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 grandes phases, divisées chacune en exercices, qui vont vous permettre de concevoir progressivement le programme final. Chacun des exercices est le sujet des séances de TME de 3 à 10. Il est conseillé de suivre les étapes données par ces différents exercices, car chaque exercice est l'application directe de notions qui sont 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.

La première phase du projet est à rendre avant le TME 8 et la dernière phase en semaine 11. Il est impératif de rendre à temps :
- Lors du TME 8, les solutions de la première phase seront diffusées à tous afin de pouvoir tous attaquer la phase 2 sur les mêmes bases,
- en semaine 11, vous devez rendre votre projet à votre chargé de TD lors de la séance de TME . A cette occasion, vous lui montrerez votre code, son fonctionnement et ses performaces. Pour le rendu, voir document en début de poly. **Attention :** Pour les étudiants qui ne peuvent être présents en TD/TME pour les séances de rendus, vous devez contacter vos chargés de TD afin de prévoir une autre date au plus près de la semaine de rendu.
- pour la phase 2, les binômes qui auront eu les meilleurs résultats auront un bonus en point sur leur note.

Attention : Cet énoncé peut connaître des petites évolutions afin d'apporter des précisions ou des indications : venez en Cours/TD/TME pour les connaître et consulter fréquemment la page du module : https ://www.licence.info.upmc.fr/lmd/licence/2011/ue/LI213-2012fev/

Introduction

• *Cadre du projet*

Une agglomération désire améliorer le réseau de fibres optiques de ses administrés.
Une première phase de travail va consister à reconstituer le plan du réseau. En effet, plusieurs opérateurs se partagent actuellement ce marché en possédant chacun quelques fibres des câbles du réseau. Le réseau ayant grossi régulièrement, il n'existe pas à ce jour de plan complet de ce 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 phase de travail va consister à réorganiser les attributions de fibres de chacun des opérateurs. En effet, la répartition des fibres n'a jamais été remise en cause jusqu'ici : certains câbles sont sous-exploités au détriment d'autres sur-exploités. En fait, chaque opérateur possède une liste de paires de clients qu'il a relié l'un à l'autre par une chaîne de tronçons de fibres optiques. Comme les

attributions de fibres optiques n'ont jamais été remises en cause, certaines chaînes sont très longues à cause de cette congestion. Ces problèmes de congestion et de longueurs excessives peuvent être résolus, ou tout du moins améliorés, en attribuant aux opérateurs des chaînes moins longues et mieux réparties dans le réseau.

L'objectif de ce projet en deux phases est de proposer à l'agglomération les meilleures méthodes possibles pour réaliser ces deux phases. Nous allons donc tester plusieurs algorithmes pour chacune des phases.

• *Modélisation et notations*

Un *câble* du réseau est un fourreau contenant exactement γ fibres optiques. Les câbles relient deux points du plans. Un point peut être un client, ou un concentrateur ou à la fois un client et un concentrateur. On appelle *clients* du réseau des points qui ont un rôle important : des entreprises clientes de l'opérateurs, des locaux techniques de l'opérateurs,... Un *concentrateur* est un point du réseau où se rejoignent plusieurs câbles : les tronçons de fibres optiques de deux câbles qui arrivent à un même concentrateur peuvent être alors reliées à ce point. Les tronçons de fibres optiques ainsi reliés bout à bout forment alors des *chaînes* dans le réseau. Une chaîne relie en fait deux points du plan : on appelle ce couple de point *une commodité*.

Il existe plusieurs opérateurs dans l'agglomération et chaque opérateur possède plusieurs chaînes de fibres optiques. On considèrera ici l'ensemble

• *Un exemple*

La figure 1 indique une instance de notre problème. Elle décrit un réseau de 7 points qui sont tous un rôle dans le réseau. Il y a 3 chaînes : une chaîne $(1, 4, 6, 7)$ reliant la commodité $(1, 7)$, une chaîne $(2, 4, 5, 6)$ reliant la commodité $(2, 6)$ et la chaîne $(3, 2, 4, 6, 7)$ reliant la commodité $(3, 7)$. On peut remarquer que les points 2, 4 et 6 sont des concentrateurs ; que le point 2 est à la fois un client et un concentrateur ; que le point 7 est aux extrémités de deux chaînes distinctes.

