

Activité : « Packing un exemple d’algorithme glouton »

Objectifs principaux de la séance :

1. Découvrir la notion d’algorithme glouton
2. Appliquer en totalité une démarche de conception d’un algorithme glouton (de la phase « idée » à la langage naturel).
3. Comparer différents algorithmes tant dans leur complexité que dans leur efficacité selon la nature des données (la taille, la répartition).

Prérequis / place de la séance dans la progression :

- Une première séance de 2h sur « La démarche complète pour écrire un algorithme (idée/langage naturel) » sur 2 exemples simples (algorithme de recherche d’une occurrence dans un tableau + algorithme de recherche d’un extremum puis de deux extremums) – les notions de correction, terminaison et complexité auront été abordées sur ces exemples.
- Une deuxième séance de 2h sur « Les algorithmes de tris par insertion, par sélection » avec écriture, terminaisons à justifier, coût quadratique dans le pire des cas.
- Une troisième séance de 2h sur un algorithme des K plus proches voisins et recherche dichotomique dans un tableau trié.

La séance que nous vous proposons sera la dernière spécifiquement centrée sur l’algorithmique : elle abordera les algorithmes gloutons.

Python sera déjà bien maîtrisé, notamment les listes, les fonctions.(apprentissage spiralé sur l’année).

Articulation de la séance :

1. Activité de découverte
2. Séance cours algorithme glouton, algorithme exacte.
3. Exercices sur d’autres algorithmes gloutons

Points du programme abordés lors de ce TP :

- Programme de 1ère : « Algorithmes gloutons : résoudre un problème avec un algorithme glouton »
- Programme de Terminale : la récursivité dans un programme Python

Nous possédons 5 poids $L=[5 \text{ kg}, 7 \text{ kg}, 8 \text{ kg}, 2 \text{ kg}, 4 \text{ kg}]$ et une grande quantité de sacs tous identiques ne pouvant supporter un poids maximal de 10kg.

Combien faudra-t-il de sacs au minimum pour y ranger l'ensemble des 5 objets ?

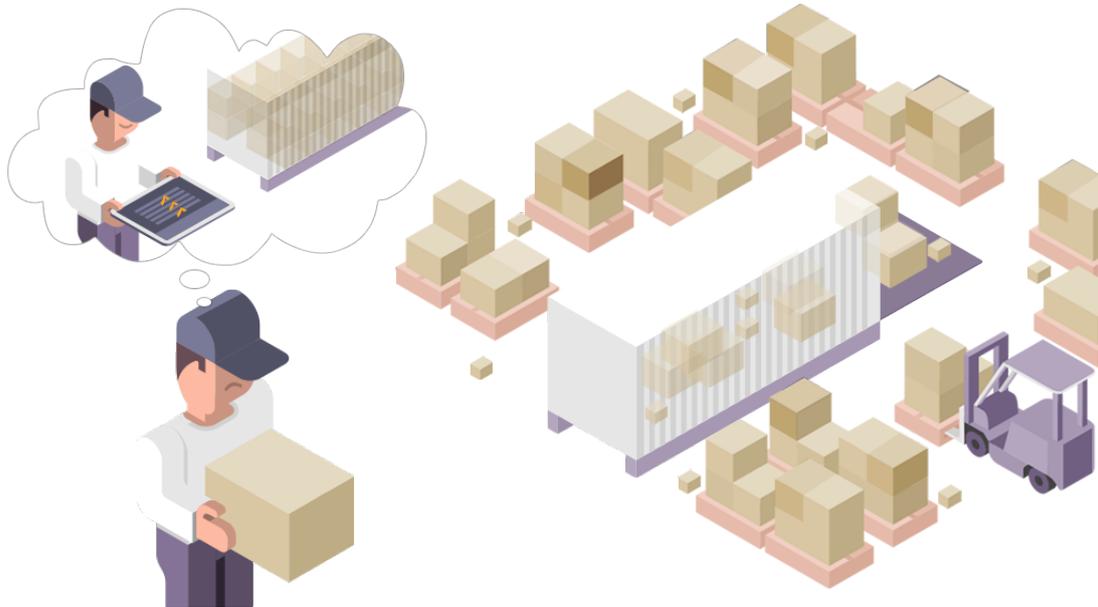


Illustration avec des cartons et des objets à y ranger

1. Que proposez-vous ?

Mise en oeuvre : le professeur donne 5 verres en carton sur lesquels les poids sont indiqués ainsi qu'un certain nombre d'assiettes identiques représentant les sacs (un nombre assez grand!). Les élèves individuellement expérimentent.

