Examen d'algorithmique

Jeudi 8 janvier 2009 – Notes de cours autorisées

Problème : Sac à dos et variantes – (11 points [1])

On s'intéresse au "problème du sac à dos". On dispose :
- d'un sac qui peut contenir des objets d'un poids total **maximal** de C kilogrammes ;
- d'un ensemble de n objets ayant chacun une valeur et un poids fixés. Les poids des objets sont stockés dans un tableau Poids$[1..n]$ et les valeurs sont stockées dans un tableau Val$[1..n]$. (toutes ces valeurs sont des entiers positifs).

Une solution du problème du sac à dos est une manière de remplir le sac sans dépasser sa capacité C et en maximisant la valeur contenue dans le sac.

On distingue deux problèmes différents selon que les objets sont fractionnables ou non.

1. Le cas des objets fractionnables. Dans cette version du problème, on peut choisir de ne mettre dans le sac qu'une partie (la moitié, un tiers, etc.) de chaque objet. Une solution au problème se représentera par un tableau S$[1..n]$ à valeur dans l'intervalle $[0;1]$ où la valeur S$[i]$ correspond à la proportion de l'objet i que l'on a choisi de mettre dans le sac. Un sac vide est représenté par un tableau S$[-]$ rempli de 0, tandis que le sac contenant tous les objets en entier sera décrit par un tableau rempli de 1.

Exemple 1 : Considérons les 6 objets définis comme ci-dessous :

n°	1	2	3	4	5	6
Poids	10	30	2	50	20	8
Val	5	20	2	10	30	14

Alors un tableau S $= [0\,;\,1\,;\,0.5\,;\,1\,;\,0.25\,;\,0.1]$ correspond à un sac rempli avec : les objets 2 et 4 en entier, la moitié de l'objet 3, un quart de l'objet 5 et un dixième de l'objet 6. Le poids de ce sac sera : $P(\mathsf{S}) = \sum_{i=1\ldots n} \mathsf{Poids}[i] \cdot \mathsf{S}[i] = 30 \cdot 1 + 2 \cdot 0.5 + 50 \cdot 1 + 20 \cdot 0.25 + 8 \cdot 0.1 = 86.8$

La valeur totale de ce sac sera :

$$V(\mathsf{S}) = \sum_{i=1\ldots n} \mathsf{Val}[i] \cdot \mathsf{S}[i] = 20 \cdot 1 + 2 \cdot 0.5 + 10 \cdot 1 + 30 \cdot 0.25 + 14 \cdot 0.1 = 39.9$$

⇒ **Nous cherchons ici un algorithme qui détermine une meilleure façon de choisir une proportion de chaque objet pour maximiser la valeur du sac tout en ne dépassant pas la capacité C du sac.**

1. Montrer que lorsque $\sum_{i=1\ldots n} \mathsf{Poids}[i] \leqslant C$, alors il y a une solution évidente au problème de remplissage du sac.

   Lorsque $\sum_{i=1\ldots n} \mathsf{Poids}[i] \geqslant C$, expliquer pourquoi une solution au problème a toujours un poids égal à C.

[1] Le barème est donné à titre indicatif.

2. Proposer une solution (et calculer sa valeur) lorsque $C = 40$ et que l'on dispose des six objets décrits dans l'exemple ci-dessus.
3. Dans un premier temps, on va s'intéresser au remplissage du sac **sans chercher à maximiser sa valeur**. On veut écrire un algorithme `Remplir-Sac-Simple` qui étant donné un tableau Poids$[1..n]$ et une capacité C retourne un tableau S$[1..n]$ correspondant à un sac rempli **complètement** (sans dépasser C) avec les **premiers** objets de la liste : On met les premiers objets en entier jusqu'à rencontrer un objet dont le poids dépasse la capacité restante du sac et que l'on doit fractionner (les derniers objets ne seront pas choisis).
   (a) Appliquer `Remplir-Sac-Simple` sur les objets de l'exemple 1 avec $C = 45$.
   (b) Ecrire l'algorithme `Remplir-Sac-Simple`.
4. On revient maintenant au problème du sac à dos. On souhaite écrire un algorithme pour trouver une solution au problème en utilisant la procédure `Remplir-Sac-Simple`. Pour cela, on va étudier les algorithmes fonctionnant en **deux** étapes : d'abord on trie les objets selon un critère, puis on applique `Remplir-Sac-Simple` (en énumérant les objets dans l'ordre défini à l'étape précédente).
   (a) Appliquer cet algorithme sur les objets de l'exemple 1 avec $C = 40$, en triant les objets par **poids croissant**.
   (b) Refaire la question précédente en triant les objets par **valeur décroissante**.
   (c) ...et en triant les objets par **rapport** $\frac{\text{Val}[i]}{\text{Poids}[i]}$ **décroissant**.

