Algorithmique 2022 - TP 4

Brian Pulfer
Brian.Pulfer@unige.ch

16.11.2022

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 les exercices 2 et 3 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
  - **tp4.py** avec votre implementations
  - *Pas d'autres fichiers* (.pyc, .ipynb, DS_Store, __pycache__, ...)

  Délai - 29.11.2022, **21:00 CET**
- Pour toute questions et remarques, merci d'utiliser le mail Brian.Pulfer@unige.ch.

1 Voyage en train (5 Points)

Commençons par nous intéresser à un cas particulier, le voyage en train le long de la ligne RE *Annemasse - St-Maurice* des CFF. Les prix différents de différentes communautés tarifaires et les prix intercommunautaires ont une structure assez particulière. Les différents prix peuvent être résumés sous la forme d'une matrice M que voici:

	Annemasse	Genève	Coppet	Nyon	Morges	Lausanne	Vevey	St-Maurice
Annemasse	0							
Genève	4.9	0						
Coppet	6	3	0					
Nyon	7.6	4.5	2.8	0				
Morges	11.4	9.3	8.3	6.5	0			
Lausanne	14	11.4	11.1	9.3	3.7	0		
Vevey	16.5	14	13.9	12.9	7.4	5.6	0	
St-Maurice	21	19	13.9	13.9	13.9	12	7.4	0

On remarque que, pour certains trajets, la somme du prix des étapes *"atomiques"* entre deux villes peut être plus grande, égale ou plus petite que le prix pour le train direct.

Dans le cas où l'on voudrait faire le trajet Annemasse-St-Maurice, la somme du prix des étapes coûterait CHF 33.90, le trajet direct CHF 21.-, mais le trajet optimal, qui consiste à faire Annemasse-Copet-St-Maurice, coûtera seulement CHF 19.90.

Nous allons pour cet exercice considérer le cas général où il y a n villes, soit un nombre maximal de $n-2$ étapes intermédiaires. Lors d'un trajet, on ne revient jamais en arrière (d'où le fait que M soit à moitié vide). On est libre de prendre n'importe quel nombre de sous-trajets. Le but est d'analyser et d'implémenter une stratégie de programmation dynamique pour trouver le coût optimal pour aller de la ville numéro 0 à la ville numéro $n-1$. Pour ce faire, nous proposons de remplir un tableau T des coûts optimaux, c'est-à-dire un tableau similaire à la matrice M ci-dessus, ne comprenant non pas les coûts *"officiels"* de la compagnie ferroviaire, mais les coûts optimaux pour chaque sous-trajet. Par exemple, la case correspondant au trajet Annemasse-St-Maurice aurait une valeur de 19.90.

1.1 Fonction de récurrence

- Nous avons défini ci-dessus en quoi consiste la matrice T des coûts optimaux. Donnez une relation de récurrence de T qui permette de trouver la valeur de T_{ij} (=coût optimal pour aller de la ville numéro i à la ville numéro j).N'oubliez pas que le coût optimal (minimal) d'un trajet est le minimum des coûts de l'ensemble des trajets possibles. Justifiez.
- Votre formule de récursion a-t-elle un impacte sur la manière de *"remplir"* T? Si oui, lequel?

1.2 Implémentation

Implémentez un solver pour ce problème avec une stratégie de programmation dynamique utilisant la fonction de récursion que vous donnez à la question 1.1.

Dans le code python que vous devez rendre, il doit contenir une fonction `get_solution` qui a comme:

- **input** la matrice des coûts de la compagnie ferroviaire M, sous forme de liste de liste.

- **output** la matrice T des coûts correspondants (sous forme de liste de liste), le chemin optimal (sous forme de liste) et le coût du chemin optimal entre la ville 0 et la ville $n-1$.

1.3 Complexité en temps

- Quelle est selon vous la complexité en temps (en fonction de n) de l'algorithme de programmation dynamique (donné au point 1.1) pour résoudre le problème? Argumentez.
- Est-ce que la complexité de temps effective de votre algorithme de programmation dynamique implémenté au point 1.2 correspond à la complexité théorique donnée au-dessus? Argumentez

NB: Pour produire le graphique demandé à la question 1.3, vous pouvez utiliser la méthode de votre choix. La librairie *matplotlib* permet néanmoins de créer rapidement des graphiques en python. **Les graphiques doivent avoir un titre ainsi que des axes nommés et avec des unités le cas échéant.**

2 Rendu de la monnaie (extra 0.5 Point)

Implémentez en python la stratégie de programmation dynamique vue en cours pour le problème du rendu de la monnaie.

3 Sequences (extra 0.5 Point)

3.1 Plus longue sous-séquence commune

Soient deux séquences $A = a_1, ...a_n$ et $B = b_1, ..., b_n$ de symboles issus d'un alphabet Ω. La plus longue séquence de symboles identiques entre A et B est nommée *plus longue sous-séquence commune* (longest common sub-sequence). Par exemple, soient les séquences $A = [a, c, t, g, a, a]$ et $B = [c, g, a, t]$: la plus longue sous-séquence est $[c, g, a]$.

- Trouver une solution dynamique au problème (le but est de retourner la longueur de la plus longue sous-séquence commune).
- $N > 2$ le nombre de séquences à comparer. Notre solution dynamique fonctionne-t-elle toujours?

3.2 Plus longue sous-chaîne commune

Soient deux séquences $A = a_1, ...a_n$ et $B = b_1, ..., b_n$ de symboles issus d'un alphabet Ω. La plus longue chaîne de symboles identiques entre A et B est nommée *plus longue sous-chaîne commune* (longest common substring).

- Le problème est-il identique au précédent ?
- Trouver une solution dynamique au problème.