1 3SUM Versions

Recall the 3SUM problem: given a set S on n integers, do there exist $a, b, c \in S$ with $a + b + c = 0$? Also, the 3SUM' problem: given sets A,B,C of n integers each, are there $a \in A, b \in B, c \in C$ with $a + b + c = 0$?

In the homework you (hopefully) showed that these two problems are equivalent, so we will be using these interchangeably. We will introduce one more version: 3SUM*: The input here is a set S of integers and one needs to decide whether there are $a, b, c \in S$ such that $a + b = c$.

Theorem 1.1. *There is an $O(n)$ time reduction from 3SUM' on n numbers to 3SUM* on n numbers.*

Proof. Let A, B, C be an instance of 3SUM' with n numbers. Suppose that the numbers are in the interval $\{-W, \ldots, W\}$. Let $M = W + 1$, so that the numbers are in $\{-M + 1, \ldots, M - 1\}$.

Let $A' = \{a - 5M \mid a \in A\}$, $B' = \{b + 13M \mid b \in B\}$ and $C' = \{8M - c \mid c \in C\}$. Let $S = A' \cup B' \cup C'$.

Notice that the range of A' is $\{-6M + 1, \ldots, -4M - 1\}$, the range of B' is $\{12M + 1, \ldots, 14M - 1\}$, and the range of C' is $\{7M + 1, \ldots, 9M - 1\}$.

If $a \in A, b \in B, c \in C$, with $a + b + c = 0$, then $(a - 5M) + (b + 13M) = (-c + 8M)$, and so if there is a 3SUM' solution, then there is a 3SUM* solution.

Suppose now that there is a 3SUM* solution $s_1 + s_2 = s_3$ with $s_1, s_2, s_3 \in S$. WLOG, $s_1 \leq s_2$.

Suppose that $s_1 \notin A'$. Then $s_1, s_2 > 7M$ and so $s_1 + s_2 > 14M$ which exceeds the range of all A', B' and C'. Hence $s_1 \in A'$.

If $s_2 \notin B'$, $s_2 < 9M$ and since $s_1 \in A'$, $s_1 < -4M$. Thus $s_1 + s_2 < 5M$, and this only intersects the range of A', but not that of B' or C'. Thus $s_1 + s_2 = s_3 \in A'$. This also means that $s_2 \in A'$, as otherwise $s_2 > 7M$, and $s_1 + s_2 > 3M$ which contradicts the previous assertion that $s_1 + s_2 \in A'$. But on the other hand, if $s_2 \in A'$, we have $s_1, s_2 < -4M$ and so $s_1 + s_2 < -8M$ which is a contradiction since all numbers in A' are $> -6M$. Thus we must have $s_1 \in A'$ and $s_2 \in B'$. But then $s_1 + s_2 > -6M + 12M = 6M$, and $s_1 + s_2 < -4M + 14M = 10M$. Hence $s_3 = s_1 + s_2 \in C'$. Thus we have $a \in A, b \in B, c \in C$ such that $(a - 5M) + (b + 13M) = (-c + 8M)$ so that $a + b + c = 0$. □

One can also reduce 3SUM* to 3SUM', so that 3SUM* is yet another equivalent version to 3SUM.

Exercise: How can you reduce 3SUM* back to 3SUM'?

2 Two 3SUM-Hard problems in Computational Geometry

Let us consider two problems. The first is **Geombase** in which we are given n points in the plane $(x_1, y_1), \ldots, (x_n, y_n)$ with integer coordinates x_i and with $y_i \in \{0, 1, 2\}$ for all i. The question is, is there a non-horizontal line that passes through 3 of the points?

Theorem 2.1. *Geombase is equivalent to 3SUM.*

Proof. Geombase is equivalent to the problem whether there exist points $(x_i, 0), (x_j, 1), (x_k, 2) \in S$ so that $x_i + x_k = 2x_j$, i.e. $(x_j, 1)$ is in the middle between $(x_i, 0)$ and $(x_k, 2)$.

> **Exercise:** Using the above fact, show how you can reduce Geombase to 3SUM', so that given an instance S of Geombase on n points you can create A,B,C on at most n integers each so that the Geombase instance has a solution if and only if there are $a \in A, b \in B, c \in C$ with $a + b + c = 0$.

