TP9 - Algorithme glouton

I Les dictionnaires

I.1 Définition

Un dictionnaire peut être vu comme une table d'association, qui à une clé, associe une valeur.

La clé permet de repérer le couple « clé : valeur » ; la clé doit donc être unique d'une part et être un élément immuable d'autre part. Ainsi une clé ne peut pas être une liste.

Contrairement à la liste qui est un tableau ordonné (suivant les indices), un dictionnaire n'est pas une structure ordonnée.

Un dictionnaire est une collection non ordonnée d'éléments.
Chaque élément est un couple constitué d'une clé et d'une valeur.
L'ensemble des couples « clé : valeur » est encadré par des accolades et les couples sont séparés par des virgules.

Exemple :

```
eleve = {"nom" : "Durand", "age" : 17, "Notes" : [12, 15, 9, 17, 16]}

dico_vide = {} #dictionnaire vide
```

Le dictionnaire `eleve` a 3 éléments. Par exemple, `"age" : 17` est un élément.
La valeur associée à la clé `"Notes"` est `[12, 15, 9, 17, 16]`.
Les clés du dictionnaire `eleves` sont `"nom"`, `"age"` et `"Notes"`.

I.2 Accès aux valeurs du dictionnaire

On accède aux valeurs du dictionnaire grâce aux clés.

Exemple :

```
eleve = {"nom" : "Durand", "age" : 17, "Notes" : [12, 15, 9, 17, 16]}
print(eleve["age"]) # 17
```

I.3 Modification de la structure d'un dictionnaire

Un dictionnaire est un objet mutable, ce qui signifie qu'on peut modifier une valeur, rajouter un élément, supprimer un élément.

Exemple :

```
eleve = {"nom" : "Durand", "age" : 17, "Notes" : [12, 15, 9, 17, 16]}

#La clé "age" existe
eleve["age"] = 16 #on modifie la valeur associée à "age"

#La clé "prenom" n'existe pas
eleve["prenom"] = "Alice" #on ajoute l'élément "prenom" : "Alice"

del eleve["Notes"] #on supprime l'élément "Notes" : [12, 15, 9, 17, 16]
```

I.4 Appartenance au dictionnaire

On peut tester si une clé est dans un dictionnaire.

Exemple :

```
eleve = {"nom" : "Durand", "age" : 17, "Notes" : [12, 15, 9, 17, 16]}

print("nom" in eleve) # True
print("Durand" in eleve) # False
```

I.5 Parcourir un dictionnaire

On peut parcourir un dictionnaire suivant les clés, suivant les valeurs, suivant le couple clé : valeur.

Exemple :

```
eleve = {"nom" : "Durand", "age" : 17, \
"Notes" : [12, 15, 9, 17, 16]}

#Parcours suivant les clés
for k in eleve:
  print("p1", k, eleve[k])
print("")

#Parcours suivant les valeurs
for v in eleve.values():
  print("p2", v)
print("")

#Parcours suivant les couple clé : valeur
for k, v in eleve.items():
  print("p3", k, v)
```

```
p1 nom Durand
p1 age 17
p1 Notes [12, 15, 9, 17, 16]

p2 Durand
p2 17
p2 [12, 15, 9, 17, 16]

p3 nom Durand
p3 age 17
p3 Notes [12, 15, 9, 17, 16]
```

Q1. Sans ordinateur, qu'est-ce qu'il s'affiche ?

