Les dictionnaires de Python

- Les dictionnaires de Python permettent de stocker des données sous forme de tableau associant une clé à une valeur :

Valeurs	v1	v2	v3	v4	...	
↑	↑	↑	↑	↑	↑	
Clés	c1	c2	c3	c4	...	uniques

Exemples

Les dictionnaires de Python

- Les dictionnaires de Python permettent de stocker des données sous forme de tableau associant une clé à une valeur :

Valeurs	v1	v2	v3	v4	...
↑	↑	↑	↑	↑	↑
Clés	c1	c2	c3	c4	...

uniques

- Un dictionnaire se note entre accolades : { et }

Exemples

Les dictionnaires de Python

- Les dictionnaires de Python permettent de stocker des données sous forme de tableau associant une clé à une valeur :

Valeurs	v1	v2	v3	v4	...	
↑	↑	↑	↑	↑	↑	
Clés	c1	c2	c3	c4	...	uniques

- Un dictionnaire se note entre accolades : { et }
- Les paires clés/valeurs sont séparés par des virgules ,

Exemples

Les dictionnaires de Python

- Les dictionnaires de Python permettent de stocker des données sous forme de tableau associant une clé à une valeur :

Valeurs	v1	v2	v3	v4	...	
↑	↑	↑	↑	↑	↑	
Clés	c1	c2	c3	c4	...	uniques

- Un dictionnaire se note entre accolades : { et }
- Les paires clés/valeurs sont séparés par des virgules ,
- Le caractère : sépare une clé de la valeur associée.

Exemples

Les dictionnaires de Python

- Les dictionnaires de Python permettent de stocker des données sous forme de tableau associant une clé à une valeur :

Valeurs	v1	v2	v3	v4	...	
↑	↑	↑	↑	↑	↑	
Clés	c1	c2	c3	c4	...	uniques

- Un dictionnaire se note entre accolades : { et }
- Les paires clés/valeurs sont séparés par des virgules ,
- Le caractère : sépare une clé de la valeur associée.

Exemples

- Un dictionnaire contenant des objets et leurs prix :

Les dictionnaires de Python

- Les dictionnaires de Python permettent de stocker des données sous forme de tableau associant une clé à une valeur :

Valeurs	v1	v2	v3	v4	...	
↑	↑	↑	↑	↑	↑	
Clés	c1	c2	c3	c4	...	uniques

- Un dictionnaire se note entre accolades : { et }
- Les paires clés/valeurs sont séparés par des virgules ,
- Le caractère : sépare une clé de la valeur associée.

Exemples

- Un dictionnaire contenant des objets et leurs prix :

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16}
```

Les dictionnaires de Python

- Les dictionnaires de Python permettent de stocker des données sous forme de tableau associant une clé à une valeur :

Valeurs	v1	v2	v3	v4	...	
↑	↑	↑	↑	↑	↑	
Clés	c1	c2	c3	c4	...	uniques

- Un dictionnaire se note entre accolades : { et }
- Les paires clés/valeurs sont séparés par des virgules ,
- Le caractère : sépare une clé de la valeur associée.

Exemples

- Un dictionnaire contenant des objets et leurs prix :
```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16}
```
- Un dictionnaire traduisant des couleurs du français vers l'anglais

Les dictionnaires de Python

- Les dictionnaires de Python permettent de stocker des données sous forme de tableau associant une clé à une valeur :

Valeurs	v1	v2	v3	v4	...	
↑	↑	↑	↑	↑	↑	
Clés	c1	c2	c3	c4	...	uniques

- Un dictionnaire se note entre accolades : { et }
- Les paires clés/valeurs sont séparés par des virgules ,
- Le caractère : sépare une clé de la valeur associée.

Exemples

- Un dictionnaire contenant des objets et leurs prix :
```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16}
```
- Un dictionnaire traduisant des couleurs du français vers l'anglais
```
couleurs = { "vert":"green" , "bleu" : "blue", "rouge" : "red" }
```

Opérations sur un dictionnaire

- On accède aux éléments d'un dictionnaire avec la syntaxe `nom_dictionnaire[cle]`

Opérations sur un dictionnaire

- On accède aux éléments d'un dictionnaire avec la syntaxe `nom_dictionnaire[cle]`

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "plat" : 30 }
```

Par exemple, `prix["verre"]` contient 12

Opérations sur un dictionnaire

- On accède aux éléments d'un dictionnaire avec la syntaxe `nom_dictionnaire[cle]`

  `prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "plat" : 30 }`

  Par exemple, `prix["verre"]` contient 12
- On peut ajouter une clé à un dictionnaire existant en effectuant une affectation `nom_dictionnaire[nouvelle_cle]=nouvelle_valeur`

Opérations sur un dictionnaire

- On accède aux éléments d'un dictionnaire avec la syntaxe `nom_dictionnaire[cle]`

  `prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "plat" : 30 }`

  Par exemple, `prix["verre"]` contient 12
- On peut ajouter une clé à un dictionnaire existant en effectuant une affectation `nom_dictionnaire[nouvelle_cle]=nouvelle_valeur`

  On ajoute un nouvel objet avec son prix :

  `prix["couteau"]=20`

Opérations sur un dictionnaire

- On accède aux éléments d'un dictionnaire avec la syntaxe `nom_dictionnaire[cle]`

  `prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "plat" : 30 }`

  Par exemple, `prix["verre"]` contient 12
- On peut ajouter une clé à un dictionnaire existant en effectuant une affectation `nom_dictionnaire[nouvelle_cle]=nouvelle_valeur`

  On ajoute un nouvel objet avec son prix :

  `prix["couteau"]=20`
- On peut modifier la valeur associée à une clé avec une affectation `nom_dictionnaire[cle]=nouvelle_valeur`

Opérations sur un dictionnaire

- On accède aux éléments d'un dictionnaire avec la syntaxe `nom_dictionnaire[cle]`

  `prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "plat" : 30 }`

  Par exemple, `prix["verre"]` contient 12
- On peut ajouter une clé à un dictionnaire existant en effectuant une affectation `nom_dictionnaire[nouvelle_cle]=nouvelle_valeur`

  On ajoute un nouvel objet avec son prix :

  `prix["couteau"]=20`
- On peut modifier la valeur associée à une clé avec une affectation `nom_dictionnaire[cle]=nouvelle_valeur`

  Le pris d'une tasse passe à 10 :

  `prix["tasse"]=10`

Présence dans un dictionnaire

- Attention, essayer d'accéder à une clé qui n'est pas dans un dictionnaire renvoie une erreur !

Présence dans un dictionnaire

- Attention, essayer d'accéder à une clé qui n'est pas dans un dictionnaire renvoie une erreur !

  Il n'y a pas de clé `"fourchette"` dans le dictionnaire prix, donc `prix["fourchette"]` renvoie une erreur (`KeyError`).

Présence dans un dictionnaire

- Attention, essayer d'accéder à une clé qui n'est pas dans un dictionnaire renvoie une erreur !

  Il n'y a pas de clé `"fourchette"` dans le dictionnaire prix, donc `prix["fourchette"]` renvoie une erreur (`KeyError`).
- On teste la présence d'une clé dans un dictionnaire avec `cle in nom_dictionnaire`

Présence dans un dictionnaire

- Attention, essayer d'accéder à une clé qui n'est pas dans un dictionnaire renvoie une erreur !

  Il n'y a pas de clé `"fourchette"` dans le dictionnaire prix, donc `prix["fourchette"]` renvoie une erreur (`KeyError`).
- On teste la présence d'une clé dans un dictionnaire avec `cle in nom_dictionnaire`

  la fourchette n'est pas dans le dictionnaire prix
  Le test `fourchette in prix` renvoie `False`

Présence dans un dictionnaire

- Attention, essayer d'accéder à une clé qui n'est pas dans un dictionnaire renvoie une erreur !

  Il n'y a pas de clé `"fourchette"` dans le dictionnaire prix, donc `prix["fourchette"]` renvoie une erreur (`KeyError`).

- On teste la présence d'une clé dans un dictionnaire avec `cle in nom_dictionnaire`

  la fourchette n'est pas dans le dictionnaire prix
  Le test `fourchette in prix` renvoie `False`

  ! Ce test d'appartenance s'effectue en temps constant (indépendant de la taille du dictionnaire)

Présence dans un dictionnaire

- Attention, essayer d'accéder à une clé qui n'est pas dans un dictionnaire renvoie une erreur !

  Il n'y a pas de clé `"fourchette"` dans le dictionnaire prix, donc `prix["fourchette"]` renvoie une erreur (`KeyError`).
- On teste la présence d'une clé dans un dictionnaire avec `cle in nom_dictionnaire`

  la fourchette n'est pas dans le dictionnaire prix

  Le test `fourchette in prix` renvoie `False`

  ❗ Ce test d'appartenance s'effectue en temps constant (indépendant de la taille du dictionnaire)
- On peut supprimer une clé existante dans un dictionnaire avec `del nom_dictionnaire[cle]`

Présence dans un dictionnaire

- Attention, essayer d'accéder à une clé qui n'est pas dans un dictionnaire renvoie une erreur !

