REPUBLIQUE DU CAMEROUN

Paix – Travail – Patrie

MINISTERE DES ENSEIGNEMENTS SUPERIEURS

REPUBLIC OF COMEROON

Peace – Work – Fatherland

MINISTRY OF HIGHER EDUCATION

RAPPORT UE INF 4017 :

PROBLEME DU SAC A DOS EN UTILISANT LA PROGRAMMATION DYNAMIQUE

Membres du groupe 3:

- KANYEP FONKOU DARRY WILFRIED (19M2551)
- MAKOLLE EBONGUE NDJOH EDMOND GHISLAIN (17B2680)
- NCHOUWET MFOUAPON KUNTZ STEPHANE (19M2214)
- NZOUETENG TCHUENTE MICAËLLE GRÂCE (19M2193)
- TAKUISSU NEKAM MARTIAL (19M2446)
- TSAPI KADJI HILBERT CHRYSHOSTOME (19M2359)

PROFESSEUR : Dr PAULIN MELATAGIA

Année scolaire 2022-2023

Table des matières

DESCRIPTION DU PROBLÈME

Le problème du sac à dos ou knapsack problem en anglais est un problème algorithmique où on dispose d'un sac ne pouvant supporter qu'un certain poids et étant donné plusieurs objets possédant chacun un poids et une valeur, nous recherchons les objets à mettre dans le sac de manière à maximiser la valeur totale du sac sans dépasser le poids maximal autorisé par le sac.

TYPOLOGIE DU PROBLÈME

Le problème du sac à dos est un problème d'**optimisation combinatoire** où on est amené à trouver la meilleure solution ou la solution optimale parmi un ensemble de solutions réalisables. Il faut donc trouver parmi toutes les solutions possibles de remplissage des objets dans le sac, la solution où on aura la plus grande valeur sans dépasser le poids du sac.

APPLICATION RÉELLE DU PROBLÈME

Le problème du sac à dos à plusieurs utilités dans la vie réelle notamment concernant **le domaine de la logistique** où il y voit plusieurs utilités parmi lesquels :

- Le chargement des produits dans un contenaire de bateau
- Le chargement des bagages dans la soute d'un avion
- Le remplissage des fournitures dans un camion de déménagement.
- Le remplissage d'un sac de voyage avec des produits de nécessité primordiale.

Dans tous les exemples cités ci haut, on est amené à remplir un récipient de nos objets (contenaire, soute, camion etc ...) avec des produits sans toutefois dépasser le poids du récipient, d'où il est primordial afin de minimiser les pertes de choisir parmi les produits ceux qui vont permettre au récipient d'avoir une valeur maximale.

FORMALISATION DU PROBLÈME

Supposons que notre sac à dos à un poids maximal **W** et nous cherchons comment y mettre **n** objets. Pour chaque objet **i**, nous avons un poids $\mathbf{w_i}$ et un valeur $\mathbf{v_i}$ **.** On définit le vecteur **X = { x_1, x_2, ….., x_n }** avec pour chaque variable $\mathbf{x_i}$ associée à un objet **i** de la façon suivante :

- $\mathbf{x_i = 1}$ si l'objet i est mis dans le sac
- $\mathbf{x_i = 0}$ si l'objet i n'est mis dans le sac

Le problème consiste donc à la recherche des valeurs du vecteur X satisfaisant la condition :

$$\max \sum_{i=1}^{n} v_i x_i$$

sachant que : $\sum_{i=1}^{n} w_i x_i \leq W$ avec $x_i \in \{0,1\}$

Caractérisation de la sous structure optimale :

Soit P(i,j), le gain maximale généré par le choix des i premiers objets dont le poids ne dépasse pas j avec $j \in \{1,..,W\}$ alors pour calculer P(i,j) on a trois cas :

- **Si** i = 0 **ou** j = 0 : Cela implique qu'on veut insérer soit 0 objets dans un sac de poids j, soit insérer i objets dans un sac de poids 0 alors dans ce cas on aura donc :

$$\mathbf{P(i,j) = 0}$$

- **Si** j < W(i) **:** Cela implique qu'en ajoutant l'objet i dans le sac de poids j alors le poids du sac sera excédé, l'objet i ne sera donc pas choisi et la capacité du sac reste inchangée d'où :

$$\mathbf{P(i,j) = P(i-1, j)}$$

- **Si j** >= W(i) : Cela implique que l'objet i peux être ajouté dans le sac, selon le fait qu'il a une valeur supérieur à celle de P(i-1,j) on aura donc :

$$\mathbf{P(i,j) = \max\{P(i-1, j), P(i-1, j-W(i)) + V(i)\}}$$

ALGORITHME DE RÉSOLUTION

L'algorithme de résolution du problème de sac à dos est donc :

