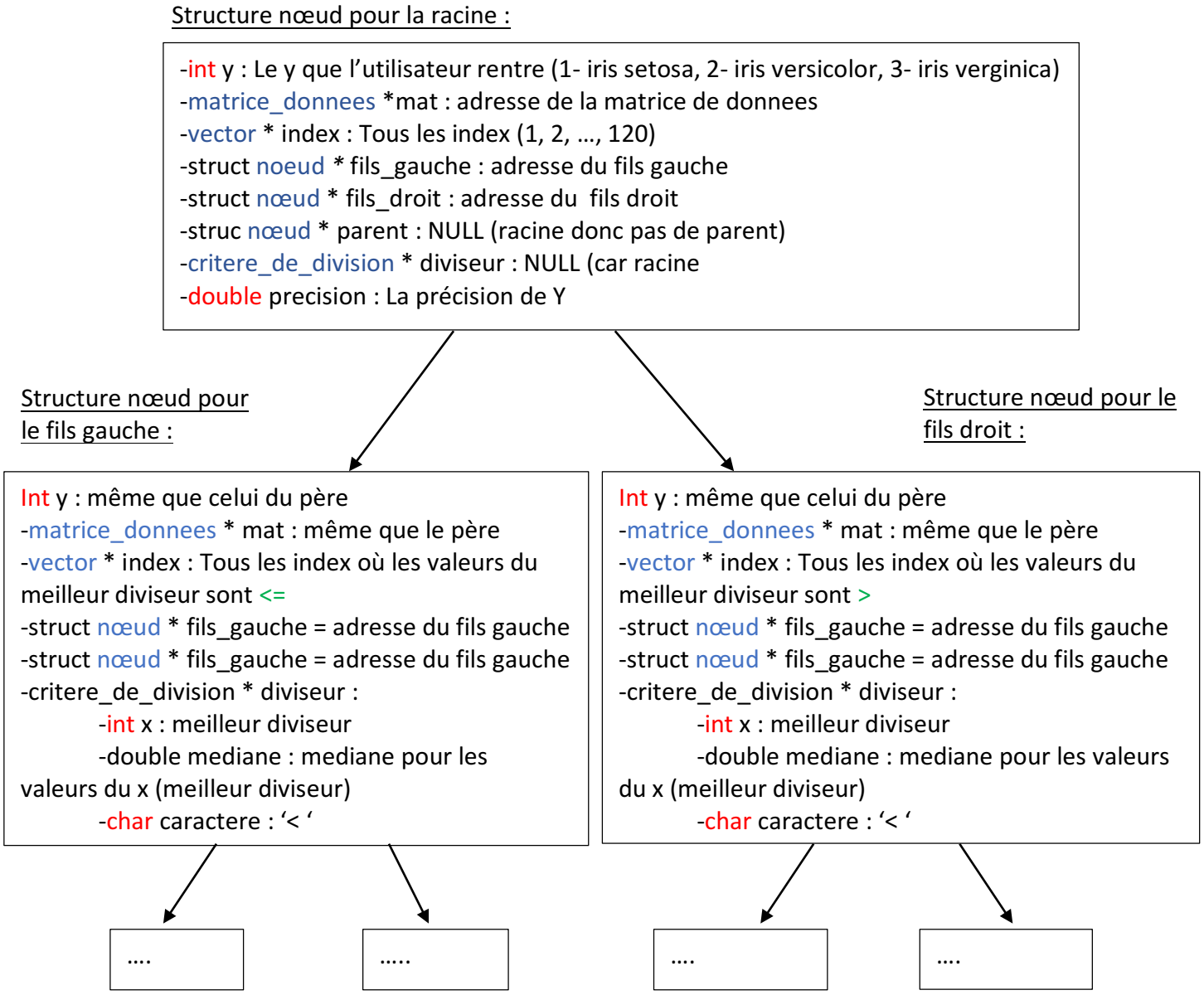
Henri de Montalembert
TD-G

Projet Structure de Données et Algorithmes

A- Définition des types

Avant de commencer la partie code, j'ai commencé par réfléchir à comment j'allais organiser mon code et aux structures que j'allais créer. Une lecture et relecture du sujet m'ont permis de les choisir intelligemment.
J'ai réalisé un petit schéma afin que vous puissiez comprendre au mieux mon raisonnement ainsi que mes classes.
Structure **nœud :** Cette structure a pour but de stocker dans chaque nœud toutes les informations que le sujet demande.
Structure **Critere_de_division :** Cette structure permet de stocker tous les éléments nécessaires pour effectuer la division entre fils droit et fils gauche.