  Il n'y a pas de clé `"fourchette"` dans le dictionnaire prix, donc `prix["fourchette"]` renvoie une erreur (`KeyError`).

- On teste la présence d'une clé dans un dictionnaire avec `cle in nom_dictionnaire`

  la fourchette n'est pas dans le dictionnaire prix

  Le test `fourchette in prix` renvoie `False`

  ❗ Ce test d'appartenance s'effectue en temps constant (indépendant de la taille du dictionnaire)

- On peut supprimer une clé existante dans un dictionnaire avec `del nom_dictionnaire[cle]`

  On supprimer le couteau :

  ```
  del prix["couteau"]
  ```

Parcours d'un dictionnaire

- Le parcours par clé s'effectue directement avec `for cle in nom_dictionnaire`

Parcours d'un dictionnaire

- Le parcours par clé s'effectue directement avec `for cle in nom_dictionnaire`

  `prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "plat" : 30 }`

  Par exemple, `for objet in prix` permettra à la variable `objet` de prendre successivement les valeurs des clés : `"verre"`, `"tasse"`, `"assiette"` et `"plat"`.

Parcours d'un dictionnaire

- Le parcours par clé s'effectue directement avec `for cle in nom_dictionnaire`

  `prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "plat" : 30 }`

  Par exemple, `for objet in prix` permettra à la variable `objet` de prendre successivement les valeurs des clés : `"verre"`, `"tasse"`, `"assiette"` et `"plat"`.

- Le parcours par valeur s'effectue en ajoutant `.values()` au nom du dictionnaire : `for valeur in nom_dictionnaire.values()`

Parcours d'un dictionnaire

- Le parcours par clé s'effectue directement avec `for cle in nom_dictionnaire`

  `prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "plat" : 30 }`

  Par exemple, `for objet in prix` permettra à la variable `objet` de prendre successivement les valeurs des clés : `"verre"`, `"tasse"`, `"assiette"` et `"plat"`.

- Le parcours par valeur s'effectue en ajoutant `.values()` au nom du dictionnaire : `for valeur in nom_dictionnaire.values()`

  Par exemple, `for p in prix.values()` permettra à la variable `p` de prendre successivement les valeurs du dictionnaire : `12`, `8` , `16` et `30`.

Exemple

On dispose d'une liste de nombres entiers et on veut obtenir le nombre d'occurence du (ou des) entiers(s) les plus fréquents dans cette liste. Par exemple si la liste est `[1,7,1,3,4,1,3,4,3,1,5,108,2,3]` alors la réponse est 4, car les entiers les plus fréquents sont 1 et 3 qui apparaissent tous les deux à 4 reprises.

1. Proposer une solution qui pour chaque élément de la liste calcule son nombre d'apparitions à l'aide d'une fonction `compte_occurence`

Exemple

On dispose d'une liste de nombres entiers et on veut obtenir le nombre d'occurence du (ou des) entiers(s) les plus fréquents dans cette liste. Par exemple si la liste est `[1,7,1,3,4,1,3,4,3,1,5,108,2,3]` alors la réponse est 4, car les entiers les plus fréquents sont 1 et 3 qui apparaissent tous les deux à 4 reprises.

1. Proposer une solution qui pour chaque élément de la liste calcule son nombre d'apparitions à l'aide d'une fonction `compte_occurence`
2. Proposer une solution utilisant un dictionnaire dont les clés sont les entiers présents dans la liste et les valeurs leurs nombre d'apparitions

Exemple

On dispose d'une liste de nombres entiers et on veut obtenir le nombre d'occurence du (ou des) entiers(s) les plus fréquents dans cette liste. Par exemple si la liste est `[1,7,1,3,4,1,3,4,3,1,5,108,2,3]` alors la réponse est 4, car les entiers les plus fréquents sont 1 et 3 qui apparaissent tous les deux à 4 reprises.

1. Proposer une solution qui pour chaque élément de la liste calcule son nombre d'apparitions à l'aide d'une fonction `compte_occurence`
2. Proposer une solution utilisant un dictionnaire dont les clés sont les entiers présents dans la liste et les valeurs leurs nombre d'apparitions
3. Commenter l'efficacité de ces deux solutions.

Correction question 1

```python
def compte_occurence(elt,liste):
    occ = 0
    for x in liste:
        if x==elt:
            occ+=1
    return occ

def plus_frequent(liste):
    max_occ = 0
    for elt in liste:
        elt_occ = compte_occurence(elt,liste)
        if elt_occ>max_occ:
            max_occ = elt_occ
    return max_occ

```

Correction question 2

```python
def plus_frequent(liste):
    nb_occ = {}
    for elt in liste:
        if elt not in nb_occ:
            nb_occ[elt] = 1
        else:
            nb_occ[elt] += 1
    return max(nb_occ[elt] for elt in nb_occ)
```

Correction question 2

```python
def plus_frequent(liste):
    nb_occ = {}
    for elt in liste:
        if elt not in nb_occ:
            nb_occ[elt] = 1
        else:
            nb_occ[elt] += 1
    return max(nb_occ[elt] for elt in nb_occ)
```

Correction question 3

La solution avec les dictionnaires est bien plus efficace car on effectue un seul parcours de la liste et que le test d'appartenance au dictionnaire est une opération élémentaire (temps constant en moyenne).

Implémentation des dictionnaires

- On crée un tableau T de liste de longueur N (donc indicé par les entiers $[\![0; N-1]\!]$).

Implémentation des dictionnaires

- On crée un tableau T de liste de longueur N (donc indicé par les entiers $[\![0; N-1]\!]$).
- Une fonction de hachage h transforme les clés en entier. Les clés doivent donc être non mutables (ce qui exclu les listes). Ces entiers sont ramenés dans l'intervalle $[\![0; N-1]\!]$ à l'aide d'un modulo.

Implémentation des dictionnaires

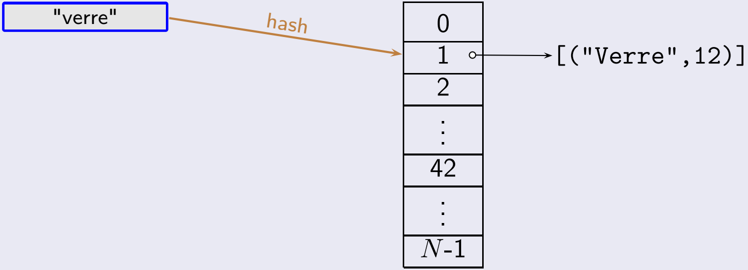
- On crée un tableau T de liste de longueur N (donc indicé par les entiers $[\![0;N-1]\!]$).
- Une fonction de hachage h transforme les clés en entier. Les clés doivent donc être non mutables (ce qui exclu les listes). Ces entiers sont ramenés dans l'intervalle $[\![0;N-1]\!]$ à l'aide d'un modulo.
- Chaque paire de clé/valeur (c,v) est stockée dans le tableau T à l'indice $h(c)$ (modulo N)

Implémentation des dictionnaires

- On crée un tableau T de liste de longueur N (donc indicé par les entiers $[\![0;N-1]\!]$).
- Une fonction de hachage h transforme les clés en entier. Les clés doivent donc être non mutables (ce qui exclu les listes). Ces entiers sont ramenés dans l'intervalle $[\![0;N-1]\!]$ à l'aide d'un modulo.
- Chaque paire de clé/valeur (c,v) est stockée dans le tableau T à l'indice $h(c)$ (modulo N)
- Le cas où dont deux clés différentes $c1$ et $c2$ produisent le même indice s'appelle une collision.

