

LE PROBLÈME DU SAC À DOS

RAPPORT DE TP1

November 12, 2018

MAHAMDI Mohammed
ADRAO Nassim

Table des Matières

1	Introduction	3
1.1	Approche de solution	3
1.2	Principe de la programmation dynamique	3
2	Implémentation de la solution	5
2.1	Approche choisie	5
2.2	Les équations de récurrence	5
2.3	Processus	5
2.4	Calcul de la complexité	11
3	Conclusion	12

Table des Figures

2.1	Page d'accueil	6
2.2	Saisie de la taile de sac à dos	7
2.3	Saisie des donnée concernant les objets	8
2.4	Calcul de gain maximal	9
2.5	Affichage d'une combinaison des éléments choisi satisfaisant les contraintes	10

Chapter 1

Introduction

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.

1.1 Approche de solution

Pour résoudre ce problème, on peut utiliser la solution classique ou glouton consistant à essayer tous les cas possibles. Cela est innéfice pour des valeurs de n supérieur à 8 par exemple.

C'est pour cela que la programmation dynamique est la solution la plus adéquate, et c'est le but de ce TP1.

1.2 Principe de la programmation dynamique

La solution récursive est la plus évidente mais la plus coûteuse, la programmation dynamique est par conséquent une amélioration qui consiste à sauvegarder les valeurs des transitions déjà calculées pour ne pas répéter les calculs inutiles.

Il existe deux approches:

- **Approche ascendante:** Pour calculer le n ième élément, on commence par calculer le premier élément, puis le deuxième, ... jusqu'à arriver au dernier élément.

- **Approche descendante:** Pour calculer le n ième élément, on calcule l'élément n-1 , puis l'élément n-2 , ... jusqu'à arriver au premier élément.

Chapter 2

Implémentation de la solution

2.1 Approche choisie

On a utilisé l'approche ascendante (voir introduction)

2.2 Les équations de récurrence

$$P_{ij} = \begin{cases} 0 & \text{si } i = 0 \text{ ou } j = 0 \\ P_{i-1j} & \text{si } j < w_i, i > 0 \\ \max\{P_{i-1,j}, P_{i-1,j-w_i} + v_i\} & \text{autre cas} \end{cases}$$

2.3 Processus

1. On lit la capacité du sac à dos, nommée *maxWeight*.
2. On lit n objets et pour chaque objet on introduit le poids et le gain correspondant sous forme de tableau comme le montre la figure suivante:
3. On construit une matrice nommée *matrix* de $n+1$ lignes (la 1 ère ligne d'indice 0 contient des 0 partout, utile juste pour l'utilisation des équations de référence) et $\text{maxWeight}+1$ colonne (la 1 ère colonne d'indice 0 contient des 0 partout, utile juste pour l'utilisation des équations de référence)
4. On remplit cette matrice en utilisant les équations de référence (voir section 2.2).
5. après la rempli de cette matrice , on trouve le gain maximum dans *matrix[n][maxWeight]*.

6. Pour la recherche des objets choisis, on procède à la comparaison des valeurs de la dernière colonne comme suit

- (a) On initialise un pointeur entier k à n, et un pointeur col à maxWeight.
- (b) on compare le gain matrix[k][col] avec matrix[k+1][col].
- (c) si matrix[k][col] != matrix[k+1][col] alors élément k choisi
- (d) Soustraire le poids w_i de l'élément choisi de col, on décrémente le k et si k != 1 aller à 2.

Les figures suivantes montrent les résultats de l'implémentation :

