

PROJET 2021: Réorganisation d'un réseau de fibres optiques

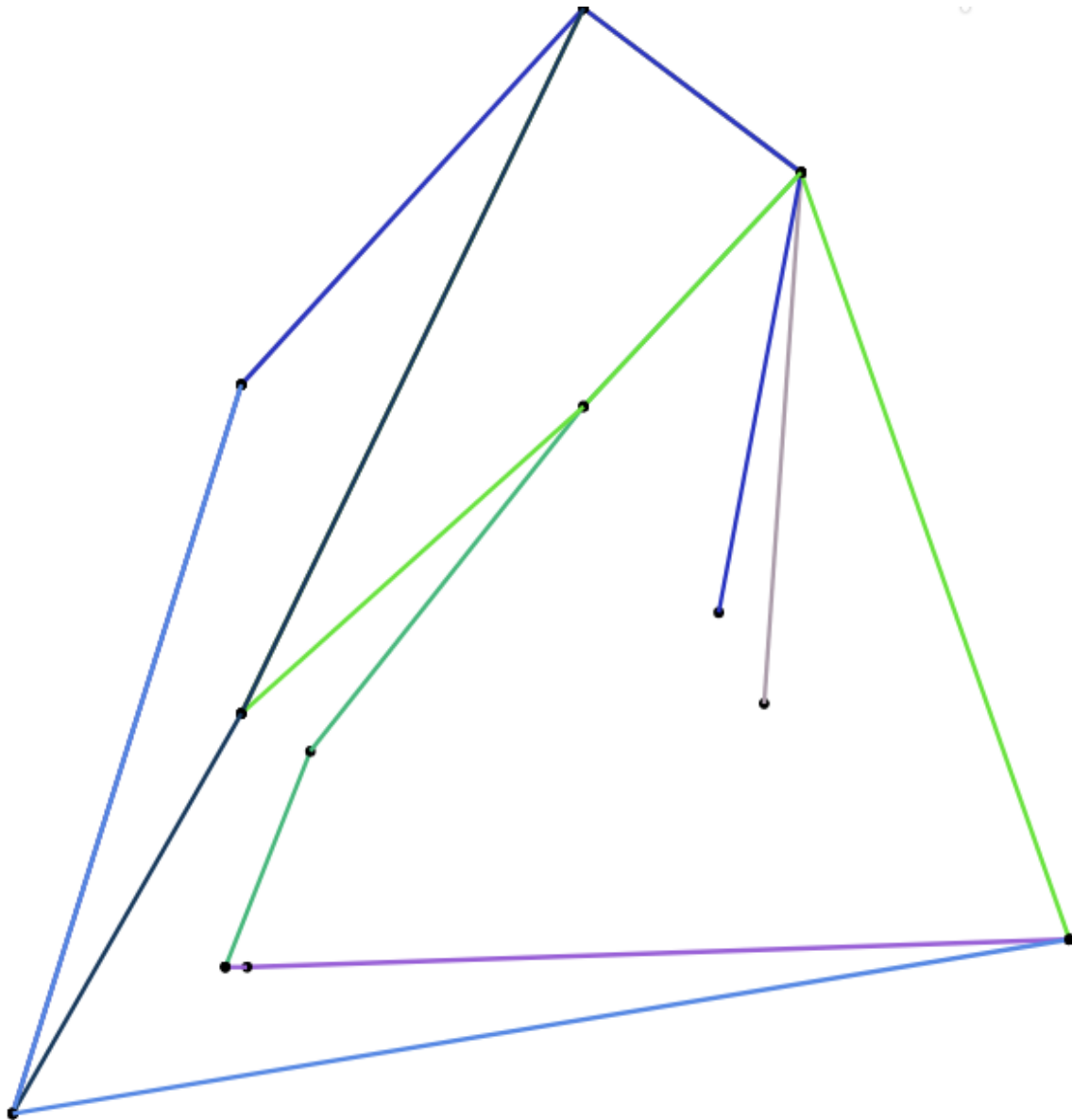
Lecture, Stockage et affichage des données

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

Une deuxième partie du travail va consister à réorganiser les attributions de fibres de chacun des opérateurs

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.



Exercice 1 : Manipulation d'une instance « Liste de Chaînes »

Fichiers : Chaine.c Chaine.h ChaineMain.c

Q 1.1

Création de la fonction `Chaines* lectureChaine(FILE *f)` dans le fichier `Chaine.c`.

La fonction `lectureChaine(FILE *f)` renvoie une instance de type `Chaines` à partir d'un fichier, on utilise les fonctions `fgets` et `sscanf` afin de récupérer la valeur de `Nbchain` et `Gamma`.