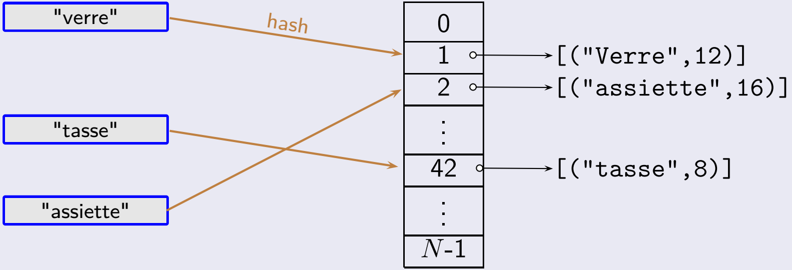
Visualisation

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "bol" : 10}
```



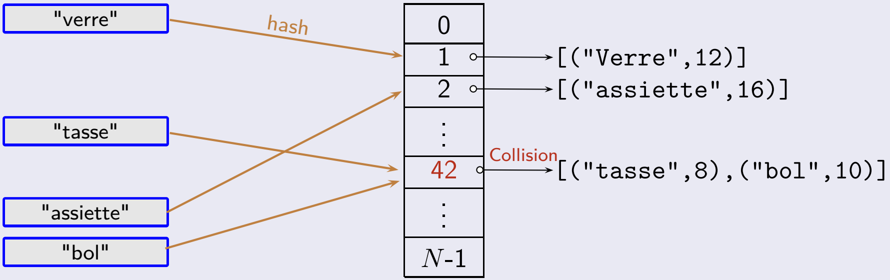
Visualisation

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "bol" : 10}
```



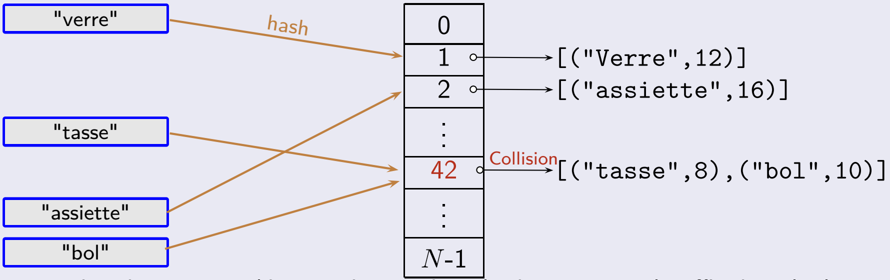
Visualisation d'une collision

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "bol" : 10}
```



Visualisation d'une collision

`prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "bol" : 10}`



Pour rechercher si une clé est présente dans le dictionnaire il suffit de calculer son *hash* et de regarder à l'indice correspondant dans le tableau.

Exercice

On dispose d'un fichier contenant des chaines de caractères (une par ligne). On souhaite écrire une fonction qui renvoie la première chaine de caractères qui apparaît une seconde fois. Par exemple si le fichier contient :

```
trkl
nbf
pmoz
hgty
nbf
rss
```

Alors la fonction doit renvoyer `nbf`. Si aucune chaine n'apparaît plus d'une fois alors le programme renvoie la chaine vide.

1. Proposer une version utilisant une liste `deja_vu` dans laquelle on enregistre au fur et à mesure du parcours les chaines de caractères déjà rencontrées.
2. Proposer une version *ayant une meilleur complexité*, utilisant un dictionnaire.

Exemple

1. Ecrire une fonction récursive qui prend en argument un entier n et renvoie le $n$ième terme de la suite de Fibonacci défini par :

$$\begin{cases} f_0 & = & 0, \\ f_1 & = & 1, \\ f_n & = & f_{n-1} + f_{n-2} \quad \text{pour tout} \quad n \geq 2. \end{cases}$$

Exemple

1. Ecrire une fonction récursive qui prend en argument un entier n et renvoie le $n$ième terme de la suite de Fibonacci défini par :
$$\begin{cases} f_0 & = & 0, \\ f_1 & = & 1, \\ f_n & = & f_{n-1} + f_{n-2} \text{ pour tout } \; n \geq 2. \end{cases}$$
2. Tracer le graphe des appels récursifs de cette fonction pour $n = 5$

Exemple

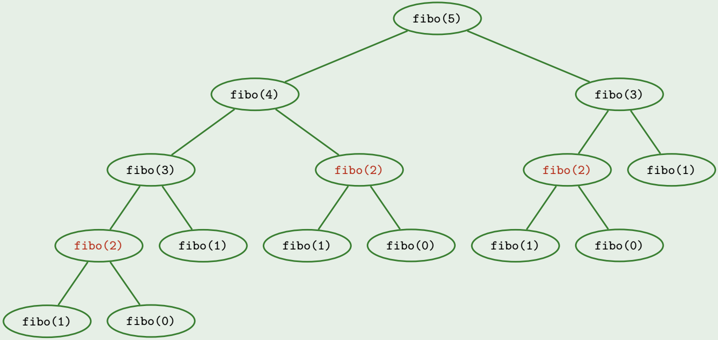
1. Ecrire une fonction récursive qui prend en argument un entier n et renvoie le $n$ième terme de la suite de Fibonacci défini par :
$$\begin{cases} f_0 & = & 0, \\ f_1 & = & 1, \\ f_n & = & f_{n-1} + f_{n-2} \text{ pour tout } n \geq 2. \end{cases}$$
2. Tracer le graphe des appels récursifs de cette fonction pour $n = 5$
3. Commenter

Correction question 1

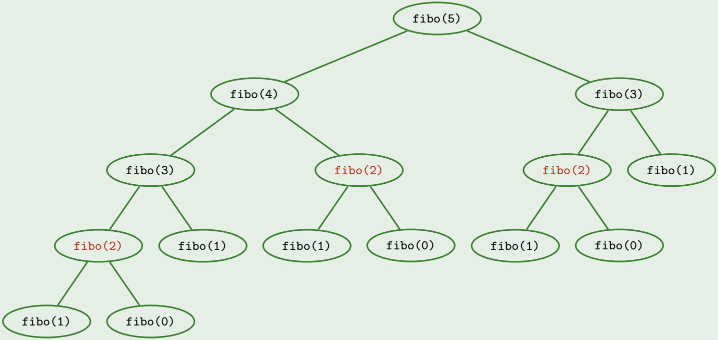
```python
def fibonacci(n):
    assert n>=0
    if n<2:
        return n
    return fibonacci(n-1)+fibonacci(n-2)

print(fibonacci(38))
```

Correction questions 2



Correction questions 3



On calcule à plusieurs reprises les *mêmes valeurs*, ici par exemple `fibo(2)` est calculé à trois reprises.

Mémoïsation

- La mémoïsation consiste à stocker dans une structure de données les valeurs renvoyées par une fonction afin de ne pas les recalculer lors des appels identiques suivant.

Exemple

Mémoïsation

- La mémoïsation consiste à stocker dans une structure de données les valeurs renvoyées par une fonction afin de ne pas les recalculer lors des appels identiques suivant.
- En Python, on utilise un dictionnaire dont les clés sont les arguments de la fonction et les valeurs les résultats de la fonction.

Exemple

Mémoïsation

- La mémoïsation consiste à stocker dans une structure de données les valeurs renvoyées par une fonction afin de ne pas les recalculer lors des appels identiques suivant.
- En Python, on utilise un dictionnaire dont les clés sont les arguments de la fonction et les valeurs les résultats de la fonction.

Exemple

Par exemple, si on stocke dans un dictionnaire la valeur de `fibo(2)` (clé : 2, valeur : 1), on n'a plus besoin de la recalculer lors des futurs appels.

Fibonnaci avec mémoïsation

```python
# Le dictionnaire pour mémoïser
memo = {}

def fibonacci(n):
    assert n>=0
    # Si la valeur se trouve dans le dictionnaire, elle a déjà été calculée
    if n in memo:
        return memo[n]
    # Sinon on calcule et on enregistre dans le dictionnaire
    if n<2:
        memo[n] = n
        return n
    memo[n] = fibonacci(n-1)+fibonacci(n-2)
    return memo[n]
```

Remarque

En python, la mémoïsation peut-être effectuée de façon automatique à l'aide du décorateur `lru_cache` du module `functools`. Après importation, on écrira simplement `@lru_cache` avant la définition de la fonction dont on veut mémoïser les appels.

Remarque

En python, la mémoïsation peut-être effectuée de façon automatique à l'aide du décorateur `lru_cache` du module `functools`. Après importation, on écrira simplement `@lru_cache` avant la définition de la fonction dont on veut mémoïser les appels.

Fibonacci mémoïsation automatique

