INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

CM6 – Algorithmique

Décomposition fonctionnelle, complexité

UNIVERSITÉ DE LYON

Glossaire du jour

- Décomposition en sous-problèmes
- Décomposition fonctionnelle
- Test unitaire
- Complexité

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

1. Décomposition en sous-problèmes

Décomposition fonctionnelle

UNIVERSITÉ DE LYON

Conception d'un algorithme

Jusqu'à présent…

- Problèmes simples à résoudre
  - «Trouver le premier élément de la liste supérieur à 10»
- Écrire directement l'algorithme complet

Maintenant…

- Des problèmes plus complexes
  - Exemple : Le jeu de la vie
- Impossible d'écrire directement l'algorithme/le code **complet**
- Par où commencer ?

Jeu de la vie – écrivez l'algorithme

Créer un monde aléatoire

Pour n fois

    Pour chaque cellule

        Calculez le nombre de voisins vivants de cette cellule.

        Modifier l'état de la cellule en fonction du nombre de voisins.

    Afficher le monde dans l'état suivant

Exemple/Démonstration

Jeu de la vie – écrivez l'algorithme

Créer un monde aléatoire

Sous-tâches à résoudre plus tard

Pour n fois

Pour chaque cellule

Calculez le nombre de voisins vivants de cette cellule.

Modifier l'état de la cellule en fonction du nombre de voisins.

Afficher le monde dans l'état suivant

Exemple/Démonstration

Conception d'un algorithme

- Une fois qu'on a un premier algorithme et une division en tâches
- **Pour chaque tâche**, spécifier clairement
  - Ce qu'elle fait/son but
  - Quelles sont les entrées/sorties ?
  - Comment la résoudre

    écrire son algorithme et éventuellement le décomposer (processus récursif !)

Important

Jeu de la vie – écrivez l'algorithme

Créer un monde aléatoire

Pour n fois

Pour chaque cellule

Calculez le nombre de voisins vivants de cette cellule.

Modifier l'état de la cellule en fonction du nombre de voisins.

Afficher le monde dans l'état suivant

Exemple/Démonstration

Jeu de la vie – écrivez l'algorithme

Créer un monde aléatoire

Pour n fois

Pour chaque cellule

Calculez le nombre de voisins viva

Modifier l'état de la cellule en fon

Afficher le monde dans l'état suivant

Créer un monde aléatoire

Crée un monde d'une taille donnée, et place aléatoirement des cellules en vie selon une probabilité.

Entrée *: taille du monde (int), proba (float)*
Sortie *: monde (liste 2D)*

Créer un monde mort

Pour chaque cellule :

Placer une cellule vivante avec proba

Exemple/Démonstration

Jeu de la vie – écrivez l'algorithme

Créer un monde aléatoire

Pour n fois

Pour chaque cellule

Créer un monde mort

Entrée *: taille du monde (int)*
Sortie *: monde (liste 2D) rempli de 0*

Algorithme connu

ins viva

en fon

suivant

Créer un monde aléatoire

Crée un monde d'une taille donnée, et place aléatoirement des cellules en vie selon une probabilité.

Entrée *: taille du monde (int), proba (float)*
Sortie *: monde (liste 2D)*

Créer un monde mort

Pour chaque cellule :

Placer une cellule vivante avec proba

Exemple/Démonstration

Décomposition fonctionnelle

- Comment passer de cette décomposition à du code Python ?
- Une tâche = une fonction
  - Facilite la répartition du travail au sein d'un groupe.
  - Facilite le débogage
  - Code plus facile à lire et à comprendre
- Pour chaque tâche/fonction
  - But, entrées et sorties : signature de la fonction
  - Comment la résoudre : pseudo-code
  - Code et **test**

Définition / Important

Résumé : comment faire une décomposition fonctionnelle ?

