Project: Reorganization of a fiber optic network

We consider in this project the reorganization of an optical fiber network of a city.

The statement of this project is subdivided into two parts:

- Part 1: Network reconstruction.

- Part 2: Network reorganization.

Each part is divided into exercises, which will allow you to gradually design the pro final gram. It is advisable to follow the steps given by these different exercises, because each exercise is the direct application of concepts that have been introduced in class and in tutorial in parallel to the project. It is imperative to work regularly so as not to fall behind and be able to take advantage of the corresponding sessions for each part of the project.

All of the two parts will have to be returned before the defense (date to be specified) which will take place during the last TD / TP session (between 05/04/2021 and 05/10/2021). On Moodle you will find a document summarizing what is expected in your rendering. For students who cannot be present on the day of the defense, you must contact your TD supervisors in order to arrange another date closest to the render week. Finally, we remind you that the note of this project is part of the module note, and is kept in the second session.

Project framework

In this project, we consider an agglomeration whose municipal services wish to improve the fiber optic network of its citizens. A network a set of cables, each containing a set of optical fibers and connecting clients.

The first part of the project consists in reconstructing the network plan of the agglomeration. Indeed, several operators currently share the market and each have a few network fibers. As the network has grown steadily, there is no complete network plan to date. On the other hand, each operator knows which sections of fiber optic they use in the network. Starting from the hypothesis that there is at least one fiber used per cable, it is thus possible to reconstitute the network in its entirety.

A second part of the work will consist in reorganizing the fiber allocations of each of the operators. Indeed, since the distribution of fibers has never been questioned, some cables are under-exploited while others are over-exploited. Each operator has a list of pairs of customers that he has connected to each other by a chain of fiber optic chunks, depending on availability fibers. Some strings are therefore very long. These problems of over-exploitation and lengths excessive can be solved, or at least improved, by reorganizing the network and allocating to operators of chains that are shorter and better distributed in the network: this will be the subject of second part of the project.

The objective of this project is to propose to the agglomeration the best possible methods for carrying out these two parts, and therefore we are going to test several algorithms.

Modeling and notations

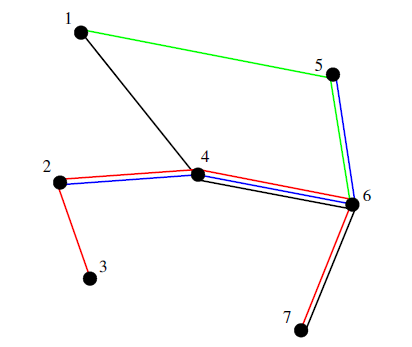
A network cable is a sheath (or sheath) containing exactly > 0 optical fibers. The cables connect two points on the plane. In the network, there are two types of points:

- A client is a point that represents a client, a client company or even a technical room of the operator.

A hub is a point in the network where several cables meet.

A point can be a client, a hub, or both. The fiber optic sections of two cables which arrive at the same concentrator can then be connected at this point. The sections of optical fibers thus connected end to end then form chains in the network. A chain connects always two points of the plane: we call this pair of points a convenience (they are the extremities of chain). There are several operators in the metropolitan area and each operator has several optical fiber chains.

An example Figure 1 represents an instance of our problem.



**Figure 1 - An example of a network.**

It describes a network made up of 7 points and 4 strings each represented by a color: a chain (1, 4, 6, 7) linking convenience (1, 7), a chain (2, 4, 6, 5) linking convenience (2, 5), a chain (2, 4, 6, 5) linking convenience (2, 5) chain (3, 2, 4, 6, 7) connecting convenience (3, 7) and chain (1, 5, 6) connecting convenience (1, 6). We may notice that:

- Points 1, 3 and 7 are only customers because they are at the end of a chain without being a point inside a chain.

- Point 4 is only a concentrator because it is always an interior point of a chain.

- The other points are both clients and concentrators.

If we no longer look at the list of channels, but at the network as a whole, we can notice that there is only 7 cables in this network (these are the edges of the graph). For example, in the cable (1, 4), only one fiber is used. In the cable (4, 6), three fibers are used. This drawing does not give the indication of the number of the maximum number of fibers usable per cable, but we can deduce that ≥ 3.