Now we show the reverse direction. Given a 3SUM' instance A, B, C, we create a Geombase instance S that contains for every $a \in A$, a point $(2a, 0)$, for every $b \in B$, a point $(2b, 2)$ and for every $c \in C$, a point $(-c, 1)$. A Geombase solution corresponds to $(2a, 0), (2b, 2), (-c, 1)$ with $2a + 2b = -2c$, i.e. $a + b + c = 0$, a 3SUM' solution. □

The second problem we'll look at is 3-Points-on-a-Line: Given n points in the plane, $(x_1, y_1), \ldots, (x_n, y_n)$ with integer coordinates x_i and y_i, are there three points that lie on the same line?

Theorem 2.2. *3SUM reduces to 3-Points-on-a-Line, so that under the 3SUM Hypothesis, 3-Points-on-a-Line requires $n^{2-o(1)}$ time.*

Proof. Given a 3SUM instance S, create an instance of 3-Points-on-a-Line by adding for every $s \in S$, the point (s, s^3).

$(a, a^3), (b, b^3), (c, c^3)$ are collinear if and only if $(c - a)/(b - a) = (c^3 - a^3)/(b^3 - a^3)$. Since $a \neq c$, $b \neq a$, this is equivalent to $(b^2 + ab + a^2) = (c^2 + ac + a^2)$, which is the same as $(b^2 - c^2) + a(b - c) = 0$. This is equivalent to $(b - c)(a + b + c) = 0$. Since $b \neq c$, this is the same as $a + b + c = 0$. I.e. (a, b, c) is a 3SUM solution if and only if $(a, a^3), (b, b^3), (c, c^3)$ is a 3-Points-on-a-Line solution. □

3 3SUM-Convolution

The 3SUM-Convolution problem is, given an integer array A of length n, are there i, j, $i \neq j$ so that $A[i] + A[j] = A[i + j]$?

This problem has a trivial $O(n^2)$ time algorithm: just try all pairs i, j. This is much more trivial than the $O(n^2)$ time algorithm for 3SUM.

Let's first show that 3SUM-Convolution can be reduced to 3SUM*. Given an instance A of length n of 3SUM-Convolution, let $S = \{(2n + 1)A[i] + i \mid i \in [n]\}$ be an instance of 3SUM*.

> **Exercise:** Show that there exist i and j s.t. $A[i] + A[j] = A[i + j]$ if and only if there are $s, s', s'' \in S$ with $s + s' = s''$.

Now, let us reduce 3SUM* to 3SUM-Convolution.

Say S is the 3SUM* instance. Suppose that we have some 1 to 1 function f that maps S to $[t]$, where $t = O(n)$ and such that $f(i) + f(j) = f(i + j)$. Then, we can create an array A of length t, and set for each $s \in S$, set $A[f(s)] = s$. Then, $i + j = k$ if and only if $A[f(i)] + A[f(j)] = A[f(i) + f(j)] = A[f(k)]$. This would be a very efficient reduction to 3SUM-Convolution.

However, we don't know how to create such a function. We will however not abandon the approach.

Simple hash functions. Suppose that our 3SUM* instance S is over the integers in $[\pm U] = \{-U, \ldots, U\}$ for some U. These can be represented using $d = \Theta(\log U)$ bits.

Let B be a parameter chosen later with $B << dn$ and let $m = dn/B$ be an integer.

Let p be a random prime in $[m/2, m)$. Define

$$h_p(x) = x \bmod p.$$

For every $s \in S$, compute $h_p(s)$.

Claim 1. *For any fixed $x, y \in S$ with $x \neq y$*

$$Pr_p[h_p(x) = h_p(y)] \leq O(\log(U)\log(m)/m).$$

Proof. Since $x \neq y$, we have that $(x - y) \neq 0$ and $z = |x - y| \leq 2U$. Thus there are at most $\log(2U)$ primes that divide z. By the prime number theorem, there are $\Theta(m/\log m)$ primes in $[m/2, m)$. So the probability that one of the primes in $[m/2, m)$ divides z is at most $O((\log(U)\log m)/m)$ and

$$Pr_p[h_p(x) = h_p(y)] = Pr[p \text{ divides } z] \leq O(\log(U)\log(m)/m).$$

□

As we chose $d = \Theta(\log U)$ we get that $Pr_p[h_p(x) = h_p(y)] \leq O(d\log(m)/m)$.

