TP15 - Algorithmes gloutons

1. Problème du sac à dos

On dispose d'un sac à dos, supportant au maximum 30kg, et des objets de différentes valeurs que l'on souhaite mettre dans le sac à dos. La taille des objets ne pose pas de problème ici et n'est donc pas mentionnée.

Objet	0	1	2	3
valeur (euros)	7	3	4	4
poids (kg)	13	7	11	12

In [1]:
```
objets = [[7, 13], [80, 20], [4, 11], [3, 10]]
poids_max = 30

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(i)
        poids = poids + poids_objet
        valeur = valeur + objet[0]
print('sac =', sac, '; poids =', poids, '; valeur =', valeur)
```

```
sac = [0, 2] ; poids = 24 ; valeur = 11
```

Les objets sont rangés dans l'ordre décroissant de Valeur/Poids. On rempli d'abord par les objets les plus rentables en terme de valeur/poids. On espère ainsi avoir à la fin une grande valeur dans le sac.

Ce n'est ici pas optimal car il vaut mieux mettre l'objet 1 en premier, puis le 3, pour une valeur totale de 83 euros.

2. Rendu de monnaie

La personne en charge de la caisse d'un magasin doit rendre régulièrement de la monnaie aux clients qui payent en espèces.

On va écrire un programme permettant de préciser quelles pièces elle doit donner au client selon la somme à rendre. **Le principe est expliqué en détail sur le sujet papier.**

On dispose d'une liste `systeme` constituée des valeurs des différentes pièces et des différents billets disponibles pour le rendu de monnaie, écrites dans l'ordre croissant.

In [2]:
```
systeme = [20, 10, 5, 2, 1] # un exemple de système de monnaie disponible

somme_à_rendre = 74 # un exemple, on pourra changer
rendu = []
i = 0

while somme_à_rendre > 0:
    piece = systeme[i]
    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)
```

```
[20, 20, 20, 10, 2, 2]
```

À propos de la qualité de la solution proposée

On applique le programme précédent aux cas particuliers évoqués dans le sujet.

In [3]:
```
systeme = [4,3,1] # un exemple de système de monnaie disponible

somme_à_rendre = 10 # un exemple, on pourra changer
rendu = []
i = 0

while somme_à_rendre > 0:
    piece = systeme[i]
    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)
```

```
[4, 4, 1, 1]
```

Le résultat n'est pas optimal : on pouvait ne rendre que 3 pièces : 2 pièces de 3 euros et 1 pièce de 4 euros.

In [4]:
```
systeme = [4,3] # un exemple de système de monnaie disponible

somme_à_rendre = 10 # un exemple, on pourra changer
rendu = []
i = 0

while somme_à_rendre > 0:
    piece = systeme[i]
    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)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[4], line 8
      5 i = 0
      7 while somme_à_rendre > 0:
----> 8     piece = systeme[i]
      9     if piece <= somme_à_rendre:
     10         rendu.append(piece)

IndexError: list index out of range
```

Cette fois, on n'arrive pas à rendre la monnaie avec les pièces disponibles !

3. 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**. Il s'agit donc, en fonction des horaires des conférences, d'établir votre planning de la journée.

Suivre l'énoncé du sujet papier.

In [ ]:
```
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']]
```

In [9]:
```
# heure de début de la conférence C3
conferences[2][0]
```

Out[9]:
```
6
```

In [10]:
```
# heure de fin de la conférence C14
conferences[13][1]
```

Out[10]:
```
7
```

In [11]:
```
# Trier les conferences
conferences_triees = sorted(conferences, key = lambda L:L[1] )
print(conferences_triees)
```

```
[[0, 1, 'C8'], [1, 2, 'C4'], [0, 2, 'C6'], [0, 2, 'C13'], [1, 3, 'C10'], [1, 4, 'C15'], [2, 5, 'C2'], [4, 5, 'C
11'], [5, 6, 'C5'], [3, 6, 'C9'], [0, 7, 'C1'], [4, 7, 'C7'], [5, 7, 'C14'], [6, 8, 'C3'], [6, 8, 'C12']]
```

In [12]:
```
n = len(conferences) # nombre de conferences

planning = []

i = 0 # indice de la première conference considérée

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

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

```
['C8', 'C4', 'C2', 'C5', 'C3']
```

4. Déplacement sur une grille

On dispose d'une grille carrée (liste de listes) comme ci-dessous. On souhaite, en partant du premier élément (en haut à gauche), arriver au dernier élément (en bas à droite) en cherchant à maximiser la somme des nombres rencontrés. Deux déplacements sont possibles : vers la droite et vers le bas.

In [5]:
```
grille = [[3,4,1,5,1,2],
          [2,2,4,6,1,2],
          [3,5,6,2,2,4],
          [6,5,3,6,1,1],
          [4,5,1,2,2,3],
          [4,6,6,6,3,4]]
```

In [6]:
```
chemin = []

n = len(grille)
i = 0
j = 0
valeur = grille[i][j]

while i<n-1 or j<n-1:
    if i<n-1 and j<n-1:
        if grille[i+1][j] > grille[i][j+1]:
            i = i+1
            mouvement = 'bas'
        else:
            j = j+1
            mouvement = 'droite'
    elif i == n-1: # dernière ligne, il faut aller à droite
        j = j+1
        mouvement = 'droite'
    elif j == n-1: # dernière colonne, on va en bas
        i = i+1
        mouvement = 'bas'
    chemin.append(mouvement)
    valeur += grille[i][j]


print(valeur)
print(chemin)
```

```
44
['droite', 'bas', 'bas', 'droite', 'bas', 'droite', 'bas', 'bas', 'droite', 'droite']
```

Ce n'est pas optimal. Par exemple, le chemin bas-bas-bas-bas-bas-droite-droite-droite-droite-droite a pour valeur $3+2+3+6+4+4+6+6+6+3+4=47$.