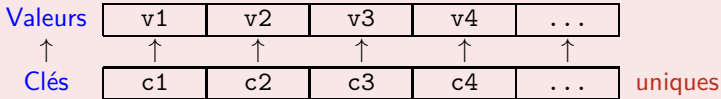


# Les dictionnaires de Python

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



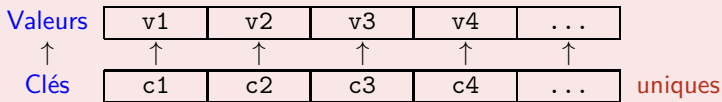
## Examples

# C18 Programmation dynamique

## 1. Les dictionnaires de Python

### Les dictionnaires de Python

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



- Un dictionnaire se note entre accolades : { et }

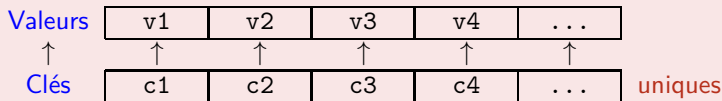
### Exemples

# C18 Programmation dynamique

## 1. Les dictionnaires de Python

### Les dictionnaires de Python

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



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

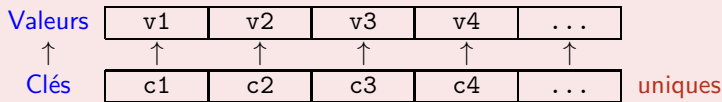
### Exemples

# C18 Programmation dynamique

## 1. Les dictionnaires de Python

### Les dictionnaires de Python

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



- 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.

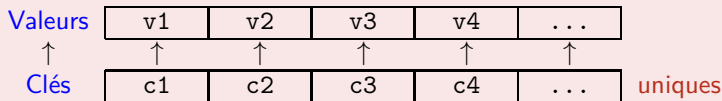
### Exemples

# C18 Programmation dynamique

## 1. Les dictionnaires de Python

### Les dictionnaires de Python

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



- 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

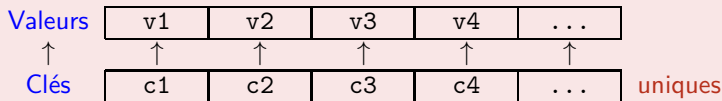
- Un dictionnaire contenant des objets et leurs prix :

# C18 Programmation dynamique

## 1. Les dictionnaires de Python

### Les dictionnaires de Python

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



- 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 :  

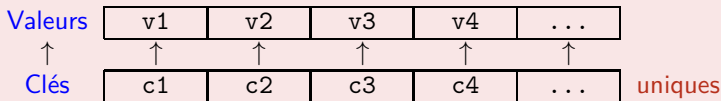
```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16}
```

# C18 Programmation dynamique

## 1. Les dictionnaires de Python

### Les dictionnaires de Python

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



- 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

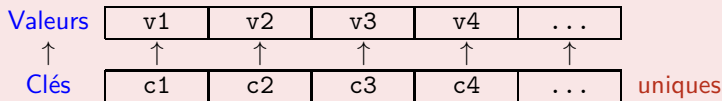
- Un dictionnaire contenant des objets et leurs prix :  
`prix = { "verre":12 , "tasse" : 8, "assiette" : 16}`
- Un dictionnaire traduisant des couleurs du français vers l'anglais

# C18 Programmation dynamique

## 1. Les dictionnaires de Python

### Les dictionnaires de Python

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



- 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'occurrence 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'occurrence 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.

- ➊ Proposer une solution qui pour chaque élément de la liste calcule son nombre d'apparitions à l'aide d'une fonction `compte_occurrence`
- ➋ 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'occurrence 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.

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

### Correction question 1

```
1  def compte_occurence(elt,liste):
2      occ = 0
3      for x in liste:
4          if x==elt:
5              occ+=1
6      return occ
7
8  def plus_frequent(liste):
9      max_occ = 0
10     for elt in liste:
11         elt_occ = compte_occurence(elt,liste)
12         if elt_occ>max_occ:
13             max_occ = elt_occ
14     return max_occ
15
```

### Correction question 2

```
1 def plus_frequent(liste):
2     nb_occ = {}
3     for elt in liste:
4         if elt not in nb_occ:
5             nb_occ[elt] = 1
6         else:
7             nb_occ[elt] += 1
8     return max(nb_occ[elt] for elt in nb_occ)
```