Idées de propositions des élèves/ réponses prof :

- 1 poids par sac : c'est une solution facile, réalisable mais non optimale, pourquoi? Il faut que qq propose une solution plus optimale. Et si on a beaucoup de poids, cette méthode donnera beaucoup de sacs! Par contre, c'est probablement le nombre de sacs maximum. Demander aux élèves pour quel type de liste de poids cela sera probablement la solution optimale : la réponse est $L=[10 \text{ kg}, 10 \text{ kg}, 10 \text{ kg}, 10 \text{ kg}, 10 \text{ kg}]$ mais aussi $L=[9 \text{ kg}, 9 \text{ kg}, 9 \text{ kg}, 9 \text{ kg}, 9 \text{ kg}]$... en fait, toutes celles pour lesquels tous les poids dépassent 5 Kg strictement.
- La technique du nombre de "sac de sable" : calculer la somme des poids, ici 26, et diviser par 10kg afin d'avoir un nombre de sacs minimal ici 2.6 sacs : il devrait être possible de faire avec 3 sacs : en effet voici une solution

n° 1	n° 2	n° 3
5		8
4	7	2

Cette méthode nous donne une idée du nombre de sacs avec lequel on devrait pouvoir trouver une solution.

Est-ce la meilleure? Probablement non mais cela nous fournira une indication.

- Certains, auront peut être l'idée de modéliser par un arbre pour visualiser une possibilité voir toutes les possibilités (type de modélisation vue en maths en probabilités) . Discuter de la longueur de cette méthode!
- Certains proposeront l'idée naïve de remplir sac par sac en suivant l'ordre des poids tels qu'ils apparaissent.
- D'autres diront qu'il faut être plus malin et sommer les poids pour obtenir 10kg ou qq chose de proche cad remplir les sacs un à un : $8 + 2 = 10$ puis $5 + 4 = 9$ puis 7 tout seul.

n° 1	n° 2	n° 3
2	4	
8	5	7

- D'autres diront qu'il vaut mieux commencer par placer le poids le plus lourd et garder cette technique tout le long.
- Certains auront sûrement l'idée de commencer par le poids le plus élevé : remplir d'abord le premier sac avec ce poids puis prendre le deuxième poids le plus lourd, le placer dans un nouveau sac car il ne rentre pas dans le premier etc. C'est le principe du First Fit Decreasing (FFD) ce qui donnera

n° 1	n° 2	n° 3
2		4
8	7	5

Bilan fait par le professeur :

L'algorithme consistant à mettre un poids par sac fournit un nombre de sac maximum. L'algorithme du nombre de sac minimal nous fournit une indication mais n'est pas très concluant. Plusieurs pbs dans ces méthodes qui n'en font en fait qu'une :

- selon quel critère choisit-on le poids au départ?
 - selon critère choisit-on de mettre les poids d'après?
2. Par groupe de 2, choisissez un critère et écrire en langage naturel l'algorithme que vous avez choisi.
 3. Prenez maintenant deux poids de plus (c'est à dire deux verres de plus) de poids respectifs 5kg et 6 kg. Votre liste de poids est maintenant de taille 7 : $L=[5 \text{ kg}, 7 \text{ kg}, 8 \text{ kg}, 2\text{kg}, 4 \text{ kg}, 5 \text{ kg}, 6 \text{ kg}]$.

Consigne Prof : Afin de voir si votre algorithme est correct, un élève prend le matériel en mains, se met dos à l'autre et exécute les commandes données par son camarade. Avec FFD ils obtiennent :

n° 1	n° 2	n° 3	n° 4
2		4	5
8	7	6	5

Votre algorithme vous fournit-il une solution valide?

Bilan de l'activité de découverte :

Pour résoudre un problème d'optimisation, on peut faire appel à un algorithme. Celui fournira une solution qui sera dite valide lorsqu'elle solutionne le problème . Cette solution valide sera dite optimale lorsqu'il n'y en a pas de meilleure. Pour prouver qu'une solution n'est pas optimale, il suffit d'en trouver une meilleure!

Il existe plusieurs algorithmes pour répondre au problème posé : les meilleurs sont ceux donnant la réponse optimale qui est 3 sacs! Parmi ces algorithmes, certains sont caractérisés de gloutons lorsqu'ils fonctionnent de manière itérative en définissant un critère de choix identique à chaque étape semblant sur le moment optimal.