```python
from functools import lru_cache

@lru_cache
def fibonacci(n):
    assert n>=0
    if n<2:
        return n
    return fibonacci(n-1)+fibonacci(n-2)
```

Position du problème

On considère une barre de métal de longueur entière 12 et pouvant être découpée en morceaux de longueurs entières ayant chacun un prix comme indiqué ci-dessous :

longueur	1	2	3	4	5	6	7	8	9	10	11	12
prix	2	4	7	8	12	14	18	23	24	25	26	31

Position du problème

On considère une barre de métal de longueur entière 12 et pouvant être découpée en morceaux de longueurs entières ayant chacun un prix comme indiqué ci-dessous :

longueur	1	2	3	4	5	6	7	8	9	10	11	12
prix	2	4	7	8	12	14	18	23	24	25	26	31

Le prix de vente des différents morceaux varie donc suivant la découpe utilisée, par exemples : la découpe $(2, 4, 6)$ a un prix de vente de $4 + 8 + 14 = 26$, tandis que la découpe $(7, 5)$ a un prix de vente de $18 + 12 = 30$

Position du problème

On considère une barre de métal de longueur entière 12 et pouvant être découpée en morceaux de longueurs entières ayant chacun un prix comme indiqué ci-dessous :

longueur	1	2	3	4	5	6	7	8	9	10	11	12
prix	2	4	7	8	12	14	18	23	24	25	26	31

Le prix de vente des différents morceaux varie donc suivant la découpe utilisée, par exemples : la découpe $(2, 4, 6)$ a un prix de vente de $4 + 8 + 14 = 26$, tandis que la découpe $(7, 5)$ a un prix de vente de $18 + 12 = 30$

Le but du problème est de trouver la valeur maximale des découpes possibles.

Position du problème

On considère une barre de métal de longueur entière 12 et pouvant être découpée en morceaux de longueurs entières ayant chacun un prix comme indiqué ci-dessous :

longueur	1	2	3	4	5	6	7	8	9	10	11	12
prix	2	4	7	8	12	14	18	23	24	25	26	31

Le prix de vente des différents morceaux varie donc suivant la découpe utilisée, par exemples : la découpe $(2, 4, 6)$ a un prix de vente de $4 + 8 + 14 = 26$, tandis que la découpe $(7, 5)$ a un prix de vente de $18 + 12 = 30$

Le but du problème est de trouver la valeur maximale des découpes possibles.

On note N la longueur de la barre, $(v_i)_{0 \leq i \leq N}$, la valeur maximale de la découpe d'une barre de taille i et $(p_i)_{0 \leq i \leq N}$ le prix d'un morceaux de longueur i.

Position du problème

On considère une barre de métal de longueur entière 12 et pouvant être découpée en morceaux de longueurs entières ayant chacun un prix comme indiqué ci-dessous :

longueur	1	2	3	4	5	6	7	8	9	10	11	12
prix	2	4	7	8	12	14	18	23	24	25	26	31

Le prix de vente des différents morceaux varie donc suivant la découpe utilisée, par exemples : la découpe $(2, 4, 6)$ a un prix de vente de $4 + 8 + 14 = 26$, tandis que la découpe $(7, 5)$ a un prix de vente de $18 + 12 = 30$

Le but du problème est de trouver la valeur maximale des découpes possibles.

On note N la longueur de la barre, $(v_i)_{0 \leq i \leq N}$, la valeur maximale de la découpe d'une barre de taille i et $(p_i)_{0 \leq i \leq N}$ le prix d'un morceaux de longueur i.

1. Donner les valeurs de v_0 et v_1.

Position du problème

On considère une barre de métal de longueur entière 12 et pouvant être découpée en morceaux de longueurs entières ayant chacun un prix comme indiqué ci-dessous :

longueur	1	2	3	4	5	6	7	8	9	10	11	12
prix	2	4	7	8	12	14	18	23	24	25	26	31

Le prix de vente des différents morceaux varie donc suivant la découpe utilisée, par exemples : la découpe $(2, 4, 6)$ a un prix de vente de $4 + 8 + 14 = 26$, tandis que la découpe $(7, 5)$ a un prix de vente de $18 + 12 = 30$

Le but du problème est de trouver la valeur maximale des découpes possibles.

On note N la longueur de la barre, $(v_i)_{0 \leq i \leq N}$, la valeur maximale de la découpe d'une barre de taille i et $(p_i)_{0 \leq i \leq N}$ le prix d'un morceaux de longueur i.

1. Donner les valeurs de v_0 et v_1.
2. Etablir une relation de récurrence liant les $(v_i)_{0 \leq i \leq N}$.

Position du problème

On considère une barre de métal de longueur entière 12 et pouvant être découpée en morceaux de longueurs entières ayant chacun un prix comme indiqué ci-dessous :

longueur	1	2	3	4	5	6	7	8	9	10	11	12
prix	2	4	7	8	12	14	18	23	24	25	26	31

Le prix de vente des différents morceaux varie donc suivant la découpe utilisée, par exemples : la découpe $(2, 4, 6)$ a un prix de vente de $4 + 8 + 14 = 26$, tandis que la découpe $(7, 5)$ a un prix de vente de $18 + 12 = 30$

Le but du problème est de trouver la valeur maximale des découpes possibles.

On note N la longueur de la barre, $(v_i)_{0 \leq i \leq N}$, la valeur maximale de la découpe d'une barre de taille i et $(p_i)_{0 \leq i \leq N}$ le prix d'un morceaux de longueur i.

1. Donner les valeurs de v_0 et v_1.
2. Etablir une relation de récurrence liant les $(v_i)_{0 \leq i \leq N}$.
3. En déduire une fonction Python récursive calculant la valeur de la découpe maximale.

Position du problème

On considère une barre de métal de longueur entière 12 et pouvant être découpée en morceaux de longueurs entières ayant chacun un prix comme indiqué ci-dessous :

longueur	1	2	3	4	5	6	7	8	9	10	11	12
prix	2	4	7	8	12	14	18	23	24	25	26	31

Le prix de vente des différents morceaux varie donc suivant la découpe utilisée, par exemples : la découpe $(2,4,6)$ a un prix de vente de $4+8+14=26$, tandis que la découpe $(7,5)$ a un prix de vente de $18+12=30$

Le but du problème est de trouver la valeur maximale des découpes possibles.

On note N la longueur de la barre, $(v_i)_{0\leq i\leq N}$, la valeur maximale de la découpe d'une barre de taille i et $(p_i)_{0\leq i\leq N}$ le prix d'un morceaux de longueur i.

1. Donner les valeurs de v_0 et v_1.
2. Etablir une relation de récurrence liant les $(v_i)_{0\leq i\leq N}$.
3. En déduire une fonction Python récursive calculant la valeur de la découpe maximale.
4. Utiliser la mémoïsation dans cette fonction.

Résolution

1. $v_0 = 0$ et $v_1 = 1$

Résolution

1. $v_0 = 0$ et $v_1 = 1$
2. En supposant qu'on connaisse les valeurs maximales de découpe pour les tailles inférieurs n, la découpe maximale pour la taille n s'en déduit en prenant le maximum parmi les découpes maximales d'une barre de longueur $k \leq n-1$ et d'un morceau de taille $n-k$, c'est à dire :
$v_n = \max\{v_k + p_{n-k}, 0 \leq k \leq n-1\}$

Résolution

1. $v_0 = 0$ et $v_1 = 1$
2. En supposant qu'on connaisse les valeurs maximales de découpe pour les tailles inférieurs n, la découpe maximale pour la taille n s'en déduit en prenant le maximum parmi les découpes maximales d'une barre de longueur $k \leq n-1$ et d'un morceau de taille $n-k$, c'est à dire :
$v_n = \max\{v_k + p_{n-k}, 0 \leq k \leq n-1\}$
3. Programme Python :

```python
def valeur_max(taille, prix):
    if taille<2:
        return prix[taille]
    else:
        return max(valeur_max(k,prix)+prix[taille-k] for k in range(taille))

print(valeur_max(12,[0, 2, 7, 11, 15, 17, 18, 23, 24, 27, 29, 33, 38]))
```

Résolution

1. $v_0 = 0$ et $v_1 = 1$
2. En supposant qu'on connaisse les valeurs maximales de découpe pour les tailles inférieurs n, la découpe maximale pour la taille n s'en déduit en prenant le maximum parmi les découpes maximales d'une barre de longueur $k \leq n-1$ et d'un morceau de taille $n-k$, c'est à dire :
$v_n = \max \{v_k + p_{n-k}, 0 \leq k \leq n-1\}$
3. Programme Python :

```python
def valeur_max(taille, prix):
    if taille<2:
        return prix[taille]
    else:
        return max(valeur_max(k,prix)+prix[taille-k] for k in range(taille))

print(valeur_max(12,[0, 2, 7, 11, 15, 17, 18, 23, 24, 27, 29, 33, 38]))
```

Le programme affiche 32.

Version avec mémoïsation

