Project 1, Part1

3SUM algorithm

Jose Eduardo Gamboa Barraza
Universidad Autonoma de Guadalajara

Guadalajara, Jalisco

The 3sum algorithm is a common programming problem, consisting of finding triplets inside a list of numbers, that summed get a predefined result, in this case, getting the result of zero. There are different methods or possible solutions to this problem, depending on the interpretation given or the different restrictions it might have.

I. Introduction

This problem will be solved using a O(n^2) with no triplet being repeated, different files, of different sizes, will be used to prove the efficiency of the algorithm used.

Answering questions for each file, such as triplets found and the time it took the algorithm to do so, we expect to see the benefits of using this method, or simply compare to other methods that might work faster or not, aswell as comparing the time it will take a O(n^3) to finish, with ours.

The lists used have random numbers ranging from -100 to 100, and N ranging from one thousand to a million, N being the size of the list used.

II. Description

This problem was divided into two parts, the first part, consisted of a group of different files, each having a different list size, with a total of six different list, starting at one thousand numbers and ending at a million, as stated before, all list contain numbers using the random() function in python, the reason for this, was to test how fast could the algorithm used find all the different triplets, especially at the million numbers list, as the last one, is the biggest, meaning that some algorithms could take years to solve the issue.

The second part of this problem, was to generate one hundred files, using different seeds, to make a list of one hundred thousand numbers, each.

Factor taken into consideration is the time it took to compute the results, as the amount of triplets found has a limit. The first one shows how efficient the algorithm is.

III. Solution

As stated before, to solve the 3SUM problem, we find three numbers among a list, that summed up get zero as a result, the worse time complexity is a O(*n*^3), being the easiest solution, if we consider the triplets as (*x,y,z*) using the first method, we have to cycle through x from zero to *n*-2, inside every *x* loop, we have another loop for *y* from *x*+1 to *n*-1, then there is a third one for *z* that goes from *y*+1 to *n*, this is called the natural solution, as it is the easiest to come up with, but also the slowest, in this case if *n* was one million, the iterations needed to cycle through every possibility would be of one quintillion iterations.

For this reason, another algorithm was used for this problem, first the list is sorted in ascending order, meaning -100 will appear first and 100 last, in case both numbers appeared in said list, followed we have the first loop for *x* that goes from zero to *n*-2, inside, there is a second loop that goes until *y* is bigger than *z*, this means *y* will start in *x*+1 in ascending order and *z* will start at *n* in descending order, if the sum of the numbers in the list positioned in x,*y,z* is greater than zero, then you subtract one from *z,* if the result is less than zero, you add one to *y,* eventually *y* and *z* will meet, meaning the whole list was cycled through and *x* moves onto the next position, this is possible, because the list was ordered previously.

In spite of using this last method, the results obtained were still to big to compute in relatively short time, even though this period was reduced from years, to approximately 9 days; duplicating triplets was not allowed to shorten the total time.

Both problems (different sized lists and a hundred lists of one thousand numbers) were solved using the same algorithm, with slight modifications to meet the differences each problem required.

IV. Results

Following, is a table with the results obtained in the first part, with O(n^3).

Table 1. Results with O(n^3)

List	Time in seconds	Iterations	Triplets found
1k	27.6	1B	124,829
2k	162	8B	501,923
5k	1,875	125B	3,161,077
10k	*14285	1T	-
100k	*14.28M	1Quad	-
1M	*14.28B	1Quint	-

*Estimated number, after calculating an average of Iterations per second in 1k, 2k and 5k lists.

As we can see in *n*=1M, the seconds, translated to years equals to more than 450, even in such a small value for *n*, as 10 thousands is, this algorithm is too slow.

Results were the second algorithm O(n^2) was used, but duplicates were considered, is showed next.

Table 2. Results with O(n^2)

List	Time in seconds	Iterations	Triplets found
1k	0.8	1M	124,829
2k	3.28	4M	501,923
5k	20.65	25M	3,161,077
10k	82.81	100M	12,451,653
100k	*8,000	10B	-
1M	*800,000	1T	-

*See table 1.

Time was reduced considerably, as seen on Table 2, but even after reducing the iterations needed, it is still too slow to find all the triplets, we can see that at just one million numbers the time is already at 9 days, as the amount of triplets found grows exponentially.

After taking this matter into consideration , duplicates were no longer allowed, with the results obtained as follows.

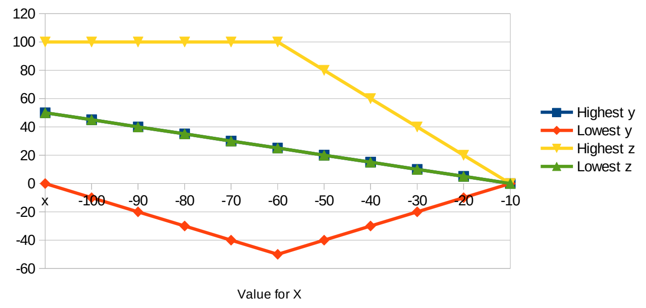
Table 3. Results with O(n^2) with no duplicates.

List	Time in seconds	Triplets found
1k	0.14	5,034
2k	0.27	5,101
5k	0.53	5,101
10k	0.91	5,101
100k	9.14	5,101
1M	102.32	5,101

We can notice how time was reduced, by being barely above one hundred seconds at one million as n's value, also it is noticeable how the amount of triplets found tops at *5101.*

After observing the results in the files, we can notice values taken by *x,y* and *z,* were *x* ranges from -100 to 0, and *y, z* behaviors can be observed in the next graph.

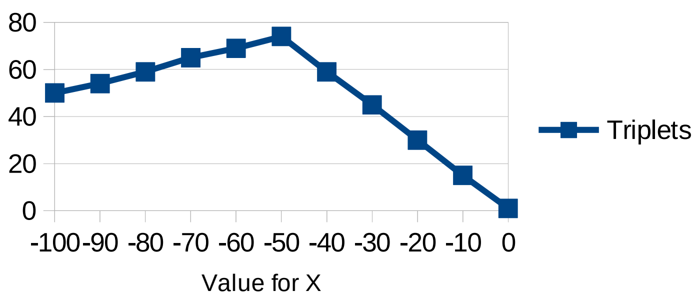
Graph 1. Behaviour for Y and Z



We can see from the graph, tat the lowest *z* and highest *y* values will always be the same, which is the absolute half for x value.

The next graph shows the amount of triplets found per each value of *x*.

Graph 2. Triplets found



Notice how the number of triplets found rises and drastically falls after *x* = 50, until there is only one triplet left and *x* = 0.

For the second part, we have one hundred files of the same size, the same algorithm was used to solved this and the results obtained are showed in the next table.

Table 4. Results for one hundred lists.

Time in seconds	Triplets
1346	510,100

We can see that the amount of triplets found equals to maximum number of triple multiplied by the number of files; time doesn't follow the same pattern, as it averages 13.5 seconds per file, as opposed to the result in Table 3, which is 9.14 seconds.