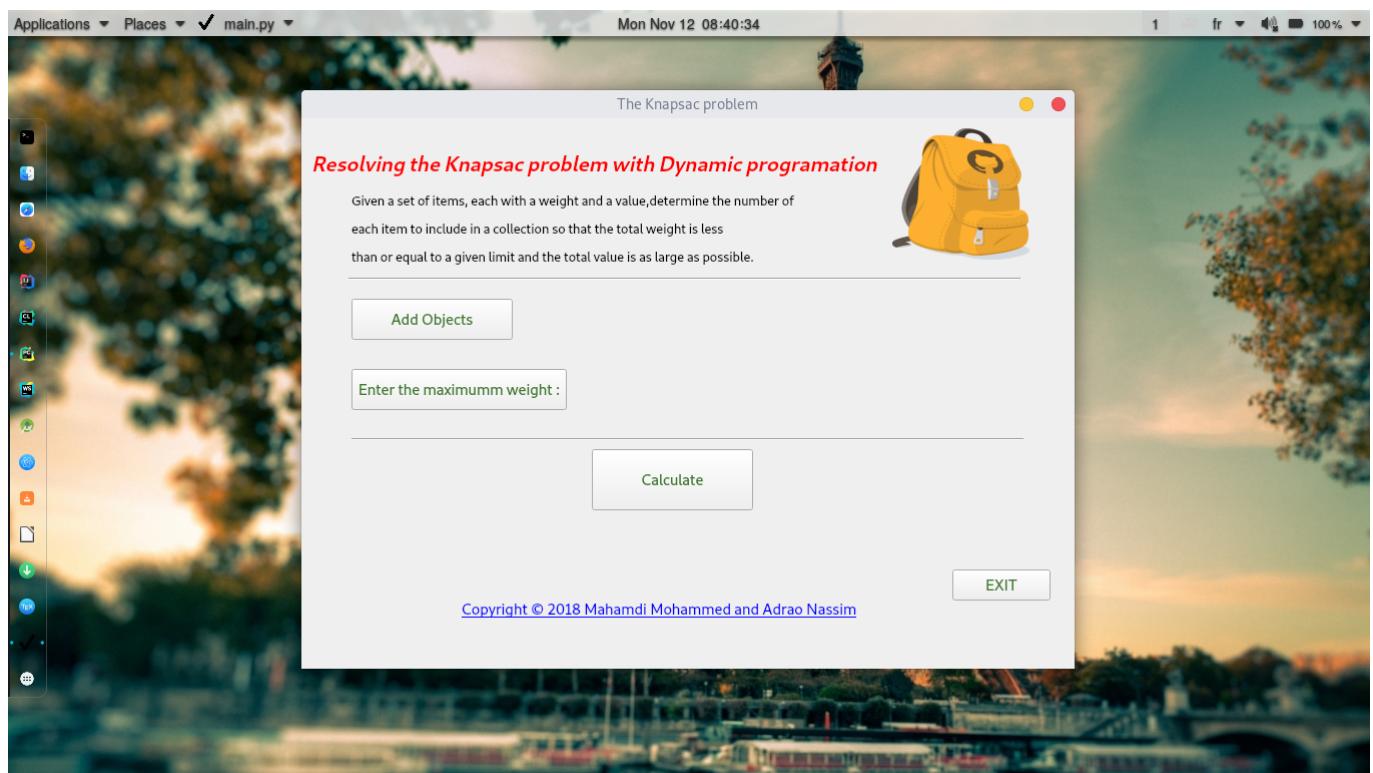


Figure 2.1: Page d'accueil

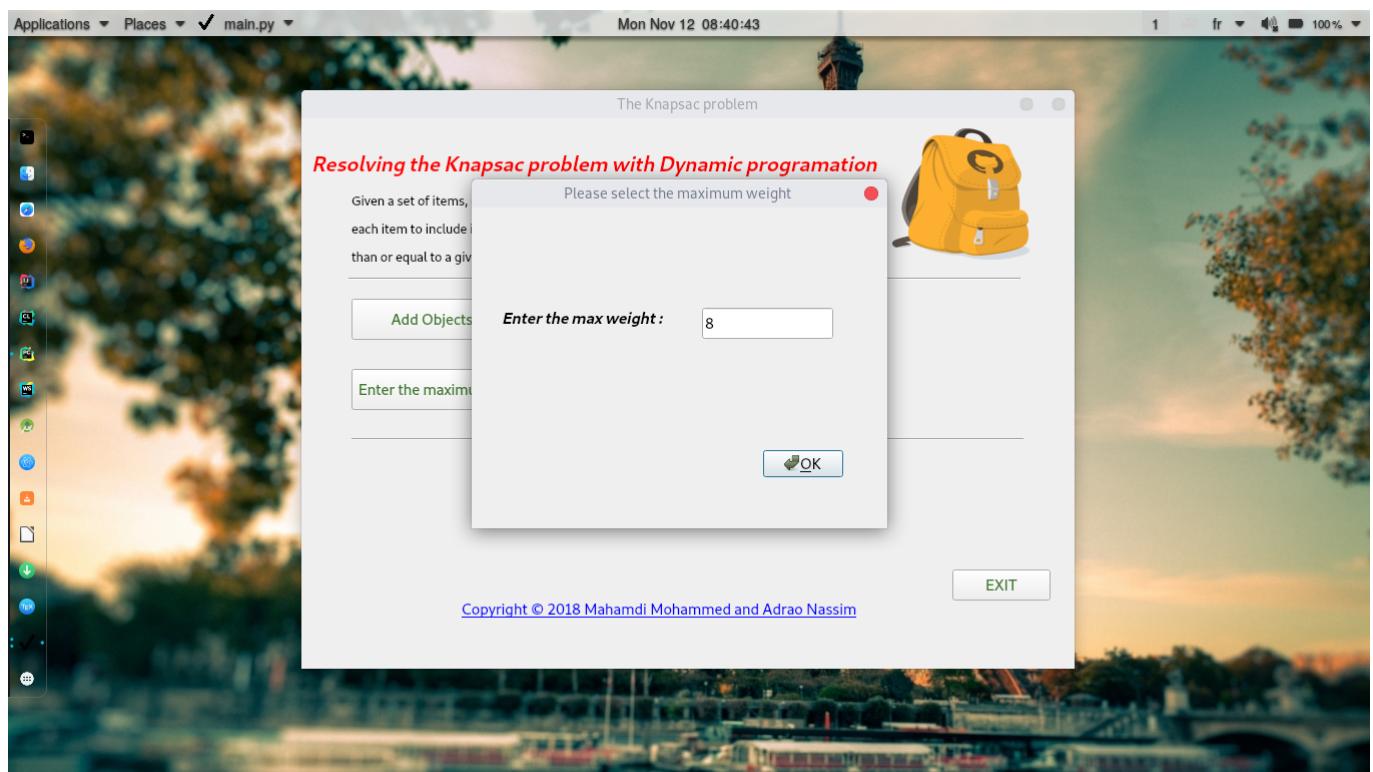


Figure 2.2: Saisie de la taile de sac à dos

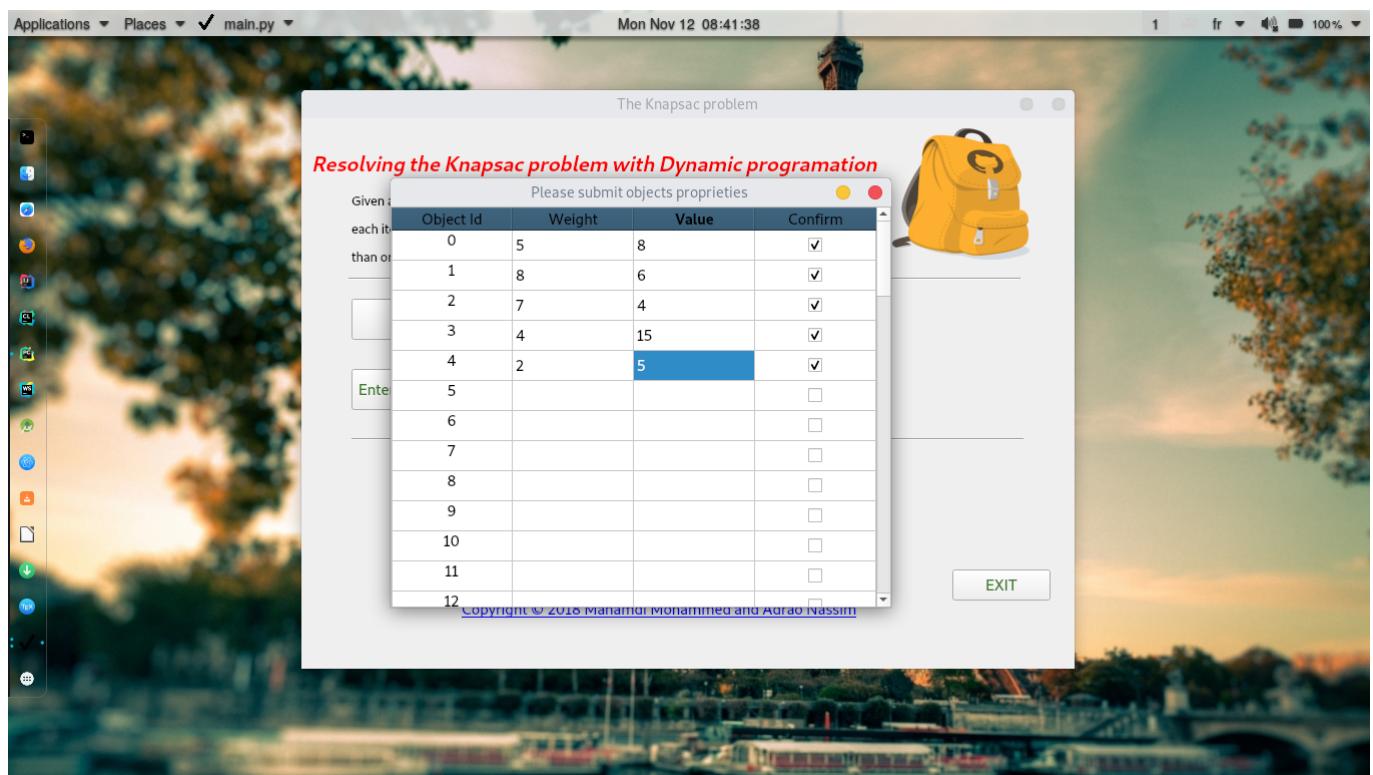


Figure 2.3: Saisie des donnée concernant les objets

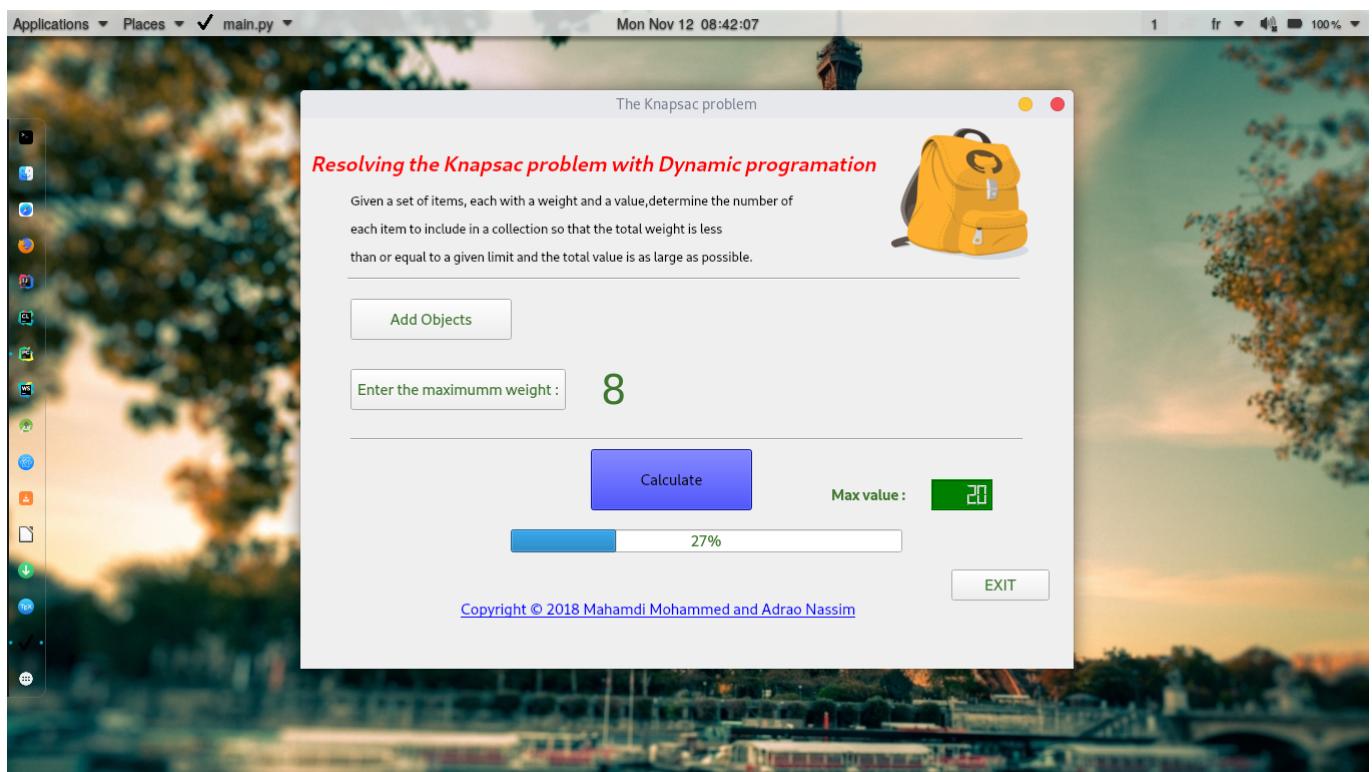


Figure 2.4: Calcul de gain maximal

