Apprendre la Cyber par les Challenges

Du Lycee a l'Universite

LearnByAoC

Professeur Kali

Base sur les challenges Advent of Code

7 décembre 2025

Table des matières

Preface

Ce livre est ne d'une conviction : **on apprend mieux en resolvant des problemes qu'en lisant des cours**.

Les challenges d'Advent of Code, crees par Eric Wastl depuis 2015, offrent une progression ideale pour apprendre la programmation et la pensee algorithmique. Plus de 250 problemes, du plus simple au plus complexe, couvrant pratiquement tous les concepts fondamentaux de l'informatique.

Comment utiliser ce livre ?

— Chaque chapitre couvre un **concept** (pas une annee d'AoC)

— Les concepts sont ordonnes par **prerequis**

— Chaque concept est illustre par plusieurs **challenges**

— Les explications vont du plus simple (10 ans) au plus technique

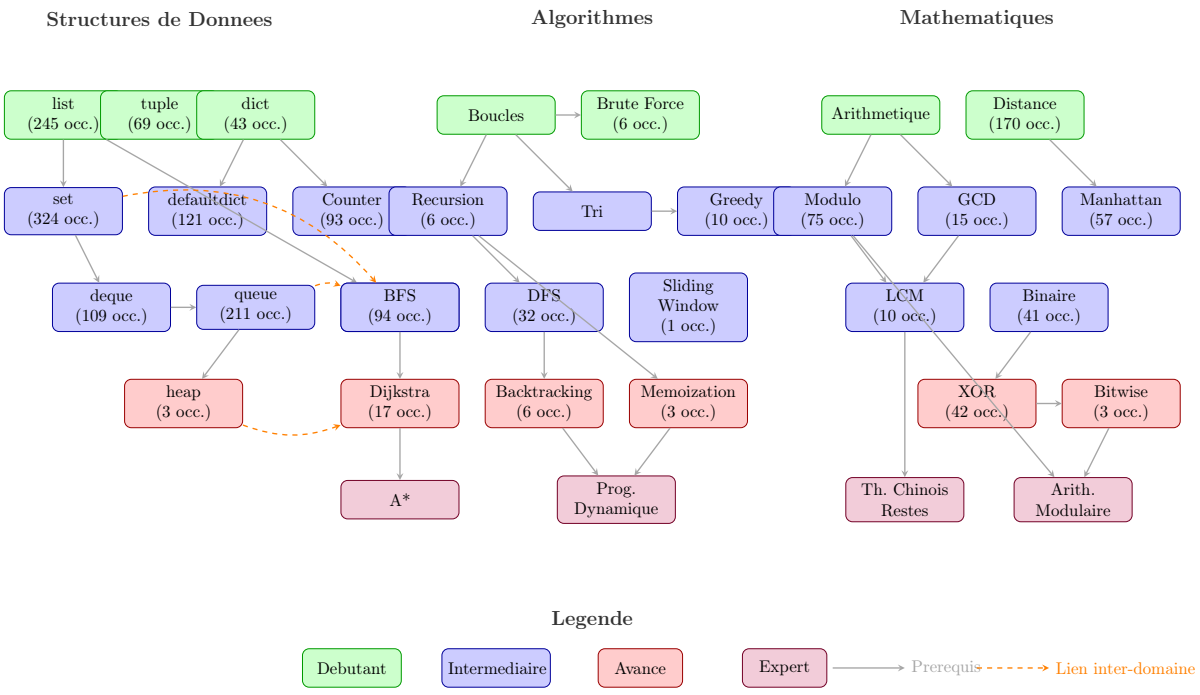
En Cybersecurite

Chaque concept est relie a une application concrete en cybersecurite. Tu ne fais pas que resoudre des puzzles - tu construis des competences reelles.

"On n'apprend pas a nager en lisant un livre. On saute dans l'eau, on boit la tasse, puis on comprend."

Carte des Concepts

Le graphe ci-dessous montre les **dependances entre concepts**. Une fleche de A vers B signifie que tu dois maitriser A avant d'apprendre B.



Legende des couleurs :

- **Vert** : Niveau Debutant (Lycee)
- **Bleu** : Niveau Intermediaire (Bac / L1)
- **Rouge** : Niveau Avance (L2 / L3)
- **Violet** : Niveau Expert (Master)

Les nombres entre parentheses indiquent combien de fois ce concept apparait dans les 257 challenges Advent of Code (2015-2025).

Première partie

Fondations

Chapitre 1

Parsing : Lire et Transformer les Donnees

L'Histoire

Imagine que tu recois une lettre codee. Avant de pouvoir comprendre le message, tu dois d'abord **decoder** les symboles. C'est exactement ce qu'on fait en programmation : transformer du texte brut en donnees utilisables.

1.1 Qu'est-ce que le parsing ?

Definition 1.1 (Parsing)**.** Le **parsing** (ou analyse syntaxique) est l'action de lire des donnees brutes (texte, fichier) et de les transformer en une structure exploitable par un programme.