As an immediate corollary we get:

Corollary 3.1. *For any fixed $x \in S$*

$$E_p[|\{y \mid y \neq x, h_p(x) = h_p(y)\}|] \leq O(nd\log(m)/m).$$

For a fixed $x \in S$, let's call $B_p(x) = \{y \mid y \neq x, h_p(x) = h_p(y)\}$ the bucket of x. The above statement says that for every $x \in S$, the expected value of $|B_p(x)|$ is $O(nd\log(m)/m)$. By Markov's inequality we also get:

Corollary 3.2. *For any fixed $x \in S$*

$$Pr_p[|B_p(x)| > t] \leq O(nd\log(m)/(mt)) = O(B\log(m)/t).$$

Thus also

$$E_p[\{x \mid |B_p(x)| > t\}] \leq O(nB\log(m)/t).$$

Let's call a bucket $B_p(x)$ "bad" if $|B_p(x)| > t$. From above we get that the expected number of elements in bad buckets is $O(nB\log(m)/t)$. Again by Markov's inequality, we get

Corollary 3.3.

$$Pr_p[|\{x \mid |B_p(x)| > t\}| > Q] \leq O(nB\log(m)/(tQ)).$$

Thus with constant probability, the number of elements $x \in S$ that are in bad buckets is $\tilde{O}(nB/t)$.

In particular, this means that there **exists a prime** $p \in [m/2, m)$ such that the number of elements $x \in S$ in bad buckets for p is $\tilde{O}(nB/t)$.

We can find this p quickly as follows:

Try all $O(m/\log(m))$ primes p in the interval and repeat:

1. Compute $h_p(s) = s \bmod p$ for each $s \in S$ in $\tilde{O}(n)$ time per prime, so $\tilde{O}(mn) = \tilde{O}(dn^2/B)$ time overall.
2. Sort the elements by hash value, again in $\tilde{O}(mn) = \tilde{O}(dn^2/B)$ time overall.
3. Scan the sorted list and count for every $\ell \in \{0, \dots, p-1\}$ the number of elements hashing to ℓ and the number of elements $s \in S$ that hash to values with $> t$ elements hashed to them (the elements in "bad" buckets). If that number is $\tilde{O}(nB/t)$, stop and return p.

By our argument above, after $\tilde{O}(dn^2/B)$ time we will have found a good prime p and a list L of $\tilde{O}(nB/t)$ elements of S that hash to bad buckets.

Handle the elements in bad buckets. Take the list L of $\tilde{O}(nB/t)$ elements of S that hash to bad buckets. For every $s \in L$ and every $s' \in S, s \neq s'$, check whether $s + s' \in S$ or $s - s' \in S$ or $s' - s \in S$ (i.e. whether s, s' are part of a 3SUM* solution. For each s, s' we can perform this check in $O(\log n)$ time, provided S is already sorted, via binary searching.

Thus, in total $\tilde{O}(n^2B/t)$ time we can check whether any of the elements in L are part of a 3SUM* solution. If any of them are, we return YES. Otherwise, remove all of L from S.

Reducing the rest to colorful 3SUM-Convolution. Now, the remaining set S has the property that for every $s \in S$, $|B_p(s)| \le t$.

We will reduce the remaining 3SUM* problem first to Colorful 3SUM Convolution: given arrays a, b, c, determine whether there exist y, z such that $a_y + b_z = c_{y+z}$. Then we will reduce Colorful 3SUM Convolution to 3SUM Convolution.

Via the linearity of $h_p(x)$, we get that if $a + b = c$, then $h_p(a) + h_p(b) = h_p(c)$.

Let's define for $y \in \{0, \dots, p-1\}$, $C(y) = \{s \in S \mid h_p(s) = y\}$. Notice that by our construction, for all y, $|C(y)| \le t$.

It suffices to check whether there exist $y, z \in \{0, 1, \dots, p-1\}$ and $a \in C(y), b \in C(z), c \in C(y+z)$ with $a + b = c$.

Let's define for $y \in \{0, \dots, p-1\}$, $i \in [t]$, $C(y)_i$ to be the ith element in $C(y)$. If $|C(y)| < t$, then let $C(y)_i = C(y)_1$ for all $i > |C(y)|$.

Now, for all t^3 choices of $i, j, k \in [t]$, create three arrays of length p:

- Array a^i with $a^i[y] = C(y)_i$ for $y \in \{0, \dots, p-1\}$,
- Array b^j with $b^j[z] = C(z)_j$ for $z \in \{0, \dots, p-1\}$,
- Array c^k with $c^k[w] = C(w)_k$ for $w \in \{0, \dots, p-1\}$.

The Colorful 3SUM Convolution of a^i, b^j, c^k gives us whether there exist $y, z \in \{0, \dots, p-1\}$ s.t. $a^i[y] + b^j[z] = c^k[y+z]$, or equivalently, $C(y)_i + C(z)_j = C(y+z)_k$.

