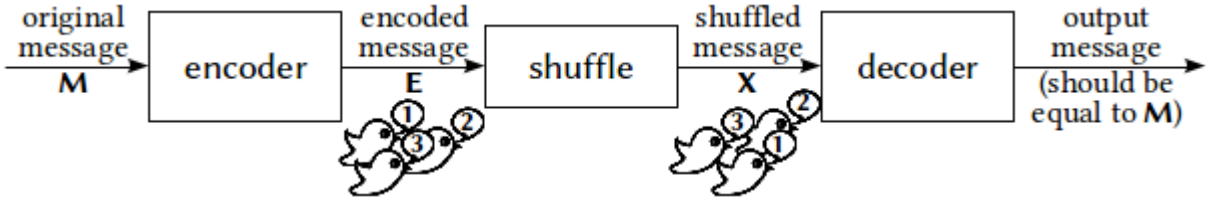
Task: Parrots

Proposed by: Jittat Fakcharoenphol

For this task, contestants are given a message M of length N, which consists of a sequence of integers between 0 and 255, inclusive, and they are asked to find an encoding and decoding scheme such that the encoded message X are invariant up to permutations. The encoded message should be as concise as possible. The scores will be rewarded based on the *ratio* of the length of the encoded message X to the length of the original message M.



There are many ways to solve this task. It will be instructive to look at the first few subtasks.

1 Subtask 1

For this subtask, recall that the original message M is just a sequence of two possible numbers, 0 or 1. Assuming that the original message is indexed from 0 to $N-1$, one can instead choose the send the positions in the original message where all number 1's are located. This technique uses at most N integers to send the message.

2 Subtask 2

For subtask 2, since the numbers in the original message M could be from 0 to 255, they can be encoded using only 8 bits. However, this subtask allows one to use up to 16 bit per number in the encoded message E. Hence, you can encode the position of each number using the higher 8 bits (similar to the solution to subtask 1), and use the lower 8 bits to encode the actual number in the original message M, i.e.

$$E_i = 256 \cdot i + M_i$$

where E_i denotes the i^{th} number in the original message.

Note that the encoded numbers are also ordered, i.e. $E_i < E_{i+1}$, for $0 \leq i < N-1$. Therefore, when we would like to decode the message from the shuffled array X, we can sort it to get E. Then, to obtain the original message from the sorted E should be pretty obvious.

This method also uses at most N parrots, which is satisfactory for this subtask.

3 Subtask 3

Thinking about the first two subtasks should get us familiar with the approach. There are many roads to take from here. However, to formalize the settings, we shall think of a general representation first.

Because we shall keep integers in the encoded array, a natural representation would be to use a sorted sequence. Note that this representation is very robust against permutation; provided that the encoded array E is sorted, by sorting a shuffled array X, we can easily recover the encoded array E, as in subtask 2.

We shall try to extend the approach in subtask 2. However, for subtasks 3–5, we do not have "extra" bits for keeping the position data. In these cases, we look at the original message as a sequence of binary digits (or bits), 0 and 1, by representing each number using only 8 bits. Thus, the original message of length N will have $8N$ bits.

This leads to a way to solve subtask 3. Note that the origianl message M contains at most $16 \times 8 = 128$ bits. For now, let B_i be the i^{th} bit of the original message, indexing from 0 to $8N - 1$. In this case, it is sufficient to take 7 bits to represent all positions of every single bit B_i, and use one last bit to represent the actual data. Hence, the encoding scheme looks like the following

$$E_i = 2 \cdot i + B_i$$

therefore, it takes 7 bits to represent posible positions and another bit to keep the data. That is, for each bit b_i, we shall encode it as $2 \cdot i + b_i$. The length of the encoded message for each test case of this subtask is $8N$ bytes.

4 Subtask 4

For this subtask, there are $32 \times 8 = 256$ bit positions; and it requires 8 bits just to represent the positions. Hence, there is no way to include the information of each bit of the original message in the encoding. If we recall the technique we use to solve subtask 1, we can improve the technique we used in subtask 3 by choosing to encode the positions of all 1 bits. This approach again sends at most $8N$ bytes.

