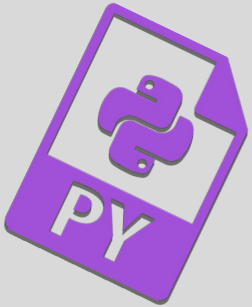
2022 - MELLAZA

L'ALGORITHME GLOUTON



L'Art d'optimiser la résolution d'un problème.

INTRODUCTION

La résolution d'un problème d'optimisation se fait généralement par étapes : **à chaque étape, on doit faire un choix.**

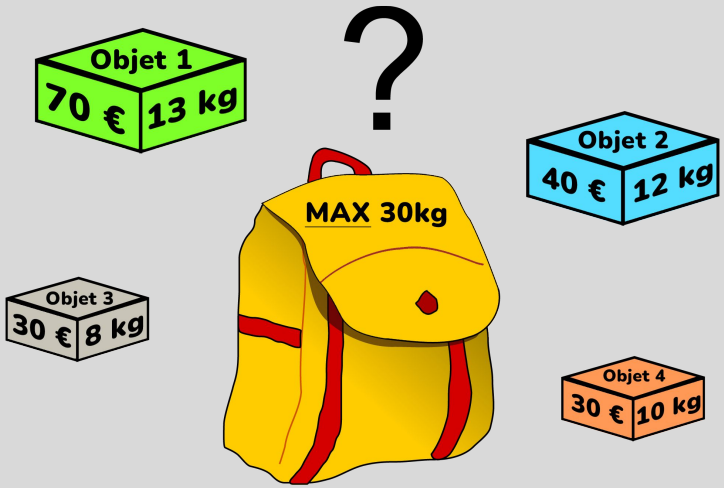
Le principe de la méthode gloutonne est de faire le choix qui semble **le plus pertinent sur le moment**, avec l'espoir qu'au bout du compte, cela nous conduira vers une solution optimale du problème à résoudre.

2022 - MELLAZA

01

“Knapsack problem”

De Richard Karp, 1972



Une situation de remplissage d'un sac à dos ne pouvant supporter plus d'un certain poids (30 kg ici), avec un ensemble donné d'objets ayant chacun un poids et une valeur.
Les objets mis dans le sac à dos doivent maximiser la valeur totale, sans dépasser le poids maximum.

Problématique

Objets	1	2	3	4
Valeur Vi (€)	70	40	30	30
Masse Mi (kg)	13	12	8	10

L'idée est d'ajouter en priorité les objets les plus efficaces, jusqu'à saturation du sac !
C'est-à-dire l'objet qui, à l'étape courante, possède la meilleure valeur pour une masse moindre.

Calcul Efficacité

Objets	1	2	3	4
Valeur Vi (€)	70	40	30	30
Masse Mi (kg)	13	12	8	10
Efficacité (Vi/Mi)	5,38	3,33	3,75	3,00

On se rend compte aisément qu'il est judicieux de prendre l'objet 1 en premier, puis le 3, le 2...
MAIS ALORS ON DÉPASSE LA MASSE MAX.

Limites de l'algorithme

Objets	1	2	3	4
Valeur Vi (€)	70	40	30	30
Masse Mi (kg)	13	12	8	10
Efficacité (Vi/Mi)	5,38	3,33	3,75	3,00

Le problème avec l'algorithme Glouton, est qu'il n'agit pas dans un but à long terme... Ici nous ne pourrons prendre seulement que deux objets, certes de valeur, mais moindre que les objets 2, 3 et 4 cumulés !

```python
# -*- coding: utf-8 -*-
"""
Mise en oeuvre d'un algorithme glouton pour le sac à dos
"""
# Masse maximum pouvant être emportée dans le sac
masse_max_sac = 30

# Définition du système d'objets : liste de sous-listes
# Sous-liste : ["Nom_objet",valeur_objet,masse_objet,valeur_objet/masse_objet]
systeme_objets=[["Objet1",70,13,70/13],["Objet2",40,12,40/12],["Objet3",30,8,30/8],
["Objet4",30,10,30/10]]

# Tri du système d'objets par valeurs décroissantes de (valeur_objet/masse_objet)

# Utilisation de key et d'une fonction lambda pour faire porter le tri sur le
# dernier élément de chaque sous-liste
systeme_objets.sort(key=lambda colonne:colonne[-1],reverse = True)

# Définition de la fonction gloutonne
def sac_a_dos(masse_max_sac, systeme_objets):
    '''Fonction gloutonne'''
    # Initialisation de la masse temporaire
    masse = 0

    # Initialisation de la liste d'objets à sélectionner
    liste_objets = []
    for i in range(len(systeme_objets)):
        if masse + systeme_objets[i][-2] <= masse_max_sac:
            masse = masse + systeme_objets[i][-2]
            liste_objets.append(systeme_objets[i])
    return liste_objets

print(sac_a_dos(masse_max_sac, systeme_objets))
```

2022 - MELLAZA

02

Rendu de monnaie

Avec un système monétaire donné (pièces et billets), comment rendre une somme donnée de façon optimale, c'est-à-dire avec le nombre minimal de pièces et billets ?

