Explanation of the MyARF code

August 3, 2016

1 What is the link between the different class

The ARF is suppose to be a tree. GeneralNode is a virtuale mother class of the two different class Node and Leaf which will compose our tree. The Node have to attribute which are GeneralNode pointer that suppos to never be null. He also got a char vetor to make an economique doolean vector that indicate if a son is or not a leaf. The Leaf only got to bollean value: the fisrt is to know is there is a Key in the multi dimenssional range modelised by the leaf in the data base , and the other is set at true when the leaf have been recently consulted. Both of the class have a pointeur to his father.

the ARF have only in attribute his root which is a node, but as all node all connected by the pointer system we can reach any leaf by navigate in the structure.

The class DataVector is not directly conected but use the ARF public funtion. It represente the data base modelised by the ARF.

2 The Only effective function of general node

The public ¡GeneralNode¿ function getFather return the adress of the current ¡GeneralNode¿ father. It is usefull when we have to split a leaf durring the addaptation for example.

3 Nothing extraordinary about the Leaf class

The attribut value is private and used is public (my mistake that should change quikly). If it is necessary there is get and seter on this attributs. There is a ¡printData¿ function who's print also the adress of the leaf to identify them precisely. Obviously it print the value of the boolean attribute ¡value¿.

4 The class Node

4.1 How to creat a Node object

If you want to creat a Node object and use him in this program you first have to call the two statitcs function ¡SetDim¿ and ¡SetminRangeSize¿ because this two static attributs are really important in the other function of the class. Once it is done, you can call the tree diffrent constructor with the default one which is not really important, the copied one which only copy the adress and do not creat similar son and father, and the construction with one attribut which is the father. The last constructor is the most useful, it is used allmost all the time in my code to split a leaf for exemple.

4.2 Useful function to add a key in the ARF

This is the ARF class which allow to navigate between the deferent node of the ARF with the ¡getChildNodes¿ function. But it also use the function ¡SonIsLeaf¿ to know if it have to continue or not. the ARF ¡addKey¿ function put the key the deepeest is possible leaf, in the leaf which modelise a multi dimenssionnal range of ¡minRangeSize¿ size. To set the coresponding leaf at true we use the ¡ChangeLeafValue¿ function. If the leaf is not deep enough, we use the ¡Split¿ function till reach the minRangeSize.

The ¡Split¿ function is called by the Node which is the father of the not deep enough leaf. This node will automaticaly be connected to the new Node object created. this node all his son will be leafs with false value. Sometime it will be necessary to insert once again some key in the ARF.

4.3 Useful function to reduce the ARF size

After add the key contained in the data base you have to reduce the SRF size. this is done by merging leafs whiwh respect few caracteritics.

Frist if one leaf got all his siblings that are leafs and with the same value, we merdge the father node. We detect this kind of leafs with the function *Only-GotLeafChild* and *LeafGotSameValue* used on the father get by the ¡getFather¿ function.

The second caracteristics is to have a minimum number of siblings which not have been consultd recently. we detect this kind of situation by using the ¡GotUsefulLeafs¿ function, and the minimum number is indicate with the parameter ¡min¿ of the function.

Once you have detected a candidate to be merge you have to call the ¡Merge¿ function and recover the leaf pointer returned. After this you have to conect the the new leaf to the great father and delete the old node merged.

5 How work the ARF

The ARF class is the main class, it manage all the little bricks previously described. it got as attributs ¡dim¿, ¡domain¿, and ¡minRangeSize¿ which describ the space of the data. It also got ¡numberOfNode¿, ¡numberOfLeaf¿, and the set of leaf pointer, ¡myLeafs¿ which contain all the leafs of the ARF, to the reducing process of the ARF. The ¡targetSize¿ is the number of bit that cost the ARF after call the reducing function. The attribut ¡minUseFullLeaf¿ his the minimum number of usefull leafs of a node to merge it.

5.1 How to creat a Node object

To creat a ARF object you only have to call his undefault contructor once you know the dimenssion and the domain of your data base. the minimum range size you will indicate will be the minimum distance between two vector detectable.

