Algorithmique 2022 - TP 6

Brian Pulfer
Brian.Pulfer@unige.ch

12.12.2022

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.
- **Points extra** - Le nombre maximum de points pouvant être marqués pour un TP est de 5, ce qui conduit à une note de 6. Vous pouvez effectuer l'exercice 2 pour obtenir des points supplémentaires. Marquer plus de 5 points est toujours arrondi à la note 6.
- **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
  - **tp6.py** avec votre implementations
  - *Pas d'autres fichiers* (.pyc, .ipynb, DS_Store, __pycache__, ...)

  Délai - 18.12.2022, 21:00 CET
- Pour toute questions et remarques, merci d'utiliser le mail Brian.Pulfer@unige.ch.

1 Vertex Cover (5 Points)

Vertex-cover est un problème NP-complet. Cependant, nous savons que l'algorithme de 2-approximation suivant est linéaire en nombre d'arêtes et de nœuds:

```
APPROX-VERTEX-COVER(G):
    C = 0
    E' = G.E
    while E' is not 0
        let (u, v) be an arbitrary edge of E'
        C = C U {u, v}
        remove from E' every edge incident on either u or v
    return C
```

Listing 1: Pseudo-code de l'algorithme d'approximation pour le problem du vertex-cover

- Implementez l'algorithme d'approximation `approx_vertex_cover(matrix)`. L'input de l'algorithme est une matrice d'adjacence symétrique $n \times n$ `matrix` (`matrix[i][j]` indique si les nœuds i et j sont connectés par une arête). L'output est l'ensemble des nœuds sélectionnés. L'algorithme doit traverser les arêtes dans l'ordre `((0,1), (0,2)...(n-1,n-1))`.

2 Knapsack problem (1 Point extra)

Knapsack problem (0-1) est un problème NP-complet. Cependant, nous savons que l'algorithme de 2-approximation suivant est polynomial:

```
APPROX-KNAPSACK(I):
    R = 0
    for j = 1, ..., n
        Ij = I[j:]
        Rj = greedy(Ij)
        R = R U Rj
    return maximum_element(R)
```

Listing 2: Pseudo-code de l'algorithme d'approximation pour le problem du TSP

- Implementez l'algorithme d'approximation `approx_knapsack(weights, values, max_weight)`. L'input de l'algorithme sont le liste des poids et des valeurs des elements et la capacité du knapsack. L'output est la liste des index sélectionnés par l'algorithme d'approximation.