Projet 7: Algorithme

Juin 3, 2021

Aperçu

Nous allons présenter 2 approches différentes pour la resolution du probleme du meilleur investissements

- Par Brute Force (approche naive)
- Par programmation dynamique (memoization)

Comprehension du probleme

Objectif

Il s'agit de maximiser le bénéfice par la meilleur sélection des actions a acheté

Contraintes

- Une action ne peut être sélectionné qu'une seule fois
- L'achat des actions ne peut excéder 500 euros
- On ne peut pas fractionner nos actions
- On ne dispose que d'un seul portefeuille de 500 euros

Solutions

- Un algorithme naive de type brute force
- Un algorithme optimisé capable de traiter les informations en entrée en un temps de traitement non exponentiel

Enonce du probleme

Cet exercice reprend le problème classique en algorithmie du sac à dos (knapsack problem 0/1)

- Maximiser le profit P
- Tout en ayant un coût <= C
- Une action peut être sélectionné 1 fois ou 0

$p_j = profit$ of item j,

$w_j = weight$ of item j,

$c = capacity$ of the knapsack,

$$\text{maximize} \quad z = \sum_{j=1}^{n} p_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \leq c,$$

$$x_j = 0 \text{ or } 1, \qquad j \in N = \{1, \ldots, n\},$$

where

$$x_j = \begin{cases} 1 & \text{if item } j \text{ is selected;} \\ 0 & \text{otherwise.} \end{cases}$$

Brute force

Le principe consiste a énuméré toutes les possibilités et à sélectionner parmis toutes, celle qui satisfait l'enonce du probleme.

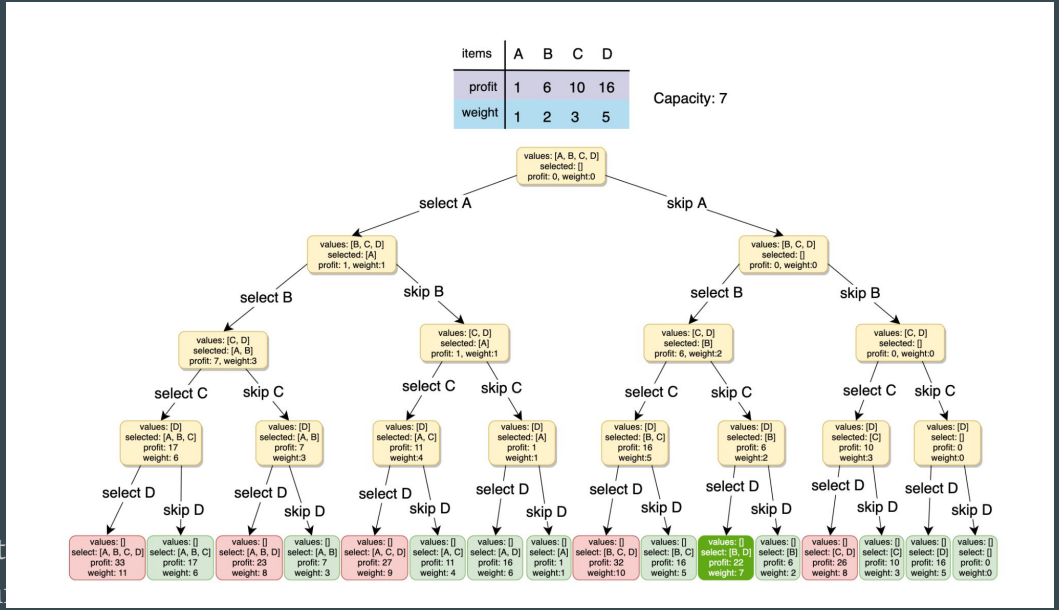
Brute Force Caracteristiques

Efficacité

- Puisqu'il y a n éléments, il y a 2n combinaisons possibles d'éléments.
- 2^20 = 1048576 possibilités
- Ainsi, le temps d'exécution sera O(2^n)

Implications:

Le temps de traitement croit exponentiellement en fonct
du nombre d'actions en entrée. Ce qui en fait un algorith
lent qu'on ne peut utiliser qu'avec un nombre limité
d'éléments