Exemple 1.1. Transformer la chaine `"123,456,789"` en liste `[123, 456, 789]`.

1.2 Lire un fichier

1.2.1 La methode de base

```
# Methode 1 : Lire tout le fichier
with open("input.txt") as f:
    contenu = f.read()

# Methode 2 : Lire ligne par ligne
with open("input.txt") as f:
    lignes = f.readlines()

# Methode 3 : Iteration directe (recommande)
with open("input.txt") as f:
    for ligne in f:
        print(ligne.strip())
```

Listing 1.1 – Lire un fichier ligne par ligne

Concept Cle

Le mot-cle `with` garantit que le fichier sera ferme automatiquement, meme en cas d'erreur. C'est une bonne pratique a toujours utiliser.

1.2.2 Nettoyer les donnees

```
ligne = "  Hello World  \n"

ligne.strip()      # "Hello World" (supprime espaces + \n)
ligne.rstrip()     # "  Hello World" (seulement a droite)
ligne.lstrip()     # "Hello World  \n" (seulement a gauche)
```

Listing 1.2 – Methodes de nettoyage

1.3 Decouper une chaine

1.3.1 split() : le couteau suisse

```
# Decouper par espaces (defaut)
"a b c".split()         # ['a', 'b', 'c']

# Decouper par un caractere specifique
"a,b,c".split(",")      # ['a', 'b', 'c']

# Decouper par une chaine
"a->b->c".split("->")  # ['a', 'b', 'c']

# Limiter le nombre de decoupages
"a,b,c,d".split(",", 2)  # ['a', 'b', 'c,d']
```

Listing 1.3 – Utilisation de split()

Piege a Eviter

`split()` sans argument decoupe par **tous les espaces blancs** (espaces, tabs, newlines) et supprime les elements vides. `split(" ")` ne decoupe que par l'espace simple.

```
"a  b".split()     # ['a', 'b'] (2 espaces ignores)
"a  b".split(" ")  # ['a', '', 'b'] (element vide!)
```

1.4 Convertir les types

```
# String vers entier
int("42")          # 42
int("  42  ")      # 42 (strip automatique)
int("42abc")       # ERREUR !

# String vers flottant
float("3.14")      # 3.14

# Entier vers string
str(42)            # "42"

# Conversion en masse avec map()
nombres = list(map(int, ["1", "2", "3"]))  # [1, 2, 3]

# Ou avec une comprehension (plus lisible)
nombres = [int(x) for x in ["1", "2", "3"]]  # [1, 2, 3]
```

Listing 1.4 – Conversions de types

1.5 Patterns de parsing courants

1.5.1 Pattern 1 : Liste de nombres

Entree :

```
123
456
789
```

```python
with open("input.txt") as f:
    nombres = [int(ligne) for ligne in f]
# [123, 456, 789]
```

Listing 1.5 – Parser une liste de nombres

1.5.2 Pattern 2 : Valeurs separees par virgules

Entree : `1,2,3,4,5`

```python
with open("input.txt") as f:
    nombres = [int(x) for x in f.read().strip().split(",")]
# [1, 2, 3, 4, 5]
```

Listing 1.6 – Parser des valeurs CSV

1.5.3 Pattern 3 : Grille 2D

Entree :

```
@.@.
.@@.
@.@.
```

```python
with open("input.txt") as f:
    grille = [list(ligne.strip()) for ligne in f]
# [['@', '.', '@', '.'],
#  ['.', '@', '@', '.'],
#  ['@', '.', '@', '.']]

# Acces : grille[ligne][colonne]
grille[0][0]  # '@'
grille[1][1]  # '@'
```

Listing 1.7 – Parser une grille

1.5.4 Pattern 4 : Sections separees par ligne vide

Entree :

```
section1
data1

section2
data2
```

```python
with open("input.txt") as f:
    sections = f.read().strip().split("\n\n")
# ['section1\ndata1', 'section2\ndata2']

# Puis parser chaque section
for section in sections:
    lignes = section.split("\n")
    # ...
```

Listing 1.8 – Parser des sections

1.6 Challenges d'application

Exercice

AoC 2015 Day 1 - Not Quite Lisp
Parse une chaine de caracteres `(` et `)` et compte combien de fois chaque caractere apparait.
Difficulte : ★
Concepts : Iteration sur string, comptage

Exercice

AoC 2020 Day 4 - Passport Processing
Parse des "passeports" avec des champs `cle:valeur` separes par espaces ou newlines.
Difficulte : ★★
Concepts : Split multiple, dictionnaires

Exercice

AoC 2016 Day 4 - Security Through Obscurity
Parse des noms de salles au format `nom-avec-tirets-123[checksum]`.
Difficulte : ★★★
Concepts : Regex, extraction de patterns

Google Colab

Notebook interactif - Parsing
`https://colab.research.google.com/github/kless/LearnByAoC/blob/main/notebooks/01_parsing.ipynb`

1.7 Application en cybersecurite

En Cybersecurite

Le parsing est **fondamental** en cyber :

- **Analyse de logs** : Parser les logs Apache, auth.log, syslog
- **Parsing de packets** : Extraire les headers HTTP, DNS queries
- **Reverse engineering** : Parser des formats binaires
- **OSINT** : Extraire des donnees de pages web

Attention : Un parsing mal fait peut creer des vulnerabilites (injection, buffer overflow). Toujours valider les donnees !