1 L'algorithme glouton

Généralisation du problème :

On considère un ensemble de sacs identiques en assez grande quantité, ne pouvant supporter qu'un poids maximal de 20kg. On considère aussi un ensemble de n objets de masses différentes mais connues inférieures ou égales à 10kg :

$$L = [p_1, p_2, p_3, \dots, p_n]$$

Question : Combien faudra-t-il de sacs au minimum pour y ranger l'ensemble des n objets ?

Le problème a une seule contrainte : le poids de chaque sac ne doit pas dépasser 10 kg.

Notations :

— Chaque objet sera numéroté i et aura donc une masse p_i connue.

— On notera 1 le premier sac, 2 le deuxième etc.

On appellera **solution** du problème la donnée d'une liste

$$S = [[\text{poids du sac 1}], [\text{poids du sac 2}], \dots]$$

Une **solution** est dite **valide** lorsqu'elle répond au problème : parmi celles qui le sont, certaines sont dites **optimales** si elles donnent le nombre de sacs minimum.

C'est un problème d'**optimisation** puisqu'il s'agit de minimiser le nombre de sacs utilisés.

Il existe deux grands types de méthodes de résolution des problèmes d'optimisation

— les méthodes **exactes** c'est à dire permettant d'obtenir une solution optimale à chaque fois.

— les méthodes **heuristiques ou approchées** c'est à dire celles permettant d'obtenir une solution mais non nécessairement optimale.

A retenir .

Un algorithme **glouton** comprend :

1. une construction itérative
2. un critère de choix s'appliquant à l'identique à chaque étape
3. un test d'arrêt

C'est le fait que le choix se fasse de manière « localement optimale » (c'est à dire à chaque étape) qui caractérise la gloutonnerie de l'algorithme.

Dans l'algorithme First Fit Decreasing (FFD) le critère glouton est : on place l'objet le lourd dans le premier sac pouvant le contenir.

a) Déterminer les 3 composantes de l'algorithme FFD.

- b) Voici l'algorithme glouton programmé en langage Python : compléter le test. fichier .py joint et reproduit ici pour l'exposé.

```
def nouvelobjet (t, t2, masse_max) :           # Ajoute l'objet d'indice 0 de la liste t
                                                # a un sac de la liste t2

    i =0
    while i<=len(t2)-1 and t2[i]+t[0] > masse_max :
        i += 1
        if i>len(t2)-1 :
            # a completer
        else:
            # a completer
    return (t,t2)

def FFD(t, masse_max) :
    t.sort(reverse=True)
    t2=[]           # liste creee pour recevoir les objets venant de t
    t2.append(t[0]) # le premier element de t2 recoit le premier objet de t
    t.pop(0)       # on supprime de la liste t le premier element qui vient
                  # d etre range dans t2

    while len(t)!= 0 :
        t, t2 = nouvelobjet (t,t2,masse_max)
    return (t, t2)

# t liste des objets
# masse_max : masse maximale que peut contenir un sac"
t=[5,4,8,2,3,2,6]
masse_max = 10
t,t2 = FFD (t, masse_max)
print(len(t2))
```

Vérifiez qu'il vous retourne une solution valide trouvée lors de l'activité de recherche $L = [5, 7, 8, 2, 9]$

- c) Le tester avec $L=[7, 3, 8, 9, 3, 9]$ et une masse maximale pour un sac de 20kg : votre solution est-elle optimale?

Prof : Ce n'est pas le cas puisque le programme retourne 3 sacs : $[[9,9],[8,7,3],[3]]$ alors que $[[9,7,3],[9,8,3]]$ est une meilleure solution donc notre algorithme glouton ne donne pas toujours une solution optimale.

- d) Nous voulons maintenant tester l'algorithme avec plus de d'objets : $n = 100$. Dans votre programme Python, remplacer t par $t=[\text{randint}(1,10) \text{ for } i \text{ in } \text{range}(100)]$ et une masse maximale pour un sac de 20kg : qu'en concluez-vous ?

Attention, n'oubliez pas d'importer randint de la bibliothèque random !

Prof : Même si le nombre d'objets augmente, cet algo reste efficace puisqu'il fournit très rapidement une solution valide. C'est un algorithme correct.

- e) Comparer le nombre de sacs obtenu précédemment au nombre de « sacs de sable ». Qu'en concluez-vous ?
- f) A la vue du programme en langage Python, évaluez la complexité de cet algorithme en fonction du nombre n d'objets.

