TP : Algorithmes gloutons

Le principe d'un algorithme glouton (greedy algorithm) est de faire toujours un choix localement optimal dans l'espoir que ce choix mènera à une solution globalement optimale.

I. Algorithme glouton sur le rendu de monnaie

Nous allons illustrer ce principe sur le problème du rendu de la monnaie : on dispose de pièces de monnaie dont les montants sont des nombres entiers de l'unité de monnaie : $t_0 < t_1 < t_2 < \ldots < t_{N-1}$.
Pour payer une somme S (entier naturel quelconque), on aimerait utiliser le nombre minimal de pièces possibles.
On donne la plus « grosse » pièce possible. Puis à nouveau la plus grosse pièce possible pour le montant restant.
Puis à nouveau la plus grosse pièce possible pour le montant restant. etc...
On suppose l'existence d'une variable globale `ListeMontants`, pointant vers une liste de toutes les valeurs possibles de pièces et billets, ordonnées de la plus petite à la plus grande.
Pour notre TP, nous utiliserons le système monétaire européen.
Ainsi, `ListeMontants = [0.01,0.02,0.05,0.1,0.2,0.5,1,2,5,10,20,50,100,200,500]`.

1. Programmation du rendu de monnaie

1. Créer une fonction `Piece_Billet(Reste)` qui prend en argument le reste de la monnaie à rendre, et qui renvoie la plus grande valeur possible à rendre dans le système utilisé.
   Testez votre fonction : `Piece_Billet(322)` donne 200, `Piece_Billet(0.99)` donne 0.5.
2. Créer une fonction `Monnaie(Somme)` qui crée et renvoie la liste du rendu de monnaie associé à la somme entrée, en respectant le principe de l'algorithme glouton.
3. Mettre en place le code permettant de déterminer et afficher dans la console la monnaie de 97 € 22. (Réponse : [50, 20, 20, 5, 2, 0.2, 0.02])

Dans le système en euros, nous admettrons que cet algorithme est optimal, on dit qu'il est « canonique ».

4. Montrer par des exemples, qu'il existe des systèmes monétaires pour lesquels l'algorithme glouton n'est pas optimal.

2. Recherche d'optimalité dans certains cas

a) Étude sur un exemple numérique

On considère dans cette partie uniquement le système monétaire suivant : $[1, 2, 5, 10]$. On cherche à obtenir la composition optimale pour une somme S.

1. Combien doit-on utiliser au plus de pièces de 5 € pour être optimal ?
2. Combien doit-on utiliser au plus de pièces de 2 € pour être optimal ?
3. Combien doit-on utiliser au plus de pièces de 1 € pour être optimal ?
4. Quelle est donc la somme maximale qu'on va payer de manière optimale avec les pièces de 1, 2 et 5 €.
5. En déduire le nombre de pièces de 10 € qui vont servir pour payer la somme S.
6. Quelle est la somme S_1 qu'il reste alors à payer avec les pièces de 1, 2 et 5 € ?
7. Reprendre le principe des questions 1 à 6 pour payer S_1 avec les pièces de 1, 2 et 5 € ?

b) Montants en progression géométrique

On considère dans cette partie uniquement le système monétaire suivant : $[1, p, p^2, p^3, \ldots, p^k]$ où $p \in \mathbb{N} \setminus \{0, 1\}$. On cherche toujours la composition optimale d'une somme S.

1. Combien doit-on utiliser au plus de pièces de valeurs p^{k-1} pour être optimal ?
2. Combien doit-on utiliser au plus de pièces de valeurs p^{k-2} pour être optimal ?
   etc...
3. En déduire la somme maximale payée de manière optimale avec les pièces $1, p, p^2, p^3, \ldots, p^{k-1}$.
4. Donner le nombre de pièces p^k utilisées pour payer la somme S de manière optimale.

II. Algorithme glouton sur l'allocation de salles de spectacles

On cherche une solution au problème d'allocation d'une salle de spectacles. On définit une liste L contenant pour chaque spectacle d'indice $i \in [\![0, n-1]\!]$ le couple d'entiers (d_i, f_i) où d_i désigne l'heure de début et f_i l'heure de fin : $L = [[d_0, f_0], [d_1, f_1], \ldots, [d_{n-1}, f_{n-1}]]$. On suppose que cette liste des listes $[d_i, f_i]$ est triée par date de fin f_i croissante. On définit `début` l'heure de début du spectacle et `fin` l'heure de fin du spectacle.

1. On cherche à maximiser le nombre de spectacles dans la salle et non le temps d'occupation. On utilise la méthode intuitive consistant à choisir au fur et à mesure des spectacles dont l'intervalle est compatible avec celui du spectacle précédent et dont l'heure de fin est la plus petite. On suppose que la liste L est triée par ordre croissant des heures de fin.
   Écrire une fonction itérative `gestion` qui admet comme arguments une liste L, un entier `début` (heure de début des spectacles) et un entier `fin` (heure de fin des spectacles). La fonction retourne le nombre maximum de spectacles que l'on peut organiser dans la salle ainsi que la liste des spectacles retenus.
2. On considère la liste $L_1 = [[0, 2], [1, 3], [2, 4], [1, 5], [3, 6], [4, 7], [5, 9], [6, 11], [9, 12]]$.
   Écrire le programme principal permettant d'afficher le nombre maximum de spectacles et la liste des spectacles que l'on peut organiser dans cette salle entre `début=1` et `fin=12`.

Remarque 1 : On peut montrer que cet algorithme donne la solution optimale. On a vu dans la partie « Rendu de monnaie » que la méthode gloutonne ne donne pas toujours la solution optimale.