TP1 : Problème Du Sac A Dos (Knapsack Problem)

TPGO SIL2 - Hidouci WK
Octobre 2018

Solution proposée par :
Guedouari Mohamed et Mestoui Yacine

Préambule :

En algorithmique, le problème du sac à dos, noté également KP (en anglais, Knapsack problem) est un problème d'optimisation combinatoire. Il modélise une situation analogue au remplissage d'un sac à dos, ne pouvant supporter plus d'un certain poids, avec tout ou partie d'un ensemble donné d'objets ayant chacun un poids et une valeur. Les objets mis dans le sac à dos doivent maximiser la valeur totale, sans dépasser le poids maximum. [1]
Dans ce tp, on doit adapter un algorithme dynamique de résolution du ce problème à une structure 3D ; ie, on suppose qu'il existe une liste d'objets possédant chacun un poids et un volume, et en sortie on aura la listes d'objets qui doivent être déposés au sac de façon à ne pas dépasser la capacité(poids et volume) max de ce dernier.

1. Source : wikipedia.org

Problème du sac à dos en 2D

Solution récursive

La fonction suivante F présente la solution récursive du problème, en ayant la liste d'objets (Obj1, Obj2, ... Objn), les poids(P1,P2, ... P3), l'utilité de chaque objet(U1,U2, ... ,Un), et le poids max du sac (Pmax).

$$F(Objn, Pmax) = max \begin{cases} F(Objn-1, Pmax) & \text{objet } !choisi \\ F(Objn-1, Pmax-Pn)+Un & \text{objet } choisi \end{cases}$$

On présente dans ce qui suit l'implémetation récursive de cette fonction en languague C. La complexité de cette fonction est de $\mathcal{O}(2^n)$.

```
1  int sac_a_dos2d_recursive(int w, int wt[], int val[], int n){
2      if(n==0 || w==0)
3      {
4          return 0;
5      }else
6      {
7         if(wt[n-1]>w)
8          {
9              return sac_a_dos2d_recursive(w, wt, val, n-1);
10         }else
11         {
12             return max(val[n-1] + sac_a_dos2d_recursive(w - wt[n-1], wt, val,
                   n-1),
13                     sac_a_dos2d_recursive(w, wt, val, n-1));
14         }
15     }
16 }
```

Solution en utilisant la programmation dynamique

Comme d'autres problèmes typiques de la programmation dynamique (DP), les re-calculs des mêmes sous-problèmes peuvent être évités en construisant un tableau temporaire K[][] de manière ascendante.
Pour résoudre donc ce problème, on commence par résoudre les plus petits sous-problèmes et on conserve les valeurs de ces sous-problèmes dans une table de programmation dynamique ;
L'algorithme suivant présente la solution en utilisation le tableau de deux dimensions. Les colonnes de K représente les objets, donc elle est de taille n, et les lignes de cette table représentent les poids possibles (0,1, .. , Pmax-1 , Pmax).

```
1  int sac_a_dos2d_dynamique(int W, int wt[], int val[], int n)
2  {
3      int i, w;
4      int **K;
5
6      K = (int **) malloc (sizeof(int **)*(n+1));
7      for (int h = 0; h < n+1; h++)
8      {
9       K[h] = (int *) malloc(sizeof(int)*(W+1));
10     }
11
12     for (i = 0; i <= n; i++)
13     {
14         for (w = 0; w <= W; w++)
15         {
16             if (i==0 || w==0)
17                 K[i][w] = 0;
18             else if (wt[i-1] <= w)
19                     K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);
20             else
21                     K[i][w] = K[i-1][w];
22         }
23     }
24   return K[n][W];
25 }
```

La complexité de cette fonction est de $\mathcal{O}(n*W)$, avec n le nombre total d'objets, et W le poids max supporté par le sac.

Problème du sac à dos en 3D

Solution récursive

La fonction suivante F présente la solution récursive du problème, en ayant la liste d'objets (Obj1, Obj2, ... Objn), les poids(P1,P2, ... Pn), les volumes : 3ième critère (V1,V2, ... , Vn),l'utilité de chaque objet(U1,U2, ... ,Un), et le poids max du sac (Pmax) ainsi que le volume max (Vmax).

$$F(Objn, Pmax, Vmax) = max \begin{cases} F(Objn-1, Pmax, Vmax) & \text{objet } !choisi \\ F(Objn-1, Pmax-Pn, Vmax-Vn)+Un & \text{objet } choisi \end{cases}$$

On présente dans ce qui suit l'implémetation en C de cette solution. Le complexité de cette fonction est de $\mathcal{O}(2^{\text{n}})$, avec n le nombre total d'objets.

```
1  int sac_a_dos3d_recursive(int w, int wt[],int v,int vol[], int val[], int n){
2      if(n==0 || w==0)
3      {
4          return 0;
5      }else
6      {
7         if(wt[n-1]>w || vol[n-1]>v )
8          {
9              return sac_a_dos3d_recursive(w, wt,v,vol,val, n-1);
10         }else
11         {
12             return max(val[n-1] + sac_a_dos3d_recursive(w - wt[n-1], wt,v-vol
                   [n-1],vol, val, n-1),
13                     sac_a_dos3d_recursive(w, wt,v,vol, val, n-1));
14         }
15     }
16 }
```