Algorithm brute force

Weights[1 ... N], Values[1 ... N], Items{1 ... N}

//Input: Array Weights contains the weights of all items
Array Values contains the values of all items
Array Items contains the names of all items

//Output: Best possible combination of items in the best_items [1 .. N]
Max_value: best profit
Cost: the cost less than or equal to 500 euros

Algorithm brute force

```
Max_value ← 0,
for index ← 0 to number of items
        for subset in Genretate_combinations(index+1)
                Cost ← 0 and profit ← 0
                        for i in subset
                                Cost ← cost + weights[i]
                                profit ← profit + values[i]
                                If  cost <= capacity and profit > max_value:
                                        Max_value ←  profit and best_items ← subset
Cost ← 0
For i in best_items
        Cost ← cost + weight[i]
Return cost, max_value, best_items
```

Dynamic programming (DP)

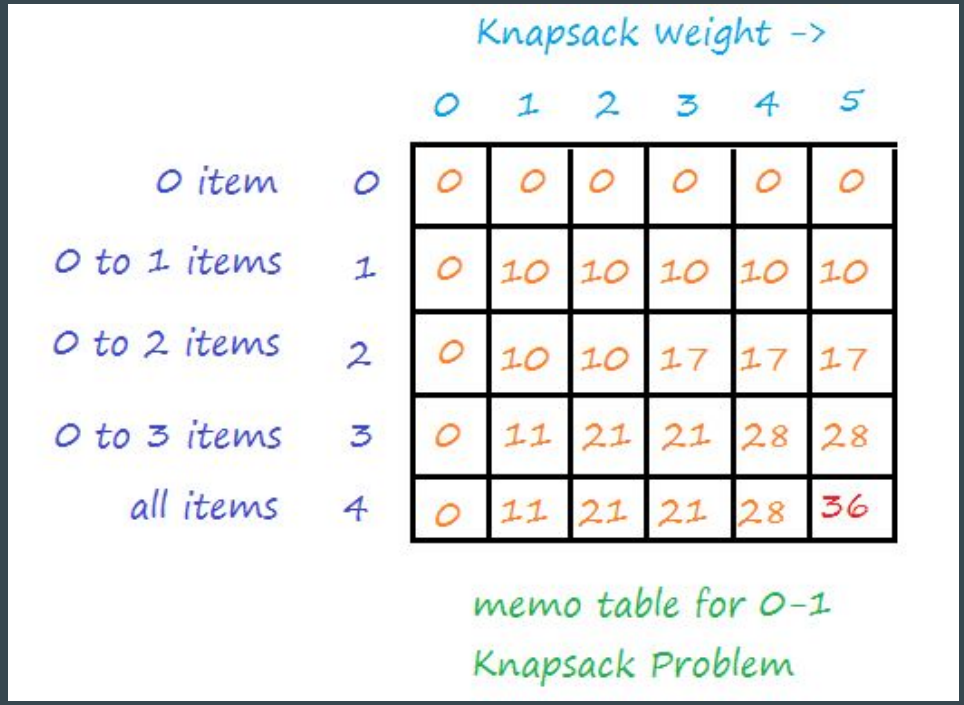
L'idée de base de la programmation dynamique est d'utiliser une table pour stocker les solutions des sous-problèmes résolus. Si vous rencontrez à nouveau un sous-problème, il vous suffit de prendre la solution dans le tableau sans avoir à le résoudre à nouveau.

Dynamic Programming Caracteristiques

Efficacité

• Puisqu'on utilise un tableau de n éléments multiplié par la capacité, il y a donc:
N * Capacité cycles
• Ex: 20 * 500 = 10.000 cycles
• Ainsi, la complexite de cet algo est:
O (N*Capacity)

En terme de mémoire, cet algorithme requiert un tableau à 2 dimensions avec les lignes correspondant au nombre d'items et les colonnes a la capacité



Algorithm dynamic programming

SI W[i] > W
    M[i][W] = M[i-1][W]
SINON
    M[i][W] = max (M[i-1][W], M[i-1][W - W[i]] + P[i])

		i\w	0	1	2	3	4	5	6	7	8
Pi	Wi	0	0	0	0	0	0	0	0	0	0
2	3	1	0	0	0	2	2	2	2	2	2
3	4	2	0	0	0	2	3	3	3	5	5
4	5	3	0	0	0	2	3	4	4	5	6
1	6	4	0	0	0	2	3	4	4	5	6

