Initiation à l'informatique

TD noté, rattrapage 2010

Le programme construit au fur et à mesure des questions devra être imprimé à la fin du TD et rendu. A chaque question correspond une fonction à écrire. Le choix des paramètres et du résultat est laissé libre bien que l'énoncé propose des suggestions. Tout document autorisé.

On souhaite au cours de cette séance traverser quelques villes de France le plus rapidement possible. Il est plutôt évident que le chemin illustré par le graphique 1 n'est pas le plus rapide. On cherchera à implémenter quelques astuces qui permettront de construire un chemin "acceptable".

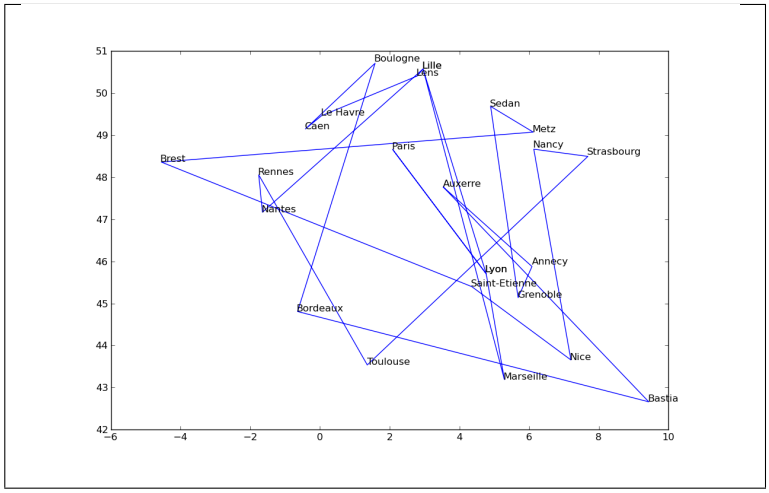

Figure 1 : *Le tour de France. On veut trouver le plus court chemin passant par toutes les villes. Ce problème est aussi connu sous le nom du problème du voyageur de commerce.*

1) La première étape consiste à représenter la liste des villes et de leurs coordonnées via des listes *Python* :

Auxerre	3,537	47,767
Bastia	9,434	42,662
Bordeaux	-0,643	44,808
Boulogne	1,580	50,709
...	...	...
Grenoble	5,684	45,139
Annecy	6,082	45,878

Elles sont accessibles depuis l'adresse http://www.xavierdupre.fr/enseignement/examen_python/villes.txt. Il s'agit de créer une fonction qui récupère ces informations, soit depuis un fichier texte, soit elles peuvent être directement insérées dans le programme sous la forme d'une seule chaîne de caractères.

L'objectif est ensuite d'obtenir une matrice avec le nom de chaque ville en première colonne, l'abscisse et l'ordonnée en seconde et troisième colonnes. Les fonctions `strip`, `replace`, `split` pourraient vous être utiles.

```
[['Auxerre', 3.537309885, 47.767200469999999],
 ['Bastia', 9.4343004229999998, 42.661758419999998],
 ...
```

L'abscisse et l'ordonnée doivent être des réels (`float`) afin d'être facilement manipulées par la suite.

(3 points) *Insérer directement dans le programme la matrice dans sa forme finale ne rapporte pas de point.*

```
def get_tour () :
    stour = """Auxerre        3,537309885        47,76720047
Bastia        9,434300423        42,66175842
Bordeaux        -0,643329978        44,80820084"""
    ...
    return tour
```

2) Ecrire une fonction `distance` qui calcule la distance euclidienne entre deux villes. On supposera que la distance à vol d'oiseau est une approximation acceptable de la distance entre deux villes. (2 points)

```
def distance (tour, i,j) :
    ...
    return d
```

3) Ecrire une fonction `longueur_tour` qui retourne la longueur d'un circuit. Un circuit est décrit par la matrice de la première question : on parcourt les villes les unes à la suite des autres dans l'ordre où elles apparaissent dans la matrice. On commence à Auxerre, on va à Bastia puis Bordeaux pour terminer à Annecy et revenir à Auxerre. (2 points)

```
def longueur_tour (tour) :
    ...
    return d
```