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

• *Instances des problèmes*

Les instances que nous allons manipuler dans ce projet sont de dimensions différentes. Elles sont issues soit de la base TSPLib [1], soit de la base du 9ème challenge DIMACS [2]. Ces deux bases contiennent des instances de réseau fréquemment utilisées pour tester les capacités d'algorithmes concernant les problèmes de réseaux. La plupart correpondantes à des villes ou des pays (Burma=Birmanie, NY=New-York, d=Allemagne, att=USA,...), d'autres proviennent de réseaux non géographiques (réseaux

[1] http ://comopt.ifi.uni-heidelberg.de/software/TSPLIB95

[2] http ://www.dis.uniroma1.it/∼ challenge9/download.shtml#benchmark

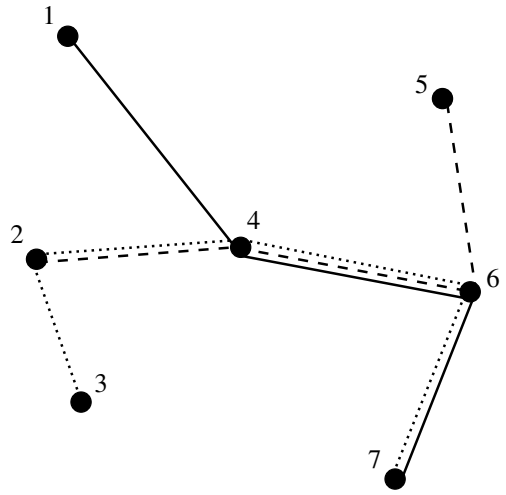

FIG. 1 – Un exemple de réseau

électroniques,...).
Ces instances ont été adaptées afin d'être utilisables pour ce projet. Sur le site, vous les trouverez sous la forme de fichiers texte compressés et classés selon leur nombre de points. Il est demandé dans le projet d'utilser ces instances pour tester vos programmes et en déduire des statistiques.

Phase 1 : Reconstitution du réseau

L'objectif de cette phase est de reconstituer le réseau à partir de la liste des chaînes qui le parcourt.

Exercice 1 – Manipulation d'une instance de Liste de Chaînes (TME 3)

Une instance de Liste de Chaînes est donc 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. Chaque instance est donnée par un fichier texte d'extension `.cha` qui respecte le format donné par l'exemple `00014_burma.cha` suivant

```
1  c NbChain: 8
2  c Gamma: 4
3  c
4  h 0 6 : 25.23 97.24 / 14.05 98.12 / 16.3 97.38 / 16.47 94.44 / 16.47 96.1 / 20.09
       92.54 /
5  h 1 5 : 14.05 98.12 / 16.3 97.38 / 16.47 94.44 / 16.47 96.1 / 22.39 93.37 /
6  h 2 5 : 16.47 96.1 / 16.47 94.44 / 16.3 97.38 / 14.05 98.12 / 25.23 97.24 /
7  h 3 4 : 14.05 98.12 / 19.41 97.13 / 22 96.05 / 20.09 94.55 /
8  h 4 2 : 22.39 93.37 / 16.47 96.1 /
9  h 5 3 : 14.05 98.12 / 19.41 97.13 / 22 96.05 /
10 h 6 3 : 16.3 97.38 / 14.05 98.12 / 25.23 97.24 /
11 h 7 4 : 22.39 93.37 / 16.47 96.1 / 16.47 94.44 / 16.3 97.38 /
```

Les lignes commençant par c sont des lignes de commentaires. Les lignes commençant par h correspondent à une chaîne. Chaque ligne de chaîne comporte dans l'ordre, la lettre h, le numéro de la

chaîne, le nombre de points de la chaîne. La liste des points dans l'ordre, chaque point étant donné par ses coordonnées.