Q 1.2

Création de la fonction `void ecrireChaine(Chaines *c, FILE *f)` dans le fichier `Chaine.c`. On utilise la fonction `fprintf` permettant d'écrire dans un fichier.

Création du fichier `ChaineMain.c`.

On implémente une fonction `main` de tel sorte que l'on utilise la ligne de commande pour passer le nom du fichier contenant l'instance à savoir « 00014_burma.cha ».

On veille à bien vérifier que lors de l'exécution, le fichier « 00014_burma.cha » est bien en ligne de commande en faisant une vérification sur `argc` (= nombre d'arguments).

Pour tester la fonction `lectureChaines` on utilise le mode « r » car il s'agit seulement ici de lire le fichier et de récupérer les informations.

Pour tester la fonction `ecrireChaines` on utilise le mode « w » car il s'agit ici d'écrire dans le fichier les données provenant de l'instance `Chaines` passée paramètre.

Q 1.3

Création de la fonction `void afficheChainesSVG(Chaines *C, char *nomInstance)` dans le fichier `Chaine.c`.

Il s'agit de reprendre le code donnée dans le fichier « `afficheChaines.txt` » en veillant à bien rajouter la bibliothèque `SVGwriter.h` grâce à la commande `#include "SVGwriter.h"`.

Cette fonction est testée dans le `main`.

Q 1.4

Création de la fonction `double longueurTotale(Chaines *c)` et `double longueurChaine (CellChaine *c)` dans le fichier `Chaine.c`.

`LongueurChaine` : calcule la longueur d'une chaîne.

`LongueurTotale` : calcule la longueur totale des chaînes.

Q 1.5

Création de la fonction `int comptePointsTotal(Chaines *C)` dans le fichier `Chaine.c`.

Cette fonction renvoie le nombre de points d'une Chaînes `C`.

On a testé la fonction `comptePointTotal` (=30 points) et `LongueurTotale`.

Exécution / Compilation du code :

commande pour compiler : `make`

commande pour exécuter : `valgrind ./ChaineMain 00014_burma.cha`

Utilisation de l'outil `valgrind` pour vérifier les fuites mémoire : rien à signaler (45 allocs 45 frees).