Instances of the problems

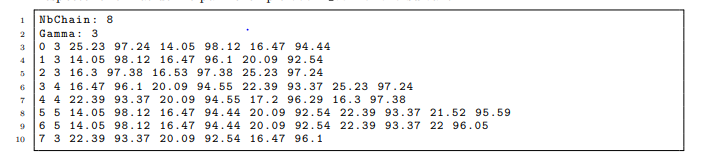
The instances that we are going to handle in this project either come from the TSPLib 1 database, or are-based on the 9th challenge DIMACS 2 database. These two databases contain network instances frequently used to test the efficiency of algorithms concerning network problems. The most correspond to cities or countries (Burma = Burma, NY = New-York, d = Germany, att = USA, etc.), others come from non-geographic networks (electronic networks, etc.). These instances have been adapted to be usable for this project. On the site, you will find them under the form of text files classified according to their number of points. It is requested in the project to use these instances to test your algorithms.

Reading, storing and displaying data

Exercise 1 - Handling a “List of Strings” instance (TME5)

In this first exercise, we will build a library of instance manipulations: read and file writing, graphical display of networks, calculating the total length of strings, and calculating the number of points.

An instance of “List of Strings” is simply given by a number of strings and by the list of chains. Each chain is a list of points on the plane. Each point is identified by its coordinates (x-coordinate and y-coordinate). Each instance is given by a text file with the extension .cha which respects the format given by the following example 00014 Burma .cha:



The first two lines give the number of strings and the maximum number of optical fibers by cable. The different chains of the network are then given. Each string line has in the order, the number of the chain, and the number of points in the chain and the list of points. Each point

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

2. http: //www.dis.uniroma1.it/challenge9/

Is given by its coordinates (abscissa and ordinate). Each chain can therefore be seen as a string list of points. We will use the following data structure in the (supplied) file Chaine.h.

|  |
| --- |
| # ifndef \_\_CHAIN\_H\_\_  #define \_\_CHAINE\_H\_\_  # include <stdio .h>  / \* Chain list of points \* /  typedef struct cellPoint {  double x, y; / \* Point coordinates \* /  struct cellPoint \* next; / \* Next cell in the list \* /  } CellPoint;  / \* Cell of a list (string) of strings \* /  typedef struct cellChaine {  int number; / \* String number \* /  CellPoint \* points; / \* List of chain points \* /  struct cellChaine \* next; / \* Next cell in the list \* /  } CellChaine;  / \* All the strings \* /  typedef struct {  int gamma; / \* Maximum number of fibers per cable \* /  int nbStrings; / \* Number of strings \* /  CellChaine \* strings; / \* The string list of strings \* /  } Channels;  Strings \* readStrings (FILE \* f);  void writeStrings (Strings \* C, FILE \* f);  void displaySVGStrings (Strings \* C, char \* instanceName);  double Totallength (Strings \* C);  int countTotalPoints (Strings \* C);  # endif |

We can notice that:

- The struct cell Point is an element of the list of points and contains the coordinates of a point.

- The struct cellChaine is an element of the string list and contains a string number and the list of points.

-The set of chains is a struct containing the maximum number of fibers per cable, the number of strings and the list of strings.

Q 1.1 In a String.c file, implement a function Strings \* read String (FILE \* f); who Allows to allocate, fill and return an instance of our structure from a file.

Q 1.2 Implement a void function write String (Strings \* C, FILE \* f); who writes in a Files the content of a String respecting the same format as that contained in the file Of origin. Create a main Chaine Main.c allowing to execute the read and write functions That you have just defined (you can use the command line to pass the name of the file Containing the instance)

Remarks:

- The write function allows you to recreate the data file, but the order of the points and strings will be reversed (because of the insertions at the top of the list).

- The purpose of this write function is to test the code of your read function on several instances, before tackling the rest of the project.

Q 1.3 We want to give a graphical representation of the instances. For this we will use the SVG (Scalable Vector Graphics) image format which is increasingly used to write simple graphics and widely used for the internet. Your code will create a file in SVG format for html which will then be read directly by your favorite internet browser. We offer you on moodle a very simple little C library which creates an SVG file with html extension. It's about of an SVGwriter struct which is manipulated by methods allowing to create the file, to add lines and dots and change colors. There is also a random color generation oF segments (we must initialize the random generation to obtain different colors). In your String.c file, add the function void display SVGString (Strings \* C, char \* instance Name); which allows to create the SVG file in html from a String struct. This function is yours given in file ”displaySVG.txt” on moodle. Test this display function in your ChaineMain.c file.