(À partir d'une tâche initiale)

1) Écrire l'algorithme principal et identifier les sous-tâches.
2) Pour chacune des sous-tâches
   - Les spécifier avec leurs entrées/sorties => signature
   - Faire *leur décomposition fonctionnelle*
3) Écrire le code
4) **Tester**

Définition / Important

Résultat attendu d'une décomposition fonctionnelle

- L'algorithme principal où les sous-tâches sont identifiées
- Pour chaque sous-tâche :
  - Sa spécification (description, entrées et sorties)
  - Son algorithme, s'il n'est pas trivial (si c'est le cas, dites-le)

Définition / Important

Découpage en sous-tâches

Ex : calculer σ(n) = la somme des diviseurs d'un entier n

Exemple/Démonstration

Découpage en sous-tâches

Ex : calculer σ(n) = la somme des diviseurs d'un entier n

1) Écrire l'algorithme / identifier les sous-tâches

- Trouver D l'ensemble de tous les diviseurs de n
- Calculer la somme des éléments de D

Exemple/Démonstration

Découpage en sous-tâches

Ex : calculer σ(n) = la somme des diviseurs d'un entier n

2b) Décomposition fonctionnelle des sous-tâches

- Trouver D l'ensemble de tous les diviseurs de n
- Calculer la somme des éléments de D

Calculer la liste des diviseurs d'un entier n

Entrée *: un entier n*
Sortie *: une liste d'entiers qui divisent n*

Calculer la somme d'une liste d'entiers

Entrée *: une liste d'entiers*
Sortie *: un entier égal à la somme de la liste*

Découpage en sous-tâches

Ex : calculer σ(n) = la somme des diviseurs d'un entier n

2b) Décomposition fonctionnelle des sous-tâches

- Trouver D l'ensemble de tous les diviseurs de n
- Calculer la somme des éléments de D

Calculer la liste des diviseurs d'un entier n

***Entrée** : un entier n*
***Sortie** : une liste d'entiers qui divisent n*

Calculer la somme d'une liste d'entiers

***Entrée** : une liste d'entiers*
***Sortie** : un entier égal à la somme de la liste*

Algorithme connu

diviseurs = []
∀ entier i ∈ [0, n[:
  Si i divise n
  Ajouter i à diviseurs

Exemple/Démonstration

Découpage en sous-tâches

Ex : calculer σ(n) = la somme des diviseurs d'un entier n

2b) Décomposition fonctionnelle des sous-tâches

- Trouver D l'ensemble de tous les diviseurs de n
- Calculer la somme des éléments de D

Calculer la liste des diviseurs d'un entier n

***Entrée** : un entier n*
***Sortie** : une liste d'entiers qui divisent n*

```
diviseurs = []
∀ entier i ∈ [0, n[ :
    Si i divise n
        Ajouter i à diviseurs
```

Calculer la somme d'une liste d'entiers

***Entrée** : une liste d'entiers*
***Sortie** : un entier égal à la somme de la liste*

Algorithme connu

Tester si un entier est divisible par un autre

***Entrées** : deux entiers a et b*
***Résultat** : booléen « a divise b ? »*

Algorithme connu

Découpage en sous-tâches

Ex : calculer σ(n) = la somme des diviseurs d'un entier n

3) Implémenter les fonctions (docstring non affichée ici)

Calculer la somme d'une liste d'entiers

Entrée *: une liste d'entiers*
Sortie *: un entier égal à la somme de la liste*

```
def somme_liste(liste):
    somme = 0
    for e in liste:
        somme += e
    return somme
```

Tester si un entier est divisible par un autre

Entrées *: deux entiers a et b*
Résultat *: booléen « a divise b ? »*

```
def divise(a,b):
        res = True
    if b%a != 0 :
        res = False
    return res
```

Calculer la liste des diviseurs d'un entier n

Entrée *: un entier n*
Sortie *: une liste d'entiers qui divisent n*

```
def diviseurs(n):
    liste = []
    for i in range(1,n//2):
        if divise(i,n):
            liste.append(i)
    return liste
```


Exemple/Démonstration

Découpage en sous-tâches

Ex : calculer σ(n) = la somme des diviseurs d'un entier n

4) Testez !

Chaque sous-fonction DOIT être testée soigneusement AVANT d'écrire et de tester le niveau supérieur.

Exemple/Démonstration

Test et décomposition fonctionnelle

- Si vous êtes sûrs que chaque sous-tâche est **correctement** résolue, vous avez plus de chances que le programme complet soit correct.
- Vérification de **chaque** fonction (**test unitaire**)
  - Vérifiez que le comportement des fonctions est correct.
- Un test unitaire
  - Vérifie les résultats d'une seule fonction
  - Basé sur un ensemble de tests composé de cas de test (vu au S1)

Important

Comment tester les fonctions ?