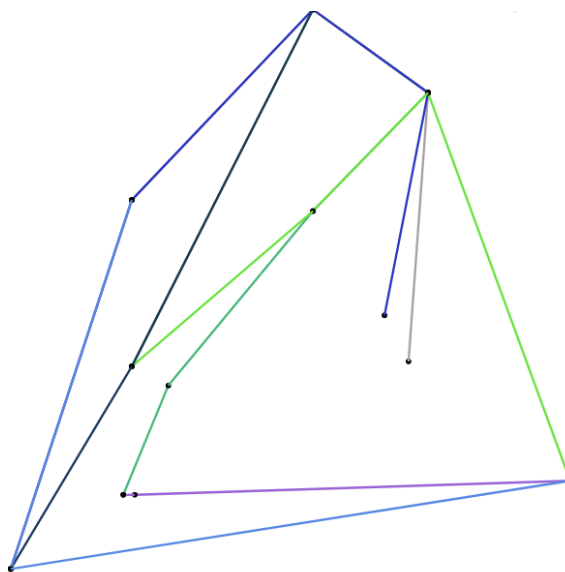
En effet, on a veillé à rajouter une fonction `liberer_Chaines` afin de libérer l'allocation faite lors du test des fonctions de lectures et écritures dans le fichier `ChaineMain.c`.

```

nadia@nadia-VirtualBox:~/Bureau/L2/S2/LUZIN006/NewVersionProjet$ valgrind ./ChaineMain 00014_burma.cha
==6048== Memcheck, a memory error detector
==6048== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==6048== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==6048== Command: ./ChaineMain 00014_burma.cha
==6048==
==6048== HEAP SUMMARY:
==6048==   in use at exit: 0 bytes in 0 blocks
==6048==   total heap usage: 45 allocs, 45 frees, 14,632 bytes allocated
==6048==
==6048== All heap blocks were freed -- no leaks are possible
==6048==
==6048== For lists of detected and suppressed errors, rerun with: -s
==6048== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

fichier : AffichageChaineExo1.html



Commande Makefile (les dépendances des fonctions de l'exercice 1) :

ChaineMain: ChaineMain.o Chaine.o SVGwriter.o

Chaine.o : Chaine.c Chaine.h SVGwriter.o

SVGwriter.o: SVGwriter.c

Exercice 2: Première Méthode : Stockage par liste chaînée

Fichiers : Reseau.c Reseau.h ReconstitueReseau.c

Création de fonctions annexes permettant de créer/libérer chacune des structures créées.

Q 2.1

Création de la fonction Noeud *rechercheCreeNoeudListe(Reseau *R, double x, double y) dans le fichier Reseau.c .

Cette fonction retourne un nœud dans le réseau, si le nœud n'est pas présent elle le crée.

Q 2.2

Création de la fonction Reseau *reconstitueReseauListe(Chaines *C) .

L'objectif de cette fonction est de créer un réseau à partir d'une structure Chaines passée en argument.

Pour pouvoir créer les nœuds de CellNoeud (= R → noeuds) du Reseau, il est nécessaire d'utiliser la fonction rechercheCreeNoeudListe précédemment définie.

Q 2.3

Création du fichier ReconstitueReseau.c .

Ce fichier contient un programme main. L'idée ici, est de créer un menu en s'inspirant du Mini Projet 2. Lors de l'exécution de ce fichier, la ligne de commande du terminal prend 3 arguments en ligne de commande (argc= 3), le fichier à exécuter(argv[0]), le fichier

« 00014_burma.ch »(argv[1]) et enfin un entier(argv[2]), compris entre 1 et 3.

on veille à convertir l'entier passé en argument (car considéré comme une chaîne de caractère par le terminal) en un entier grâce à la fonction atoi().

Exercice 3 : Manipulation du réseau

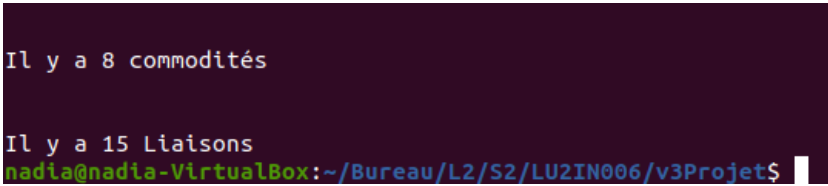
Q 3.1

Création des fonctions nbCommodites(Reseau *R) et nbLiaisons(Reseau *R) .