1.
```
dico = {4 : "un", 2 : "deux", 1 : "trois"}
dico[2] = "two"
print(dico)
```

2.
```
dico = {4 : "un", 2 : "deux", 1 : "trois"}
dico[3] = "three"
print(dico)
```

3.
```
dico = {4 : "un", 2 : "deux", 1 : "trois"}
print("un" in dico)
```

```
dico = {4 : "un", 2 : "deux", 1 : "trois"}
print(len(dico))
```

```
dico = {4 : "un", 2 : "deux", 1 : "trois"}
for k in dico :
    print(dico[k])
```

```
dico = {4 : "un", 2 : "deux", 1 : "trois"}
for k in dico :
    print(k + k)
```

```
dico = {4 : "un", 2 : "deux", 1 : "trois"}
for k, v in dico.items() :
    print(k * v)
```

Q2. Sans ordinateur, corriger toutes les erreurs en sachant que le commentaire de la fonction est juste.

```
def somme_valeur(dico) :
    """renvoie la somme des valeurs de dico"""
    s = 0
    for k in dico :
        s = dico
    return s

assert somme_valeur({5 : 2, 3 : 1, 4 : 5}) == 12
```

Q3. Points au scrabble
Une chaîne de caractère peut être vue comme une liste de caractères.
A tester...

```
ch = 'OK'
for lettre in ch :
    print(lettre)
print(ch[0])

```

Compléter le fonction `points(mot)` qui prend en paramètre une chaîne de caractère mot (avec uniquement des majuscules) et qui renvoie le nombre de points au scrabble de ce mot.

```
def points(mot):
    scrabble = {"A":1,"B":3,"C":3,"D":2,"E":1,"F":4,"G":2,"H":4,"I":1,\
    "J":8,"K":10,"L":1,"M":2,"N":1,"O":1,"P":3,"Q":8,"R":1\
    ,"S":1,"T":1,"U":1,"V":4,"W":10,"X":10,"Y":10,"Z":10}
```

Q4. Statistiques d'un texte
Ecrire une fonction `occurrence(txt)` qui renvoie un dictionnaire dont les clés sont les lettres de l'alphabet contenues dans `txt` et les valeurs le nombre de fois que la lettre apparaît dans `txt`.

```
print(occurrence("abracadabra")) # {'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1}
```

II Algorithmes gloutons

II.1 Problème d'optimisation

Résoudre un problème d'**optimisation** consiste à déterminer parmi les solutions au problème, celles qui **minimisent** (ou maximisent) une fonction objectif.

Si on note F la fonction objectif, $\mathcal{S}$ l'ensemble des solutions, on appelle **solution optimale**, toute solution $s_0 \in \mathcal{S}$ qui minimise F, c'est-à-dire :

$$\forall s \in \mathcal{S} \qquad F(s_0) \leqslant F(s)$$

On peut citer par exemple le problème du choix d'un itinéraire dans un réseau routier. On cherche à déterminer le trajet pour aller d'un point A à un point B qui **minimise le temps de parcours**, tout en respectant un ensemble de contraintes : circuler sur les routes (on ne coupe pas à travers champs !) et respecter les sens de circulation.
Dans cet exemple :

- la fonction objectif associe à chaque trajet son temps de trajet ;
- une solution est un trajet allant de A à B en passant par des routes et en respectant le sens de circulation.
- une solution optimale est un trajet allant de A à B qui minimise le temps de parcours.

Les algorithmes gloutons sont utilisés pour résoudre des problèmes d'optimisation.

II.2 Résolution d'un problème d'optimisation

Le rendu de monnaie : On souhaite rendre 16 € avec le moins de pièces possibles sachant que les pièces utilisées ont pour valeur 10 € 7€, 2€ et 1€.