### Correction question 2

```
1 def plus_frequent(liste):
2     nb_occ = {}
3     for elt in liste:
4         if elt not in nb_occ:
5             nb_occ[elt] = 1
6         else:
7             nb_occ[elt] += 1
8     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  $\llbracket 0; N - 1 \rrbracket$ ).

### Implémentation des dictionnaires

- On crée un tableau  $T$  de liste de longueur  $N$  (donc indicé par les entiers  $\llbracket 0; N - 1 \rrbracket$ ).
- 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  $\llbracket 0; N - 1 \rrbracket$  à l'aide d'un modulo.

### Implémentation des dictionnaires

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

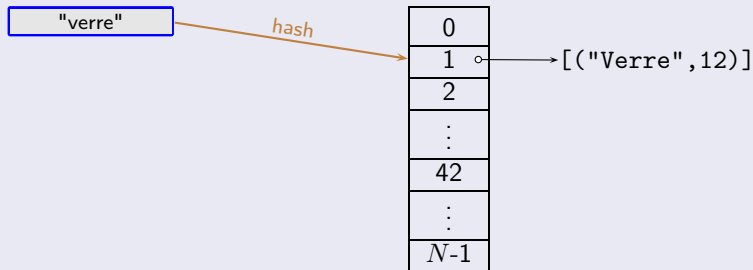
- On crée un tableau  $T$  de liste de longueur  $N$  (donc indicé par les entiers  $\llbracket 0; N - 1 \rrbracket$ ).
- Une **fonction de hachage**  $h$  transforme les clés en entier. Les clés doivent donc être **non mutables** (ce qui exclu les listes). Ces entiers sont ramenés dans l'intervalle  $\llbracket 0; N - 1 \rrbracket$  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 deux clés différentes  $c1$  et  $c2$  produisent le même indice s'appelle une **collision**.

# C18 Programmation dynamique

## 2. Table de hachage

### Visualisation

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

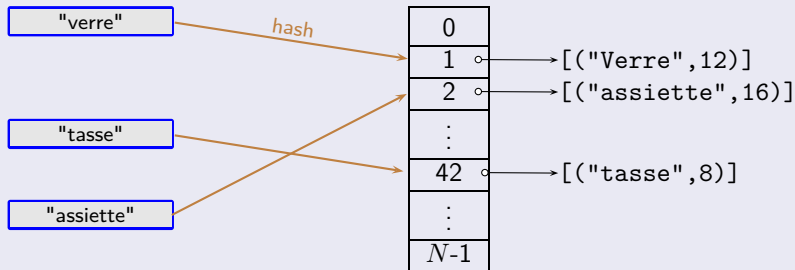


# C18 Programmation dynamique

## 2. Table de hachage

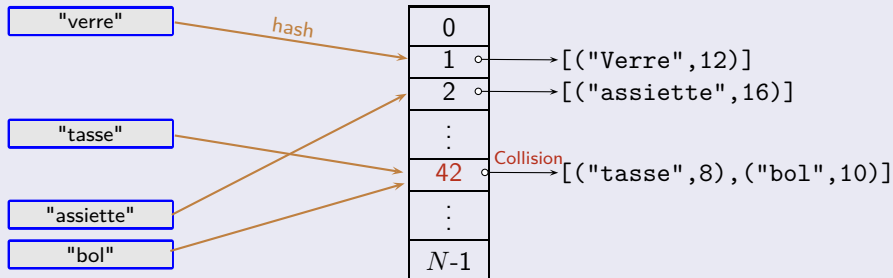
### Visualisation

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



### Visualisation d'une collision

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

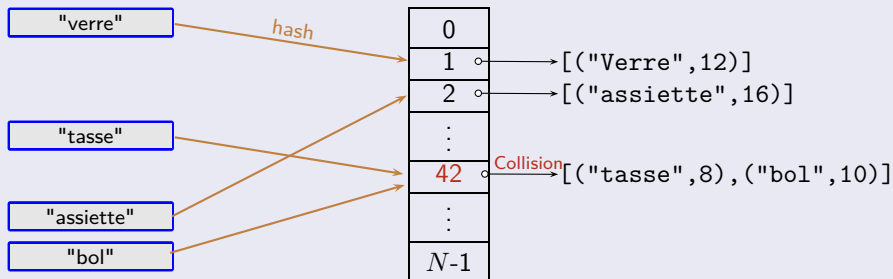


# C18 Programmation dynamique

## 2. Table de hachage

### 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 chaînes de caractères (une par ligne). On souhaite écrire une fonction qui renvoie la première chaîne 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 chaîne n'apparaît plus d'une fois alors le programme renvoie la chaîne vide.

- ➊ Proposer une version utilisant une liste `deja_vu` dans laquelle on enregistre au fur et à mesure du parcours les chaînes de caractères déjà rencontrées.
- ➋ 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 :

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

### Exemple

- 1 Ecrire une fonction récursive qui prend en argument un entier  $n$  et renvoie le  $n$ ème 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  $n = 5$

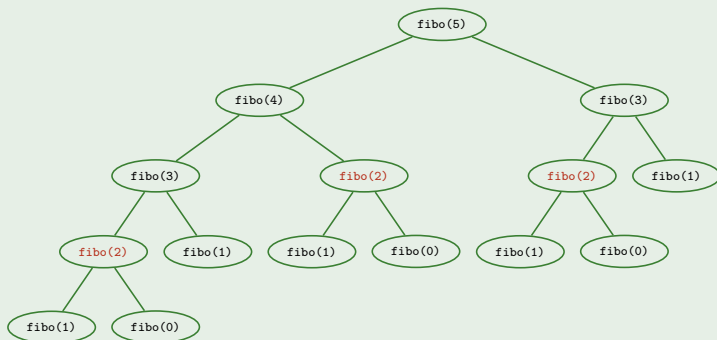
### Exemple

- 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

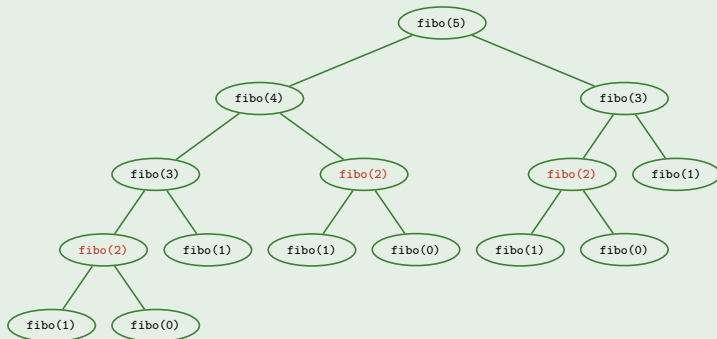
### Correction question 1

```
1 def fibonacci(n):  
2     assert n>=0  
3     if n<2:  
4         return n  
5     return fibonacci(n-1)+fibonacci(n-2)  
6  
7 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 `fibonacci(2)` (clé : 2, valeur : 1), on n'a plus besoin de la recalculer lors des futurs appels.