1.8 Resume

Methode	Usage
`open() / with`	Lire un fichier
`.strip()`	Nettoyer les espaces
`.split()`	Decouper une chaine
`int() / float()`	Convertir en nombre
`map()`	Appliquer une fonction
`[x for x in ...]`	Comprehension de liste

Chapitre 2

Arithmetique : Compter et Calculer

L'Histoire

Tu te souviens quand tu as appris a compter sur tes doigts ? En programmation, on fait pareil - mais avec des millions de nombres, et beaucoup plus vite !

2.1 Les operations de base

Definition 2.1 (Operateurs arithmetiques)**.** Python utilise les operateurs classiques :

- `+` : Addition
- `-` : Soustraction
- `*` : Multiplication
- `/` : Division (resultat flottant)
- `//` : Division entiere
- `%` : Modulo (reste de la division)
- `**` : Puissance

```
17 / 5    # 3.4   (division flottante)
17 // 5   # 3     (division entiere)
17 % 5    # 2     (reste : 17 = 3*5 + 2)
2 ** 10   # 1024  (2 puissance 10)
```

Listing 2.1 – Operations arithmetiques

2.2 Le modulo : ton nouvel ami

Concept Cle

Le **modulo** (%) donne le **reste** de la division. C'est l'operation la plus utile en programmation !

```
# Pair ou impair ?
n % 2 == 0  # True si n est pair

# Position dans un cycle (0 a n-1)
jour = total_jours % 7  # Jour de la semaine (0-6)

# Eviter les depassements de liste
```

```
index = i % len(liste)  # Revient au debut si depasse

# Extraire le dernier chiffre
dernier = n % 10  # Chiffre des unites
```

Listing 2.2 – Usages courants du modulo

Exercice

AoC 2015 Day 1 - Not Quite Lisp
Utilise un compteur simple : `(` ajoute 1, `)` soustrait 1. Trouve l'etage final.
Concepts : Iteration, comptage, conditions

2.3 GCD et LCM : cycles et periodicite

Definition 2.2 (PGCD et PPCM). — **PGCD** (GCD) : Plus Grand Commun Diviseur — **PPCM** (LCM) : Plus Petit Commun Multiple

```
import math

def lcm(a, b):
    return a * b // math.gcd(a, b)

# Quand 3 cycles de periodes 4, 6, 9 se synchronisent ?
from functools import reduce
periodes = [4, 6, 9]
sync = reduce(lcm, periodes)  # 36
```

Listing 2.3 – Calcul du LCM

En Cybersecurite

Le PGCD est utilise en **cryptographie** (RSA) et le PPCM pour analyser les **cycles** dans les malwares.

2.4 Challenges d'application

Exercice

AoC 2020 Day 13 - Shuttle Search
Trouve quand plusieurs bus avec des periodes differentes passent en meme temps.
Difficulte : ★★★★
Concepts : LCM, modulo, arithmetique modulaire

2.5 Resume

Concept	Usage principal
`%` (modulo)	Cycles, parite, position circulaire
`//` (div. entiere)	Quotient sans decimales
`math.gcd()`	Plus grand diviseur commun
`math.lcm()`	Synchronisation de cycles

Chapitre 3

Recherche : Trouver ce qu'on Cherche

L'Histoire

Imagine que tu cherches un mot dans le dictionnaire. Tu ne lis pas toutes les pages depuis le debut ! Tu ouvres au milieu, tu regardes si c'est avant ou apres, et tu recommences. C'est la **recherche binaire**.

3.1 Recherche lineaire

Definition 3.1 (Recherche lineaire)**.** Parcourir tous les elements un par un jusqu'a trouver ce qu'on cherche. Complexite : O(n).