Solution en utilisant la programmation dynamique

L'algorithme suivant présente une adaptation de l'algorithme 2D en utilisant un tableau de trois dimensions : K[nombre d'objets, Poids max, Volume max] ;

```
1  int sac_a_dos3d_dynamique(int W, int wt[],int P,int vol[], int val[], int n)
2  {
3      int i,w,k;
4      int ***K;
5
6      K = (int ***) malloc (sizeof(int ***)*(n+1));
7      for (int h = 0; h < n+1; h++)
8      {
9       K[h] = (int **) malloc(sizeof(int*)*(W+1));
10      for (int r = 0; r < (W+1); r++)
11      {
12        K[h][r] = (int *) malloc(sizeof(int)*(P+1));
13      }
14     }
15
16     for (i = 0; i <= n; i++)
17     {
18         for (w = 0; w <= W; w++)
19         {
20             for(k = 0; k <= P ; k++)
21              {
22                  if (i==0 || w==0 || k==0)
23                  {
```

```
24                      K[i][w][k] = 0;
25                  }else if (wt[i−1] <= w && vol[i−1]<=k)
26                  {
27                      K[i][w][k] = max(val[i−1] + K[i−1][w−wt[i−1]][k−vol[i−1]],
                            K[i−1][w][k]);
28                  }else
29                  {
30                      K[i][w][k] = K[i−1][w][k];
31                  }
32
33              }
34
35          }
36      }
37      return K[n][W][P];
38  }
```

La complexité de cette adaptation est de $\mathcal{O}(n * W * V)$, avec W (réspectivement V) le poids max (réspectivement le volume max) supporté par le sac à dos.

Tests et Exécution

Spécification de la machine

Les temps d'exécution des algorithmes diffèrent, relativement, d'une machine à une autre , et cela à cause de différents paramètres machine (Hardware). Le tableau suivant résume les spécifications de la machine dont laquelle les tests ont été lancés.

Marque	Lenovo
Laptop	IdeaPad G510
Processeur	Intel Core i5-4200U
Fréquence de l'horloge	2.5 GHz
Mémoire Cache	3 Mo

Tests de la version 2 dimensions

Le tableau suivant présente quelques résultats du temps d'exécution de l'algorithme qui résout la version 2D du problème du sac à dos ;
On remarque bien que le temps d'exécution de la version récurrence est relatif au nombre total d'objets, et croît de manière exponentielle. Par contre, le temps d'exécution de la version qui utilise le tableau 2D dépend des deux paramètres (nombre d'objets et le Poids max à ne pas dépasser), et qui croît de manière linéaire.
On déduit à partir du tableau précédent que la 2ème version est plus efficace que

Nombre d'objets	Poids max	Temps d'éxècution (ms)	
		Version Récursive	Version Programmation Dynamique
10	50	145	24
20	100	233	48
30	250	3671474	106
30	500	10748652	187
30	1000	10759105	340

la version recursive.

Tests de la version 3 dimensions

Test des valeurs de l'énoncé du TP

Le tableau suivant présente les 20 objets données dans l'exemple du TP ;

NumObjet	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Poids	20	30	50	20	40	60	30	10	14	36	72	86	05	03	07	23	49	57	69	12
Volume	10	15	25	10	20	30	15	05	07	18	36	43	03	02	04	12	25	29	35	06
Utilité	06	08	14	06	13	17	10	04	05	11	26	35	02	01	02	07	15	17	30	03

Les résultats concernant ce jeu de test sont les suivants :
Les objets retenus sont : 1, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19. Et la valeur maximale cumulée est de 183.

Autres jeux de tests

Le tableau en dessus présente quelques résultats du temps d'exécution de l'adaptation de l'algorithme résout la version 3D du problème du sac à dos ;

Nombre d'objets	Poids max	Volume max	Temps d'éxècution (ms)	
			Version Récursive	Version Programmation Dynamique
10	20	20	160	264
20	20	20	175	402
30	25	25	328	863
30	50	50	1643	2352
30	100	100	11467	5461
30	250	250	855975	24991
30	500	500	11291327	100188
30	100	500	13444	18628
20	520	300	13235	42796

On remarque bien que les performances de l'algorithme utilisant la programmation dynamique ne sont pas aussi intéressante que la solution récursive. Tout dépends des trois paramètres (nbr Objets, Pmax et Vmax). Il ne faut pas négliger le fait que la version 2 de l'algorithme crée un tableau de taille N*(Pmax+1)*(Vmax+1).

Le graphe suivant montre bien le fait que la taille du tableau 3D influence sur le temps d'éxécution, mais dès que le nombre d'objets augmente, on aura toujours une solution plus rapide avec l'algorithme dynamique.
En rouge : version récursive.
En bleu : version algorithme dynamique.