# Élaboration d'algorithmes favorisant la maximisation des profits.

## Structure du code : bruteforce.py

Une image contenant texte, diagramme, croquis, dessin

Description générée automatiquement

Tout d'abord, notre objectif est d'obtenir la liste des différentes combinaisons pour un ensemble d'actions donné.

Pour cela, nous utilisons la fonction "create\_set" qui retourne une liste contenant les différentes combinaisons. Voici quelques exemples avec 2 et 3 actions :

Si le nombre est égal à 2, en utilisant la fonction "create\_set(number)", nous obtenons une liste de jeux de données : [[00], [01], [10], [11]].

Si le nombre est égal à 3, en utilisant la fonction "create\_set(number)", nous obtenons une liste de jeux de données : [[000], [001], [010], [011], [100], [101], [110], [111]].

Il est nécessaire de créer 2n + 1 éléments, ce qui est équivalent à O(2n) en termes de complexité algorithmique.

Il est nécessaire de conserver en mémoire une liste contenant 2n + 1 éléments.

Si n est égal à 20, alors le nombre de combinaisons possibles serait de 2^20 + 1, ce qui équivaut à 1 048 577 combinaisons.

Pour chaque combinaison, on souhaite déterminer le coût d'investissement et le bénéfice associé. Pour cela, nous utilisons la fonction "get\_set\_of\_max\_value(list\_of\_set)" qui parcourt toutes les combinaisons et conserve en mémoire le meilleur ensemble (celui ayant le bénéfice maximal pour un coût donné maximum).

Pour chaque combinaison, la fonction "get\_cost\_and\_value(set)" est utilisée pour calculer le coût d'investissement et le bénéfice associé.

Voici un exemple de l'utilisation de la fonction "get\_set\_of\_max\_value(list\_of\_set)" avec deux actions :

list\_of\_set = [[00], [01], [10], [11]]

best\_set = get\_set\_of\_max\_value(list\_of\_set)

La fonction "get\_set\_of\_max\_value" parcourt chaque ensemble dans la liste **list\_of\_set** et détermine le meilleur ensemble en termes de bénéfice pour un coût maximum donné. Dans cet exemple, la fonction sélectionnera l'ensemble qui offre le bénéfice maximal en considérant le coût d'investissement.

Après l'exécution de la fonction, la variable **best\_set** contiendra le meilleur ensemble trouvé.

Voici un exemple de dataset avec deux actions et un coût maximal de 20€ :

Action 1 : Coût = 7€, Bénéfice = 5€ Action 2 : Coût = 14€, Bénéfice = 10€

Pour trouver le meilleur investissement avec un coût inférieur à 20€, nous utilisons les fonctions "get\_set\_of\_max\_value(list\_of\_set)" et "get\_cost\_and\_value(set)".

Dans cet exemple, la fonction "get\_set\_of\_max\_value" sélectionnera l'ensemble [1, 1] (correspondant à Action 2) car il offre un bénéfice de 10€ (le plus élevé) pour un coût total de 14€ (inférieur à 20€).

La fonction "get\_cost\_and\_value" calcule ensuite le coût d'investissement et le bénéfice associé à l'ensemble sélectionné.

Enfin, le résultat affiché sera "Meilleur investissement : (14, 10)", indiquant un coût d'investissement de 14€ et un bénéfice de 10€ pour le meilleur ensemble.

## Structure du code : optimized-glouton.py

Une image contenant diagramme, ligne, origami, conception

Description générée automatiquement

Nous souhaitons investir dans les actions les plus rentables. Voici les étapes à suivre :

Tout d'abord, nous souhaitons connaître la rentabilité de chaque action. Voici un exemple :

Action 1 :

Coût = 7€

Bénéfice = 4€

% de bénéfice = 57,14%

Action 2 :

Coût = 14€

Bénéfice = 10€

% de bénéfice = 71,43%

Action 3 :

Coût = 5€

Bénéfice = 3€

% de bénéfice = 60%

Ensuite, nous trions ces actions de la plus rentable à la moins rentable en utilisant la fonction "sort\_actions\_by\_profit\_percentage(actions)" qui utilise la fonction "sorted".

La fonction "sort\_actions\_by\_profit\_percentage(actions)" effectue le tri en fonction du pourcentage de bénéfice de chaque action. Par exemple, si nous avons les actions dans l'ordre [action 1, action 2, action 3], la fonction les réorganisera dans l'ordre [action 2, action 3, action 1].