3.2 Recherche dans un set : O(1)

Concept Cle

Un `set` utilise une **table de hachage**. La recherche est instantanee (O(1)).

```python
# LENT : recherche dans une liste O(n)
if x in liste:  # parcourt toute la liste
    ...

# RAPIDE : recherche dans un set O(1)
ensemble = set(liste)
if x in ensemble:  # acces direct
    ...
```

Listing 3.1 – Comparaison list vs set

3.3 Recherche binaire

Definition 3.2 (Recherche binaire)**.** Diviser l'espace de recherche en deux a chaque etape. Complexite : O(log n).

```python
import bisect

liste = [1, 3, 5, 7, 9]
bisect.bisect_left(liste, 6)   # 3 (position d'insertion)
```

Listing 3.2 – Module bisect

3.4 Challenges d'application

Exercice

AoC 2020 Day 1 - Report Repair
Trouve deux nombres dont la somme est 2020. Optimise avec un set.
Concepts : Recherche dans set, complement

En Cybersecurite

— **Bruteforce intelligent** : Recherche binaire sur l'espace de cles
— **Analyse de logs** : Recherche rapide dans des GB de logs

3.5 Resume

Methode	Complexite	Quand
Liste + `in`	O(n)	Petite liste
Set + `in`	O(1)	Recherches multiples
Recherche binaire	O(log n)	Donnees triees

Chapitre 4

Grilles 2D : Naviguer dans l'Espace

L'Histoire

Tu connais les echecs ? Une grille 8x8 ou chaque piece peut bouger dans certaines directions. En programmation, on utilise constamment des grilles !

4.1 Representer une grille

```python
# Parser un fichier en grille
with open("input.txt") as f:
    grille = [list(ligne.strip()) for ligne in f]

# Acces : grille[ligne][colonne]
grille[0][0]  # coin haut-gauche

# Dimensions
hauteur = len(grille)
largeur = len(grille[0])
```

Listing 4.1 – Creer et parser une grille

Piege a Eviter

Attention : `grille[ligne][colonne]` = `grille[y][x]` !

4.2 Les 4 directions

```python
DIRECTIONS = [
    (-1, 0),  # Haut
    (1, 0),   # Bas
    (0, -1),  # Gauche
    (0, 1)    # Droite
]
```

Listing 4.2 – Directions cardinales

4.3 Distance de Manhattan

Definition 4.1 (Distance de Manhattan)**.** Distance en blocs : $d = |x_1 - x_2| + |y_1 - y_2|$

```python
def manhattan(p1, p2):
    return abs(p1[0] - p2[0]) + abs(p1[1] - p2[1])
```

Listing 4.3 – Calcul Manhattan

4.4 Challenges d'application

Exercice

AoC 2015 Day 3 - Perfectly Spherical Houses
Suis des directions et compte les maisons visitees.
Concepts : Directions, set de positions

En Cybersecurite

— **Analyse d'images** : Steganographie, detection
— **Cartographie reseau** : Visualisation de topologie

Deuxième partie

Structures de Donnees

Chapitre 5

Ensembles (Sets) : L'Unicite

L'Histoire

Imagine un sac magique ou tu ne peux mettre qu'un seul exemplaire de chaque objet. Si tu essaies de mettre deux pommes identiques, le sac n'en garde qu'une. C'est un **ensemble** !

5.1 Qu'est-ce qu'un set ?

Definition 5.1 (Set)**.** Un `set` est une collection **non ordonnee** d'elements **uniques**. Recherche en O(1).

```
# Creation
ensemble = {1, 2, 3}
ensemble = set([1, 2, 2, 3])  # {1, 2, 3}

# Ajout / Suppression
ensemble.add(4)
ensemble.remove(1)      # Erreur si absent
ensemble.discard(10)    # Pas d'erreur si absent

# Test d'appartenance O(1)
if x in ensemble:
    ...
```

Listing 5.1 – Operations de base

5.2 Operations ensemblistes

```
a = {1, 2, 3}
b = {2, 3, 4}

a | b   # {1, 2, 3, 4}  Union
a & b   # {2, 3}        Intersection
a - b   # {1}           Difference
a ^ b   # {1, 4}        Difference symetrique
```

Listing 5.2 – Union, intersection, difference

5.3 Pattern : Positions visitees

```python
visited = set()
position = (0, 0)

while True:
    if position in visited:
        print("Deja visite !")
        break
    visited.add(position)
    position = move(position)
```

Listing 5.3 – Tracker les positions

5.4 Challenges d'application

Exercice

AoC 2015 Day 3 - Perfectly Spherical Houses
Compte les maisons uniques visitees par le Pere Noel.
Concepts : Set de tuples, positions

En Cybersecurite

— **Deduplication** : Supprimer les doublons dans les logs
— **Detection d'anomalies** : Nouvelles IPs, nouveaux users

Chapitre 6

Dictionnaires : Associer Cles et Valeurs

L'Histoire

Un dictionnaire, c'est comme ton carnet de contacts : tu cherches un nom (la cle) et tu trouves le numero (la valeur). Instantanement.

