Partie 1 : Bruteforce

**Contraintes** :

* Une action peut être achetée qu’une seule fois
* Une action ne peut se vendre fractionnée
* Total d’achat maximum 500€

**Nombre de possibilités :**

* 20 actions comprenant 2 possibilités :
  + Acheter l’action
  + Ne pas acheter l’action

Donc 220 = 1 048 576 possibilités

**Proposition d’algorithme** :

A partir de la liste *(listActions[220])* de toutes les combinaisons possible, sélectionner celles dont le cout ne dépasse pas 500€.

Pour chacun de ces lignes, calculer le rendement et vérifier si supérieur à la valeur précédente.

Si supérieur : stockage de la combinaison d’action dans une liste (*meilleurCombinaisonActions[]*) et stocker rendement dans une variable(*meilleurRendement*).

Pour faire le choix des actions à acheter, j’utilise un « comptage binaire » sur 20 digits :

1 = 00000000000000000001

2 = 00000000000000000010

3 = 00000000000000000011

Etc.

Chaque *digits* correspond à une action et le chiffre (0 ou 1) à un acte :

* 0 = ne pas acheter l’action
* 1 = acheter l’action

Prenons l’exemple suivant avec le chiffre 405 874 (base 10) = 01100011000101110010 (base 2)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **BIN** | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| **N° action** | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Cette combinaison sera donc l’achat des actions 19, 18, 14, 13, 9, 7, 6 et 5.

Il faudra donc ensuite calculer le rendement et le comparer au précedent meilleur.

**Pseudo-code :**

Algorithme meilleurRendement

Variable

listActions[220] : LISTE CONTENANT TOUTES LES COMBINAISONS DE SELECTION

meilleurCombinaisonActions[] : LISTE VIDE POUR STOCKER MEILLEUR CHOIX D’ACHAT

meilleurRendement <- 0 : RENDEMENT ASSOCIE A LA MEILLEURE COMBINAISON D’ACTION

Début

Pour i <- 1 ; taille listCombinaisonActions[220] ; i = i + 1

SI cout(listCombinaisonActions[i]) =< 500 :

SI rendement(listCombinaisonActions[i]) > meilleurRendement :

meilleurCombinaisonActions[] <- listCombinaisonActions[i]

meilleurRendement <- rendement(listCombinaisonActions[i])

Fin SI

Fin SI

Fin POUR

Fin

**Optimisation :**

Le montant maximum étant de 500€, si j’essaye de m’approcher de ces 500€ en achetant que les plus petites actions (petite = les moins cher), cela signifie que n’importe quels autres achats (avec des actions qui seront donc forcément plus cher) seront obligatoirement supérieurs à 500€.

|  |  |  |
| --- | --- | --- |
| **Actions #** | **Coût par action** | **Bénéfice (après 2 ans)** |
| Action-20 | 114 € | 18% |
| Action-12 | 110 € | 9% |
| Action-6 | 80 € | 25% |
| Action-4 | 70 € | 20% |
| Action-5 | 60 € | 17% |
| Action-3 | 50 € | 15% |
| Action-9 | 48 € | 13% |
| Action-11 | 42 € | 17% |
| Action-13 | 38 € | 23% |
| Action-10 | 34 € | 27% |
| Action-2 | 30 € | 10% |
| Action-8 | 26 € | 11% |
| Action-19 | 24 € | 21% |
| Action-7 | 22 € | 7% |
| Action-1 | 20 € | 5% |
| Action-15 | 18 € | 3% |
| Action-14 | 14 € | 1% |
| Action-18 | 10 € | 14% |
| Action-16 | 8 € | 8% |
| Action-17 | 4 € | 12% |
|  | 448 € |  |

Nous ne pouvons donc pas dépasser l’achat de 16 actions ce qui réduit à 216 le nombre de possibilité (et donc la durée de traitement).

Nous passons donc de 1 048 576 à 65 536 combinaisons ! Le nombre binaire sur lequel s’appuie la sélection d’action à acheter ne doit donc pas comporter pour de 16 « 1 ».

**Pseudo-code optimisé :**

Algorithme meilleurRendement

Variable

listActions[220] : LISTE CONTENANT TOUTES LES COMBINAISONS DE SELECTION

meilleurCombinaisonActions[] : LISTE VIDE POUR STOCKER MEILLEUR CHOIX D’ACHAT

meilleurRendement <- 0 : DECIAM, RENDEMENT ASSOCIE A LA MEILLEURE COMBINAISON D’ACTION

Début

Pour i <- 0 ; taille listCombinaisonActions[220] ; i = i + 1

**SI bin(i) contient moins de 16 « 1 » :**

SI cout(listCombinaisonActions[i]) =< 500 :

SI rendement(listCombinaisonActions[i]) > meilleurRendement :

meilleurCombinaisonActions[] <- listCombinaisonActions[i]

meilleurRendement <- rendement(listCombinaisonActions[i])

Fin SI

Fin SI

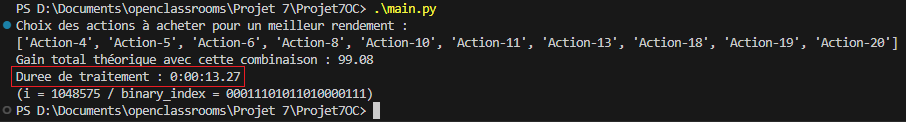
Fin SI

Fin POUR

Fin

**Conclusion** :

Avec cette optimisation, le temps de traitement est d’environ 13 secondes sur une vieille machine (la mienne, Lenovo de plus de 5 ans)

****

Partie 2 : optimisation

**Méthode du knapsack01** :

Cette méthode utilise la récursivité pour tester l’ensemble des combinaisons possible. Nous avons un portefeuille ne pouvant contenir que pour 500€ d’actions. Chaque élément a un cout et un rendement. Le but de cette méthode est donc de trouver la meilleure combinaison pour obtenir le meilleur rendement coutant le moins cher. Exprimé différemment : trouver le meilleur ratio cout / rendement pour obtenir la combinaison qui coute le moins cher pour un rendement le plus élevé.

Value = [20, 30, 50, 70, 60, 80, 22, 26, 48, 34, 42, 110, 38, 14, 18, 8, 4, 10 , 24, 114]

Profits = [5, 10 , 15, 20, 17, 25, 7 , 11, 13, 27, 17, 9 , 23, 1, 3, 8, 12, 14, 21, 18]

CostMax = 550

Il faut donc se baser sur les actions qui ont le meilleur rendement et en acheter jusqu’à arriver à la limite des 500€.

1 ) créer une liste contenant le ratio Profits / Value

2 ) trier les listes COST et PROFITS dans le même ordre que cette nouvelle liste

3 ) lancer une routine de sélection des actions (trié dans ce nouvel ordre) jusqu’à atteindre, et sans dépasser, la limite des 500€

4 ) ajouter les actions à acheter dans une liste, calculer le montant des achats et le rendement