### Fibonnaci avec mémoïsation

```
1  # Le dictionnaire pour mémoïser
2  memo = {}
3
4  def fibonacci(n):
5      assert n >= 0
6      # Si la valeur se trouve dans le dictionnaire, elle a déjà été calculée
7      if n in memo:
8          return memo[n]
9      # Sinon on calcule et on enregistre dans le dictionnaire
10     if n < 2:
11         memo[n] = n
12         return n
13     memo[n] = fibonacci(n-1) + fibonacci(n-2)
14     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

```
1  from functools import lru_cache
2
3  @lru_cache
4  def fibonacci(n):
5      assert n>=0
6      if n<2:
7          return n
8      return fibonacci(n-1)+fibonacci(n-2)
```

# C18 Programmation dynamique

## 4. Programmation dynamique : exemple introductif

### 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

# C18 Programmation dynamique

## 4. Programmation dynamique : exemple introductif

### 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  $18 + 12 = 30$

# C18 Programmation dynamique

## 4. Programmation dynamique : exemple introductif

### 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.



# C18 Programmation dynamique

## 4. Programmation dynamique : exemple introductif

### 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  $i$ .

# C18 Programmation dynamique

## 4. Programmation dynamique : exemple introductif

### 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  $v_1$ .

# C18 Programmation dynamique

## 4. Programmation dynamique : exemple introductif

### 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  $(v_i)_{0 \leq i \leq N}$ .

# C18 Programmation dynamique

## 4. Programmation dynamique : exemple introductif

### 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.

# C18 Programmation dynamique

## 4. Programmation dynamique : exemple introductif

### 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 inférieures  $n$ , 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 inférieures  $n$ , 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 :

```
1 def valeur_max(taille, prix):
2     if taille < 2:
3         return prix[taille]
4     else:
5         return max(valeur_max(k, prix) + prix[taille - k] for k in range(taille))
6
7 print(valeur_max(12, [0, 2, 7, 11, 15, 17, 18, 23, 24, 27, 29, 33, 38]))
```



# C18 Programmation dynamique

## 4. Programmation dynamique : exemple introductif

### Résolution

- 1  $v_0 = 0$  et  $v_1 = 1$
- 2 En supposant qu'on connaisse les valeurs maximales de découpe pour les tailles inférieures  $n$ , 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 :

```
1 def valeur_max(taille, prix):
2     if taille < 2:
3         return prix[taille]
4     else:
5         return max(valeur_max(k, prix) + prix[taille - k] for k in range(taille))
6
7 print(valeur_max(12, [0, 2, 7, 11, 15, 17, 18, 23, 24, 27, 29, 33, 38]))
```

Le programme affiche 32.

# C18 Programmation dynamique

## 4. Programmation dynamique : exemple introductif

### Version avec mémoïsation

```
1  vmax = {}
2  def valeur_max(taille, prix):
3      if taille in vmax:
4          return vmax[taille]
5      if taille < 2:
6          vmax[taille] = prix[taille]
7          return prix[taille]
8      else:
9          vmax[taille] = max(valeur_max(k, prix) + prix[taille - k] for k in range(1, taille))
10         return vmax[taille]
```

# C18 Programmation dynamique

## 4. Programmation dynamique : exemple introductif

### 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.

# C18 Programmation dynamique

## 4. Programmation dynamique : exemple introductif

### Calcul de bas en haut (*bottom up*)

La mémorisation 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

$$v_n = \max \{v_k + p_{n-k}, 0 \leq k \leq n - 1\}$$

# C18 Programmation dynamique

## 4. Programmation dynamique : exemple introductif

### Calcul de bas en haut (*bottom up*)

La mémorisation 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* :

# C18 Programmation dynamique

## 4. Programmation dynamique : exemple introductif

### Calcul de bas en haut (*bottom up*)

La mémorisation 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* :

```
1 def valeur_max(taille, prix):
2     v = {0:0,1:prix[1]}
3     for i in range(2,taille+1):
4         v[i] = max(v[k]+prix[i-k] for k in range(i))
5     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  $(v_k)_{0 \leq k \leq N}$  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  $(v_k)_{0 \leq k \leq N}$  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  $(v_k)_{0 \leq k \leq N}$  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.

# C18 Programmation dynamique

## 4. Programmation dynamique : exemple introductif

### 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  $(v_k)_{0 \leq k \leq N}$  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 mémorisation 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 mémorisation 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 mémorisation 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.



# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Position du problème

On considère deux chaînes 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 chaîne  $w$  telle que :

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Position du problème

On considère deux chaînes 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 chaîne  $w$  telle que :

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

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Position du problème

On considère deux chaînes 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 chaîne  $w$  telle que :

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

### Position du problème

On considère deux chaînes 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 chaîne  $w$  telle que :

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

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Position du problème

On considère deux chaînes 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 chaîne  $w$  telle que :

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

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

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Position du problème

On considère deux chaînes 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 chaîne  $w$  telle que :

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

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

- PROGRAMMATION

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Position du problème

On considère deux chaînes 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 chaîne  $w$  telle que :

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

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

- PROGRAMMATION
- DYNAMIQUE

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Position du problème

On considère deux chaînes 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 chaîne  $w$  telle que :

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

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

- PROGRAMMATION
- DYNAMIQUE

Donc ici, la longueur de la plssc est 3.



# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### 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 chaîne 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  $v_j$ .

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### 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 chaîne 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  $\text{lplssc}(u_{i-1}, v_{j-1})$  ?

### 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 chaîne 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  $\text{plssc}(u_{i-1}, v_j)$

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### 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 chaîne 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 chaînes vides notés "") :

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### 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 chaîne 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 chaînes vides notés "") :

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### 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 chaîne 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  $\text{plssc}(u_{i-1}, v_j)$   
$$\text{lplssc}(u_i, v_j) = \max(\text{lplssc}(u_i, v_{j-1}), \text{lplssc}(u_{i-1}, v_j))$$
- Déterminer les cas de base (ceux où  $u$  et  $v$  sont des chaînes vides notés "") :

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### 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 chaîne 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  $\text{plssc}(u_{i-1}, v_j)$   
$$\text{lplssc}(u_i, v_j) = \max(\text{lplssc}(u_i, v_{j-1}), \text{lplssc}(u_{i-1}, v_j))$$
- Déterminer les cas de base (ceux où  $u$  et  $v$  sont des chaînes vides notés "") :  
$$\text{lplssc}(u_i, "") = 0$$
$$\text{lplssc}("", v_j) = 0$$

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Programme Python pour plssc

```
1  from functools import lru_cache
2
3  @lru_cache
4  def lplssc(mot1,mot2):
5      if len(mot2)==0 or len(mot1)==0:
6          return 0
7      if mot1[-1]==mot2[-1]:
8          return 1 + lplssc(mot1[:-1],mot2[:-1])
9      return max(lplssc(mot1[:-1],mot2),lplssc(mot1,mot2[:-1]))
```



# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### 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

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### 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

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### 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

```
1 @lru_cache
2 def plssc(mot1,mot2):
3     if len(mot2)==0 or len(mot1)==0:
4         return ""
5     if mot1[-1]==mot2[-1]:
6         return plssc(mot1[:-1],mot2[:-1])+mot1[-1]
7     if len(plssc(mot1[:-1],mot2))>len(plssc(mot1,mot2[:-1])):
8         return plssc(mot1[:-1],mot2)
9     else:
10        return plssc(mot1,mot2[:-1])
```

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

On peut construire la matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€							
I							
M							
P							
E							
R							
I							
A							
L							

Cette matrice permet de reconstruire *une* solution, c'est à dire de déterminer une plus longue sous séquence commune.

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	1	1	1	1	1
P	0	0	1	2	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	3	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	1	1	1	1	1
P	0	0	1	2	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	3	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

On peut utiliser ce résultat pour construire la PLSSC :

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	1	1	1	1	1
P	0	0	1	2	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	3	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

On peut utiliser ce résultat pour construire la PLSSC :

- on part de la dernière case en bas et à droite

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	1	1	1	1	1
P	0	0	1	2	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	3	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

On peut utiliser ce résultat pour construire la PLSSC :

- on part de la dernière case en bas et à droite
- si les lettres sur la ligne et la colonne sont identiques on ajoute à la PLSSC et on remonte en diagonale



# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	1	1	1	1	1
P	0	0	1	2	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	3	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

On peut utiliser ce résultat pour construire la PLSSC :

- on part de la dernière case en bas et à droite
- si les lettres sur la ligne et la colonne sont identiques on ajoute à la PLSSC et on remonte en diagonale
- sinon on va à gauche ou en haut suivant la case qui a plus grande valeur

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	1	1	1	1	1
P	0	0	1	2	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	3	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

On peut utiliser ce résultat pour construire la PLSSC :

- on part de la dernière case en bas et à droite
- si les lettres sur la ligne et la colonne sont identiques on ajoute à la PLSSC et on remonte en diagonale
- sinon on va à gauche ou en haut suivant la case qui a plus grande valeur

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	1	1	1	1	1
P	0	0	1	2	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	3	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

On peut utiliser ce résultat pour construire la PLSSC :

- on part de la dernière case en bas et à droite
- si les lettres sur la ligne et la colonne sont identiques on ajoute à la PLSSC et on remonte en diagonale
- sinon on va à gauche ou en haut suivant la case qui a plus grande valeur

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	1	1	1	1	1
P	0	0	1	2	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	3	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

On peut utiliser ce résultat pour construire la PLSSC :

- on part de la dernière case en bas et à droite
- si les lettres sur la ligne et la colonne sont identiques on ajoute à la PLSSC et on remonte en diagonale
- sinon on va à gauche ou en haut suivant la case qui a plus grande valeur

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	1	1	1	1	1
P	0	0	1	2	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	3	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

On peut utiliser ce résultat pour construire la PLSSC :

- on part de la dernière case en bas et à droite
- si les lettres sur la ligne et la colonne sont identiques on ajoute à la PLSSC et on remonte en diagonale
- sinon on va à gauche ou en haut suivant la case qui a plus grande valeur

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	1	1	1	1	1
P	0	0	1	2	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	<u>3</u>	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

On peut utiliser ce résultat pour construire la PLSSC :

- on part de la dernière case en bas et à droite
- si les lettres sur la ligne et la colonne sont identiques on ajoute à la PLSSC et on remonte en diagonale
- sinon on va à gauche ou en haut suivant la case qui a plus grande valeur

