ALGORITHMES GLOUTONS

Table des matières

1) Le problème du voyageur

1.1) Problématique

Un voyageur se fixe un certain nombre de villes à visiter impérativement : Paris, Reims, Troyes, Nancy, Metz.

Il devra élaborer un trajet en partant d'une des villes de la liste (ville de départ), devra organiser son trajet pour visiter une seule fois toutes les autres villes de la liste et revenir en fin de parcours à la ville de départ. Pour des raisons budgétaires, il souhaite optimiser la longueur de son trajet.

A l'aide du site https://geoportail.gouv.fr, il détermine les distances minimales entres ces différentes villes :

1.2) Le nombre de possibilités

Pour la ville de départ, le voyageur a _ choix possibles ;
Pour le choix de la 2nde ville, il lui reste _ possibilités ;
Pour le choix de la 3ème ville, il lui reste _ possibilités ;
Pour le choix de la 4ème ville, il lui reste _ possibilités ;
Pour le choix de la 5ème ville, il lui reste _ possibilités ;
Au final, déterminer le nombre de trajets possibles pour 5 villes, 6 villes ... n villes.

Parmi les différents trajets possibles, y en aura-t-il qui auront systématiquement la même distance ? En déduire le nombre de distances de distances possibles pour n villes. Calculer ce nombre pour 10 villes, 20 villes ...

1.3) Nécessité d'algorithmes gloutons

Les algorithmiciens qui se sont penchés sur ce problème ont de solides raisons de penser qu'il n'existe aucun algorithme qui donnera une solution optimale en un temps optimal lorsque n est grand. Cependant, certains algorithmes, dit gloutons, sont capables de donner rapidement une solution acceptable.

Résolution approchée : L'idée est de décomposer le problème global dont la solution optimale est quasiment impossible à déterminer en raison de son coût machine par une succession d'étapes durant lesquelles ont fera un choix local et optimal (rapide en coût machine). Sur notre exemple, à chaque étape, on va se poser la question suivante : Quelle est la ville la plus proche ? Le meilleur choix est facile à déterminer et rapide. Puis on procède ainsi jusqu'à avoir parcouru toutes les villes. La question est de savoir si en faisant une série de choix localement optimaux, on fini par aboutir à une solution optimale. C'est parfois le cas mais pas toujours !

1.4) Algorithme du voyageur

Dans un 1er temps, on va créer la liste (de string) des villes à visiter :

```
villes=["Nancy","Metz","Paris","Reims","Troyes"]
```

On va ensuite indiquer le tableau des distances ci-dessus (liste de listes d'entiers) :

```
distances=[[0,55,303,188,183],
        [55,0,306,176,203],
        [303,306,0,142,153],
        [188,176,142,0,123],
        [183,203,153,123,0]]
```

Lancer le programme puis depuis la console python écrire l'instruction qui permet d'afficher :

- "Paris" instruction :
- "Nancy" instruction :
- La distance Paris-Nancy instruction :
- La distance Paris-Metz instruction :

Existe-t-il un lien entre la liste villes et la liste de listes distance (dans la liste villes, peut on inverser 2 éléments sans retoucher la liste distance ?)

Créer un tableau de booléen nommé visitees qui indiquera si la ville a été visitée (True) ou non (False). Au départ, comment faut-il initialiser les éléments de la liste ?

Écrire une fonction nommée plusProche, qui prend en arguments l'indice de la ville, la liste visitees et le tableau distance et qui va retourner l'indice de la ville la plus proche. Attention à bien initialiser le tableau visitees pour tester la fonction, la ville choisie est supposée avoir été visitée. On utilisera une variable locale nommée pp, initialement initialisée avec la valeur None, et à qui l'on assignera ensuite l'indice de la ville la plus proche.

Écrire maintenant une fonction nommée voyage, qui prend en arguments l'indice de la ville de départ, le tableau distance et la liste villes), affiche les distances parcourues entre chaque ville et retourne la distance totale parcourue. On utilisera 2 variables locales courante et suivante à qui l'on assignera l'indice de la ville en cours de visite et de la ville suivante.

Tester maintenant la globalité du programme en prenant comme ville de départ Nancy.

Le tableau ci-dessous donne l'ensemble des trajets avec comme ville de départ Nancy. Vérifier que votre algorithme glouton n'a pas donné la solution optimale mais malgré toute une solution très satisfaisante.