Une Liste de chaînes peut donc être vu comme une liste chaînée de Chaînes ; et chaque chaîne comme une liste chaînées de points. On utilisera la structure de données suivantes :

```
#ifndef __CHAINE_H__
#define __CHAINE_H__

/* Structure d'un point d'une chaine */
typedef struct point{

  double x,y;  /* Coordonnees du point */

  struct point *ptSuiv;  /* Pointeur vers le point suivant dans la chaine */

} Point;


/* Element de la liste chaine des chaines */
typedef struct chaine{

  int numero; /* Numero de la chaine */

  Point *uneExtremite;   /* Une des 2 extremites de la chaine */

  struct chaine *chSuiv;  /* Lien vers la chaine suivante */

} Chaine;

/* Liste chainee des chaines */
typedef struct {

  int gamma; /* Nombre maximal de fibres par chaine */

  int nbchaine;   /* Nombre de chaines */

  Chaine *LCh;  /* La liste des chaines */

} ListeChaine;

#endif
```

On peut remarquer :
- que le struct `Chaine` contient la liste des points en donnant simplement une extrémité de la liste chaînée des points.
- que le struct `Chaine` contient un pointeur sur la chaîne suivante dans la liste des chaînes et qu'ainsi le struct `ListeChaine` est une liste chaînée de `Chaine`.

Dans ce premier exercice, nous allons construire une bibliothèque de manipulations d'instances`Chaine` : lecture du fichier et affichage graphique.

Q 1.1 Implémentez une fonction `void lecture_chaine(FILE *f, ListeChaine *L)` ; qui permet d'allouer et de remplir une instance de notre structure à partir d'un fichier d'entrée. (Vous pouvez utiliser la bibliothèque d'Entrée/Sortie qui vous a été fournie au TME1-2).

Q 1.2 Dans le but de valider votre code de lecture d'instance, construisez une fonction `void ecrit_chaine_txt(ListeChaine *L, FILE *f)` ; qui écrit dans un fichier le contenu d'une `Chaine` en respectant

le même format que celui contenu dans le fichier d'origine (cela revient à re-créer le fichier d'entrée). Tester votre code sur plusieurs instances.

Q 1.3 On désire donner une représentation graphique des instances. Pour cela, on pourrait utiliser par exemple un outil de dessin de segments (ou de graphe). Il est néanmoins difficile de tracer des instances de grandes tailles avec ces outils. Nous allons plutôt utiliser un logiciel de dessin nommé `xfig`. Un fichier d'exemple permettant d'afficher un point et un trait est disponible sur le site du module, ainsi qu'une documentation sur le logiciel xfig.
Implémentez une fonction `void ecrit_chaine_xfig(ListeChaine *L, FILE *f, int zoom, int epaisseur) ;` qui écrit le contenu texte d'un fichier xfig correspondant au dessin des chaines. On peut noter que plusieurs traits ou points sont superposés : ceci est normal et nous allons d'ailleurs en discuter dans les questions suivantes.
Suivant les instances, les points ont des coordonnées plus ou moins éloignées les unes des autres, il est utile de prévoir dans la fonction visualisation des paramètres `zoom` qui permettent de multiplier les coordonnées par une constante et `epaisseur` qui permet de définir l'épaisser des traits (attention, les coordonnées comme l'épaisseur sont des entiers en xfig).

Q 1.4 Implémentez les fonctions `double longueurChaine(Chaine *l) ;` puis `double longueurTotale (ListeChaine *L) ;` qui permette de calculer la longueur d'une chaîne et la longueur totale des chaînes de votre instance. Donnez également la fonction `int compte_point(ListeChaine *L) ;` qui donne le nombre total de points dans l'instance (en comptant les différentes occurences du même point).