Pour la fonction nbCommodites, il s'agit de compter le nombre d'éléments de la liste chaînées 'commodités' du Réseau R.

Pour la fonction nbLiaisons , les liaisons sont des câbles qui relient les extrémités de nœuds voisins. Dès lors, il s'agit dans ce cas de parcourir chacun des voisins de la nœud dans le Reseau, pour savoir si on est déjà passé par ce voisin, il suffit de regarder le numéro du Noeud (celui ci doit inférieur a celui du nœud Courant).

On teste ces deux fonctions dans le fichier ReconstitueReseau.c :



```
Il y a 8 commodités
```

```
Il y a 15 Liaisons
```

```
nadia@nadia-VirtualBox:~/Bureau/L2/S2/LU2IN006/v3Projet$
```

Cela correspond bien à ce qui est dans le fichier 00014_burma.res .

Q 3.2

Création de la fonction `ecrireReseau(Reseau *R, FILE *f)` .

L'objectif de cette fonction est d'écrire dans un fichier passé en argument, un réseau en récupérant les données du Réseau passé en argument.

Ligne 1 à 4 :

L'écriture des première lignes du réseau se fait directement :

Pour récupérer la valeur `NbNoeuds` et de `Gamma`, on utilise directement la fonction `fscanf` et de même pour `nbLiaisons` et `nbCommodites` : il suffit d'utiliser les fonctions prédéfinies.

Ligne 6 à 17 :

Il s'agit ici de récupérer le numéro ainsi que les coordonnées des nœuds. Pour cela il suffit de parcourir la liste chaînée 'nœuds' du Réseau ($R \rightarrow \text{noeuds} \rightarrow \text{nd} \rightarrow \text{num}/x/y$) .

Ligne 19 à 33 :

Il s'agit ici de récupérer les liaisons/câbles entre chaque nœuds. Pour cela il suffit de récupérer les numéro des nœuds voisins au nœuds courant en parcourant la liste chaînée 'nœuds' de Réseau .

Ligne 19 à 33 :

Il s'agit ici de récupérer les commodités, c'est-à-dire les paires de nœuds devant être reliée. Pour cela il suffit de parcourir la liste chaînée 'commodités' de Réseau et récupérer le numéro associé `Noeud extraA` et `extraB` .

Q 3.3

Récupération de la fonction `afficheReseauSVG`, donnée dans le fichier `affichageReseau.txt`. Cette fonction se trouve dans le fichier `SVGWriter.c`

Test de la fonction `afficheReseauSVG` dans le main.

Résultat dans le fichier `AffichageReseauListe.html` (Aperçu ci-dessous)

Exécution / Compilation du code :

commande pour compiler : `make`

commande pour exécuter : `valgrind ./ReconstitueReseau 00014_burma.cha 1`

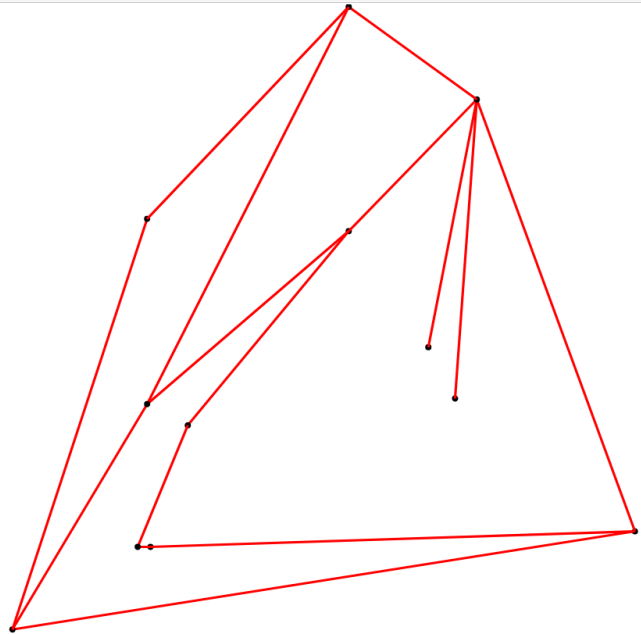
Utilisation de l'outil `valgrind` pour vérifier les fuites mémoire : rien à signaler . En effet, on a veillé à rajouter une fonction `liberer_Reseau` afin de libérer l'allocation .

