Introduction à l'algorithmique
(avec l'exemple du sac à dos)

Initiation à l'algorithmique et programmation

Revekka Kyriakoglou

Plan (3h, très guidé)

- Algorithme : entrée, sortie, cas limites
- Coût : intuition (linéaire, quadratique, exponentiel)
- Deux algorithmes simples (max, recherche) + traces à la main
- Récursivité : cas de base / étape récursive + trace
- **Sac à dos 0/1** : brute force $\rightarrow$ programmation dynamique (table)
- Exercices progressifs (papier puis Python)

Algorithme : 3 questions obligatoires

Avant d'écrire du code, on doit répondre à :

1. **Entrée :** quelles données ? quels types ?
2. **Sortie :** que renvoie-t-on ?
3. **Cas limites :** que faire si l'entrée est vide, nulle, etc. ?

En examen : une bonne spécification vaut déjà des points.

Exercice 1 (papier) : spécifier

Pour chaque fonction, écrivez : **entrée**, **sortie**, **cas limites**.

1. `maximum(L)` : maximum d'une liste d'entiers.
2. `contient(L, x)` : renvoie True si `x` est dans `L`.

Coût : une intuition de base

Ordres de grandeur

- Lire n éléments : $\sim n$
- Double boucle sur n : $\sim n^2$
- Tester toutes les combinaisons d'objets : $\sim 2^n$

On ne calcule pas précisément : on compare des **familles** de coûts.

Exercice 2 (papier) : classer des coûts

Classer du plus petit au plus grand quand n grand :

$$n, \quad n^2, \quad 2^n$$

Puis dire quel coût correspond plutôt à :

- parcourir une liste une fois,
- comparer toutes les paires d'éléments,
- tester toutes les sous-listes possibles.

Algorithme 1 : maximum (idée)

Idée

On garde un candidat `m`. On le met à jour si on trouve plus grand.

```
def maximum(L):
    # suppose L non vide
    m = L[0]
    for x in L:
        if x > m:
            m = x
    return m
```

Exercice 3 (papier) : trace de `maximum`

Sans exécuter, compléter la table pour `maximum([3, 1, 10, 2])` :

x lu	m avant	m après
3		
1		
10		
2		

Algorithme 2 : recherche linéaire (idée)

Idée

On compare `x` à chaque élément. On s'arrête dès qu'on trouve.

```
def contient(L, x):
    for y in L:
        if y == x:
            return True
    return False
```

Exercice 4 (papier) : nombre de comparaisons

Pour `L = [3, 1, 10, 2]` :

1. Combien de comparaisons pour `contient(L, 10)` ?
2. Combien de comparaisons pour `contient(L, 7)` ?

Justifiez en une phrase.

Récursivité : comment penser

On écrit toujours :

1. **Cas de base** (le plus simple)
2. **Étape récursive** (on réduit le problème)

Si le problème ne diminue pas, la récursion ne terminera pas.

Exemple : somme d'une liste (sans boucle)

Décomposition

`somme([4,2,1]) = 4 + somme([2,1])`

```
def somme_rec(L):
    if L == []:
        return 0
    return L[0] + somme_rec(L[1:])
```

Exercice 5 (papier) : trace des appels

Écrire les appels successifs puis les retours :

```
somme_rec([4,2,1])
```

Indication : écrire d'abord la descente (appels), puis la remontée (valeurs renvoyées).

Sac à dos

Problème

Vous avez n objets. Objet i :

- poids w_i
- valeur v_i

Capacité du sac : W.

Choisir un sous-ensemble d'objets (0 ou 1 fois chacun) pour :

maximiser la valeur totale sous la contrainte poids total $\leq W$.

Mini-exemple : tester à la main

Capacité $W = 7$. Objets :

$$1 : (w = 2, v = 3) \quad 2 : (3, 4) \quad 3 : (4, 5) \quad 4 : (5, 8)$$

Exercice 6 (papier, 5 min) : proposez **deux** choix possibles (sous-ensembles), et calculez (poids total, valeur totale). Lequel est meilleur ?

Objectif : sentir qu'il y a beaucoup de combinaisons possibles.