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	1	1	1	1	1
P	0	0	1	2	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	<u>3</u>	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

On peut utiliser ce résultat pour construire la PLSSC :

- on part de la dernière case en bas et à droite
- si les lettres sur la ligne et la colonne sont identiques on ajoute à la PLSSC et on remonte en diagonale
- sinon on va à gauche ou en haut suivant la case qui a plus grande valeur

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	1	1	1	1	1
P	0	0	1	2	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	<u>3</u>	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

On peut utiliser ce résultat pour construire la PLSSC :

- on part de la dernière case en bas et à droite
- si les lettres sur la ligne et la colonne sont identiques on ajoute à la PLSSC et on remonte en diagonale
- sinon on va à gauche ou en haut suivant la case qui a plus grande valeur



# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	1	1	1	1	1
P	0	0	1	2	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	<u>3</u>	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

On peut utiliser ce résultat pour construire la PLSSC :

- on part de la dernière case en bas et à droite
- si les lettres sur la ligne et la colonne sont identiques on ajoute à la PLSSC et on remonte en diagonale
- sinon on va à gauche ou en haut suivant la case qui a plus grande valeur

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	1	1	1	1	1
P	0	0	1	<u>2</u>	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	<u>3</u>	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

On peut utiliser ce résultat pour construire la PLSSC :

- on part de la dernière case en bas et à droite
- si les lettres sur la ligne et la colonne sont identiques on ajoute à la PLSSC et on remonte en diagonale
- sinon on va à gauche ou en haut suivant la case qui a plus grande valeur

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	1	1	1	1	1
P	0	0	1	<u>2</u>	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	<u>3</u>	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

On peut utiliser ce résultat pour construire la PLSSC :

- on part de la dernière case en bas et à droite
- si les lettres sur la ligne et la colonne sont identiques on ajoute à la PLSSC et on remonte en diagonale
- sinon on va à gauche ou en haut suivant la case qui a plus grande valeur

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	<u>1</u>	1	1	1	1
P	0	0	1	<u>2</u>	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	<u>3</u>	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

On peut utiliser ce résultat pour construire la PLSSC :

- on part de la dernière case en bas et à droite
- si les lettres sur la ligne et la colonne sont identiques on ajoute à la PLSSC et on remonte en diagonale
- sinon on va à gauche ou en haut suivant la case qui a plus grande valeur

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	<u>1</u>	1	1	1	1
P	0	0	1	<u>2</u>	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	<u>3</u>	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

On peut utiliser ce résultat pour construire la PLSSC :

- on part de la dernière case en bas et à droite
- si les lettres sur la ligne et la colonne sont identiques on ajoute à la PLSSC et on remonte en diagonale
- sinon on va à gauche ou en haut suivant la case qui a plus grande valeur

# C18 Programmation dynamique

## 6. Exemple résolu : plus longue sous séquence commune

### Reconstruction de la solution

La matrice obtenue pour les mots : IMPERIAL et EMPIRE est :

	€	E	M	P	I	R	E
€	0	0	0	0	0	0	0
I	0	0	0	0	1	1	1
M	0	0	<u>1</u>	1	1	1	1
P	0	0	1	<u>2</u>	2	2	2
E	0	1	1	2	2	2	3
R	0	1	1	2	2	3	3
I	0	1	1	2	<u>3</u>	3	3
A	0	1	1	2	3	3	3
L	0	1	1	2	3	3	3

On peut utiliser ce résultat pour construire la PLSSC :

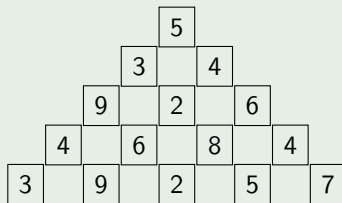
- on part de la dernière case en bas et à droite
- si les lettres sur la ligne et la colonne sont identiques on ajoute à la PLSSC et on remonte en diagonale
- sinon on va à gauche ou en haut suivant la case qui a plus grande valeur

# C18 Programmation dynamique

## 7. Exemple résolu : chemin de somme maximale dans une pyramide

### Position du problème

Rechercher le chemin de somme maximale partant du haut d'une pyramide d'entiers à sa base :

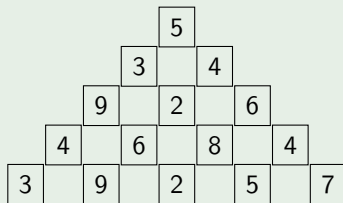


# C18 Programmation dynamique

## 7. Exemple résolu : chemin de somme maximale dans une pyramide

### Position du problème

Rechercher le chemin de somme maximale partant du haut d'une pyramide d'entiers à sa base :



- 1 déterminer cette somme et un chemin possible sur l'exemple ci-dessus.

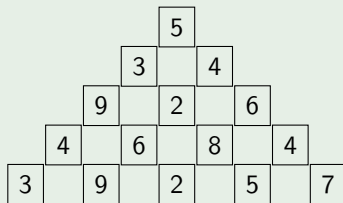


# C18 Programmation dynamique

## 7. Exemple résolu : chemin de somme maximale dans une pyramide

### Position du problème

Rechercher le chemin de somme maximale partant du haut d'une pyramide d'entiers à sa base :



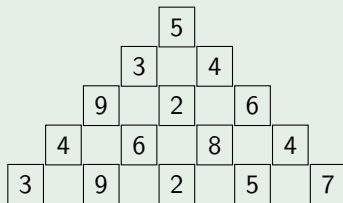
- 1 déterminer cette somme et un chemin possible sur l'exemple ci-dessus.
- 2 que penser d'une solution qui testerait toutes les possibilités ?