Pour réaliser ce tri, nous devons créer une liste avec la fonction "sorted", ce qui nécessite un nombre d'éléments égal à n x log(n), soit une complexité de O(N log N).

Enfin, il est nécessaire de conserver en mémoire une liste contenant n éléments, où n est le nombre d'actions.

Nous souhaitons investir dans les actions les plus rentables tout en respectant un budget maximum. Pour cela, nous utilisons la fonction "get\_invest(sorted\_actions)" pour obtenir les actions à sélectionner.

Pour calculer les différentes étapes, commençons par trier les actions par ordre décroissant de taux de bénéfice :

Action 2 : Coût = 14€, Bénéfice = 10€, Taux de bénéfice = 71,43% Action 3 : Coût = 5€, Bénéfice = 3€, Taux de bénéfice = 60% Action 1 : Coût = 7€, Bénéfice = 4€, Taux de bénéfice = 57,14%

Maintenant, nous allons sélectionner les actions dans l'ordre jusqu'à ce que le coût total atteigne ou dépasse le budget maximum de 20€.

Étape 1 : Sélection de l'action 2

Coût total : 14€

Bénéfice total : 10€

Étape 2 : Sélection de l'action 3

Coût total : 19€ (14€ + 5€)

Bénéfice total : 13€ (10€ + 3€)

À ce stade, si nous sélectionnons l'action 1, le coût total dépassera le budget maximum. Donc, nous arrêtons ici.

Résultat final :

Coût total : 19€

Bénéfice total : 13€

La fonction "get\_invest(sorted\_actions)" dans ce cas renvoie : Coût = 19€ Bénéfice = 13€

# Structure du code : optimized-dynamic.py

Une image contenant diagramme, croquis, dessin, origami

Description générée automatiquement

Pour résoudre le problème, nous utilisons une approche basée sur deux tableaux de taille similaire. Le premier tableau, de dimension 2 x (Nactions + 1) x (maxcost + 1), est utilisé pour stocker les bénéfices maximums pour chaque combinaison de coût et d'action. Chaque cellule du tableau représente le bénéfice maximum obtenu pour un certain coût et une certaine action.

Le deuxième tableau, également de dimension 2 x (Nactions + 1) x (maxcost + 1), est utilisé pour garder en mémoire quelles actions sont retenues. Chaque cellule du tableau indique si une action est retenue (valeur 1) ou exclue (valeur 0) pour une certaine combinaison de coût et d'action.

Prenons un exemple pour illustrer le processus. Supposons que nous ayons les actions suivantes :

|  |  |  |
| --- | --- | --- |
| **Action** | **Coût** | **Bénéfice** |
| 1 | 3 | 2 |
| 2 | 2 | 1 |
| 3 | 5 | 4 |

Et supposons que le coût maximum soit de 5€.

En utilisant les deux tableaux, nous parcourons chaque cellule du tableau des bénéfices maximums en calculant si l'inclusion ou l'exclusion de l'action que nous testons est plus rentable. Nous sélectionnons le bénéfice maximum entre l'inclusion et l'exclusion pour chaque cellule, ce qui nous permet de remplir le tableau des bénéfices maximums.

En parallèle, nous gardons en mémoire dans le tableau des actions retenues quelles actions sont retenues ou exclues pour chaque combinaison de coût et d'action.

À la fin du processus, le bénéfice maximum obtenu se trouve dans la dernière cellule du tableau des bénéfices maximums pour le coût maximum. En utilisant le tableau des actions retenues, nous pouvons retrouver quelles actions ont été retenues pour obtenir ce bénéfice maximum.

Par exemple, en utilisant les données ci-dessus, le tableau des bénéfices maximums ressemblerait à ceci :

Tableau des bénéfices maximums :

0 1 2 3 4 5

0 [0, 0, 0, 0, 0, 0]

1 [0, 0, 0, 2, 2, 2]

2 [0, 0, 1, 2, 3, 3]

3 [0, 0, 1, 2, 3, 4]

Et le tableau des actions retenues ressemblerait à ceci :

Tableau des actions retenues :

0 1 2 3 4 5

[0, 0, 0, 0] [0, 0, 0, 0] [0, 0, 0, 0] [0, 1, 1, 0] [0, 1, 1, 0] [0, 1, 1, 0]

[0, 0, 0, 0] [0, 0, 0, 0] [0, 1, 1, 0] [0, 1, 1, 0] [0, 1, 1, 0] [0, 1, 1, 0]

[0, 0, 0, 0] [0, 0, 1, 0] [0, 1, 1, 0] [0, 1, 1, 0] [0, 1, 1, 0] [0, 1, 1, 0]

En utilisant ces tableaux, vous pouvez déterminer quelles actions sont retenues pour obtenir le bénéfice maximum en respectant les contraintes de coût.