Bien qu'il existe différents systèmes monétaire dans le monde, le problème reste le même.

Le but est d'utiliser le moins de pièces et de billets possibles, en utilisant le système EURO par exemple. Un humain lambda choisira la pièce ou le billet dont la valeur se rapproche le plus possible du rendu restant, tout en ne le dépassant pas !

Si l'on doit donner **253€** par exemple, (nous disposons de tous les éléments du système), alors je choisis un billet de **200€**, puis **50€** et enfin une pièce de **2€** et de **1€** !

```python
# -*- coding: utf-8 -*-
"""
Auteur : Romain MELLAZA
Mise en oeuvre d'un algorithme glouton pour le rendu de monnaie
"""
# Somme devant être rendue
somme_a_rendre = float(input("Saisissez la somme devant être rendue :\n"))

# Définition du système de monnaie (liste)
# Hypothèse : pièces et billets disponibles en nombre infini
systeme_monnaie = [0.01,0.02,0.05,0.10,0.20,1,2,5,10,20,50,100,200,500]

# Initialisation de la liste de la monnaie (pièces + billets) à rendre
liste_monnaie_rendue = []

# Définition de la fonction gloutonne
def monnaie_a_rendre(somme_a_rendre, systeme_monnaie):
    '''Fonction gloutonne'''
    # Liste de la monnaie à rendre
    liste_monnaie_rendue = []

    # Indice de la première monnaie à comparer à la somme à rendre
    # On commence par la monnaie de plus grande valeur (dernier élément de la liste)
    i = len(systeme_monnaie) - 1

    while somme_a_rendre > 0:
        valeur = systeme_monnaie[i]
        if somme_a_rendre < valeur:
            i = i - 1
        else:
            liste_monnaie_rendue.append(valeur)
            somme_a_rendre = somme_a_rendre - valeur
        if i <= 0 :
            break
    return liste_monnaie_rendue

# Programme principal (appel de la fonction)
result = monnaie_a_rendre(somme_a_rendre,systeme_monnaie)
```

```python
def resolve_float_bug(expect, solution):
    """
    Cette fonction résout l'erreur de virgule flottante.
    Cela affecte seulement les pièces de 1 centime.
    """
    expect = round(expect, 2)
    solution = sum(solution) # On additionne tous les éléments de la liste résultat.

    if expect != solution :
        print("Il vous faut 1 pièce(s) de 0.01 €")


# Programme principal (appel de la fonction)
result = monnaie_a_rendre(somme_a_rendre,systeme_monnaie)


already_found = [] # Je stocke les éléments déjà comptés pour éviter de les recompter.

for element in result:
    """
    On classe la liste des éléments à rendre.
    """
    if element not in already_found :
        already_found.append(element)
        counter = result.count(element)
        if counter > 0 :
            if systeme_monnaie.index(element) >= 7 :
                print("Il vous faut", counter,"billet(s) de",element,"€")
            else :
                print("Il vous faut", counter,"pièce(s) de",element,"€")

        counter = 0 # Réinitialisation

resolve_float_bug(somme_a_rendre, result)
```

2022 - MELLAZA

03

“Problème du voyageur”

