3SUM

From Wikipedia, the free encyclopedia

In computational complexity theory, the **3SUM** problem asks if a given set of n real numbers contains three elements that sum to zero. A generalized version, rSUM, asks the same question of r numbers. 3SUM can be easily solved in $O(n^2)$ time, and matching $\Omega(n^{\lceil r/2 \rceil})$ lower bounds are known in some specialized models of computation (Erickson 1999).

Unsolved problem in computer science:

Is there an algorithm to solve the 3SUM problem in time $O(n^{2-\epsilon})$, for some $\epsilon > 0$?

(more unsolved problems in computer science)

It was widely conjectured that any deterministic algorithm for the 3SUM requires $\Omega(n^2)$ time. In 2014, the original 3SUM conjecture was refuted by Allan Grønlund and Seth Pettie who gave a deterministic algorithm that solves 3SUM in time $O(n^2/(\log n/\log\log n)^{2/3})$.[1] It is still conjectured that 3SUM is unsolvable in $O(n^{2-\Omega(1)})$ expected time.[2]

When the elements are integers in the range $[-N, \dots, N]$, 3SUM can be solved in $O(n + N\log N)$ time by representing the input set S as a bit vector, computing the set $S + S$ of all pairwise sums as a discrete convolution using the Fast Fourier transform, and finally comparing this set to $-S$.

Contents

Quadratic algorithm

Suppose the input array is $S[0..n-1]$. 3SUM can be solved in $O(n^2)$ time on average by inserting each number $S[i]$ into a hash table, and then for each index i and j, checking whether the hash table contains the integer $-(S[i] + S[j])$.

Alternatively, the algorithm below first sorts the input array and then tests all possible pairs in a careful order that avoids the need to binary search for the pairs in the sorted list, achieving worst-case $O(n^2)$ time, as follows.[3]

```
sort(S);
for i=0 to n-3 do
   a = S[i];
   start = i+1;
   end = n-1;
   while (start < end) do
      b = S[start]
      c = S[end];
      if (a+b+c == 0) then
         output a, b, c;
         // Continue search for all triplet combinations summing to zero.
          end = end - 1
      else if (a+b+c > 0) then
         end = end - 1;
      else
         start = start + 1;
      end
   end
end
```

The following example shows this algorithm's execution on a small sorted array. Current values of **a** are shown in green, values of **b** and **c** are shown in red.

```
-25 -10 -7 -3 2 4 8 10  (a+b+c==-25)
-25 -10 -7 -3 2 4 8 10  (a+b+c==-22)
. . .
-25 -10 -7 -3 2 4 8 10  (a+b+c==-7)
-25 -10 -7 -3 2 4 8 10  (a+b+c==-7)
-25 -10 -7 -3 2 4 8 10  (a+b+c==-3)
-25 -10 -7 -3 2 4 8 10  (a+b+c==2)
-25 -10 -7 -3 2 4 8 10  (a+b+c==0)
```

The correctness of the algorithm can be seen as follows. Suppose we have a solution a + b + c = 0. Since the pointers only move in one direction, we can run the algorithm until the leftmost pointer points to a. Run the algorithm until either one of the remaining pointers points to b or c, whichever occurs first. Then the algorithm will run until the last pointer points to the remaining term, giving the affirmative solution.

Variants

Non-zero sum

Instead of looking for numbers whose sum is 0, it is possible to look for numbers whose sum is any constant *C* in the following way:

- Subtract *C*/3 from all elements of the input array.
- In the modified array, find 3 elements whose sum is 0.

3 different arrays

Instead of searching for the 3 numbers in a single array, we can search for them in 3 different arrays. I.e., given three arrays X, Y and Z, find three numbers $a \in X$, $b \in Y$, $c \in Z$, such that $a+b+c=0$. Call the 1-array variant 3SUMx1 and the 3-array variant 3SUMx3.