Brute force : pourquoi ça explose

Idée brute force

Tester **toutes** les combinaisons : pour chaque objet, on décide **prendre** / **ne pas prendre**.

Nombre de combinaisons : 2^n (exponentiel).

Même si une combinaison se teste vite, 2^n devient vite trop grand.

Programmation dynamique : l'idée en une phrase

Au lieu de recalculer tout, on mémorise les résultats de sous-problèmes.

Sous-problème

$F(i, w) =$ meilleure valeur possible avec les i premiers objets et c

Que signifie $F(i, w)$? (lecture)

Interprétez (en français) :

1. $F(0, 7)$
2. $F(2, 3)$
3. $F(4, 7)$

On lit : « avec les **i premiers objets** et un sac de capacité **w** ».

Récurrence : deux choix

Pour l'objet i

- **Ne pas prendre** l'objet i : valeur $F(i-1, w)$
- **Prendre** l'objet i (si $w_i \leq w$) : valeur $v_i + F(i-1, w-w_i)$

Récurrence (formule)

Formule

$$F(i,w) = \begin{cases} F(i-1,w) & \text{si } w_i > w \\ \max\left(F(i-1,w),\ v_i + F(i-1,w-w_i)\right) & \text{sinon} \end{cases}$$

C'est exactement : « ne pas prendre » vs « prendre ».

Construire la table : règles de remplissage

On remplit ligne par ligne :

- Ligne $i = 0$: 0 partout (aucun objet)
- Pour chaque $i = 1..n$ et chaque $w = 0..W$:
  - si $w_i > w$: on copie la valeur au-dessus
  - sinon : max(au-dessus, valeur+case au-dessus à gauche)

Exercice 7 (papier) : table à compléter

Capacité $W = 7$. Objets (poids, valeur) :

$$1 : (2, 3) \quad 2 : (3, 4) \quad 3 : (4, 5) \quad 4 : (5, 8)$$

Remplissez la table $F(i, w)$ pour $i = 0..4$, $w = 0..7$.

$i \backslash w$	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0						
2	0	0						
3	0	0						
4	0	0						

Correction (table finale)

$i \backslash w$	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	3	3	3	3	3	3
2	0	0	3	4	4	7	7	7
3	0	0	3	4	5	7	8	9
4	0	0	3	4	5	8	8	11

Réponse : $F(4, 7) = 11$.

Comprendre une case (exemple)

Expliquez comment on obtient $F(4,7) = 11$:

- **sans prendre** l'objet 4 : $F(3,7) = 9$
- **en prenant** l'objet 4 (poids 5, valeur 8) : $8 + F(3,2) = 8 + 3 = 11$
- on prend le maximum : 11

Implémentation Python

```
def sac_a_dos_01(poids, valeurs, W):
    n = len(poids)
    F = []
    for i in range(n + 1):
        F.append([0] * (W + 1))

    for i in range(1, n + 1):
        wi = poids[i - 1]
        vi = valeurs[i - 1]
        for w in range(W + 1):
            if wi > w:
                F[i][w] = F[i - 1][w]
            else:
                sans = F[i - 1][w]
                avec = vi + F[i - 1][w - wi]
                if avec > sans:
                    F[i][w] = avec
                else:
                    F[i][w] = sans
    return F[n][W]
```

Exercice 8 (Python) : tester

Testez avec :

$$W = 7, \quad \textit{poids} = [2, 3, 4, 5], \quad \textit{valeurs} = [3, 4, 5, 8]$$

Résultat attendu : `11`.

Si le résultat n'est pas 11 : vérifier les indices `i-1` et `w-wi`.

Exercice 9 (bonus) : retrouver les objets pris

Après avoir rempli la table `F`, remontez pour retrouver les objets :

- partir de `i=n, w=W`
- si `F[i][w] == F[i-1][w]` : objet i non pris, faire `i=i-1`
- sinon : objet i pris, faire `w = w - poids[i-1]` puis `i=i-1`

À retenir

- Brute force : simple, mais peut être trop lent (2^n)
- Programmation dynamique : on résout des **petits problèmes** et on mémorise
- Sac à dos 0/1 : table $F(i, w)$ remplie ligne par ligne