Algorithmique 2022 - TP 2: Greedy & Complexité

Brian Pulfer
Brian.Pulfer@unige.ch

12 Octobre 2022

Remarques:

Veuillez suivre attentivement les spécifications de l'énoncé.

- **Réponses** - Essayez d'être exhaustif dans vos réponses. Les réponses comme "oui" et "non" ne permettent pas d'évaluer vos connaissances.
- **Tracés** - Mettez toujours des étiquettes d'axe et des titres pour les tracés.
- **Implémentation** - Veuillez utiliser exactement les noms fournis pour les fonctions. Vous pouvez utiliser *pytest* avec le script de test donné pour vérifier votre implémentation.
- **Soumission**
  Merci de téléchargé vos devoirs sur moodle:
  - **NomPrenom.pdf** avec votre rapport pour tout les exercices
  - **tp2.py** avec votre implementations
  - *Pas d'autres fichiers* (.pyc, .ipynb, DS_Store, __pycache__, ...)

  Délai - 25 Octobre 2022, 23:59 CET
- Pour toute questions et remarques, merci d'utiliser le mail Brian.Pulfer@unige.ch.

1 Knapsack (1 Points)

Le problème du knapsack est le suivant: étant donné un ensemble d' éléments avec des poids et des valeurs associés, trouvez le sous-ensemble qui maximise la somme des valeurs tout en maintenant la somme des poids sous une valeur constante (taille du sac). Voir Table 1 pour un exemple.

Nr. Élément	**Poids**	**Valeur**
1	10	5
2	3	2
3	7	4
4	12	9
5	1	2

Table 1: Exemple d'éléments dans un problème de Knapsack

La solution optimale dépend du poids que le knapsack peut supporter. Pour une capacité de 25, la solution optimale est de choisir les éléments 2, 3, 4, 5 avec un poids total de 23 (< 25) et une valeur totale de 17.

Implémentez trois algorithmes greedy. Utilisez le script de test pour vérifier votre implémentation. Les algorithmes continuent d'insérer l'élément non pris avec:

a la valeur la plus élevée;

b le poids le plus bas;

c le rapport $\frac{valeur}{poids}$ le plus élevé.

Pour chaque algorithme proposé:

- Indiquez quelle est la complexité en temps.
- Indiquez s'il est optimal ou non. Si c'est le cas, prouvez pourquoi. Sinon, donnez un contre-exemple.

2 Rendu de monnaie britannique (2 Points)

Avant la décimalisation, la monnaie britannique contenait les pièces suivantes:

- la *demi-couronne* d'une valeur de 30 pence;
- le *florin* d'une valeur de 24 pence;
- le *shilling* d'une valeur de 12 pence;
- le *sixpence* d'une valeur de 6 pence;
- le *threepence* d'une valeur de 3 pence;
- le *penny*.

Dans cet exercice, on ne prendra pas en compte le demi penny (12 pence) ni le farthing (14 pence). Notez que 'pence' est le pluriel de 'penny'.

- Donnez un pseudocode correspondant à l'algorithme Greedy vu en cours pour résoudre ce problème.
- Pourquoi peut-on dire que c'est un algorithme de type Greedy?
- Quelle est la complexité en temps de cet algorithme?
- Cet algorithme est-il toujours optimal? Si oui, prouvez-le, sinon donnez un contre-exemple.
- Implémentez cet algorithme (correspondant à votre pseudocode) en Python au travers d'une fonction `compute_change(money, coin_set)`, où money est la monnaie totale qui doit être changée et `coin_set` une simple liste ordonnée contenant la valeur des pièces disponibles pour le change. Cette fonction doit retourner une liste contenant, dans l'ordre, les pièces changées contre la somme initiale. Utilisez le script de test pour vérifier votre implémentation.

```
1   from tp2 import compute_change
2
3   def test_compute_change():
4       coin_set = [30, 24, 12, 6, 3, 1]
5       assert compute_change(0, coin_set) == []
6       assert compute_change(2, coin_set) == [1, 1]
7       assert compute_change(4, coin_set) == [3, 1]
8       assert compute_change(10, coin_set) == [6, 3, 1]
9       assert compute_change(33, coin_set) == [30, 3]
10      assert compute_change(100, coin_set) == [30, 30, 30, 6, 3, 1]
```

Listing 1: Code de test pour la fonction `compute_change`

3 Minimum Spanning Tree (1 Point)

Étant donné une matrice d'adjacence symétrique A d'un graphe G, où chaque élément $A_{i,j} = A_{j,i}$ représente le poids des arêtes reliant les nœuds i et j, implémentez l'algorithme de Kruskal qui retourne les arêtes de l'arbre couvrant minimum (MST). Utilisez le script de test pour vérifier votre implémentation.

- Cet algorithme va-t-il toujours trouver un arbre couvrant minimum? Si oui, prouvez pourquoi. Sinon, donnez un contre-exemple.

4 4-SAT (1 Point)

4-SAT: Déterminer si une formule booléenne en CNF, ou chaque disjonction comprend exactement 4 éléments, est satisfaisable.

Montrez que 4-SAT est NP-Complet.