```
Il y a 8 commodités

Il y a 15 Liaisons
==26178==
==26178== HEAP SUMMARY:
==26178==    in use at exit: 0 bytes in 0 blocks
==26178==   total heap usage: 107 allocs, 107 frees, 12,360 bytes allocated
==26178==
==26178== All heap blocks were freed -- no leaks are possible
==26178==
==26178== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
nadia@nadia-VirtualBox:~/Bureau/L2/S2/LU2IN006/travaildessus$
```

Création des fichiers : « AffichageReseauListe.html » et « fileReseauListe .res»

file:///home/nadia/Bureau/L2/S2/LU2IN006/travaildessus/AffichageReseauListe.html



Commande Makefile (les dépendances des fonctions de l'exercice 2) :

Reseau.o : Reseau.c SVGwriter.o

ReconstitueReseau.o : ReconstitueReseau.c SVGwriter.o (pour l'instant !)

ReconstitueReseau : ReconstitueReseau.o Reseau.o SVGwriter.o Chaine.o ReconstitueReseau.o

Exercice 4 : Deuxième Méthode : Stockage par table de hachage

Fichiers : Hahchage.c Hachage.h

Q 4.1

Définition de la structure TableHachage.

Q 4.2

Création de la fonction cleH(double x ,double y) dans le fichier Reseau.c . La fonction semble appropriée. En effet, en ayant testant la fonction cleH(x, y) pour x et y allant de 1 à 10, les la valeur entière renvoyée par la fonction semble différer pour chaque valeur de x et y. Ainsi, cette fonction semble adaptées car cela nous permet de limiter les valeurs de clé identiques et par conséquent, les collisions.

Q 4.3

Création de la fonction de hachage fonctionHachage(int cle, int m) dans le fichier Reseau.c . A une clé on lui applique un fonction de hachage et cela permet de renvoyer un entier, qui est l'indice du tableau de notre table.

Q 4.4

Création de la fonction *rechercheCreeNoeudHachage(Reseau *R, TableHachage *H, double x, double y) dans le fichier Reseau.c .

Cette fonction permet de rechercher la présence d'un nœud du Réseau R défini par ses coordonnées x et y dans la table de Hachage. Si ce nœud n'est pas présent, il faut le créer et l'ajouter dans le Réseau ainsi que dans la table de hachage H.

L'idée est de directement aller à l'indice de la table correspondant à la fonction de hachage.

Q 4.5

Création de la fonction reconstitueReseauHachage(Chaines *C, int M) dans le fichier Reseau.c, cette fonction permet de créer un Reseau à partir de C et d'une table de hachage de taille M.

Exécution / Compilation du code :

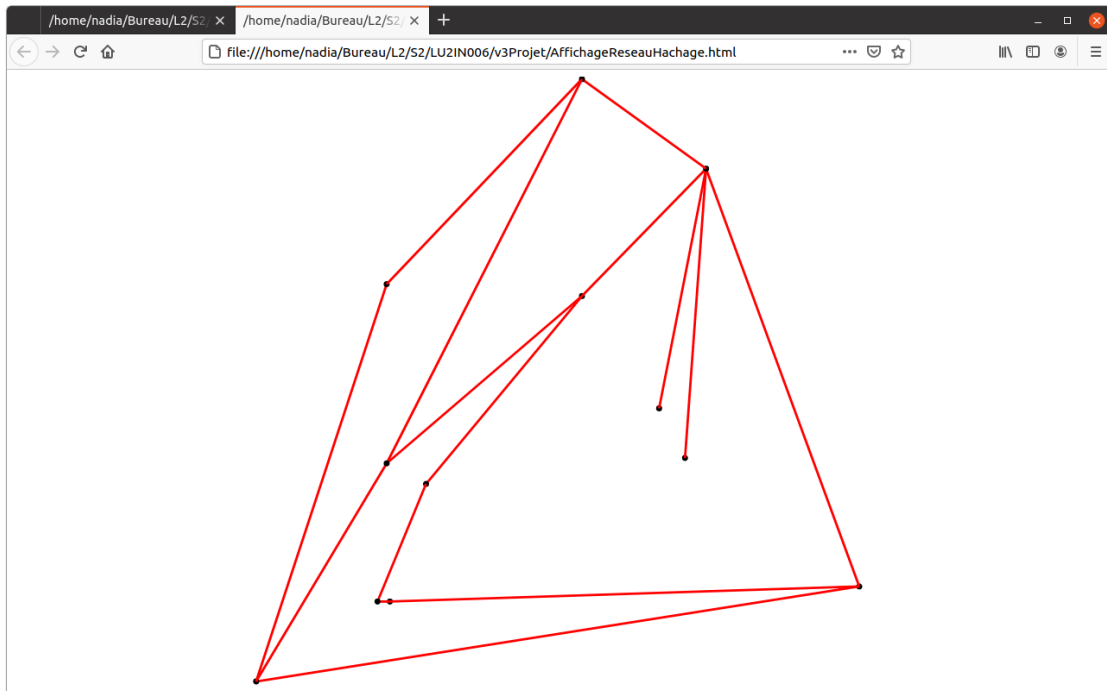
commande pour compiler : make

commande pour exécuter : valgrind ./ReconstitueReseau 00014_burma.cha 2

Utilisation de l'outil valgrind pour vérifier les fuites mémoire : rien à signaler .

