A 3SUM algorithm on the binary addition level of atomic generated real numbers

Carolin Zöbelein*

December 15, 2020

DRAFT

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Keywords: 3SUM Problem, Real Numbers, Irrational Numbers, Data Structures, Binary Representation, Data Storage Representation, Nonnumerical Algorithm, Complexity
ACM Subject Classes: E1, E2, F1.3, F2.2

Contents

*The author believes in the importance of the independence of research and is funded by the public community. If you also believe in this values, you can find ways for supporting the author's work here: `https://research.carolin-zoebelein.de/funding.html`, Email: `contact@carolin-zoebelein.de`, PGP: D4A7 35E8 D47F 801F 2CF6 2BA7 927A FD3C DE47 E13B, `https://research.carolin-zoebelein.de` LaTeXsource availabe at `https://github.com/Samdney/notes-3sum`, `id: notes_0003`, 😎

Preamble

The following content is a sketch for discussion purposes only, without warranty for mathematical completeness.

1 Introduction

In this notes we will give a sketchy discussion of an algorithm for solving the 3SUM problem.

2 Preliminaries

Some basic definitions and preparatory work.

2.1 Problem Statement

Let L be a list of n integers n_i, $i \in \mathbb{N} : i = 1, \ldots, n$. All n integers are given in a binary representation of b bits each. Negative integers are given by their binary two's complement representation.

Definition 1 (3SUM Problem)**.** *The 3SUM problem asks, if a given list L of n integers contains three elements n_i, n_j and n_k, such that $n_i + n_j + n_k = 0$ for $i \neq j \neq k$.*

2.2 Binary addition

Our algorithm will work on the binary representation level of the given integers n. For this, we will have a short look at the different possible cases of bits for the addition of three numbers n_i, n_j and n_k.

Notations. We will index the bits b_i of n_i from right to left with $q = 0, \ldots, b-1$ and denote the q'th bit of n_i by $b_i[q]$.

2.2.1 Zero sum binary addition

Since we are interested in the special case of zero sum of three numbers, we discuss this case, at first.

We start with bit $q = 0$. Here, we always have exactly three bits for our addition (the 0'th bit of each of our numbers n_i, n_j and n_k). Since, we are only interested in the zero sum case, we have to think for which combinations of three bits we can get 0, at all. This are:

1. $\mathcal{C}_{0,0} = 3, \mathcal{C}_{0,1} = 0, c_0 = 0$: $(0,0,0)$
2. $\mathcal{C}_{0,0} = 1, \mathcal{C}_{0,1} = 2, c_0 = 1$: $(1,1,0), (1,0,1), (0,1,1)$

$\mathcal{C}_{q,0}$ denotes the number of 0's and $\mathcal{C}_{q,1}$ the number of 1's in our three q bits $b_i[q]$, $b_j[q]$ and $b_k[q]$. c_q denotes the carrier of the given bit addition $b_i[q] + b_j[q] + b_k[q]$.

If we have $c_0 = 0$, then, for bit $q = 1$, we will have the same situation like for bit $q = 0$. Hence, let's look at the case of having a carrier $c_0 = 1$ and what will happen then for bit $q = 1$. Because of the carriere $c_0 = 1$, we have one bit more, hence four bits, now. We get zero sum for:

1. $\mathcal{C}_{1,0} = 0, \mathcal{C}_{1,1} = 4, c_1 = 10$: $(1,1,1,1)$
2. $\mathcal{C}_{1,0} = 2, \mathcal{C}_{1,1} = 2, c_1 = 1$: $(1,1,0,0), (1,0,1,0), (1,0,0,1)$

In the first case with carrier $c_1 = 10$, we will get for $q = 2$ the three bit case again, and for $q = 3$ the four bit case again. Depending if we choose $(0,0,0)$ or $\{(1,1,0),(1,0,1),(0,1,1)\}$ for bit $q = 2$, we will finally get for bit $q = 3$ a four respectively five bit case (see also table 1). In the second case with carrier $c_1 = 1$, we will get for $q = 2$ a four bit case again.

Table 1: Binary addition of three numbers with maximal carrier.

		1		1				
1	0	1	0	1	0	1		
$\cdots$	$\cdots$	1	1	1	1	1	1	
$\cdots$	$\cdots$	1	1	1	1	1	1	
$\cdots$	$\cdots$	0	0	0	0	1	0	
$\cdots$	$\cdots$	0	0	0	0	0	0	
$\cdots$	$\cdots$	5	4	3	2	1	0	q

So, let's assume we have a five bit case for bit $q = 3$ and get zero sum for:

1. $\mathcal{C}_{3,0} = 1, \mathcal{C}_{3,1} = 4, c_3 = 10$: $(1,1,0,1,1),(1,1,1,0,1),(1,1,1,1,0)$
2. $\mathcal{C}_{3,0} = 3, \mathcal{C}_{3,1} = 2, c_3 = 1$: $(1,1,0,0,0)$

Here, we have a very similiar situation like for the case with four bits at $q = 1$. If we have carrier $c_3 = 10$ for bit $q = 3$, we will have the three bit case for $q = 4$ and the four or five bit case for $q = 5$. If we have carriere $c_3 = 1$, we will have the four bit case for bit $q = 4$.

After this short discussion of possible cases, we will write this in a more formal way, next.

