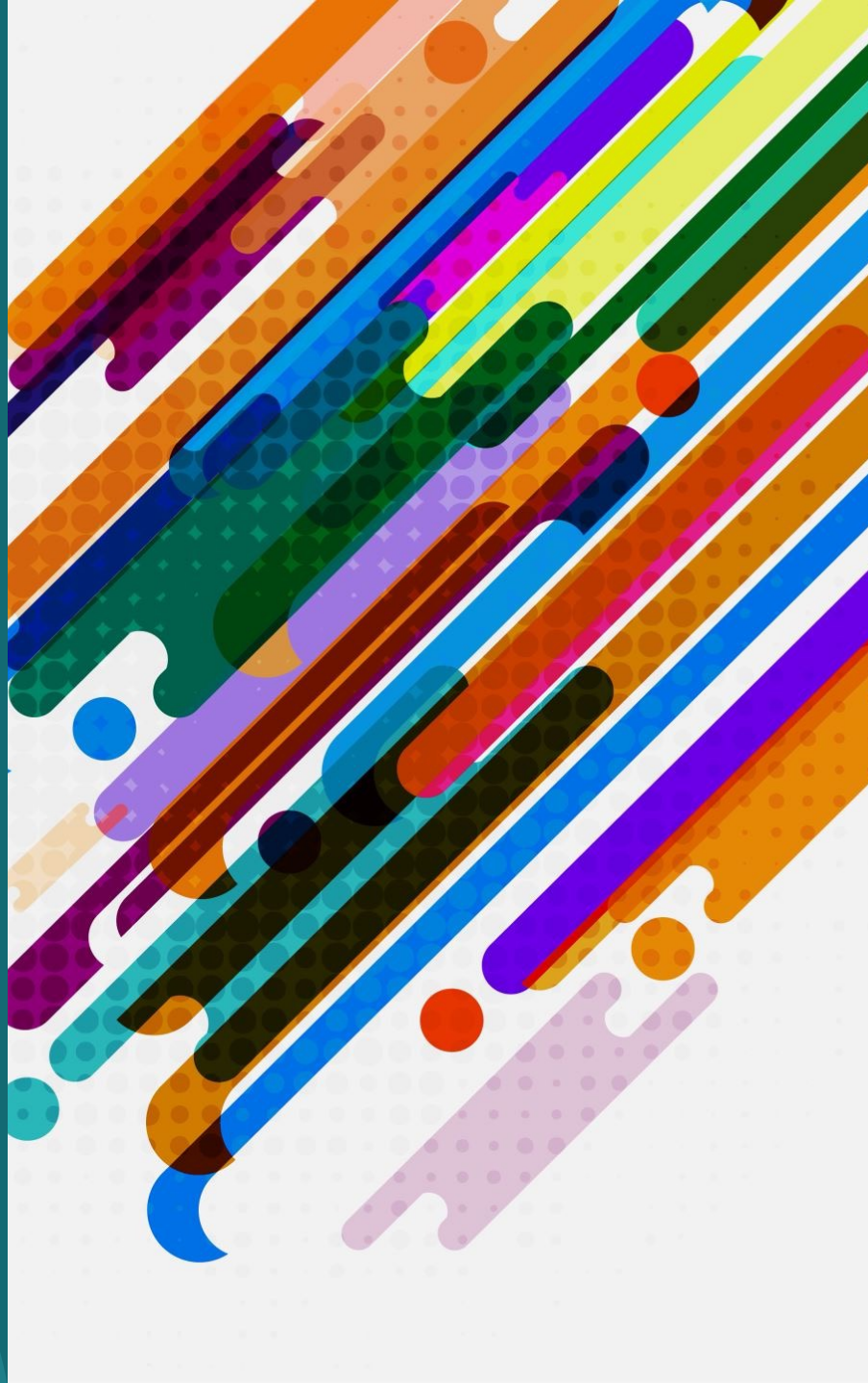
Projet 7 : Résolvez des problèmes en utilisant des algorithmes en Python



Introduction :

Nous devons résoudre un problème knapsack classique en utilisant les deux approches suivantes :

- Un algorithme « Brute force » qui passe sur toutes les données
- Un algorithme optimisé basé sur la programmation dynamique