Q 1.4 Implement the functions:

- double lengthString (CellString \* c); which calculates the physical length of a string.

- double Totallength (Strings \* C); which calculates the total physical length of the strings. To calculate the length of a chain, we must sum the distances between the different points which make up the chain. As a reminder, the distance between two points A and B with coordinates (xA, yA) and (xB, yB) is given by d (A, B) = .

Note: with the <math.h> library, remember to add the -lm option when creating your executable.

Q 1.5 Write the int function countTotalPoints (Strings \* C); which gives the total number of occurrences rences of points (points that appear more than once are counted more than once).

Network reconstitution

The goal of this part is to efficiently reconstitute the network from the chains. From the list of channels, this is:

­­-find the list of network nodes (we eliminate point redundancies). Thus, at a given coordinate, there can be only one node.

- identify all the cables that come from a node. This keeps the list of nodes neighbors at a given node.

- retrieve and keep the list of network amenities.

The reconstruction algorithm is very simple. The pseudo-code is as follows:

We use a set of nodes V which is initialized empty: V ← ∅

We go through each chain one by one:

For each point p of the chain:

If p 6∈ V (we test if the point has not already been met before)

We add in V a node corresponding to point p.

We update the list of neighbors of p and those of its neighbors.

We keep the convenience of the chain.

In this part, we are concerned with the optimization of the "p 6∈ V" test. For that, we will study three methods which correspond to three data structures to implement the set V: a list string, a hash table and trees.

Exercise 2 - First method: storage by chained list (TME5-TME6)

In this exercise, we want to implement the network reconstruction algorithm by coding the set network nodes by a linked list. For this we need to define a structure for manipulate a network. A network is presented as a collection of nodes, cables and conveniences.In this structure:

- Each node v will be identified by its coordinates, and we will know the list of pointers to nodes which are connected to v by a cable. When the network is reconstituted, we will attribute to each node a unique integer number that will be chosen incrementally.

- Each cable is given by pointers on its two end nodes.

- Each commodity is a pair of pointers to the network nodes that must be linked by a string.

Thus, to store the data of the network, we will use the following data structure (supplied file), defined in the Reseau.h file:

|  |
| --- |
| # ifndef \_\_NETWORK\_H\_\_  #define \_\_RESEAU\_H\_\_  # include ”String. h ”  typedef struct node Node;  / \* Chain list of nodes (for the list of network nodes AND the lists of  neighbors of each node) \* /  typedef struct cellnode {  Node \* nd; / \* Pointer to the stock node \ 'e \* /  struct cellnode \* next; / \* Next cell in the list \* /  } CellNode;  / \* Network node \* /  node struct {  int num; / \* Node number \* /  double x, y; / \* Coordinates of the node \* /  CellNoeud \* neighbors; / \* List of neighbors of the node \* /  };  / \* Chain list of commodities \* /  typedef struct cellCommodite {  Node \* extrA, \* extrB; / \* Knots at the ends of the convenience \* /  struct cellCommodite \* next; / \* Next cell in the list \* /  } CellCommodite;  / \* A network \* /  typedef struct {  int nbNodes; / \* Number of network nodes \* /  int gamma; / \* Maximum number of fibers per cable \* /  CellNode \* nodes; / \* List of network nodes \* /  CellCommodite \* commodites; / \* List of commodities to link \* /  } Network;  Node \* searchCreeNodeList (Network \* R, double x, double y);  Network \* reconstitutesReseauList (Strings \* C);  void writeReseau (Network \* R, FILE \* f);  int nbLiaisons (Network \* R);  int nbCommodites (Network \* R);  void displaySVGNetwork (Network \* R, char \* instanceName);  # endif |

This structure makes it possible to store a Network as a chained list of Nodes and a chained list of Convenience. Each Node v is given by its number, its coordinates and the chain list of nodes neighbors, that is, the nodes which are linked to node v by a cable. A Convenience is simply given by the two nodes which will be connected by a chain.

Q 2.1 Create a Node \* searchCreeNodeList function (Network \* R, double x, double y); which returns a Node of the network R corresponding to the point (x, y) in the list chained nodes of R.

Note that if this point exists in nodes, the function returns a node existing in nodes and that,otherwise, the function creates a node and adds it to the list of nodes in R's network. number of a new node is simply chosen by taking the number nbNodes + 1 (just before update to the value nbNodes).

