INF8775 – Analyse et conception d'algorithmes

TP2 – Hiver 2023

Nom, prénom, matricule des membres	NOM, Prénom, 1234567 NOM, Prénom, 1234567
Note finale / 13	

Informations techniques

- Répondez directement dans ce document. Veuillez ne pas inclure le texte en italique servant de directive.
- La correction se fait sur ce même rapport.

  Vous devez faire une remise électronique sur Moodle avant le

  14 mars à 23h59 pour le groupe B2

  21 mars à 23h59 pour le groupe B1

  en suivant les instructions suivantes :
- Vos fichiers doivent être remis dans une archive zip à la racine de laquelle on retrouve :
  - Ce rapport sous format ODT ou docx.
  - Un script nommé *tp.sh* servant à exécuter les différents algorithmes du TP. L'interface du script est décrite à la fin du rapport.
  - Le code source et les exécutables.
- Vous avez le choix du langage de programmation utilisé mais vous devrez utiliser les mêmes langage, compilateur et ordinateur pour toutes vos implantations. **Le code et les exécutables soumis devront être compatibles avec les ordinateurs de la salle L-4714.**
- Si vous utilisez des extraits de codes (programmes) trouvés sur Internet, vous devez en mentionner la source, sinon vous serez sanctionnés pour plagiat.

Mise en situation

Ce travail pratique se répartit sur deux séances de laboratoire et porte sur l'analyse et la conception d'algorithmes développés suivant différents patrons de conception.

On vous demande d'y résoudre le problème du voyageur de commerce (*Traveling Salesman Problem*, TSP) en utilisant trois patrons de conception distincts :

1. Algorithme glouton : en démarrant d'une ville arbitraire et en faisant une tournée en sélectionnant à chaque fois le plus proche voisin ;
2. Algorithme de programmation dynamique : en résolvant le problème à travers la relation de récurrence suivante :

$$D[i,S] = \begin{cases} \min\limits_{j \in S} d_{ij} + D[i, S \setminus \{j\}] & \text{si } S \neq \emptyset \\ d_{i0} & \text{sinon} \end{cases}$$

3. Algorithme approximatif (1-relatif) : en construisant un arbre sous-tendant minimum (Minimum Spanning Tree, MST) du graphe et en le parcourant en préordre à partir d'une ville arbitraire.

On s'intéresse dans ce TP au problème du TSP métrique : les villes appartiennent à un plan N^2 muni de la distance euclidienne **arrondie à l'entier le plus proche**.

Jeux de données

Pour tester les algorithmes, vous devez générer des jeux de données en utilisant le script fourni :

```
$ ./inst_gen.py [-h] -s NB_VILLES [-n NB_EXEMPLAIRES] [-x PRÉFIXE]
```

On suggère d'utiliser le script bash suivant pour automatiser la génération des exemplaires :

```
#!/bin/bash

# Pour glouton et approx
for n in {"1000","5000","10000","50000","100000"}; do
    ./inst_gen.py -s $n -n 5
done

# Pour tous les algorithmes
for n in {"5","10","15","20","25"}; do
    ./inst_gen.py -s $n -n 5 -x DP
done
```

La première ligne de chaque exemplaire contient le nombre de villes à visiter (qui est aussi la taille du problème *N*). Les *N* lignes subséquentes contiennent chacune les coordonnées d'une ville à la fois, séparées par des espaces.

Les coordonnées des villes sont chacune comprises entre 0 et 2000 et aucune paire de villes n'est telle que la distance entre elles est de 0.

Présentation des résultats

0	/ 4 pt

Tableau des résulats

Exécutez chacun des trois algorithmes en notant leur temps d'exécution ainsi que la longueur minimale de votre tour à travers les N villes indiquées. Rapportez dans un tableau vos résultats ainsi que la moyenne des temps d'exécution pour chaque série d'une même taille.

Pour l'algorithme de programmation dynamique, ne le lancez que sur les tailles d'exemplaires de 25 et moins.

Graphiques pour analyse hybride

Voir questions plus bas.

Analyse et discussion

0	/ 6 pt

Faites une analyse asymptotique du temps de calcul pour chaque algorithme.

Joignez un PDF à la remise avec la réponse à cette question.

Servez-vous de vos temps d'exécution pour confirmer et/ou préciser l'analyse asymptotique théorique de vos algorithmes avec la méthode hybride de votre choix.

La méthode peut varier d'un algorithme à l'autre. Justifiez les choix ici et mettez les graphiques dans la section précédente.

Discutez des trois algorithmes en fonction de la qualité respective des solutions obtenues, de la consommation de ressources (temps de calcul, espace mémoire) et de la difficulté d'implantation.

Indiquez sous quelles conditions vous utiliseriez chaque algorithme.

On vous fournit 5 exemplaires difficiles à résoudre jusqu'à optimalité[1] (fichiers avec préfixe `hard`). Tentez de les résoudre à l'aide de vos algorithmes glouton et approximatif et discuter des écarts obtenus.

Fichier	Nombre de villes	Solution optimale (L2 arrondie)
`hard_N52`	52	551609
`hard_N91`	91	1228726
`hard_N130`	130	1928734
`hard_N169`	169	2600546
`hard_N199`	199	3139778

[1] Tirés de http://www.or.uni-bonn.de/~hougardy/HardTSPInstances.html, les coordonnées peuvent dépasser 2000.

Autres critères de correction

Respect de l'interface `tp.sh`

	/ 1 pt

Utilisation :

```
$ ./tp.sh -a {glouton, progdyn, approx} -e CHEMIN_EXEMPLAIRE [-p] [-t]
```

Arguments optionnels :

> `-p` affiche dans l'ordre, sur chaque ligne, les indices des villes à visiter en commençant par 0 et en finissant par 0, sans texte superflu. Rapportez le chemin tel que la deuxième ville visitée ait un indice inférieur à celui de l'avant dernière ville affichée (voir exemple).
> `-t` affiche le temps d'exécution en millisecondes, sans unité ni texte superflu.

Important : l'option `-e` doit pouvoir accepter des chemins absolus.

```
user@host folder $ ./tp.sh -p -e N5_0 -a progdyn
0
1
4
3
2
0
```

Qualité du code

	/ 1 pt

Présentation générale

	/ 1 pt

- Concision
- Qualité du français

Pénalité retard

0

- -1 pt / journée de retard, arrondi vers le haut. Les TPs ne sont plus acceptés après 3 jours.