```python
vmax = {}
def valeur_max(taille, prix):
    if taille in vmax:
        return vmax[taille]
    if taille<2:
        vmax[taille] = prix[taille]
        return prix[taille]
    else:
        vmax[taille] = max(valeur_max(k,prix)+prix[taille-k] for k in range(taille))
        return vmax[taille]
```

Calcul de bas en haut (*bottom up*)

La mémoïsation construit la solution de façon "descendante", on lance les appels récursif sur les plus grandes valeurs de taille de la barre. Une autre stratégie dite ascendante ou de bas en haut (*bottom up*) consiste à construire la solution en partant des instances les plus petites du problème.

Calcul de bas en haut (*bottom up*)

La mémoïsation construit la solution de façon "descendante", on lance les appels récursif sur les plus grandes valeurs de taille de la barre. Une autre stratégie dite ascendante ou de bas en haut (*bottom up*) consiste à construire la solution en partant des instances les plus petites du problème.
Pour la découpe de la barre on part donc des valeurs connues v_0 et v_1 et on construit v_2 puis v_3, en utilisant la relation de récurrence
$v_n = \max\{v_k + p_{n-k}, 0 \leq k \leq n-1\}$

Calcul de bas en haut (*bottom up*)

La mémoïsation construit la solution de façon "descendante", on lance les appels récursif sur les plus grandes valeurs de taille de la barre. Une autre stratégie dite ascendante ou de bas en haut (*bottom up*) consiste à construire la solution en partant des instances les plus petites du problème.
Pour la découpe de la barre on part donc des valeurs connues v_0 et v_1 et on construit v_2 puis v_3, en utilisant la relation de récurrence
$v_n = \max \{v_k + p_{n-k}, 0 \leq k \leq n-1\}$
Ce qui se traduit en Python par une solution itérative :

Calcul de bas en haut (*bottom up*)

La mémoïsation construit la solution de façon "descendante", on lance les appels récursif sur les plus grandes valeurs de taille de la barre. Une autre stratégie dite ascendante ou de bas en haut (*bottom up*) consiste à construire la solution en partant des instances les plus petites du problème.
Pour la découpe de la barre on part donc des valeurs connues v_0 et v_1 et on construit v_2 puis v_3, en utilisant la relation de récurrence
$v_n = \max\{v_k + p_{n-k}, 0 \leq k \leq n-1\}$
Ce qui se traduit en Python par une solution itérative :

