Examen de
Complexité

Durée : 1 h
Tous documents autorisés
4 novembre 2016

— Les réponses aux questions doivent être dûment justifiées.
— Les programmes demandés peuvent être écrits avec le langage de programmation de votre choix.
— Le barème est donné à titre indicatif.

Problème du sac à dos non borné

Le problème du sac à dos non borné (*Unbounded Knapsack*) est similaire au problème du sac à dos classique, mais on peut utiliser plusieurs exemplaires (une quantité non bornée) du même objet pour remplir le sac. Il s'agit donc de choisir une collection d'objets, issus d'un ensemble de n prototypes d'objets $X = \{x_1, \ldots, x_n\}$ (que l'on peut donc réutiliser autant de fois que l'on souhaite), de telle manière que la somme de leur **poids** ne dépasse pas la **capacité** c du sac à dos et que la somme de leurs **valeurs** soit **maximale**. On notera $w(x)$ le poids de l'objet x et $v(x)$ sa valeur (ces deux fonctions renvoient des valeurs **entières strictement positives**).

On peut résoudre ce problème NP-difficile grâce à la relation de récurrence sur $t(j)$ définie comme la valeur totale maximale possible pour une collection d'objets dont le poids total ne dépasse pas j :

$$\forall j > 1, \quad t(j) = \max_{\forall i \in [1,n] \text{ t.q. } w(x_i) \leq j} (t(j - w(x_i)) + v(x_i))$$

La solution sera ainsi donnée par le calcul de $t(c)$.

1. [6pt] Écrire une fonction **récursive directe** qui calcule $t(c)$ en prenant en paramètres les listes (ou tableaux) `w` de poids et `v` de valeurs, ainsi que la capacité maximale `c` du sac à dos.
2. [4pt] Estimer les **complexités temporelle et spatiale** en pire cas de cet algorithme récursif, en considérant que $\forall i, w(x_i) = 1$.
3. [6pt] Écrire une fonction avec les mêmes paramètres que la fonction précédente, mais qui utilise la technique de **programmation dynamique**.
4. [4pt] Calculer les **complexités temporelle et spatiale** en pire cas de l'algorithme de programmation dynamique.