PROJET 7: RÉSOLVEZ DES PROBLÈMES EN UTILISANT DES ALGORITHMES EN PYTHON

PARTIE 2

Réalisé par : Sabah EL-AOUNI

Parcours: Développeur d'application python

ANALYSE DE L'ALGORITHME FORCE BRUTE

- Objectif de l'algorithme : Suggérer une liste des actions les plus rentables à acheter pour maximiser le profit d'un client au bout de deux ans.
- Le programme se base sur des données d'un document CSV dont les colonnes sont les suivantes : ***Actions #*** | ***Coût par action (en euros)*** | ***Bénéfice (après 2 ans)***
- Le programmes est constitué principalement de 3 fonctions :

Fonction	Description	Return
calculate_profit(combinaison)	Calcul du profit en pourcentage pour chaque combinaison : **Cout * Bénéfice**	Profit
calculate_cost(combinaison)	Calcul de la **sommes des couts** cela pour vérifier dans la fonction qui suit (best_combination) que le montant max d'investissement de 500 euro n'est pas dépassé.	Somme des couts
best_combination(data_list)	Elle prend comme paramètre les données CSV (mises sous forme de liste), puis à l'aide de la méthode combinations de Itertools et des deux fonctions précédentes, on va parcourir toutes les combinaisons possibles et faire le calcul nécessaire afin d'afficher **le meilleur investissement.**	Affiche la liste des actions les plus rentables

PSEUDOCODE DU PROCESSUS DE RÉFLEXION DE LA SOLUTION OPTIMISÉE

```
Algorithme optimized

Début

-- création d'une matrice



Fin
```

L'ALGORITHME CHOISI POUR LA VERSION OPTIMISÉE

- Ajouter texte
- Limite de l'algorithme

COMPARAISON DE L'EFFICACITÉ ET DES PERFORMANCES DES DEUX ALGORITHMES 1/2

```
def best_combination(data_list):
    profit = 0 <-- une affectation
    best_combinaison = [ ] <-- une 2eme affectation

    for i in tqdm(range(20)): <-- utilisation de boucle + range
        combs = combinations(data_list, i+1)

        for comb in combs: <-- 2eme boucle
            total_cost = calculate_cost(comb)  <-- n appels de fonction
                                                   + n affectations


            if total_cost <= Max_invest:  <-- n Comparaisons
                total_profit = calculate_profit(comb)  <-- n appels de
                                                           fonction + n
                                                           affectation
                if total_profit > profit:    <-- n comparaisons
                 profit = total_profit <-- n nouvelles affectations
                 best_combinaison = comb<-- n nouvelles affectations
    print("Liste des actions les plus rentables : " ,best_combinaison)
```

La notation Big-O:

• Il est à noter que la vitesse de l'algorithme en secondes n'est pas un critère de performance. Celle ci est **mesurée par le taux de croissance de l'algorithme c'est-à-dire le nombre d'opérations qu'il faut pour terminer.**

•Evaluation du nombre d'opérations nécessaires que la machine va devoir faire pour résoudre le problème. dans la solution forcebrute il y a beaucoup d'opérations.

• Prenons l'exemple de la fonction best_combination() :

<---------

COMPARAISON DE L'EFFICACITÉ ET DES PERFORMANCES DES DEUX ALGORITHMES 2/2

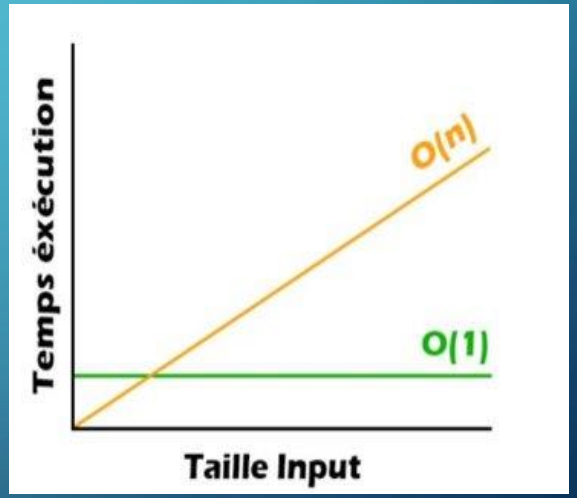
- Comme résultat l'algorithme force brute est inutilisable dans le cas de1000 données ; l'exécution du programme n'avance plus pour arriver au résultat comme dans le cas du premier CVS (avec 20 données)

- Les opérations nécessaires pour l'algorithme vont être multipliées par la grandeur de l'input qui sont les données CSV

  => La fonction best_combination() a donc une tendance linéaire de O(n).

- Avec la solution optimisée, les opérations pour résoudre leproblème ne vont pas augmenter en même temps que l'input . 20 ou 1000 données, l'algorithme va fonctionner avec la même performance. **La fonction**

  => La fonction optimisée a donc une tendance constante de O(1).



- Quelles sont les limites de ton algorithme. Où est-ce que ça bloque?
- Par exemple si j'augmente le prix max? Si je donne un prix max avec une précision plus élevée que le centime (e.g cryptomonnaie)? Si je rajoute des actions, jusqu'à quel point il tiendra la charge (par exemple, l'algorithme de force brute s'écroule rapidement) ? Si je donne des valeurs extrêmes à mes actions, est-ce que je peux faire flancher ton algorithme?

- Bref tout ce qui peut "casser" ton programme.

- Pour la complexité temporelle il s'agit d'étudier le nombre d'opérations en fonction du nombre d'actions.

- Par exemple si tu renvoies systématiquement la première action de ta liste, tu as une complexité constante O(1). Evidemment ce n'est pas un algorithme viable (ai-je besoin d'expliquer pourquoi?)
- Si tu regardes tous les éléments de ta liste, et que tu renvoies juste celui avec le meilleur rendement, tu as une complexité linéaire O(n) (tu dois parcourir toute ta liste qui a n éléments). Encore une fois évidemment ce n'est pas ton algorithme.
- Pour l'algorithme de force brute, pour chaque action tu as deux possibilités (tu prends ou tu ne prends pas), la complexité va donc être O(2^n), i.e tu fais deux fois plus de travail quand tu rajoutes une action.

- A toi de voir ce que fait ton algorithme.

- Pour la complexité spatiale, au lieu de regarder le nombre d'opérations, tu regardes le nombre de variables différentes dont tu as besoin simultanément.
- Par exemple étant donné que tu as une liste qui contient tes actions, ta complexité spatiale est a minima O(n). Selon ton algorithme tu peux avoir une plus grande complexité spatiale.