- Implémentation des tests dans une fonction
  - Teste une fonction (somme_liste ici)
  - Renvoie True si tous les tests réussissent, False sinon, et affiche un message d'erreur en cas d'échec.
  - Permet de relancer tous les tests lors de modifications du code, ne la supprimez pas !

```
def test_somme_liste():
    res = True
    if somme_liste([1,2,3]) != 6 :
        print(f"Test faux: somme_liste([1,2,3]) != 6")
        res = False
    ...
    if somme_liste([1]) != 1 :
        print(f"Test faux: somme_liste([1]) != 1")
        res = False
    return res
```

On appelle la fonction avec un cas de test

Si le résultat n'est pas correct, on affiche un message d'alerte

et on retourne False

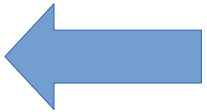
Découpage en sous-tâches

Ex : calculer σ(n) = la somme des diviseurs d'un entier n

3) Implémenter le programme principal

```
def sigma(n):
    liste_div = diviseurs(n)
    somme_div = somme_liste(liste_div)
    return somme_div

# programme principal
print(sigma(10))
```



```
def divise(a,b):
    ...
def diviseurs(n):
    ...
def somme_liste(liste):
    ...
```


Exemple/Démonstration

En résumé

- La décomposition fonctionnelle permet de
  - de rendre le code plus facile à comprendre...
  - et à lire : plusieurs fonctions simples plutôt qu'un gros bloc de code complexe
  - De distribuer le travail entre plusieurs personnes (c.f. projets)
  - **De s'assurer plus facilement que le code fonctionne/ trouver les bugs**
- Il nécessite
  - d'identifier les tâches principales et élémentaires d'un algorithme
  - d'identifier les entrées et sorties de ces tâches
  - (et surtout) de prendre le temps de réfléchir avant de se lancer tête baissée dans l'écriture du code :-)

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

2. Complexité

UNIVERSITÉ DE LYON

Complexité

- Vous **devez** réfléchir à la complexité (en temps) de vos algorithmes.
  - *Combien de temps faudra-t-il pour l'exécuter ?*
  - Mesuré en nombre d'opérations élémentaires
    - Affectations, opérations arithmétiques, comparaisons...
- **Cas le plus intéressant** : elle dépend de la taille de l'entrée
  - Nombre d'éléments d'une séquence (souvent)
- Mais elle dépend aussi des données de la séquence !
  - On général, on retient le **pire cas.**

Complexité

- Exemple dans les algorithmes de tri (ISN1) :
  - Vous avez compté le nombre de comparaisons dans le code
  - Ce nombre était différent pour chaque liste
- En général, on la calcule mathématiquement
  - La complexité est une fonction qui prend n (nombre d'éléments) comme entrée

$$T : \mathbb{N} \to \mathbb{N}$$

$$n \mapsto T(n)$$

Complexité

```python
liste = [...] #liste de taille n
s = 0
for i in range(len(liste)):
    s = s + liste[i]
avg = s/n
```

T(n) = n + 1
(une addition par élément + 1 multiplication)

```python
liste = [...] #liste de taille n
s = 0
for i in range(len(liste)):
    s = s + liste[i] * 2
```

*T(n) = 2*n*
(une addition et une multiplication par élément)

Complexité

```
liste = [...] #liste de taille n
s = 0
for i in range(len(liste)):
    s = s + liste[i]
avg = s/n
```

T(n) = n + 1
(une addition par élément + 1 multiplication)

```
liste = [...] #liste de taille n
s = 0
for i in range(len(liste)):
    s = s + liste[i] * 2
```

*T(n) = 2*n*
(une addition et une multiplication par élément)

- On regarde le comportement asymptotique.
  - Sinon, c'est trop difficile (voire impossible) à calculer.
  - Ça ne modifie pas fondamentalement les performances de l'algorithme.

