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/dataset[n]_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_best_combination(path):
  all_shares = get_data(path)  # Récupérer les données des actions
  dp_table = tableau de taille [nombre_total_d'actions + 1][budget_maximal + 1]
  initialiser dp_table avec des zéros

   Pour chaque action i allant de 1 à nombre_total_d'actions:
        Pour chaque budget j allant de 1 à budget_maximal:
              share_price = abs(all_shares[i].share_price)
              share_profit = all_shares[i].share_profit

       Si share_price > j OU share_price == 0:
          dp_table[i][j] = dp_table[i - 1][j]  # Ne pas sélectionner cette action
       Sinon:
          dp_table[i][j] = max(dp_table[i - 1][j],
                               dp_table[i - 1][j - share_price] + (share_price * share_profit) / 100)

  action_selectionnées = []
  i = nombre_total_d'actions
  j = budget_maximal

  Tant que i > 0 ET j > 0:
     Si dp_table[i][j] != dp_table[i - 1][j]:
        action_selectionnées.ajouter(all_shares[i])  # Ajouter l'action sélectionnée
        j = j - abs(all_shares[i].share_price)
     i = i - 1

  investissement_total = somme des prix absolus des actions dans action_selectionnées
  profit_total = somme des profits des actions dans action_selectionnées

  Afficher "Actions sélectionnées :", action_selectionnées
  Afficher "Investissement total :", investissement_total, "$"
  Afficher "Profit total :", profit_total, "$"
```

L'algorithme choisi pour la version optimisée :

```
ALGORITHME get_best_combination(path):
  all_shares = get_data(path)  # Récupérer les données des actions

  # Initialisation de la table de programmation dynamique
  dp_table = tableau de taille [nombre_total_d'actions + 1][budget_maximal + 1]
  POUR i allant de 0 à nombre_total_d'actions:       dp_table[i][0] = 0
  FIN POUR

  POUR i allant de 1 à nombre_total_d'actions:
    POUR j allant de 1 à budget_maximal:
      share_price = abs(all_shares[i].share_price)
      share_profit = all_shares[i].share_profit

      SI share_price > j OU share_price == 0:
        dp_table[i][j] = dp_table[i - 1][j]
      SINON:
        dp_table[i][j] = max(dp_table[i - 1][j],
                             dp_table[i - 1][j - share_price] + (share_price * share_profit) / 100)
      FIN SI
    FIN POUR
  FIN POUR

  action_selectionnées = Liste vide
  i = nombre_total_d'actions
  j = budget_maximal

  TANT QUE i > 0 ET j > 0:
    SI dp_table[i][j] != dp_table[i - 1][j]:
      action_selectionnées.ajouter(all_shares[i])
      j = j - abs(all_shares[i].share_price)
    FIN SI
    i = i - 1
  FIN TANT QUE

  investissement_total = somme des prix absolus des actions dans action_selectionnées
  profit_total = somme des profits des actions dans action_selectionnées

  AFFICHER "Actions sélectionnées :", action_selectionnées
  AFFICHER "Investissement total :", investissement_total, "$"
  AFFICHER "Profit total :", profit_total, "$"

