1 3XOR, 3SUM and listing triangles

Given a set $S = \{a_1, a_2, \ldots, a_n\}$ of $n$ integers, where $|a_i| \leq \text{poly}(n)$.

- (3SUM): are there $a_i, a_j, a_k$ so that $a_i + a_j + a_k = 0$?
- (3XOR): replace plus with bit-wise xor.

They can easily be solved in,

- $\tilde{O}(n^3)$: enumerate each tuple.
- $\tilde{O}(n^2)$: enumerate $a_i, a_j$ and binary search $-(a_i + a_j)/a_i + a_j$ in sorted array;

Can we do better than $\tilde{O}(n^2)$? In the special case of $a_i$ length $l \leq (2 - \Omega(1)) \log n$, we can achieve $\tilde{O}(2^l) = \tilde{O}(n^{2-\Omega(1)})$.

- 3SUM: Fast fourier transform (FFT).
  - Let $f(x) = \sum_{i=1}^{n} x^{c+a_i}$ where $c$ is chosen such that $c$ is the smallest integer with $c+a_i \geq 0$. Then caculate the coefficient for $x^{3c}$ in $(f(x))^3$.
  - Since degree of $f(x)$ or $f(x)^2$ is at most $M = O(n^{2-\Omega(1)})$, it takes $O(M \log M) = \tilde{O}(n^{2-\Omega(1)})$ time. to use FFT for polynomial multiplication.

- 3XOR: Fast Walsh-Hadamard transform (FWHT).

Listing triangles: given adjacency list of a graph with $m$ edges and $z$ triangles,

- (List all triangles): the best we can hope $\tilde{O}(m^{1.5})$ (maximal number of triangles = $\theta(m^{1.5})$). It can be achieved as following
  - Algorithm: list the $O(m\sqrt{m})$ triangles going through a node of degree $\leq \sqrt{m}$. then, triangles using nodes of degree $>\sqrt{m}$ only.
  - Analysis: # of nodes of degree $>\sqrt{m} \leq O(\sqrt{m})$, brute force takes $O((\sqrt{m})^3) = O(m^{1.5})$ degree of nodes $\leq \sqrt{m}$, then running time $\leq \sqrt{m} \times$ total degree $= O(m^{1.5})$.

- (List min($z, m$) triangles): conceivably may be $O(m)$. In fact, we only know a $\tilde{O}(m^{1.5-\Omega(1)})$ algorithm, assuming that the exponent of matrix multiplication is $O(m^{1.4})$. 

2 Reduction without hashing

2.1 Reducing 3XOR to C3XOR


We can reduce 3XOR to C3XOR. For an instant $a_1, \ldots, a_n$, let $A[a_i] = a_i$ and untouched $A[i]$ to are set large enough so as to never in a solution. Then $a_i + a_j = a_k \Leftrightarrow A[a_j] + A[a_i] = A[a_k] = A[a_i + a_j]$.

Correct! But the reduction is very slow, since size of $A$ will be very large (poly($n$) instead of $n$).

2.2 Reducing C3XOR to listing triangles

Given $A$, $\exists i, j, A[i] + A[j] = A[i + j] \Leftrightarrow \exists a, b, A[a + b_h] + A[a + b_l] = A[a]$ where $b_h, b_l$ are each half bits in $b$. (solution $i, j \Leftrightarrow$ solution $b = i + j, a = i + j_h$) Then we construct a tripartite graph. First part has $\sqrt{n} \times n$ nodes with form $(b_h, x)$, second part has $n$ nodes with form $(a)$, and third part has $\sqrt{n} \times n$ nodes with form $(b_l, y)$.

(a) If $A[a + b_h] = x$, connect $(b_h, x)$ and $(a)$.
(b) If $A[a + b_l] = y$, connect $(b_l, y)$ and $(a)$.
(c) If $A[b] = x + y$, connect $(b_h, x)$ and $(b_l, y)$.

(a) or (b) has $n\sqrt{n} = n^{1.5}$ edges ($a$ and $b_h/b_l$ determine $x/y$). But (c) has $\sqrt{n}\sqrt{n} = n^2$ edges ($b_l$ and $b_h$ and $x$ determine $y$). Call listing triangles on those edges. Correct! But $n^2$ edges are not good. Even linear time $n^2$ is sufficient to solve C3XOR. The reduction is also very slow.

3 Linear pairwise hashing

Linear pairwise independent hash function $h$, from $\{0,1\}^l$ to $\{0,1\}^r$. We let the construction be $h(x) = Mx$ where $M$ is $l \times r$ matrix over $GF(2)$ and each entry of $M$ to be 1 with probability $1/2$.

- Linearity: $h(0) = 0, h(x + y) = h(x) + h(y)$.
- Pairwise independent: $\Pr[h(x) = h(y)] \leq 1/2^r$ for any $x \neq y$.
- Useful lemma: Denote $R = 2^r$. Let $S$ be a set of $n$ elements. Expected number of elements from $S$ hashing into a bucket with load $\geq 3n/R$ is $\leq R$. (When hashing $n$ elements to $[R]$ buckets, the expected load of each bucket is $n/R$. This lemma says, with constant probability , at most $O(R)$ elements are hashing into large buckets.)
4 Plug in hashing

4.1 Reduce element length of 3XOR

The length $l > 3\log n$ can be reduced into $3\log n$ via hashing. We choose a hash $\{0,1\}^l \to \{0,1\}^{3\log n}$. And consider $S' = \{h(a_1), h(a_2), \ldots, h(a_n)\}$. Solve the new 3XOR problem.

