

Algorithmes gloutons

Fiche de synthèse

① PROBLÈMES D'OPTIMISATION

Un problème d'**optimisation** cherche la meilleure solution parmi de nombreuses possibilités : rendre la monnaie avec le moins de pièces, remplir un sac de la plus grande valeur. Essayer toutes les solutions est vite impossible : avec n objets, il y a 2^n choix, plus d'un milliard pour 30 objets.

Définition 1 (Algorithme glouton) Un **algorithme glouton** fait à chaque étape le choix qui semble le meilleur sur le moment, le **choix localement optimal**, sans jamais revenir en arrière.

☰ Schéma d'un algorithme glouton.

1. Trier les éléments selon un critère.
2. Les parcourir dans cet ordre.
3. Prendre chaque élément s'il respecte les contraintes.
4. Ne jamais revenir sur un choix.

Un algorithme glouton est simple et rapide, souvent en $O(n \log n)$ à cause du tri ; mais son résultat n'est pas toujours optimal.

② LE RENDU DE MONNAIE

```
def rendu_monnaie(montant, pieces):
    pieces_triees = sorted(pieces, reverse=True)
    resultat = []
    reste = montant
    for piece in pieces_triees:
        while reste >= piece:
            resultat.append(piece)
            reste -= piece
    return resultat
```

```
systeme = [200, 100, 50, 20, 10, 5, 2, 1]
rendu_monnaie(678, systeme)
# [200, 200, 200, 50, 20, 5, 2, 1]
```

On prend à chaque fois la plus grosse pièce possible. Avec les pièces en euros, le résultat est toujours optimal. Avec un système $[6, 4, 1]$ et 8 centimes, le glouton rend $6 + 1 + 1$, trois pièces, alors que $4 + 4$ n'en demande que deux : le meilleur choix local ne mène pas au meilleur résultat global.

③ LE SAC À DOS

Un sac supporte 15 kg ; on veut emporter la plus grande valeur.

Objet	1	2	3	4	5	6	7
Valeur	500	1000	1500	700	800	900	400
Poids	1	3	5	4	1	2	2

Critère de tri	Valeur emportée
valeurs décroissantes	4 900
poids croissants	4 300
rapport valeur/poids décroissant	5 100

Le rapport valeur/poids donne ici le meilleur résultat.

- **Sac à dos fractionnaire**, où l'on peut prendre une partie d'un objet : le glouton par rapport valeur/poids est toujours optimal.
- **Sac à dos 0/1**, où chaque objet est pris en entier ou pas du tout : il ne l'est plus. Avec A (165, 3 kg), B (100, 2 kg) et C (100, 2 kg) et un sac de 4 kg, le glouton prend A seul, 165 ; B et C valent 200. La programmation dynamique, vue en terminale, résout ce problème.

```
def rapport(o):
    return o[1] / o[2]

def sac_a_dos_01(poids_max, objets):
    tries = sorted(objets, key=rapport,
                    reverse=True)
    poids_actuel = 0
    sac = []
    for nom, valeur, poids in tries:
        if poids_actuel + poids <= poids_max:
            sac.append(nom)
            poids_actuel += poids
    return sac
```

Chaque objet est un tuple (nom, valeur, poids) ; `sorted` les range par rapport valeur/poids décroissant.

📌 À retenir.

- Un glouton fait à chaque étape le meilleur choix local, sans revenir en arrière.
- Il est simple et rapide, mais pas toujours optimal.
- Rendu de monnaie en euros et sac à dos fractionnaire : optimal ; sac à dos 0/1 : pas toujours.