6.1 Qu'est-ce qu'un dict ?

Definition 6.1 (Dictionnaire)**.** Un `dict` associe des **cles** a des **valeurs**. Acces en O(1).

```
# Creation
scores = {"Alice": 100, "Bob": 85}

# Acces
scores["Alice"]       # 100
scores.get("Eve", 0) # 0 (defaut si absent)

# Modification
scores["Alice"] = 110
scores["Eve"] = 95    # Nouvelle entree

# Iteration
for nom, score in scores.items():
    print(f"{nom}: {score}")
```

Listing 6.1 – Operations de base

6.2 defaultdict : le dict intelligent

```
from collections import defaultdict

# Comptage
compteur = defaultdict(int)
for mot in mots:
    compteur[mot] += 1  # Pas besoin d'initialiser !

# Listes de valeurs
graphe = defaultdict(list)
graphe["A"].append("B")  # Pas besoin de verifier si "A" existe
```

Listing 6.2 – defaultdict pour eviter les KeyError

6.3 Counter : compter automatiquement

```python
from collections import Counter

texte = "abracadabra"
freq = Counter(texte)
# Counter({'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1})

freq.most_common(3)  # [('a', 5), ('b', 2), ('r', 2)]
```

Listing 6.3 – Counter pour les frequences

6.4 Challenges d'application

Exercice

AoC 2024 Day 1 - Historian Hysteria
Compte les occurrences et calcule un score de similarite.
Concepts : Counter, multiplication

En Cybersecurite

— **Analyse de frequence** : Casser le chiffrement par substitution
— **Cache/Memoization** : Accelerer les calculs

Chapitre 7

Piles et Files : L'Ordre de Traitement

L'Histoire

Une pile d'assiettes : tu poses en haut, tu prends en haut (LIFO). Une file d'attente au cinema : le premier arrive est le premier servi (FIFO).

7.1 La Pile (Stack) : LIFO

Definition 7.1 (Pile)**.** Last In, First Out. Le dernier element ajoute est le premier retire.

```
pile = []
pile.append(1)  # Push
pile.append(2)
pile.pop()      # 2 (Pop)
pile[-1]        # 1 (Peek sans retirer)
```

Listing 7.1 – Pile avec une liste

7.2 La File (Queue) : FIFO

Definition 7.2 (File)**.** First In, First Out. Le premier element ajoute est le premier retire.

```
from collections import deque

file = deque()
file.append(1)     # Ajouter a droite
file.append(2)
file.popleft()     # 1 (Retirer a gauche)
```

Listing 7.2 – File avec deque

Concept Cle

`deque` est O(1) aux deux extremites. Une liste est O(n) pour `pop(0)` !

7.3 Applications

— **Pile** : Parentheses equilibrees, DFS, undo/redo

— **File** : BFS, traitement dans l'ordre d'arrivee

7.4 Challenges d'application

Exercice

AoC 2021 Day 10 - Syntax Scoring
Verifie l'equilibrage des parentheses avec une pile.
Concepts : Pile, matching de caracteres

En Cybersecurite

— **Buffer overflow** : Comprendre la pile d'execution
— **Message queues** : Communication inter-processus

Chapitre 8

Arbres : Structures Hierarchiques

L'Histoire

Un arbre genealogique : grand-parents en haut, parents au milieu, enfants en bas. Chaque personne a un parent (sauf la racine) et peut avoir plusieurs enfants.

8.1 Qu'est-ce qu'un arbre ?

Definition 8.1 (Arbre)**.** Structure hierarchique avec une **racine**, des **noeuds** et des **feuilles** (noeuds sans enfants).

```python
# Arbre comme dict parent -> enfants
arbre = {
    "COM": ["B"],
    "B": ["C", "G"],
    "C": ["D"],
    "G": ["H"],
}

# Ou enfant -> parent (pour remonter)
parents = {
    "B": "COM",
    "C": "B",
    "D": "C",
}
```

Listing 8.1 – Representation avec dict

8.2 Parcours d'arbre

```python
def parcours(noeud, arbre, profondeur=0):
    print("  " * profondeur + noeud)
    for enfant in arbre.get(noeud, []):
        parcours(enfant, arbre, profondeur + 1)
```

Listing 8.2 – Parcours recursif

8.3 Challenges d'application

Exercice

AoC 2019 Day 6 - Universal Orbit Map
Compte les orbites directes et indirectes dans un arbre.
Concepts : Parcours d'arbre, comptage de profondeur

En Cybersecurite

— **Systeme de fichiers** : Arborescence de repertoires
— **DOM/XML** : Parsing de documents structures

Troisième partie

Algorithmes

Chapitre 9

Recursion : Se Rappeler Soi-meme

L'Histoire

Pour comprendre la recursion, il faut d'abord comprendre la recursion. C'est une blague de programmeur ! Une fonction qui s'appelle elle-meme pour resoudre un probleme plus petit.

9.1 Qu'est-ce que la recursion ?

Definition 9.1 (Recursion)**.** Une fonction qui s'appelle elle-meme avec un cas de base pour s'arreter.

```python
def factorielle(n):
    if n <= 1:          # Cas de base
        return 1
    return n * factorielle(n - 1)  # Appel recursif

# 5! = 5 * 4 * 3 * 2 * 1 = 120
```

Listing 9.1 – Exemple : factorielle

9.2 Memoization : eviter les recalculs

```python
from functools import lru_cache

@lru_cache(maxsize=None)
def fib(n):
    if n < 2:
        return n
    return fib(n-1) + fib(n-2)

# Sans cache : O(2^n) - EXPLOSIF
# Avec cache : O(n) - lineaire
```

Listing 9.2 – Fibonacci avec memoization

Concept Cle

La **memoization** stocke les resultats deja calcules. Essentiel pour la programmation dynamique !

