Parcours: Développeur d'application python

OPENCLASSROOMS

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

Réalisé par : **Sabah EL-AOUNI**

Mentor : **Idriss Ben Geloune**

PLAN DE LA PRÉSENTATION

- Analyse de l'algorithme de force brute
- Pseudocode de la solution optimisée
- Algorithme choisi et ses limites
- Analyse des performances et de l'efficacité des deux algorithmes
- Comparaison des résultats avec les choix de Sienna
- Conclusion

ANALYSE DE L'ALGORITHME FORCE BRUTE 1/2 :

- **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 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

ANALYSE DE L'ALGORITHME FORCE BRUTE 2/2

- Le programme se base sur des données d'un document CSV de 20 données dont les colonnes sont les suivantes : ***Actions #*** | ***Coût par action (en euros)*** | ***Bénéfice (après 2 ans)***
- ***<u>En résumé:</u>*** La solution Force brute donne la meilleure solution en terme de profit , puisque elle parcourt et traite toutes les combinaison possibles , on peut dire donc que d'un point de vue mathématique que c'est la meilleure solution , mais ce n'est pas une solution optimale au sens algorithmique dans le cas de traitement d'un grand nombre de données

L'ALGORITHME CHOISI POUR LA VERSION OPTIMISÉE

- L'algorithme choisie pour la solution optimisée est celui du problème du sac à dos , qui est basé sur les 4 élements ci-dessous:

  Valeurs -- qui correspond au profit

  Poids -- qui correspond au cout

  Nombre d'éléments distincts -- qui correspond au nombre d'action

  Capacité -- qui correspond au max investissement (500 euro)

PSEUDOCODE DU PROCESSUS DE RÉFLEXION DE LA SOLUTION OPTIMISÉE

Algorithme optimized:

-- création d'une matrice initialisée à 0: créer un tableau de largeur 500 (investissement max) et de hauteur égal au nombre d'actions

-- Parcours des données pour une hauteur égal au nombre d'actions:

-- Pour chaque élèment de la boucle Parcourir les données pour une largeur de 500:

-- Si le cout de l'élèment courant est inférieur à l'investissement max:

-- comparaison du profit en cours avec le bénéfice possible précèdent pour le même investissement

-- Ajout dans la matrice du meilleur résultat = max entre

(Profit de l'élément en cours + solution optimisée de la ligne d'avant qui a un cout qui correspond à la différence de poids obtenue entre le max invest et le cout de l'élément en cours)

et (la solution optimisée de la ligne d'avant pour le même investissement)

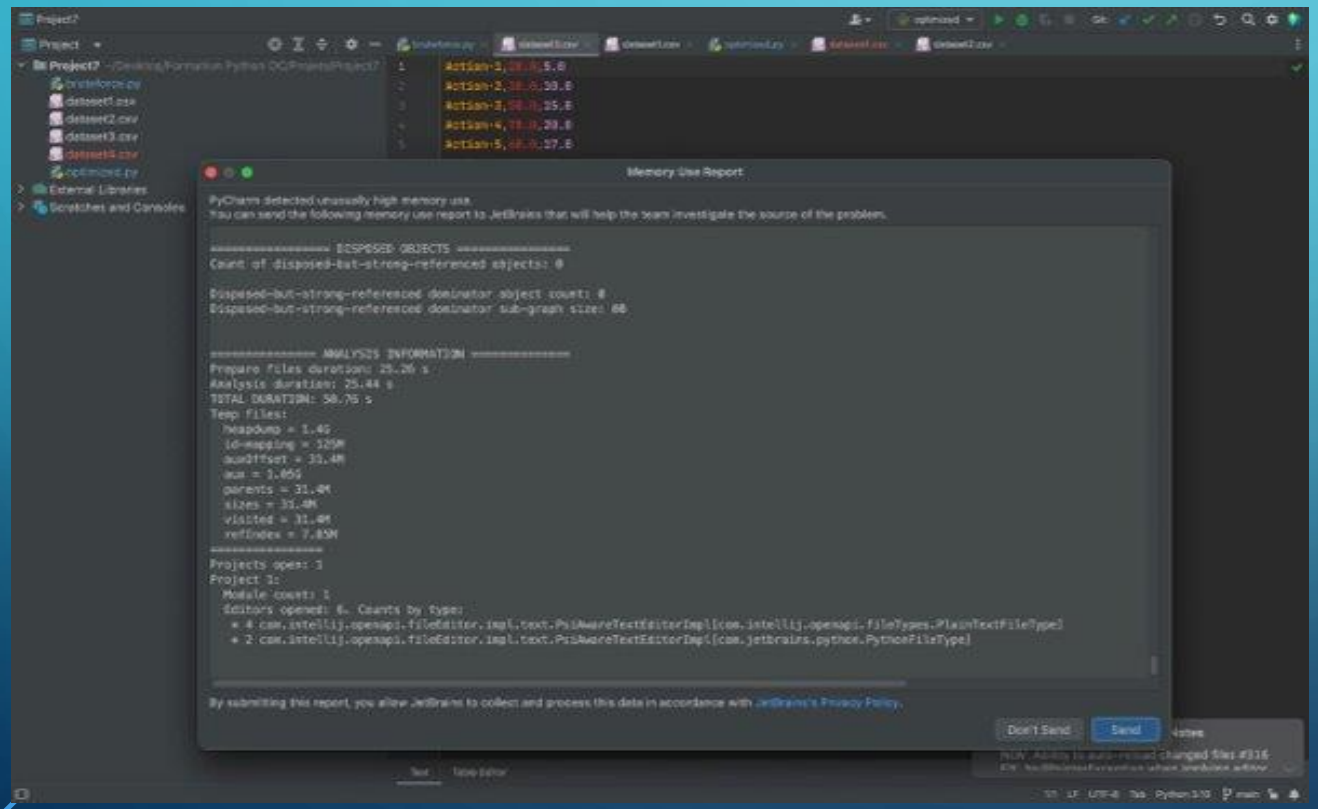
-- Sinon si l'investissement max est dépassé:

-- prendre la solution optimisée de la ligne d'avant

LIMITES CONSTATÉES DE L'ALGORITHME OPTIMISÉ:

- En cas d'augmentation du max invest ou

- En cas d'augmentation du nombre de données cela nécessite un temps important ainsi qu 'une utilisation énorme des ressources de la machine.



- Rapport : L'IDE a détecté une utlilsation importante de la mémoire

Nombre d'actions	Temps d'exécution
1497	0.52S
3000	0,7s
8982	3.31s
17964	7.10s
35928	14.50s
71856	29.08s
215568	84.15s

Evolution du temps d'exécution en fonction de l'évolution des données traitées:

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:

- Definition: La notation Big O est une notation qui indique comment les performances d'un algorithme se comporteront si l'entrée de données augmente.

• 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 force brute il y a beaucoup d'opérations à faire.

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

ANALYSE DE LA MÉMOIRE

- La Sollicitation de la mémoire augmente en fonction de l'évolution des données:

Alogorithme	Donnée	Mémoire
Brute force	20 données	880
Optimisé	20 données	500
Optimisé	956 (Dataset1)	500
Optimisé	541 (Dataset2)	500
Optimisé	Test dur 9560 données	900

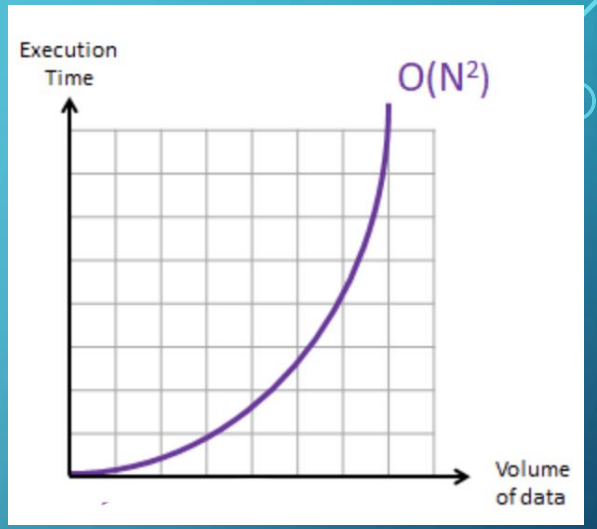
COMPARAISON DE L'EFFICACITÉ ET DES PERFORMANCES DES DEUX ALGORITHMES 2/2

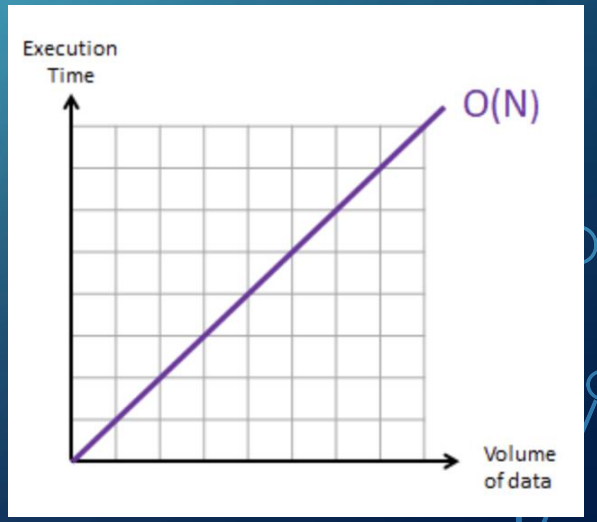
- Comme résultat l'algorithme force brute est inutilisable dans le cas de1000 données ; l'exécution du programme n'avance plus pour arriver au résultat comme dans le cas du premier CVS (avec 20 données)
- Les opérations nécessaires pour l'algorithme vont être multipliées par la grandeur de l'input qui sont les données CSV

=> **La fonction best_combination() a donc une tendance** exponentielle **de O(n^2).**

- Avec la solution optimisée, qui ne parcourent pas tous les cas possibles mais qui donnent une réponse optimisée, les opérations pour résoudre le problème ne vont pas augmenter en même temps que l'input . 20 ou 1000 données, l'algorithme va fonctionner avec la même performance.