Pour déterminer les actions qui sont conservées pour obtenir le bénéfice maximum tout en respectant les contraintes de coût, vous pouvez remonter le tableau des actions retenues.

En partant de la dernière cellule du "Tableau des actions retenues" qui correspond au coût maximum, vous pouvez suivre le chemin jusqu'à la première cellule (0, 0, 0). À chaque étape, si la valeur dans la cellule actuelle est de 1, cela signifie que l'action correspondante est conservée pour le coût donné.

En suivant ce processus pour l'exemple donné, nous pouvons retrouver les actions qui sont conservées pour obtenir le bénéfice maximum :

Pour un coût de 5€ :

L'action 1 est exclue.

L'action 2 est incluse.

L'action 3 est exclue.

Pour un coût de 4€ :

L'action 1 est exclue.

L'action 2 est incluse.

L'action 3 est exclue.

Pour un coût de 3€ :

L'action 1 est exclue.

L'action 2 est incluse.

L'action 3 est exclue.

Ainsi, pour un coût maximum de 5€, les actions conservées pour obtenir le bénéfice maximum sont l'Action 2.

En résumé, en utilisant les deux tableaux, nous pouvons calculer les bénéfices maximums pour chaque combinaison de coût et d'action. De plus, le tableau des actions retenues nous permet de déterminer quelles actions inclure ou exclure pour atteindre le bénéfice maximum dans les contraintes de coût données.

# backtesting.py

Exploration de données

Pour le dataset1 :

"Sienna" a un coût total de 498,76€ et un bénéfice de 196,61€.

"optimized-glouton" a un coût total de 499,93€ et un bénéfice de 198,49€.

"optimized-dynamic" a un coût total de 499,94€ et un bénéfice de 198,53€.

Pour le dataset2 :

"Sienna" a un coût total de 489,24€ et un bénéfice de 193,78€.

"optimized-glouton" a un coût total de 499,7€ et un bénéfice de 197,72€.

"optimized-dynamic" a un coût total de 499,90€ et un bénéfice de 197,95€.

En comparant les résultats, nous pouvons noter ce qui suit :

Coûts totaux : "optimized-glouton" et "optimized-dynamic" ont des coûts totaux légèrement supérieurs à ceux de "Sienna" pour les deux jeux de données.

Bénéfices : "optimized-glouton" et "optimized-dynamic" ont des bénéfices légèrement supérieurs à ceux de "Sienna" pour les deux jeux de données.

Sur la base de ces résultats, il semble que les méthodes "optimized-glouton" et "optimized-dynamic" obtiennent des résultats légèrement meilleurs en termes de bénéfices par rapport à la méthode "Sienna".

# Conclusion :

En conclusion, nous pouvons constater les différences clés entre les algorithmes "optimized-glouton" et "optimized-dynamic" :

Mémoire :

L'algorithme "optimized-glouton" utilise une liste de n éléments, ce qui correspond à une complexité en espace de O(N).

L'algorithme "optimized-dynamic" utilise deux listes de n x Maxcost éléments, ce qui correspond à une complexité en espace de O(N x Maxcost).

Efficacité :

L'algorithme "optimized-glouton" a une complexité en temps de O(N log N).

L'algorithme "optimized-dynamic" a une complexité en temps de O(N x Maxcost).

Exploration des données :

L'algorithme "optimized-dynamic" maximise les profits par rapport aux résultats antérieurs. Cela signifie qu'il utilise une approche dynamique qui prend en compte les résultats précédents pour maximiser les bénéfices.

En termes de mémoire, l'algorithme "optimized-dynamic" a une complexité plus élevée en raison de l'utilisation de deux listes pour stocker les résultats. Cependant, il offre une exploration plus précise des données et vise à maximiser les profits par rapport aux résultats antérieurs.

L'algorithme "optimized-glouton", quant à lui, utilise moins de mémoire mais peut ne pas être aussi précis ou optimal que l'algorithme "optimized-dynamic".

Il est important de noter que le choix de l'algorithme dépend des contraintes spécifiques du problème, notamment des ressources disponibles (mémoire, temps de calcul) et des objectifs souhaités (maximisation des profits, efficacité, etc.).