1 3SUM

Definition 1 (3SUM Problem)**.** Given a set A of n integers decide if there exist $x, y, z \in A$ such that $x + y + z = 0$.

Algorithms. The naive algorithm for 3SUM runs in $\mathcal{O}(n^3)$ time. It is easy to come up with a $\mathcal{O}(n^2 \log n)$ time algorithm: Put the input numbers into a balanced BST, then iterate over n^2 pairs $(x, y) \in A \times A$ and search for $-(x + y)$ in the BST. Replacing the BST with a hash table gives randomized $\mathcal{O}(n^2)$ time algorithm.

There is also a simple deterministic $\mathcal{O}(n^2)$ time algorithm. First, we sort the input numbers, $a_1 \leqslant a_2 \leqslant \cdots \leqslant a_n$. Now, for every $k \in [n]$ we will determine in linear time if there exist $i, j \in [n]$ such that $a_i + a_j = -a_k$. We start with $i = 1$, $j = n$. If $a_i + a_j < -a_k$, we know that for every other $j' < j$ we will have $a_i + a_{j'} < -a_k$, therefore i cannot be a part of a pair we are looking for, and we can safely increase i by one. Analogously, if $a_i + a_j > -a_k$, we decrease j by one. We stop when we find $a_i + a_j = -a_k$, or when i becomes greater than j and we know that there is no such pair we were looking for.

The currently fastest algorithm – due to Baran, Demaine, and Pătraşcu (Algorithmica 2008) – runs in time $\mathcal{O}(n^2 \log \log n / \log^2 n)$. Hence the following hypothesis.

Hypothesis (3SUM Hypothesis)**.** *There is no $\mathcal{O}(n^{2-\varepsilon})$ time algorithm for 3SUM, for any $\varepsilon > 0$.*

Variants. Sometimes it is more convenient to consider a variant of 3SUM in which we are looking for a triple with $x + y = z$ instead of $x + y + z = 0$, or in which we are given three sets A, B, C and we are looking for a triple $x \in A, y \in B, z \in C$. We may also insist on the three numbers being distinct or not. All these variants can be reduced to each other, so they have the same time complexity, up to constant factors. We will implicitly switch between the variants when convenient.

2 Hashing

A key tool for reductions from 3SUM, first used by Baran, Demaine, and Pătraşcu, is almost-linear hashing.

Lemma 1. *For every positive integer input size w and output size s, there exists a family of hash functions $H \subseteq [U] \to [R]$, for $U = 2^w$, $R = 2^s$, such that*

1. *for all integers $x, y \in [U]$ and all hash functions $h \in H$*

$$h(x) + h(y) = h(x + y) + \delta, \text{for some} \delta \in \Delta_h,$$

   where Δ_h is a set of constant size;

2. *given an integer $z \in [U]$ and two sets of n integers $A, B \subseteq [U]$ such that there are no $x \in A, y \in B$ with $x + y = z$, the probability, over hash functions h drawn uniformly at random from H, that there exist $x \in A$, $y \in B$ such that $h(x) + h(y) = h(z) + \delta$ for some $\delta \in \Delta_h$ is at most $\mathcal{O}(n^2/R)$;*

3. *given an integer $y \in [U]$ and a set of n integers $A \subseteq [U]$, if $R \leqslant n$, the probability, over hash functions h drawn uniformly at random from H, that "x is in an overfull bucket", i.e., $|\{y \in A \mid h(x) = h(y)\}| > 100n/R$, is at most $1/6$.*

The family they use is $h(x) = \lfloor (ax/2^{w-s}) \rfloor \bmod 2^s$, where a is a random w-bit odd number. The set Δ_h is in that case $\{-1, 0, 1, R-1, R, R+1\}$. A proof that the family has the desired properties can be found, e.g. in Karl Bringmann's slides `https://conferences.mpi-inf.mpg.de/adfocs-18/karl/3-3SUM.pdf`. It is rather straightforward given some standard probabilistic analysis tricks.

As a first application of this tool, observe that, unlike APSP, 3SUM with numbers bounded by n^3 is as hard as the general case (at least for randomized algorithms). Indeed, it is enough to set $R = n^3$, apply a random hash function to all the input numbers, and solve a separate instance of 3SUM for each $\delta \in \Delta_h$. Here it is convenient to think of the tripartite variant so that we can add the offset δ only to candidates for z. Probability of a false positive per element is $\mathcal{O}(1/n)$, so by union bound the total probability of a false positive is at most a constant. There are no false negatives.

3 Convolution 3SUM

In general, it is usually easier to make fine-grained reductions from problems for which the naive algorithms are (conjectured to be) optimal. Now we will see such a problem that is equivalent to 3SUM.

Definition 2 (Convolution 3SUM problem)**.** Given a sequence of n integers $a_1, a_2, \ldots, a_n$, decide if there exist $i, j \in [n]$ such that $a_i + a_j = a_{i+j}$.

Theorem 1. *If 3SUM is in $T(n)$ time, then Convolution 3SUM is in $\mathcal{O}(T(n))$ time.*

Proof. Given an instance of Convolution 3SUM, replace each number a_i with $a_i*(2n+1)+i$ and solve 3SUM on these numbers. Observe that $(a_i * (2n+1) + i) + (a_j * (2n+1) + j) = (a_k * (2n+1) + k)$ if and only if $a_i + a_j = a_k$ and $i + j = k$. □

Theorem 2. *If Convolution 3SUM is in $T(n)$ time, then 3SUM is in randomized $\mathcal{O}(T(n))$ time.*

Proof sketch. The idea is take a random hash function for $R = n$ and put each number $x \in A$ at position $a_{h(x)} = x$. An issue is that there will be multiple numbers hashing to the same value. To deal with that we let each position a_i to become a bucket storing many numbers. If there are more than 100 elements in a bucket, we discard it completely. The probability that we remove that way at least one element of a triple being a solution is at most $3 \cdot 1/6$. Hence the probability of a false negative is at most $1/2$. Now that each bucket contains at most 100 elements, we can solve 100^3 instances of Convolution 3SUM where in each instance we guess the index within a bucket for each element of a triple. □

4 Jumbled Indexing

Definition 3 (Jumbled Indexing problem)**.** Preprocess a text T over alphabet Σ to answer queries asking whether there exists a substring (continuous fragment) of T with frequency counts for each character matching the given query vector $v : \Sigma \to \mathbb{N}$.

There is are two naive algorithms for Jumbled Indexing. One preprocesses all answers in $\mathcal{O}(n^2\Sigma)$ time, allowing $\mathcal{O}(\Sigma)$ time queries afterwards; another one does no preprocessing and takes $\mathcal{O}(n \log \Sigma)$ time per query. It turns out these two algorithms are best possible (up to subpolynomial factors) under 3SUM Hypothesis.

Theorem 3. *If Jumbled Indexing, with alphabet size* $\log n$*, can be solved with* $\mathcal{O}(n^{2-\varepsilon})$ *preprocessing time and* $\mathcal{O}(n^{1-\varepsilon})$ *query time, then Convolution 3SUM can be solved in* $\widetilde{\mathcal{O}}(n^{2-\varepsilon})$ *time.*

For a proof see Section 2 in the paper by Amir et al. available at `https://arxiv.org/abs/1405.0189`.