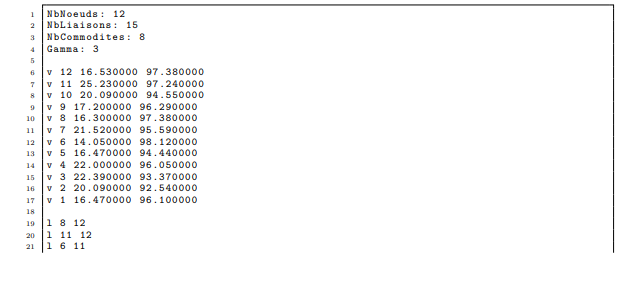
Q 2.2 Implement a function Network \* reconstituteReseauList (Strings \* C); who rebuilds the network R from the list of chains C as indicated in the pseudo-code given at the beginning of this part. Directly use the chain node list of the network to perform type tests

”P 6∈ V”, by exploiting the previous question function.

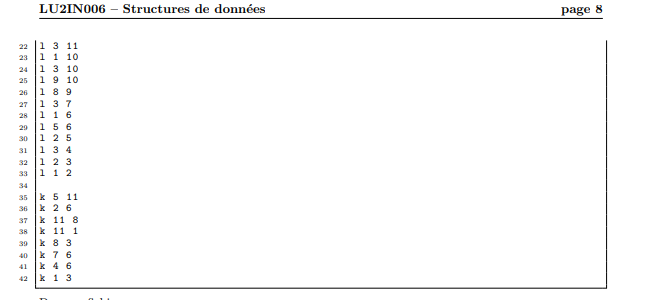
Q 2.3 Create a main ReconstitueReseau.c program that uses the command line to take a .cha file as a parameter and an integer indicating which method one wishes to use (list, hash table, or tree)

Exercise 3 - Handling a network (TME6)

We now want to build methods to handle and display a Network struct. For it, we will store a Network on disk using the format illustrated by the 00014 burma instance which is given by the following file 00014 burma.res (obtained by the previous exercise).



Data structures



In this file:

- The first four lines give the number of network nodes, the number of cables (linksounds), the number of amenities, and the maximum number of optical fibers per cable.

- Then, the lines starting with ”v” give the network nodes. The nodes are marked by their number and their two coordinates.

- Lines starting with a ”l” contain a link (a cable) given by the numbers from both ends.

- Lines starting with a "k" correspond to a convenience, that is to say a pair of numbers of nodes which must be connected by a chain.

Q 3.1 To write such a file, start by implementing the nbCommodites (Network \* R) functions; and int nbLiaisons (Network \* R); which count the number of facilities and links in the R network.

Q 3.2 Implement a void writeReseau function (Network \* R, FILE \* f); who writes in a fi shit the contents of a Network respecting the same format of the 00014 burma.res file.

Q 3.3 In the displayReseau.txt file, retrieve the function void displaysReseauSVG (Network \* R, char \* instanceName); which allows you to create an SVG file in html to visualize a network. Test your code on several instances by comparing it with the display of strings to validate (in part) your duties.

Exercise 4 - Second method: hash table storage (TME7)

For this exercise, we will use a hash table with collisions management by chaining. The hash table will therefore contain an array of pointers to a list of nodes. During the course from the list of points constituting a chain, the hash table will allow us to determine quickly if a node has already been stored in the network

Warning: if it has not yet been stored, it will need to be stored both in the hash table and in the network.

It is hoped that the use of a hash table will speed up the reconstruction of the network.

Q 4.1 Give a TableHash structure that allows you to implement a hash table with management of collisions by chaining. Define it in a file named Hash.h.

Q 4.2 The value to be stored is given by the coordinates (x, y) of a point. You can use the key function f (x, y) = y + (x + y) (x + y + 1) / 2. Test the keys generated for points (x, y) with x integer ranging from 1 to 10 and y integer ranging from 1 to 10. Does the key function seem appropriate to you?

Q 4.3 For a hash table of M cells, we will use the hash function



2 for any key k. Later, you will experimentally test several values

of M in order to determine the most appropriate value (cf. Exercise 6). It is also possible to use

very different hash functions: you can offer other functions.sssss

Q 4.4 Implement a Node function \* searchCreeNodeHash (Network \* R, TableHash \*

H, double x, double y); which returns a Node of the network R corresponding to the point (x, y) in the hash table H. Note that if this point exists in H, the function returns a point existing inH and that, in the opposite case, the function creates a node and adds it in H as well as in the list of nodes of the R.

