--- Day 10: Knot Hash ---

You come across some programs that are trying to implement a software emulation of a hash based on knot-tying. The hash these programs are implementing isn't very strong, but you decide to help them anyway. You make a mental note to remind the Elves later not to invent their own cryptographic functions.

This hash function simulates tying a knot in a circle of string with 256 marks on it. Based on the input to be hashed, the function repeatedly selects a span of string, brings the ends together, and gives the span a half-twist to reverse the order of the marks within it. After doing this many times, the order of the marks is used to build the resulting hash.

```
  4--5   pinch   4  5           4   1
 /    \  5,0,1  / \/ \  twist  / \ / \
3      0  -->  3      0  -->  3   X   0
 \    /         \ /\ /         \ / \ /
  2--1           2  1           2   5
```

To achieve this, begin with a list of numbers from 0 to 255, a current position which begins at 0 (the first element in the list), a skip size (which starts at 0), and a sequence of lengths (your puzzle input). Then, for each length:

- Reverse the order of that length of elements in the list, starting with the element at the current position.
- Move the current position forward by that length plus the skip size.
- Increase the skip size by one.

The list is circular; if the current position and the length try to reverse elements beyond the end of the list, the operation reverses using as many extra elements as it needs from the front of the list. If the current position moves past the end of the list, it wraps around to the front. Lengths larger than the size of the list are invalid.

Here's an example using a smaller list:

Suppose we instead only had a circular list containing five elements, 0, 1, 2, 3, 4, and were given input lengths of 3, 4, 1, 5.

- The list begins as [0] 1 2 3 4 (where square brackets indicate the current position).
- The first length, 3, selects ([0] 1 2) 3 4 (where parentheses indicate the sublist to be reversed).
- After reversing that section (0 1 2 into 2 1 0), we get ([2] 1 0) 3 4.
- Then, the current position moves forward by the length, 3, plus the skip size, 0: 2 1 0 [3] 4. Finally, the skip size increases to 1.

- The second length, 4, selects a section which wraps: 2 1) 0 ([3] 4.
- The sublist 3 4 2 1 is reversed to form 1 2 4 3: 4 3) 0 ([1] 2.
- The current position moves forward by the length plus the skip size, a total of 5, causing it not to move because it wraps around: 4 3 0 [1] 2. The skip size increases to 2.

- The third length, 1, selects a sublist of a single element, and so reversing it has no effect.
- The current position moves forward by the length (1) plus the skip size (2): 4 [3] 0 1 2. The skip size increases to 3.

- The fourth length, 5, selects every element starting with the second: 4) ([3] 0 1 2. Reversing this sublist (3 0 1 2 4 into 4 2 1 0 3)

produces: `3) ([4] 2 1 0)`.
- Finally, the current position moves forward by `8`: `3 4 2 1 [0]`. The skip size increases to `4`.

In this example, the first two numbers in the list end up being `3` and `4`; to check the process, you can multiply them together to produce `12`.

However, you should instead use the standard list size of `256` (with values `0` to `255`) and the sequence of lengths in your puzzle input. Once this process is complete, what is the result of multiplying the first two numbers in the list?

Your puzzle answer was `212`.

--- Part Two ---

The logic you've constructed forms a single round of the Knot Hash algorithm; running the full thing requires many of these rounds. Some input and output processing is also required.

First, from now on, your input should be taken not as a list of numbers, but as a string of bytes instead. Unless otherwise specified, convert characters to bytes using their ASCII codes. This will allow you to handle arbitrary ASCII strings, and it also ensures that your input lengths are never larger than `255`. For example, if you are given `1,2,3`, you should convert it to the ASCII codes for each character: `49,44,50,44,51`.

Once you have determined the sequence of lengths to use, add the following lengths to the end of the sequence: `17, 31, 73, 47, 23`. For example, if you are given `1,2,3`, your final sequence of lengths should be `49,44,50,44,51,17,31,73,47,23` (the ASCII codes from the input string combined with the standard length suffix values).

Second, instead of merely running one round like you did above, run a total of `64` rounds, using the same length sequence in each round. The current position and skip size should be preserved between rounds. For example, if the previous example was your first round, you would start your second round with the same length sequence (`3, 4, 1, 5, 17, 31, 73, 47, 23`, now assuming they came from ASCII codes and include the suffix), but start with the previous round's current position (`4`) and skip size (`4`).

Once the rounds are complete, you will be left with the numbers from `0` to `255` in some order, called the sparse hash. Your next task is to reduce these to a list of only `16` numbers called the dense hash. To do this, use numeric bitwise XOR to combine each consecutive block of `16` numbers in the sparse hash (there are `16` such blocks in a list of `256` numbers). So, the first element in the dense hash is the first sixteen elements of the sparse hash XOR'd together, the second element in the dense hash is the second sixteen elements of the sparse hash XOR'd together, etc.

For example, if the first sixteen elements of your sparse hash are as shown below, and the XOR operator is `^`, you would calculate the first output number like this:

```
65 ^ 27 ^ 9 ^ 1 ^ 4 ^ 3 ^ 40 ^ 50 ^ 91 ^ 7 ^ 6 ^ 0 ^ 2 ^ 5 ^ 68 ^ 22 = 64
```

Perform this operation on each of the sixteen blocks of sixteen numbers in your sparse hash to determine the sixteen numbers in your dense hash.

Finally, the standard way to represent a Knot Hash is as a single hexadecimal string; the final output is the dense hash in hexadecimal notation. Because each number in your dense hash will be between `0` and `255` (inclusive), always represent each number as two hexadecimal digits (including a leading zero as necessary). So, if your first three numbers

are `64, 7, 255`, they correspond to the hexadecimal numbers `40, 07, ff`, and so the first six characters of the hash would be `4007ff`. Because every Knot Hash is sixteen such numbers, the hexadecimal representation is always `32` hexadecimal digits (`0`-`f`) long.

Here are some example hashes:

- The empty string becomes `a2582a3a0e66e6e86e3812dcb672a272`.
- `AoC 2017` becomes `33efeb34ea91902bb2f59c9920caa6cd`.
- `1,2,3` becomes `3efbe78a8d82f29979031a4aa0b16a9d`.
- `1,2,4` becomes `63960835bcdc130f0b66d7ff4f6a5a8e`.

Treating your puzzle input as a string of ASCII characters, what is the Knot Hash of your puzzle input? Ignore any leading or trailing whitespace you might encounter.

Your puzzle answer was `96de9657665675b51cd03f0b3528ba26`.

Both parts of this puzzle are complete! They provide two gold stars: **

At this point, you should return to your advent calendar and try another puzzle.

If you still want to see it, you can get your puzzle input.

You can also [Share] this puzzle.