Résolvez des problèmes en utilisant des algorithmes en Python

Projet 07 de la formation Python

DUMET Mark / 20.02.2021

Solution de l’algorithme de force brute.

La recherche exhaustive ou recherche par **force brute** est une méthode algorithmique qui consiste principalement à essayer toutes les solutions possibles d’un problème donné.

Ici, notre algorithme de **force brute** est très efficace sur des petites quantités d’actions, mais des qu’il s’agit de grandes quantités il devient très inefficace voir même impossible à finir sa tache tant il y a de calcule possible.

Solution de l’algorithme de force brute.

La **complexité d'un algorithme** est la quantité de ressources nécessaires pour traiter des entrées.

Ici pour notre cas à 20 actions différentes, la **complexité de l’algorithme** vas être de 2^n, car il y a deux solutions à chaque actions, soit on la garde dans le budget, donc 1, soit on ne la garde pas, donc 0.

Pour nos 20 actions, 2^{20} , cela nous fait une **complexité de 1048576** opérations. On peut voir que ce nombre est calculable dans des temps acceptables, mais si non voulons 1000 actions, cela prendrais 2^{1000}, soit une **complexité de 1.071509e+301**, donc incalculable dans des temps raisonnables.

Solution de l’algorithme optimisé.

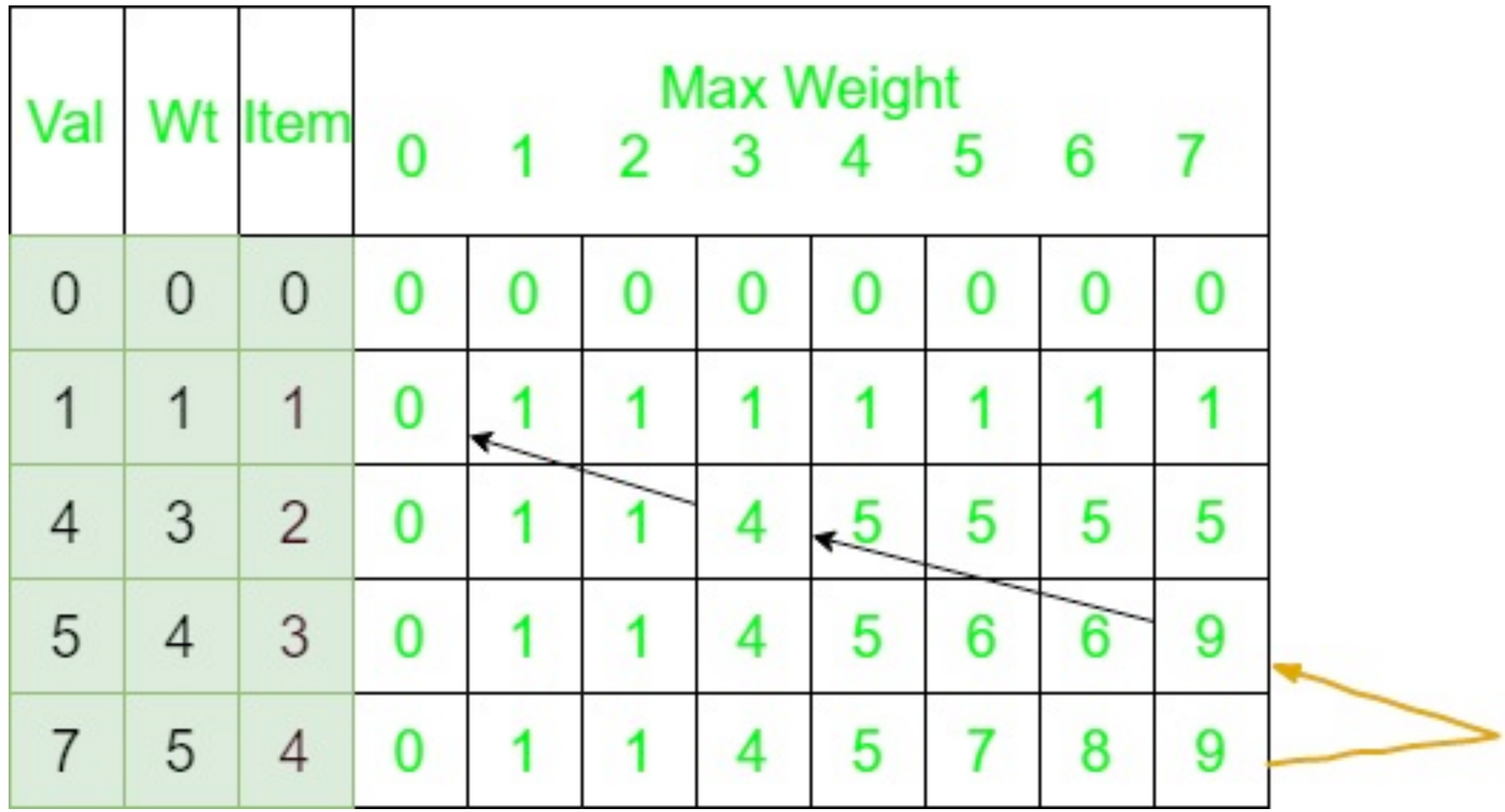
Comme nous venons de le voir, la force brute est utilisable pour de petites bases de données, mais pas pour des nombres plus grands.

C’est pour cela nous allons utiliser une solution optimisée, qui permet d’avoir la solution dans des temps raisonnable et en utilisant le moins de ressources possible.

Solution de l’algorithme optimisé.

Ici une solution dynamique.

Val	Wt	Item	Max Weight 0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	1	1	1	1	1	1
4	3	2	0	1	1	4	5	5	5	5
5	4	3	0	1	1	4	5	6	6	9
7	5	4	0	1	1	4	5	7	8	9



Solution de l’algorithme optimisé.

Pseudocode de la solution dynamique :

```
def optimized(budget, stocks):
    Tableau = budget, stocks

    pour i entre 1 et nombres_d_actions:
        Pour w entre 1 et budget:
            si la valeur de l’action i est <= au budget dispo, alors:
                Tableau = max entre stock i et stock i-1
            sinon:
                tableau = stock i-1

    stocks_selection = [ ]

    Tant que w >= 0 et n >= 0:
        e = selection de la dernière action

        Si tableau == tableau:
            stocks_selection.append(e)
            budget -= prix de l’action

        nombres_d_actions -= 1

    return stocks_selection
```

Solution de l’algorithme optimisé.

Pour calculer la complexité nous utiliserons la notation big O.

La notation Big O (ou complexité algorithmique) est une manière standard de mesurer la performance d’un algorithme. **C’est une manière mathématique de juger de l’efficacité d’un algorithme**.

Ici notre premier algorithme de force brute est de complexité $O(n^2)$. La complexité quadratique **$O(n^2)$** va évoluer au carré de l’input. Le nombre d’opérations grossit au carré de l’input en entrée.

Notre algorithme optimisé est lui de complexité O(n). La complexité linéaire **O(n)** évolue en lien direct avec la taille de l’input. Le nombre d’opérations grossit avec le nombre de l’input en entrée.