Exercice 2 – Manipulation d'une instance de réseau (TME3-4)

Le but de la première phase est de reconstituer un réseau. Le reseau obtenu sera alors l'instance de départ de la phase 2. Nous allons dès à présent construire une bibliothèque de manipulation de réseau.

Un réseau se présente comme un ensemble de nœuds, de câbles et de commodités.
Chaque nœud est repéré par son numéro entier unique et par ses coordonnées. De plus, pour chaque nœud u, on connaît la liste des nœuds qui sont reliés à u par un câble.
Chaque câble est donné par les numéros de ses deux nœuds extrémités.
Une commodité est une paire de nœuds du réseaux qui doivent être reliés par une chaîne.

Une instance de réseau sera donnée par le format donné par l'exemple `00014_burma.res` suivant :

```
1  c NbNodes: 10
2  c NbCables: 9
3  c Gamma: 4
4  c
5  v 5 14.05 98.12
6  v 0 16.3 97.38
7  v 1 16.47 94.44
8  v 2 16.47 96.1
9  v 7 19.41 97.13
10 v 9 20.09 92.54
11 v 8 20.09 94.55
12 v 6 22 96.05
13 v 3 22.39 93.37
14 v 4 25.23 97.24
15 c
16 a 5 7
```

```
a 0 5
a 0 1
a 1 2
a 2 9
a 2 3
a 6 8
a 6 7
a 4 5
c
k 9 4
k 3 5
k 4 2
k 8 5
k 2 3
k 6 5
k 4 0
k 0 3
```

Dans ce format, les lignes débutant par un v contiennent les nœuds, donnés par leur numéro et leurs deux coordonnées. Les lignes commençant par un a contiennent un câble donné 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.

On peut par exemple utiliser la structure de données suivantes

```c
#ifndef __RESEAU_H__
#define __RESEAU_H__

/* Element de la liste des noeuds voisins d'un noeud du reseau */
typedef struct voisin{

  int v;  /* Numero du voisin dans le Reseau*/

  struct voisin *voisSuiv;  /* Pointeur sur le voisin suivant */

} Voisin;


/* Noeud du reseau */
typedef struct noeud{

  int u;   /* Numero du noeud dans le Reseau */

  double x, y; /* Coordonnees du noeud */

  Voisin * LVoisins; /* Liste des noeuds voisins de u*/

} Noeud;

/* Element de la liste chainee des noeuds du reseau */
typedef struct celluleLNoeud{

  Noeud *ptrnoeud; /* Pointeur sur un noeud du reseau */

 struct celluleLNoeud *noeudSuiv; /* Noeud suivant dans la liste des noeuds */

} CelluleLNoeud;


/* Element de la liste chainee des commodites du reseau */
typedef struct celluleLCommodite{
```

```
37
38    int extr1,extr2;
39
40    struct celluleLCommodite *commSuiv;
41
42  } CelluleLCommodite;
43
44
45  /* Un reseau */
46  typedef struct{
47
48    int nbNoeuds;  /* Nombre total de noeuds dans le Reseau */
49
50    int gamma;  /* Nombre maximal de fibres dans un cable */
51
52    CelluleLCommodite *LCommodites; /* Liste des commodites a relier */
53
54    CelluleLNoeud *LNoeuds;  /* Liste des noeuds du Reseau */
55
56
57  } Reseau;
58
59
60
61  #endif
```

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

Q 2.1 Implémentez une fonction `void lecture_fichier_reseau(FILE *f, Reseau *R)` ; capable de lire un fichier au format réseau et d'allouer et remplir la structure précédente.

Q 2.2 Implémentez une fonction `int compteCable(Reseau *R)` ; qui compte le nombre de câble contenu dans un réseau.

Q 2.3 Implémentez une fonction `void ecrit_reseau_disque(Reseau *R, FILE *f)` ; qui écrit sur disque le contenu d'un `Reseau` en respectant le même format que celui contenu dans le fichier d'origine (cela revient à re-créer le fichier d'entrée). Tester votre code les instances pour lesquels vous avez déjà ce fichier réseau (les autres fichiers, ce sera à vous de les obtenir lors des exos suivants).

