Algorithmes gloutons

Résumé

Face à un problème d'optimisation (rendre la monnaie avec le moins de pièces, remplir un sac au mieux, planifier un emploi du temps), une stratégie naturelle consiste à faire, à chaque étape, le choix qui semble le plus avantageux sur le moment. C'est le principe de l'algorithme **glouton** : on avance pas à pas, en prenant toujours le meilleur choix immédiat, sans jamais revenir en arrière. Cette approche est souvent rapide et simple, mais elle ne donne pas toujours la meilleure solution possible. Savoir quand un glouton fonctionne (et quand il échoue) est une compétence fondamentale en algorithmique.

Prérequis : Les listes, Tris naïfs.

① Mémo

Définition 1 (Principe général)

Un **algorithme glouton** construit une solution étape par étape en faisant, à chaque étape, le choix qui paraît le meilleur **localement**, sans jamais revenir en arrière.

Avantage Simple à concevoir, souvent rapide (complexité faible).
Inconvénient Ne garantit pas toujours la solution optimale.
Quand ça marche ? Lorsque le problème possède la **propriété de choix glouton** : un choix localement optimal mène toujours à une solution globalement optimale.

Le problème du rendu de monnaie

Objectif : rendre une somme S avec le minimum de pièces, à partir d'un système de pièces donné.
Stratégie gloutonne : à chaque étape, choisir la plus grande pièce inférieure ou égale au montant restant.
Exemple : rendre 37 centimes avec les pièces $\{1,\ 2,\ 5,\ 10,\ 20,\ 50\}$:

- $37 \geqslant 20 \rightarrow$ on prend 20, reste 17 ;
- $17 \geqslant 10 \rightarrow$ on prend 10, reste 7 ;
- $7 \geqslant 5 \rightarrow$ on prend 5, reste 2 ;
- $2 \geqslant 2 \rightarrow$ on prend 2, reste 0.

Résultat : quatre pièces. Avec le système euro, l'algorithme glouton donne toujours l'optimum.

Quand le glouton échoue

Avec un système de pièces non standard, l'algorithme glouton peut échouer.
Exemple : pièces $\{1,\ 3,\ 4\}$, rendre 6.

- Glouton : $4 + 1 + 1 = 6$ (trois pièces).
- Optimal : $3 + 3 = 6$ (deux pièces).

Le glouton a choisi la plus grande pièce (4) alors qu'il fallait prendre deux pièces de 3. Il faut dans ce cas utiliser la **programmation dynamique**.

Le problème du sac à dos (version fractionnaire)

Objectif : remplir un sac de capacité C avec des objets ayant chacun un poids et une valeur, en maximisant la valeur totale. On peut prendre une fraction d'un objet.
Stratégie gloutonne : trier les objets par rapport valeur/poids décroissant, puis prendre les objets dans cet ordre tant que le sac n'est pas plein.
Cette stratégie donne l'optimum **uniquement** dans la version fractionnaire. Dans la version 0/1 (on prend l'objet en entier ou pas du tout), le glouton ne garantit pas l'optimum.

Le problème de l'ordonnancement d'activités

Objectif : choisir le maximum d'activités compatibles (qui ne se chevauchent pas) parmi un ensemble d'activités ayant chacune une heure de début et une heure de fin.
Stratégie gloutonne : trier les activités par heure de fin croissante, puis sélectionner chaque activité dont l'heure de début est postérieure ou égale à la fin de la dernière activité sélectionnée.
Ce glouton donne **toujours** l'optimum (prouvable par échange).

Pièges fréquents

- Croire qu'un algorithme glouton donne toujours la solution optimale : c'est faux en général.
- Appliquer le glouton du rendu de monnaie sans vérifier que le système de pièces le permet.
- Confondre glouton et force brute : le glouton ne teste pas toutes les possibilités.
- Oublier de trier les données avant d'appliquer la stratégie gloutonne.