Q 4.5 Implement a Network function \* reconstituteHashNetwork (Strings \* C, int M); who reconstructs the network R from the list of chains C and using a hash table H of size M

Exercise 5 - Third method: storage by quaternary tree (TME8)

For this exercise, a quaternary tree will be used for the reconstruction of the network. As for the hash table, the quaternary tree will allow us to quickly determine if a node already has Been stored in the network.

A quaternary tree is a tree where each node has four children. In a space with two di- mensions, a quaternary tree represents a rectangular cell. Its center makes it possible to identify son, which represent the northwest, northeast, southeast, and southwest parts of space in relation to this center (see Figure 2).

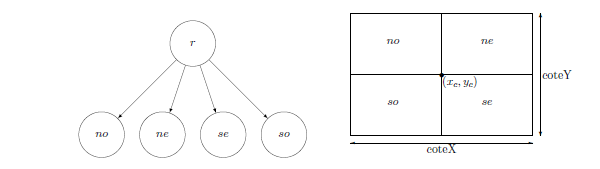


Figure 2 - Representation of a quaternary tree in a two-dimensional space.

The nodes of our network can be stored at the level of the leaves of the quaternary tree. Indeed ,we can associate a data with each leaf of the tree, identified by its x and y coordinates. In within the framework of this project, the data associated with each leaf will therefore be a pointer to a Node of the network. For example, for a network with four nodes n1, n2, n3 and n4, we could have the tree and the representation in Figure 3.

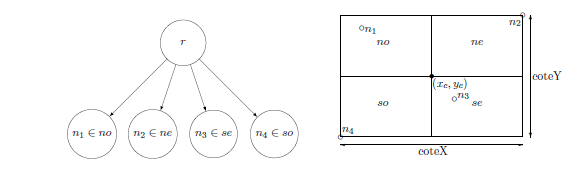


Figure 3 - Representation of a quaternary tree containing four nodes.

We will use the following data structure (attached file):

|  |
| --- |
| # ifndef \_\_ARBRE\_QUAT\_H\_\_  #define \_\_ARBRE\_QUAT\_H\_\_  / \* Quaternary tree containing the nodes of the network \* /  typedef struct treeQuat {  double xc, yc; / \* Coordinates of the center of the cell \* /  double dimension X; / \* Cell length \* /  double dimensionY; / \* Cell height \* /  Knot \* knot; / \* Pointer to the network node \* /  struct treeQuat \* so; / \* Southwest subtree, for x <xc and y <yc \* /  struct treeQuat \* se; / \* Southeast subtree, for x> = xc and y <yc \* /  structQuat tree \* no; / \* Northwest subtree, for x <xc and y> = yc \* /  struct treeQuat \* ne; / \* Northeast subtree, for x> = xc and y> = yc \* /  } TreeQuat;  # endif |

To be able to create the root node of the quaternary tree which will contain all the nodes of the network, it is necessary to identify the length (dimensionX) and height (dimensionY) of the cell. For this, we can use the minimum and maximum coordinates of the points to be stored in the structure.

Q 5.1 Implement a void function stringCoordMinMax (Strings \* C, double \* xmin, double \* ymin, double \* xmax, double \* ymax); which determines the minimum and maximum coordinates of the points constituting the various chains of the network.

Q 5.2 Write the function TreeQuat \* createTreeQuat (double xc, double yc, double dimensionX, double dimensionY); which allows to create a quaternary tree cell, of center (xc, yc), of length dimensionX and height dimensionY. This function will initialize the network node, the north-west, north east, southwest and southeast to NULL.

Q 5.3 Implement a void function insertTreeNode (Node \* n, TreeQuat \*\* a, TreeQuat \*parent); allowing to insert a Node of the network in a quaternary tree. The parent argument is used in this function to determine the dimensions of the new cell if it is to be created (and stored at the address indicated by a). Indeed, when inserting a node, three cases are consider (empty tree, leaf and internal cell):

- Empty tree: In case the tree is empty ((\* a == NULL)), you must create a tree using the createQuatTree function. The coordinates of the center of the new tree, as well as its length and height, will be identified by the position of the point to be inserted (we must determine if we inserted in the northwest, northeast, southeast or southwest part of the parent tree) and thanks to the length and height of the parent tree.