9.3 Challenges d'application

Exercice

AoC 2015 Day 7 - Some Assembly Required
Evalue des circuits logiques avec des dependances recursives.
Concepts : Recursion, memoization, graphe de dependances

En Cybersecurite

— **Parsing recursif** : Grammaires, expressions
— **Exploration de systemes** : Fichiers, processus

Chapitre 10

Graphes : BFS, DFS et Dijkstra

L'Histoire

Un reseau social est un graphe : les personnes sont des noeuds, les amities sont des aretes. Comment trouver le chemin le plus court entre deux personnes ?

10.1 Qu'est-ce qu'un graphe ?

Definition 10.1 (Graphe)**.** Ensemble de **noeuds** (sommets) relies par des **aretes** (liens).

```python
from collections import defaultdict

graphe = defaultdict(list)
graphe["A"].append("B")
graphe["A"].append("C")
graphe["B"].append("D")
```

Listing 10.1 – Representation par liste d'adjacence

10.2 BFS : Largeur d'abord

```python
from collections import deque

def bfs(start, goal, graphe):
    queue = deque([(start, 0)])
    visited = {start}

    while queue:
        node, dist = queue.popleft()

        if node == goal:
            return dist

        for voisin in graphe[node]:
            if voisin not in visited:
                visited.add(voisin)
                queue.append((voisin, dist + 1))

    return -1
```

Listing 10.2 – BFS - Plus court chemin non pondere

10.3 DFS : Profondeur d'abord

```python
def dfs(node, visited, graphe):
    if node in visited:
        return
    visited.add(node)

    for voisin in graphe[node]:
        dfs(voisin, visited, graphe)
```

Listing 10.3 – DFS recursif

10.4 Dijkstra : Chemins ponderes

```python
import heapq

def dijkstra(start, goal, graphe):
    pq = [(0, start)]
    distances = {start: 0}

    while pq:
        dist, node = heapq.heappop(pq)

        if node == goal:
            return dist

        if dist > distances.get(node, float('inf')):
            continue

        for voisin, poids in graphe[node]:
            new_dist = dist + poids
            if new_dist < distances.get(voisin, float('inf')):
                distances[voisin] = new_dist
                heapq.heappush(pq, (new_dist, voisin))

    return -1
```

Listing 10.4 – Dijkstra avec heapq

10.5 Challenges d'application

Exercice

AoC 2016 Day 13 - A Maze of Twisty Little Cubicles
BFS dans un labyrinthe genere par une formule.
Difficulte : ★★
Concepts : BFS, generation procedurale

Exercice

AoC 2021 Day 15 - Chiton
Trouve le chemin de risque minimal dans une grille.
Difficulte : ★★★
Concepts : Dijkstra, grille comme graphe

En Cybersecurite

— **Scan reseau** : Cartographie de topologie
— **Analyse de malware** : Graphe d'appels de fonctions
— **Pathfinding** : Routage, navigation

Chapitre 11

Programmation Dynamique : Optimiser par Sous-problemes

L'Histoire

Tu veux monter un escalier. A chaque marche, tu peux faire 1 ou 2 pas. Combien de facons differentes d'arriver en haut ? Au lieu de tout recalculer, on reutilise les resultats precedents.

11.1 Qu'est-ce que la DP ?

Definition 11.1 (Programmation Dynamique)**.** Technique qui resout un probleme en le decomposant en **sous-problemes** et en stockant leurs solutions pour eviter les recalculs.

```python
def escalier(n):
    # dp[i] = nombre de facons d'atteindre marche i
    dp = [0] * (n + 1)
    dp[0] = 1  # 1 facon de rester en bas
    dp[1] = 1  # 1 facon d'atteindre marche 1

    for i in range(2, n + 1):
        dp[i] = dp[i-1] + dp[i-2]

    return dp[n]
```

Listing 11.1 – Exemple : Escalier

11.2 Les deux approches

— **Top-down** : Recursion + memoization

— **Bottom-up** : Iteration, remplir un tableau

11.3 Challenges d'application

Exercice

AoC 2020 Day 10 - Adapter Array
Compte les arrangements possibles d'adaptateurs.
Concepts : DP, comptage de chemins

En Cybersecurite

— **Alignement de sequences** : Comparaison ADN/malware
— **Optimisation** : Allocation de ressources

Chapitre 12

Simulation : Modeliser le Monde

L'Histoire

Le Jeu de la Vie de Conway : des cellules naissent et meurent selon des regles simples. Pourtant, des structures complexes emergent. Bienvenue dans la simulation !

