Projet : Réorganisation d'un réseau de fibres optiques

Nous considérons dans ce projet la réorganisation d'un réseau de fibres optiques d'une agglomération. L'énoncé de ce projet se subdivise en deux parties :

- Partie 1 : Reconstitution du réseau.
- Partie 2 : Réorganisation du réseau.

Chaque partie est divisée en exercices, qui vont vous permettre de concevoir progressivement le programme final. Il est conseillé de suivre les étapes données par ces différents exercices, car chaque exercice est l'application directe de notions qui ont été introduites en cours et en TD en parallèle au projet. Il est impératif de travailler régulièrement afin de ne pas prendre de retard et de pouvoir profiter des séances correspondantes pour chaque partie du projet.

L'ensemble des deux parties sera à rendre avant la soutenance (date à précisée) qui aura lieu pendant la dernière séance de TD/TP (entre le 04/05/2021 et le 10/05/2021). Sur moodle, vous trouverez un document récapitulant ce qui est attendu dans votre rendu. Pour les étudiants qui ne peuvent être présents le jour de la soutenance, vous devez contacter vos chargés de TD afin de prévoir une autre date au plus près de la semaine de rendu. Enfin, nous vous rappelons que la note de ce projet fait partie de la note du module, et est conservée en seconde session.

Remarque : Cet énoncé va être complété au fur et à mesure du temps : récupérer l'énoncé sur moodle avant chaque séance de TP.

Cadre du projet

Dans ce projet, nous considérons une agglomération dont les services municipaux désirent améliorer le réseau de fibres optiques de ses administrés. Un *réseau* un ensemble de câbles, chacun contenant un ensemble de fibres optiques et reliant des clients.

La première partie du projet consiste à reconstituer le plan du réseau de l'agglomération. En effet, plusieurs opérateurs se partagent actuellement le marché et possèdent chacun quelques fibres du réseau. Le réseau ayant régulièrement grossi, il n'existe pas à ce jour de plan complet du réseau. En revanche, chaque opérateur connaît les tronçons de fibres optiques qu'il utilise dans le réseau. En partant de l'hypothèse qu'il y a au moins une fibre utilisée par câble, il est ainsi possible de reconstituer le réseau dans son intégralité.

Une deuxième partie du travail va consister à réorganiser les attributions de fibres de chacun des opérateurs. En effet, la répartition des fibres n'ayant jamais été remise en cause, certains câbles sont sous-exploités alors que d'autres sont sur-exploités. Chaque opérateur possède une liste de paires de clients qu'il a reliés l'un à l'autre par une chaîne de tronçons de fibres optiques, suivant les disponibilités des fibres. Certaines chaînes sont donc très longues. Ces problèmes de sur-exploitation et de longueurs excessives peuvent être résolus, ou tout du moins améliorés, en réorganisant le réseau et en attribuant aux opérateurs des chaînes moins longues et mieux réparties dans le réseau : ce sera l'objet de la seconde partie du projet.

L'objectif de ce projet est de proposer à l'agglomération les meilleures méthodes possibles pour réaliser ces deux parties, et donc nous allons donc tester plusieurs algorithmes.

Modélisation et notations

Un *câble* du réseau est un fourreau (ou une gaine) contenant exactement $\gamma > 0$ fibres optiques. Les câbles relient deux points du plan. Dans le réseau, il existe deux types de points :

— un *client* est un point qui représente un client, une entreprise cliente ou encore un local technique de l'opérateur.

— un *concentrateur* est un point du réseau où se rejoigne plusieurs cables.

Un point peut être un client, un concentrateur ou les deux à la fois. Les tronçons de fibres optiques de deux câbles qui arrivent à un même concentrateur peuvent alors être reliés à ce point. Les tronçons de fibres optiques ainsi reliés bout à bout forment alors des *chaînes* dans le réseau. Une chaîne relie toujours deux points du plan : on appelle ce couple de points *une commodité* (ce sont les extrémités de la chaîne). Il existe plusieurs opérateurs dans l'agglomération et chaque opérateur possède plusieurs chaînes de fibres optiques.

Un exemple

La figure 1 représente une instance de notre problème.

