OpenClassRooms : Projet 7

Résolution de problèmes en utilisant des algorithmes en Python

Présentation

- La Méthode de Force Brute
- Notre algorithme optimisé
- Comparaison de notre algorithme avec celui de Sienna

L'Algorithme de force brute

- Présentation
- Exemple
- Avantages
- Limite

L'Algorithme de force brute

Présentation

Crée toutes les combinaisons possible de n éléments.

En considérant n comme étant le nombre d'entrées, le nombre de combinaison possibles sera de : `2^n`

L'Algorithme de force brute

Exemple

Analyse de la fonction brute_force (bruteforce.py)

Exemple :
Output sur brute_force(a,b,c) :

```
[[], ['c'], ['b'], ['c', 'b'], ['a'], ['c', 'a'], ['b', 'a'], ['c', 'b', 'a']]
```

```python
def brute_force(elements):
    if len(elements) == 0:
        return [[]]
    firstElement = elements[0]
    # Copy the array without first element
    rest = elements[1:]

    combsWithoutFirst = brute_force(rest)
    combsWithFirst = []

    for comb in combsWithoutFirst:
        combWithFirst = [*comb, firstElement]
        combsWithFirst.append(combWithFirst)
    # If combinations are still possible
    return [*combsWithoutFirst, *combsWithFirst]
```

L'Algorithme de force brute

Avantages & inconvénients

Avantages :

- Taux d'efficacité de 100%
- Aucune erreur ou marge d’erreur possible, donc méthode parfaite du point de vue des résultats

Limites:

- Complexité exponentielle
- Prend énormément de temps sur les machines conventionnelles

Optimized.py : Une réponse adaptée aux besoins clients.

- Présentation du pseudo-code
- Analyse des performances / Comparaison des résultats
- Avantages et limites de l’algorithme

Optimized.py

Présentation du pseudo code

```
Set budget to 500
# We will try the algorithm on two lists of the same actions

 One list is sorted by pure benefits.
 One list is sorted by benefits / € spent.

# We will run these two lists into our algorithm
# and compare the results to pick the best output.

for each list :
    WHILE there are still actions to be bought and we have enough money left to at least
    buy the cheapest one :
        Try to buy the best action
            If we can afford it:
               # Buy it
                Add the action to our shopping list
                Remove the price of the action from the budget
                Remove the action from the actions to loop from
            Else:
                #Don't buy it

        Set the next action as the best action
        Go to the beginning of the loop and repeat



  # This loop will run until one of the break condition is met
```

Préparation de l'environnement

Exécution de l'algorithme

Optimized.py

L'algorithme et ses performances

Dataset 1 :

```
Lancement de l'analyse de 956 action(s).

Il nous reste 1.240000000000009 € sur 500 € de budget de départ.
Nous avons un profit total de : 196.61119200000002 € répartis sur 1 action(s).
Le choix secondaire nous rapportait 151.18885100000003 € répartis sur 88 action(s).
{'name': 'Share-GRUT', 'cost': '498.76', 'percent_benef': '39.42', 'benef': 196.61119200000002, 'densite':
 0.07903600930307163}
Le calcul de possibilités a pris 0.004986286163330078 secondes
```

```
Sienna bought:

Share-GRUT

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

Dataset 2 :

```
Lancement de l'analyse de 541 action(s).

Il nous reste 0.7799999999999887 € sur 500 € de budget de départ.
Nous avons un profit total de : 182.578906 € répartis sur 12 action(s).
Le choix secondaire nous rapportait 104.100902 € répartis sur 54 action(s).
{'name': 'Share-JWGF', 'cost': '48.69', 'percent_benef': '39.93', 'benef': 19.441917, 'densite': 0
 .8200862600123229}
{'name': 'Share-MBQU', 'cost': '51.46', 'percent_benef': '35.78', 'benef': 18.412388, 'densite': 0
 .6952973130548}
{'name': 'Share-QEVK', 'cost': '49.77', 'percent_benef': '34.38', 'benef': 17.110926000000003, 'densite':
 0.6907775768535263}
{'name': 'Share-DLNE', 'cost': '44.06', 'percent_benef': '36.74', 'benef': 16.187644000000002, 'densite':
 0.8338629142078984}
{'name': 'Share-IJFT', 'cost': '40.91', 'percent_benef': '38.89', 'benef': 15.909898999999998, 'densite':
 0.9506233194817894}
{'name': 'Share-ANFX', 'cost': '38.55', 'percent_benef': '39.72', 'benef': 15.312059999999999, 'densite':
 1.0303501945525293}
{'name': 'Share-MALJ', 'cost': '46.37', 'percent_benef': '32.88', 'benef': 15.246456, 'densite': 0
 .7090791459995688}
{'name': 'Share-OPBR', 'cost': '39.0', 'percent_benef': '38.95', 'benef': 15.190500000000002, 'densite':
 0.9987179487179488}
{'name': 'Share-FWMV', 'cost': '41.68', 'percent_benef': '35.8', 'benef': 14.921439999999997, 'densite':
 0.8589251439539347}
{'name': 'Share-HATC', 'cost': '43.45', 'percent_benef': '34.14', 'benef': 14.83383, 'densite': 0
 .7857307249712313}
{'name': 'Share-XGNC', 'cost': '41.86', 'percent_benef': '35.14', 'benef': 14.709603999999999, 'densite':
 0.8394648829431438}
{'name': 'Share-XQII', 'cost': '13.42', 'percent_benef': '39.51', 'benef': 5.302242, 'densite': 2
 .9441132637853946}
Le calcul de possibilités a pris 0.00299072265625 secondes
```

```
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,¬
```

Optimized.py

Avantages & inconvénients

Avantages :

- Exécution très rapide (- de 0,01s)
- Tri effectué selon plusieurs paramètres et comparaison des résultats pour une sortie optimale

Limites:

- Marge d'erreur potentielle par rapport à la méthode de force brute (d'environ 5% en se référant à l'algorithme de Sienna)

Comparaison

En considérant n comme le nombre d'actions de la liste à analyser :

Algorithme de force brute

- Complexité en temps : O(2^n)
- Complexité en mémoire : O(2^n)
- Exponentielle

Algorithme optimisé

- Complexité en temps : O(n)
- Complexité en mémoire : O(n)
- Linéaire