12.1 Automates cellulaires

```python
def step(grille):
    nouvelle = set()

    # Compter les voisins
    for cell in grille:
        voisins = count_neighbors(cell, grille)
        if voisins in [2, 3]:
            nouvelle.add(cell)

    # Naissance
    for cell in candidates(grille):
        if count_neighbors(cell, grille) == 3:
            nouvelle.add(cell)

    return nouvelle
```

Listing 12.1 – Game of Life simplifie

12.2 Detection de cycles

```python
def find_cycle(initial_state, step_func):
    seen = {initial_state: 0}
    state = initial_state
    step = 0

    while True:
        state = step_func(state)
        step += 1

        if state in seen:
            cycle_start = seen[state]
            cycle_length = step - cycle_start
            return cycle_start, cycle_length

        seen[state] = step
```

Listing 12.2 – Trouver un cycle

12.3 Challenges d'application

Exercice

AoC 2020 Day 17 - Conway Cubes
Game of Life en 3D et 4D.
Concepts : Simulation, dimensions multiples

En Cybersecurite

— **Analyse de malware** : Execution symbolique
— **Fuzzing** : Simulation d'entrees

Quatrième partie

Applications Cyber

Chapitre 13

Cryptographie : Cacher et Reveler

L'Histoire

Les espions ont toujours utilise des codes secrets. Aujourd'hui, c'est pareil mais avec des mathematiques. Comment cacher un message pour que seul le destinataire puisse le lire ?

13.1 XOR : le chiffrement reversible

Definition 13.1 (XOR). Operation binaire : 1 XOR 1 = 0, 1 XOR 0 = 1, 0 XOR 0 = 0. Propriete magique : A XOR B XOR B = A

```python
def xor_encrypt(message, key):
    return bytes([m ^ k for m, k in zip(message, cycle(key))])

# Dechiffrement = meme operation !
def xor_decrypt(ciphertext, key):
    return xor_encrypt(ciphertext, key)
```

Listing 13.1 – Chiffrement XOR

13.2 Hachage : l'empreinte unique

```python
import hashlib

message = b"secret"
hash_md5 = hashlib.md5(message).hexdigest()
# '5ebe2294ecd0e0f08eab7690d2a6ee69'
```

Listing 13.2 – MD5 en Python

13.3 Challenges d'application

Exercice

AoC 2015 Day 4 - The Ideal Stocking Stuffer
Trouve un nombre qui produit un hash MD5 commencant par des zeros.
Concepts : Hachage, bruteforce

Exercice

AoC 2016 Day 5 - How About a Nice Game of Chess ?
Derive un mot de passe a partir de hash MD5 successifs.
Concepts : Hachage iteratif

En Cybersecurite

— **Cracking de mots de passe** : Rainbow tables, hashcat
— **Integrite** : Verification de fichiers
— **Blockchain** : Proof of work

Chapitre 14

Parsing Avance : Regex et Grammaires

L'Histoire

Parfois, les donnees sont tellement complexes qu'un simple `split()` ne suffit plus. Il faut des outils plus puissants : les expressions regulieres.

14.1 Expressions regulieres (Regex)

```python
import re

texte = "Il y a 42 pommes et 17 oranges"

# Trouver tous les nombres
re.findall(r'\d+', texte)  # ['42', '17']

# Extraire avec groupes
match = re.search(r'(\d+) pommes', texte)
match.group(1)  # '42'

# Remplacer
re.sub(r'\d+', 'X', texte)  # 'Il y a X pommes et X oranges'
```

Listing 14.1 – Regex de base

14.2 Patterns courants

Pattern	Signification
`\d`	Chiffre
`\w`	Lettre ou chiffre
`\s`	Espace
`.`	N'importe quel caractere
`+`	1 ou plus
`*`	0 ou plus
`?`	0 ou 1
`()`	Groupe de capture

14.3 Challenges d'application

Exercice

AoC 2015 Day 5 - Doesn't He Have Intern-Elves For This ?
Valide des strings avec des regles complexes.
Concepts : Regex, validation

En Cybersecurite

— **Log parsing** : Extraction d'IPs, timestamps
— **Input validation** : Securiser les formulaires
— **Grep/sed/awk** : Outils Unix essentiels

Chapitre 15

Optimisation : Faire Plus Vite

L'Histoire

Ton programme fonctionne, mais il met 10 minutes. Comment le faire tourner en 10 secondes ? L'optimisation, c'est l'art de faire plus avec moins.