Q 2.4 Implémentez une fonction `void ecrit_reseau_xfig(ListeChaine *L, FILE *f)` ; qui écrit le contenu texte d'un fichier xfig correspondant au dessin du réseau. Pour les instances que vous avez déjà, vérifier visuellement que la visualisation des chaînes et du réseau est bien identique.

Q 2.5 Implémentez une fonction `int recherche_voisin(Noeud *nd, int u)` ; qui retourne le numéro du sommet voisin dans un réseau.

Exercice 3 – Reconstruction du réseau 1 : liste maintenue triée (TME 4)

Le but de cet exercice est de reconstituer le réseau à partir d'une liste de chaînes. Pour cela, nous allons ici utiliser une liste de nœuds maintenue triée. Cette liste va tout simplement être la liste `LNeouds` du réseau en cours de construction (en effet, au début cette liste est vide).
Le principe va être le principe simple suivant. Les points de chacune des chaînes vont être testé pour

être inséré l'un après l'autre dans `LNoeuds` : soit il existe déjà un nœud correspondant à ce point dans `LNoeuds`, soit on crée et on ajouter un tel nœud.

Q 3.1 Implémentez une fonction `Noeud* recherche_cree_nœud(Reseau *R, double x, double y) ;` retournant un `Noeud` du réseau R correspondant au point (x, y) dans la liste `LNoeuds` de R. Noter que si ce point existe dans R, la fonction retourne un point existant dans R et que, dans le cas contraire, la fonction crée un nœud et l'ajoute dans `LNoeuds` à la bonne position, de manière à ce que R reste triée.

Q 3.2 Donnez une fonction `int compare(double x1, double y1, double x2, double y2) ;` indiquant qu'un point (x_1, y_1) est avant un point (x_2, y_2) si $x_1 < x_2$ ou si $|x_1 - x_2| < \epsilon$ avec $y_1 < y_2$.

Q 3.3 Implémentez une fonction `void recree_reseau(ListeChaine *L, Reseau *R) ;` qui reconstruit le réseau R à partir de la liste des chaînes L. Pour cela, on applique la fonction `recherche_cree_noeud` pour chaque sommet de chaque chaîne. De plus, cette fonction permet également de remplir la liste `LCommodites` des commodités.

Q 3.4 Tester la fonction précédente de récréation des réseaux sur les instances pour lesquels vous avez déjà les fichiers solutions. Utiliser également la fonction pour les autres instances. Quelle est la complexité de cette fonction en fonction du nombre total de sommets de l'instance.

Exercice 4 – Reconstruction du réseau 2 : table de hachage (TME 5)

Pour cet exercice, nous allons reproduire les question de l'exo 3 en utilisant cette fois une table de hachage à la place de la liste maintenue triée.
Cette table de hachage H va contenir des pointeurs sur `Noeud`. Comme cette table de hachage H ne fait pas partie de la structure du réseau, à chaque fois que l'on ajoute un nouveau `Noeud` dans la liste, il sera nécessaire de l'ajouter en plus en tête dans la liste `LNoeuds` de R.
Dans ce cas de structure, on espère que l'utilisation d'une table de hachage va faire gagner du temps lors de l'insertion et de la recherche d'un point dans la structure.

Q 4.1 Donnez une structure `Hachage` qui permet d'implémenter une structure de liste de hachage par chaînage. La valeur à stocker est donnée par les coordonnées (x, y) d'un point. Vous utiliserez la fonction clef $f(x, y) = y+(x+y)(x+y+1)/2$: déterminer le nombre K de clefs donné par cette fonction. Pour une table de hachage de m cases, on utilisera la fonction de hachage $h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor$ où $A = \frac{\sqrt{5}-1}{2}$ pour toute clef k entre 0 et K. Vous testerez expérimentalement plusieurs valeurs de m afin de déterminer la valeur la plus appropriée.
Il est également possible d'utiliser des fonctions de hachage bien différente : vous pouvez également proposer d'autres fonctions.