```
numero = 7, nbPoints = 3
x=22.389999 y=93.370003
x=20.090000 y=92.540001
x=16.469999 y=96.099998
==28929==
==28929== HEAP SUMMARY:
==28929==    in use at exit: 0 bytes in 0 blocks
==28929== total heap usage: 121 allocs, 121 frees, 12,968 bytes allocated
==28929==
==28929== All heap blocks were freed -- no leaks are possible
==28929==
==28929== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
nadia@nadia-VirtualBox:~/Bureau/L2/S2/LU2IN006/travaildessus$
```

Création du fichier AffichageReseauHachage.html



Commande Makefile (les dépendances des fonctions de l'exercice 4) :

Hachage.o : Hachage.c Hachage.h SVGwriter.o

ReconstitueReseau.o : ReconstitueReseau.c SVGwriter.o Hachage.o (pour l'instant !)

ReconstitueReseau : ReconstitueReseau.o Reseau.o SVGwriter.o Hachage.o

Exercice 5 : Troisième Méthode : Stockage par arbre quaternaire

Q 5.1

Fonction : `chaineCoordMinMax(Chaines* C, double* xmin, double* ymin, double* xmax, double* ymax)` dans fichier `ArbreQuat.c`.

Cette fonction renvoie les coordonnées minimales et maximales de Chaines C.

L'idée ici est de parcourir pour chaque élément de CellChaines, la liste de points.

Pour chaque points, on compare les coordonnées du point courant et du point suivant.

Q 5.2

Fonction : `creerArbreQuat(double xc, double yc, double coteX, double coteY)` dans fichier `ArbreQuat.c`.

Cette fonction crée et renvoie une cellule de l'arbre quaternaire.

Q 5.3

Fonction : `insererNoeudArbre(Noeud *n, ArbreQuat** a, ArbreQuat* parent)` dans fichier `ArbreQuat.c`.

Cette fonction insère un Noeud du Réseau dans l'arbre Quaternaire. On prend en compte les 3 cas : arbre vide/ feuille/cellule interne.

Q 5.4

Fonction `Reseau* rechercheCreeNoeudArbre(Reseau *R, ArbreQuat **a, ArbreQuat *parent, double x, double y)` dans le fichier `ArbreQuat.c` retourne un Noeud du Réseau R correspondant au point x et y dans l'arbre Quaternaire. Si ce nœud existe dans l'arbre, la fonction retourne le nœud si ce n'est pas le cas, la fonction crée un Noeud et l'ajoute dans l'arbre.

Q 5.5

Création de la fonction `reconstitueReseauArbre(Chaines *C)` dans le fichier `ArbreQuat.c`.

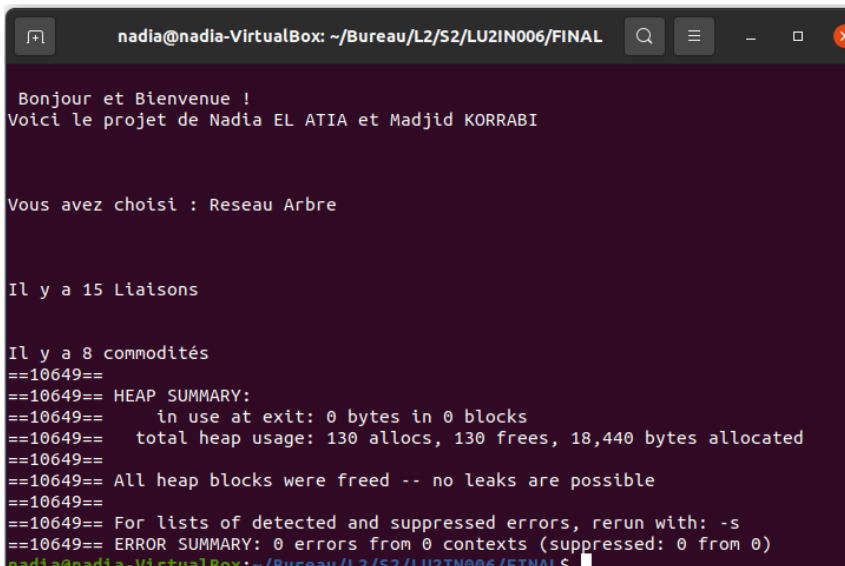
Cette fonction renvoie un Réseau créé à partir de Chaines* C et en utilisant un arbre quaternaire.

Exécution / Compilation du code :

commande pour compiler : `make`

commande pour exécuter : `valgrind ./ReconstitueReseau 00014_burma.cha 3`

Utilisation de l'outil `valgrind` pour vérifier les fuites mémoire : rien à signaler.