15.1 Identifier les goulots

```python
import time

start = time.time()
# Code a mesurer
result = slow_function()
print(f"Temps: {time.time() - start:.2f}s")
```

Listing 15.1 – Profiling simple

15.2 Complexite algorithmique

Complexite	Exemple	1M elements
O(1)	Acces dict	Instantane
O(log n)	Recherche binaire	20 ops
O(n)	Parcours liste	1M ops
O(n log n)	Tri	20M ops
$O(n^2)$	Double boucle	1T ops
$O(2^n)$	Bruteforce	Impossible

15.3 Techniques courantes

— Utiliser `set` au lieu de `list` pour les recherches

— Memoization pour eviter les recalculs

— Eviter les copies inutiles de listes

— Utiliser des generateurs au lieu de listes

15.4 Challenges d'application

Exercice

AoC 2025 Day 7 - Laboratories
Optimise une simulation exponentielle avec `defaultdict(int)`.
Concepts : Comptage au lieu d'enumeration

En Cybersecurite

— **Timing attacks** : Mesurer les differences de temps
— **DoS** : Exploiter la complexite algorithmique

Cinquième partie

Annexes

Annexe A

Reference Python

A.1 Collections

```python
from collections import deque, Counter, defaultdict
from itertools import permutations, combinations, product
from functools import lru_cache, reduce
import heapq
import math
import re
```

Listing A.1 – Imports essentiels

A.2 Comprehensions

```python
# Liste
[x*2 for x in range(10) if x % 2 == 0]

# Dict
{k: v for k, v in items}

# Set
{x % 3 for x in range(10)}

# Generator
(x*2 for x in range(10))
```

Listing A.2 – Syntaxe des comprehensions

A.3 Fonctions utiles

Fonction	Usage
`enumerate()`	Indices + valeurs
`zip()`	Parcourir plusieurs listes
`map()`	Appliquer une fonction
`filter()`	Filtrer des elements
`sorted()`	Trier (renvoie nouvelle liste)
`reversed()`	Inverser
`any()` / `all()`	Tests logiques

Annexe B

Complexite Algorithmique

B.1 Notation Big-O

Definition B.1 (Big-O)**.** La complexite decrit comment le temps d'execution evolue avec la taille de l'entree.

B.2 Complexites courantes

Complexite	**Nom**	**Exemples**
O(1)	Constante	Acces dict/set, operations mathematiques
O(log n)	Logarithmique	Recherche binaire, operations sur heap
O(n)	Lineaire	Parcours de liste, recherche lineaire
O(n log n)	Linearithmique	Tri (merge sort, timsort)
$O(n^2)$	Quadratique	Double boucle imbriquee
$O(n^3)$	Cubique	Triple boucle, multiplication de matrices
$O(2^n)$	Exponentielle	Bruteforce, backtracking naif
O(n !)	Factorielle	Permutations completes

B.3 Regles de calcul

1. On garde le terme dominant : $O(n^2 + n) = O(n^2)$
2. On ignore les constantes : $O(3n) = O(n)$
3. Boucles imbriquees : on multiplie
4. Boucles sequentielles : on additionne

B.4 Complexite spatiale

Definition B.2 (Complexite spatiale)**.** Quantite de memoire utilisee en fonction de la taille de l'entree.

— Liste de n elements : O(n)

— Grille n x n : $O(n^2)$

— Set de positions visitees : O(nombre de positions uniques)

Annexe C

Index des Challenges

C.1 Par Concept Principal

C.1.1 Parsing et I/O

— 2015 Day 1-5 : Parsing basique

— 2020 Day 4 : Parsing complexe (passeports)

— 2016 Day 4 : Regex

C.1.2 Grilles et Navigation

— 2015 Day 3 : Set de positions

— 2015 Day 6 : Grille booleenne

— 2020 Day 11 : Game of Life

— 2016 Day 1 : Distance Manhattan

C.1.3 BFS / Pathfinding

— 2016 Day 13 : BFS dans labyrinthe

— 2022 Day 12 : BFS avec contraintes

— 2021 Day 15 : Dijkstra

— 2018 Day 22 : A* avance

C.1.4 Recursion et DP

— 2015 Day 7 : Recursion avec memoization

— 2020 Day 10 : DP comptage

— 2015 Day 17 : Backtracking

C.1.5 Mathematiques

— 2020 Day 13 : Theoreme chinois des restes

— 2019 Day 12 : LCM pour cycles

— 2015 Day 4 : Hachage MD5

C.2 Par Difficulte

C.2.1 Niveau Debutant (★)

2015 Days 1-5, 2020 Day 1, 2024 Day 1

C.2.2 Niveau Intermediaire (★★)

2016 Day 13, 2015 Day 6-10, 2020 Days 3-8

C.2.3 Niveau Avance (★★★)

2021 Day 15, 2015 Day 7, 2020 Days 10-15

C.2.4 Niveau Expert (★★★★)

2020 Day 13 (CRT), Days 20-25 de toutes les annees

C.3 Challenges Emblematiques

Challenge	Pourquoi c'est un classique
2015 Day 7	Introduction parfaite a la recursion avec memoization
2016 Day 13	BFS pur dans un espace genere
2021 Day 15	Meilleur challenge pour apprendre Dijkstra
2020 Day 13	Le theoreme chinois des restes en pratique
2020 Day 17	Game of Life en N dimensions

Index des Challenges