Analyse de l'algorithme Force brute

L'algorithme *Force brut* prend un fichier CSV contenant des données sur des actions financières avec leurs noms, prix et gains potentiels. Il vise à trouver la combinaison optimale d'actions à acheter avec une dépense maximale de 500 dollars, de manière à maximiser le profit.

Voici une analyse étape par étape de l'algorithme :

1. **Le code commence par importer les modules nécessaires : itertools.combinations pour générer toutes les combinaisons d'actions possibles et csv pour lire les données depuis un fichier CSV.**
2. **La variable MAX_SPEND est définie à 500, indiquant la dépense maximale autorisée.**
3. **La fonction get_data(path) est définie pour lire les données à partir du fichier CSV donné par path. Les données de chaque action sont stockées sous forme de tuples dans la liste all_shares, contenant le nom de l'action, son prix et son gain potentiel.**
4. **La fonction get_best_combination(path) est définie pour trouver la combinaison optimale d'actions en fonction de la dépense maximale et du profit potentiel.**
5. **La fonction appelle get_data(path) pour obtenir toutes les données sur les actions du fichier CSV.**
6. **Ensuite, elle initialise certaines variables pour garder une trace de la meilleure combinaison trouvée et le profit associé.**
7. **En utilisant une boucle for, l'algorithme génère toutes les combinaisons possibles d'actions en utilisant la fonction combinations de itertools. La variable all_combination stocke toutes les combinaisons possibles.**
8. **Ensuite, l'algorithme itère sur chaque combinaison et calcule le prix total (spend) et le profit total (profit) de cette combinaison.**
9. **Si le coût total (spend) est inférieur ou égal à la dépense maximale autorisée (MAX_SPEND) et que le profit total (profit) est supérieur ou égal au profit final trouvé jusqu'à présent, alors cette combinaison devient la nouvelle meilleure combinaison. Le profit final et le coût de cette combinaison sont également mis à jour en conséquence.**
10. **Une fois toutes les combinaisons testées, l'algorithme imprime la meilleure combinaison d'actions à acheter, la dépense totale associée et le profit total attendu.**
11. **Enfin, l'algorithme appelle la fonction get_best_combination("data/dataset4_Python+P7.csv") pour exécuter l'analyse sur le fichier CSV "dataset4_Python+P7.csv".**

Remarque : L'algorithme utilise une approche par force brute pour générer toutes les combinaisons possibles d'actions, ce qui peut être inefficace pour de grands ensembles de données. Pour des ensembles de données plus importants, des techniques d'optimisation plus avancées, telles que la programmation dynamique, pourraient être utilisées pour améliorer les performances de l'algorithme.

Pseudocode décrivant le processus de réflexion sous-tendant la solution optimisée

```
FONCTION get_data(path):
    all_shares <- liste vide
    OUVRIR fichier en lecture (path)
    csvreader <- CREER lecteur CSV avec fichier
    IGNORER première ligne (entêtes)
    POUR CHAQUE ligne DANS csvreader FAIRE:
        AJOUTER (nom_action, prix, gain) DANS all_shares
    RETOURNER all_shares

FONCTION get_best_combination(path):
    all_shares <- APPELER get_data(path)
    TRIER all_shares PAR gain DESCENDANT
    best_combination <- liste vide
    share_price_list <- liste vide
    share_profit_list <- liste vide
    max_spend <- MAX_SPEND

    POUR CHAQUE action DANS all_shares FAIRE:
        share_name <- NOM de action
        share_price <- PRIX_POSISTIVE de action
        share_profit <- GAIN de action
        profit_by_share <- (share_price * share_profit) / 100

        SI (max_spend - share_price) >=  0 ET SHARE_PRICE  !=  0 ALORS:
            AJOUTER share_name À best_combination
            AJOUTER share_price À share_price_list
            AJOUTER profit_by_share À share_profit_list
            max_spend <- max_spend - share_price

    investement <- SOMME de share_price_list
    final_profit <- SOMME de share_profit_list

    AFFICHER "Meilleure combinaison : ", best_combination
    AFFICHER "Investissement : ", arrondir(investement, 2), "$"
    AFFICHER "Profit : ", arrondir(final_profit, 2), "$"
```