Given a solver for 3SUMx1, the 3SUMx3 problem can be solved in the following way (assuming all elements are integers):

- For every element in X, set: $X[i] \leftarrow X[i]*10+1$.

- For every element in Y, set: Y[i] ← Y[i]*10+2.
- For every element in Z, set: Z[i] ← Z[i]*10-3.
- Let S be a concatenation of the arrays X, Y and Z.
- Use the 3SUMx1 oracle to find three elements a'∈S, b'∈S, c'∈S such that a'+b'+c'=0.
- Because the LSD (least significant digit) of the sum is 0, the LSDs of a', b' and c' must be 1, 2 and 7 (in any order). Suppose wlog that the LSD of a' is 1, b' is 2 and c' is 7.
- Return a ← (a'-1)/10, b ← (b'-2)/10, c ← (c'+3)/10.

By the way we transformed the arrays, it is guaranteed that a∈X, b∈Y, c∈Z.[4]

Convolution sum

Instead of looking for arbitrary elements of the array such that:

$$S[k] = S[i] + S[j]$$

the *convolution 3sum* problem (Conv3SUM) looks for elements in specific locations:[5]

$$S[i+j] = S[i] + S[j]$$

Reduction from Conv3SUM to 3SUM

Given a solver for 3SUM, the Conv3SUM problem can be solved in the following way.[5]

- Define a new array *T*, such that for every index *i*: $T[i] = 2nS[i] + i$ (where *n* is the number of elements in the array, and the indices run from 0 to *n*-1).
- Solve 3SUM on the array *T*.

Correctness proof:

- If in the original array there is a triple with $S[i+j] = S[i] + S[j]$, then $T[i+j] = 2nS[i+j] + i + j = (2nS[i] + i) + (2nS[j] + j) = T[i] + T[j]$, so this solution will be found by 3SUM on *T*.
- Conversely, if in the new array there is a triple with $T[k] = T[i] + T[j]$, then $2nS[k] + k = 2n(S[i] + S[j]) + (i + j)$. Because $i + j < 2n$, necessarily $S[k] = S[i] + S[j]$ and $k = i + j$, so this is a valid solution for Conv3SUM on *S*.

Reduction from 3SUM to Conv3SUM

Given a solver for Conv3SUM, the 3SUM problem can be solved in the following way.[2][5]

The reduction uses a hash function. As a first approximation, assume that we have a linear hash function, i.e. a function *h* such that:

$$h(x+y) = h(x) + h(y)$$

Suppose that all elements are integers in the range: 0...*N*-1, and that the function *h* maps each element to an element in the smaller range of indices: 0...*n*-1. Create a new array *T* and send each element of *S* to its hash value in *T*, i.e., for every *x* in *S*:

$$T[h(x)] = x$$

Initially, suppose that the mappings are unique (i.e. each cell in *T* accepts only a single element from *S*). Solve Conv3SUM on *T*. Now:

- If there is a solution for 3SUM: $z = x + y$, then: $T[h(z)] = T[h(x)] + T[h(y)]$ and $h(z) = h(x) + h(y)$, so this solution will be found by the Conv3SUM solver on *T*.
- Conversely, if a Conv3SUM is found on *T*, then obviously it corresponds to a 3SUM solution on *S* since *T* is just a permutation of *S*.

This idealized solution doesn't work, because any hash function might map several distinct elements of *S* to the same cell of *T*. The trick is to create an array *T** by selecting a single random element from each cell of *T*, and run Conv3SUM on *T**. If a solution is found, then it is a correct solution for 3SUM on *S*. If no solution is found, then create a different random *T** and try again. Suppose there are at most *R* elements in each cell of *T*. Then the probability of finding a solution (if a solution exists) is the probability that the random selection will select the correct element from each cell, which is $(1/R)^3$. By running Conv3SUM R^3 times, the solution will be found with a high probability.