```
def valeur_max(taille, prix):
    v = {0:0,1:prix[1]}
    for i in range(2,taille+1):
        v[i] = max(v[k]+prix[i-k] for k in range(i))
    return v[taille]
```

Construction d'une solution

On a pour le moment déterminé la valeur maximale de la découpe, mais pas la découpe elle-même. D'autre part, plusieurs découpes différentes peuvent avoir cette même valeur maximale. Pour rechercher *une* découpe de valeur maximale, on peut par exemple :

- construire le tableau $(v_k)_{0 \leq k \leq \mathbb{N}}$ et l'utiliser afin d'en déduire la découpe.

Construction d'une solution

On a pour le moment déterminé la valeur maximale de la découpe, mais pas la découpe elle-même. D'autre part, plusieurs découpes différentes peuvent avoir cette même valeur maximale. Pour rechercher *une* découpe de valeur maximale, on peut par exemple :

- construire le tableau $(v_k)_{0 \leq k \leq \mathbb{N}}$ et l'utiliser afin d'en déduire la découpe.
  Par exemple, si $v_{12} = v_8 + p_4$, cela signifie que pour avoir la valeur maximale de la découpe d'une barre de taille 12, une possibilité est d'utiliser une découpe maximale d'une barre de taille 8 et un morceau de taille 4. En remontant ainsi de proche en proche, on obtient une découpe maximale possible

Construction d'une solution

On a pour le moment déterminé la valeur maximale de la découpe, mais pas la découpe elle-même. D'autre part, plusieurs découpes différentes peuvent avoir cette même valeur maximale. Pour rechercher *une* découpe de valeur maximale, on peut par exemple :

- construire le tableau $(v_k)_{0 \leq k \leq \mathbb{N}}$ et l'utiliser afin d'en déduire la découpe. Par exemple, si $v_{12} = v_8 + p_4$, cela signifie que pour avoir la valeur maximale de la découpe d'une barre de taille 12, une possibilité est d'utiliser une découpe maximale d'une barre de taille 8 et un morceau de taille 4. En remontant ainsi de proche en proche, on obtient une découpe maximale possible
- Modifier notre fonction afin qu'elle renvoie la découpe maximale et non pas la valeur de cette découpe.

Construction d'une solution

On a pour le moment déterminé la valeur maximale de la découpe, mais pas la découpe elle-même. D'autre part, plusieurs découpes différentes peuvent avoir cette même valeur maximale. Pour rechercher *une* découpe de valeur maximale, on peut par exemple :

- construire le tableau $(v_k)_{0 \leq k \leq \mathbb{N}}$ et l'utiliser afin d'en déduire la découpe. Par exemple, si $v_{12} = v_8 + p_4$, cela signifie que pour avoir la valeur maximale de la découpe d'une barre de taille 12, une possibilité est d'utiliser une découpe maximale d'une barre de taille 8 et un morceau de taille 4. En remontant ainsi de proche en proche, on obtient une découpe maximale possible
- Modifier notre fonction afin qu'elle renvoie la découpe maximale et non pas la valeur de cette découpe.

Ces deux possibilités seront abordées en TP.

Principes généraux

La programmation dynamique s'applique généralement à la résolution d'un problème d'optimisation vérifiant les conditions suivantes :

Principes généraux

La programmation dynamique s'applique généralement à la résolution d'un problème d'optimisation vérifiant les conditions suivantes :

1. Sous structure optimale : ce problème peut-être résolu à partir de problèmes similaires mais plus petits

Principes généraux

La programmation dynamique s'applique généralement à la résolution d'un problème d'optimisation vérifiant les conditions suivantes :

1. Sous structure optimale : ce problème peut-être résolu à partir de problèmes similaires mais plus petits
   La découpe maximale d'une barre de taille N s'obtient comme découpe maximale d'une barre de taille strictement inférieure k et d'un morceau de taille $N - k$.

Principes généraux

La programmation dynamique s'applique généralement à la résolution d'un problème d'optimisation vérifiant les conditions suivantes :

1. Sous structure optimale : ce problème peut-être résolu à partir de problèmes similaires mais plus petits
   La découpe maximale d'une barre de taille N s'obtient comme découpe maximale d'une barre de taille strictement inférieure k et d'un morceau de taille $N-k$.
2. Chevauchement de sous problème : une solution récursive produit des appels identiques. Pour pallier ce problème, on utilise la mémoïsation dans les solutions récursives.

Principes généraux

La programmation dynamique s'applique généralement à la résolution d'un problème d'optimisation vérifiant les conditions suivantes :

1. Sous structure optimale : ce problème peut-être résolu à partir de problèmes similaires mais plus petits
   La découpe maximale d'une barre de taille N s'obtient comme découpe maximale d'une barre de taille strictement inférieure k et d'un morceau de taille $N-k$.
2. Chevauchement de sous problème : une solution récursive produit des appels identiques. Pour pallier ce problème, on utilise la mémoïsation dans les solutions récursives.
   Pour rechercher la découpe maximale d'un barre de taille 5, on est amené à chercher celle d'une barre de taille 4,3,2,1. Et pour chercher celle d'une barre de taille 4, on fera de nouveau appel à celle d'une barre de taille 3,2,1 ...

Principes généraux

La programmation dynamique s'applique généralement à la résolution d'un problème d'optimisation vérifiant les conditions suivantes :

1. Sous structure optimale : ce problème peut-être résolu à partir de problèmes similaires mais plus petits
   La découpe maximale d'une barre de taille N s'obtient comme découpe maximale d'une barre de taille strictement inférieure k et d'un morceau de taille $N-k$.
2. Chevauchement de sous problème : une solution récursive produit des appels identiques. Pour pallier ce problème, on utilise la mémoïsation dans les solutions récursives.
   Pour rechercher la découpe maximale d'un barre de taille 5, on est amené à chercher celle d'une barre de taille 4,3,2,1. Et pour chercher celle d'une barre de taille 4, on fera de nouveau appel à celle d'une barre de taille 3,2,1 ...

! L'étape cruciale est de déterminer les relations de récurrence entre les différentes instances du problème. Les différentes méthodes d'implémentation relèvent du choix du programmeur.

Position du problème

On considère deux chaines de caractères u et v de longueurs respectives n et m.
On cherche à déterminer la longueur de leur plus longue sous séquence commune (plssc), c'est à dire la chaine w telle que :

Position du problème

On considère deux chaines de caractères u et v de longueurs respectives n et m.
On cherche à déterminer la longueur de leur plus longue sous séquence commune (plssc), c'est à dire la chaine w telle que :

- w est une sous séquence (c'est à dire une suite extraite) de u,

Position du problème

On considère deux chaines de caractères u et v de longueurs respectives n et m.
On cherche à déterminer la longueur de leur plus longue sous séquence commune (plssc), c'est à dire la chaine w telle que :

- w est une sous séquence (c'est à dire une suite extraite) de u,
- w est une sous séquence de w,

Position du problème

On considère deux chaines de caractères u et v de longueurs respectives n et m. On cherche à déterminer la longueur de leur plus longue sous séquence commune (plssc), c'est à dire la chaine w telle que :

- w est une sous séquence (c'est à dire une suite extraite) de u,
- w est une sous séquence de w,
- w est de longueur maximale.

Position du problème

On considère deux chaines de caractères u et v de longueurs respectives n et m. On cherche à déterminer la longueur de leur plus longue sous séquence commune (plssc), c'est à dire la chaine w telle que :

- w est une sous séquence (c'est à dire une suite extraite) de u,
- w est une sous séquence de w,
- w est de longueur maximale.

Par exemple, u="PROGRAMMATION" et v="DYNAMIQUE" ont comme sous séquence commune "AMI" (et c'est la plus longue)

Position du problème

On considère deux chaines de caractères u et v de longueurs respectives n et m. On cherche à déterminer la longueur de leur plus longue sous séquence commune (plssc), c'est à dire la chaine w telle que :

- w est une sous séquence (c'est à dire une suite extraite) de u,
- w est une sous séquence de w,
- w est de longueur maximale.

Par exemple, u="PROGRAMMATION" et v="DYNAMIQUE" ont comme sous séquence commune "AMI" (et c'est la plus longue)

- PROGRAMMATION

Position du problème

On considère deux chaines de caractères u et v de longueurs respectives n et m. On cherche à déterminer la longueur de leur plus longue sous séquence commune (plssc), c'est à dire la chaine w telle que :

- w est une sous séquence (c'est à dire une suite extraite) de u,
- w est une sous séquence de w,
- w est de longueur maximale.

Par exemple, u="PROGRAMMATION" et v="DYNAMIQUE" ont comme sous séquence commune "AMI" (et c'est la plus longue)

- PROGRAMMATION
- DYNAMIQUE

Position du problème

On considère deux chaines de caractères u et v de longueurs respectives n et m. On cherche à déterminer la longueur de leur plus longue sous séquence commune (plssc), c'est à dire la chaine w telle que :

- w est une sous séquence (c'est à dire une suite extraite) de u,
- w est une sous séquence de w,
- w est de longueur maximale.

Par exemple, u="PROGRAMMATION" et v="DYNAMIQUE" ont comme sous séquence commune "AMI" (et c'est la plus longue)

- PROGRAMMATION
- DYNAMIQUE

Donc ici, la longueur de la plssc est 3.

Résolution

On cherche les relations de récurrence entre des instances du sous-problème. Pour cela on note u_i $(0 \leq i \leq n)$ la chaine composée des i premiers caractères de u, et v_j $(0 \leq j \leq m)$ celle composée des j premiers caractères de v. Et on note $\text{lplssc}(u_i, v_j)$ la longueur de la plssc de u_i et de v_j.

Résolution

On cherche les relations de récurrence entre des instances du sous-problème. Pour cela on note u_i $(0 \leq i \leq n)$ la chaine composée des i premiers caractères de u, et v_j $(0 \leq j \leq m)$ celle composée des j premiers caractères de v. Et on note $\mathrm{lplssc}(u_i, v_j)$ la longueur de la plssc de u_i et de v_j.

- Si $u[i] = v[j]$ alors, quelle est la relation entre $\mathrm{lplssc}(u_i, v_j)$ et $\mathrm{lplssc}(u_{i-1}, v_{j-1})$?

Résolution

On cherche les relations de récurrence entre des instances du sous-problème. Pour cela on note u_i $(0 \leq i \leq n)$ la chaine composée des i premiers caractères de u, et v_j $(0 \leq j \leq m)$ celle composée des j premiers caractères de v. Et on note $\mathrm{lplssc}(u_i, v_j)$ la longueur de la plssc de u_i et de v_j.

- Si $u[i] = v[j]$ alors, quelle est la relation entre $\mathrm{lplssc}(u_i, v_j)$ et $\mathrm{lplssc}(u_{i-1}, v_{j-1})$?
- Sinon, exprimer $\mathrm{plssc}(u_i, v_j)$ en fonction de $\mathrm{plssc}(u_i, v_{j-1})$ et $\mathrm{plssc}(u_{i-1}, v_j)$

Résolution

On cherche les relations de récurrence entre des instances du sous-problème. Pour cela on note u_i $(0 \leq i \leq n)$ la chaine composée des i premiers caractères de u, et v_j $(0 \leq j \leq m)$ celle composée des j premiers caractères de v. Et on note $\text{lplssc}(u_i, v_j)$ la longueur de la plssc de u_i et de v_j.

- Si $u[i] = v[j]$ alors, quelle est la relation entre $\text{lplssc}(u_i, v_j)$ et $\text{lplssc}(u_{i-1}, v_{j-1})$?
- Sinon, exprimer $\text{plssc}(u_i, v_j)$ en fonction de $\text{plssc}(u_i, v_{j-1})$ et $\text{plssc}(u_{i-1}, v_j)$
- Déterminer les cas de base (ceux où u et v sont des chaines vides notés "") :

Résolution

On cherche les relations de récurrence entre des instances du sous-problème. Pour cela on note u_i $(0 \leq i \leq n)$ la chaine composée des i premiers caractères de u, et v_j $(0 \leq j \leq m)$ celle composée des j premiers caractères de v. Et on note $\text{lplssc}(u_i, v_j)$ la longueur de la plssc de u_i et de v_j.

- Si $u[i] = v[j]$ alors, quelle est la relation entre $\text{lplssc}(u_i, v_j)$ et $\text{lplssc}(u_{i-1}, v_{j-1})$?
  $\text{lplssc}(u_i, v_j) = 1 + \text{lplssc}(u_{i-1}, v_{j-1})$
- Sinon, exprimer $\text{plssc}(u_i, v_j)$ en fonction de $\text{plssc}(u_i, v_{j-1})$ et $\text{plssc}(u_{i-1}, v_j)$
- Déterminer les cas de base (ceux où u et v sont des chaines vides notés "") :

Résolution

On cherche les relations de récurrence entre des instances du sous-problème. Pour cela on note u_i $(0 \leq i \leq n)$ la chaine composée des i premiers caractères de u, et v_j $(0 \leq j \leq m)$ celle composée des j premiers caractères de v. Et on note $\mathrm{lplssc}(u_i, v_j)$ la longueur de la plssc de u_i et de v_j.

- Si $u[i] = v[j]$ alors, quelle est la relation entre $\mathrm{lplssc}(u_i, v_j)$ et $\mathrm{lplssc}(u_{i-1}, v_{j-1})$?
  $\mathrm{lplssc}(u_i, v_j) = 1 + \mathrm{lplssc}(u_{i-1}, v_{j-1})$
- Sinon, exprimer $\mathrm{plssc}(u_i, v_j)$ en fonction de $\mathrm{plssc}(u_i, v_{j-1})$ et $\mathrm{plssc}(u_{i-1}, v_j)$
  $\mathrm{lplssc}(u_i, v_j) = \max\left(\mathrm{lplssc}(u_i, v_{j-1}), \mathrm{lplssc}(u_{i-1}, v_j)\right)$
- Déterminer les cas de base (ceux où u et v sont des chaines vides notés "") :

Résolution

On cherche les relations de récurrence entre des instances du sous-problème. Pour cela on note u_i $(0 \leq i \leq n)$ la chaine composée des i premiers caractères de u, et v_j $(0 \leq j \leq m)$ celle composée des j premiers caractères de v. Et on note $\mathrm{lplssc}(u_i, v_j)$ la longueur de la plssc de u_i et de v_j.

- Si $u[i] = v[j]$ alors, quelle est la relation entre $\mathrm{lplssc}(u_i, v_j)$ et $\mathrm{lplssc}(u_{i-1}, v_{j-1})$?
  $\mathrm{lplssc}(u_i, v_j) = 1 + \mathrm{lplssc}(u_{i-1}, v_{j-1})$
- Sinon, exprimer $\mathrm{plssc}(u_i, v_j)$ en fonction de $\mathrm{plssc}(u_i, v_{j-1})$ et $\mathrm{plssc}(u_{i-1}, v_j)$
  $\mathrm{lplssc}(u_i, v_j) = \max\left(\mathrm{lplssc}(u_i, v_{j-1}), \mathrm{lplssc}(u_{i-1}, v_j)\right)$
- Déterminer les cas de base (ceux où u et v sont des chaines vides notés "") :
  $\mathrm{lplssc}(u_i, ””) = 0$
  $\mathrm{lplssc}(””, v_j) = 0$

Programme Python pour plssc

