Projet d'Algorithmique appliquée (2021-2022)

Présentation générale

Dans ce projet, l'objectif est de placer astucieusement des centres de vaccination dans une ville de telle sorte que toute personne se trouve à proximité d'un centre de vaccination. Ce projet se déroulera en plusieurs étapes :

1. modélisation du problème ;
2. implémentation d'algorithmes résolvant le problème : brute force, algorithmes basés sur de la programmation dynamique, algorithmes progressifs, d'approximation etc. ;
3. test de vos algorithmes sur différentes instances ;
4. extensions possibles.

Concernant l'organisation :

- groupes de 3 ou 4 étudiants ;
- langages de programmation autorisés : C, C++, Java ou Python ;
- la version finale devra être rendue avant les vacances de Noël, et des soutenances auront lieu en janvier ;
- possibles rendus intermédiaires (plus de détails à venir...).

Format des fichiers d'entrée

Deux principaux formats doivent pouvoir être pris en compte comme fichier d'entée :

1. une image stockée au format PPM ;
2. sous forme de graphe, que l'on peut représenter par exemple sous forme de liste d'adjacence.

Le format PPM

Description

Le format PPM (pour *Portable Pixmap*) est un moyen très simple de décrire une image sous forme textuelle. Dans notre cas, il s'agira de fichiers ASCII de ce format (voir Wikipedia) :

```
P3 # fichier PPM ASCII sous format RGB
3 2 # l'image contient 3 colonnes et 2 lignes
255 # chaque composante R, G et B est représentée par un entier entre 0 et 255
```

```
255  0    0     0  255  0       0   0  255 # première ligne
0   255   0     0   0  255     255  0   0  # deuxième ligne
```

Le fichier de configuration associé

En plus du fichier au format PPM, un fichier de configuration `config.txt` permet d'indiquer :

- le niveau d'échelle souhaité ;
- pour chaque couleur (i.e., chaque triplet `(R,G,B)`) présente dans le fichier PPM, on associe un indice.

Le niveau d'échelle `scale` est un nombre entier supérieur ou égal à 1 qui permet de partionner l'image en carrés disjoints de `scale * scale` pixels. L'objectif est de réduire la taille de vos instances (à vous de voir comment !), qui risque d'être trop grosses selon la taille de l'image choisie.

Enfin, associer à chaque couleur présente dans l'image un indice permet de représenter le coût potentiel pour traverser cette zone (i.e. ce pixel). Par exemple, pour se déplacer en `(x+1,y)` depuis `(x,y)` en se déplaçant d'un pixel vers la droite, le coût est donné par `coût[(x,y)] + coût[(x+1,y)]`. On peut voir ceci comme le poids de l'arête `(x,y) - (x+1,y)`. Attention, il ne s'agit pas forcément d'un chemin de coût minimum pour aller en `(x+1,y)` depuis `(x,y)`. Enfin, notons qu'il n'est pas toujours possible de traverser un pixel (par exemple s'il se situe sur de l'eau) : nous utiliserons dans ce cas l'indice `0` pour la couleur correspondante.

Si on reprend l'exemple du fichier PPM ci-dessus, on s'aperçoit que celui-ci contient seulement 3 couleurs différentes :

1. (0, 0, 255) : bleu ;
2. (0, 255, 0) : vert ;
3. (255, 0 ,0) : rouge.

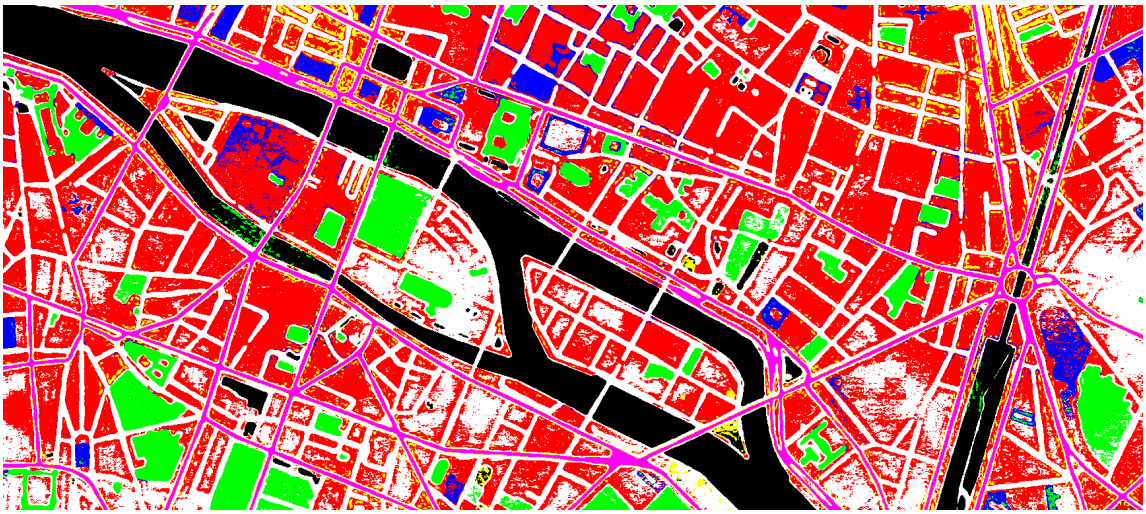
Un exemple de fichier de configuration `config.txt` associé à cette image peut être par exemple :

```
1 # échelle 1, i.e., on ne modifie pas l'image
0 255 0 0 # l'indice de la couleur (255,0,0) est 0
1  0 255 0  # l'indice de la couleur verte est 1
3  255 0 0
```

Dans cet exemple, on indique notamment qu'il n'est pas possible de se déplacer sur un pixel de couleur rouge.