- if $a_i + a_j + a_k = 0$, then $h(a_i) + h(a_j) + h(a_k) = 0$
- otherwise, $h(a_i) + h(a_j) + h(a_k) = 0$ w.p. at most $1/2^r = 1/n^3$. By union bound, algorithm fails w.p at most $(n^3)/n^3 < 1/6$.

4.2 Revisit reducing 3XOR to C3XOR

**Lemma 1.** If C3XOR can be solved in time $n^{2-\Omega(1)}$ with error 1%, so can 3XOR.

Intuition is using linear hash to 1-1 map $a_1, \ldots, a_n$ to range $[n]$ and let $A[h(a_i)] = a_i$. Then $a_i + a_j = a_k \Rightarrow A[h(a_j)] + A[h(a_i)] = A[h(a_k)] = A[h(a_i + a_j)]$. But no such hash exists. The solution is to implement the hash-function based solution, and handle the few collisions separately.

**Proof.** Use $h$ maps $l = O(\log n)$ bits to $r = (1 - \alpha) \log n$ bits for a constant $\alpha$ to be determined. The range $R = 2^r = n^{1-\alpha}$.

- (Handle the few collisions) By lemma 2, at most $R$ elements (the expected number of elements) fall into buckets with load $\geq 3n/R$. $O(n)$ time is sufficient to check each such element so that $\tilde{O}(nR)$ to deal with them with high probability by a Markov bound.

- (Hash-function based solution) Now consider solution $x + y + z = 0$ with $x, y, z$ all be hashed into not overloaded buckets. There are at most $R$ buckets with load $\leq 3n/R$. Enumerate position tuple $(i,j,k)$ of $x, y, z$. Copy any element in position $i$ to $A[h(x)01]$, $j$ to $A[h(x)10]$ and $k$ to $A[h(x)11]$. Solve this C3XOR in $O(R)$ array. If there solution for 3XOR, there will be in C3XOR, also solution for C3XOR is valid for 3XOR.

Time complexity for second case: $(3n/R)^3 R^{2-\Omega(1)} = \tilde{O}(n^3 R^{-1-\Omega(1)}) = \tilde{O}(n^{2+\alpha-\Omega(1)}) = \tilde{O}(n^{2-\Omega(1)})$ when $\alpha$ is sufficiently small. The first part takes $O(n^{2-\alpha})$, so overall $\tilde{O}(n^{2-\Omega(1)})$. □

4.3 Revisit reducing C3XOR to listing triangles

**Lemma 2.** Suppose that given the adjacency list of a graph with $m$ edges and $z$ triangles (and $O(m)$ nodes) one can list $\min(z, m)$ triangles in time $m^{1.33-\Omega(1)}$. Then one can solve C3XOR on a set of size $n$ in time $n^{2-\Omega(1)}$ with error 1%.
Intuition is reducing the $n$ possibility of $A[i]$ via hashing into $\sqrt{n}$ so that (c) part will have only $n^{1.5}$ edges. So we choose $R = \sqrt{n}$, and add edges when $h(A[a + b_i]) = x$ or $h(A[a + b_n]) = y$ or $h(A[b]) = x + y$. But error will happen, $h(A[a + b_i]) + h(A[a + b_i]) = h(A[b])$ when $A[a + b_i] + A[a + b_i] = A[b]$. So we need to bound the error pairs of $(a, b)$.

Proof. We expect expect $\leq n^2/R$ of pairs of $a, b$ which are not solution but $h(A[a + b_i]) + h(A[a + b_i]) = h(A[b])$. So that with constant probability (can be amplified be repetition) by Markov argument, we have $\leq 2n^2/R = 2n^{1.5}$ pairs $a, b$ are not solution ;i.e.,$h(A[a + b_i]) + h(A[a + b_i]) \neq h(A[b])$ but $h(A[a + b_i]) + h(A[a + b_i]) = h(A[b])$.

We do reduction in this case. We construct a tripartite graph by replacing $A[i]$ by its hashing value $h(A[i])$. Call listing triangles with this tripartite graph of $3R\sqrt{n} = 3n^{1.5}$ edges.

Triangles are $1 - 1$ correspondence with pairs $a, b$ satisfying $h(A[a + b_i]) + h(A[a + b_i]) = h(A[b])$. We check all $a, b$ if it corresponds to a solution for C3XOR. $m = 3n^{1.5}$ pairs are sufficient to find a solution for C3XOR because of $\leq 2n^{1.5}$ error pairs. So that if we can solve listing triangles in $n^{4/3-\Omega(1)}$, then we can solve C3XOR in $(n^{1.5})^{4/3-\Omega(1)} = n^{2-\Omega(1)}$ with error probability 1%.