5 Subtask 5

For this subtask, contestants are allowed to obtain some partial scores, depending on the ratio of the encoding message to the original message. In our point of view, to obtain *nearly perfect* scores like 98 is relatively much easier than to obtain the full points. We consider the last 2 points to be *the reward* to those who choose to implement more sophisticated method of encoding that achieve better results than our expectations.

We discuss many potential solutions that might achieve some partial scores.

5.1 A *ratio=12* approach

We are going to generalize the solution to the previous subtask case since the number of bits is more than 256. In this case, there are at most 512 bit positions.

To deal with this issue, we partition the message into 256 pairs of bits. We define the i^{th} bit pair P_i to be (M_{2i}, M_{2i+1}). Again, the bit pair P should be indexed from 0 to $4N-1$.

Note that there are 4 possible values for each bit pair, which can be represented with integers from 0 to 3. One possible method is to send the positions of the bit pair (which uses 8 bit to encode) in the encoding data, and then the actual bit pair data will be encoded as the *number of occurrences* of such positions.

Using this approach, for each pair of bits, we may have to send at most 3 bytes. Therefore, we send at most $\frac{3}{2} \cdot 8N = 12N$ bytes. Contestants will achieve 7 points on this subtask for implementing this method.

5.2 A *ratio=6.25* approach

With a simple observation, we can reduce the size of the encoded messages by roughly a factor of two. Note that the best case (in terms of encoding length) for the previous encoding method is when the original message only contains zero bits, and the worst case is when the message contains only one bits. We shall try to get the average of these two cases.

Consider two encoding scheme, called ONE and ZERO. For a pair of bits (b_{2i}, b_{2i+1}), the ONE scheme sends $2 \cdot b_{2i} + b_{2i+1}$ copies of i, but the ZERO scheme sends $3 - (2 \cdot b_{2i} + b_{2i+1})$ copies. Note that both schemes work, provided that the decoder knows which of the schemes the encoder actually uses.

For each pair of bits, the sum of the number of encoded bytes used by both scheme is always $2 \cdot b_{2i} + b_{2i+1} + 3 - (2 \cdot b_{2i} + b_{2i+1}) = 3$ bytes. Considering all pairs, the sum is $3 \cdot \frac{8N}{2} = 12N$. Now, if we choose the scheme with the lower number of encoding data, we will need at most half of this number, i.e., at most $6N$ bytes.

There are many ways to signal the decoder which encoding scheme the encoder uses without too much penalty on the ratio. For example, we can add 4 copies of 255 iff the ZERO scheme is used. This will make the ratio goes up by $\frac{4}{N}$. However, since $N \geq 16$, the worst ratio is at most 6.25. Contestants will achieve 17 points on this subtask for implementing this method.

5.3 Better approaches

There are many other approaches that give better compression ratio.

5.3.1 One byte to 4 numbers

Consider the number of encoded messages of length *no greater than* 7, only consisting of numbers 0, 1, 2, and 3 (0, 0, 1, 1, 1, 1, 3 is one such message). To analyze this, it might be easier to consider an equivalent scheme: encoded messages of length *exactly* 7, only consisting of numbers 0, 1, 2, 3, and BLANK. There are $\binom{4+7}{4} = \binom{11}{4} = 330$ of them, which is enough to encode one byte of the original message.

Since the original message is at most 64 bytes long, it is possible to allocate 4 numbers per byte (0, 1, 2, and 3 for the first byte, for example), and therefore numbers from 0 to 255 are sufficient. This scheme yields a compression ratio of at most 7.

5.3.2 Optimal ratio

Theoretically, one can encode 64 bytes of data to the minimum of 261 bytes of encoded message without loss of information. There are $\binom{256+261}{256} = \binom{517}{256} \approx 1.47 \times 10^{154}$ different encoded messages using no more than 261 bytes. This is just more than $256^{64} \approx 1.34 \times 10^{154}$, the number of different 64-byte original messages. The compression ratio achieved is about 4.08. For smaller data, the ratio is even smaller.

This method yields a perfect score for this task. However, implementation of this scheme is comparatively difficult.