Détails du problème :

Objectif :

Choisir l'ensemble d'actions le plus rentable parmi une liste en respectant un montant de dépenses maximum défini

Contraintes :

- Chaque action ne peut être achetée qu'une seule fois
- Le portefeuille maximum est fixé à 500€
- Les actions ne sont pas divisibles

Solutions :

- Un algorithme ''Brute-force'' avec une approche naïve
- Un algorithme ''optimisé'' avec une approche en programmation dynamique

Algorithme Brute-Force

- La solution brute force consiste à essayer toutes les combinaisons d'actions possibles et de choisir la plus rentable.
- Cette solution est très gourmande en temps et extrêmement sensible à l'augmentation de N (nombre d'éléments passés à l'algorithme)

Pseudo-code Brute-Force

```
brute_force(actions, W):
        n = len(actions)
        for i in range (n):
                for combination in combinations(actions, i):
                        compare combination rentability
```

Analyse Brute-Force

- **Analyse temporelle :**

L'algorithme Brute-Force itère **n** (nombre d'éléments passés dans l'algorithme) fois sur la fonctionne itertools.combinaison dont la complexité temporelle est **O(n!)**

La complexité temporelle est donc **n*n!** soit **O(n!)** selon la notation BigO

- **Analyse de la mémoire :**

L'algorithme crée n liste de complexité spatiale **O(n)**, sa complexité spatiale est donc de O(n^2).

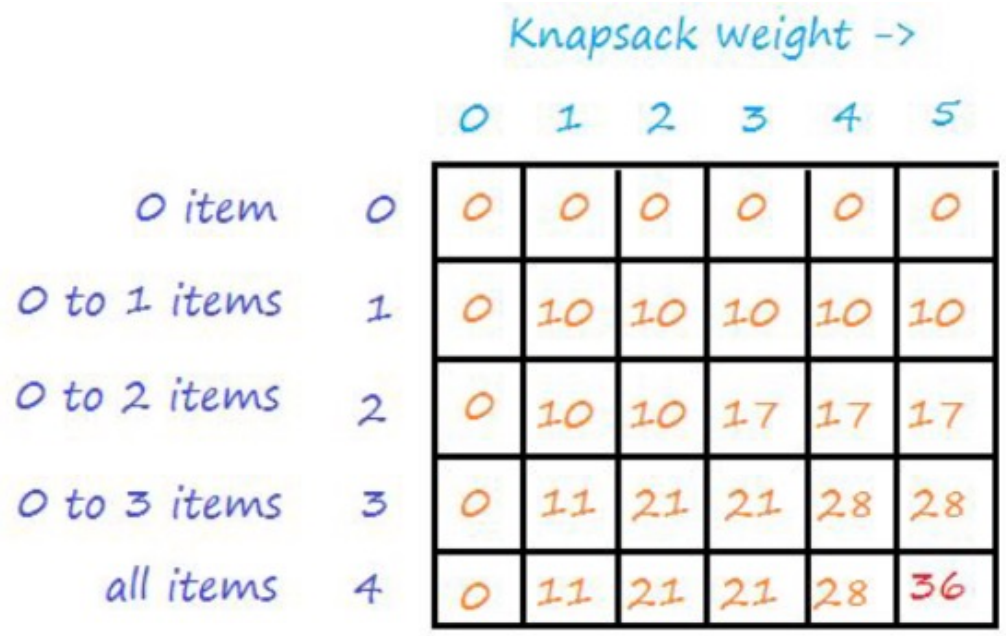
Résultats Brute-Force

```
Actions boughts : ['Action-4', 'Action-5', 'Action-6', 'Action-8', 'Action-10', 'Action-11', 'Action-13', 'Action-18', 'Action-19', 'Action-20']
Total profits : €99.08€
Total cost : 498.0€
Solving time : 1.7706937789916992
```

- Le total des profits est de 99,08€ après deux ans
- Le prix total est de 498€
- L'algorithme met 1,77sec à trouver la solution pour ce premier dataset

Algorithme optimisé

- La solution optimisée s'appuie sur le principe de la **programmation dynamique**.
- L'algorithme va créer une **table** pour stocker la solution de chaque **sous problème** rencontré. Cette table lui permet de ne pas effectuer plusieurs fois les mêmes calculs (ou sous problèmes) et de gagner un temps considérable par rapport à une approche naïve.