Algorithme **pd_sac_à_dos** (v : tableau de valeurs , w : tableau des poids , W : poids total , n : nombre d'objets)

Précondition : Avoir au préalable le tableau de poids, de valeurs et le poids total du sac.

Post condition : L'algorithme retourne la valeur maximale et aussi un tableau X de taille n spécifiant si un objet i est sélectionné ou pas

Début :

```
// on crée une matrice P de taille n+1 * w+1
Pour i allant de 0 à n faire
    Pour j allant de 0 à W faire
        Si i = 0 ou j = 0 alors
            P[i][j] ← 0 ;
        Sinon
            SI ( j – w[i] < 0 ) alors
                P[i][j] ← P[i-1][j] ;
            Sinon
                P[i][j] ← max(P[i-1][j] , P[i-1][j – w[i]] + v[i] ;
            Finsi
        Finsi
    FinPour
FinPour
// la valeur maximale est égale à la dernière case de la matrice
Valeur ← P[n][W] ;
```

```
        // On parcourt la matrice pour avoir les objets sélectionnés
        i ←n , nombre ← P[n][W], trouve ← false ;
        Tant que i > 0 faire
                i ← i -1 ;
                Pour k allant de 0 à W faire
                        Si P[i][k] = nombre alors
                                Trouve ← true ;
                                Break ;
                        Finsi
                FinPour
                Si !trouve alors
                        X[i] ← 1 ;
                        nombre ← nombre – v[i+1]
                Sinon
                        X[i] ← 0 ;
                Finsi
                Trouve ← false ;
        FinTantque

        Retourner (X, valeur) ;
Fin
```

COMPLEXITÉ DE L'ALGORITHME

En temps

Dans cet algorithme, on a 4 boucles qui s'effectuent.

Les 2 premières concernent le remplissage de la matrice P, qui a n lignes et W colonne. Pour ce remplissage, on a une complexité de :

f1(n, W) = (W+1)(n+1)

(W+1) représente le parcours des colonnes et (n+1) le parcours des lignes.

Les 2 dernières concernent l'affichage des objets qui ont étés mis dans le sac. La boucle **Tant que** s'effectuent dans le pire des cas n fois car sa condition dépend de i qui est initialisé à n et se décrémente de 1 à chaque itération. La boucle interne à la boucle **Tant que** elle s'effectue (W+1) fois. La complexité de cette deuxième partie revient donc à :

f2(n, W) = n(W+1)

En conclusion, cette algorithme a donc une complexité sensible à :

f(n, W) = f1(n, W) + f2(n, W) = 2*W*n + W + n + 2

Soit **f(n, W)** ∈ **O(nW)**

En espace

Sachant que nous manipulons les entiers et qu'un entier équivaut à 4 octets mémoire on a donc :

- Valeur: 4 octets
- P étant une matrices d'entier de taille n+1*W+1 on aura donc **4(n+1)(W+1) octets** en mémoire donc 4nW+4n+4W+4
- X étant un tableau d'entier de taille n donc **4*n octets en mémoire**