# C18 Programmation dynamique

## 7. Exemple résolu : chemin de somme maximale dans une pyramide

### Position du problème

Rechercher le chemin de somme maximale partant du haut d'une pyramide d'entiers à sa base :



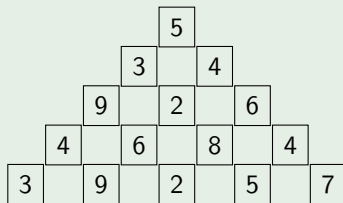
- 1 déterminer cette somme et un chemin possible sur l'exemple ci-dessus.
- 2 que penser d'une solution qui testerait toutes les possibilités ?

# C18 Programmation dynamique

## 7. Exemple résolu : chemin de somme maximale dans une pyramide

### Position du problème

Rechercher le chemin de somme maximale partant du haut d'une pyramide d'entiers à sa base :



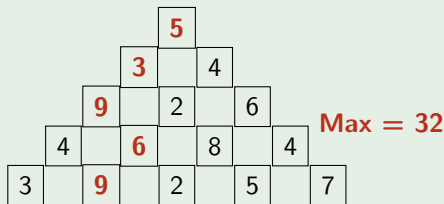
- 1 déterminer cette somme et un chemin possible sur l'exemple ci-dessus.
- 2 que penser d'une solution qui testerait toutes les possibilités ?

# C18 Programmation dynamique

## 7. Exemple résolu : chemin de somme maximale dans une pyramide

### Position du problème

Rechercher le chemin de somme maximale partant du haut d'une pyramide d'entiers à sa base :



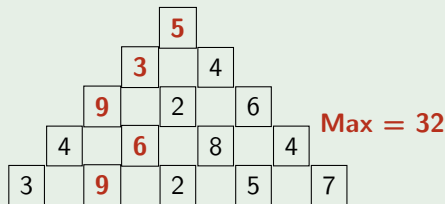
- 1 déterminer cette somme et un chemin possible sur l'exemple ci-dessus.
- 2 que penser d'une solution qui testerait toutes les possibilités ?

# C18 Programmation dynamique

## 7. Exemple résolu : chemin de somme maximale dans une pyramide

### Position du problème

Rechercher le chemin de somme maximale partant du haut d'une pyramide d'entiers à sa base :



- 1 déterminer cette somme et un chemin possible sur l'exemple ci-dessus.
- 2 que penser d'une solution qui testerait toutes les possibilités ?
- 3 obtient-on la solution optimale en choisissant l'entier maximal à chaque étape ?
- 4 vérifier qu'on retrouve les caractéristiques d'un problème soluble par programmation dynamique.

# C18 Programmation dynamique

## 7. Exemple résolu : chemin de somme maximale dans une pyramide

### Correction

- 1 Pour tester chaque possibilité on doit énumérer tous les chemins possibles depuis le sommet vers un point de la base, en notant  $n$  le nombre de niveau, on a deux choix à chaque niveau (excepté le premier), le nombre de chemins est donc  $2^{n-1}$ . La complexité est donc en  $\mathcal{O}(2^n)$  c'est à dire *exponentielle*. Cette méthode n'est donc pas utilisable pour des valeurs de  $n$  importantes.

# C18 Programmation dynamique

## 7. Exemple résolu : chemin de somme maximale dans une pyramide

### Correction

- ① Pour tester chaque possibilité on doit énumérer tous les chemins possibles depuis le sommet vers un point de la base, en notant  $n$  le nombre de niveau, on a deux choix à chaque niveau (excepté le premier), le nombre de chemins est donc  $2^{n-1}$ . La complexité est donc en  $\mathcal{O}(2^n)$  c'est à dire *exponentielle*. Cette méthode n'est donc pas utilisable pour des valeurs de  $n$  importantes.
- ② Sur l'exemple ci-dessus on constate que prendre le maximum à chaque étape (c'est à dire adopter un **algorithme glouton**) ne conduit pas à la solution optimale.

# C18 Programmation dynamique

## 7. Exemple résolu : chemin de somme maximale dans une pyramide

### Correction

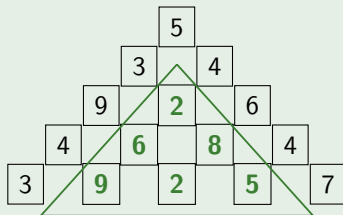
- ❶ Pour tester chaque possibilité on doit énumérer tous les chemins possibles depuis le sommet vers un point de la base, en notant  $n$  le nombre de niveau, on a deux choix à chaque niveau (excepté le premier), le nombre de chemins est donc  $2^{n-1}$ . La complexité est donc en  $\mathcal{O}(2^n)$  c'est à dire *exponentielle*. Cette méthode n'est donc pas utilisable pour des valeurs de  $n$  importantes.
- ❷ Sur l'exemple ci-dessus on constate que prendre le maximum à chaque étape (c'est à dire adopter un *algorithme glouton*) ne conduit pas à la solution optimale.
- ❸ C'est bien un problème d'*optimisation* et résoudre un problème pour une pyramide de  $n$  niveau et de sommets  $S$ , c'est :
  - résoudre le même problème sur les *deux* pyramides de  $n - 1$  niveaux et de sommets respectifs  $S_g$  et  $S_d$  (fils gauche et droit de  $S$ ). Puis prendre le maximum des deux résultats, on trouve la propriété de *sous structure optimale*.
  - Les sous problèmes se *chevauchent* car on retrouve bien la même pyramide centrale en traitant celle de sommet  $S_g$  ou celle de sommet  $S_d$  comme illustré ci-dessous.