```
nadia@nadia-VirtualBox: ~/Bureau/L2/S2/LU2IN006/FINAL
Bonjour et Bienvenue !
Voici le projet de Nadia EL ATIA et Madjid KORRABI

Vous avez choisi : Reseau Arbre

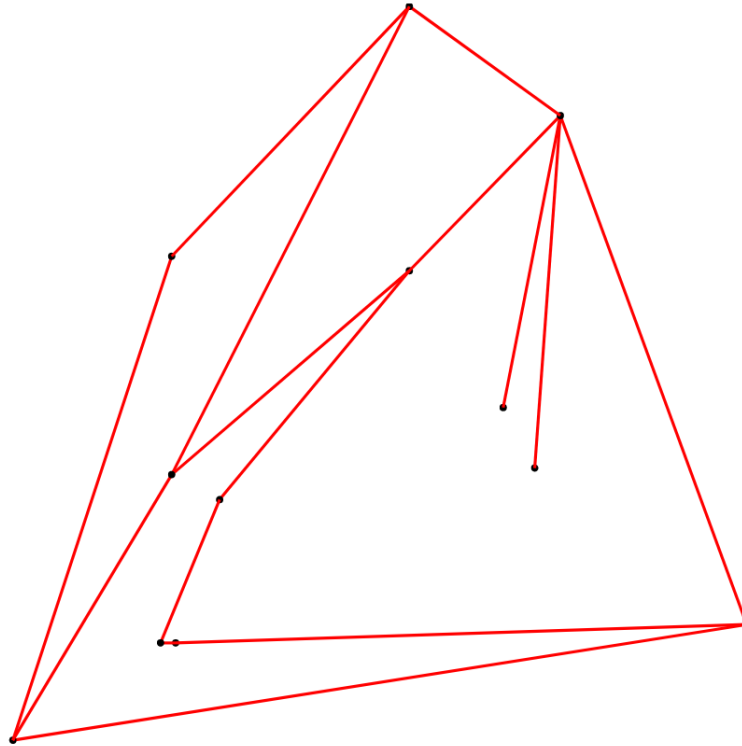
Il y a 15 Liaisons

Il y a 8 commodités
==10649==
==10649==  HEAP SUMMARY:
==10649==    in use at exit: 0 bytes in 0 blocks
==10649==   total heap usage: 130 allocs, 130 frees, 18,440 bytes allocated
==10649==
==10649== All heap blocks were freed -- no leaks are possible
==10649==
==10649== For lists of detected and suppressed errors, rerun with: -s
==10649== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
nadia@nadia-VirtualBox:~/Bureau/L2/S2/LU2IN006/FINAL$
```

Création des fichiers

AffichageReseauArbre.html et fileReseauArbre.cha

file:///home/nadia/Bureau/L2/S2/LU2IN006/FINAL/AffichageReseauArbre.html



Commande Makefile (les dépendances des fonctions de l'exercice 4) :

ArbreQuat.o : ArbreQuat.c ArbreQuat.h SVGwriter.o

ReconstitueReseau.o : ReconstitueReseau.c SVGwriter.o Hachage.o ArbreQuat.o

ReconstitueReseau : ReconstitueReseau.o Reseau.o SVGwriter.o Hachage.o ArbreQuat.o

Exercice 6 : Comparaison des trois structures

Q 6.1

Création de la fonction TempsCalculs.c .

Les temps de calculs sont sauvegardés dans les fichiers :

6.1_ArbreQuat.txt → temps : 0.011018

6.1_Hachage.txt → entre 0.004806 et 0.000172

6.1_Liste.txt → temps : 0.00248

On observe que pour les listes chaînées sont plus rapides que les tables de Hachage. Elles mêmes plus rapide que les Arbres Quaternaire(Efficacité temporelle : Listes > Tables de Hachage > Arbres Quaternaires).

Toutefois, il est à noter que plus on augmente la table de hachage, le temps de calculs diminue, jusqu'à atteindre un niveau quasiment stable. On peut supposer que quand la table de hachage a une plus grande taille, il y a moins de collisions, donc on accède directement à la cellule recherchée en fonction de l'indice.

Q 6.2

Création de la fonction Chaines* generationAleatoire(int nbChaines, int nbPointsChaine, int xmax, int ymax) dan le fichier TempsCalculs.c .

Cette fonction renvoie une Chaines crée contenant nbChaines chaînes et génère nbPoints points entre (0,0) et (5000,5000) pour chacune des chaînes .

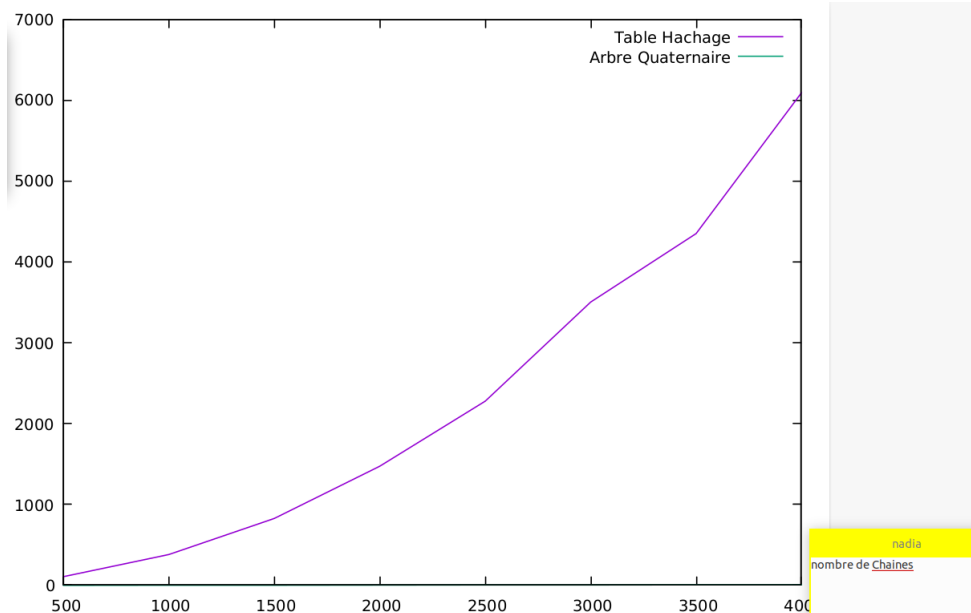
Fonction annexe : generateRandomDouble génère des valeurs entre 0 et 5000 exclus .

Q 6.3 + Q 6.4 Analyse des résultats

/! Étant donné que le nombre de chaînes monte jusqu'à 5500 , le temps d'exécutions est très long (plus de 9h).

Partie 1_Graphe 1 : Création du fichiers donneesGrapheHachageArbre.txt . On trace le graphe à l'aide de l'outil gnuplot (en reprenant la même méthode que dans le mini projet 1).

Titre : Nombre de chaînes en fonction du Temps en millisecondes en utilisant un Arbre et une Table de hachage.



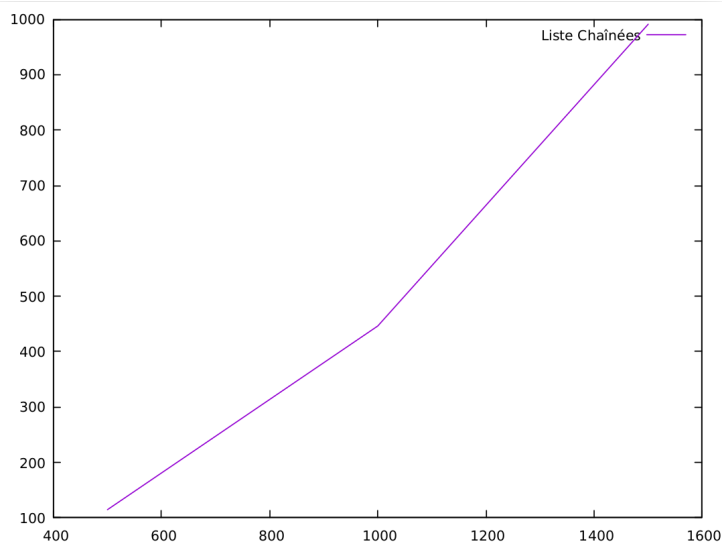
