TP10 - Algorithmes gloutons

→ *Q.1) Récupérez le fichier* `TP10.py` *sur cahier de prépa dans lequel vous pourrez sauvegarder votre travail.*

A) Rendu de monnaie

Le problème du rendu de monnaie consiste à déterminer le nombre minimal de pièces à utiliser pour obtenir une somme donnée. Il y a deux paramètres au problèmes : la somme à obtenir et les valeurs des pièces disponibles. On suppose que les pièces sont à valeurs entières, qu'on peut en prendre autant qu'on veut de chaque sorte, et que la plus petit pièce vaut 1.

A.1 Recherche exhaustive

Pour rendre `s` euros :

— soit s est nul ;

— soit on prend une pièce de `x` euros puis on cherche la solution optimale pour `s-x` euros. On effectue cette recherche pour toutes les valeurs de `x` possibles et on prend le nombre minimal de pièces nécessaires pour obtenir `s`.

→ *Q.2) Écrire une fonction récursive* `rendu_exh(s, L)` *qui prend en paramètre un entier naturel et un système monétaire et renvoie le nombre minimal de pièces nécessaires pour obtenir* `s`. *On pourra utiliser la fonction* `min` *qui renvoie le minimum d'une liste.*

→ *Q.3) On fournit le système monétaire (en centimes d'euros) dans la liste* `euros`. *Tester la fonction* `rendu_exh` *avec ce système et sur quelques sommes (petites).*

→ *Q.4) Quels sont les désavantages de cet algorithme ?*

→ *Q.5) Bonus : modifier la fonction* `rendu_exh` *pour qu'elle renvoie la liste des pièces à utiliser.*

→ *Q.6) Bonus bis : optimisez la fonction* `rendu_exh` *en sauvegardant les pièces à rendre pour obtenir les sommes inférieures ou égales à* `s` *dans une liste.*

A.2 Algorithme glouton

Pour rendre `s` euros, on commence par prendre la pièce `x` de la valeur la plus grande, puis on recommence avec `s-x` euros.
Par exemple, pour rendre 23 euros, on va rendre $20+2+1$ euros.

→ *Q.7) Écrire une fonction* `rendu_glouton(systeme, somme)` *qui implémente ce principe pour tenter de répondre au problème du rendu de monnaie.*

→ *Q.8) Commenter et tester cette fonction avec* `euros`.

→ *Q.9) Comparer les deux fonctions* `rendu_glouton` *et* `rendu_exh`.

Un **algorithme glouton** est un algorithme qui cherche une solution optimale à un problème en suivant le principe de faire à chaque étape le choix optimum local, c'est-à-dire le choix le meilleur pour l'étape en cours. L'objectif est d'obtenir à la fin une solution qui est optimale globalement.
Ce n'est pas toujours le cas : on dit alors que l'algorithme suit une **heuristique gloutonne**.

→ *Q.10) On donne une liste* `autre`. *Tester l'algorithme glouton avec ce système monétaire. Que remarquez-vous ?*

Il se trouve qu'on peut montrer que l'algorithme glouton fonctionne toujours avec des euros !

B) Sélection d'activités

La salle 225 est, c'est bien connu, la meilleur salle du lycée. Le proviseur adjoint veut donc mettre un maximum de cours dans cette salle.
Chaque cours a lieu sur un intervalle de temps $[d_i, f_i[$: deux cours sont compatibles si leurs intervalles sont disjoints.
On dispose donc de n couples d'entiers $(d_1, f_1), \dots, (d_n, f_n)$ et on souhaite trouver un ensemble $J \subset [\![1, n]\!]$ de cardinal maximal tel que tous les cours correspondants soient compatibles deux à deux.
Voici trois exemples de données :

1. $C_1 : [3,4[$, $C_2 : [0,1[$, $C_3 : [2,3[$ et $C_4 : [1,2[$;
2. $C_1 : [0,3[$, $C_2 : [2,4[$ et $C_3 : [3,6[$.

3. $C_1 : [0,3[$, $C_2 : [1,2[$ et $C_3 : [2,3[$.

Pour tenter de résoudre ce problème avec un algorithme glouton, on peut s'intéresser à trois critères à optimiser à chaque étape :

— la durée du cours ;

— l'heure de début ;

— l'heure de fin.

→ *Q.11) On donne la priorité au cours qui commence en premier à chaque décision : tester à la main sur les 3 exemples de données.*

→ *Q.12) On donne la priorité au cours le plus court à chaque décision : tester à la main sur les 3 exemples de données.*

→ *Q.13) On donne maintenant la priorité au cours qui finit en premier. Écrire une fonction* `cours_glouton(C)` *qui prend en argument une liste d'éléments du type* `["Cj", dj, fj]` *où* `Cj` *est le nom du cours,* `dj` *et* `fj` *sont les heures de début et de fin du cours (de type* `int`*). Elle doit renvoyer la liste des cours choisis (sous la même forme que* `C`*).*
On pourra utiliser l'instruction `C.sort(key=takeThird)` *qui utilise la fonction* `takeThird` *définie pour trier la liste* `C` *par ordre croissant selon les troisièmes valeurs de chaque élément de* `C`.

→ *Q.14) Tester cette fonction sur les trois exemples précédents.*

→ *Q.15) Écrire une fonction* `cours_alea(debut, fin, n)` *qui prend une heure de début, une heure de fin et renvoie une liste aléatoire de* `n` *cours qui commencent après* `debut` *et finissent avant* `fin`.

→ *Q.16) Tester la fonction* `cours_glouton` *avec des listes obtenues avec la fonction* `cours_alea`.

C) Pour les plus motivés

Exercice 1 -

1. On considère le système monétaire `[1, x]` avec `x` un entier strictement plus grand que 1. Montrer que l'algorithme glouton donne la solution optimale pour ce système.
2. La fonction `rendu_glouton` renvoie une liste pour représenter les pièces rendues. Écrire une nouvelle fonction qui renvoie plutôt un dictionnaire dont les clés sont les valeurs de pièces rendues et dont les valeurs associées sont leur nombre.

D) Bilan

▷ Pour résoudre un problème d'optimisation, il existe plusieurs méthodes mais qui sont parfois coûteuses.

▷ Les algorithmes gloutons sont une alternative, simples à programmer, rapides, mais le résultat obtenu n'est pas toujours optimal : on a alors simplement une heuristique gloutonne.

▷ On peut en général utiliser un algorithme glouton lorsque :

(a) une solution du problème peut être obtenue à partir de solutions partielles ;

(b) les solutions partielles sont obtenues en suivant un critère simple qui permet d'évaluer leur optimalité.