Homework 1 – OOP

Developing non-oop applications using Python modules.

Problem 9:

Sa se proiecteze si se implementeze un modul pentru matrici rare tridimensionale de numere reale. O matrice este rara daca numarul de elemente nenule este mai mic decat jumatatea numarului total de elemente. Reprezentarea matricilor se realizeaza prin memorarea doar a valorilor nenule, specificate ca-tupluri $\langle i, j, k, v \rangle$ unde i, j si k reprezinta indicii din matrice, corespunzatori valorii nenule v. Modulul trebuie sa permita rezolvarea urmatoarele probleme:

- Crearea matricii;

- Afisarea matricii;

- Determinarea dimensiunii matricii;

- Citirea unei valori din matrice si scrierea valorii in matrice (elementul se specifica prin indici); - Operatiile de adunare si scadere a doua matrici;

- Accesul la un element din matrice prin indici;

- Crearea unei matrici ca matrice unitate;

- Crearea unei matrici ca matrice nula;

Student: Efrem Ion-Stefan

Facultatea de Automatica, Calculatoare si Electronica

Anul de Studiu 2, Grupa CEN 2.1

Profesor indrumator : Marius Brezovan

1. Application Design

The problem requires us to create 3D matrixes and functions/modules that work with the information from the matrixes. But in this case we don't have normal matrixes, but rare matrixes that have less than half of their elements different than 0.

In this case it is considered a waste of memory to store the matrix as a three dimensional list so we store each element in a 4-element list of form < i, j, k, v> where:

- i is the line position of the element
- j is the column position of the element
- k is depth position element (here I use depth instead of layer cause I like this notation more for the third dimension)
- v is the value of the element stored at position <i, j, k>

Then we store all of the elements given through these tuples of 4 < i, j, k, v> in another list that holds all the elements of that matrix called "matrix_elements".

For each matrix we will also have another tuple saved in a 4-element list "dimensions" that holds:

- max_line is the maximum line value from the elements saved in the matrix_elements
- max_col is the maximum column value from the elements saved in the matrix_elements
- max_depth is the maximum depth value from the elements saved in the matrix_elements
- number_of_elements which holds the number of non-null elements in the case of every non-null matrix(in the case of a null matrix it holds the number of all elements)(also I know len(matrix_elements) would give me the number of elements of the matrix but I liked storing the value in the tuple and just went with it)

The first function createMatrix(matrix_elements) receives an empty list of elements that will be filled with elements given as input from the keyboard, after every input, we verify if the coordinates of the new element are bigger than the maximums we have and update the maximums accordingly. We return the list “dimensions” for that matrix.

The second function printMatrix(matrix_elements, dimensions) receives the two lists of the matrix and prints all the values in the matrix, be it a number or 0. It prints the 3D matrix as a number of 2D matrixes of the same size with blank lines separating them for a better view of the matrix (we print 2D matrixes of i lines and j columns k times)

The third function matrixDimension(dimensions) receives the list of dimensions of the matrix and outputs them on the screen, it also outputs the number of elements different than 0 (all elements in the case of the null matrix)

The 4th function readWriteElement(matrix_elements, dimensions, line, col, depth) solves both the 4th and the 6^{th} requirements of the problem as it allows you to read a value, write/modify a value or add a value and even extend the matrix by adding it, so it solves both questions. It receives the two lists previously mentioned and the coordinates of the element you wish to get access to as parameters.

The next 2 funtcions are pretty similar createUnitMatrix(matrix_elements) and createNullMatrix(matrix_elements), they are similar to the createMatrix function as they also return the “dimensions” list but they create a unit matrix of the size entered by the keyboard or a null matrix.

The last function addSubMatrix(matrix_elementsA, dimensionsA, matrix_elementsB, dimensionsB, matrix_elementsC) adds or subtracts the two matrixes whose lists are given as parameters as well as the empty list of elements for the third matrix that will be created as the sum or difference of matrix A and B.

This function is interesting as it gives you 2 choices:

1. You create a B matrix with the same maximum size as the A one and add/subtract them as in any math problem. It requires you to create a matrix with the exact same size or the program will give you an error and ask you to redo the creation of the B matrix.
2. You create a B matrix at your choice and the C matrix created by the sum/difference of the 2 matrixes will have the maximum size of the two. For ex, if A is of size <2, 3, 3> and B is of size <1, 4, 2> the C matrix will have a size of <2, 4, 3>.

2. Input specification

The input is required for every step, from the creation of the matrixes to the way you want to add/subtract the matrixes, and the requirement to create a same size matrix

3. Output specification

The output is mainly made of messages that guide your input and also the matrixes that are always printed to make sure that the program is correct and that the one that inputs the values has a good idea of what the outcome of the operations of adding/subtracting the matrixes will be.

4. Achievments and Conclusions

I really liked how this problem tested my ability to transition my knowledge from C++ to Python. I started solving the problem in C++ at first to get an idea of it and I liked it but I found it time consuming at first with all the dynamic memory allocation and reallocation when you wanted to add an element and so on. But, when I started working on the problem in Python, 2 days after I researched the language more, watched videos and played with some small problems and programs, I realized the problem was going a lot smoother than I thought. The number of useful functions you can use with list and the fast way to iterate through them by using “for I in x” rather than iterating in C++ really made me like this language and made me solve problems that in C would take a lot more

lines of code and might be ambiguous to the reader. I still have troubles understanding how you can save the information you give to a list in a module/function, I realized that if you append a value to it, it keeps the value even after the function ends but if you a sign a list to another list, for example matrix_listC = matrix_listA it will not keep the assigned value after the function ends. I will research this more but this is what caught my attention, the scope of the elements are still a little unclear to me but I will look into that until the next assignment.