Erreurs classiques

Erreur	Correction	Explication
Trier par valeur décroissante pour le sac à dos	Trier par rapport valeur/poids décroissant	Un objet lourd et cher peut être moins rentable qu'un objet léger et bon marché.
Ne pas vérifier que la pièce est $\leqslant$ au reste	Ajouter `if pieces[i] <= reste`	Sans cette vérification, on tente de rendre plus que le montant restant.
Utiliser le glouton avec les pièces $\{1, 3, 4\}$ pour rendre 6	Utiliser la programmation dynamique	Le glouton donne trois pièces $(4 + 1 + 1)$ au lieu de deux $(3 + 3)$.
Trier les activités par durée	Trier par heure de fin	Trier par durée ne donne pas l'optimum : une activité courte peut chevaucher plusieurs autres.

② Exemples

Programme 1 · Rendu de monnaie glouton

```python
def rendu_monnaie(montant, pieces):
    """Renvoie la liste des pièces utilisées (stratégie gloutonne)."""
    pieces_triees = sorted(pieces, reverse=True)
    resultat = []
    reste = montant
    for p in pieces_triees:
        while reste >= p:
            resultat.append(p)
            reste -= p
    return resultat

# Système euro
print(rendu_monnaie(37, [1, 2, 5, 10, 20, 50]))
# [20, 10, 5, 2]

# Système pathologique
print(rendu_monnaie(6, [1, 3, 4]))
# [4, 1, 1]  -> 3 pièces (non optimal : 3+3 = 2 pièces)
```

Programme 2 · Ordonnancement d'activités

```python
def activites_max(activites):
    """Renvoie la liste des activités sélectionnées (compatibles).
    activites : liste de tuples (debut, fin)."""
    # Trier par heure de fin croissante
    triees = sorted(activites, key=lambda a: a[1])
    selection = [triees[0]]
    for i in range(1, len(triees)):
        if triees[i][0] >= selection[-1][1]:
            selection.append(triees[i])
    return selection

activites = [(1, 3), (2, 5), (3, 6), (5, 7), (6, 8), (8, 10)]
print(activites_max(activites))
# [(1, 3), (3, 6), (6, 8), (8, 10)] -> 4 activités
```

Programme 3 · Sac à dos fractionnaire

```python
def sac_a_dos_fractionnaire(capacite, objets):
    """objets : liste de tuples (poids, valeur).
    Renvoie (valeur maximale, détail des choix) - objets fractionnables."""
    # Trier par rapport valeur/poids décroissant
    tries = sorted(objets, key=lambda o: o[1]/o[0], reverse=True)
    valeur_totale = 0
    reste = capacite
    choix = []
    for poids, valeur in tries:
        if reste >= poids:
            choix.append((poids, valeur, 1.0))
            valeur_totale += valeur
            reste -= poids
        elif reste > 0:
            fraction = reste / poids
            choix.append((poids, valeur, fraction))
            valeur_totale += valeur * fraction
            reste = 0
    return valeur_totale, choix

objets = [(10, 60), (20, 100), (30, 120)]
val, detail = sac_a_dos_fractionnaire(50, objets)
print(f"Valeur maximale : {val}")  # 240.0
```

③ Exercices

Exercice 1 *Rendu de monnaie*

1. Appliquer l'algorithme glouton à la main pour rendre 63 centimes avec le système $\{1, 2, 5, 10, 20, 50\}$. Combien de pièces sont nécessaires ?
2. Même question avec le système $\{1, 7, 10\}$ pour rendre 14. Le glouton donne-t-il l'optimum ?
3. Écrire une fonction `rendu_monnaie(montant, pieces)` qui implémente l'algorithme glouton et renvoie la liste des pièces utilisées.

Exercice 2 *Ordonnancement d'activités* Un élève souhaite participer au maximum d'activités dans la journée. Voici les créneaux proposés :

Activité	Début	Fin
Piscine	8h	10h
Escalade	9h	11h
Tennis	10h	12h
Cinéma	11h	14h
Vélo	13h	15h
Lecture	14h	16h

1. Trier les activités par heure de fin. Appliquer l'algorithme glouton à la main. Quelles activités sont sélectionnées ?
2. Implémenter cet algorithme en Python.