Weights = {3, 4, 5, 6}

Profits = {2, 3, 4, 1}

W = 8

n = 4

Max profit is 6

Algorithm dynamic programming

```
TANT QUE W > 0
        TANT QUE i > 0 ET M[i][W] EGALE M[i-1][W]
                décrémente i
        W = W - W[i]
        SI W > 0
                Ajoute-item ( W[i] )
        décrémente i
```

		i\w	0	1	2	3	4	5	6	7	8
Pi	Wi	0	0	0	0	0	0	0	0	0	0
2	3	1	0	0	0	2←	2	2	2	2	2
3	4	2	0	0	0	2←	3	3	3	5	5
4	5	3	0	0	0	2	3	4	4	5	6←
1	6	4	0	0	0	2	3	4	4	5	6←

I = 4 ptr = 6
I = 3 ptr = 6 break W = 8 - 5 = 3
Selected {5}
I = 2 ptr = 2
I = 1 ptr = 2 break W = 3 - 3 = 0
Selected {5, 3}

Comparaison pour le dataset 1

Mon resultat

The shares bought:
Share-KMTG | Share-GHIZ | Share-NHWA | Share-UEZB | Share-LPDM | Share-MTLR | Share-USSR | Share-GTQK | Share-FKJW | Share-MLGM | Share-QLMK | Share-WPLI | Share-LGWG | Share-ZSDE | Share-SKKC | Share-QQTU | Share-GIAJ | Share-XJMO | Share-LRBZ | Share-KZBL | Share-EMOV | Share-IFCP

Total cost: 499.95€

Total return: 198.54€

Time solving: 29.50s

Le résultat de Sienna

Sienna bought:

Share-GRUT

Total cost: 498.76€
Total return: 196.61€,

Observations

- Le bénéfice que je trouve est sensiblement supérieur à celui de Sienna.
- Toutefois, cette dernière trouve une solution avec un nombre d'actions acheté nettement plus réduit que le mien pour un bénéfice assez similaire. Ce qui peut être une considération à prendre en compte

Comparaison pour le dataset 2

Mon resultat

The shares bought:

Share-ECAQ | Share-IXCI | Share-FWBE | Share-ZOFA | Share-PLLK | Share-LXZU | Share-YFVZ | Share-ANFX | Share-PATS | Share-SCWM | Share-NDKR | Share-ALIY | Share-JWGF | Share-JGTW | Share-FAPS | Share-VCAX | Share-LFXB | Share-DWSK | Share-XQII | Share-ROOM

Total cost: €499.90

Total return: €197.96

Time solving: 16.52s

Le résultat de Sienna

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: 489.24€,

Profit: 193.78€,

Observations

- On observe que nous avons tous 2 sélectionné les mêmes actions a une différence près: mon algo a ajouté Share-LXZU en plus
- Ce qui a pour consequence que j'ai un benefice > a celui de sienna.

Comparaison brute force / dynamic programming

Brute Force

The shares bought:

Action-4 | Action-5 | Action-6 | Action-8 | Action-10 | Action-11 | Action-13 | Action-18 | Action-19 | Action-20

Total cost: €498.00
Total return: €99.08

Time solving: 3.29s

Complexite:
O(2^n)
2^20 = 1.048.576

Dynamic programming

The shares bought:

Action-20 | Action-19 | Action-18 | Action-13 | Action-11 | Action-10 | Action-8 | Action-6 | Action-5 | Action-4

Total cost: €498.00
Total return: €99.08

Time solving: 0.52s

Complexite:
O(nW)
20 * 500 = 10.000

Observations

On constate d'abord que les résultats obtenus sont rigoureusement les mêmes. (même set d'actions)

La différence se note au niveau de la durée de traitement. En effet, on note que l'algo DP est nettement plus rapide que le brute force.

Brute Force	DP
3.29s	0.52s

DP 3,5x plus rapide que Brute Force. Compte tenu de la difference de complexité des 2 algos