Circuit	Détail étapes	Total
Metz - Paris - Reims - Troyes	$55+306+142+123+183$	**809**
Metz - Paris - Troyes - Reims	$55+306+153+123+188$	**825**
Metz - Troyes - Paris - Reims	$55+203+153+142+188$	**741**
Troyes - Metz - Paris - Reims	$183+203+306+142+188$	**1022**
Troyes - Metz - Reims - Paris	$183+203+176+142+303$	**1007**
Metz - Troyes - Reims - Paris	$55+203+123+142+303$	**826**
Metz - Reims - Troyes - Paris	$55+176+123+153+303$	**810**
Metz - Reims - Paris - Troyes	$55+176+142+153+183$	**709**
Reims - Metz - Paris - Troyes	$188+176+306+153+183$	**1006**
Reims - Metz - Troyes - Paris	$188+176+203+153+303$	**1023**
Reims - Troyes - Metz - Paris	$188+123+203+306+303$	**1123**
Troyes - Reims - Metz - Paris	$183+123+176+306+303$	**1091**

2) Le problème du sac à dos

2.1) Problématique

Un voleur souhaite emporter des objets d'une valeur totale maximale dans son sac à dos qui ne peut supporter plus d'une certaine masse, à définir.

Objet	Valeur (euros)	Masse (kg)	valeur/masse(euros/kg)
Objet 1	126	14	9
Objet 2	32	2	16
Objet 3	20	5	4
Objet 4	5	1	5
Objet 5	18	6	3
Objet 6	80	8	10

2.2) Stratégies gloutonnes

Proposer 3 stratégies gloutonnes pour résoudre cette problématique.

Suivant la stratégie gloutonne adoptée, il sera sans doute nécessaire de rajouter une colonne au tableau et de trier le tableau suivant un ordre croissant ou décroissant d'une des colonnes. Afin de trier le tableau, on utilisera la fonction sorted() déjà utilisée dans le chapitre précédent.

Écrire une fonction nommée SacDosGlouton qui prend en argument le tableau d'objets, la masse maximale supportée par le sac à dos, l'indice de la colonne à trier, un booléen qui prend la valeur True pour un tri décroissant sinon la valeur False pour un tri croissant, et qui retourne un tableau dont le 1 éléments et le tableau des objets choisis, le 2nd élément la masse totale des objets emportés et le 3ème élément la valeur totale des objets emportés.

Appliquer vos différentes stratégies et vérifier la cohérence des réponses.

3) Problème du rendu de monnaie

3.1) Algorithme glouton

Considérons le problème d'un commerçant devant rendre de la monnaie à l'un de ses clients. Il souhaite le faire en utilisant le moins de pièces et de billets possibles. On suppose que l'on manipule les pièces et coupures habituelles des euros (1€, 2€, 5€, 10 €, 20€, 50€, 100€ et 200€, oublions les centimes) et que le commerçant dispose d'une réserve suffisamment importante de chaque espèce.

1. Donner les combinaisons possibles de rendu de monnaie sur 9 €. Établir un tableau.
2. A l'aide du tableau précédent, proposer une stratégie gloutonne pour rendre la monnaie.
3. Écrire une fonction qui prend en argument la somme à rendre (nombre entier) et retourne le tableau des pièces ou coupures.

3.2) Systèmes canoniques

On appelle système canonique un système où un algorithme glouton donne toujours le rendu optimal. En observant les réponses fournies par votre algorithme, pensez-vous que le système en euros soit canonique ?

Montrons de manière théorique que le système euro est canonique Montrer que dans un système de rendu optimal :

- Les pièces ou coupures de 1, 5,10, 50 et 100 € ne peuvent être utilisées qu'une seule fois ;
- Les pièces ou coupures de 2 et 20 € ne peuvent être chacun utilisés que deux fois ;
- On ne peut rendre à la fois 2 pièces de 2 € et 1 pièce de 1 € (ou 2 billets de 20 € et 1 billet de 10 €) ;
- La somme totale des billets ou coupures de moins de 200 € ne peut dépasser la somme de 199 € (au-delà de cette somme, un billet de 200 € sera alors choisi par l'algorithme glouton) ;
- La somme totale des billets ou coupures de moins de 100 € ne peuvent dépasser la somme de 99 € (au-delà de cette somme, un billet de 100 € sera alors choisi par l'algorithme glouton) ;
- On peut poursuivre ce raisonnement avec les autres coupures...

Appliquer votre algorithme glouton à un système monétaire dans lequel les pièces ou coupures auraient les valeurs suivantes : 1, 6, 10 et 22 €. Montrer sur un exemple que le rendu de monnaie n'est pas optimal et donc que ce système n'est pas canonique.