Théorie des Programmes & Résolution de Problèmes - TP1

Par:
Kerim Yahia
Khelil Mohamed Wael
SIL2 TPGO 2018/2019

Le problème du sac à dos

l'énoncé

" Etant donné plusieurs objets possédant chacun un poids et une valeur et étant donné un poids maximum pour le sac, quels objets faut-il mettre dans le sac de manière à maximiser la valeur totale sans dépasser le poids maximal autorisé pour le sac ? "

Formulation mathématique

Soient :

- $n \in \mathbb{N}$ indique le nombre des objets,
- $X \in \{0, 1\}^n$ dont chaque élément x_i vaut 1 dans le cas où le i eme objet et mis dans le sac, il vaut 0 sinon,
- $P \in \mathbb{R}_+^n$ dont chaque élément p_i indique le poids du i eme objet,
- $V \in \mathbb{R}_+^n$ dont chaque élément v_i indique la valeur du i eme objet,
- $P_{max} \in \mathbb{R}_+^n$ le poids maximal autorisé pour le sac.

La fonction objectif sera :

$$\phi(X) = max\, X^t V = max \sum_{i=1}^{n} x_i v_i$$

La contrainte est :

$$X^t P \leq P_{max} \text{ ou bien } \sum_{i=1}^{n} x_i p_i \leq P_{max}$$

Les solutions algorithmique

```
struct _Item{
   unsigned int weight;
   unsigned int value;
}typedef Item; //contient les informations d'un objets

struct _Data{
   unsigned int numberOfObjects;
```

```
   Object* objects;
}typedef Data; //contient le nombre et le tableau d'objets

struct _Solution{
   bool *X;
   unsigned int value;
}typedef Solution; //contient la solution et la valeur maximale
```

l'algorithme récursif

```
Solution* f(unsigned int n,unsigned int maxWeight) {

   Solution *f1;
   Solution *f2;

   if (n <= 0)
       return initSolution(data->numberOfObjects);
   else {
       //On suppose que l'object i n'est pas pris
       f1 = f(n - 1, maxWeight);
       if(maxWeight < data->objects[n - 1].weight)
           return f1;
       else{
           //On suppose que l'object i est pris
           f2 = f(n - 1, maxWeight - data->objects[n - 1].weight);
           f2->value += data->objects[n - 1].value;
       }
       if (f1->value >= f2->value) {
           free(f2);
           return f1;
       } else {
           f2->X[n-1] = true;
           free(f1);
           return f2;
       }
   }
}
```

Programmation dynamique

Fonction utilitaires

```
Solution* initSolution(unsigned int length){
   Solution *sol = malloc(sizeof(Solution));
   sol->X = calloc(length, sizeof(bool));
   sol->value = 0;
   return sol;
}
```

```
Solution* cloneSolution(Solution* sol){
```

```
   Solution *solC = malloc(sizeof(Solution));
   solC->value = sol->value;
   solC->X = malloc(data->numberOfObjects*sizeof(bool));
   for(unsigned int i = 0; i < data->numberOfObjects ;i++) {
       solC->X[i] = sol->X[i];
   }
   return solC;
};
```

l'algorithme

```
Solution* fDynamique(unsigned int n,unsigned int maxWeight) {

   Solution *tabF[n + 1][maxWeight + 1];
   Solution *f1;
   Solution *f2;

      for(unsigned int i = 0; i < n + 1; i++){
          for(unsigned int j = 0; j < maxWeight + 1; j++) {
              if (i == 0) {
               tabF[i][j] = initSolution(data->numberOfObjects);
           }
           else {
               //On suppose que l'object i n'est pas pris
               if(j < data->objects[i - 1].weight) {
                   tabF[i][j] = cloneSolution(tabF[i - 1][ j]);
               }else{
                   //On suppose que l'object i est pris
                   f1 = cloneSolution(tabF[i-1][j]);
                   f2 = cloneSolution(tabF[i-1][j - data->objects[i-1].weight]);
                   f2->value = f2->value + (data->objects[i - 1].value);
                   if (f1->value >= f2->value) {
                       tabF[i][j] = f1;
                       free(f2);
                   } else {
                       tabF[i][j] = f2;
                       tabF[i][j]->X[i - 1] = true;
                       free(f1);
                   }
               }
           }
       }
   }
   return tabF[n][maxWeight];
}
```

Sac à dos multidimensionnel (Camion 2-dimension)

l'énoncé

“ Etant donné plusieurs objets possédant chacun un poids, un volume et une valeur et étant donné un poids maximum un volume maximum pour le camion, quels objets faut-il mettre dans le camion de manière à maximiser la valeur totale sans dépasser le poids maximal et le volume maximal autorisé pour le camion ? “

Formulation mathématique

Soient :

- $n \in \mathbb{N}$ indique le nombre des objets,
- $X \in \{0, 1\}^n$ dont chaque élément x_i vaut 1 dans le cas où le i eme objet et mis dans le camion, il vaut 0 sinon,
- $P \in \mathbb{R}_+^n$ dont chaque élément p_i indique le poids du i eme objet,
- $W \in \mathbb{R}_+^n$ dont chaque élément W_i indique le volume du i eme objet,
- $V \in \mathbb{R}_+^n$ dont chaque élément v_i indique la valeur du i eme objet,
- $P_{max} \in \mathbb{R}_+^n$ le poids maximal autorisé pour le camion.
- $W_{max} \in \mathbb{R}_+^n$ le volume maximal autorisé pour le camion.