Henri de Montalembert
TD-G

```
typedef struct _precision
{
    double p_inf;
    double p_sup;

}precision;
```

J'ai créé une structure précision afin de pouvoir stocker, à chaque fois que le programme recherche le meilleur diviseur, toutes les précisions inférieures et supérieures des Xi

J'ai rajouté à mon projet les fichiers **vector.h** et **vector.c** pour faciliter la gestion les tableaux d'index et de pouvoir connaître la taille logique et physique à tout moment. J'ai créé dans vector.c une fonction bool **est_dans_le_vecteur** qui prend en paramètre un vecteur et un int et renvoie true si ce int est dans le vecteur. Cette fonction m'a permis de parcourir la matrice seulement aux lignes des individus dont l'index est stocké dans le vecteur d'index du nœud.

B- Construction automatique de l'arbre de décision :

Pour cela, j'ai créé une fonction dont la propriété est la suivant :

```
void creat_arbre(noeud * arbre , double precision_inf , double precision_sup ,double nb_indiv_minim, double hauteur_max)
```

Elle prend en paramètre un nœud, qui correspond à la racine que je crée au préalable (voir schéma ci-dessus), auquel j'affecte le y que l'utilisateur a rentré. Puis la fonction prend en paramètre toutes les conditions à respecter pour que le nœud puisse créer des nouveaux nœuds, c'est l'utilisateurs qui les rentre : la précision minimum, maximum, le nombre d'individu maximum et la hauteur maximum.

La fonction commence par vérifier si toutes les conditions sont respectées pour savoir s'il est possible de diviser le nœud. Ensuite, elle appelle une fonction **chercher_diviseur** qui renvoie le meilleur **critere_de_division** possible par rapport aux individus que le nœud contient dans son vecteur d'index.

La fonction **chercher_diviseur** commence tout a d'abord par trier tous les Xi pour ensuite pouvoir calculer toutes les médianes correspondantes. Après cela, je calcule les précisions inférieures, par rapport à Y, des individus dont les Xi sont inférieurs à la médiane. Je fais de même pour les précisions maximales. Toutes ses précisions sont stockées dans un tableau de **précison** composé d'une case par Xi. Après cela, je cherche la précision maximale de mon tableau et je renvoie sa position + 1. Si plusieurs précisions sont égales, j'ai décidé de récupérer le premier Xi. Suite à ça je crée le nouveau nœud « gauche » à qui j'affecte le critère division ainsi que le bon nouveau vecteur d'index.
Je répète ces deux appelles de fonction pour créer le fils droit.

Pour créer les vecteurs d'index pour le fils droit et gauche, j'utilise la méthode **add** qui est dans le fichier **vector.h**. Le vecteur de la racine est tout simplement composé de tous les index de la matrice de données. En ce qui concerne les autres nœuds, j'ai parcouru toute la colonne du meilleur diviseur (Xi) **dont l'index est contenu dans le vecteur d'index du père** grâce à la fonction est_dans_le_vecteur et je rajoute aux vecteurs d'index « gauche » les index dont le Xi est inférieur ou égale à la médiane contenue dans le critère de division. Pour le vecteur d'index « droit », je lui affecte tous les autres index.

Après avoir créé le nœud droit et le nœud gauche, j'affecte au père les deux nœuds et aux deux fils le père afin de les relier et de pouvoir parcourir l'arbre par la suite. Et c'est qu'après tout cela, que j'appelle en récursif la méthode deux fois : une fois avec le fils gauche comme nœud et une fois avec le fils droit. Ces appelles permettent de créer l'arbre en recréant des fils droits et gauches jusqu'à que les conditions ne soient plus respectées.

C- **Fonctionnalité du menu**

1)Afficher hauteur de l'arbre :

Pour pourvoir récupérer la hauteur de l'arbre, je me suis aidé d'une fonction que nous avons réalisé en TD. Le but est de trouver le chemin le plus long qui part de la racine jusqu'à une feuille. Pour cela, j'ai créé une fonction récursif **height** qui prend en paramètre un nœud et qui renvoie la hauteur en int. Tout a d'abord la fonction vérifie la condition d'arrêt : si l'arbre est NULL elle renvoie 0. Sinon, la fonction renvoie 1 + le maximum entre la taille du fils_gauche et du fils_droit. Puis cette fonctionnalité l'affiche sur la console.

2)Afficher la largeur de l'arbre :

Pour un arbre binaire, la largeur est tout simplement sont nombres de feuilles. C'est pourquoi, j'ai réalisé une fonction récursive qui prend en paramètre un nœud et qui retourne le nombre de feuilles nommé **nombre_feuilles** que nous avions déjà réalisé en TD. Cette méthode renvoie 0 si le nœud est NULL et renvoie 1 si le nœud est une feuille. Et elle finit par retourner le nombre de feuilles du fils gauche + le nombre de feuilles du fils droit. Et termine par l'afficher sur la console.