Exercice 3 *Sac à dos fractionnaire* Un randonneur a un sac de capacité 15 kg. Il dispose des objets suivants :

Objet	Poids (kg)	Valeur	Rapport valeur/poids
A	5	30	?
B	10	50	?
C	8	40	?
D	3	24	?

1. Calculer le rapport valeur/poids de chaque objet et les trier.
2. Appliquer l'algorithme glouton fractionnaire à la main. Quelle est la valeur maximale ?
3. Implémenter l'algorithme en Python.

Solutions des exercices

Corrigé de l'exercice 1.

1. Glouton pour 63 avec $\{1, 2, 5, 10, 20, 50\}$:
   - 50 (reste 13), 10 (reste 3), 2 (reste 1), 1 (reste 0).

   Résultat : quatre pièces $\{50, 10, 2, 1\}$. C'est l'optimum pour le système euro.
2. Glouton pour 14 avec $\{1, 7, 10\}$:
   - 10 (reste 4), 1 (reste 3), 1 (reste 2), 1 (reste 1), 1 (reste 0).

   Résultat glouton : cinq pièces $\{10, 1, 1, 1, 1\}$. Or $7 + 7 = 14$ ne nécessite que deux pièces. Le glouton ne donne **pas** l'optimum ici.
3. Implémentation :

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

Corrigé de l'exercice 2.

1. Activités triées par heure de fin : Piscine (8-10), Escalade (9-11), Tennis (10-12), Cinéma (11-14), Vélo (13-15), Lecture (14-16).

   Application du glouton (on compare toujours avec la **dernière activité sélectionnée**) :
   - Piscine (8-10) : sélectionnée (première activité). Dernière sélectionnée : Piscine (fin 10h).
   - Escalade (9-11) : rejetée (début 9h $<$ fin 10h de Piscine).
   - Tennis (10-12) : sélectionnée (début 10h $\geqslant$ fin 10h de Piscine). Dernière sélectionnée : Tennis (fin 12h).
   - Cinéma (11-14) : rejeté (début 11h $<$ fin 12h de Tennis).
   - Vélo (13-15) : sélectionné (début 13h $\geqslant$ fin 12h de Tennis). Dernière sélectionnée : Vélo (fin 15h).
   - Lecture (14-16) : rejetée (début 14h $<$ fin 15h de Vélo).

   Résultat : **Piscine, Tennis, Vélo** (trois activités).
2. Implémentation :

```python
def activites_max(activites):
    triees = sorted(activites, key=lambda a: a[1])
    selection = [triees[0]]
    for i in range(1, len(triees)):
        if triees[i][0] >= selection[-1][1]:
            selection.append(triees[i])
    return selection

activites = [(8, 10), (9, 11), (10, 12), (11, 14), (13, 15), (14, 16)]
print(activites_max(activites))
# [(8, 10), (10, 12), (13, 15)]
```

Corrigé de l'exercice 3.

1. Rapports : A = 30/5 = 6, B = 50/10 = 5, C = 40/8 = 5, D = 24/3 = 8.
   Tri décroissant par rapport : D (8), A (6), B (5), C (5).
2. Application du glouton (capacité 15 kg) :
   - D : poids 3 kg $\leqslant$ 15, on le prend en entier. Valeur : 24. Reste : 12 kg.
   - A : poids 5 kg $\leqslant$ 12, on le prend en entier. Valeur : 24 + 30 = 54. Reste : 7 kg.
   - B : poids 10 kg > 7, on prend 7/10 = 0,7 de B. Valeur : $54 + 50 \times 0{,}7 = 54 + 35 = 89$.

   Valeur maximale : **89**.
3. Implémentation :

```python
def sac_a_dos_fractionnaire(capacite, objets):
    tries = sorted(objets, key=lambda o: o[1]/o[0], reverse=True)
    valeur_totale = 0
    reste = capacite
    for poids, valeur in tries:
        if reste >= poids:
            valeur_totale += valeur
            reste -= poids
        elif reste > 0:
            valeur_totale += valeur * (reste / poids)
            reste = 0
    return valeur_totale

objets = [(5, 30), (10, 50), (8, 40), (3, 24)]
print(sac_a_dos_fractionnaire(15, objets))  # 89.0
```