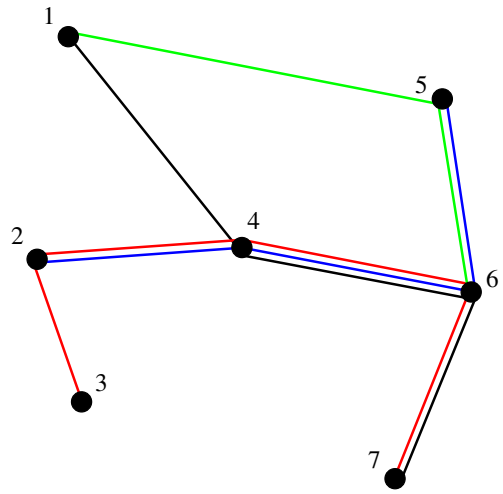

FIGURE 1 – Un exemple de réseau.

Elle décrit un réseau composé de 7 points et 4 chaînes représentées chacune par une couleur : une chaîne $(1, 4, 6, 7)$ reliant la commodité $(1, 7)$, une chaîne $(2, 4, 6, 5)$ reliant la commodité $(2, 5)$, une chaîne $(3, 2, 4, 6, 7)$ reliant la commodité $(3, 7)$ et la chaîne $(4, 1, 5, 6)$ reliant la commodité $(4, 6)$. On peut remarquer que :

— les points 1, 3 et 7 sont uniquement clients car ils sont au bout d'une chaîne sans être un point intérieur d'une chaîne.

— le point 4 est uniquement un concentrateur car c'est toujours un point intérieur d'une chaîne.

— les autres points sont à la fois des clients et des concentrateurs.

Si l'on regarde non plus la liste des chaînes, mais le réseau dans sa globalité, on peut noter qu'il existe seulement 7 câbles dans ce réseau. Par exemple, dans le câble $(1, 4)$, une seule fibre est utilisée. Dans le câble $(4, 6)$, trois fibres sont utilisées. Ce dessin ne donne pas l'indication du nombre γ du nombre maximal de fibres utilisables par câble, mais on peut déduire que $\gamma \geq 3$.

Instances des problèmes

Les instances que nous allons manipuler dans ce projet sont soit issues de la base TSPLib [1], soit issues de la base du 9ème challenge DIMACS [2]. Ces deux bases contiennent des instances de réseaux fréquemment utilisées pour tester l'efficacité d'algorithmes concernant les problèmes de réseaux. La plupart correspondent à des villes ou des pays (Burma=Birmanie, NY=New-York, d=Allemagne, att=USA, etc.), d'autres proviennent de réseaux non géographiques (réseaux électroniques, etc.).
Ces instances ont été adaptées afin d'être utilisables pour ce projet. Sur le site, vous les trouverez sous la forme de fichiers-texte classés selon leur nombre de points. Il est demandé dans le projet d'utiliser ces instances pour tester vos algorithmes.

Lecture, stockage et affichage des données

Exercice 1 – Manipulation d'une instance de "Liste de Chaînes"

Dans ce premier exercice, nous allons construire une bibliothèque de manipulations d'instances : lecture et écriture de fichier, affichage graphique de réseaux, calcul de la longueur totale des chaînes, et calcul du nombre de points.

Une instance de "Liste de Chaînes" est simplement donnée par un nombre de chaînes et par la liste des chaînes. Chaque chaîne est une liste de points du plan. Chaque point est repéré par ses coordonnées (abscisse x et ordonnée y). Chaque instance est donnée par un fichier texte d'extension `.cha` qui respecte le format donné par l'exemple `00014_burma.cha` suivant :

```
1  NbChain: 8
2  Gamma: 3
3
4  0 3 25.23 97.24 14.05 98.12 16.47 94.44
5  1 3 14.05 98.12 16.47 96.1 20.09 92.54
6  2 3 16.3 97.38 16.53 97.38 25.23 97.24
7  3 4 16.47 96.1 20.09 94.55 22.39 93.37 25.23 97.24
8  4 4 22.39 93.37 20.09 94.55 17.2 96.29 16.3 97.38
9  5 5 14.05 98.12 16.47 94.44 20.09 92.54 22.39 93.37 21.52 95.59
10 6 5 14.05 98.12 16.47 94.44 20.09 92.54 22.39 93.37 22 96.05
11 7 3 22.39 93.37 20.09 92.54 16.47 96.1
```

Les deux premières lignes donnent le nombre de chaînes et le nombre maximal γ de fibres optiques par câble. Les différentes chaînes du réseau sont ensuite données. Chaque ligne de chaîne comporte dans l'ordre, le numéro de la chaîne, le nombre de points de la chaîne et la liste des points. Chaque point