   En déduire que deux de ces algorithmes ne sont pas corrects. Prouver que le troisième est correct (c'est-à-dire qu'il fournit une solution au problème du sac à dos). Dans la suite, on appellera cet algorithme `Algo-Sac-à-Dos-1`.
5. Quelle est la complexité de `Algo-Sac-à-Dos-1` ? A quelle famille d'algorithme appartient-il ?
6. On veut maintenant traiter le problème suivant : étant donnés une liste d'objets avec un poids et une valeur, et **deux** sacs ayant une capacité C et C', quelle est la **valeur** maximale que l'on placer dans ces deux sacs tout en respectant leur capacité ?
   Comment utiliser `Algo-Sac-à-Dos-1` pour résoudre le nouveau problème ? Expliquer.

2. Le cas des objets non fractionnables. Dans cette version, si un objet est choisi, il doit être mis en entier dans le sac. Une solution se représentera donc par un tableau S$[1..n]$ à valeur dans l'ensemble $\{0; 1\}$: S$[i]$ vaut 1 si et seulement si l'objet i est mis (en entier) dans le sac.

1. On définit l'algorithme `Algo-2` comme une variante de `Algo-Sac-à-Dos-1` pour le cas des objets non fractionnables :
   - On trie les objets par rapport $\frac{\text{Val}[i]}{\text{Poids}[i]}$ décroissant ;
   - On énumère tous les objets dans l'ordre ci-dessus et à l'itération i, on met l'objet i dans le sac si et seulement si il reste assez de place dans le sac.

   Appliquer cet algorithme sur les objets l'exemple 1. Donner la valeur du sac ainsi construit.
2. On considère l'exemple suivant :
   Exemple 2 : Considérons les 3 objets définis comme ci-dessous :

n°	1	2	3
Poids	15	20	15
Val	15	22	14

Appliquer l'algorithme `Algo-2` sur les objets de l'exemple 2 avec un sac de capacité 30.

En déduire que l'algorithme `Algo-2` n'est pas correct (*i.e.* ne retourne pas une solution optimale) pour le cas des objets non fractionnables.

3. On cherche un algorithme (appelé `Algo-Sac-à-Dos-2`) qui calcule la **valeur maximale** que peut contenir un sac de capacité C étant donnés les tableaux Poids$[1..n]$ et Val$[1..n]$. Pour cela, on veut calculer les quantités $V[l,k]$ (avec $l \in \{0,\ldots,C\}$ et $k \in \{0,\ldots,n\}$) correspondant à la valeur maximale pouvant être stockée dans un sac de capacité l avec les objets $1,\ldots,k$. On a bien sûr $V[l,0]=0$ et $V[0,k]=0$ car le seul sac possible est alors vide. **Exprimer $V[l,k]$ en fonction de $V[l,k-1]$ et $V[l-\text{Poids}[k],k-1]$ (justifier votre réponse). En déduire l'algorithme `Algo-Sac-à-Dos-2`, préciser sa complexité.**
4. Appliquer votre algorithme sur l'exemple 2.
5. (* facultatif *) On veut maintenant traiter le problème du remplissage de **deux sacs**. Montrer que dans ce cas, on ne peut pas utiliser directement `Algo-Sac-à-Dos-2` contrairement au cas des objets fractionnables.

   Proposer un algorithme basé sur la même idée que `Algo-Sac-à-Dos-2` mais qui résout le problème pour deux sacs.

Exercice 1 : Arbres binaires de recherche – (3 points)

On considère des arbres binaires de recherche où l'on dispose, en plus des champs clé, FilsG et FilsD, d'un champ Père qui permet de remonter au père (lorsqu'il existe) d'un noeud.