Once the ARF is creat you can print it by call the function ¡printARF¿

5.2 How to navigate inside the ARF

Once your ARF is created you will can navigate inside him to find the corresponding leaf to one data fro example. To do it you will call the function ¡navigate¿ which will return the Ã©eaf pointer coresponding to your data value. That function will use the ¡nextNode¿ and ¡nextMiddleAndRangeSize¿ functions to run. When you call this function which run recursivly, you will have to indicate you key but also the middle of th entier space and his size. This parameters will change during the running. They allow to implicitly know where you are in the tree at any moment.

5.3 How to add a Key to your ARF

If you want to add one key you only have to call the function ¡AddKey¿ and indicate a key with the same dimenssion and domaine that the ARF caller. In contrary to the bloom filter ther is no discretisation needed. This function use the navigation previously described.

5.4 How to check if a key is present or got a neigbor in the database

If you want to know if a key is exactely present in the data base (i mean we accept a rate of false positif), you can call the ¡CheckKey¿ function and indicate a key with the same dimenssion and domaine that the ARF caller.

If you want to look around a Key in the data base, you have to descretise previously (with the functions of the DataVector class in my program) your key and indicate in a vector the discretised neighbor of your key. The neighbors will be checked one by one and this is it which will allow you to know if there our not a neighbor of your key in the database. The ¡DiscretiseWV¿ function of the class also allow to discretise one key to make the discretised necessary neighbors.

5.5 How to addapt your ARF and reduce it

If you have un uniform query or data, it is ore efficient to have an unbalanced tree. So we have implemented the ¡DoAddapt¿ function which split a Leaf when it give a wrong answer. You only to indicate he concerned leaf, the database, the middle of the space represented by the concerned leaf and its size.

Once you have split the leaf the node created only got leaf setted at false, so the ¡DoAddapt¿ function call the ¡QuickAddKey¿ which set the leafs at true when it is necessary but do not modify the tree like the ¡AddKey¿ function do.

The ¡DoAddapt¿ function reduce automatically the ARF theoric size to the wanted size indicate at the construction. It is just necessary to to call the ¡erase¿ function.

6 The DataVector class

The DataVector class will contain either the data or the request for the ARF that we want to test.

Thus the class contain a vector of vector of int, the one int vector is one real data (we mean continous data) and the main vector represent all the real data in the data base. It also containt th dimenssion, and the domain of the space. Equaly it contain its number of elements, and the minimum size of a range represented by a leaf of the ARF which it will test.

6.1 How to build one DataVector

First you have to call one of the two constructor function , the default one or the other which allow to choose the dimension the domain an dthe number of real vector in your object.

There is also a copy constructor which load into the new object the important vector.

6.2 How to creat specifics DataVector

We have two function which allow to creat a distant enough or a similar DataVector which will permit to know the false positiv rate of the ARF.

To creat a distance enough DataVector of your current object you have to call the makeDifferent function that will return a DataVector object with the number of real element specified and which have all his vector distant enough of all real vectors of the caller compare to the similarity placed in parameters. If the similarity, the domaine and the number of the real elemants of the caller do not allow to creat easily the different data vector it will be disp on the standar output, beacause you take a risk to make a repartition of request not uniforme and false results.

To creat a similar DataVector of your current object you have to call the makeSimilar function that will return a DataVector object with the number of real elements specified (but can not be bigger than the number of real elements of the caller), and which have all his vector similar of at least on of the real vectors of the caller compare to the similarity placed in parameters.

6.3 How to interacte with the ARF

Once you have creat your object, you can interate with a ARF. You can add all the key coresping to your real value of your DataVector object by call the function ¡AddKey¿ which insert all the key into the ARF placed in parameters.

You can also check the exact presence of all the key of your DataVector object by call the function checkPresence which test the presence into the ARF placed in parameters.
And finaly you can check around all the keys contained into the object by call the function ¡checkPresneceAround¿ and indicate the ARF in parameter and the data base which is a DataVctor pointer for the addaptation.

6.4 Disp the DataVector

If you want to disp the values of your data vector on the standard output, you can call the function ¡printDataVector¿. The boolean parameters allow to choose which vector of the object you want to disp.