1. http ://comopt.ifi.uni-heidelberg.de/software/TSPLIB95
2. http ://www.dis.uniroma1.it/challenge9/

est donné par ses coordonnées (abscisse et ordonnée). Chaque chaîne peut donc être vue comme une liste chaînée de points. On utilisera la structure de données suivante dans le fichier (fourni) `Chaine.h`.

```
#ifndef __CHAINE_H__
#define __CHAINE_H__
#include<stdio.h>

/* Liste chainee de points */
typedef struct cellPoint{
  double x,y;                   /* Coordonnees du point */
  struct cellPoint *suiv;       /* Cellule suivante dans la liste */
} CellPoint;

/* Celllule d une liste (chainee) de chaines */
typedef struct cellChaine{
  int numero;                   /* Numero de la chaine */
  CellPoint *points;            /* Liste des points de la chaine */
  struct cellChaine *suiv;      /* Cellule suivante dans la liste */
} CellChaine;

/* L'ensemble des chaines */
typedef struct {
  int gamma;                    /* Nombre maximal de fibres par cable */
  int nbChaines;                /* Nombre de chaines */
  CellChaine *chaines;          /* La liste chainee des chaines */
} Chaines;

Chaines* lectureChaines(FILE *f);
void ecrireChaines(Chaines *C, FILE *f);
void afficheChainesSVG(Chaines *C, char* nomInstance);
double longueurTotale(Chaines *C);
int comptePointsTotal(Chaines *C);

#endif
```

On peut remarquer que :

- le struct `cellPoint` est un élément de la liste des points et contient les coordonnées d'un point.
- le struct `cellChaine` est un élément de la liste des chaînes et contient un numéro de chaîne et la liste des points.
- l'ensemble des chaînes est un struct contenant le nombre maximal de fibres par câble, le nombre de chaînes et la liste des chaînes.

Q 1.1 Dans un fichier `Chaine.c`, implémenter une fonction `Chaines* lectureChaine(FILE *f);` qui permet d'allouer, de remplir et de retourner une instance de notre structure à partir d'un fichier.

Q 1.2 Implémenter une fonction `void ecrireChaine(Chaines *C, FILE *f);` qui écrit dans un fichier le contenu d'une `Chaines` en respectant le même format que celui contenu dans le fichier d'origine. Créer un main `ChaineMain.c` permettant d'exécuter les fonctions de lecture et d'écriture que vous venez de définir (vous pouvez utiliser la ligne de commande pour passer le nom du fichier contenant l'instance).

Remarques :

— La fonction d'écriture permet de recréer le fichier de données, mais l'ordre des points et des chaînes sera inversé (à cause des insertions en tête de liste).

— Le but de cette fonction d'écriture est de tester le code de votre fonction de lecture sur plusieurs instances, avant d'attaquer la suite du projet.

Q 1.3 On désire donner une représentation graphique des instances. Pour cela, nous allons utiliser le format d'images SVG (Scalable Vector Graphics) qui est de plus en plus employé pour décrire des graphiques simples et qui est très utilisé pour internet. Votre code va créer un fichier au format SVG pour html qui sera ainsi lu directement par votre explorateur internet préféré. Nous vous proposons sur moodle une petite librairie C très très simple qui crée un fichier SVG avec extension html. Il s'agit d'un struct `SVGwriter` qui est manipulé par des méthodes permettant de créer le fichier, ajouter des lignes et des points et changer de couleurs. Il y a également une génération aléatoire de couleur de segments (il faut initialiser la génération aléatoire pour obtenir des couleurs diverses). Dans votre fichier `Chaine.c`, ajouter la fonction `void afficheChaineSVG(Chaines *C, char* nomInstance);` qui permet de créer le fichier SVG en html à partir d'un struct `Chaines`. Cette fonction vous est donné dans le fichier "affichageSVG.txt" sur moodle. Tester cette fonction d'affichage dans votre fichier `ChaineMain.c`.

Q 1.4 Implémenter les fonctions :
- `double longueurChaine(CellChaine *c);` qui calcule la longueur physique d'une chaîne.
- `double longueurTotale(Chaines *C);` qui calcule la longueur physique totale des chaînes.