Un exemple concret

Voici un exemple concret d'image que vous devez pouvoir manipuler :



Le fichier PPM associé à cette image peut être téléchargé ici. Si on ouvre ce fichier, on s'aperçoit notamment que l'image est de taille `1727*764` et que chaque ligne du fichier ne contient qu'un seul triplet (R,G,B), et non autant de triplets qu'il n'y a de colonnes (i.e., la largeur de l'image). Il s'agit d'un stockage correspondant à un *column-major order* (voir Wikipedia) : les 764 premiers triplets représentent la première colonne, les triplets 765 à 1528 la deuxième colonne et ainsi de suite. Cette image contient sept couleurs différentes :

1. (0,0,0) : noir ;
2. (0,0,255) : bleu ;
3. (0,255,0) : vert ;
4. (255,0,0) : rouge ;
5. (255,0,255) : magenta ;
6. (255,255,0) : jaune ;
7. (255,255,255) : blanc.

On peut ainsi remarquer que la couleur noire représente les zones d'eau (comme la Seine), la couleur magenta les routes principales etc. On peut ainsi associer le fichier de configuration suivant :

```
1 # échelle 1, i.e., on ne modifie pas l'image
0 0 0 0 # la couleur noire a un indice de 0
1  255 0 255
3  255 255 255
4 255 0 0
7 0 0 255
10 0 255 0
17 255 255 0
```

Création d'autres images PPM

Vous pouvez facilement créer d'autres instances de votre choix. Pour cela, vous devez :

1. aller sur uMap puis cliquer sur "Créer une carte" ;
2. changer le fond de carte (menu vertical à droite) et choisir "OSM Watercolor" ;
3. faire une capture d'écran de la zone souhaitée dans un fichier `file.png` .

Pour exporter l'image en PPM :

1. ouvrir `file.png` avec GIMP ;
2. Fichier > Exporter sous > `file.ppm` > Exporter ;
3. choisir ASCII.

Il vous faudra ensuite lancer la commande suivante depuis un terminal :

```
foo@bar:~$ awk '{ ++x; if(x<=4) print $0; else { l1=$0; getline; l2=$0; getline; prin
```

Ceci permet de récupérer le fichier PPM final dans un fichier `res.ppm` .

On peut également modifier l'image et réduire le nombre de couleurs sous GIMP :

1. Image > Mode > Couleurs indexées ;
2. Générer une palette optimale et choisir le nombre maximal de couleurs.

Enfin, on peut pour chacune des couleurs indexées choisir la valeur `(R,G,B)` :

1. Fenêtres > Fenêtres ancrables > Palette des couleurs indexées ;
2. Cliquer sur la couleur en question ;
3. Dans Notation HTML, écrire la couleur désirée en hexadécimal (000000 à ffffff).

Pensez ensuite à bien exporter de nouveau l'image au format PPM et à lancer la commande `awk` .

Le format graphe

Votre programme doit également pouvoir lire des graphes en entrée. Ceux-ci sont représentés sous forme de liste d'adjacence comme ceci :

```
0: 3 5 7
1: 2 3
2: 1 4 ...
...
n-1: ...
```

Pour générer des exemples de graphes, vous pourrez utiliser gengraph.c dont la documentation est disponible ici. Pour compiler gengraph sous Linux :

```
foo@bar:~$ gcc gengraph.c -lm -lbsd -o gengraph
```

ou sous MacOS :

```
foo@bar:~$ gcc gengraph.c -o gengraph
```

Vous pouvez par exemple générer une grille de taille `3x3` en faisant :

```
foo@bar:~$ ./gengraph mesh 3 3 -format list
0: 1 3
1: 0 2 4
2: 1 5
3: 0 4 6
4: 1 3 5 7
5: 2 4 8
6: 3 7
7: 4 6 8
8: 5 7
```

Vous pouvez aussi tester `./gengraph mesh 3 3 -format list -header` qui vous donnera des informationss complémentaires, et `./gengraph -list` pour la liste des 207 graphes que vous pouvez visualiser avec, par exemple:

```
foo@bar:~$ gengraph mesh 10 10 -dele 0.2 -visu; open g.pdf
```

L'option `dele 0.2` permet de supprimer chaque arête du graphe généré avec probabilité `0.2` . L'option `delv p` fonctionne de façon similaire.

On considérera alors que les coûts sont tous égaux (à 1 par exemple) pour tous les sommets du graphe. Il n'y a alors pas de fichier de configuration associé.

Énoncé plus formel

Ainsi, on voit bien que l'on peut représenter chaque instance sous forme de graphe. Il vous appartient de modéliser le problème comme un problème de graphe. La problématique qui nous intéresse est donc la suivante :

Étant donné un graphe dont les sommets sont pondérés, et un nombre réel positif `r` , on souhaite placer le plus petit nombre de centres de vaccinations de telle sorte que chaque personne (peu importe sa localisation géographique) soit à une distance au plus `r` d'un centre vaccination (à vous de définir la notion de distance).

Supposons maintenant que l'on dispose d'un budget limité permettant de construire au plus k centres. La question est donc ce savoir où placer ces k centres de façon à minimiser la distance maximale d'un point (i.e. un pixel de l'image) à un centre.