Séance 16 - Algorithmes gloutons

1) Définition générale

Les problèmes d'optimisation (recherche d'une meilleure solution selon un critère bien défini) sont des problèmes courant en mathématiques et en informatique.

Définition (Algorithme glouton):

Un algorithme glouton est un algorithme dans lequel on procède étape par étape en faisant, à chaque étape, le meilleur choix possible, sans jamais remettre en cause les choix précédents.

Remarque : cette méthode n'est pas toujours la meilleure méthode globalement mais il s'agit souvent d'une méthode simple à mettre en place.

1) Un premier exemple : le rendu de monnaie

1) Présentation de la situation

L'algorithme de rendu de monnaie consiste à choisir, à chaque étape, la plus grande pièce possible que l'on peut rendre, puis à réitérer sur la somme restante. C'est un exemple d'algorithme glouton qui permet de renvoyer le montant à rendre en minimisant le nombre de pièces utilisées.

On se place dans la position du vendeur.

2) Ecriture du code Python :

1. Créer une liste `L` contenant les différents montants des billets et pièces disponibles en euros dans la caisse.
2. Informer l'acheteur du prix à payer et lui demander la somme qu'il va verser. *(Utiliser la commande 'input').*
3. Créer une liste nommée `Rendu` qui contient le détail des billets/pièces à rendre, avec éventuellement des répétitions.
   Par exemple si la somme à payer est 16 et que le client donne 100, alors `Rendu = [50, 20, 10, 2,2]`.
4. Faire afficher dans des phrases "conviviales" le montant total à rendre ainsi que la décomposition en chaque billet/pièce.
5. **(Amélioration facultative)** A partir de la liste `Rendu`, créer une liste nommée `nombres_billets` telle que pour tout k, `nombres_billets[k]` soit égal au nombre de billets/pièces valant $L[k]$ qui sont à rendre.
   Puis afficher dans une phrase conviviale le nombre de billets de chaque sorte à rendre.
   Par exemple si `Rendu = [50,20,10, 2, 2 ]`,
   alors la phrase sera du type `"Vous devez rendre un  de 50, un de 20, un de 10, et deux  de 2"`.
6. **(Amélioration facultative)** Améliorer l'étape 5 pour que la phrase finale fasse la différence entre les pièces et les billets.

3) Un deuxième exemple : plannification optimale de conférences

a) Présentation du problème

Des conférenciers ont été invités lors d'une journée afin de présenter leurs travaux de recherche. Un seul auditorium est disponible pour les accueillir et du fait des contraintes liées à leur agenda bien chargé, les conférenciers ne peuvent pas intervenir à n'importe quel moment de la journée et ils ont donc proposé des plages horaires précises pour intervenir (plages horaires dont la longueur varie en fonction de la durée de leur exposé). Malheureusement, les plages horaires qu'ils proposent ne permettent pas aux organisateurs de faire venir l'ensemble des conférenciers (certaines plages horaires ayant une intersection non vide). Les organisateurs doivent donc faire des choix, sachant qu'ils veulent faire venir le maximum de conférenciers.
On note n le nombre de conférenciers. Chaque conférencier est identifié par une lettre C_i, où i est un entier entre 0 et $n-1$. A chaque conférencier C_i est associé un intervalle temporel $[d_i; f_i[$, où d_i et f_i désignent respectivement l'heure de début et l'heure de fin de l'intervention pour le conférencier C_i.
Afin de dégager une tactique de résolution du problème, commençons par analyser plusieurs situations.

b) Analyse de plusieurs situations

- **Situation 1 : Créneaux disjoints.**
  Quatre conférenciers interviennent sur les intervalles suivants :
  $C_1 : [3, 4[$, $C_2 : [0, 1[$, $C_3 : [2, 3[$, $C4 : [1, 2[$.

  1. Représenter graphiquement la situation. Y a t-il un choix de conférenciers à faire ?

- **Situation 2 : Créneaux non disjoints**
  Quatre conférenciers interviennent sur les intervalles suivants :
  $C_1 : [2, 4[$, $C_2 : [0, 1[$, $C_3 : [1, 3[$, $C_4 : [0, 2[$.
  Dans cette situation, les intervalles ne sont plus disjoints. Nous dirons que ces intervalles ne sont pas **compatibles**. Des choix doivent être faits et certains conférenciers peuvent ne pas être retenus pour construire un planning.

  2. Représenter graphiquement la situation.

  3. Lister les différentes possibilités d'occupation de l'auditorium si on ne met aucune contrainte de maximisation du nombre de conférenciers.

  4. Combien y a t-il de possibilités et quelles sont celles qui maximisent le nombre de conférenciers ?

Normalement, vous avez vu que plusieurs solutions optimales sont possibles. Mais laquelle de ces solutions retenir ? Pour y répondre, il convient de se demander comment un algorithme pourrait aboutir à la construction de ces solutions.

7. **Méthode 1. Classement par ordre croissant des heures de début d_i des intervalles compatibles.**
   Sur la situation précédente, construire un planning où, à chaque étape, pour désigner le conférencier suivant, on choisit parmi les créneaux compatibles avec la conférence qui vient de se terminer, celui qui peut commencer le plus rapidement. A quelles solutions amènent-ils ?