2 Méthode exacte

La question qui se pose est de savoir s'il est « envisageable » de chercher une solution optimale. On se propose donc d'explorer toutes les configurations possibles en commençant par le nombre minimal de sacs (sac de sable).

Dans la suite la capacité des sacs est fixée à 20 et le poids des objets ne dépasse pas 10 kg.

1. Proposer une donnée de 30 poids qui demande « le moins de calcul ».
2. Proposer une donnée de 30 poids qui demande « le plus de calcul ».
3. La fonction `listeCombinaison` ci-dessous génère le nombre de combinaisons pour `nbsac` nombre de sacs et `nbObjet` nombre d'objets :

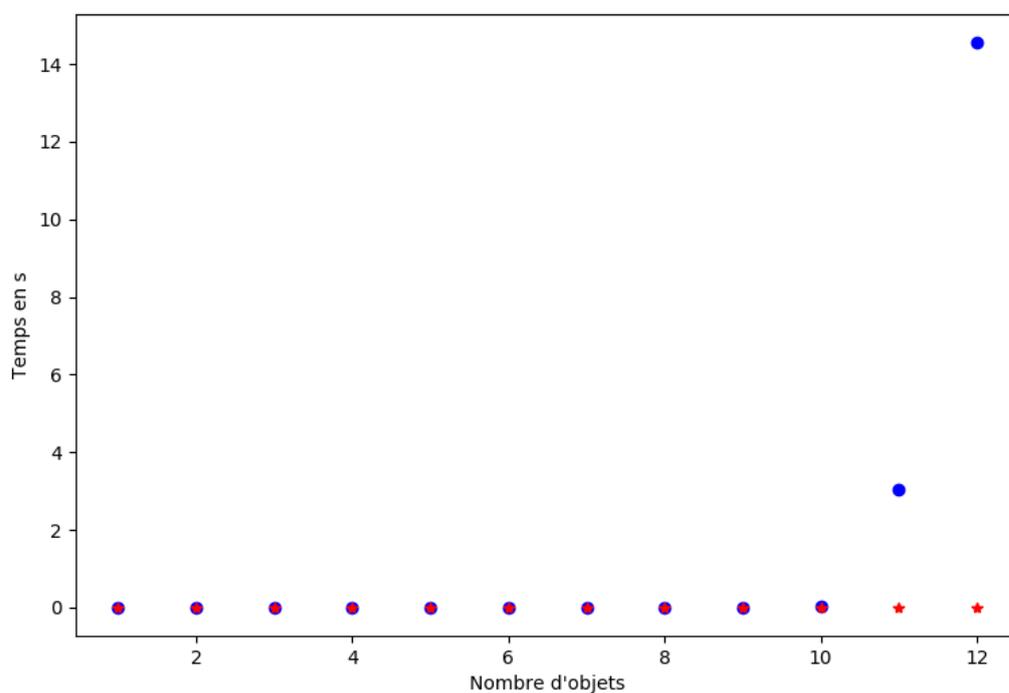
```
def produit2Listes(l, m):
    prod=[]
    for i in l:
        for j in m:
            prod.append(i+j)
    return prod

def puissanceListe(l, n):
    """ Donne les combinaisons avec repetition de n elements de la liste l """
    if n == 1:
        return l
    else:
        listeAvant=puissanceListe(l,n-1)
        return(produit2Listes(l,tuple(listeAvant)))

def listeCombinaison(nbsac, nbObjet):
    l=[[i] for i in range(1,nbsac+1)]
    return puissanceListe(l,nbObjet)
```

