ALGORITHME DU SAC À DOS

25 octobre 2020

.

Probleme du sac a dos
resolution par le methode de Programmation Dynamique

1 Specifications

On dispose d'un sac à dos ne pouvant supporter qu'un certain poids, et l'on considère un ensemble d'objets ayant chacun un poids et une valeur.

La question est la suivante : quels objets peut-on mettre dans le sac sans dépasser sa capacité de poids, afin de maximiser la valeur totale ?

La spécification mathématiques du problème est la suivante :

- On peut utiliser un ensemble de **n-uplet de couple** ***S****=((w1,v1),(w2,v2),....(wn,vn))* pour modeliser l'ensemble des n objets où *wi* represente le poids de l'objet i et v_i sa valeur.
- On note **W** la capacité du sac et l'on suppose que $\forall i$, $1 \leq i \leq$n, $w_i > 0$ et $v_i > 0$.
- On émet egalement l'hypothese que $\sum_{i=1}^{n} w_i > W$
- Un choix d'objet est un **n-uplets d'entier naturels** *$(x_1, x_2, ...xn)$*, où x_i vaut 1 ou 0, selon que l'on prenne ou non l'objet i. Le poids d'une telle sélection est donc $\sum_{i=1}^{n} x_i w_i$ et sa valeur $\sum_{i=1}^{n} x_i v_i$

Notre question peut alors être reformuler comme suit : on cherche un n-uplet de binaires *$(x_1, x_2, ...xn)$*, qui **maximise** $\sum_{i=1}^{n} x_i v_i$ sous la contrainte $\sum_{i=1}^{n} x_i w_i \leqslant W$

2 Algorithme

On va se focaliser d'abord sur la valeur maximale totale des objets que l'on va pouvoir mettre dans le sac. La reconstitution de la liste des objets correspondant à cette valeur sera effectuée dans un second temps.

Nous avons ici deux cas de figures a savoir :

- Si l'on met le i-ème élément dans le sac, on est amené à calculer récursivement la valeur maximale que l'on peut mettre dans un sac de capacité diminuée de w_i avec les premiers *i-1* objets
- Si l'on ne met pas le i-ème élément dans le sac, on est amené à calculer récursivement la valeur maximale que l'on peut mettre dans un sac de capacité inchangée avec les premiers *i-1* objets

Puis Il ne nous restera plus qu'à déterminer la liste des objets correspondant à la valeur maximale, c'est-à-dire reconstituer la solution optimale.

On obtient donc l'algorithme suivant :

```
Function SacadosPD(V, P, T, Keep, W){

        // ou W  la contenance max du sac
        // V[i] est la valeur de l'objet i
        // P[i] le poid de l'objet i
        // T et Keep le tableau de len(V) ligne et W colonne

        for i := 1 to len(V) do
                for j := 1 to W do
                        T[i][j] := 0
                        Keep[i][j] := 0
                endfor
        endfor

        // Recherche de la valeur maximal pouvant etre contenu dans le sac

        for i:=1 to len(V) do
                for c:=0 to W do
                        if c>=P[i] do
                                T[i,c] := max(T[i-1,c], T[i-1, c-P[i]] + V[i])
                                Keep[i][j] := 1
                        else
                                T[i,c] := T[i-1,c]
                        endif
                endfor
        endfor

          // Construction de la solution optimal

        w := W
        L := []
        j := len(V)
        while(j>=1) do
                if Keep[j][w] == 1 do
                        L.append(i)
                        w = w-P[i]
                endif
                j = j - 1
        endwhile

        /* T[len(V)][W] retourne la valeur maximal pouvant
        etre contenu dans le sac de capacite W
        L contient les indices des objets qui on ete mis dans le sac */

        return ( T[len(V)][W], L )
```

```
}
```

3 Complexité

L'on remarque que l'algorithme SacadosPD est pricipalement basé sur l'imbrication de deux boucles a savoir la boucle *"for i :=1 to len(V) do"* et *"for c :=0 to W do"* alors avec n le nombre d'objets (n=len(V)) la complexité de cette algorithme est en $\ominus(nW)$. Mains nous ne pouvons pas conclure a une complexité **linaire** par rapport à W, car W est exponentielle par rapport a son codage c'est a dire, si les poids des objets sont décimaux, cela oblige à multiplier les poids des objets et la capacité du sac afin de les rendre entiers. On aura donc $\boldsymbol{W} = 2^{log_2(W)}$ ainsi la complexité de l'algorithme est **exponentielle** par rapport à la taille de W