LE PROBLÈME DU SAC À DOS

RAPPORT DE STAGE

November 7, 2018

Table des Matières

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 possible. Cela est innéficace 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 PROGRAMMTION 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 déja calculer pour ne pas répéter les calculs inutile.

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

(les equations mathématiques à écrire)

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'utiliation des équations de réferrence) et maxWeight+1 colone (la 1 ère colone d'indice 0 contient des 0 partout, utile juste pour l'utiliation des équations de réferrence)
4. On remplit cette matrice en utilisant les équations de réferrence (voir section 2.2).
5. après la rempli de cette matrice , on trouve le gain maximun dans *matrix[n][maxWeight]*.