=> **La fonction optim**isée a donc une tendance linéaire de O(n x W) ce qui équivaut à O(n),





Source: www.101computing.net

COMPARAISON AVEC LES RÉSULTATS DE SIENNA POUR LE DATASET1

Résultat Sienna

```
Sienna bought:

Share-GRUT

Total cost: 498.76â,¬
Total return: 196.61â,¬
```

L'Optimisé

```
-- Actions les plus rentables :

('Share-GRUT', 498.76, 196.61119200000002)
('Share-HITN', 0.67, 0.22438300000000003)
-- le profit est de :  196.835575
-- Total investissement :  499.43
```

Analyse

Le résultat de l'algorithme optimisé sont meilleurs de 0,22 euro

COMPARAISON AVEC LES RÉSULTATS DE SIENNA POUR LE DATASET2

Résultat Sienna

```
Sienna bought:
Share-ECAQ 3166
Share-IXCI 2632
Share-FWBE 1830
Share-ZOFA 2532
Share-PLLK 1994
Share-YFVZ 2255
Share-ANFX 3854
Share-PATS 2770
Share-NDKR 3306
Share-ALIY 2908
Share-JWGF 4869
Share-JGTW 3529
Share-FAPS 3257
Share-VCAX 2742
Share-LFXB 1483
Share-DWSK 2949
Share-XQII 1342
Share-ROOM 1506

Total cost: 489.24â,¬
Profit: 193.78â,¬
```

Mes Résultats

```
-- Actions les plus rentables :

('Share-ECAQ', 31.66, 12.502534)
('Share-PLLK', 19.94, 7.958054)
('Share-OPBR', 39.0, 15.190500000000002)
('Share-ANFX', 38.55, 15.312059999999999)
('Share-IJFT', 40.91, 15.909898999999998)
('Share-IFCZ', 35.97, 13.298109)
('Share-GHIS', 22.48, 7.90172)
('Share-XQII', 13.42, 5.302242)
('Share-EBJX', 34.76, 12.426699999999999)
('Share-UBWD', 26.4, 9.253199999999998)
('Share-FFZA', 4.57, 1.57665)
('Share-ROOM', 15.06, 5.908038)
('Share-YCGH', 24.45, 9.435255000000002)
('Share-WERP', 24.67, 8.671505000000002)
('Share-ENZZ', 37.24, 14.221956)
('Share-MHUQ', 29.91, 11.332899)
('Share-AJGB', 28.6, 10.4247)
('Share-GBGY', 27.08, 9.231572)
-- le profit est de :  194.900973
-- Total investissement :  494.66999999999996
```

Analyse

Le résultat de l'algorithme optimisé sont meilleurs de 1,12 euro

COMPARAISON AVEC LES RÉSULTATS DE SIENNA – ANALYSE GLOBALE ET EXPLORATION DES ENSEMBLES DES DONNÉES

La base de la comparison avec les résultats de Sienna :

- le montant de d'investissement
- Le profit
- Les actions les plus rentables

Les informations manquantes:

- l'algorithme suivi
- le temps d'exécution

Analyse globale en se basant sur les données fournies:

- Avec l'algorithme optimisé on pourra investir davantage

Nettoyage adopté pour les données dataset:

- Vu que les données contenaient des erreurs on a retenu que certaines actions parmi les 1000 actions proposée:

Fichier	Nbr données retenues
Dataset1	956
Dataset2	541

- Type de nettoyage : Élimination des éléments avec cout ou profit nuls o négatifs

CONCLUSION

- **En résumé :** l'algorithme Force brute fournit la solution idéale et précise, mais c'est un algorithme optimisé qui fournira la solution optimale en tenant compte des contraintes de mémoire et de temps.
- Il est important de prendre en compte ce concept dans l'optimisation de son code. cela permet de détecter rapidement et efficacement les problèmes de lenteur d'exécution et optimiser le bouts de code en relation.
- Ce projet m'a permis de découvrir et d'appréhender la programmation dynamique. Cette nouvelle compréhension de fond a changé complètement la façon dont j'abordais la programmation, il s'agit d'une compétence à raffiner et à améliorer au fur et à mesure de l'évolution de sa carrière.

SOURCES UTILISÉES

www.stackoverflow.com

www.jesuisundev.com

www.101computing.net