Unfortunately, we do not have linear perfect hashing, so we have to use an almost linear hash function, i.e. a function *h* such that:

$$h(x + y) = h(x) + h(y) \text{ or}$$
$$h(x + y) = h(x) + h(y) + 1$$

This requires to duplicate the elements of *S* when copying them into *T*, i.e., put every element $x \in S$ both in $T[h(x)]$ (as before) and in $T[h(x)] - 1$. So each cell will have 2*R* elements, and we will have to run Conv3SUM $(2R)^3$ times.

3SUM-hardness

A problem is called **3SUM-hard** if solving it in subquadratic time implies a subquadratic-time algorithm for 3SUM. The concept of 3SUM-hardness was introduced by Gajentaan & Overmars (1995). They proved that a large class of problems in computational geometry are 3SUM-hard, including the following ones. (The authors acknowledge that many of these problems are contributed by other researchers.)

- Given a set of lines in the plane, are there three that meet in a point?
- Given a set of non-intersecting axis-parallel line segments, is there a line that separates them into two non-empty subsets?
- Given a set of infinite strips in the plane, do they fully cover a given rectangle?
- Given a set of triangles in the plane, compute their measure.
- Given a set of triangles in the plane, does their union have a hole?
- A number of visibility and motion planning problems, e.g.,
  - Given a set of horizontal triangles in space, can a particular triangle be seen from a particular point?
  - Given a set of non-intersecting axis-parallel line segment obstacles in the plane, can a given rod be moved by translations and rotations between a start and finish positions without colliding with the obstacles?

By now there are a multitude of other problems that fall into this category. An example is the decision version of X + Y sorting: given sets of numbers X and Y of n elements each, are there n^2 distinct $x + y$ for $x \in X, y \in Y$?[6]

See also

- Subset sum problem

Notes

1. Gronlund, A.; Pettie, S. (2014). *Threesomes, Degenerates, and Love Triangles*. 2014 IEEE 55th Annual Symposium on Foundations of Computer Science. p. 621. doi:10.1109/FOCS.2014.72. ISBN 978-1-4799-6517-5.
2. Kopelowitz, Tsvi; Pettie, Seth; Porat, Ely (2014). "3SUM Hardness in (Dynamic) Data Structures". arXiv:1407.6756 [cs.DS].
3. Visibility Graphs and 3-Sum (http://www.ti.inf.ethz.ch/ew/courses/CG09/materials/v12.pdf) by Michael Hoffmann
4. For a reduction in the other direction, see Variants of the 3-sum problem (http://cs.stackexchange.com/questions/37888/variants-of-the-3-sum-problem).
5. Patrascu, M. (2010). *Towards polynomial lower bounds for dynamic problems*. Proceedings of the 42nd ACM symposium on Theory of computing - STOC '10. p. 603. doi:10.1145/1806689.1806772. ISBN 9781450300506.
6. Demaine, Erik; Erickson, Jeff; O'Rourke, Joseph (20 August 2006). "Problem 41: Sorting X + Y (Pairwise Sums)". *The Open Problems Project*. Retrieved 23 September 2014.

References

- Baran, Ilya; Demaine, Erik D.; Pătraşcu, Mihai (2008), "Subquadratic algorithms for 3SUM", *Algorithmica*, **50** (4): 584–596, doi:10.1007/s00453-007-9036-3.
- Demaine, Erik D.; Mitchell, Joseph S. B.; O'Rourke, Joseph (July 2005), "Problem 11: 3SUM Hard Problems", *The Open Problems Project*.
- Erickson, Jeff (1999), "Lower bounds for linear satisfiability problems", *Chicago Journal of Theoretical Computer Science*, MIT Press, **1999**.
- Gajentaan, Anka; Overmars, Mark H. (1995), "On a class of O(n^2) problems in computational geometry", *Computational Geometry: Theory and Applications*, **5** (3): 165–185, doi:10.1016/0925-7721(95)00022-2.
- King, James (2004), *A survey of 3SUM-hard problems* (PDF).

Categories: Computational geometry | Polynomial-time problems | Unsolved problems in computer science