Pour calculer la longueur d'une chaîne, il faut sommer les distances entre les différents points qui composent la chaîne. Pour rappel, la distance entre entre deux points A et B de coordonnées (x_A, y_A) et (x_B, y_B) est donnée par $d(A,B) = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$.

Q 1.5 Écrire la fonction `int comptePointsTotal(Chaines *C);` qui donne le nombre total d'occurrences de points (les points qui apparaissent plusieurs fois sont comptés plusieurs fois).

Reconstitution du réseau

Le but de cette partie est de reconstituer efficacement le réseau à partir des chaînes. À partir de la liste des chaînes, il s'agit de :
- trouver la liste des nœuds du réseau (on élimine les redondances de points). Ainsi, à une coordonnée donnée, il ne peut y avoir qu'un seul nœud.
- identifier tous les câbles qui sont issus d'un nœud. Ceci permet de conserver la liste des nœuds voisins à un nœud donné.
- de récupérer et conserver la liste des commodités du réseau.

L'algorithme de reconstitution est très simple. Le pseudo-code est le suivant :

```
On utilise un ensemble de nœuds V qui est initialisé vide: V ← ∅
On parcourt une à une chaque chaîne:
    Pour chaque point p de la chaîne:
        Si p ∉ V (on teste si le point n'a pas déjà été rencontré auparavant)
            On ajoute dans V un nœud correspond au point p.
        On met à jour la liste des voisins de p et celles de ses voisins.
    On conserve la commodité de la chaîne.
```

Dans cette partie, on s'intéresse à l'optimisation du test "$p \notin V$". Pour cela, on va étudier trois méthodes qui correspondent à trois structures de données pour implémenter l'ensemble V : une liste chaînée, une table de hachage et des arbres.

Exercice 2 – Première méthode : stockage par liste chaînée

Dans cet exercice, on désire implémenter l'algorithme de reconstitution de réseau en codant l'ensemble des nœuds du réseau par une liste chaînée. Pour cela, nous avons besoin de définir une structure pour manipuler un réseau. Un réseau se présente comme un ensemble de nœuds, de câbles et de commodités. Dans cette structure :

- Chaque nœud v sera repéré par ses coordonnées, et on connaîtra la liste des pointeurs sur nœuds qui sont reliés à v par un câble. Lors de la reconstitution du réseau, on attribuera à chaque nœud un numéro entier unique qu'on lui choisira incrémentalement.
- Chaque câble est donné par des pointeurs sur ses deux nœuds extrémités.
- Chaque commodité est une paire de pointeurs sur les nœuds du réseau qui doivent être reliés par une chaîne.

Ainsi, pour stocker les données du réseau, on utilisera la structure de données suivante (fichier fourni), définie dans le fichier `Reseau.h` :

```
#ifndef __RESEAU_H__
#define __RESEAU_H__
#include "Chaine.h"

typedef struct noeud Noeud;

/* Liste chainee de noeuds (pour la liste des noeuds du reseau ET les listes des
    voisins de chaque noeud) */
typedef struct cellnoeud {
    Noeud *nd;                /* Pointeur vers le noeud stock\'e */
    struct cellnoeud *suiv;          /* Cellule suivante dans la liste */
} CellNoeud;

/* Noeud du reseau */
struct noeud{
   int num;                          /* Numero du noeud */
   double x, y;                      /* Coordonnees du noeud*/
   CellNoeud *voisins;               /* Liste des voisins du noeud */
};

/* Liste chainee de commodites */
typedef struct cellCommodite {
    Noeud *extrA, *extrB;        /* Noeuds aux extremites de la commodite */
    struct cellCommodite *suiv;      /* Cellule suivante dans la liste */
} CellCommodite;

/* Un reseau */
typedef struct {
    int nbNoeuds;                     /* Nombre de noeuds du reseau */
    int gamma;                        /* Nombre maximal de fibres par cable */
    CellNoeud *noeuds;                /* Liste des noeuds du reseau */
    CellCommodite *commodites;        /* Liste des commodites a relier */
} Reseau;

Noeud* rechercheCreeNoeudListe(Reseau *R, double x, double y);
Reseau* reconstitueReseauListe(Chaines *C);
void ecrireReseau(Reseau *R, FILE *f);
int nbLiaisons(Reseau *R);
int nbCommodites(Reseau *R);
void afficheReseauSVG(Reseau *R, char* nomInstance);
#endif
```

Cette structure permet de stocker un `Reseau` comme une liste chaînée de `Noeud` et une liste chaînée de `Commodite`. Chaque `Noeud` v est donné par son numéro, ses coordonnées et la liste chaînée des nœuds voisins, c'est-à-dire les nœuds qui sont liés au noeud v par un câble. Une `Commodite` est simplement donnée par les deux nœuds qui seront à relier par une chaîne.

