INF8775 – Analyse et conception d'algorithmes

TP3 – Hiver 2023

Nom, prénom, matricule des membres	Nom, Prénom, matricule Nom, Prénom, matricule
Note finale / 20	**0**

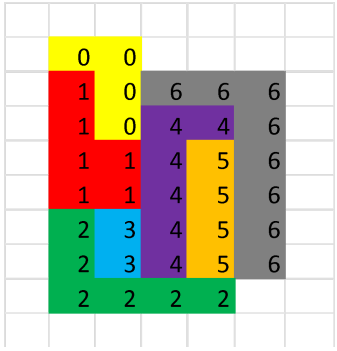
Informations sur la correction

- Répondez directement dans ce document. La correction se fait à même le rapport.
- La date limite pour rendre ce TP est :
  - 18 avril à 23h55 pour le groupe B2
  - 24 avril à 23h55 pour le groupe B1
- Vous devez faire une *remise électronique sur Moodle* en suivant les instructions suivantes :
  - Le dossier remis doit se nommer matricule1_matricule2_tp1 et doit être compressé sous format zip.
  - À la racine de ce dernier, on doit retrouver :
    - Ce rapport au 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 l'exécutable.
- Vous avez le choix du langage de programmation utilisé. Notez que le code et l'exécutable soumis seront testés sur les ordinateurs de la salle L-4714 et doivent être compatibles avec cet environnement. En d'autres mots, tout doit fonctionner correctement lorsque le correcteur exécute votre script `tp.sh` sur un des ordinateurs de la salle.
- La commande `chmod +x mon_script.sh` rendra le script `mon_script.sh` exécutable. Pour l'exécuter il s'agira de faire `./mon_script.sh`

Ce dernier travail pratique se fera dans le cadre du concours du meilleur algorithme pour la session d'hiver 2023. Le travail demandé consiste à concevoir et implanter un algorithme de votre cru pour résoudre un problème combinatoire. Le classement des équipes déterminera votre note pour la *qualité de l'algorithme*. Votre algorithme sera exécuté sur 3 exemplaires de notre choix pendant 2 minutes chacun.

Le but du TP est de placer des enclos dans un zoo pour minimiser le temps de déplacement des clients entre les enclos. Ainsi, chaque enclos a un poids différent pour tous les autres, représentant la quantité de gens qui veulent se rendre d'un enclos donné à un autre. On peut le représenter par un graphe pondéré fortement connexe G avec n sommets (les enclos) et les poids des arêtes E sont le nombre de gens qui veulent se rendre d'un sommet u à v.

Les enclos peuvent avoir des tailles variables de 2 à 20 cases (carrés de 1 de côté), placés sur une grille cartésienne 2D infinie. Les cases d'un enclos doivent être orthogonalement adjacentes, de sorte que les enclos soient contigus, mais toutes les formes sont autorisées. Par exemple :



Le terme à minimiser est le suivant :

$$\sum_{E} distance(u,v) * poids\_arête(u,v)$$

Où la distance(u,v) est la distance de manhattan entre les deux cases les plus près des enclos u et v (i.e. distance de 1 pour des enclos adjacents), et poids_arête(u,v) est la valeur de l'arête entre u et v.
Dans l'exemple ci-haut, la distance entre les enclos 0 et 1 (ou 0 et 6) serait 1, et la distance entre 0 et 2 serait de 4.

Il y a un autre aspect à prendre en compte dans votre placement des enclos: placer un ensemble donné d'enclos à une distance maximale les uns des autres ajoute à l'attrait du zoo (par exemple, un thème félins). On a un sous-ensemble *S* du graphe G, de taille m. Si tous les enclos de *S* se trouvent à une distance inférieure à *k* les uns des autres, la condition est remplie et l'on obtient un attrait supplémentaire égale au carré du nombre d'enclos dans le sous-ensemble.
On obtient donc la contrainte suivante : $V = \{m^2 \; si \; max_{u,v \in S} \; distance(u,v) \leq k \; 0 \; sinon$
Par exemple, pour les sous-ensembles $\{0,\ 1,\ 6\}$ *et* $\{0,\ 2,\ 4,\ 5\}$ *avec une distance max de 3, on obtient la valeur ajoutée de 9 pour* $\{0,\ 1,\ 6\}$ *car distance(1,6) = 3, mais pas 16 pour* $\{0,\ 2,\ 4,\ 5\}$ *car distance(0,2) = 4*
L'attrait total du zoo à maximiser est donc :

$$V \; - \; \sum_{E} distance(u,v) * poids\_arête(u,v)$$

Jeu de données

Nous vous fournissons un générateur d'exemplaires qui fonctionne comme suit :

```
python inst_gen.py -n 10 -m 5
```

Arguments :

- n = nombre d'enclos (nombre de sommets)
- m = taille du sous-ensemble à placer à proximité (<= n)

Un fichier au format ex_n_m.txt est enregistré dans le répertoire d'exécution. Par exemple :

```
10 5 5
9 5 1 7 3
11
9
20
20
9
9
7
9
18
9
0 55 76 12 63 76 6 92 34 23
47 0 19 94 59 55 76 41 51 7
55 52 0 72 22 50 34 52 5 65
44 45 9 0 36 30 29 12 33 84
65 91 44 10 0 82 77 14 16 78
1 57 3 73 60 0 14 75 15 21
9 76 29 40 30 26 0 38 31 92
83 51 6 5 20 62 34 0 88 40
65 28 27 16 50 37 55 100 0 37
42 28 97 82 69 11 7 66 12 0
```

La première ligne correspond au nombre d'enclos n, suivi du d'nombre d'enclos m dans le sous-ensemble S, et de la distance maximale à respecter k pour les enclos de ce sous-ensemble.