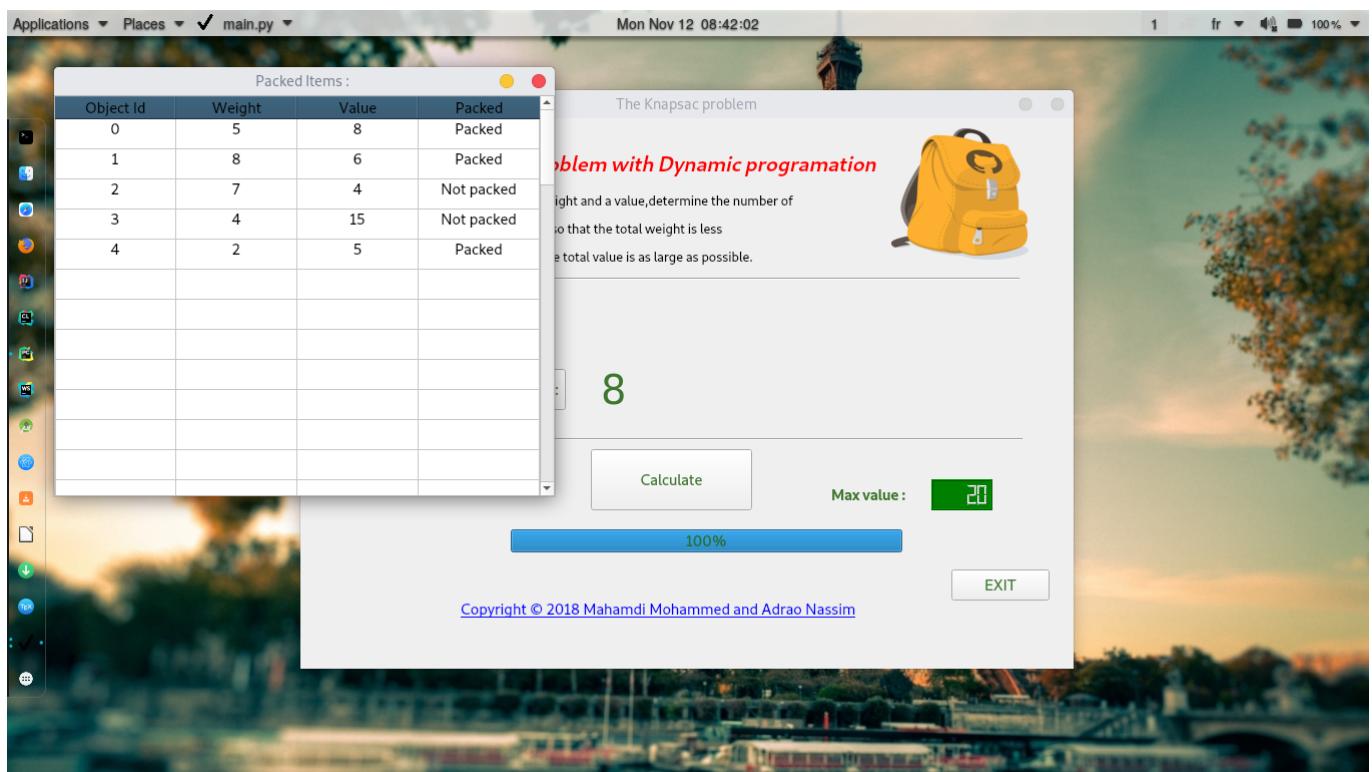


Figure 2.5: Affichage d'une combinaison des éléments choisi satisfaisant les contraintes

2.4 Calcul de la complexité

La complexité de l'algorithme : $O(m^2)$

preuve : On doit remplir la matrice de n lignes et p colonnes, donc la complexité est $O(m^2)$ tel que $m=\max(n,p)$; Puis on parcourt la dernière colonne (complexité= $O(n)$) n est le nombre de lignes

Donc la complexité finale sera : $\max(O(m^2), O(n)) = O(m^2)$

Chapter 3

Conclusion

La programmation dynamique sert à résoudre de très grand nombre de problèmes, on remarque cela dans ce TP.

La complexité de la programmation dynamique est polynomial tout en gardant la simplicité de la récursivité avec une complexité base

La programmation dynamique est un cas de l'intelligence artificielle

Le problème de sac à dos est l'un des problèmes classiques en informatique, qui reste jusqu'ici valable pour les problèmes qui sont résolu en conséquent.