  FIN ALGORITHME
```

Limites de l'algorithme (cas limites) :

1. **Cas où le budget maximal est très petit :**
   - **Limite : Si le budget maximal est trop petit pour acheter même une seule action, l'algorithme ne pourra sélectionner aucune action, même si certaines actions pourraient potentiellement générer des profits.**
   - **Cela peut se produire lorsque le budget est inférieur à tous les prix des actions disponibles.**
2. **Cas où toutes les actions ont un prix élevé :**
   - **Limite : Si toutes les actions ont des prix très élevés par rapport au budget maximal, il se peut qu'aucune action ne puisse être achetée. Même si certaines actions peuvent générer des profits élevés, le budget insuffisant limitera les choix.**
   - **Cela peut entraîner une sous-utilisation des ressources disponibles.**
3. **Cas où les profits ne sont pas proportionnels aux prix :**
   - **Limite : L'algorithme suppose que le profit est directement proportionnel au prix de l'action (à travers le pourcentage de profit associé). Cependant, dans la réalité, il peut y avoir des actions avec des prix bas mais des profits élevés, et vice versa. L'algorithme peut manquer de telles opportunités.**
4. **Cas où plusieurs combinaisons offrent le même profit :**
   - **Limite : Lorsque plusieurs combinaisons d'actions génèrent le même profit total, l'algorithme peut ne pas choisir la combinaison optimale en termes de diversification ou de gestion des risques.**
5. **Cas où l'historique des prix et des profits n'est pas représentatif :**
   - **Limite : Si les données historiques fournies ne représentent pas fidèlement le comportement futur des actions, les décisions prises par l'algorithme peuvent être basées sur des informations incorrectes, conduisant à des résultats sous-optimaux.**
6. **Cas où des actions ont des prix négatifs :**
   - **Limite : Bien que l'algorithme traite les prix négatifs en prenant leur valeur absolue, cela peut ne pas représenter correctement la réalité. Les actions avec des prix négatifs pourraient être des erreurs dans les données ou avoir des implications non gérées par l'algorithme.**
7. **Cas où les pourcentages de profit ne sont pas constants :**
   - **Limite : L'algorithme suppose que le pourcentage de profit par action est constant. Si les pourcentages de profit varient dans le temps ou en fonction d'autres facteurs, cela peut affecter la précision des prédictions de profit.**
8. **Cas de complexité temporelle :**
   - **Limite : Bien que l'algorithme utilise la programmation dynamique pour améliorer l'efficacité, la complexité temporelle peut devenir un problème avec un grand nombre d'actions, car la table de programmation dynamique devient plus grande.**
9. **Cas de données manquantes ou incorrectes :**
   - **Limite : Si les données fournies contiennent des erreurs, des valeurs manquantes ou des incohérences, cela peut entraîner des résultats incorrects**

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

1. Notation Big-O :

- **Force brute :**
  - **Calculer toutes les combinaisons possibles implique de parcourir chaque sous-ensemble de n actions, où n est le nombre total d'actions.**
  - **Complexité temporelle : O(2^n) car il y a 2^n sous-ensembles possibles.**
  - **Complexité spatiale : O(1) car il n'est pas nécessaire de stocker tous les sous-ensembles.**
- **Algorithme optimisé (avec DP) :**
  - **Complexité temporelle : O(n * B), où n est le nombre d'actions et B est le budget maximal. Cela résulte du remplissage de la table de programmation dynamique.**
  - **Complexité spatiale : O(n * B), en raison de la table de programmation dynamique de taille n * B.**

2. Complexité Temporelle :

- **Force brute : La complexité exponentielle de l'algorithme de force brute (O(2^n)) rend son exécution extrêmement lente dès que le nombre d'actions augmente. Il devient rapidement impraticable.**
- **Algorithme optimisé : Grâce à l'utilisation de la programmation dynamique, l'algorithme optimisé a une complexité temporelle beaucoup plus efficace (O(n * B)), ce qui le rend adapté à des ensembles de données plus larges. Cependant, la complexité dépend également du budget maximal.**

3. Analyse de la Mémoire :

- **Force brute : En utilisant des méthodes de génération de sous-ensembles, l'algorithme de force brute n'a pas besoin de beaucoup d'espace mémoire supplémentaire, car il traite les sous-ensembles au fur et à mesure de leur génération.**
- **Algorithme optimisé : En utilisant la programmation dynamique, l'algorithme optimisé nécessite une table de mémoire (tableau) pour stocker les résultats intermédiaires de sous-problèmes. La taille de cette table dépend du nombre d'actions et du budget maximal, ce qui peut augmenter l'utilisation de la mémoire pour des problèmes complexes.**

4. Efficacité :

- **Force brute : En raison de sa complexité exponentielle, l'algorithme de force brute devient rapidement inefficace et impraticable pour des ensembles de données plus importants.**
- **Algorithme optimisé : Grâce à sa complexité temporelle plus faible, l'algorithme optimisé est beaucoup plus efficace pour des ensembles de données plus grands. Il peut traiter plus d'actions dans un délai raisonnable, ce qui en fait une solution pratique.**
-

En résumé, l'algorithme optimisé avec la programmation dynamique est beaucoup plus efficace et performant que l'algorithme de force brute, en particulier pour des ensembles de données de taille significative. Il peut traiter davantage d'actions tout en ayant une meilleure complexité temporelle. Cependant, il convient de noter que l'efficacité dépend également des spécificités du problème, comme le budget maximal et les pourcentages de profit.

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-CUSU', 'Share-DBUJ', 'Share-KMTG', 'Share-KXOH', 'Share-DXOW', ... (liste complète)], Investissement : 499 $, Profit : 204.53 $

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 : 498 $, Profit : 206.22 $

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, j'ai 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.