# C18 Programmation dynamique

## 7. Exemple résolu : chemin de somme maximale dans une pyramide

### Position du problème



Ce sous problème apparaît dans la résolution du sous problème la pyramide droite et celui de la pyramide gauche.

# C18 Programmation dynamique

## 7. Exemple résolu : chemin de somme maximale dans une pyramide

### Éléments de résolution

- On représente la pyramide de façon matricielle par une liste de liste. Par exemple la pyramide ci-dessous est la liste :  
[[5], [3, 4], [9, 2, 6], [4, 6, 8, 4], [3, 9, 2, 5, 7]]. On note  $p_{i,j}$  le coefficient de cette matrice situé en ligne  $i$  et en colonne  $j$ .

## C18 Programmation dynamique

### 7. Exemple résolu : chemin de somme maximale dans une pyramide

#### Éléments de résolution

- On représente la pyramide de façon matricielle par une liste de liste. Par exemple la pyramide ci-dessous est la liste :  
[[5], [3, 4], [9, 2, 6], [4, 6, 8, 4], [3, 9, 2, 5, 7]]. On note  $p_{i,j}$  le coefficient de cette matrice situé en ligne  $i$  et en colonne  $j$ .
- Toute sous pyramide est alors identifiée par les indices de ligne et de colonne de son sommet dans cette liste. Par exemple le sommet de la pyramide initial est (0,0).

# C18 Programmation dynamique

## 7. Exemple résolu : chemin de somme maximale dans une pyramide

### Éléments de résolution

- On représente la pyramide de façon matricielle par une liste de liste. Par exemple la pyramide ci-dessous est la liste :  
[[5], [3, 4], [9, 2, 6], [4, 6, 8, 4], [3, 9, 2, 5, 7]]. On note  $p_{i,j}$  le coefficient de cette matrice situé en ligne  $i$  et en colonne  $j$ .
- Toute sous pyramide est alors identifiée par les indices de ligne et de colonne de son sommet dans cette liste. Par exemple le sommet de la pyramide initial est (0,0).
- D'après l'étude qui précède en notant  $M_{i,j}$  la somme d'un chemin maximal pour la pyramide de sommet  $(i,j)$ , on obtient immédiatement l'équation suivante :  $M_{i,j} = p_{i,j} + \max \{M_{i+1,j} ; M_{i+1,j+1}\}$ .

## C18 Programmation dynamique

### 7. Exemple résolu : chemin de somme maximale dans une pyramide

#### Éléments de résolution

- On représente la pyramide de façon matricielle par une liste de liste. Par exemple la pyramide ci-dessous est la liste :  
[[5], [3, 4], [9, 2, 6], [4, 6, 8, 4], [3, 9, 2, 5, 7]]. On note  $p_{i,j}$  le coefficient de cette matrice situé en ligne  $i$  et en colonne  $j$ .
- Toute sous pyramide est alors identifiée par les indices de ligne et de colonne de son sommet dans cette liste. Par exemple le sommet de la pyramide initial est (0,0).
- D'après l'étude qui précède en notant  $M_{i,j}$  la somme d'un chemin maximal pour la pyramide de sommet  $(i,j)$ , on obtient immédiatement l'équation suivante :  $M_{i,j} = p_{i,j} + \max \{M_{i+1,j} ; M_{i+1,j+1}\}$ .
- Le cas de base est lorsque  $i = n - 1$  (on atteint la base de la pyramide).

# C18 Programmation dynamique

## 7. Exemple résolu : chemin de somme maximale dans une pyramide

### Éléments de résolution

- On représente la pyramide de façon matricielle par une liste de liste. Par exemple la pyramide ci-dessous est la liste :  
[[5], [3, 4], [9, 2, 6], [4, 6, 8, 4], [3, 9, 2, 5, 7]]. On note  $p_{i,j}$  le coefficient de cette matrice situé en ligne  $i$  et en colonne  $j$ .
- Toute sous pyramide est alors identifiée par les indices de ligne et de colonne de son sommet dans cette liste. Par exemple le sommet de la pyramide initial est (0,0).
- D'après l'étude qui précède en notant  $M_{i,j}$  la somme d'un chemin maximal pour la pyramide de sommet  $(i,j)$ , on obtient immédiatement l'équation suivante :  $M_{i,j} = p_{i,j} + \max \{M_{i+1,j} ; M_{i+1,j+1}\}$ .
- Le cas de base est lorsque  $i = n - 1$  (on atteint la base de la pyramide).
- Voir l'exercice de TP pour une résolution complète.