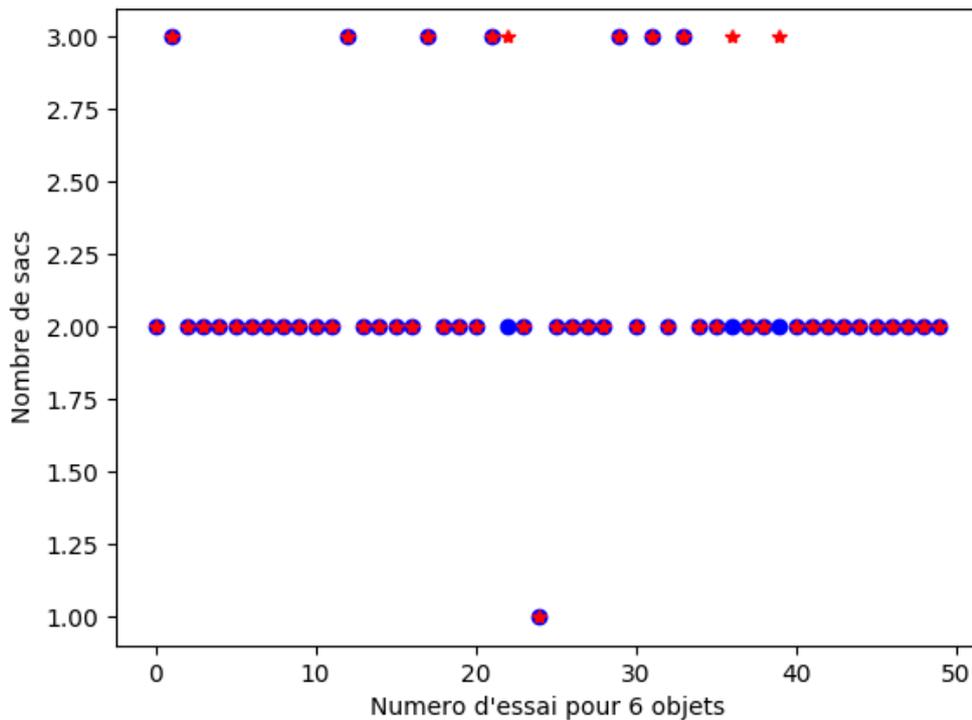
Donner le nombre de combinaisons dans les deux cas précédents des questions précédentes. Puis testez-la pour différentes valeurs (plus petites).

4. Voici un graphique qui compare l'algorithme FFD (rouge) et un algorithme exact (bleu).



Qu'en pensez-vous?

5. Bien entendu la question qui se pose alors est de savoir si l'algorithme exact est plus « efficace » au sens que FFD donne des solutions souvent très éloignées de l'optimal. Voici un graphique.



Qu'en pensez-vous ?

6. rechercheOptimal donne la première solution optimale. Comparer FFD et cet algorithme avec $L = [7, 3, 8, 9, 3, 9]$.

Pouvez-vous trouver un autre cas où les résultats diffèrent ?

Nous pouvons donc conclure que cette méthode est intéressante car elle fournit toujours des solutions optimales mais qu'elle est peu efficace en terme de temps !

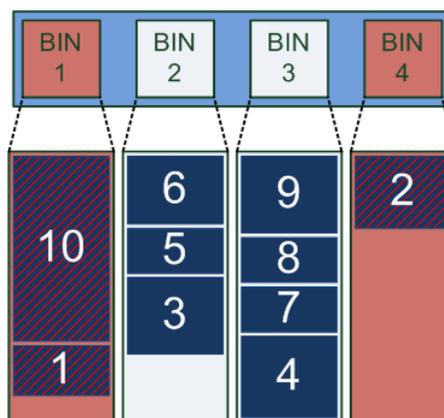
3 D'autres algorithmes

Sachez qu'il existe bien d'autres algorithmes gloutons résolvant ce problème :

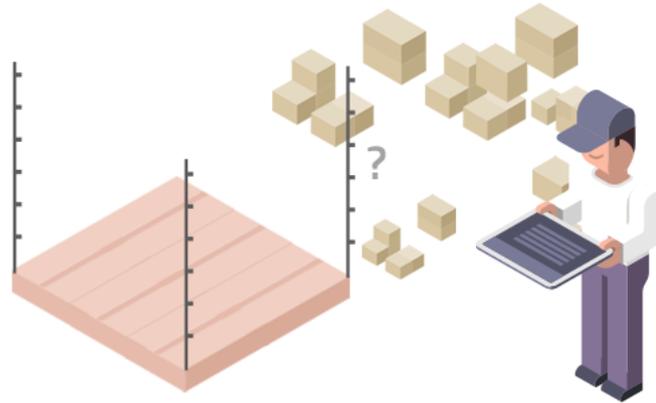
- Le Worst Fit Decreasing qui dispose lui aussi les objets les uns après les autres mais avec le critère glouton de mettre l'objet dans le sac le moins lourd à ce moment.
- Le Best Fit Decreasing qui dispose lui aussi les objets les uns après les autres mais avec le critère glouton de mettre l'objet dans le sac le plus lourd à ce moment.