Technically, we have reduced the problem to whether there is some y, z such that $a^i[y] + b^j[z] = c^k[y + z \mod p]$, however, we can make c^k twice the length where we put two copies of c^k next to each other. Now, if $y + z \ge p$, then in the new c^k, at index $y + z$ we actually have $C(y + z \mod p)_k$. Let's assume we have done this from now on and we will ignore this issue. In particular in the below, a^i, b^j, c^k have the same length (you can think of doubling a^i and b^j as well, or adding some ∞ entries that do not participate in any 3SUM* solutions).

Thus, if there is some choice of (i, j, k) such that $C(y)_i + C(z)_j = C(y+z)_k$, then there is some $a \in C(y), b \in C(z), c \in C(y+z)$ such that $a + b = c$. I.e. we have reduced 3SUM* to t^3 instances of Colorful 3SUM Convolution on $O(m) = O(dn/B)$ length arrays. The reduction time is within polylogs

$$n^2B/t + dn^2/B.$$

If Colorful 3SUM Convolution on N length arrays is in $O(N^{2-\varepsilon})$ time for some $\varepsilon > 0$, via the reduction we can solve 3SUM in time, within polylogs,

$$n^2B/t + dn^2/B + t^3(dn/B)^{2-\varepsilon}.$$

As $d = O(\log U)$, we will omit the dependence on d for now and then multiply the final running time by $\log U$, for simplicity. (One can also minimize the dependence on $\log U$ but it gets messier.)

To minimize the running time, set $n^2B/t = n^2/B$, i.e. $B = \sqrt{t}$, and $n^2\sqrt{t}/t = t^3(n/(\sqrt{t}))^{2-\varepsilon}$. This latter equality is:

$$n^{\varepsilon} = t^{3.5-1+\varepsilon/2}$$

so that we set (by squaring the above)

$$t^{5+\varepsilon} = n^{2\varepsilon},$$

$$t = n^{2\varepsilon/(5+\varepsilon)}.$$

Plugging into the running time we get a running time, within polylogs

$$n^2/\sqrt{t} = n^2/\sqrt{n^{2\varepsilon/(5+\varepsilon)}} = n^{2-\varepsilon/(5+\varepsilon)}.$$

The running time is thus $\tilde{O}(n^{2-\varepsilon'} \log(U))$ for $\varepsilon' = \varepsilon/(5+\varepsilon) > 0$.

Reducing Colorful 3SUM Convolution to 3SUM Convolution. Suppose we are given three arrays a, b, c of length m where we want to know if there are $y, z \in \{0, \ldots, m-1\}$ s.t. $a_y + b_z = c_{y+z}$.

We will create a new single array A of length $8m$ and embed a, b, c into A as follows.

For each $y \in \{0, \ldots, m-1\}$, set $A[8y+1] = a_y$, set $A[8y+3] = b_y$, $A[8y+4] = c_y$. Set all remaining elements of A to ∞ (or some sufficiently large element that cannot participate in a 3SUM* solution).

Suppose that $a_y + b_z = c_{y+z}$. Then $A[8y+1] + A[8z+3] = A[8(y+z)+4]$, a 3SUM-Convolution solution. On the other hand, suppose that $A[8y+s_1] + A[8z+s_2] = A[8w+s_3]$ and $8y+s_1+8z+s_2 = 8w+s_3$, for some $s_1, s_2, s_3 \in \{1, 3, 4\}$ (as all positions of the array $A(t)$ with $t \mod 8 \notin \{1, 3, 4\}$ do not participate in a 3SUM).

Now, $s_1 + s_2 = s_3 \mod 8$ has a unique solution $s_1 = 1, s_2 = 3, s_3 = 4$, and in fact then $s_1 + s_2 = s_3 \mod 8$ is equivalent to $s_1 + s_2 = s_3$. Thus also $8y + 1 + 8z + 3 = 8w + 4$ implies $y + z = w$.

Exercise: Convince yourself of the above statement.

We get, $A[8y+1] + A[8z+3] = A[8(y+z)+4]$ and hence $a_y + b_z = c_{y+z}$, a 3SUM* solution.

Thus if 3SUM Convolution can be solved in $O(m^{2-\varepsilon})$ time on an $O(m)$ length array, then Colorful 3SUM Convolution can also be solved in $O(m^{2-\varepsilon})$ time on $O(m)$ length arrays.