# *TP d'informatique n° 10 : algorithmes gloutons*

# *PCSI 2023 – 2024*

*Objectifs : Le but est de découvrir les algorithmes gloutons qui permettent de trouver rapidement des solutions à des problèmes. On mettra en évidence « expérimentalement » que ces algorithmes ne donnent pas forcément les « meilleures solutions ».*

*Avant le TP : pour ce TP nous aurons besoin d'un algorithme de tri. Revoir le codage du tri à bulle que nous utiliserons durant ce TP*

*La méthode gloutonne*
*Principe*

*On cherche une solution optimale à un problème.*

*La technique la plus basique pour résoudre ce type de problème consiste à énumérer de façon exhaustive toutes les solutions possibles, puis à choisir la meilleure.*
*Cette approche par force brute, toujours possible, n'est sans doute pas optimale en termes de temps de calcul.*

*La méthode gloutonne consiste à choisir des solutions locales d'un problème dans le but d'obtenir une solution globale et rapide en termes de temps de calcul au problème. On ne trouvera pas forcément la « meilleure solution ». Le but des problèmes suivants est de le mettre en évidence.*

## *I Rendu de Monnaie*

*Le problème du rendu de monnaie est un problème d'algorithmique. Il s'énonce de la façon suivante : étant donné un système de monnaie (pièces et billets), comment rendre une somme donnée de façon optimale, c'est-à-dire avec le nombre minimal de pièces et billets ?*

*Dans la zone euro, le système de pièces et de billets en vigueur est, en mettant de côté les centimes d'euros :* **{1, 2, 5, 10, 20, 50, 100, 200 €}**

### Résolution exacte par la méthode de « force brute »

*Exercice n°1 Vincent veut acheter une nouvelle souris de gamer pour son PC. Celle-ci coûte 53 euros. A la caisse automatique d'une grande surface qui accepte le paiement en monnaie, il paye avec un billet de 100 euros. La caisse automatique doit lui rendre 47 euros. Indiquer ( à la main) quelques-unes des combinaisons possibles de pièces et / ou de billets permettant de le faire. Est-il facile de toutes les dénombrer ?*

*Exercice n°2 Parmi toutes les combinaisons précédentes possibles, quelle est, selon vous, celle permettant de minimiser le nombre de pièces et / ou de billets utilisés ? Cette solution sera qualifiée d'optimale.*

### Résolution approchée par la méthode de l'algorithme glouton

*Utiliser un algorithme glouton consiste à optimiser souvent en terme de temps de calcul la résolution d'un problème avec contrainte en utilisant l'approche suivante : on procède étape, par étape, en faisant à chaque étape, le choix qui semble être le meilleur, sans jamais remettre en question les choix précédents. Dans de nombreuses situations de la vie quotidienne, nous employons*

*intuitivement, sans le savoir, des algorithmes gloutons. Le problème du rendu de monnaie en est une illustration typique.*
*Dans notre cas de rendu de monnaie, on prend le système de monnaie sur forme d'une liste triée du plus grand au plus petit élément, par exemple monnaie_euro = [200, 100, 50, 20,10, 5, 2, 1]. Pour essayer de rendre la monnaie de façon optimale : dans notre cas avec le moins de pièces et de billets avec votre algorithme glouton, on choisit toujours la plus grande monnaie que l'on peut rendre tant que la somme totale à rendre n'est pas nulle. Avec la liste monnaie_euro, on commence par voir si on peut rendre la monnaie avec des billets de 200 euros, puis lorsque la somme restante à rendre est inférieure à 200 euros, des billets de 100 euros, ...*

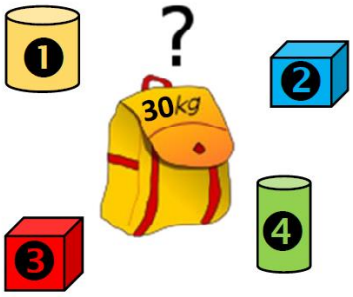
*Exercice n°3 Définir une fonction monnaie_a_rendre(somme_a_rendre : int , système_de_monnaie :List), et qui renvoie la liste des billets et/ou pièces choisies par l'algorithme glouton.*

*Exercice n°4 Tester votre fonction pour résoudre le problème de Vincent. Tester ensuite avec un système de monnaie où il manquerait les billets de 5 et 10 euros et où il faut rendre 63 euros. L'algorithme glouton est-il optimal dans ce deuxième cas ? [Rappel : solution optimale lorsque l'on utilise le moins de pièces et de billet pour rendre la monnaie]*