L'algorithme choisi pour la version optimisée :

```
ALGORITHME get_best_combination(path):
    all_shares <- APPELER get_data(path)
    TRIER all_shares PAR gain DESCENDANT
    best_combination <- liste vide
    share_price_list <- liste vide
    share_profit_list <- liste vide
    max_spend <- MAX_SPEND

    POUR CHAQUE action DANS all_shares FAIRE:
        share_name <- NOM de action
        share_price <- PRIX_POSITIVE de action
        share_profit <- GAIN de action
        profit_by_share <- (share_price * share_profit) / 100

        SI (max_spend - share_price) >= 0 ET SHARE_PRICE != 0 ALORS:
            AJOUTER share_name À best_combination
            AJOUTER share_price À share_price_list
            AJOUTER profit_by_share À share_profit_list
            max_spend <- max_spend - share_price

    investement <- SOMME de share_price_list
    final_profit <- SOMME de share_profit_list

    AFFICHER "Meilleure combinaison : ", best_combination
    AFFICHER "Investissement : ", arrondir(investement, 2), "$"
    AFFICHER "Profit : ", arrondir(final_profit, 2), "$"
```

Limites de l'algorithme (cas limites) :

1. **Complexité temporelle : L'algorithme utilise une approche gloutonne pour sélectionner les actions en fonction du gain potentiel décroissant. Cela peut donner de bons résultats dans de nombreux cas, mais dans certains scénarios, cela peut ne pas aboutir à la solution optimale. Par exemple, si le gain potentiel n'est pas strictement positif pour toutes les actions, l'algorithme pourrait mal se comporter.**
2. **Valeurs négatives : L'algorithme ne gère pas les valeurs négatives pour les prix des actions ou les gains potentiels. Si le jeu de données contient des valeurs négatives, cela pourrait entraîner des résultats imprévisibles.**
3. **Dépendance de l'ordre des données : L'algorithme trie les actions en fonction du gain potentiel, ce qui signifie que l'ordre initial des actions peut affecter les résultats. Si les actions ne sont pas triées par le gain potentiel à l'origine, cela pourrait conduire à des résultats différents.**
4. **Dépassement de la limite de dépense : Bien que l'algorithme vérifie si l'ajout d'une action dépasse la limite de dépense maximale, il ne gère pas les cas où le coût d'une seule action est supérieur à la limite de dépense. Cela pourrait conduire à une meilleure combinaison vide si aucune combinaison d'actions individuelles n'est possible dans les limites de dépense.**
5. **Résultats non optimaux : Comme l'algorithme utilise une approche gloutonne, il peut donner une solution proche de l'optimale, mais pas nécessairement la meilleure solution. Dans certains cas, une combinaison différente d'actions peut fournir un profit plus élevé tout en respectant la limite de dépense.**

Ces limites doivent être prises en compte lors de l'utilisation de cet algorithme dans des scénarios réels. Pour des problèmes d'optimisation financière plus complexes et pour obtenir une solution réellement optimale, des approches algorithmiques plus avancées, telles que la programmation dynamique ou des algorithmes d'optimisation linéaire, pourraient être envisagées.

Comparaison de l'efficacité et des performances de l'algorithme de force brute par rapport à l'algorithme optimisé :

1. **Notation Big-O (Complexité temporelle) :**

- **Algorithme de Force brute : L'algorithme de force brute génère toutes les combinaisons possibles d'actions, ce qui signifie qu'il aura une complexité exponentielle. La complexité temporelle est généralement exprimée comme O(2^n), où 'n' est le nombre d'actions. Cela signifie que le temps d'exécution augmentera rapidement avec la taille du jeu de données, rendant l'algorithme inefficace pour de grandes quantités d'actions.**
- **Algorithme optimisé : L'algorithme optimisé trie d'abord les actions en fonction du gain potentiel, puis sélectionne les actions rentables jusqu'à atteindre la limite de dépense. Le tri initial a une complexité temporelle de O(n log n) (en fonction de l'algorithme de tri utilisé), et la boucle principale a une complexité linéaire O(n). Dans l'ensemble, la complexité temporelle de l'algorithme optimisé est O(n log n), ce qui est beaucoup plus efficace que l'algorithme de force brute pour de grandes quantités d'actions.**

1. **Analyse de la mémoire :**

- **Algorithme de Force brute : L'algorithme de force brute génère toutes les combinaisons possibles d'actions, ce qui signifie qu'il nécessitera une grande quantité d'espace mémoire pour stocker toutes ces combinaisons. La complexité d'espace de l'algorithme de force brute est O(2^n) car il doit stocker toutes les combinaisons possibles dans une structure de données.**
- **Algorithme optimisé : L'algorithme optimisé trie les actions, mais ne stocke pas toutes les combinaisons possibles. Il utilise simplement des listes pour stocker les meilleures actions sélectionnées et les valeurs associées. Par conséquent, la complexité d'espace de l'algorithme optimisé est O(n), car il ne stocke que les informations nécessaires pour trouver la meilleure combinaison.**
-