Ce problème de Bin Packing trouve son utilité dans de nombreux domaines :

- en informatique : le rangement des fichiers dans un support informatique, le transport de données.



— dans les transports de marchandises : remplissage de camions avec comme des contraintes sur le poids des marchandises, pas sur le nombre de camions.



Dans tous ces domaines, peu importe que la solution fournie soit optimale (à condition qu'elle n'en soit pas trop loin) à condition qu'elle soit valide et obtenue rapidement !

Exercice 1. (Rendu de monnaie)

On dispose d'une quantité suffisante de pièces de 1, 2, et 5 dirham. On souhaite rendre N dirham en donnant le nombre minimal de pièces.

- Définir :
 - l'étape d'itération
 - les choix possibles
 - le critère du « meilleur » choix
 - le test d'arrêt
- Ecrire sur papier un algorithme glouton reprenant les critères précédents pour répondre au problème.
- Tester l'algorithme avec $N = 23$ toujours sur une feuille de papier
- Tester sur papier l'algorithme avec $N = 23$ mais un jeu de pièce de 5 et 6 dirhams.
- Ecrire cet algorithme en python.

Exercice 2. (Ramasseur de balle)

Un ramasseur de balle doit ramasser un ensemble de balles sur un terrain de tennis et souhaite parcourir la plus petite distance possible.

On propose comme critère de meilleur choix : le ramasseur de balles va jusqu'à la balle la plus proche de lui et la ramasse.

- Ce critère est-il le meilleur choix ? Proposer un schéma pour lequel ce n'est pas le cas.
- Proposer un critère permettant d'améliorer l'algorithme.

Exercice 3.

On considère le code ci-dessous :

- définir :
 - l'étape d'itération
 - les choix possibles
 - le critère du « meilleur » choix
 - le test d'arrêt

Que fait ce code sur le jeu de données : [5, 7, 8, 2, 4]

Algorithme Exercice 3.

```
from random import randint
masse_max=10
masses=[5,7,8,2,4]
sac=[]

for i in range(len(masses)-1):
    indice_sac=0
    indice=random.randint(0,len(masses)-1)
    while sum(sac[indice_sac])+masses[indice]>masse_max:
        if indice_sac>=len(sac)-1:
            sac.append([])
        indice_sac+=1
    sac[indice_sac].append(masses[indice])
    masses.pop(indice)
print(sac)
print('nb de sac',len(sac))
```

Bilan « bin packing »

Énoncé du problème traité :

On considère un ensemble de sacs identiques et un ensemble d'objets dont on connaît le poids. Sachant que les sacs ne peuvent supporter qu'un poids maximum, combien faudra-t-il au minimum de sacs pour y ranger l'ensemble des objets considérés ?

Traitement du problème :

Pour résoudre ce problème, on a utilisé un **algorithme glouton** qui n'est pas toujours optimal mais qui permet d'obtenir de bons résultats en pratique.

Notre algorithme : **FFT (First Fit Decreasing)**

1. on trie la liste des n objets par ordre décroissant de poids ;
2. *first-fit*, on prend les objets les uns après les autres et pour chacun, on le place dans le premier sac pouvant le contenir

Remarque : la solution exacte de ce problème nécessiterait un algorithme de complexité d'ordre n^p (ce problème est dit « NP complet », alors que l'algorithme glouton a une complexité d'ordre maximal n^2).

Qu'est-ce qu'un **algorithme glouton** ?

Algorithme **simple** et **efficace** caractérisé par :

- son mode **itératif**

Les objets passent de la liste initiale à la liste finale un par un.

- une liste de choix possibles et un ou **des critères de choix** appliqués de manière systématique à chaque itération

Critère « FFD » : on place chaque objet du plus lourd au plus léger dans le premier sac pouvant le contenir.

- un **test d'arrêt**

Test sur la taille de la liste initiale contenant les objets qui est amputée à chaque itération de son premier élément

Que renvoie un algorithme glouton ?

En fonction du problème, des données initiales et des critères choisis, un algorithme glouton peut renvoyer :

- une **solution valide ou non**,
- une **solution optimale ou non**

Pour certains problèmes, il existe un algorithme glouton dont on peut faire la preuve qu'il est optimal, pour d'autres problèmes, aucun algorithme glouton n'est optimal.

Des applications du « packing » dans la vie :

En une dimension pour le rangement de fichiers sur un support informatique, pour la découpe de câbles, pour le remplissage de camions ou de containers avec une seule contrainte.