8. **Méthode 2. Classement par ordre croissant des heures de fin f_i des intervalles compatibles.**
   Sur la situation précédente, construire un planning où, à chaque étape, pour désigner le conférencier suivant, on choisit parmi les créneaux compatibles avec la conférence qui vient de se terminer, celui qui termine sa conférence le plus tôt. En procédant ainsi, vérifier que vous retrouver les mêmes solutions que précédemment.

Bilan des essais : il semble que ces deux idées mènent à des résultats identiques. En outre, elles n'ont pas permis d'éliminer des solutions afin de n'en fournir qu'une seule.
Recherches et réflexions personnelles : *(Voyez vous des inconvénients ou des avantages à une des solutions ? Que pourrait-on ajouter comme contraintes ? ...)*

C'est à ce niveau que la **stratégie gloutonne** intervient. Celle-ci va faire un premier choix de conférencier en suivant un critère à préciser. **Ce choix ne sera jamais remis en question et la même stratégie sera appliquée pour trouver les conférenciers suivants.**

9. Après avoir classé les intervalles par valeurs croissantes des heures de début, sélectionner l'intervalle de la **première plus petite** valeur, puis celui de la deuxième plus petite valeur compatible avec la précédente, et ainsi de suite. Quelles solutions restent-ils ?

10. Après avoir classé les intervalles par valeurs croissantes des heures de fin, sélectionner l'intervalle de la première plus petite valeur, puis celui de la deuxième plus petite valeur compatible avec la précédente, et ainsi de suite. Quelles solutions restent-ils ?

11. Quelle vous semble être la meilleure stratégie ?

Conclusion de la phase de recherche.
Une solution semble émerger des observations précédentes. On peut donc proposer la stratégie suivante.

- 1. Classer les conférences par heures de fin croissantes.
- 2. Choisir la conférence classée en premier.
- 3. Choisir parmi les suivantes qui sont compatibles avec celle-ci, celle qui se termine en premier.
- 4. Recommencer ainsi les choix jusqu'à ce qu'il n'y ait plus de conférences à traiter.

Un tel algorithme est effectivement de type glouton : chaque choix fait sélectionne l'une des meilleures possibilités et ne remet jamais en cause les choix précédents.

12. Quelle solution obtient-on en appliquant à la main cet algorithme avec les conférenciers suivants :

$$C_1 : [0,7[,\ C_2 : [2,5[,\ C_3 : [6,8[,\ C_4 : [1,2[,\ C_5 : [5,6[,\ C_6 : [0,2[,\ C_7 : [4,7[,\ C_8 : [0,1[,\ C_9 : [3,6[,\ C_{10} : [1,3[$$

$$C_{11} : [4,5[,\ C_{12} : [6,8[,\ C_{13} : [0,2[,\ C_{14} : [5,7[,\ C_{15} : [1,4[$$

e) Mise en oeuvre sous Python

Dans le programme suivant, nous allons implémenter l'algorithme présenté précédemment.
On suppose que les conférenciers sont représentés par des numéros à partir de 0.
On appelle `planning` la fonction permettant d'organiser le temps d'occupation de la salle :

- Elle prend deux entrées :
  - une liste `debut` qui contient l'horaire de début des conférenciers (ainsi `debut[i]` correspond à l'horaire de début du conférencier n° i)
  - une liste `fin` qui contient l'horaire de fin des conférenciers (ainsi`fin[i]` correspond à l'horaire de fin du conférencier n° i).
- Elle renvoie la liste des conférenciers à sélectionner pour maximiser le nombre de conférences dans la journée.

A l'intérieur de la fonction `planning`, interviennent les variables suivantes :

- `conferenciers_dispo` : contient la liste des conférenciers encore disponibles après chaque étape
- `choix` : contient la liste des conférenciers choisis à chaque étape et est donc renvoyée par la fonction à la fin.

```
def planning(debut,fin):
    n=len(debut)
    conferenciers_dispo=[i for i in range(n)]
    choix=[]
    while conferenciers_dispo!=[]:
        for i in range(len(conferenciers_dispo)):
            if fin[conferenciers_dispo[i]] == min([fin[x] for x in conferenciers_dispo]):
                j=conferenciers_dispo[i]
        choix.append(j)
        conferenciers_dispo = [x for x in conferenciers_dispo if debut[x]>=fin[j]]
    return choix
```

13. Que représente la variable n ?

14. Commenter l'instruction `conferenciers_dispo=list(range(n))`.

15. Expliquer la condition `if fin[conferenciers_dispo[i]] == min([fin[x] for x in conferenciers_dispo])]`.

16. Expliquer pourquoi et comment l'avant dernière ligne permet de modifier la liste des conférenciers disponible après le choix d'un nouveau conférencier.

17. Construire les listes `debut` et `fin` correspondant à la situation de la question 12.

18. L'exécution de la commande `print(planning(debut_test,fin_test))` renvoie l'affichage :
    `[7, 3, 10, 4, 11]`
    Cela correspond-il à la solution trouvée précédemment "à la main" ?