1. Donner un algorithme pour trouver la clé maximale stockée dans un arbre binaire de recherche. Evaluer sa complexité.
2. Donner (et expliquer !) un algorithme pour trouver la deuxième clé maximale stockée dans un arbre binaire de recherche. (On veut un algorithme en $O(h)$ où h désigne la hauteur de l'arbre).

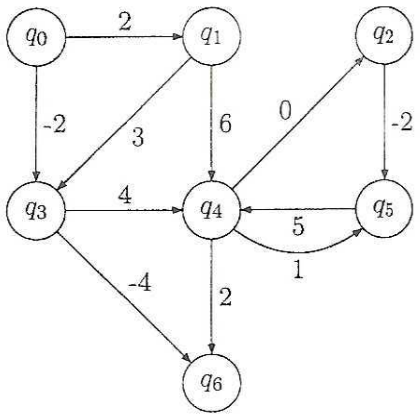
Exercice 2 : Algorithme sur les graphes – (3 points)

On considère l'algorithme de `Algo-BF` pour calculer les distances des plus courts chemins (lorsqu'ils existent) depuis un sommet s donné :

```
Algo-BF(  G=(S,A) : graphe orienté, w: fonction de pondération, s:sommet)
d[] : un tableau de |S| entiers initialisé à +infini
d[s]:=0
pour i de 1 à |S| faire
  pour chaque arc (u,v) de A faire
     si d[u] + w(u,v) < d[v]
               d[v] := d[u] + w(u,v)

pour chaque arc (u,v) de A faire
    si d[u] + w(u,v) < d[v] retourner Faux
retourner vrai
```

1. Appliquer l'algorithme `Algo-BF` au graphe ci-dessous et q_0 (donner la valeur de `d[-]` à chaque itération de `i`).



2. Pour cela, on pourra montrer que si il existe un plus court chemin de s à u contenant k arcs, alors après l'itération k, `d[u]` vaut $\delta(s,u)$ où $\delta(s,u)$ désigne la distance d'un plus court chemin entre s et u.

   En déduire que si G ne contient pas de cycle négatif atteignable depuis s, alors à la fin de l'algorithme on a : `d[v]` $= \delta(s,v)$ pour tout sommet $v \in S$
3. Evaluer la complexité de l'algorithme `Algo-BF` .
4. A quoi sert la dernière boucle de l'algorithme ? Expliquer.
5. Comparer cet algorithme avec l'algorithme de Dijkstra vu en cours.
6. Montrer que lorsqu'il existe des cycles strictement négatifs accessibles depuis s, alors les valeurs `d[u]` ne correspondent pas toujours à des distances de chemins **simples**.

   (NB : on dit qu'un chemin est simple lorsqu'il ne passe pas deux fois par le même sommet).

Exercice 3 : Arbres couvrants minimaux – (3 points)

Dans cet exercice on considère uniquement des graphes $G = (S, A, w)$ non-orientés, valués et connexes.

1. Donner un exemple de graphe admettant plusieurs *arbres couvrants minimaux différents*.
2. Soit $T \subseteq A$ un arbre couvrant minimal de G tel que l'arête reliant deux sommets x et y de S appartienne à T. Montrer que l'ensemble d'arêtes $T' = T \backslash \{(x,y)\}$ est un arbre couvrant minimal du graphe G' où les deux sommets x et y ont été "fusionnés" en un nouveau sommet z.

   (G' est le graphe (S', A', w') avec $S' = (S \backslash \{x,y\}) \cup \{z\}$ et les arêtes A' sont celles de $A \backslash \{(x,y)\}$ où les sommets x et y sont remplacés par z. La fonction w' renvoie le même poids que w. Et si il existe $(x,u) \in A$ et $(y,u) \in A$, on ne garde dans G' qu'une seule arête (z,u) et son poids est $\min(w(x,u), w(y,u))$.)
3. Montrer que si la fonction poids w de $G = (S, A, w)$ est telle que le poids de chaque arête de A est unique, alors il existe un unique arbre couvrant minimal pour G.

   Pour cette démonstration, on pourra utiliser le résultat précédent ainsi que les techniques de démonstration vues en cours.