TP 07

Algorithme glouton

Savoirs et compétences :

- *AA.C9 : Choisir un type de données en fonction d'un problème à résoudre*
- *AA.S12 : Fichiers*

Activité 1 – Problème glouton du rendu de monnaie

La société Sharp commercialise des caisses automatiques utilisées par exemple dans des boulangeries. Le client glisse directement les billets ou les pièces dans la machine qui se charge de rendre automatiquement la monnaie.

Objectif Afin de satisfaire les clients, on cherche à déterminer un algorithme qui va permettre de rendre le moins de monnaie possible.

La machine dispose de billets de 20€, 10€ et 5€ ainsi que des pièces de 2€, 1€, 50, 20, 10, 5, 2 et 1 centimes.

On se propose donc de concevoir un algorithme qui demande à l'utilisateur du programme la somme totale à payer ainsi que le montant donné par l'acheteur. L'algorithme doit alors déterminer quels sont les billets et les pièces à rendre par le vendeur.

Pour ne pas faire d'erreurs d'approximation, tous les calculs seront faits en centimes.

Le contenu de la caisse automatique et le contenu du porte-monnaie du client seront modélisés par un tableau ayant la forme suivante :

```
caisse = [[2000,5],[1000,5],[500,5],[200,5],[100,5],[50,5],[20,5],[10,5],[5,5],[2,5],[1,5]]
```

Cela signifie que la caisse contient 5 billets de 20€, 5 billets de 10€...

Dans la prochaine question, on fait l'hypothèse que la caisse contient suffisamment de billets et de pièces de chaque valeur.

Question 1 *Ecrire une fonction* `rendre_monnaie(caisse:list, cout:float, somme_client:float)->list` *prenant en arguments deux flottants* `cout` *et* `somme_client` *représentant le coût d'un produit et la somme donnée par le client en€ ainsi que le contenu de la caisse. Cette fonction renvoie la liste des billets à rendre par le client.*

Ainsi, l'instruction `rendre_monnaie(caisse, 16, 20)` renvoie la liste `[200, 200]`.
`rendre_monnaie(caisse,15.92,20)` renvoie `[200, 200, 5, 2, 1]`

Question 2 *Ecrire une fonction* `rendre_monnaie_v2(caisse:list, cout:float, somme_client:float)->list` *ayant le même objectif que la précédente. Cette fonction devra de plus mettre à jour la caisse. Elle devra prendre en compte que la caisse peut manquer de billets. Elle renverra une liste vide s'il n'est pas possible de rendre la monnaie.*

Pour l'instruction `rendre_monnaie_v2(caisse, 15.99, 200)`, la liste renvoyées est la suivante : `[2000, 2000, 2000, 2000, 2000, 1000, 1000, 1000, 1000, 1000, 500, 500, 500, 500, 500, 200, 200, 200, 200, 100, 1]`. Le contenu de la caisse est alors : `[[2000, 0], [1000, 0], [500, 0], [200, 1], [100, 4], [50, 5], [20, 5], [10, 5], [5, 5], [2, 5], [1, 4]]`.

L'instruction `rendre_monnaie_v2(caisse, 15.99, 200)` renvoie `[]`.

On suppose que la caisse est maintenant la suivante.

```
caisse = [[5000,10],[2000,10],[1000,10],[800,10],[100,10]]
```

Le client achète un article de 34€ avec un billet de 50€.

Question 3 *Que retourne la fonction* `rendre_monnaie` *? Est-ce le rendu optimal?*

Pour l'instruction `rendre_monnaie_v2(caisse, 34, 50)`, l'algorithme renverra `[1000, 100, 100, 100, 100, 100, 100]`.

Question 4 *Conclure « qualitativement ».*

Activité 2 –

D'après documents de Serge Bays.

Nous considérons la variante « entière » du problème du sac à dos. Nous sommes devant un ensemble de n objets. Chaque objet noté o_i a une valeur notée v_i et un poids noté p_i. Il s'agit d'emporter dans son sac à dos l'ensemble d'objets qui a la plus grande valeur sachant que le sac supporte un poids maximum P. Comment résoudre ce problème, quels objets doit-on prendre?

Pour appliquer une stratégie gloutonne, nous devons définir ce que nous entendons par le meilleur choix à chaque étape.

Il y a trois manières ici de définir un meilleur choix. Parmi les objets qui n'ont pas encore été pris, soit on choisit un objet qui a la valeur maximale, soit un objet qui a le poids minimal, soit un objet qui a le rapport valeur/poids maximal.

Nous ne considérons que les objets ayant un poids $p_i \leq P$. L'algorithme glouton consiste, à chaque étape, à choisir parmi ces objets celui qui représente le choix optimal. Nous le notons O_1, sa valeur V_1 et son poids P_1. Ensuite, nous recommençons parmi les objets de poids $p_i \leq P - P_1$. Et ainsi de suite.

Cette variante est dite entière parce que chaque objet est pris ou pas. Il existe une version fractionnaire. Le principe de base est le même, mais cette fois il est possible de prendre des fractions d'objets. Un algorithme glouton donne une solution optimale à cette variante fractionnaire. Prenons un exemple : le sac à dos peut contenir 15 kg. Les poids des objets sont en kg, les valeurs en euro.

Objet	Valeur	Poids	Valeur/Poids
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

Un objet est représenté par une liste comme `['objet 1', 126, 14]`.

On a donc :

```
objets = [['objet 1', 126, 14], ['objet 2', 32, 2], ['objet 3', 20, 5], ['objet 4', 5, 1], ['objet 5', 18, 6], ['objet 6', 80, 8]]
```

Question 5 *Définir la fonction* `def valeur(objet:list) -> float` *qui renvoie la valeur d'un objet.*

Question 6 *Définir la fonction* `def poids(objet:list) -> float` *qui renvoie le poids d'un objet.*

Question 7 *Définir la fonction* `def rapport(objet:list) -> float` *qui renvoie le rapport valeur/poids d'un objet.*

L'algorithme glouton du sac à dos est défini ainsi :

- définir la fonction `glouton` qui prend en paramètres une liste d'objets, un poids maximal (celui que peut supporter le sac à dos) et le type de choix utilisé (par valeur, par poids, ou par valeur/poids);
- trier la liste d'objets suivant le type de choix utilisé par ordre décroissant;
- définir la variable `reponse:list` sert à stocker les objets choisis;
- parcourir la liste triée et ajouter les noms des objets un par un tant que le poids ne dépasse pas le poids maximal.
- stocker valeur totale et le poids total sont stockés dans deux variables `valeur` et `poids`.

R Tri d'une liste selon un critère :

```
copie = sorted(liste, key=critere, reverse=True)
```

`critere` peut être une des fonctions précédemment définies (`rapport`, `poids` ou `valeur`) .

Question 8 *Implémenter la fonction* `glouton` *qui renvoie (dans l'ordre)* `reponse` *et* `valeur`.

Question 9 *Exécuter la fonction* `glouton` *pour les différents types de choix. On observant la nature des différents objets, le choix optimal est-il parmi les choix proposés ?*

Question 10 *Estimer le nombre de tours de boucle nécessaire pour exécuter l'algorithme* `glouton` *en fonction du nombre d'objets* `n`.

On cherche maintenant à tester l'ensemble des combinaisons possibles permettant de remplir le sac à dos.

Question 11 *Estimer (grossièrerement) le nombre de combinaisons possibles parmi la liste d'objets qui permettrait de remplir le sac à dos. Conclure sur l'intérêt d'un algorithme glouton.*

Question 12 *Proposer une variante récursive de l'algorithme glouton.*