TP07 – Algorithmes Gloutons

Pour diverses situations – comme rendre la monnaie de manière optimale (en rendant le moins de pièce possible), pour remplir un coffre de voiture avec plusieurs bagages de différentes dimensions, ou pour attribuer des plages horaires à des conférenciers qui ont des contraintes d'emploi du temps en faisant en sorte d'un maximum de conférences puissent se tenir – une idée est de procéder par opérations successives de sorte qu'à chaque étape, on fait le choix qui semble le plus pertinent, en espérant qu'à la fin, cela conduira vers une solution optimale du problème à résoudre. On appelle cela les *algorithmes gloutons*.

I Le problème du sac à dos

Vous êtes un voleur et souhaitez emporter les objets pour maximiser la valeur totale du butin. Cependant, on dispose d'un sac pouvant supporter seulement 10 kg. Il s'agit de choisir les objets à emporter dans le sac afin maximiser la valeur totale tout en respectant la contrainte du poids maximal. C'est un problème d'*optimisation avec contrainte*. Parmi les objets suivants, quels objets faut-il prendre pour maximiser le butin tout en ne dépassant pas la masse maximale de 10 kg ?

Objet	A	B	C	D	E	F
Masse (kg)	7	6	4	3	2	1
Valeur (€)	9100	7200	4800	2700	2600	200
$\frac{\text{valeur}}{\text{masse}}$ (€/kg)	1300	1200	1200	900	1300	200

Pour ce problème, on dispose de plusieurs stratégies gloutonnes.

- **Stratégie 1 :** Prendre toujours l'objet de plus faible masse en n'excédant pas la capacité restante du sac.

Objet Choisi	Poids ajouté	Valeur ajoutée	Capacité restante du sac
F	1 kg	200 €	9 kg
E	2 kg	2600 €	7 kg
D	3 kg	2700 €	4 kg
C	4 kg	4800 €	0 kg

Finalement, pour cette stratégie,

Objets dans le sac	Valeur totale du sac
F, E, D, C	10 300 €

- **Stratégie 2 :** Prendre toujours l'objet de plus grande valeur n'excédant pas la capacité restante du sac.

Objet Choisi	Poids ajouté	Valeur ajoutée	Capacité restante du sac
A	7 kg	9100 €	3 kg
D	3 kg	2700 €	0 kg

Finalement, pour cette stratégie,

Objets dans le sac	Valeur totale du sac
A, D	11 800 €

Le code Python qui permet de mettre en place cette stratégie est le suivant.

Entrée [1]:
```
objets = [[9100, 7], [7200, 6], [4800, 4],
[2700, 3], [2600, 2], [200, 1]]
poids_max = 10
sac = []
poids = 0
valeur = 0
for i in range(len(objets)):
    objet = objets[i]
    poids_objet = objet [1]
    if poids + poids_objet <= poids_max:
        sac.append(objet)
        poids = poids + poids_objet
        valeur = valeur + objet [0]
print('sac =', sac, '; poids =', poids, '; valeur =', valeur)
```

Out [1]:
```
ac = [[9100, 7], [2700, 3]] ; poids = 10 ; valeur = 11800
```

- **Stratégie 3 :** Prendre toujours l'objet de plus grand rapport $\frac{\text{valeur}}{\text{masse}}$ en n'excédant pas la capacité restante du sac.

Objet Choisi	Poids ajouté	Valeur ajoutée	Capacité restante du sac
A	7 kg	9 100 €	3 kg
E	2 kg	2 600 €	1 kg
F	1 kg	200 €	0 kg

Finalement, pour cette stratégie,

Objets dans le sac	Valeur totale du sac
A, E, F	11 900 €

Finalement, parmi ces trois stratégies, la stratégie optimale est la stratégie numéro 3. Cependant, ce n'est même pas la stratégie optimale ici... Pouvez-vous trouver une meilleure solution ?

B 6 kg 7 200 €
C 4 kg 4 800 €
total : 12 000 €

II Le problème du rendu de monnaie

II.1 Le principe

La personne en charge de la caisse d'un magasin doit rendre régulièrement de la monnaie aux client · e · s qui payent en espèces. On cherche à rendre la monnaie de manière optimale, c'est-à-dire en se servant d'un minimum de pièce (une pièce ici pouvant désigner aussi un billet). Ici la stratégie gloutonne consiste à se servir en premier des pièces ayant la plus grande valeur possible tant que la somme à rendre est supérieure à cette valeur, puis d'aller en décroissant en gardant le même principe : se servir de la plus grande valeur possible.

II.2 Des exemples

On suppose que l'on dispose de billets 200€, 100€, 50€, 20€, 10€et de pièces de 2€et 1€.

1. Comment rendre 94 € ?

Pièce utilisée	Somme restante à rendre
50 €	44 €
20 €	24 €
20 €	4 €
2 €	2 €
2 €	0 €

2. Comment rendre 74 € ?

Pièce utilisée	Somme restante à rendre
50 €	34 €
20 €	14 €
10 €	4 €
2 €	2 €
2 €	0 €

Faux : à refaire

3. Quel rendu de monnaie un algorithme glouton effectue-t-il si l'on dispose uniquement de pièces de 1€, 3 €et 4€pour rendre 10€ ?

Pièce utilisée	Somme restante à rendre
4 €	6 €
4 €	2 €
1 €	1 €
1 €	0 €

Peut-on trouver une meilleure solution ?

2 pièces de 3 €
1 pièces de 4 €

II.3 L'algorithme

Complétez le programme suivant permettant de déterminer les pièces à rendre dans l'exemple 1.