Matrice de calcule

Pseudo-code Algorithme optimisé

- Création d'une matrice pour recevoir le résultats des sous problèmes :

```
mat = [[0 for x in range( W + 1 )] for x in range ( n + 1 )]
```

- Peuplement de la matrice en résolvant les sous problèmes :

```
for i in range(n + 1):
    for j in range(W + 1):
        if i == 0 or j == 0:
            mat[i][j] = 0
        elif actions[i - 1].price < j:
            mat[i][j] = max(actions[i - 1].profitability + mat[i - 1][j - actions[i - 1].price], mat[i - 1][j])
        else:
            mat[i][j] = mat[i - 1][j]
```

Analyse algorithme optimisé

- **Analyse temporelle :**

L'algorithme optimisé crée une matrice (ou table) de n (nombre d'items passés en paramètres) par w (capacité, ici appelée portefeuille). Il itère ensuite sur toutes les cellules de cette matrice (soit n*w cellules).

La complexité temporelle est donc O(n*w) selon la notation BigO

- **Analyse de la mémoire :**

L'algorithme crée une matrice en deux dimensions de taille n*w. Sa complexité spatiale est donc O(n*w).

Comparaison brute-force/optimisé

- Sur le dataset 0, les deux algorithmes fournissent les valeurs suivantes :

Brute force :

Prix total : 498,0€
Bénéfices après deux ans : 99,08€

Temps d'exécution ≃ 1,9sec

Algorithme optimisé:

Prix total : 498,0€
Bénéfices après deux ans : 99,08€

Temps d'exécution ≃ 0,7sec

- On observe que les deux algorithmes trouvent les mêmes résultats mais le second est presque 3 fois plus rapide sur ce dataset.

Comparaison avec Sienna Dataset 1

Mes résultats

Actions achetées : (Share-KMTG), (Share-GHIZ), (Share-NHWA), (Share-UEZB), (Share-LPDM), (Share-MTLR), (Share-USSR), (Share-GTQK), (Share-FKJW), (Share-MLGM), (Share-QLMK), (Share-WPLI), (Share-LGWG), (Share-ZSDE), (Share-SKKC), (Share-QQTU), (Share-GIAJ), (Share-XJMO), (Share-LRBZ), (Share-KZBL), (Share-EMOV), (Share-IFCP)

Prix total : 499,95€
Bénéfices après deux ans : 198,54€

Résultats de Sienna

Actions achetées : Share-GRUT

Prix total : 498.76€
Bénéfices après deux ans : 196.61€

Cette comparaison n'est pas valable car les données fournies par Sienna ne sont pas cohérentes
Et donc pas utilisables.

Comparaison avec Sienna Dataset 2

Mes résultats

Actions achetées : Share-ECAQ, Share-IXCI, Share-FWBE, Share-ZOFA, Share-PLLK, Share-YNGA, Share-ANFX, Share-PATS, Share-PUCI, Share-VCXT, Share-NDKR, Share-ALIY, Share-JWGF, Share-MPJI, Share-CDAN, Share-JGTW, Share-QCTW, Share-AMXX, Share-FAPS, Share-LFXB, Share-DWSK, Share-JVCL, Share-XQII, Share-PVHB, Share-ROOM

Prix total : 499.99€
Bénéfices après deux ans : 198.25€

Résultats de Sienna

Actions achetées : Share-ECAQ, Share-IXCI, Share-FWBE, Share-ZOFA, Share-PLLK, Share-YFVZ, Share-ANFX, Share-PATS, Share-NDKR, Share-ALIY, Share-JWGF, Share-JGTW, Share-FAPS, Share-VCAX, Share-LFXB, Share-DWSK, Share-XQII, Share-ROOM

Prix total : 489.24€
Bénéfices après deux ans : 193.78€

En jaune des deux côtés sont les actions que l'autre n'a pas achetées. En retraitant les données du Dataset 2 pour convertir les prix négatifs en positifs, mon algorithme a exploité plus d'actions que celui de Senna.