CORRIGÉ ALGORITHMES GLOUTONS

1 Problème du rendu de monnaie

1.

```
def est_un_systeme(S):
    l = len(S)
    if l==0 or S[0]!=1:
        return False
    for i in range(l-1):
        if S[i+1]<=S[i]:
            return False
    return True
```

2.

```
def glouton_monnaie_rec(x,S,i):
    if i ==1:
        R = [x]
    else:
        n = x // S[i-1]
        y = x % S[i-1]
        R = glouton_monnaie_rec(y,S,i-1)
        R.append(n)
    return R
```

3.

```
def glouton_monnaie(x, S) :
    return glouton_monnaie_rec(x,S,len(S))
```

4.

```
def glouton_monnaie_euro(x) :
    S = [1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000]
    l = len(S)
    R = glouton_monnaie(100*x, S)
    print(R)
    print("Il faut rendre :")
    for i in range(l-1, l-3, -1) :
        print("- ", R[i], "billets de ", S[i]//100, "euros")
    for i in range(l-3, l-6, -1) :
        print("- ", R[i], "pieces de ", S[i]//100, "euros")
    for i in range(l-6, -1, -1) :
        print("- ", R[i], "pieces de ", S[i], "centimes d euros")
```

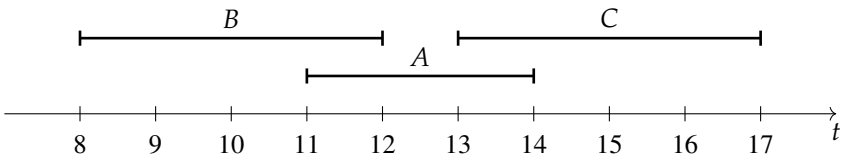
5. Soit le système (s_1, s_2) on a donc $s_1 = 1 < s_2$. Une solution optimale (c'est-à-dire utilisant le moins de pièces possible) utilise le maximum de pièces de valeur s_2 c'est-à-dire au plus $\lfloor \frac{x}{s_2} \rfloor$ avec x la somme à rendre. Ensuite on complète avec autant de pièces de valeur $s_1 = 1$ que nécessaire. C'est exactement ce que fait l'algorithme glouton. Ainsi $\boxed{\text{tout système } (s_1, s_2) \text{ est canonique}}$.

6. Par exemple $(1, 15, 27)$. Si on doit rendre 30 l'algorithme glouton renverra qu'il faut rendre 1 pièces de valaur 27 et 3 de valeur 1. Ce qui fait 4 pièces au total. Alors qu'on peut rendre 2 pièces de valeur 15, ce qui fait 2 pièces au total. L'algorithme glouton ne renvoie donc pas dans ce cas la solution optimale, $\boxed{\text{le système } (1, 15, 27) \text{ n'est pas canonique}}$.

7. Il suffit de remarquer par exemple que pour rendre 48, l'algorithme glouton va rendre $30 + 12 + 6$ alors que 2×24 est la solution optimale.

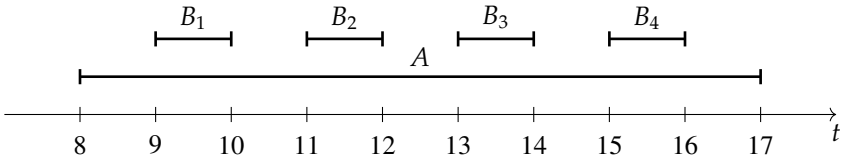
2 Réservation d'une salle

8. (a) la durée du cours :



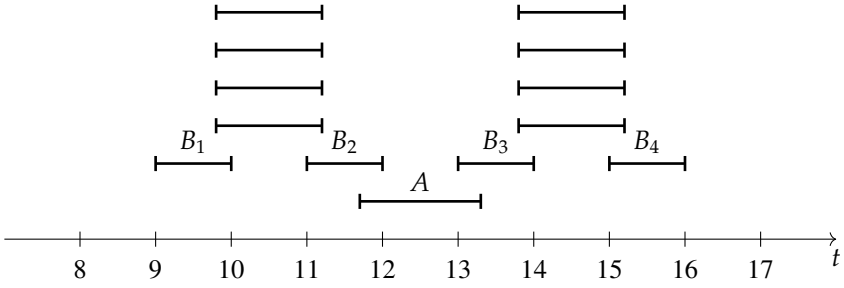
Dans ce cas, l'algorithme choisit le cours A, alors que le choix des cours B et C est optimal.

(b) la date de début du cours ;



Dans ce cas, l'algorithme choisit le cours A, alors que le choix des cours B_1, B_2, B_3 et B_4 est optimal.

(c) le nombre d'intersections du cours avec un autre cours :



L'algorithme va choisir le cours A puis les cours B_1 et B_4, alors que le choix des cours B_1, B_2, B_3 et B_4 est optimal.

9. La focntion doit à chaque cours associer sa date de fin. Un cours est représenté par une liste `[d_1, f_1,'nom']`, on propose donc :

```
cours = [[8,10,'MPSI'], [9.5,12,'MP'], [10,12,'PCSI'],
                        [11,13,'PSI'], [10.5,12.5,'PC']]
def f(l):
    return l[1]
```

10.

```
def planning(tab_cours):
    nb_cours  = len(tab_cours)
    list.sort(tab_cours, key = f)
    tab_planning = [tab_cours[0]]
    j=0 #garde en memoire le dernier cours ajoute au planning
    for i in range(1,nb_cours) :
        if tab_cours[i][0] >= tab_cours[j][1] :
            tab_planning.append(tab_cours[i])
            j = i
    return tab_planning
```

11. On note f la date de fin la plus petite.
Soit $\Gamma = \{f_1, f_2, \dots, f_k\}$ un ensemble de dates de fin d'épreuves définissant une solution optimale avec $f_1 < f_2 < \cdots < f_k$. L'ensemble $\Gamma' = \{f, f_2, \dots, f_k\}$ est donc également optimal (il comporte le même nombre de cours).
On note ensuite f' la date de fin la plus petite parmi les dates de fin d'épreuve compatibles avec f : c'est-à-dire les épreuves de de date de début strictement supérieure à f.
Si $f_2 \neq f'$, on remplace une épreuve correspondant à f_2 par une épreuve correspondant à f', cela est possible car f' est compatible avec f et $f' \leq f_2 < \cdots < f_k$. L'ensemble $\Gamma'' = \{f, f', f_3, \dots, f_k\}$ est donc également optimal (même nombre de cours).
Et ainsi de suite. Ainsi l'ensemble fourni par l'algorithme glouton $\{f, f', f'', \dots\}$ est optimal.

12. On note n le nombre de cours. Une fois la liste triée, l'algorithme glouton consiste en un parcours de la la liste des cours, donc cela se fait en $O(n)$. Sachant que le tri peut se faire en $O(n \ln n)$, l'algorithme glouton a une complexité en $O(n \ln n)$.

Fin