II.2.1 Recherche exhaustive

Pour résoudre ce problème, on peut faire une recherche exhaustive en explorant tous les choix possibles, et retenir les solutions optimales.
On peut modéliser la solution « 1 pièce de 10 €, 3 pièces de 2 € » par le dictionnaire `{10:1, 7:0, 2:3, 1:0}` .
A l'aide d'un programme simple à écrire, on trouve toutes les solutions.

```
[{10: 0, 7: 0, 2: 0, 1: 16}, {10: 0, 7: 0, 2: 1, 1: 14}, {10: 0, 7: 0, 2: 2, 1: 12},
    {10: 0, 7: 0, 2: 3, 1: 10}, {10: 0, 7: 0, 2: 4, 1: 8}, {10: 0, 7: 0, 2: 5, 1: 6},
     {10: 0, 7: 0, 2: 6, 1: 4}, {10: 0, 7: 0, 2: 7, 1: 2}, {10: 0, 7: 0, 2: 8, 1: 0},
     {10: 0, 7: 1, 2: 0, 1: 9}, {10: 0, 7: 1, 2: 1, 1: 7}, {10: 0, 7: 1, 2: 2, 1: 5},
     {10: 0, 7: 1, 2: 3, 1: 3}, {10: 0, 7: 1, 2: 4, 1: 1}, {10: 0, 7: 2, 2: 0, 1: 2},
     {10: 0, 7: 2, 2: 1, 1: 0}, {10: 1, 7: 0, 2: 0, 1: 6}, {10: 1, 7: 0, 2: 1, 1: 4},
     {10: 1, 7: 0, 2: 2, 1: 2}, {10: 1, 7: 0, 2: 3, 1: 0}]
```

Dans cet exemple, il y a une solution optimale qui est `{10: 0, 7: 2, 2: 1, 1: 0}`, c'est-à-dire 2 pièces de 7 €et 1 pièce de 2 €.

II.2.2 Algorithme glouton

Le principe d'un algorithme glouton est de faire un choix localement optimal dans l'espoir que ce choix mènera à une solution globalement optimale.
L'algorithme glouton construit une solution pas à pas :

- Etape 1 : on fait le choix qui semble le meilleur : c'est le choix glouton ;
- Etape 2 : on traite le sous-problème résultant du choix glouton. Pour celà, on reprend l'étape 1, etc.

Pour le rendu de monnaie avec 16 € :

- choix glouton : 10
- Sous-problème résultant du choix glouton : rendre 6
- choix glouton : 2
- Sous-problème résultant du choix glouton : rendre 4
- choix glouton : 2
- Sous-problème résultant du choix glouton : rendre 2
- choix glouton : 2
- Fin

La solution donnée par l'algorithme glouton est : 1 pièce de 10 €et 3 pièces de 2 €. Elle est simple à mettre à oeuvre et se rapproche de la solution optimale.

Remarque : Un algorithme glouton donne une solution optimale si les deux propriétés suivantes sont vérifiées :

- **Propriété du choix glouton** : il existe toujours une solution optimale qui contient un premier choix glouton.
- **Propriété de sous-structure optimale** : toute solution optimale contient une sous-structure optimale.

  ceci veut dire :

  Si $\mathcal{S}_0$ est une solution optimale au problème $\mathcal{P}$ contenant le choix c, alors $\mathcal{S}_0 - \{c\}$ est une solution du sous-problème $\mathcal{P}_c$ résultant du choix c dans le problème $\mathcal{P}$.

Dans le problème du rendu de monnaie, la propriété du choix glouton n'est pas vérifiée.

III Application

III.1 Le problème de choix d'activité

Le problème du choix d'activité est un problème dans lequel plusieurs activités souhaitent se voir attribuer une unique ressource.
Une application concrète est l'allocation d'une salle à différents conférenciers au cours d'une journée.

Plusieurs personnes souhaitent effectuer une conférence, mais ces conférences ont lieu à des horaires définis. Certaines conférences démarrent alors que d'autres ne sont pas encore terminées. Il n'est donc pas possible d'attribuer la salle à chaque conférencier, et il faut faire des choix.
Une solution optimale est celle qui permettra au plus grand nombre de conférences de se tenir.