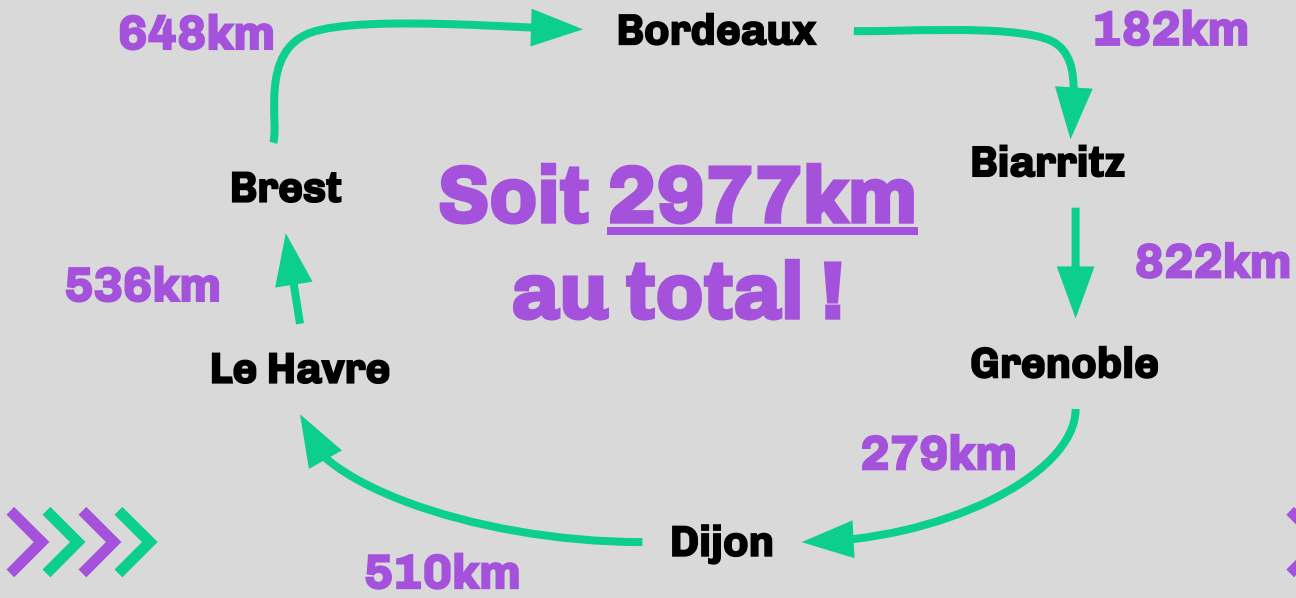
Un voyageur doit passer une fois dans chaque ville, comment doit il choisir son trajet pour minimiser la distance ?

Distance entre villes.

	Biarritz	Bordeaux	Brest	Dijon	Grenoble	Le havre
Biarritz	0	182	830	918	822	841
Bordeaux	182	0	648	619	675	659
Brest	830	648	0	793	1051	536
Dijon	918	619	793	0	279	510
Grenoble	822	675	1051	279	0	774
Le havre	841	659	536	510	774	0

On considère le problème du voyageur du commerce avec un **départ à Bordeaux**, on applique l'algorithme glouton qui consiste à choisir pour la ville suivante, la ville la plus proche.

Application :



```python
# -*- coding: utf-8 -*-
# Auteur : Romain MELLAZA

# Matrice des distances entre villes
distance = [[0,182,830,918,822,841],    # Biarritz
            [182,0,648,619,675,659],    # Bordeaux
            [830,648,0,793,1051,536],   # Brest
            [918,619,793,0,279,510],    # Dijon
            [822,675,1051,279,0,774],   # Grenoble
            [841,659,536,510,774,0]]    # Le Havre

# Nom des villes. On travailler avec les indices.
villes = ["Biarritz","Bordeaux","Brest","Dijon","Grenoble","Le Havre"]

def prochaine_ville(indice_ville_actuelle,indice_villes_visitees) :
    """ La fonction donne l'indice de la ville la plus proche de celle d'indice actuel
    et qui n'a pas été visitée. """
    mini = 1000000000
    indice_mini = -1
    for i in range(len(distance)) :
        if indice_villes_visitees[i] == False :
            if distance[indice_ville_actuelle][i] < mini :
                mini = distance[indice_ville_actuelle][i]
                indice_mini = i
    return indice_mini

def affiche_parcourt(chemin) :
    """ Fonction qui affiche le trajet final du voyageur."""
    trajet =""
    for numero_ville in chemin:
        trajet += villes[numero_ville] + " - "
    trajet += villes[chemin[0]]
    print(trajet)

def calcul_distance(chemin) :
    """ Fonction qui calcul la distance du trajet """
    distance_trajet = 0
    for i in range(len(chemin)-1) :
        distance_trajet += distance[chemin[i]][chemin[i+1]]
    distance_trajet += distance[chemin[-1]][chemin[0]]
    return distance_trajet
```

```python
def glouton(ville_debut):
    """ La fonction donne le chemin a suivre du voyageur de commerce sous forme d'un tableau
d'indice."""
    indice_villes_visitees = [False] * 6 # Au départ les villes ne sont pas visitées
    chemin = [0]* (len(distance))   # Le chemin du voyageur au départ il est est vide.

    indice_ville_actuelle = villes.index(ville_debut) # On trouve l'indice de la ville
    indice_villes_visitees[indice_ville_actuelle] = True #On dit que la ville est vistée.
    chemin[0] = indice_ville_actuelle

    for i in range(1,len(distance)) :
        indice_prochaine_ville= prochaine_ville(indice_ville_actuelle,indice_villes_visitees)
        chemin[i] = indice_prochaine_ville
        indice_ville_actuelle = indice_prochaine_ville
        indice_villes_visitees[indice_ville_actuelle] = True # La ville est visitée elle ne peut plus
être prise.
    affiche_parcourt(chemin)
    print(calcul_distance(chemin),"km")

glouton("Bordeaux")
```

2022 - Romain MELLAZA

Merci pour votre attention !