Q 4.2 Implémentez une fonction `Noeud* recherche_cree_noeud_hachage(Reseau *R, Hachage *H, double x, double y) ;` qui retourne un `Noeud` du réseau R correspondant au point (x, y) dans la table de hachage de R. Noter que si ce point existe dans H, la fonction retourne un point existant dans H et que, dans le cas contraire, la fonction crée un nœud et l'ajoute dans H ainsi que l'ajoute en tête de la liste `LNoeuds` de R.

Q 4.3 Implémentez une fonction `void recree_reseau_hachage(ListeChaine *L, Reseau *R) ;` qui reconstruit le réseau R à partir de la liste des chaînes L en utilisant la table de hachage H.

Q 4.4 Tester la fonction précédente de récréation des réseaux sur les instances pour lesquels vous

avez déjà les fichiers solutions. Utiliser également la fonction pour les autres instances. Comparer les résultats obtenus avec la méthode précédentes pour valider vos deux méthodes l'une par rapport à l'autre.

Exercice 5 – Reconstruction du réseau 3 : Arbre binaire équilibré (TME 6 et 7)

Pour cet exercice, nous allons reproduire les question de l'exo 3 en utilisant cette fois un arbre binaire de recherche équilibré (ABRe).

Q 5.1 Donnez une structure *ABRe* qui contient des pointeurs de `Noeud` repéré chacun par la double clef (x, y). On utilisera la fonction `compare` pour classer les éléments de l'ABRe.

Q 5.2 Donnez la fonction de recherche d'un élément dans un ABRe. Donner la fonction d'insertion d'un élément dans l'ABRe.

Q 5.3 Implémentez une fonction `void recree_reseau_ABRE(ListeChaine *L, Reseau *R)` ; qui reconstruit le réseau R à partir de la liste des chaînes L en utilisant un ABRe.

Q 5.4 Comparer les résultats obtenus avec les méthode précédentes pour valider les méthodes les unes par rapport aux autres. Quelle est la complexité de cette méthode ?

Q 5.5 Construisez des statistiques (courbes) en mettant le temps CPU en fonction du nombre total de points dans la liste des chaînes. Comparez les temps nécessaires pour la reconstruction des réseaux par les trois méthodes proposées. Construisez des statistiques (courbes) en mettant le temps CPU en fonction du nombre total de points dans la liste des chaînes.

Q 5.6 Analysez et commentez les résultats obtenus.

RENDU SEMAINE 8 : Vous devez rendre les résultats obtenus pour la phase I en semaine 8. Pour cela, vous enverrez à votre chargé de TD un fichier .tar les fichiers source, ainsi que les fichiers réseaux que vous avez obtenus par vos 3 méthodes pour 3 instances. Vous choisirez les 3 instances parmi les plus grandes que vous aurez pu résoudre. Le fichier .tar contient également un rapport de 2-3 pages correspondant à vos courbes expérimentales.
Nous vous incitons également à commencer le rapport final qui sera à rendre en semaine 11.

Phase 2 : Réorganisation du réseau

Pour cette seconde phase, nos instances de départ ne sont plus les listes de chaînes, mais les réseaux. C'est-à-dire que l'on connaît les points, les câbles et les commodités d'un réseau, et que l'on veut donner les chaînes correspondant à chacune des commodités.
Pour cette réorganisation, nous allons étudier deux cas bien différents.
Le premier cas est un problème facile : nous allons considérer que les câbles peuvent contenir un nombre infini de fibres optiques. Dans ce cas, la solution optimale pour la réorganisation du réseau et de tracer des chaînes qui correspondent à des plus courtes chaînes entre les deux points d'une commodité.
Le deuxième cas est un problème difficile que nous n'allons pas chercher à résoudre optimalement : nous considérerons que chaque câble possède un nombre maximale γ de fibres optiques. Dans ce cas, une solution ne peut pas faire passer qu'au plus γ chaînes par câble.