Le temps mis par la structure Arbre Quaternaire est quasiment nulle par rapport au temps mis par la Structure Table de hachage. Ainsi, on peut aisément en déduire que plus le nombre de chaînes augmente plus le temps mis pour reconstituer un Réseau en utilisant une table de hachage est très grand. On peut expliquer cela par le fait que si le nombre de chaînes est plus important il y aura alors, des problèmes de collisions.

Efficacité temporelle (Arbre quaternaire > Table Hachage) .

Il est préférable d'utiliser un Arbre Quaternaire.

Partie 2_Graphe 2

Titre : Nombre de chaînes en fonction du Temps en millisecondes en utilisant une Liste Chaînées.



Malheureusement, nous ne pouvons pas avoir un nombre de chaînes allant jusqu'à 5000, car cela prenait plus de 6h, nous nous sommes donc arrêté à 2000. Toutefois, on peut constater, que le temps de calculs croît de manière exponentielle. En effet, pour 1500 chaînes le temps de calculs est de 1000 millisecondes, c'est bien plus que pour les tables de hachage .

Conclusion :

Si l'on souhaite construire un réseau avec un très grand nombre de chaînes, il est préférable d'utiliser une structure de type Arbre Quaternaire. Dont l'efficacité temporelle est bien meilleure que la Table de Hachage, elle même bien meilleure que la Liste Chaînée .

Exercice 7 : Parcours en largeur

Q 7.1

Création de la fonction Graphe *creerGraphe(Reseau *R) dans le fichier Graphe.c .

On considère que le nombre de Sommets du Graphe (nbsom) est égale au nombre Noeud du Reseau r (NbNoeud). Ici, la création du graphe est plus complexe dans la mesure où nous devons créer le tableau de So mmet (T_som) et le tableau de commodités (T_commod) .

Q 7.2

Création de la fonction plusPetitCheminUV(Graphe * G, int r, int s) dans le fichier Graphe.c . Cette fonction renvoie le plus petit nombre d'arêtes entre 2 sommets (u et v) . Utilisation de la bibliothèque File.

Pour cette question, nous ne sommes pas vraiment sûrs de notre programme, nous préférons tout de même vous faire part de notre travail. De même pour les tests, nous n'avions pas réussi à tester notre fonction.

Q 7.3

Création de la fonction arborescence(Graphe * g, int u , int v) dans le fichier Graphe.c . Cette fonction renvoie une liste d'entiers correspondant à l'arborescence des chemins entre u et v

Pour cette question, nous ne sommes pas vraiment sûrs de notre programme, nous préférons tout de même vous faire part de notre travail. De même pour les tests, nous n'avions pas réussi à tester notre fonction.

Q 7.4

Création de la fonction reorganiseReseau(Reseau *r) dans le fichier Graphe.c . Cette fonction renvoie vrai (1) si le nombre d'arêtes est inférieur à gamma. L'idée est de créer une matrice 2D, nous permettant de compter le nombre de chaînes passant par u et v.