera' les graphes donnés d'un vrai réseau en les fusionnant avec ce graphe, que l'on interprétera comme un réseau d'autoroutes.

On prendra comme ensemble des hubs associés à Marseille (par exemple) la plus grande ville dans le deuxième tiers de chacun de ces chemins.

Montrez qu'on peut alors obtenir, y compris sur des données de grande taille, le plus court chemin entre deux villes.

Quatrième chose à faire

Adaptez votre algorithme pour qu'il donne des chemins alternatifs, qu'il indique les parties d'autoroutes & de routes 'normales', ...

Cinquième chose à faire

Quels sont les problèmes résiduels ? Comment les traiter ?

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

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

Last update: **2018/11/19 14:38**