Exercice 6 – Un tas pour l'algorithme de Dijkstra (TME 8)

Dans le but de pouvoir déterminer efficacement des plus courtes chaînes, nous allons implémenter l'algorithme de Dijkstra qui utilise un tas. Nous allons donc tout d'abord construire un tas spécifique à l'algorithme de Dijkstra.
Le tas que l'on va considérer contient un élément composé d'un entier i et d'un réel c. Le réel c sera appelé clef et l'entier i s'appelle le numéro. Le tas a pour but de rechercher rapidement l'élément de plus petite clef.
Dans cette structure de tas, nous allons uniquement insérer des éléments dont le numéro i est un entier entre 0 et $n-1$ où n est un entier connu dès le départ. De plus chaque élément contiendra au plus un seul élément par numéro (donc le tas contient au plus n éléments).

Q 6.1 Proposez une structure de tas correspondant à cette description.

Q 6.2 Donnez les fonctions d'insertion et d'extraction du minimum.

Q 6.3 Donner une fonction permettant de rechercher en $\Theta(1)$ s'il existe un élément de numéro i dans le tas.

Q 6.4 Donner une fonction en $O(log(n))$ permettant de supprimer un élément contenant le numéro i.

Q 6.5 Tester vos fonctions sur des exemples.

Exercice 7 – Algorithme de Dijkstra (TME 9-10)

Dans cet exercice, on désire implémenter l'algorithme de Dijkstra vu en cours et en TD. On utilisera la structure de tas implémentée dans l'exercice précédent pour à la fois déterminer le sommet le plus proche des sommets visités et pour déterminer s'il y a encore des sommets à visiter.
Tester votre algorithme sur la première commodité de quelques réseaux.

Remarque : Il est possible d'améliorer les capacités de votre algorithme en sachant que vous connaissez les deux sommets aux extrémités de la chaîne (remarque hors barème).

Exercice 8 – Détermination des chaînes (TME 10)

Dans cet exercice, nous allons construire des chaînes pour un réseau.

Q 8.1 Dans le cas où l'on considère que chaque câble contient une infinité de fibres optiques, utilisez l'algorithme de Dijkstra pour déterminer la solution optimale. Calculer la longueur totale des chaînes obtenue par votre méthode et comparez-là à la longueur totale des instances de la première phase.

Q 8.2 Dans le cas où l'on considère que chaque câble contient au plus γ câble qui est une donnée des instances, proposez et mettez en oeuvre des méthodes permettant d'obtenir des solutions réalisables. Comme il s'agit ici d'un problème difficile, nous allons considérer qu'il n'est pas nécessaire de déterminer des chaînes pour chaque commodité.

Q 8.3 Ainsi, les méthodes que vous proposez peuvent être évaluées selon deux critères : la longueur totale des chaînes et le pourcentage des chaînes tracées.
Pour la liste des instances données sur le site, vous pouvez proposer (par un mail à vos chargés de TD), des fichiers solutions accompagnés des deux critères (longueur totale et pourcentage de chaînes tracées). Les meilleures méthodes seront classées et se verront attribuer des points bonus.

RENDU FINAL SEMAINE 11 : Vous devez rendre l'intégralité du travail (y compris ce qui a été rendu en semaine 8) en semaine 11. Il s'agit là du travail complet comportant les fichiers sources, les fichiers résultats et le rapport complet (y compris les statistiques). Veuillez vous reporter au document concernant les rendus sur le site ou sur votre brochure de TD.
A l'occasion de la 11ème séance de TME, vous présenterez votre travail à votre chargé de TD lors d'une mini-soutenance de quelques minutes. **Remarque :** En attendant votre tour de passage, vous pouvez mettre en oeuvre les exercices du TD11.