*Exercice n°5 Pour finir tester votre programme avec le système_de_monnaie : [30, 24, 12, 6, 3, 1] et une somme de 49 euros à rendre. Est-ce optimale ? idem Q4*

*Le problème du rendu de monnaie montre que la solution optimale dépend du système de monnaie utilisé. Et donc que l'algorithme glouton ne donne pas forcément la « meilleur solution ». Le choix du système monétaire européen actuel permet justement de garantir que l'algo glouton donne dans tous les cas une solution optimale.*

# *II Remplissage du sac à dos*




*Problématique : quels objets choisir afin de maximiser « la valeur d'argent » emportée dans le sac à dos tout en ne dépassant pas les 30kg autorisés ?*

*Pour cela vous avez une liste de valeurs des 4 éléments : valeurs = [70, 40, 30, 30] et une liste de poids supposée ordonnée de la même manière : poids = [13, 12, 8, 10].*

*Chaque objet ne peut être utilisé qu'une fois.*

## Résolution exacte par la méthode de « force brute » :

*Exercice n°6 Sur cet exemple de 4 éléments, quelle est la combinaison qui permet de mettre les objets représentants la plus grande valeur d'argent dans le sac à dos. Notez les différentes solutions que vous testez sur une feuille. Avez-vous testé toutes les solutions possibles ? Si il y avait n objets, combien de possibilité y-aurait-il à tester ?*

*La solution naïve consiste à énumérer toutes les combinaisons possibles des objets puis choisir la solution optimale, c'est-à-dire celle qui maximise la valeur totale des objets emportés et dont la masse totale ne dépasse pas 30 kg. Elle conduit à coup sûr à l'optimum mais est gourmande en temps de calculs.*

### Résolution approchée par la méthode de l'algorithme glouton

*L'idée est de remplir le sac à dos avec en premier les objets les plus « efficaces ». La première problématique est de définir ce qu'est l'objet le plus efficace. Dans notre cas l'objet le plus efficace est celui qui a le plus grand prix/kilogramme.*

*Algorithme à coder pourrait être :*

- ⇨ *Définir une liste efficacite dont chaque élément est le rapport de sa valeur et de son poids.*
- ⇨ *Trier cette la liste efficacite du plus grand au plus petit élément*
- ⇨ *Sélectionner les éléments un à un dans l'ordre du tri et ajouter l'objet sélectionné dans le sac tant que le poids maximum n'est pas atteint.*

*Exercice n°7 Définir une fonction sac_a_dos (valeurs : List, poids : List , poids_max : int) qui prend en arguments une liste de poids, une liste de valeurs correspondantes et le poids maximum que l'on peut prendre dans le sac à dos, qui renvoi une liste de tuple (valeurs,poids) pris dans le sac à dos et le tuple (poids du sac , valeur totale dans le sac). [ il est conseillé de coder des fonctions intermédiaires telles que : fonction qui permet à partir de deux listes de calculer les efficacités sous forme de liste ; une fonction qui permet de trier une liste et qui renvoie les trois listes triées suivant l'efficacité ; ....]*

*Exercice n°8 Tester votre fonction pour l'exemple des 4 objets et du poids de 30 kg. Obtient-on le résultat optimal ?[rappel solution optimale : emporter les objets qui maximisent la valeur totale emportée]*

*Exercice n°9 Tester votre fonction sur d'autres exemples valeurs, poids, poids_max dans le sac à dos. Regarder à chaque fois si vous avez le résultat optimal.*