- Leaf : If the node is to be inserted at the level of a leaf of the tree, that is to say if a node has already been stored in the cell corresponding to the tree (((\* a) -> node! = NULL)), you need to times insert node n but also the old leaf node ((\* a) -> node), using the insertTreeNode function recursively. For example, if we take the case of

Figure 3 and we want to insert a node n5 whose coordinates appear in the tree northwest of the root, we see that this tree already contains node n1. So it is necessary divide the northwest cell into four so that n1 and n5 are in two sheets different from the tree. The result we get is shown in Figure 4.

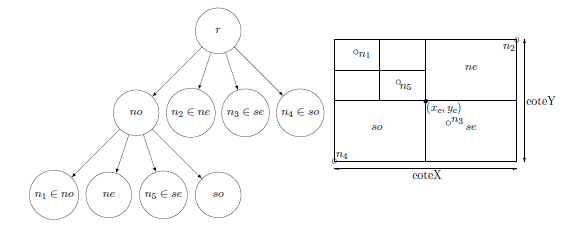


Figure 4 - Representation of a quaternary tree containing five network nodes. –

Internal cell: In the case where we are on an internal cell of the tree ((\* a! = NULL)&& ((\* a) -> node == NULL)), we must recursively determine in which cell of the tree place the network node (we use the coordinates (x, y) of the network node that we compare to the coordinates of the center of the tree).

Q 5.4 Write the function Node \* searchCreeNodeTree (Network \* R, TreeQuat \*\* a, TreeQuat \*parent, double x, double y); which returns a Node of the network R corresponding to the point of co-ordinates (x, y) in the quaternary tree. Note that if this node exists in the quaternary tree, the function returns an existing node in the tree and that, otherwise, the function creates a node and adds it to the tree as well as to the list of nodes in R's network. We recall that three cases are to be distinguished (empty tree, leaf and internal cell):

- Empty tree: If the tree is empty ((\* a == NULL)), the corresponding node must be created dant to point, then insert it into the network and into the tree (with the function insertTreeNode).

- Leaf: If we are on a leaf of the tree ((a-> node! = NULL)), we see if the node that the search matches that of the leaf. If so, we return the knot. If it's not in this case, we must create the node corresponding to the point, then insert it in the network and in the tree (with the insertTreeNode function).

- Internal cell: In the event that we find an internal cell of the tree ((\* a! = NULL) && ((\* a) -> node == NULL)), we must recursively determine in which cell ofthe tree looks for the node of the network (thanks to the coordinates (x, y)).

Q 5.5 Implement a Network \* function reconstituteTreeNetwork (Strings \* C); who rebuilds the network R from the list of chains C and using the quaternary tree.

Exercise 6 - Comparison of the three structures (TME9)

In this exercise, you will compare the computation times obtained with the three data structures

used to test the existence of a node in the network: the linked list, the hash table and

the quaternary tree.

Q 6.1 Create a main program or script that automatically executes the three functions of reconstruction and which only calculates their computation time. Save the times in a file calculation obtained for each of the three functions, for the different instances provided. What do you observe you?

Note: For the hash table, it is also necessary to vary the size of the table.

Q 6.2 We will now compare the computation times for new randomly generated data.Create a function Strings \* generationAleatory (int nbStrings, int nbPointsString, int xmax, int ymax) which allows you to create chains of points. This function takes as parameter the number of chains to be created, the number of points per chain and the maximum coordinates of the points. For each of the strings, this function must randomly create nbPointsString points located between the points (0,0) and (xmax, ymax).

Q 6.3 We now want to construct graphs taking the total number of points on the abscissa of the strings and on the ordinate the computation time according to the data structure used. We can by example use the following data: nbPointsString = 100, xmax = 5000 and ymax = 5000, in varying the number of chains from 500 to 5000 in steps of 500. Make two different graphs:

- A graph giving the computation times with the chained list.

- A graph including the results obtained with the hash table and the quaternary tree.

Note: For the hash table, it will also be necessary to vary the size of the table.

Q 6.4 Analyze your results.

Warning : The project is not finished. One more exercise is missing,do during the TME10 session. The rest of the topic will be uploaded before the next TP session! !