La fonction objectif sera :

$$\phi(X) = max\, X^t V = max \sum_{i=1}^{n} x_i v_i$$

La contrainte est :

$$\begin{cases} X^t P \leq P_{max} \\ X^t W \leq W_{max} \end{cases}$$

Solution algorithmique

La structure modifiée est Item, le champ volume est ajouté à la structure _Item ancienne.

```
struct _Item{
   unsigned int weight;
   unsigned int volume;
   unsigned int value;
}typedef Item;
```

La fonction est quasiment la même sauf que les paramètres et les conditions d'arrêt ont été changés.

```
Solution* f(unsigned int n,unsigned int maxWeight, unsigned int maxVolume) {

   Solution *f1;
   Solution *f2;

   if (n <= 0)
       return initSolution(data->numberOfObjects);
   else {
       //On suppose que l'object i n'est pas pris
       f1 = f(n - 1, maxWeight, maxVolume);
       if((maxWeight < data->objects[n - 1].weight)||(maxVolume < data->objects[n -
1].volume))
           return f1;
       else{
           //On suppose que l'object i est pris
           f2 = f(n - 1, maxWeight - data->objects[n - 1].weight, maxVolume -
data->objects[n - 1].volume);
           f2->value += data->objects[n - 1].value;
       }
       if (f1->value >= f2->value) {
           free(f2);
           return f1;
       } else {
           f2->X[n-1] = true;
           free(f1);
           return f2;
       }
   }
}
```

Programmation dynamique

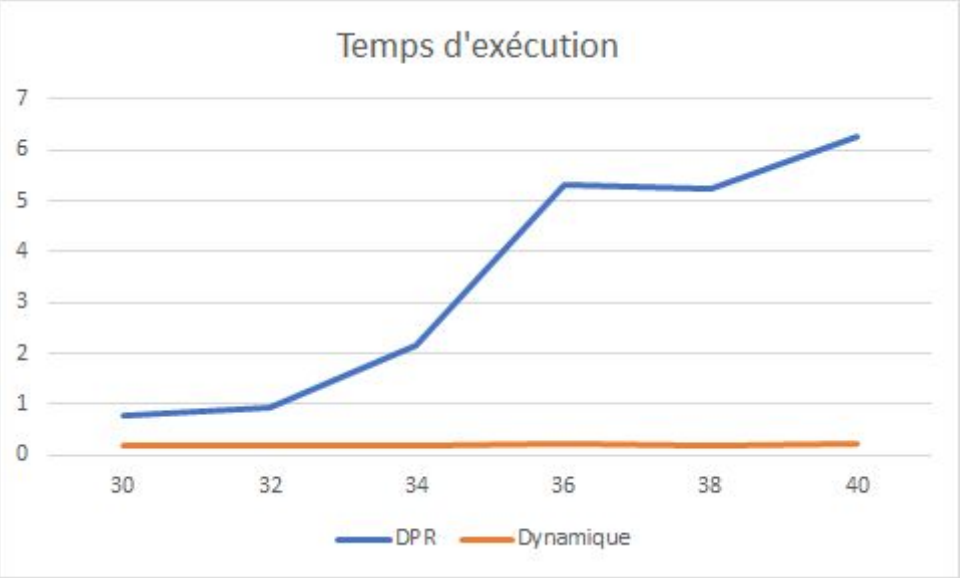
```
Solution* fDynamique(unsigned int n,unsigned int maxWeight, unsigned int maxVolume) {

   Solution *f2;
   for (unsigned int i = 0; i < n + 1; i++) {
       for (unsigned int j = 0; j < maxWeight + 1; j++) {
           for(unsigned int k = 0; k < maxVolume + 1; k++) {
               if (i == 0) {
               } else {
                   if ((j < data->objects[i - 1].weight)
                       ||(k < data->objects[i - 1].volume)){
                       //Volume ou poids insufissant pour ajouter cet objet
                       tabF[i].tab[j][k] = tabF[i - 1].tab[j][k];
                   } else {
                       //le cas ou l'objet est pris
                       f2 = tabF[i - 1].tab
                               [j - data->objects[i - 1].weight]
                               [k - data->objects[i - 1].volume];
                       //test sur le gain total dans les deux cas
                       if ((tabF[i - 1].tab[j][k]->value) >=
                               (f2->value + (data->objects[i - 1].value))) {
```

```
                        //l'object i n'est pas pris
                        tabF[i].tab[j][k] = tabF[i - 1].tab[j][k];
                    } else {
                        //l'object i est pris
                        cloneSolution(tabF[i].tab[j][k], f2, i);
                        tabF[i].tab[j][k]->value = f2->value
                                                + data->objects[i - 1].value;
                        tabF[i].tab[j][k]->X[i - 1] = true;
                    }
                }
            }
        }
    }
    return tabF[n].tab[maxWeight][maxVolume];
}
```

Résultat

Afin de comparer entre les deux méthodes, on va générer des échantillons aléatoires en augmentant à chaque fois le nombre d'objets. Le poids et le volume d'un objet ne dépasse pas la valeur 100.
On faisons des tests avec des échantillons de taille entre 30 et 40 , on obtient le graphe suivant :



Interprétation

On augmentons la taille d'échantillons, On remarque bien que la méthode Dynamique est plus performante que la méthode DPR, car on évite les opérations déjà calculé on les sauvegardons dans un tableau.