Two Way Tasks

Mamnoon Siam

Draft May 30, 2021

1 Strategies

- Work with relaxed constraints (can't stress enough how important it is).
- Try to plug in old ideas.
- Binary representation (different number means there's one bit in which they are off).
- Xor of everything.
- Sum modulo something.
- Group slots together (monkes strong together).
- Relabel everything.
- Hashing/randomized mapping.
- Split elements into modulo classes (e.g. odd-even).

2 Problems

Problem 2.1 (JOISC'17 Broken Device)**.**

Encoder: Encode the non-negative integer $X(\leqslant 10^{18})$ in the $N(=150)$ length binary array A. $K(\leqslant 40)$ indices $P_0, P_1, \ldots, P_{K-1}$ from this array are broken i.e. no matter what the encoder puts in those indices, it will reset to 0. The encoder will be supplied with N, K, P, X.

Decoder: Decode X. The decoder will only be supplied with N and A.

Subtasks	$K \leqslant 1$	$K \leqslant 15$	$K \leqslant 20$	$K \leqslant 24$	$K \leqslant 37$
Points	8	41	51	59	85

Solution. Follow the first strategy – work with relaxed constraints.
Idea for $K = 1$: Encode $A[2i] = A[2i+1] = X_i$. We can decode $X_i = A[2i] \vee A[2i+1]$.
Idea for $K = 2$: Can get away with the previous encoding? Kinda. It will be problematic if there's some i such that $P_0 = 2i$ and $P_1 = 2i+1$. One fix for this is sending binary representation of such i at the end of A – put i in $A[120 \ldots 120 + \lg(i)]$. Another fix is just skipping $2i$ and $2i+1$. But how do we distinguish a pair of skipped indices from an actual $X_i = 0$? Now, here's another idea – send `01` or `10` if we want to send an actual $X_i = 0$, and send `00` if it's actually a skip. But wait, say $X_i = 1$ and either $2i \in P$ or $2i+1 \in P$, then the

decoder will get `01` or `10`, which we defined as an actual $X_i = 0$. To settle all these fuss, we can just skip $2i$ and $2i + 1$ altogether if either one of these indices are in P – put `00`. Now, we lost $2K$ indices, but other indices are intact. You can see that this strategy will work for $K \leqslant \frac{150-2\lg(X)}{2}$.
Okay, now grouping together two adjacent indices to send one bit of information gives us the following mapping:

$$\begin{array}{lcl} \texttt{00} & \rightarrow & \text{skip} \\ \texttt{01} & \rightarrow & X_i = 0 \\ \texttt{10} & \rightarrow & X_i = 0 \\ \texttt{11} & \rightarrow & X_i = 1 \end{array}$$

Mapping both `01` and `10` to $X_i = 0$ seems like a waste. Write X in base 3, another strategy – be open to other bases instead of only considering the binary representation. Most of the times the base will come naturally, but if it doesn't, you have to hardthink these old ideas. If $K \leqslant 37$, we lose at most 37 pairs out of the initial 75. That leaves us with 38 usable pairs, and we can represent any number in $[0, 3^{38})$. As $3^{38} \approx 1.35 \times 10^{18}$, this gives us 85 points.
What about grouping three consecutive indices together and finding some clever mapping like before? Because, one rule of thumb for communication problems is that the more you group together, the more efficiently you are able to send the information, with the obvious drawback that it gets hardar and harder to find mapping, as you make the group size bigger.
As we are expanding the group size, we should be able to send more information per group. So the amount of bits we are going to send using each group should depend on the number of broken indices in $\{3i, 3i + 1, 3i + 2\}$ – more broken indices means less info, less broken means more info. Indeed, if the three indices are intact, we can send 2 bits of information, 1 bit of information if 1 bit is broken, and totally skip if more than 1 are broken.
How to find such a mapping? I tried to find one by hand, e.g. if all bits are intact, send `1XX`, where `XX` is the two bits... but eventually I got stuck in every possible way. So we have to find this mapping by bruteforce. How do we model this as a bipartite graph (because we need some kind of bijection)? At first let's define $b' = f(b, S)$ where b and b' are some fixed length binary words (in this case the length is 3). $b'_i = 0$ if either $i \in S$ or $b_i = 0$, and $b'_i = 1$ otherwise. In words, we just reset b's broken bits from S by applying f.
The left part of the bipartite graph consists of four sets of nodes A_0, A_1, A_2, A_3. A_i means that we want to encode an integer that is i. Each of the nodes in the sets A_i will correspond to a subset S of $\{0, 1, 2\}$ meaning that the j ($j \in S$)-th bit of current block is broken. Let's describe them:

$$\begin{aligned} A_0 &= \{\{0\}_0, \{1\}_0, \{2\}_0, \varnothing_0\} \\ A_1 &= \{\{0\}_1, \{1\}_1, \{2\}_1, \varnothing_1\} \\ A_2 &= \{\varnothing_2\} \\ A_3 &= \{\varnothing_3\} \end{aligned}$$

Sets have subsript at the end to denote in which A_i they belong to. In total there are 10 nodes in A. What about the right B side of the bipartite graph?