La 2e ligne donne la liste des enclos à placer à une distance de moins de k pour obtenir le bonus $V = m^2$.

Les n lignes suivantes donnent la taille de l'enclos i pour i allant de 0 à n-1,

Finalement, il y a n lignes supplémentaires qui représentent les poids de l'enclos i vers les n autres enclos. (poids de 0 pour la distance avec soi-même)

Vous devez générer des exemplaires de la taille de votre choix, voici un ordre d'idée pour les valeurs : nombre d'enclos **n** entre 100 et 1000, taille max du sous-ensemble d'enclos **m** entre 10 et 500.

Q1 – Description de votre algorithme

Décrivez en quelques phrases votre algorithme.

0	/ 2 pt

Q2 – Présentation

Sous forme de pseudo-code et incluant une analyse de complexité théorique des principales fonctions. Si vous préférez écrire vos équations en Latex, vous pouvez ajouter un pdf à la remise avec la réponse à cette question et le mentionner ici. Pas besoin de faire une analyse empirique de la complexité.

0	/ 4 pt

Q3 – Justification de l'originalité de vote algorithme

La conception de votre algorithme sera jugée avec les critères suivants :

- *Lien avec le contenu du cours*
- *Originalité*
- *Initiative*

0	/ 4 pt

Q4 – Votre algorithme est-il assuré de trouver la solution optimale ?

Répondez simplement "oui" ou "non". Aucune justification requise.

0	/ 1 pt

Autres critères de correction

Respect de l'interface tp.sh

0	/ 2 pt

Utilisation :

```
tp.sh -e [chemin_vers_exemplaire] -p
```

Lorsque le script est exécuté sans le paramètre -p, le programme affiche uniquement l'attrait total du zoo, à chaque fois qu'une meilleure solution est trouvée. Votre programme est sensé s'exécuter tant et aussi longtemps qu'il n'est pas manuellement interrompu.

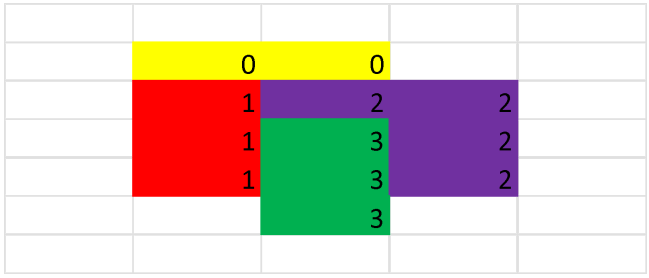
Argument optionnel :

`-p` Chaque fois qu'une meilleure solution est trouvée, le programme affiche cette nouvelle solution (**au lieu** d'afficher le nombre d'arêtes retirées). Le format de cette solution est décrit ci-dessous

Important : l'option -e doit accepter des fichiers avec des chemins absolus.

Le script tp.sh ne vous est pas fourni, mais vous pouvez facilement adapter celui du TP2.

Format de la solution : voici un exemple d'affichage. Le zoo correspondant est le suivant :



Note : ici, la case la plus haute de l'enclos 3 a les coordonnées (0,0) mais il n'y a pas de contrainte là-dessus. Les coordonnées négatives sont autorisées.

```
-1 2 0 2
-1 1 -1 0 -1 -1
0 1 1 1 1 0 1 -1
0 0 0 -1 0 -2

-1 2 0 2
-1 1 -1 0 -1 -1
0 1 1 1 1 0 1 -1
0 0 0 -1 0 -2
```

Chaque enclos occupe une ligne, avec les coordonnées de tous les cases de l'enclos selon la convention (x,y) d'un plan cartésien, séparées d'un espace. Entre chaque nouvelle solution trouvée, **vous devez laisser une ligne vide**.

Un script de vérification de solution vous est fourni (check_sol.py, les instructions d'utilisation sont dans le code source). Ce script vous indiquera si l'affichage est correct, si votre solution est valide, ainsi que la valeur de votre solution. C'est ce script qui sera utilisé pour la correction, donc assurez-vous qu'il reconnaisse vos solutions.

Qualité de l'algorithme

Les points de cette question sont répartis comme suit :
Nous allons exécuter votre code sur 3 exemplaires de notre choix. Pour chaque exemplaire la sortie de votre code sera envoyée vers le script de vérification, et nous classerons sur chaque exemplaire les différents binômes.
Pour chacun des 3 exemplaires, le quart des groupes ayant la meilleure solution remportera 1 point, le deuxième quart 0.75, le troisième quart 0.5 et le dernier quart 0.25 points.

Un bonus de 2 point est donné si vous trouvez une solution ***meilleure qu'une certaine baseline obtenue par un algorithme de base*** *dans le temps imparti.*

Cela signifie que si vous avez une solution relativement bonne mais un algorithme peu performant vous aurez quand même 2.75/5.

0	/ 5 pt

Qualité du code

0	/ 1 pt

Présentation générale (concision, qualité du français, etc.)

0	/ 1 pt

Pénalités

0

- Retard : -1 pt / journée de retard, arrondi vers le haut. Les TPs ne sont plus acceptés après 3 jours.
- Autres : Le correcteur peut attribuer d'autres pénalités (par exemple si les exécutables sont manquants, etc.)