La complexité en espace est donc **O(4nw + 8n + 4w + 8) = O(nW)**

IMPLÉMENTATION EN C++ DE L'ALGORITHME

```
void sac_a_dos(vector<int> &w , vector<int> &v , int n , int W){
    int valeur = 0 ;
    //on crée la matrice de taille n+1 * W+1;
    vector<vector<int>> P(n + 1,vector<int>(W + 1));
    vector<int> X(n);

    for(int i = 0 ; i < n+1 ; i++){
        for(int j = 0 ; j < W+1 ; j++){
            if(i == 0 || j == 0) {
                P[i][j] = 0;
            }
            else if( j - w[i-1] < 0 ){
                P[i][j] = P[i-1][j];
            }
            else {
                P[i][j] =  max(P[i-1][j], P[i-1][j - w[i-1]] + v[i-1]);
            }
        }
    }

    for(int i = 0 ; i < n+1 ; i++){
        for(int j = 0 ; j < W+1 ; j++){
            cout << P[i][j] << " ";
        }
        cout <<endl;
    }
    valeur = P[n][W];

    //partie pour afficher les objets selectionnés
    int i = n , nombre = P[n][W];
    bool trouve = false;
    while( i > 0 ){
        i = i -1;
        for(int k =0 ;k < W+1 ; k++) {
            if(P[i][k] ==  nombre){
                trouve = true;
                break;
            }
        }

        if( !trouve){
            X[i] = 1;
            nombre -= v[i];
        } else {
            X[i] = 0;
        }
        trouve = false;

    }

    cout << endl << "la valeur maximale pouvant etre obtenue est "<< valeur <<endl;

    //affichage
    for(int i = 0 ; i < n  ; i++){
        cout << "objet " << i+1 << " : poids = " << w[i] << " valeur = " << v[i] << " nombre = " << X[i] << endl;
    }

}
```

EXPÉRIMENTATION

Expérimentation 1 :

Poids du sac : **25Kg**

Objet	1	2	3	4
Valeur	7	4	3	3
Poids	13	12	8	10

Solution optimale :

- Objets à prendre : **objets 1 et 2 (X = { 1 , 1, 0,0})**
- Valeur maximale : **11**

Solution du code :

```
Veuillez entrer le poids total de votre sac : 25
Veuillez entrer votre nombre d'objets : 4

veuillez entrer le poids de l'objet 1 : 13
veuilez entrer la valeur de l'objet 1 : 7

veuillez entrer le poids de l'objet 2 : 12
veuilez entrer la valeur de l'objet 2 : 4

veuillez entrer le poids de l'objet 3 : 8
veuilez entrer la valeur de l'objet 3 : 3

veuillez entrer le poids de l'objet 4 : 10
veuilez entrer la valeur de l'objet 4 : 3

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 7 7 7 7 7 7 7 7 7 7 7 7 7
0 0 0 0 0 0 0 0 0 0 0 0 4 7 7 7 7 7 7 7 7 7 7 7 11
0 0 0 0 0 0 0 0 3 3 3 3 4 7 7 7 7 7 7 7 7 10 10 10 10 11
0 0 0 0 0 0 0 0 3 3 3 3 4 7 7 7 7 7 7 7 7 10 10 10 10 11

la valeur maximale pouvant etre obtenue est 11
objet 1 : poids = 13 valeur = 7 nombre = 1
objet 2 : poids = 12 valeur = 4 nombre = 1
objet 3 : poids = 8 valeur = 3 nombre = 0
objet 4 : poids = 10 valeur = 3 nombre = 0

Process returned 0 (0x0)   execution time : 31.058 s
Press any key to continue.
```

Expérimentation 2 :

Poids du sac : **8Kg**

Objet	1	2	3	4
Valeur	1	2	5	6
Poids	2	3	4	5

Solution optimale :

- Objets à prendre : **objets 2 et 4 (X = { 0 , 1, 0,4})**
- Valeur maximale : **8**

Solution du code :

```
Veuillez entrer le poids total de votre sac : 8
Veuillez entrer votre nombre d'objets : 4

veuillez entrer le poids de l'objet 1 : 2
veuilez entrer la valeur de l'objet 1 : 1

veuillez entrer le poids de l'objet 2 : 3
veuilez entrer la valeur de l'objet 2 : 2

veuillez entrer le poids de l'objet 3 : 4
veuilez entrer la valeur de l'objet 3 : 5

veuillez entrer le poids de l'objet 4 : 5
veuilez entrer la valeur de l'objet 4 : 6

0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1
0 0 1 2 2 3 3 3 3
0 0 1 2 5 5 6 7 7
0 0 1 2 5 6 6 7 8

la valeur maximale pouvant etre obtenue est 8
objet 1 : poids = 2 valeur = 1 nombre = 0
objet 2 : poids = 3 valeur = 2 nombre = 1
objet 3 : poids = 4 valeur = 5 nombre = 0
objet 4 : poids = 5 valeur = 6 nombre = 1

Process returned 0 (0x0)   execution time : 19.395 s
Press any key to continue.
```