Entrée [2]:
```
systeme = [200, 100, 50, 20, 10, 2, 1]
somme_à_rendre = 94 #un exemple, on pourra changer
rendu = [] #liste des pièces utilisées
i=0

#tant qu'il reste de l'argent à rendre
while somme_à_rendre > 0 :
    #on prend la piece numero i
    piece = systeme[i]
    #si la somme à rendre est plus imp. que la pièce
    if piece <= somme_à_rendre :
        rendu.append(piece)
        somme_à_rendre = somme_à_rendre - piece
    else:
        i = i+1 # on change de pièce
print(rendu)
```

III Planning d'un séminaire

Vous assistez à un séminaire présentant un certains nombre de conférences qui vous intéressent. Vous souhaitez voir le maximum de conférences (en nombre, pas en durée). Il s'agit donc, en fonction des horaires des conférences, d'établir votre planning de la journée.

1. **Exemple 1 :** On a quatre conférences C1, C2, C3 et C4 et leurs créneaux sont indiquées sous la forme d'un intervalle dans la figure ci-dessous.

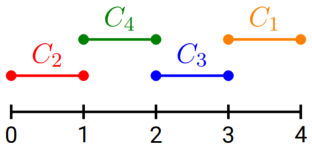

FIGURE 1 – Situation 1

Quelle est le nombre maximal de conférences qu'il est possible de voir (en donnant l'emploi du temps associé) ? Y'a-t-il unicité de la solution ?

Nbre max de conf.	Emploi du temps.	Remarques
4	C_2-C_4-C_3-C_1	Unicité

2. **Exemple 2 :**

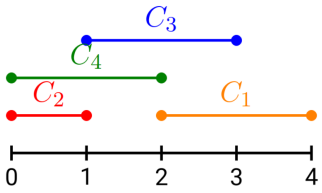

FIGURE 2 – Situation 2

Quelle est le nombre maximal de conférences qu'il est possible de voir (en donnant l'emploi du temps associé) ? Y'a-t-il unicité de la solution ?

Nbre max de conf.	Emploi du temps.	Remarques
2	$C_2 - C_1$	Bof car trou
2	$C_2 - C_3$	on veut finir le plus tôt possible
2	$C_4 - C_1$	

III.1 L'algorithme

Ici, l'algorithme glouton permettant de déterminer le nombre maximal de conférences qu'il est possible de voir peut se détailler de la manière suivante. On peut montrer ici qu'il s'agit de l'algorithme optimal.

1. On classe les conférences par heure de fin croissante (en cas d'égalité, on mettra d'abord la conférence C_i avec le i le plus petit possible par convention).
2. On choisit la première conférence de la liste, on l'ajoute au planning.
3. On s'intéresse à la deuxième conférence de la liste. Si elle est compatible (on ne veut pas débarquer en plein milieu d'une conférence...), on l'ajoute au planning, sinon on passe à la conférence suivante.
4. On recommence l'étape précédente jusqu'à avoir passé en revue toutes les conférences.

On souhaite appliquer cet algorithme à l'exemple suivant.

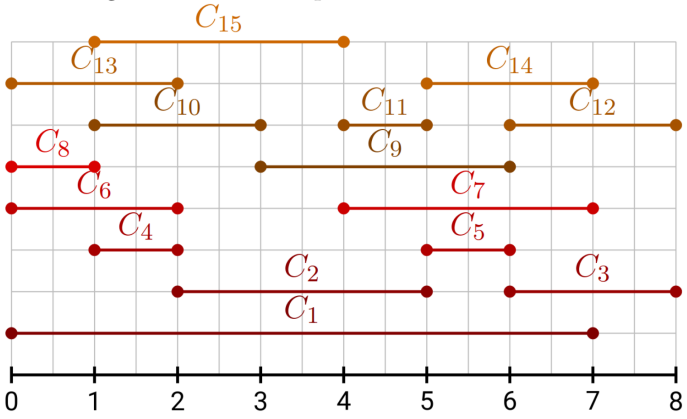

FIGURE 4 – Une situation plus complexe.

1. Classer les conférences par heure de fin croissante.

$C_8 \preceq C_4 \preceq C_6 \preceq C_{13} \preceq C_{10} \preceq C_{15} \preceq C_2 \preceq C_{11} \preceq C_5 \preceq C_9 \preceq C_1 \preceq C_7 \preceq C_{14} \preceq C_3 \preceq C_{12}$

2. En déduire le nombre maximal de conférences qu'il est possible de voir et l'emploi du temps associé.

max = 5
emploi du temps = $C_8 - C_4 - C_2 - C_5 - C_3$

3. Complétez le programme suivant permettant de retrouver le résultat précédent à l'aide de Python.

```
conferences = [[0,7,'C1'], [2,5,'C2'], [6,8,'C3'], [1,2,'C4'],
[5,6,'C5'], [0,2,'C6'], [4,7,'C7'], [0,1,'C8'], [3,6,'C9'],
[1,3,'C10'],[4,5,'C11'], [6,8,'C12'], [0,2,'C13'], [5,7,'C14'],
[1,4,'C15']]

# Tri des conferences par date de fin croissante.
conferences_triees = sorted(conferences, key = lambda L:L[1] )
print(conferences_triees)

n = len(conferences)... # nombre de conferences

planning = []

i = 0 # indice de la première conference consideree

planning.append(conference_triees[i.][2.]) # on choisit la 1ère conf

for j in range(1,n):
    # heure de fin de la dernière conférence vue
    heure = conferences_triees[i.][1.]
    # heure de début de la conférence numéro j
    debut_j = conferences_triees[j.][0.]

    if debut_j>=heure: # si on peut aller voir la conférence j

        planning.append(conferences_triees[j.][2.])

        i = j. # numéro de la dernière conférence vue


print(planning)
```