2. **Efficacité :**

- **L'algorithme de force brute est simple à implémenter, mais il est inefficace pour les jeux de données de grande taille en raison de sa complexité exponentielle. Il devient rapidement impraticable lorsque le nombre d'actions augmente.**
- **L'algorithme optimisé est beaucoup plus efficace que l'algorithme de force brute, car il évite de générer toutes les combinaisons possibles. En triant les actions en fonction du gain potentiel, il sélectionne les actions rentables en premier, ce qui permet de trouver une solution proche de l'optimale sans énumérer toutes les possibilités.**

En résumé, l'algorithme optimisé est beaucoup plus efficace en termes de temps d'exécution et d'utilisation de la mémoire que l'algorithme de force brute. Il peut gérer de plus grandes quantités d'actions de manière raisonnable, ce qui en fait une approche préférée pour résoudre ce type de problème d'optimisation financière.

Rapport d'exploration des données - Comparaison entre l'algorithme "optimise" et les choix de Sienna

Dans le cadre de cet rapport, nous allons comparer les résultats de l'algorithme "optimise" et les choix de Sienna pour différentes jeux de données. L'objectif est d'analyser les performances de chaque approche d'investissement et de déterminer si l'algorithme "optimise" offre des avantages par rapport aux choix de Sienna.

Données - data1 :

Algorithme "optimise" :

Meilleure combinaison : ['Share-XJMO', 'Share-MTLR', 'Share-KMTG', 'Share-LRBZ', 'Share-GTQK', ... (liste complète)], Investissement : 499,94 $, Profit : 198,51 $

Choix de Sienna :

Action choisie : Share-GRUT, Investissement : 498,76 $, Profit : 196,6 $

Données - data2 :

Algorithme "optimise" :

Meilleure combinaison : [Share-PATS', 'Share-JWGF', 'Share-ALIY', 'Share-NDKR', 'Share-PLLK', ... (liste complète)], Investissement : 499,99 $, Profit : 198,23 $

Choix de Sienna :

Actions choisies : [Share-ECAQ,Share-IXCI,Share-FWBE,...(liste complète)], Investissement total : 489,24 $, Profit total : 193,78 $

Observation importante concernant data2 :

Lors de l'analyse des choix d'investissement pour data2, nous avons observé la présence de valeurs négatives pour les prix des actions. Les prix négatifs peuvent survenir dans des circonstances exceptionnelles, mais en règle générale, ils ne sont pas possibles dans le contexte des marchés financiers.
Il est essentiel de prendre en compte cette observation lors de l'analyse des performances d'investissement pour data2. Des valeurs négatives peuvent entraîner des résultats inattendus et doivent être investiguées de plus près pour comprendre leur origine.
Dans notre cas , l'algorithme "optimise" convertit tous les prix négatifs en prix positifs et on suppose que Sienna a fait la meme chose.

Analyse des résultats :

Data1 : L'algorithme "optimise" a proposé une combinaison différente d'actions pour l'investissement par rapport au choix de Sienna. L'investissement total de l'algorithme "optimise" était légèrement supérieur à celui de Sienna, mais le profit réalisé était également légèrement plus élevé. Cela indique que l'algorithme a pu trouver une combinaison plus rentable pour les données spécifiques.
Data2 : Encore une fois, l'algorithme "optimise" a proposé une combinaison d'actions différente de celle de Sienna. L'investissement total de Sienna était inférieur à celui de l'algorithme "optimise", mais le profit réalisé était également inférieur. Cela suggère que les choix de l'algorithme "optimise" ont permis d'obtenir de meilleurs rendements pour ces données.

Conclusion :

La comparaison des résultats de l'algorithme "optimise" et des choix de Sienna pour les deux ensembles de données montre que l'algorithme a pu trouver des combinaisons d'actions plus rentables, conduisant à des profits plus élevés. Cependant, il est essentiel de noter que le choix d'actions d'investissement peut être complexe et dépend de nombreux facteurs tels que les conditions du marché, les préférences personnelles et les objectifs d'investissement.
L'algorithme "optimise" peut être un outil précieux pour explorer différentes stratégies d'investissement et identifier des combinaisons d'actions potentiellement plus rentables. Cependant, il est toujours recommandé de faire preuve de prudence et de faire ses propres recherches avant de prendre des décisions d'investissement. Enfin, il est important de garder à l'esprit que les performances passées ne garantissent pas les résultats futurs, et les investissements en bourse comportent toujours un certain niveau de risque. Les investisseurs doivent donc être prudents et diversifier leurs portefeuilles pour réduire les risques potentiels.