3)Afficher l'arbre en arborescence

Pour cela, j'ai récupéré le code que Mr. Rimbault nous aviez partagé sur moodle pour le TD7 en modifiant seulement ce qu'on affiche à chaque nœud. En effet, j'affiche sur la console pour chaque nœud : la précision, le nombre d'individus ainsi que le « comment » (Ex : X1<=4.4) en vérifiant si c'est un fils gauche ou droit pour afficher le bon caractère de division. J'ai fait attention de ne pas afficher le « comment » pour la racine et d'afficher « FEUILLE » au-dessus des feuilles, c'est à dire les nœuds dont le fils_gauche et fils_droit sont NULL. Afin d'améliorer l'affichage, j'ai créé un pointeur sur un compteur qui me permet d'afficher « FEUILLE 1 » pour la première, « FEUILLE 2 » pour la deuxième et ainsi de suite.

4)Afficher feuilles :

Le but de cette fonctionnalité est d'afficher toutes les informations clés de toutes les feuilles de l'arbre. Pour cela, j'ai réalisé une fonction récursive qui parcourt tout l'arbre tant que le nœud n'est pas une feuille. Dès que le nœud est une feuille, j'affiche premièrement « FEUILLE N » avec N le numéro de la feuille que j'ai récupéré grâce à un pointeur de compteur, sa précision, son nombre d'individu ainsi que son chemin. Pour ce dernier, j'ai créé une fonction **affichage_chemin** qui prend en paramètre la feuille et qui parcourt l'arbre jusqu'à la racine grâce à l'accès au père dans chaque nœud. Cette fonction s'appelle tant que le nœud en

paramètre n'est pas la racine puis affiche le «comment». L'affichage se fait après la récursivité, ce qui permet d'afficher le chemin dans le bon sens.

5)Prédire :

Cette fonctionnalité a pour but de prédire avec les caractéristiques d'un individu s'il correspond ou non à un type précis. Elle demande tout d'abord à l'utilisateur de rentrer les variable (X1, X2, X3, X4) avec lesquelles il souhaite réaliser sa prédiction. J'ai créé une fonction **predire** qui prend en paramètre toutes les variables ainsi qu'un nœud. Cette fonction va parcourir l'arbre tout en comparant la médiane du Xi diviseur avec la variable correspondant, rentrée par l'utilisateur tant que le nœud n'est pas une feuille. Dès que la condition d'arrêt est atteinte, la fonction renvoie la précision de la feuille. J'appelle cette fonction avec les trois arbres possibles (Y=1, Y=2, Y=3) et je stock les précisions retournées dans trois variables. Ensuite, j'affiche les trois précisions en précisant le type. Et je termine la fonctionnalité en comparant les trois précisions pour afficher sur la console le type qui correspond le mieux aux trois variables rentrées en paramètre, c'est à dire le type où la précision est la plus forte.

Comme demandé dans le sujet, j'ai testé ma fonction prédire avec tous les individus ci-dessous. Au lieu de prédire seulement avec des *iris versicolor,* j'ai préféré testé avec les trois *iris* pour pourvoir avoir une prédiction plus précise.

Pour tester ces individus, j'ai rentré les conditions suivantes :

-Seuil_max = 90%
-Seuil_min = 10 %
-Nb individu max =12
-Hauteur_max = 8

	Y=1 (%)	Y=2 (%)	Y=3 (%)	X1	X2	X3	X4
1	0	3.44828	100	7.7	3.0	6.1	2.3
2	0	3.44828	100	6.2	2.8	4.8	1.8
3	0	100	0	5.5	2.5	4.0	1.3
4	0	0	100	6.7	3.3	5.7	2.5
5	0	94.44	6.6667	6.0	2.2	5.0	1.5
6	0	3.448	6.6667	6.0	2.7	5.1	1.6
7	0	100	0	5.7	2.6	3.5	1.0
8	0	100	0	5.8	2.6	4.0	1.2
9	100	0	0	5.1	3.4	1.5	0.2
10	100	0	0	5.4	3.9	1.3	0.4

Conclusion :

Au-delà de me perfectionner dans le langage c, le projet m'a permis de bien comprendre les arbres binaires en générale ainsi que son utilité. Aujourd'hui, la gestion de données est un secteur passionnant et le projet en est une preuve. Avec une grosse quantité de donnée, l'homme est capable d'énormément de choses.