## 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 :

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

## Résolution

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

```
1  def rm(somme,k):
2      if (somme,k) in memo:
3          return memo[(somme,k)]
4      if k>=len(systeme):
5          memo[(somme,k)]=inf
6          return inf
7      if systeme[k]>somme:
8          memo[(somme,k)] = rm(somme,k+1)
9          return memo[(somme,k)]
10     memo[(somme,k)] = min(1+rm(somme-systeme[k],k),rm(somme,k+1))
11     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	$\infty$
1					$\infty$
2					$\infty$
3					$\infty$
4					$\infty$
5					$\infty$
6					$\infty$
7					$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$

## 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	$\infty$
1	1	1	1	1	$\infty$
2					$\infty$
3					$\infty$
4					$\infty$
5					$\infty$
6					$\infty$
7					$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$



## 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	$\infty$
1	1	1	1	1	$\infty$
2	2	2	2	2	$\infty$
3					$\infty$
4					$\infty$
5					$\infty$
6					$\infty$
7					$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$

## 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	$\infty$
1	1	1	1	1	$\infty$
2	2	2	2	2	$\infty$
3	1	1	1	3	$\infty$
4					$\infty$
5					$\infty$
6					$\infty$
7					$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$

## 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	$\infty$
1	1	1	1	1	$\infty$
2	2	2	2	2	$\infty$
3	1	1	1	3	$\infty$
4	1	1	2	4	$\infty$
5					$\infty$
6					$\infty$
7					$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$

## 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	$\infty$
1	1	1	1	1	$\infty$
2	2	2	2	2	$\infty$
3	1	1	1	3	$\infty$
4	1	1	2	4	$\infty$
5	1	2	3	5	$\infty$
6					$\infty$
7					$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$

## 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	$\infty$
1	1	1	1	1	$\infty$
2	2	2	2	2	$\infty$
3	1	1	1	3	$\infty$
4	1	1	2	4	$\infty$
5	1	2	3	5	$\infty$
6	2	2	2	6	$\infty$
7					$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$

## 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	$\infty$
1	1	1	1	1	$\infty$
2	2	2	2	2	$\infty$
3	1	1	1	3	$\infty$
4	1	1	2	4	$\infty$
5	1	2	3	5	$\infty$
6	2	2	2	6	$\infty$
7	2	2	3	7	$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$

## 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	$\infty$
1	1	1	1	1	$\infty$
2	2	2	2	2	$\infty$
3	1	1	1	3	$\infty$
4	1	1	2	4	$\infty$
5	1	2	3	5	$\infty$
6	2	2	2	6	$\infty$
7	2	2	3	7	$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$

## 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	$\infty$
1	1	1	1	1	$\infty$
2	2	2	2	2	$\infty$
3	1	1	1	3	$\infty$
4	1	1	2	4	$\infty$
5	1	2	3	5	$\infty$
6	2	2	2	6	$\infty$
7	2	2	3	7	$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$



## 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	$\infty$
1	1	1	1	1	$\infty$
2	2	2	2	2	$\infty$
3	1	1	1	3	$\infty$
4	1	1	2	4	$\infty$
5	1	2	3	5	$\infty$
6	2	2	2	6	$\infty$
7	2	2	3	7	$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$

## 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	$\infty$
1	1	1	1	1	$\infty$
2	2	2	2	2	$\infty$
3	1	1	1	3	$\infty$
4	1	1	2	4	$\infty$
5	1	2	3	5	$\infty$
6	2	2	2	6	$\infty$
7	2	2	3	7	$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$

## 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	$\infty$
1	1	1	1	1	$\infty$
2	2	2	2	2	$\infty$
3	1	1	1	3	$\infty$
4	1	1	2	4	$\infty$
5	1	2	3	5	$\infty$
6	2	2	2	6	$\infty$
7	2	2	3	7	$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$

## Résolution

③ Cette méthode correspond au programme suivant :

```
1  def pieces(memo,somme):
2      pieces = []
3      while somme!=0:
4          min = inf
5          piece = None
6          for i in range(len(systeme)+1,-1,-1):
7              if (somme,i) in memo and memo[(somme,i)]<min:
8                  min = memo[(somme,i)]
9                  piece = systeme[i]
10         somme = somme-piece
11         pieces.append(piece)
12     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	$\infty$
1					$\infty$
2					$\infty$
3					$\infty$
4					$\infty$
5					$\infty$
6					$\infty$
7	✓				$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$

## 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	$\infty$
1					$\infty$
2	✓				$\infty$
3					$\infty$
4					$\infty$
5					$\infty$
6					$\infty$
7	✓	✓			$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$

## 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	$\infty$
1					$\infty$
2	✓	✓			$\infty$
3		✓			$\infty$
4					$\infty$
5					$\infty$
6					$\infty$
7	✓	✓	✓		$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$

## 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	$\infty$
1					$\infty$
2	✓	✓	✓		$\infty$
3		✓	✓		$\infty$
4			✓		$\infty$
5					$\infty$
6					$\infty$
7	✓	✓	✓	✓	$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$



## 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	$\infty$
1					$\infty$
2	✓	✓	✓	✓	$\infty$
3		✓	✓	✓	$\infty$
4			✓	✓	$\infty$
5				✓	$\infty$
6					$\infty$
7	✓	✓	✓	✓	$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$

## Résolution

Remarque : sur cet exemple, les appels récurifs 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	$\infty$
1					$\infty$
2	✓	✓	✓	✓	$\infty$
3		✓	✓	✓	$\infty$
4			✓	✓	$\infty$
5				✓	$\infty$
6				✓	$\infty$
7	✓	✓	✓	✓	$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$

## 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	$\infty$
1				✓	$\infty$
2	✓	✓	✓	✓	$\infty$
3		✓	✓	✓	$\infty$
4			✓	✓	$\infty$
5				✓	$\infty$
6				✓	$\infty$
7	✓	✓	✓	✓	$\infty$

$$p_0 = 5, p_1 = 4, p_2 = 3, p_3 = 1$$