4) Il est facile de vérifier visuellement si un chemin est absurde comme celui de la figure 1. La fonction suivante vous aidera à tracer ce chemin. Il faut la compléter. (2 points)

```
import pylab
def graph (tour) :
    x = [ t[1] for t in tour ]
    y = [ t[2] for t in tour ]
    ....
    ....
    pylab.plot (x,y)
    for ville,x,y in tour :
        pylab.text (x,y,ville)
    pylab.show ()
```

5) La première idée pour construire un chemin plus court est de partir du chemin initial. On échange deux villes choisies aléatoirement puis on calcule la distance du nouveau chemin. Si elle est plus courte, on conserve la modification. Si elle est plus longue, on annule cette modification. On continue tant qu'il n'est plus possible d'améliorer la distance. (4 points)

```
def permutation (tour) :
```

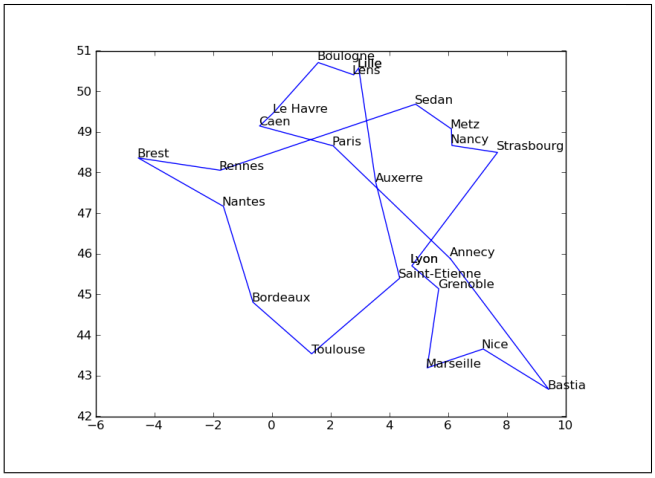

Figure 2 : *Le tour de France lorsque des chemins se croisent. Ce chemin n'est pas optimal de manière évidente.*

6) Le résultat n'est pas parfait. Parfois, les chemins se croisent comme sur la figure 2. Pour cela on va essayer de retourner une partie du chemin. Il s'agit ici de construire une fonction `retourne` qui retourne un chemin entre deux villes i et j. (3 points)

Avant l'exécution de cette fonction, on a :

$$v_1 \to v_2 \to \ldots \to v_{i-1} \to v_i \to v_{i+1} \to \ldots \to v_{j-1} \to v_j \to v_{j+1} \to \ldots$$

Après l'exécution de cette fonction, on a :

$$v_1 \to v_2 \to \ldots \to v_{i-1} \quad v_i \leftarrow v_{i+1} \leftarrow \ldots \leftarrow v_{j-1} \leftarrow v_j \quad v_{j+1} \to \ldots$$

Ou encore :

$$v_1 \to v_2 \to \ldots \to v_{i-1} \to v_j \to v_{j-1} \to \ldots \to v_{i+1} \to v_i \to v_{j+1} \to \ldots$$

```
def retourne (tour, i,j) :
```

7) De la même manière qu'à la question 5, on choisit deux villes au hasard. On applique la fonction précédente entre ces deux villes. Si la distance est plus courte, on garde ce changement, sinon, on revient à la configuration précédente. On répète cette opération tant que la distance du chemin total diminue. (2 points)

```
def croisement (tour) :
```

8) On termine par l'exécution des fonctions `permutation`, `croisement`, `graph`. On vérifie que le chemin obtenu est vraisemblable même s'il n'est pas optimal.

```
def resoud_et_dessine (tour) :
```

Les deux transformations proposées pour modifier le chemin sont décrites par les fonctions `permutation` et `croisement`. A aucun moment, on ne s'est soucié du fait que le chemin est circulaire. Est-ce nécessaire ? Justifier. (2 points)