## *III Choix d'activité*

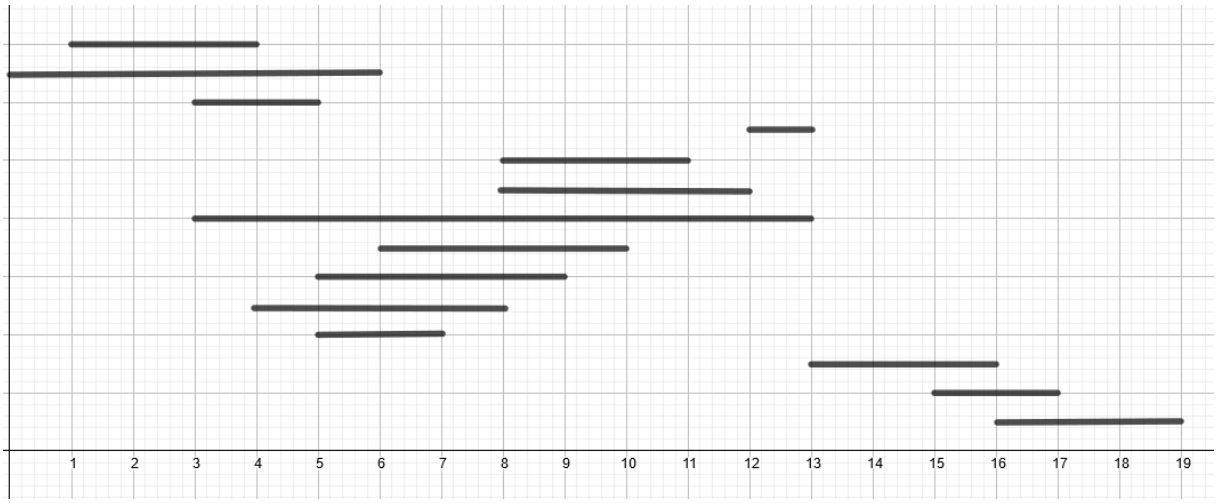
*Le problème du choix d'activités*
*On considère un ensemble d'activités, chacune possédant une date de début et une date de fin.*
*Pour des raisons logistiques, on ne peut pas programmer des activités se déroulant en même temps.*
*La question est la suivante : quel est le nombre maximal d'activités que l'on va pouvoir planifier ?*

*Considérons l'ensemble d'activités sous forme d'une liste de tuple suivante :*

*S = [(1,4),(0,6),(3,5),(12,13),(8,11),(8,12),(3,13),(6,10),(5,9),(4,8),(5,7),(13,16),(15,17),(16,19)]*

*Chaque tuple représente une activité de la forme (début, fin), le début d'une activité peut-être coïncidant avec la fin de l'activité précédente.*

*Exercice n°10 Résoudre le problème à la main à l'aide de la représentation suivante :*




*Vous avez dû trouver un nombre maximal d'activités de 6.*
*Nous allons essayer de retrouver ce résultat en mettant en place des stratégies gloutonnes. Nous allons mettre en avant par cet exemple, l'importance du choix de la stratégie gloutonne.*

*Nous allons tester 3 stratégies :*

- ⇒ *Tri par durée : on choisit les activités en commençant par celle dont la durée est minimum*
- ⇒ *Tri par date de début : on commence par prendre l'activité qui commence en premier, on choisit ensuite l'activité dont la date de début est la plus proche de la fin de la précédente, ...*
- ⇒ *Tri par date de fin : on choisi en premier l'activité qui à la date de fin la plus petite, puis celle qui à la date de fin la plus petite compatible avec la première, ...*

*L'algorithme pour les trois stratégies est toujours le même :*

- ⇒ *Trier la liste d'activités S par rapport au choix glouton.*
- ⇒ *Initialiser une liste C des activités à choisir vide.*
- ⇒ *Ajouter à C la première activité de la liste triée S.*
- ⇒ *Parcourir dans l'ordre la liste S des activités, et rajouter l'activité courante C si elle n'est pas incompatible avec celles déjà planifiées.*

*Exercice n°11 Réaliser pour les stratégies gloutonnes énumérées un script python en suivant l'algorithme proposé.*

*Exercice n°12 Quelle stratégie gloutonne semble répondre à la problématique de maximiser le nombre d'activités à planifier ?*

## *Pour la prochaine fois*

# *IV Stations-services*

*Le problème*

- ⇒ *Vous souhaitez vous rendre de Nancy à Brest (ou une autre ville de votre choix !! ) en voiture électrique*
- ⇒ *Votre batterie vous permet de rouler R Km*
- ⇒ *Vous connaissez la liste des bornes à charges rapides disponibles sur la route, donnée sous la forme d'une liste S=[d1 , d2 , ..., dk] où chaque di donne la distance qui le sépare de son précédent*

*S[0] à d1 kilomètres du départ*
*S[1] à d2 kilomètres de s[1]*
*Etc.*
*On suppose di<= R pour i=1..k, dk symbolisant l'arrivée*
*On souhaite faire le moins d'arrêts possibles*

*Exercice n°13 Écrire un algorithme glouton résolvant le problème, c'est à dire renvoyant la liste des indices des bornes à charges rapides où l'on doit s'arrêter.*

*Exercice n°14 Pour une autonomie de votre voiture de 250 Km, tester avec la liste de borne [120,142,90,70,130,150,84,25,77] cela doit renvoyer : [ 0, 2, 4, 6]*