Complexité

- On utilise la notation O (utilisée en mathématiques), lorsque n tend vers l'infini

```
liste = [...] #liste de taille n
s = 0
for i in range(len(liste)):
    s = s + liste[i]
avg = s/n
```

```
liste = [...] #liste de taille n
s = 0
for i in range(len(liste)):
    s = s + liste[i] * 2
```

Les deux algorithmes en $O(n)$ (linéaire)

- Complexités habituelles et leurs propriétés/utilisations

Constante	$O(1)$	opérations de base
logarithmique	$O(\log_2(n))$	Rapide, recherche dichotomique rapide
Linéaire	$O(n)$	rapide, parcourir une liste une fois
Quasi-linéaire	$O(n\log_2(n))$	tri le plus efficace
Polynôme	$O(n^k)$	lent si k>3
Exponentiel	$O(k^n)(k>1)$	inutilisable pour les grandes données

Complexité : comment la calculer

- Réfléchissez au paramètre : *quel est n ? Combien y a-t-il de paramètres ?*
- Multiplier par le nombre de boucles

```
for i in range(n):
  # opérations simples
```

$O(n)$

```
for i in range(n):
  for i in range(m):
    # opérations simples
```

$O(n*m)$

```
for i in range(n):
    # code en O(n)
```

$O(n^2)$

- Lors de l'appel de fonctions : prendre en compte leur complexité.
  - Pour vos fonction : la calculer
  - De Python/bibliothèques : chercher l'information
- Ne conserver que le degré le plus haut du polynome

Important

Complexité des opérations Python sur les listes

Operations	List (Amortised worst case in tim)
Add element	`l.insert(i,v)` $\mathcal{O}(n)$
Add first	`l.insert(0,v)` $\mathcal{O}(n)$
Add last	`l.append(v)` $\mathcal{O}(1)$
Remove a value	`l.remove(v)` $\mathcal{O}(n)$
Remove first	`l.pop(0)` $\mathcal{O}(n)$
Remove last	`l.pop()` $\mathcal{O}(1)$
Remove a key	`l.pop(k)` $\mathcal{O}(n)$

Index (Read/write)	`l[k] = 42` $\mathcal{O}(1)$
Containment (value)	`if v in l` $\mathcal{O}(n)$
Containment (key)	
Merge	`l = l1 + l2` $\mathcal{O}(len(l1) + len(l2))$

Complexité

Calculez la complexité de `calcul_ecart_type` dans les deux cas (différences entourées en rouge).

```python
def calcul_moyenne(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)

def calcul_ecart_type(liste):
    res = 0
    moyenne = calcul_moyenne(liste)
    for i in range(len(liste)):
        res += (liste[i] - moyenne)**2
    return sqrt(res / len(liste))
```

```python
def calcul_moyenne(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)

def calcul_ecart_type(liste):
    res = 0
    for i in range(len(liste)):
        res += (liste[i] - calcul_moyenne(liste))**2
    return sqrt(res / len(liste))
```


Exemple/Démonstration

Complexité

Calculez la complexité de `calcul_ecart_type` dans les deux cas .

Que vaut n ? La taille de la liste

```
def calcul_moyenne(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)

def calcul_ecart_type(liste):
    res = 0
    moyenne = calcul_moyenne(liste)
    for i in range(len(liste)):
        res += (liste[i] - moyenne)**2
    return sqrt(res / len(liste))
```

```
(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)

def calcul_ecart_type(liste):
    res = 0
    for i in range(len(liste)):
        res += (liste[i] - calcul_moyenne(liste))**2
    return sqrt(res / len(liste))
```


Exemple/Démonstration

Complexité

Calculez la complexité de `calcul_ecart_type` dans les deux cas.

```python
def calcul_moyenne(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)

def calcul_ecart_type(liste):
    res = 0
    moyenne = calcul_moyenne(liste)
    for i in range(len(liste)):
        res += (liste[i] - moyenne)**2
    return sqrt(res / len(liste))
```

$O(n)$

$O(n)$

```python
def calcul_moyenne(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)

def calcul_ecart_type(liste):
    res = 0
    for i in range(len(liste)):
        res += (liste[i] - calcul_moyenne(liste))**2
    return sqrt(res / len(liste))
```


