TP 7 - Algorithmes gloutons

I Introduction

En informatique, une stratégie est qualifiée de gloutonne lorsqu'à chaque étape on sélectionne un optimum local dans l'espoir d'obtenir un résultat optimum global.

Un algorithme glouton (greedy algorithm en anglais, parfois appelé aussi algorithme gourmand, ou goulu) est un algorithme suivant une stratégie gloutonne.

Par exemple, dans le problème du rendu de monnaie (donner une somme avec le moins possible de pièces), l'algorithme consistant à répéter le choix de la pièce de plus grande valeur qui ne dépasse pas la somme restante est un algorithme glouton.

Une telle stratégie peut parfois marcher et parfois être un échec.

Par exemple, toujours dans le cas du rendu de monnaie, imaginons que l'on ait une quantité illimitée de pièces de valeurs comprises dans l'ensemble $\{1, 3, 6, 12, 24, 30\}$ et que l'on cherche à rendre 49. Alors l'algorithme glouton proposerait comme solution $49 = 30 + 12 + 6 + 1$ mais la solution optimale globale est en fait $49 = 2 \times 24 + 1$.

II Le rendu de monnaie

On cherche à coder un algorithme "essayant" de résoudre de manière gloutonne le problème du rendu de monnaie décrit dans l'introduction. On supposera qu'une solution au problème est toujours possible.

Question 1. *Implémenter une fonction* `rendu_classique(syst,n)` *ayant pour paramètre une liste d'entiers* `syst` *dont les éléments sont les valeurs des pièces de monnaie dont on dispose triée dans l'ordre décroissant et* `n` *un entier indiquant la somme que l'on doit rendre. cette fonction devra renvoyer les pièces rendu sous la forme d'une liste d'entiers dont les éléments sont les valeurs des pièces rendues.*

Question 2. *Et si la liste* `syst` *est triée dans l'ordre croissant ? Et si elle n'est pas triée du tout ?*

Question 3. *On suppose désormais que l'on a pas une infinité de pièces de chaque valeur.*

Implémenter une fonction `rendu_complique(syst,poss,n)` *résolvant le même problème que* `rendu_classique` *mais avec ce nouveau paramètre* `poss` *étant une liste telle que* `poss[i]` *indique le nombre de pièces de valeur* `syst[i]` *que l'on possède.*

III Un problème de choix d'activités

Notre second exemple concerne le problème de l'ordonnancement de plusieurs activités qui rivalisent pour l'utilisation d'une ressource commune.

Supposez que l'on ait un ensemble $S = \{a_1, a_2, \ldots, a_n\}$ de n activités qui veulent utiliser une ressource, par exemple une salle de cours ne pouvant servir qu'à une seule classe à la fois. Chaque activité a_i a une heure de début s_i et une heure de fin f_i avec $0 \leqslant s_i < f_i < +\infty$. Si elle est sélectionnée, l'activité a_i a lieu pendant l'intervalle temporel $[s_i, f_i[$. Les activités a_i et a_j sont dites compatibles si les intervalles $[s_i, f_i[$ et $[s_j, f_j[$ sont disjoints.

Le problème du choix d'activités consiste à sélectionner un sous-ensemble d'activités compatibles deux à deux de taille maximale.

Pour résoudre cela, on supposera l'ensemble des activités donné sous la forme d'une liste L Pyhton composée de listes de deux entiers. Ainsi, si il n'y a que deux activités alors $L = [[s_1, f_1], [s_2, f_2]]$. On supposera de plus que les activités sont triées par ordre croissant des heures de fin (i.e $f_1 \leqslant f_2 \leqslant \ldots \leqslant f_n$).

III.A Faire le choix glouton

Qu'entend-on par choix glouton pour le problème de la sélection d'activités ?

L'intuition nous dicte de choisir la première à se terminer. Ainsi, cela laisserait la ressource disponible pour le plus grand nombre d'activités qui suivent. Le choix glouton consiste donc à prendre systématiquement la première activité de la liste (puisqu'on l'a supposée triée par ordre croissant des heures de fins).

On doit ensuite considérer la première des activités restantes qui est compatible avec la dernière que l'on a choisit et ainsi de suite.

Mais il reste une question : Est ce qu'une telle stratégie nous amène à une solution optimale ?

Question 4. *Soit $S = \{a_1, a_2, \ldots, a_n\}$ un ensemble d'activités triées par ordre croissant d'heures de fin et soit $S' \subset S$ un ensemble optimale d'activités compatibles. Montrer que si a_j est l'activité de S' ayant la plus petite heure de fin alors on peut remplacer cette dernière par a_1 dans S' et garder notre propriété d'optimalité.*

Qu'en conclure ?

III.B Coder l'algorithme glouton

D'après la discussion précédente, notre but va être de parcourir la liste L avec une boucle `for` en gardant en mémoire les activités déjà sélectionnées ainsi que l'heure de fin de la dernière activité et de mettre à jour cela à chaque fois que l'on trouve une nouvelle activité compatible.

Question 5. *Codez moi ça !*

Question 6. *Pouvez vous résoudre le problème à l'aide une fonction récursive ayant pour argument la liste L' des activités restantes à parcourir ainsi que l'heure de fin de la dernière activité sélectionnée (ces deux paramètres initialement égaux à L et 0). Cette dernière ne serait constituée que d'un test conditionnel et utiliserait le slicing de liste pour construire les arguments de l'appel récursif.*

III.C D'autres raisonnements gloutons

Question 7. *Supposez que, au lieu de sélectionner systématiquement la première activité à se terminer, on sélectionne la dernière activité à démarrer qui soit compatible avec toutes les activités précédemment sélectionnées.*

Expliquer en quoi cette approche est une stratégie gloutonne et prouver qu'elle donne une solution optimale. Tant qu'à faire, codez la également.

Question 8. *Supposez qu'on ait un ensemble d'activités à répartir sur un grand nombre de salles de cours, chaque activité pouvant avoir lieu dans n'importe quelle salle. On souhaite planifier toutes les activités avec le minimum de salles de cours. Donner un algorithme glouton efficace qui détermine quelle est l'activité qui doit avoir lieu dans telle ou telle salle de cours.*