```python
from functools import lru_cache

@lru_cache
def lplssc(mot1,mot2):
    if len(mot2)==0 or len(mot1)==0:
        return 0
    if mot1[-1]==mot2[-1]:
        return 1 + lplssc(mot1[:-1],mot2[:-1])
    return max(lplssc(mot1[:-1],mot2),lplssc(mot1,mot2[:-1]))
```

Construction effective d'une solution

On peut modifier notre programme donnant la longueur de la plssc afin d'obtenir la plssc :

- Si le dernier caractère est commun, il fait partie de la plssc et on relance la récursion sur les parties restantes de chaque chaîne

Construction effective d'une solution

On peut modifier notre programme donnant la longueur de la plssc afin d'obtenir la plssc :

- Si le dernier caractère est commun, il fait partie de la plssc et on relance la récursion sur les parties restantes de chaque chaîne
- Sinon on regarde quel appel produit la plssc et on renvoie le résultat de cet appel

Construction effective d'une solution

On peut modifier notre programme donnant la longueur de la plssc afin d'obtenir la plssc :

- Si le dernier caractère est commun, il fait partie de la plssc et on relance la récursion sur les parties restantes de chaque chaîne
- Sinon on regarde quel appel produit la plssc et on renvoie le résultat de cet appel

```python
@lru_cache
def plssc(mot1,mot2):
    if len(mot2)==0 or len(mot1)==0:
        return ""
    if mot1[-1]==mot2[-1]:
        return plssc(mot1[:-1],mot2[:-1])+mot1[-1]
    if len(plssc(mot1[:-1],mot2))>len(plssc(mot1,mot2[:-1])):
        return  plssc(mot1[:-1],mot2)
    else:
        return plssc(mot1,mot2[:-1])
```

Position du problème

On dispose d'un *système monétaire* c'est à dire d'un ensemble de valeurs possibles pour les pièces et les billets. Le problème du rendu de monnaie consiste à déterminer le nombre minimal de pièces à utiliser pour former une somme donnée.

Position du problème

On dispose d'un *système monétaire* c'est à dire d'un ensemble de valeurs possibles pour les pièces et les billets. Le problème du rendu de monnaie consiste à déterminer le nombre minimal de pièces à utiliser pour former une somme donnée. Par exemple si le système monétaire est $\{5, 4, 3, 1\}$ et la somme 7,

Position du problème

On dispose d'un *système monétaire* c'est à dire d'un ensemble de valeurs possibles pour les pièces et les billets. Le problème du rendu de monnaie consiste à déterminer le nombre minimal de pièces à utiliser pour former une somme donnée. Par exemple si le système monétaire est $\{5, 4, 3, 1\}$ et la somme 7, alors on peut utiliser au minimum 2 pièces $(4 + 3)$.

Position du problème

On dispose d'un *système monétaire* c'est à dire d'un ensemble de valeurs possibles pour les pièces et les billets. Le problème du rendu de monnaie consiste à déterminer le nombre minimal de pièces à utiliser pour former une somme donnée. Par exemple si le système monétaire est $\{5, 4, 3, 1\}$ et la somme 7, alors on peut utiliser au minimum 2 pièces $(4 + 3)$.

Position du problème

On dispose d'un *système monétaire* c'est à dire d'un ensemble de valeurs possibles pour les pièces et les billets. Le problème du rendu de monnaie consiste à déterminer le nombre minimal de pièces à utiliser pour former une somme donnée. Par exemple si le système monétaire est $\{5, 4, 3, 1\}$ et la somme 7, alors on peut utiliser au minimum 2 pièces $(4 + 3)$.

Rappel : l'algorithme glouton qui consiste à rendre à tout moment la pièce de plus forte valeur possible ne fournit pas toujours la solution optimale. Ici, on obtiendrait $5, 1, 1$ et donc 3 pièces.

Position du problème

On dispose d'un *système monétaire* c'est à dire d'un ensemble de valeurs possibles pour les pièces et les billets. Le problème du rendu de monnaie consiste à déterminer le nombre minimal de pièces à utiliser pour former une somme donnée. Par exemple si le système monétaire est $\{5, 4, 3, 1\}$ et la somme 7, alors on peut utiliser au minimum 2 pièces $(4 + 3)$.

Rappel : l'algorithme glouton qui consiste à rendre à tout moment la pièce de plus forte valeur possible ne fournit pas toujours la solution optimale. Ici, on obtiendrait $5, 1, 1$ et donc 3 pièces.

1. Ecrire une relation de récurrence entre les différentes instances du problème en donnant les solutions des cas de base.

Position du problème

On dispose d'un *système monétaire* c'est à dire d'un ensemble de valeurs possibles pour les pièces et les billets. Le problème du rendu de monnaie consiste à déterminer le nombre minimal de pièces à utiliser pour former une somme donnée. Par exemple si le système monétaire est $\{5, 4, 3, 1\}$ et la somme 7, alors on peut utiliser au minimum 2 pièces $(4 + 3)$.

Rappel : l'algorithme glouton qui consiste à rendre à tout moment la pièce de plus forte valeur possible ne fournit pas toujours la solution optimale. Ici, on obtiendrait $5, 1, 1$ et donc 3 pièces.

1. Ecrire une relation de récurrence entre les différentes instances du problème en donnant les solutions des cas de base.
2. Ecrire un programme python permettant de répondre au problème.

Position du problème

On dispose d'un *système monétaire* c'est à dire d'un ensemble de valeurs possibles pour les pièces et les billets. Le problème du rendu de monnaie consiste à déterminer le nombre minimal de pièces à utiliser pour former une somme donnée. Par exemple si le système monétaire est $\{5, 4, 3, 1\}$ et la somme 7, alors on peut utiliser au minimum 2 pièces $(4 + 3)$.

Rappel : l'algorithme glouton qui consiste à rendre à tout moment la pièce de plus forte valeur possible ne fournit pas toujours la solution optimale. Ici, on obtiendrait $5, 1, 1$ et donc 3 pièces.

1. Ecrire une relation de récurrence entre les différentes instances du problème en donnant les solutions des cas de base.
2. Ecrire un programme python permettant de répondre au problème.
3. Construire la liste effective des pièces à rendre.

Résolution

Résolution

1. On note :

Résolution

1. On note :
   - S la somme à rendre,

Résolution

1. On note :
   - S la somme à rendre,
   - $(p_i)_{0\leq i\leq n}$ les valeurs des pièces rangées dans l'ordre décroissant

Résolution

1. On note :
    - S la somme à rendre,
    - $(p_i)_{0 \leq i \leq n}$ les valeurs des pièces rangées dans l'ordre décroissant
    - $m(S, k)$ le nombre minimal de pièce pour rendre la somme S en utilisant les pièces à partir de la k-ième.

Résolution

1. On note :
   - S la somme à rendre,
   - $(p_i)_{0 \leq i \leq n}$ les valeurs des pièces rangées dans l'ordre décroissant
   - $m(S, k)$ le nombre minimal de pièce pour rendre la somme S en utilisant les pièces à partir de la k-ième.

   Avec ces notations, on doit donc trouver $m(S, 0)$ et on dispose des relations suivantes :

Résolution

1. On note :
   - S la somme à rendre,
   - $(p_i)_{0 \leq i \leq n}$ les valeurs des pièces rangées dans l'ordre décroissant
   - $m(S, k)$ le nombre minimal de pièce pour rendre la somme S en utilisant les pièces à partir de la k-ième.

   Avec ces notations, on doit donc trouver $m(S, 0)$ et on dispose des relations suivantes :

$$\begin{cases} m(0, k) & = & 0 \text{ pour tout } 0 \leq k \leq n, \\ m(S, n+1) & = & +\infty \\ m(S, k) & = & m(S, k+1) \text{ si } S < p_k, \\ m(S, k) & = & min \left\{ 1 + m(S - p_k, k), m(S, k+1) \ \right\} sinon. \end{cases}$$

Résolution

2. Programme Python (avec mémoïsation dans le dictionnaire `memo`)