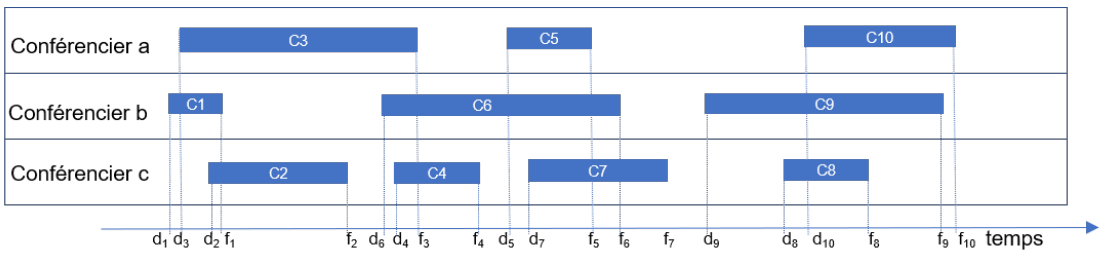

FIGURE 1 – Conférences

Les conférences sont définies par leur heure de début et leur heure de fin : la conférence c_i aura pour heure de début d_i et pour heure de fin f_i.

- Nous supposerons que les conférences ont été triées dans l'ordre croissant de leurs dates de fin. Ainsi, si nous travaillons avec un ensemble $\mathcal{C} = \{c_1, \cdots, c_n\}$ de n conférences, on a :

$$f_1 < f_2 < \ldots < f_{n-1} < f_n$$

- Les conférences doivent être compatibles et donc ne pas se chevaucher ; ainsi l'heure de début de l'une doit être après l'heure de fin de l'autre.

Le choix glouton consiste à choisir la conférence qui se termine le plus tôt possible.
Q5. Donner la solution gloutonne du problème illusté à la figure 1.

Implémentation
On modélise :

- une conférence par un dictionnaire dont les clés `"label"`, `"d"`, `"f"` représentent l'intitulé, l'heure de début et l'heure de fin de la conférence.
  Ainsi la conférence c_1 est modélisée par `{"label" : "c1", "d" : 8, "f" : 9}`.
- l'ensemble des conférences par une liste de dictionnaires représentant des conférences ordonnées suivant l'odre croissant de l'heure de fin de la conférence. Ainsi, l'exemple représenté sur la figure 1 pourrait être modélisé par la liste :

```
Conf = [ {"label" : "c1", "d" : 8, "f" : 9}, {"label" : "c2", "d" : 8.45, "f" : 10.3},
{"label" : "c3", "d" : 8.1, "f" : 11.3}, {"label" : "c4", "d" : 11.15, "f" : 11.45},
{"label" : "c5", "d" : 12, "f" : 12.45}, {"label" : "c6", "d" : 11, "f" : 13}, {"la-
bel" : "c7", "d" : 12.3, "f" : 14}, {"label" : "c8", "d" : 16, "f" : 17}, {"label"
: "c9", "d" : 15, "f" : 18} ]
```

Q6. Ecrire la fonction `choix_conferences(C)` qui prend en argument une liste de conférences `C` comme décrite précédemment et qui renvoie une liste de dictionnaires représentant la solution gloutonne du choix d'activités.
Q7. Ecrire le code permettant d'afficher la liste des intitulés des conférences retenues lors de la solution gloutonne.
Q8. On observe que le résultat obtenu est bien une solution optimale. Cependant, il existe d'autres solutions optimales. Lesquelles ?

III.2 Le rendu de monnaie

Le problème du rendu de monnaie consiste à déterminer des pièces utilisées pour rendre la monnaie. On travaille avec les valeurs entières du système monétaire européen :
On note `S` le tuple : `S=(500,200,100,50,20,10,5,2,1)`
Q9. Ecrire la fonction `rendu_monnaie(S, v)` qui a pour paramètre un tuple S correspondant au système de monnaie et une valeur entière `v` et qui renvoie le dictionnaire des pièces utilisées.
Par exemple `rendu_monnaie((100, 50, 20, 10, 5, 2, 1), 49)` renvoie `{20 : 2, 5: 1, 2 : 2}`