2.2.2 Zero sum bit rules

For the binary addition of the three numbers n_i, n_j and n_k we can put the results for the zero sum case together to a complete set of rules. We will write short $\mathcal{C}_{q,b}^{c_q} := (\mathcal{C}_{q,0}, \mathcal{C}_{q,1}, c_q, b)$ with b be the number of bits of the belonging case, for our possible bit cases. With this we can write:

- Case: $b = 3$

$$\begin{aligned} \mathcal{C}_{q,3}^{0} &\to \{\mathcal{C}_{q+1,3}^{0}, \mathcal{C}_{q+1,3}^{1}\} \\ \mathcal{C}_{q,3}^{1} &\to \{\mathcal{C}_{q+1,4}^{10}, \mathcal{C}_{q+1,4}^{1}\} \end{aligned} \tag{1}$$

- Case: $b = 4$

$$\begin{aligned} \mathcal{C}_{q,4}^{10} &\to \{\mathcal{C}_{q+1,3}^{0} \to \{\mathcal{C}_{q+2,4}^{10}, \mathcal{C}_{q+2,4}^{1}\}, \mathcal{C}_{q+1,3}^{1} \to \{\mathcal{C}_{q+2,5}^{10}, \mathcal{C}_{q+2,5}^{1}\}\} \\ \mathcal{C}_{q,4}^{1} &\to \{\mathcal{C}_{q+1,4}^{10}, \mathcal{C}_{q+1,4}^{1}\} \end{aligned} \tag{2}$$

- Case: $b = 5$

$$\begin{aligned} \mathcal{C}_{q,5}^{10} &\to \{\mathcal{C}_{q+1,3}^{0} \to \{\mathcal{C}_{q+2,4}^{10}, \mathcal{C}_{q+2,4}^{1}\}, \mathcal{C}_{q+1,3}^{1} \to \{\mathcal{C}_{q+2,5}^{10}, \mathcal{C}_{q+2,5}^{1}\}\} \\ \mathcal{C}_{q,5}^{1} &\to \{\mathcal{C}_{q+1,4}^{10}, \mathcal{C}_{q+1,4}^{1}\} \end{aligned} \tag{3}$$

Notations. We write $\mathcal{C}_{q,b}^{c_q}(0)$ to get the value of $\mathcal{C}_{q,0}$ from $\mathcal{C}_{q,b}^{c_q}$, $\mathcal{C}_{q,b}^{c_q}(1)$ to get the value of $\mathcal{C}_{q,1}$ from $\mathcal{C}_{q,b}^{c_q}$, $\mathcal{C}_{q,b}^{c_q}(c)$ to get the value of c_q from $\mathcal{C}_{q,b}^{c_q}$ and $\mathcal{C}_{q,b}^{c_q}(b)$ to get the value of b from $\mathcal{C}_{q,b}^{c_q}$.

3 The 3SUM Algorithm

Now we want to describe the final algorithm.

3.1 Idea

We want to outline our 3SUM algorithm, at first.

3.1.1 Basic concept

Typical 3SUM algorithms like XXX [**?**] and YYY [**?**] works on the decimal base angel of view by going in different ways through the numbers n of the given list L. In our algorithm, we move away from this approach. Instead, we will go through the binary representation of the given numbers. So, the final algorithm will be mainly dominated by the number of bits b of integer representation which we have to examine and not by the number of elements n of L.

What we do. Since, we examine the 3SUM problem, we consider the binary addition of three integer numbers. We know from our previous discussion that only certain combinations of bits resulting in zero sums and that the possibilities for bits of position q depend from the possibilities for bits of position $q-1$. We begin with the whole list L and split it into two sublists. One which contains all numbers with 0 at bit position $q=0$ and the other list which contains all numbers with 1 at bit position $q=0$. We know, that for bit $q=0$ we have two possible rules which lead to a zero bit $(\mathcal{C}^0_{q,3}, \mathcal{C}^1_{q,3})$. We check for the first rule if the two sublists have enough elements (we also save the amount of numbers of each sublist) to statisfy this rule. If yes, then we safe this rule, if no, we throw it away. Then we do the same for the second rule. So, we have solved the 3SUM problem for bit $q=0$.
In the next step, we split again each of our sublists into two further sublists. At this time now, we do it for bit $q=1$ instead of bit $q=0$, of course. Then, we go through the list of our saved rules. We take the first saved rule and determine its two successor rules. We take the first succssor rule and check our new sublists if they have enough elements to statisfy this rule. If yes, we also have to check if the necessary amounts of numbers in this sublists are not in conflict with the amounts from our step before. If no, then safe this new rule, else through it away. In the same way, we do it for all possible rule successors.
After repeating this steps for all bits b of our bit representation, we get left with the set of all rules and their belonging numbers which solve the 3SUM problem.
Hence, this algorithm depends on the number of bits of bit representation but not primary on the number of elements of n. Each time if we do a list split we can already count the number of elements of this list during spliting and no search, at no point in the algorithm, is necessary, if we implement it with two binary trees (one for the number lists and one for the rules) and pointers which always point to the last necessary elements of this trees.

3.1.2 Algorithm

3.1.3 Pseudocode

3.1.4 Complexity

Acknowledgement

Thanks to the private donators who financially support this work.