```python
def rm(somme,k):
    if (somme,k) in memo:
        return memo[(somme,k)]
    if k>=len(systeme):
        memo[(somme,k)]=inf
        return inf
    if systeme[k]>somme:
        memo[(somme,k)] = rm(somme,k+1)
        return memo[(somme,k)]
    memo[(somme,k)] = min(1+rm(somme-systeme[k],k),rm(somme,k+1))
    return memo[(somme,k)]
```

Résolution

3. Pour construire la liste des pièces à rendre, on peut partir des valeurs de $m(S,k)$:

$S \backslash k$	0	1	2	3	4
0	0	0	0	0	∞
1					∞
2					∞
3					∞
4					∞
5					∞
6					∞
7					∞

$$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$$

Résolution

3. Pour construire la liste des pièces à rendre, on peut partir des valeurs de $m(S,k)$:

S \ k	0	1	2	3	4
0	0	0	0	0	∞
1	1	1	1	1	∞
2					∞
3					∞
4					∞
5					∞
6					∞
7					∞

$$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$$

Résolution

3. Pour construire la liste des pièces à rendre, on peut partir des valeurs de $m(S,k)$:

$S \backslash k$	0	1	2	3	4
0	0	0	0	0	∞
1	1	1	1	1	∞
2	2	2	2	2	∞
3					∞
4					∞
5					∞
6					∞
7					∞

$$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$$

Résolution

3. Pour construire la liste des pièces à rendre, on peut partir des valeurs de $m(S,k)$:

$S \backslash k$	0	1	2	3	4
0	0	0	0	0	∞
1	1	1	1	1	∞
2	2	2	2	2	∞
3	1	1	1	3	∞
4					∞
5					∞
6					∞
7					∞

$$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$$

Résolution

3. Pour construire la liste des pièces à rendre, on peut partir des valeurs de $m(S, k)$:

$S \backslash k$	0	1	2	3	4
0	0	0	0	0	∞
1	1	1	1	1	∞
2	2	2	2	2	∞
3	1	1	1	3	∞
4	1	1	2	4	∞
5					∞
6					∞
7					∞

$$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$$

Résolution

3. Pour construire la liste des pièces à rendre, on peut partir des valeurs de $m(S,k)$:

S \ k	0	1	2	3	4
0	0	0	0	0	∞
1	1	1	1	1	∞
2	2	2	2	2	∞
3	1	1	1	3	∞
4	1	1	2	4	∞
5	1	2	3	5	∞
6					∞
7					∞

$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$

Résolution

3. Pour construire la liste des pièces à rendre, on peut partir des valeurs de $m(S, k)$:

$S \backslash k$	0	1	2	3	4
0	0	0	0	0	∞
1	1	1	1	1	∞
2	2	2	2	2	∞
3	1	1	1	3	∞
4	1	1	2	4	∞
5	1	2	3	5	∞
6	2	2	2	6	∞
7					∞

$$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$$

Résolution

3. Pour construire la liste des pièces à rendre, on peut partir des valeurs de $m(S,k)$:

S \ k	0	1	2	3	4
0	0	0	0	0	∞
1	1	1	1	1	∞
2	2	2	2	2	∞
3	1	1	1	3	∞
4	1	1	2	4	∞
5	1	2	3	5	∞
6	2	2	2	6	∞
7	2	2	3	7	∞

$$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$$

Résolution

3. Pour construire la liste des pièces à rendre, on peut partir des valeurs de $m(S,k)$:

S \ k	0	1	2	3	4
0	0	0	0	0	∞
1	1	1	1	1	∞
2	2	2	2	2	∞
3	1	1	1	3	∞
4	1	1	2	4	∞
5	1	2	3	5	∞
6	2	2	2	6	∞
7	2	$\boxed{2}$	3	7	∞

$$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$$

Résolution

3. Pour construire la liste des pièces à rendre, on peut partir des valeurs de $m(S,k)$:

$S \backslash k$	0	$\boxed{1}$	2	3	4
0	0	0	0	0	∞
1	1	1	1	1	∞
2	2	2	2	2	∞
3	1	1	1	3	∞
4	1	1	2	4	∞
5	1	2	3	5	∞
6	2	2	2	6	∞
7	2	$\boxed{2}$	3	7	∞

$$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$$

Résolution

3. Pour construire la liste des pièces à rendre, on peut partir des valeurs de $m(S,k)$:

S \ k	0	$\boxed{1}$	2	3	4
0	0	0	0	0	∞
1	1	1	1	1	∞
2	2	2	2	2	∞
3	1	1	1	3	∞
4	1	1	2	4	∞
5	1	2	3	5	∞
6	2	2	2	6	∞
7	2	$\boxed{2}$	3	7	∞

$$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$$

Résolution

3. Pour construire la liste des pièces à rendre, on peut partir des valeurs de $m(S, k)$:

$S \backslash k$	0	$\boxed{1}$	2	3	4
0	0	0	0	0	∞
1	1	1	1	1	∞
2	2	2	2	2	∞
3	1	1	$\boxed{1}$	3	∞
4	1	1	2	4	∞
5	1	2	3	5	∞
6	2	2	2	6	∞
7	2	$\boxed{2}$	3	7	∞

$$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$$

Résolution

3. Pour construire la liste des pièces à rendre, on peut partir des valeurs de $m(S,k)$:

$S \backslash k$	0	$\boxed{1}$	$\boxed{2}$	3	4
0	0	0	0	0	∞
1	1	1	1	1	∞
2	2	2	2	2	∞
3	1	1	$\boxed{1}$	3	∞
4	1	1	2	4	∞
5	1	2	3	5	∞
6	2	2	2	6	∞
7	2	$\boxed{2}$	3	7	∞

$$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$$

Résolution

3. Cette méthode correspond au programme suivant :

```python
def pieces(memo,somme):
    pieces = []
    while somme!=0:
        min = inf
        piece = None
        for i in range(len(systeme)+1,-1,-1):
            if (somme,i) in memo and memo[(somme,i)]<min:
                min = memo[(somme,i)]
                piece = systeme[i]
        somme = somme-piece
        pieces.append(piece)
    return pieces
```

Résolution

Remarque : sur cet exemple, les appels récursifs depuis $m(7,0)$ ne construisent pas la totalité de la matrice des valeurs de $m(S,k)$. En effet, le calcul de $m(7,0)$ se fait à partir de celui de $m(7,1)$ et de $m(7-p_0,0)$ et ainsi de suite en remontant dans la matrice :

$S \backslash k$	0	1	2	3	4
0	0	0	0	0	∞
1					∞
2					∞
3					∞
4					∞
5					∞
6					∞
7	✓				∞

$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$

Résolution

Remarque : sur cet exemple, les appels récursifs depuis $m(7,0)$ ne construisent pas la totalité de la matrice des valeurs de $m(S,k)$. En effet, le calcul de $m(7,0)$ se fait à partir de celui de $m(7,1)$ et de $m(7-p_0,0)$ et ainsi de suite en remontant dans la matrice :

S \ k	0	1	2	3	4
0	0	0	0	0	∞
1					∞
2	✓				∞
3					∞
4					∞
5					∞
6					∞
7	✓	✓			∞

$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$

Résolution

Remarque : sur cet exemple, les appels récursifs depuis $m(7,0)$ ne construisent pas la totalité de la matrice des valeurs de $m(S,k)$. En effet, le calcul de $m(7,0)$ se fait à partir de celui de $m(7,1)$ et de $m(7-p_0,0)$ et ainsi de suite en remontant dans la matrice :

S \ k	0	1	2	3	4
0	0	0	0	0	∞
1					∞
2	✓	✓			∞
3		✓			∞
4					∞
5					∞
6					∞
7	✓	✓	✓		∞

$$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$$

Résolution

Remarque : sur cet exemple, les appels récursifs depuis $m(7,0)$ ne construisent pas la totalité de la matrice des valeurs de $m(S,k)$. En effet, le calcul de $m(7,0)$ se fait à partir de celui de $m(7,1)$ et de $m(7-p_0,0)$ et ainsi de suite en remontant dans la matrice :

S \ k	0	1	2	3	4
0	0	0	0	0	∞
1					∞
2	✓	✓	✓		∞
3		✓	✓		∞
4			✓		∞
5					∞
6					∞
7	✓	✓	✓	✓	∞

$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$

Résolution

Remarque : sur cet exemple, les appels récursifs depuis $m(7,0)$ ne construisent pas la totalité de la matrice des valeurs de $m(S,k)$. En effet, le calcul de $m(7,0)$ se fait à partir de celui de $m(7,1)$ et de $m(7-p_0,0)$ et ainsi de suite en remontant dans la matrice :

$S \backslash k$	0	1	2	3	4
0	0	0	0	0	∞
1					∞
2	✓	✓	✓	✓	∞
3		✓	✓	✓	∞
4			✓	✓	∞
5				✓	∞
6					∞
7	✓	✓	✓	✓	∞

$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$

Résolution

Remarque : sur cet exemple, les appels récursifs depuis $m(7,0)$ ne construisent pas la totalité de la matrice des valeurs de $m(S,k)$. En effet, le calcul de $m(7,0)$ se fait à partir de celui de $m(7,1)$ et de $m(7-p_0,0)$ et ainsi de suite en remontant dans la matrice :

S \ k	0	1	2	3	4
0	0	0	0	0	∞
1					∞
2	✓	✓	✓	✓	∞
3		✓	✓	✓	∞
4			✓	✓	∞
5				✓	∞
6				✓	∞
7	✓	✓	✓	✓	∞

$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$

Résolution

Remarque : sur cet exemple, les appels récursifs depuis $m(7,0)$ ne construisent pas la totalité de la matrice des valeurs de $m(S,k)$. En effet, le calcul de $m(7,0)$ se fait à partir de celui de $m(7,1)$ et de $m(7-p_0,0)$ et ainsi de suite en remontant dans la matrice :

S \ k	0	1	2	3	4
0	0	0	0	0	∞
1				✓	∞
2	✓	✓	✓	✓	∞
3		✓	✓	✓	∞
4			✓	✓	∞
5				✓	∞
6				✓	∞
7	✓	✓	✓	✓	∞

$\boxed{p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1}$