Exemple/Démonstration

Complexité

Calculez la complexité de `calcul_ecart_type` dans les deux cas.

```python
def calcul_moyenne(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)
```

$O(n)$

```python
def calcul_ecart_type(liste):
    res = 0
    moyenne = calcul_moyenne(liste)
    for i in range(len(liste)):
        res += (liste[i] - moyenne)**2
    return sqrt(res / len(liste))
```

$O(n)$ $O(n)$

```python
def calcul_moyenne(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)

def calcul_ecart_type(liste):
    res = 0
    for i in range(len(liste)):
        res += (liste[i] - calcul_moyenne(liste))**2
    return sqrt(res / len(liste))
```


Exemple/Démonstration

Complexité

Calculez la complexité de `calcul_ecart_type` dans les deux cas.

```python
def calcul_moyenne(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)
```

$O(n)$

```python
def calcul_ecart_type(liste):
    res = 0
    moyenne = calcul_moyenne(liste)
    for i in range(len(liste)):
        res += (liste[i] - moyenne)**2
    return sqrt(res / len(liste))
```

$O(n)$ $O(n)$ $O(1)$

```python
def calcul_moyenne(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)

def calcul_ecart_type(liste):
    res = 0
    for i in range(len(liste)):
        res += (liste[i] - calcul_moyenne(liste))**2
    return sqrt(res / len(liste))
```


Exemple/Démonstration

Complexité

Calculez la complexité de `calcul_ecart_type` dans les deux cas.

```
def calcul_moyenne(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)
```

$O(n)$

```
def calcul_ecart_type(liste):
    res = 0
    moyenne = calcul_moyenne(liste)
    for i in range(len(liste)):
        res += (liste[i] - moyenne)**2
    return sqrt(res / len(liste))
```

$O(n)$ $O(n)$ $O(1)$

$$O(2n)=O(n)$$

```
def calcul_moyenne(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)

def calcul_ecart_type(liste):
    res = 0
    for i in range(len(liste)):
        res += (liste[i] - calcul_moyenne(liste))**2
    return sqrt(res / len(liste))
```

Complexité

Calculez la complexité de `calcul_ecart_type` dans les deux cas.

```python
def calcul_moyenne(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)

def calcul_ecart_type(liste):
    res = 0
    moyenne = calcul_moyenne(liste)
    for i in range(len(liste)):
        res += (liste[i] - moyenne)**2
    return sqrt(res / len(liste))
```

$$O(2n)=O(n)$$

```python
def calcul_moyenne(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)

def calcul_ecart_type(liste):
    res = 0
    for i in range(len(liste)):
        res += (liste[i] - calcul_moyenne(liste))**2
    return sqrt(res / len(liste))
```

$O(n)$

Exemple/Démonstration

Complexité

Calculez la complexité de `calcul_ecart_type` dans les deux cas.

```python
def calcul_moyenne(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)

def calcul_ecart_type(liste):
    res = 0
    moyenne = calcul_moyenne(liste)
    for i in range(len(liste)):
        res += (liste[i] - moyenne)**2
    return sqrt(res / len(liste))
```

$$O(2n) = O(n)$$

```python
def calcul_moyenne(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)

def calcul_ecart_type(liste):
    res = 0
    for i in range(len(liste)):        # O(n)
        res += (liste[i] - calcul_moyenne(liste))**2   # O(n)
    return sqrt(res / len(liste))
```


Exemple/Démonstration

Complexité

Calculez la complexité de `calcul_ecart_type` dans les deux cas.

```
def calcul_moyenne(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)

def calcul_ecart_type(liste):
    res = 0
    moyenne = calcul_moyenne(liste)
    for i in range(len(liste)):
        res += (liste[i] - moyenne)**2
    return sqrt(res / len(liste))
```

$O(2n)=O(n)$

```
def calcul_moyenne(liste):
    somme = 0
    for i in range(len(liste)) :
        somme += liste[i]
    return somme / len(liste)

def calcul_ecart_type(liste):
    res = 0
    for i in range(len(liste)):
        res += (liste[i] - calcul_moyenne(liste))**2
    return sqrt(res / len(liste))
```

$O(n)$ $O(n)$

$O(n^2)$

Exemple/Démonstration