Prison – Analysis

Idea: Sultan Alaiban and Viktor Kozhuharov, Implementation and Analysis: Emil Indzhev

1 Introduction

We can think of the problem in several ways. One of them is the one from the statement – i.e. encoding and decoding. Another is to think about generating a list of triples such that no two triples have a pair in common. We can simply generate all of these in the precompute and then just return cached values in encode. Similarly we can cache the decodings of all pairs and return using that in decode.

2 Upper bound for N

Let's start by trying to figure out an upper bound for N, as a function of M. The decoder gets an (unordered) pair and there are $M + M(M-1)/2$ possible pairs. However, each triple maps to multiple pairs. If the triple has 3 distinct numbers, it maps to 3 possible pairs. If it has the same number 3 times, it maps to only 1 pair. It is bad to have triples with 2 equal numbers and 1 different, since those are strictly worse than the triple with the duplicate number being repeated 3 times. Therefore, the M equal number pairs can map to distinct As and the other $M(M-1)/2$ pairs must be in groups of 3. In total, we get $N \leq M + M(M-1)/6$ (roughly $M^2/6$). We can see that the targets are very close to this.

3 Naive ideas

Before looking at some better solutions (ones with $N = O(M^2)$), we can briefly talk about some simpler ideas. Firstly, we can just do all triples of equal numbers ($N = M$). This gets 10.69 points.

A better idea is to also have triples of the form $(i, i+1, i+2)$ for all i-s that are multiples of 3. This can then be extended with $(i, i+3, i+6)$, for all i-s which when divided by 3 (rounded down) are multiples of 3. Then follows $(i, i+7, i+14)$ and so on with the offset always being double the previous offset plus 1. This ends up with $N = O(M \log M)$. This gets 23.07 points.

4 Bucket solutions

4.1 3 buckets

The problem would be a lot easier if we knew which number is which when decoding. So let's split the M numbers up into 3 buckets and the first number is always from the first bucket, the second from the second and the third from the third. That way the triple effectively becomes ordered and when decoding we know the order of the 2 numbers and also which of 3 numbers is missing.

Now, there are many ways to encode A. Easiest is to encode it using the first 2 numbers and have the third number act as a sort of checksum to recover the missing number, in case one of the first 2 is missing. An alternative, which is slightly worse, is to use the Chinese remainder theorem and have the three buckets have co-prime sizes, encoding A as its remainder modulo the 3 bucket sizes (big enough that the product of any 2 of them is $\geq N$).

Now onto the proposed, easier way: We can encode A with two numbers in base $\sqrt{N}$ using the first 2 numbers. Alternatively, we can say we just iterate through all (ordered) pairs over the first 2 numbers and each of those uniquely defines a triple. Then, as explained, we only need to figure out a scheme for the last number, so that using it and whichever of the first 2 numbers, we can recover the other one.

One easy idea is to use the XOR of the first 2 numbers. However, this needs the last bucket to be a power of 2 (note that the first 2 buckets don't need to be a full power of 2). Another idea is to just use the sum of the first 2 numbers, but this needs the last bucket to be twice as big. The simplest best thing to do is to make the last number be the sum of the first 2 modulo their bucket size (which makes all 3 buckets have the same size).

This solution generates $M + (M/3)^2$ triples (around $M^2/9$) and gets 58.96 points.

4.2 2 buckets

The issue for all of these is that we don't use any pairs where 2 numbers come from the same bucket. One improvement is to instead only have 2 buckets, the first two numbers being from the first bucket and encoding A as an unordered pair (instead of an ordered one) and the third number (from the second bucket) working the same way as before. Alternatively, we can say we iterate through all unordered pairs from the first bucket and compute the third number for each.

This solution generates $M + (M/2) * (M/2 - 1)/2$ triples (roughly $M^2/8$) and gets 67.2 points.

Regardless, the main issue remains. Here, we don't have any pairs where both numbers are from the second bucket.

4.3 Divide and conquer

We can make a clever observation which works on top of either of the above solutions. Since the only pairs we're not using are the ones where the 2 numbers are from the same bucket, they can be generated from triples where all 3 numbers are from that same bucket. In fact these are the only types of triples left available to us, since any other triples would overlap (in pairs) with already existing triples. Then, notice that generating triples where all 3 numbers are from the same bucket is the exact same problem as the full one. Thus, we just apply a standard divide and conquer approach and run our solution on the bucket (in the case of the 3 bucket solution – on each; in the case of the 2 bucket one – on the second one).

We can see that these solutions fully cover all pairs (up to rounding) and therefore get over 99 points. Find an optimized implementation of a recursive solution based on the 2 buckets one in: `emil_rec_best.cpp`

5 Explicit constructions

The above solutions are derivable if we start with a bucket based solution. However, there are also some direct constructions that work. Some would work best only for some specific types of values of M, but can be partially adapted to other ones.

In general, constructions can be produced by using any symmetric associative invertible operation and choosing the target result of applying the operation on all 3 numbers. This is because we can infer the missing number from any 2 other ones, without knowing their order (or which 2 they are).

One idea is to use XOR and use a target value of 0. Here the missing number is always the XOR of the other two numbers. We can construct the triples by iterating over all pairs and computing the third number (and then enforcing that the numbers are non-increasing/non-decreasing). Naively, this works only for M being a power of 2. A partial fix is to use numbers up to the largest power of $2 \leq M$. Better, we can use the smallest power of $2 \geq M$ and keep only the triples where all 3 numbers are less than M.

The very small issue with this is that it produces triples of the form $(X, X, 0)$, which is wasteful. We can substitute them with (X, X, X) triples, but this means we don't use 0 anymore (except for $(0, 0, 0)$). Therefore, we can pretend to add 1 to all numbers for the purposes of XORing and generate only the triples with all 3 numbers being distinct. For this solution, the best values are actually 1 less than a power of 2, but we can use the same trick (i.e. choosing a larger power and keeping only the valid triples). This solution gets 78.3 points. You can find an implementation in: `emil_constr_xor_impr.cpp`

Another idea is to use a similar to XOR operation on the base 3 representation of the numbers. The operation is per digit sum modulo 3. This being 0 means that either all 3 digits are the same or all 3 are different. These triples are actually exactly the same as the sets in the popular game Set. The approach for the solution is exactly the same as before, except we need to compute the third number digit by digit (if the first 2 number's digits match, we use the same, otherwise we use the missing digit). The (small) advantage of this operation is that that it naturally produces the triples of all 3 equal numbers. However, its best values are powers of 3, which are rarer than powers of 2. The solution gets 59.85 points. You can find an implementation in: `emil_constr_set.cpp`

Finally, we can do a similar thing to what we did with out buckets based solutions. Just use the operation sum modulo M. Then, we take all triples which sum up to 0. Notice that we can compute the third number as the M minus the first 2 (modulo M). The small issue is again the same as with the XOR solution – we have triples of the form (X, X, Y). There is no easy fix like with XOR, so this solution has about $N = 2M/3 + M(M-1)/6$. However, it works well on all values of M, so this is the model solution and thus gets 100 points. You can find it here: `emil_constr_modsum.cpp`

6 Greedy solution

One alternative type of solution that gets 77.91 points is a greedy one. Simply iterate over all triples of numbers in some order (say lexicographic) and, if the current triple is valid (i.e. no previous triple has any of its pairs), generate/use that triple. The issue is that this is too slow so needs to be optimized by investigating the pattern in the triples this outputs. It can be fully worked out and computed in $O(N)$ (i.e. $O(M^2)$) time (though this is fairly difficult). Find an implementation of this solution in: `encho_greedy_optimized.cpp`