Q 2.1 Créer une fonction `Noeud* rechercheCreeNoeudListe(Reseau *R, double x, double y);` qui retourne un `Noeud` du réseau R correspondant au point (x, y) dans la liste chaînée `noeuds` de R. Noter que si ce point existe dans `noeuds`, la fonction retourne un nœud existant dans `noeuds` et que, dans le cas contraire, la fonction crée un nœud et l'ajoute dans la liste des nœuds du réseau de R. Le numéro d'un nouveau nœud est simplement choisi en prenant le nombre `nbNoeuds+1` (just'avant de mettre à jour à la valeur `nbNoeuds`).

Q 2.2 Implémenter une fonction `Reseau* reconstitueReseauListe(Chaines *C);` qui reconstruit le réseau R à partir de la liste des chaînes C comme indiqué dans le pseudo-code donné au début de cette partie. Utiliser directement la liste chaînée `noeuds` du réseau pour effectuer les tests de type "$p \notin V$", en exploitant la fonction de question précédente.

Q 2.3 Créer un programme main `ReconstitueReseau.c` qui utilise la ligne de commande pour prendre un fichier `.cha` en paramètre et un nombre entier indiquant quelle méthode (parmi les 3 de cette partie) l'on désire utiliser.

Exercice 3 – Manipulation d'un réseau

On veut à présent construire des méthodes pour manipuler et afficher un struct `Reseau`. Pour cela, on va stocker sur disque un `Reseau` en utilisant le format illustré par l'instance `00014_burma` qui est donné par le fichier suivant `00014_burma.res` (obtenu par l'exercice précédent).

```
NbNoeuds: 12
NbLiaisons: 15
NbCommodites: 8
Gamma: 3

v 12 16.530000 97.380000
v 11 25.230000 97.240000
v 10 20.090000 94.550000
v 9 17.200000 96.290000
v 8 16.300000 97.380000
v 7 21.520000 95.590000
v 6 14.050000 98.120000
v 5 16.470000 94.440000
v 4 22.000000 96.050000
v 3 22.390000 93.370000
v 2 20.090000 92.540000
v 1 16.470000 96.100000

l 8 12
l 11 12
l 6 11
l 3 11
l 1 10
l 3 10
l 9 10
l 8 9
l 3 7
l 1 6
```

```
l 5 6
l 2 5
l 3 4
l 2 3
l 1 2

k 5 11
k 2 6
k 11 8
k 11 1
k 8 3
k 7 6
k 4 6
k 1 3
```

Dans ce fichier :

- Les quatre premières lignes donnent le nombre de nœuds du réseau, le nombre de câbles (liaisons), le nombre de commodités, et le nombre maximal γ de fibres optiques par câble.
- Ensuite, les lignes commençant par "v" donnent les nœuds du réseau. Les nœuds sont repérés par leur numéro et leurs deux coordonnées.
- Les lignes commençant par un "l" contiennent une liaison (un câble) donnée par les numéros de ses deux extrémités.
- Les lignes commençant par un "k" correspondent à une commodité, c'est-à-dire une paire de numéros de nœuds qui devront être reliés par une chaîne.

Q 3.1 Pour écrire un tel fichier, commencer par implémenter les fonctions `nbCommodites(Reseau *R);` et `int nbLiaisons(Reseau *R);` qui comptent le nombre de commodités et de liaisons du réseau `R`.

Q 3.2 Implémenter une fonction `void ecrireReseau(Reseau *R, FILE *f);` qui écrit dans un fichier le contenu d'un `Reseau` en respectant le même format du fichier `00014_burma.res`.

Q 3.3 Dans le fichier `affichageReseau.txt`, récupérer la fonction `void afficheReseauSVG(Reseau *R, char* nomInstance);` qui permet de créer un fichier SVG en html pour visualiser un réseau. Tester votre code sur plusieurs instances en le comparant avec l'affichage des chaînes pour valider (en partie) vos fonctions.

Attention : Le projet n'est pas fini. Le reste du sujet sera mis en ligne avant la prochaine séance de TP ! !