Each of the encoding will be translated into a 3 bit binary word. Those are `000`, `001`, ..., `111`. But `000` will solely be reserved to express a "skip". So $B = \{\texttt{001}, \texttt{010}, \texttt{011}, \texttt{100}, \texttt{101}, \texttt{110}, \texttt{111}\}$.
Now, how do we add edges? For each S_i, where $S \in \{\{0\}, \{1\}, \{2\}, \varnothing\}$ and $0 \leqslant i \leqslant 3$ (that is, $S_i \in A$), and $b \in B$, we will add an edge between them *iff* there exists some length 3 binary word b' such that $f(b', S) = b$.
At this point, we just need to find a perfect matching, but our version of perfect matching is a bit different.

> Define left side of bipartite graph to be $A = \bigcup_{i=1}^{k} A_i$, where $A_i \cap A_j = \varnothing$ for $i \neq j$, and right side to be B. Our definition of perfect matching of this type of bipartite graph is:
>
> 1. Every node $a \in A$ must be matched to exactly one node $b \in B$.
> 2. A node $b \in B$ can be unmatched, matched to exactly one $a \in A$ or matched to multiple $a \in A$, with an extra restriction that b can be matched to nodes that belong to the same A_i for some i.
>
> Find necessary and sufficient condition for a bipartite graph to have perfect matching.

We can find the matching in our small graph by plain bruteforce. □

Solution. Another idea is hashing/randomized mapping. Let's create a random sequence of 60 bit integers of length 150 $(r_1, r_2, \ldots, r_{150})$ and hardcode this into both the encoder and the decoder. Now, these 150 bit vectors' span has very very very high probability to cover $[2]^{60}$. Write X as a linear combination (under modulo 2, that is, xor) of r_i and pass that to P. But if some indices are broken, they will be 0 regardless. Then again, the encoder knows the broken indices, and can simply ignore those r_is that are broken and try to write X as the linear combination of the rest of 110 numbers. In fact, even 110 numbers have very very high possibility of covering the whole $[2]^{60}$ space. □

Problem 2.2 (CEOI 2014 Question)**.** There are two questions x and y. For simplicity, we can think of them as intergers between 1 and 920 (their hash value mod 920 + 1, okay?). The answer to x and y is yes and no respectively. Anna knows the values of x and y, but bob doesn't. Bob will attend an interview in which he will be asked one of these two questions. Anna can communicate with Bob beforehand by sending an integer $1 \leqslant h$. How efficiently can she do it? I.e. the smaller h is, the better score you get.

h	$\geqslant 21$	20	19	18	17	16	15	14	13	$\leqslant 12$
Total points	0	27	30	33	37	42	50	60	75	100

Solution. $h \leqslant 20$ is easy, just send the index of $\texttt{msb}(x \oplus y)$ and value of that bit in x. The only reason 10 becomes 20 is that we have to send the information of one of the bits of either x or y. Only if we were told that $x < y$ or something like that, we'd be done by now! Because, $x < y$ implies $\texttt{msb}(x \oplus y)$-th bit of x is 0. This wishful thinking inspires us to look for some i such that not only x and y differ in their i-th bit, but also x's i-th bit is 0, then we can just send this i.

Obviously this will not be possible always. But what happens when there's no valid i? Well, first of all, $x \supset y$. Although it turns out that this doesn't directly help in this problem, it tells us that x's digit sum d_x is strictly less than y's digit sum d_y. Note that $d_y \leqslant 10$ and $\lfloor \log_2 y \rfloor + 1 \leqslant 4$. So, in this case we can send $10 + \mathtt{msb}(d_x \oplus d_y)$.
We can achieve 60 points with these scrap techniques, not more, unfortunately. For 100 points, we need some heavy stuff.
Let K be the maximum number h that A is allowed to shout over to B. Assume A and B aggreed on some sets $M_i \subseteq \{1, \ldots, K\}$ for $1 \leq i \leq N$. Let B answer "yes" *iff* $h \in M_q$. Then, A can simply shout any number $h \in M_x \backslash M_y$ over to B, provided that M_x is not a subset of M_y. Hence, this strategy works out if no set M_i is a subset of any other set M_j. We can simply choose M_i to be pairwise distinct $\lfloor K/2 \rfloor$ -element subsets of $\{1, \ldots, K\}$.
Remark. This choice of the M_i is optimal, i.e., the problem is solvable *iff* $\binom{K}{\lfloor K/2 \rfloor} \geq N$ (cf. Sperner's theorem). □

Problem 2.3 (Russia)**.** Arutyun and Amayak perform a magic trick as follows. A spectator writes down on a board a sequence of N (decimal) digits. Amayak covers two adjacent digits by a black disc. Then Arutyun comes and says both closed digits (and their order). For which minimal N can this trick always work? NOTE: Arutyun and Amayak have a strategy determined beforehand.

Solution. (by [1]) 101 digits will be enough – that allows 100 different positions of two adjacent digits, which allows Amayak to communicate the sums of odd-position and even-position digits modulo 10. Let s_0 be the sum of even indices mod 10, and s_1 be that of odd indices, and let $p = s_0 \cdot 10 + s_1$. Amayak should cover p and $p + 1$-th digits. On the other hand, 100 digits (or less) is not enough. There are only 99 choices for which two digits to cover, so only $99 \cdot 10^{98}$ different states of the blackboard for Arutyun to observe when he returns, which is fewer than the 10^{100} sequences the spectator has to choose between. □

References

[1] hmakholm left over Monica (https://math.stackexchange.com/users/14366). Algorithm to uniquely determine a number using two adjacent digits. Mathematics Stack Exchange. URL https://math.stackexchange.com/q/1439470. URL:https://math.stackexchange.com/q/1439470 (version: 2015-09-17).