Problem 3:
Making the title for this programming problem was a lot of work
13+8=21 Points

Problem ID: `haiku`
Rank: `2+3`

Introduction

Problem Summary

Given $\mathbf{N}$ words and their syllable count, construct a haiku: a three line poem with words containing 5 syllables in total on line one, 7 on line two, and 5 on line three.

All words are unique and can only be used once. If it's impossible to construct a haiku with the words given, output `IMPOSSIBLE` on all three lines instead.

You can find the original problem statement here:
https://calico.berkeley.edu/files/calsuco-22/contest/haiku/haiku.pdf

Big Ideas

The main test set can be solved by trying every possible haiku in factorial or exponential time. However, the bonus required the use of dynamic programming to optimize the exponential time solution down to linear, or we can take a completely different approach by examining integer partitions.

The problem introduction features Anya and Damian from the anime Spy x Family which recently finished airing! You can also legally stream it on Crunchyroll.

Main Test Set

We only have 9 words for this test set. This makes multiple brute force approaches viable.

Poetry Recital *Solution available in Python*

When reciting a poem to a friend, you basically just say all of the words in order out loud! We can take inspiration from this simple fact and have our program do something similar.

We can enumerate through every permutation of all words, and for each permutation, go through the words in order and see if the word breaks align with the syllable breaks at 5, 12 (5 + 7), and 17 (5 + 7 + 5) syllables. Python's `itertools.permutations` and C++'s `std::next_permutation` provide easy and efficient ways to do this.

If we find a permutation that works, we can use the word breaks to construct our haiku. If not, then there must be no way to construct a haiku, so we conclude it's impossible.

As we need to consider $\mathbf{N}!$ permutations and need $O(N)$ time to check each permutation, our overall runtime is $O(N! \cdot N)$.

Chomsky Who? *Solutions available in C++, Python*

Since we don't care about grammar, the order in which words occur on each line doesn't matter at all! We can use this observation to speed up our program. This wasn't necessary to pass the main test set, but it helps us see an even bigger optimization in the next section.

Whereas before each word has $\mathbf{N}$ different positions to be in within each permutation, observe that only 4 of these are relevant: in the first line, in the second line, in the third line, or in no line.

Implementing this solution can be done easily by looping up to `1 << (2 * N)` and using a bitmask. If you're using Python, `itertools.product` can also help with implementation. If you really want to, you can also write an octuple nested for loop.

For each word, we need to consider which of the 4 options to assign it. Since we need to do this for all words, we need to check $4^{\mathbf{N}}$ assignments in total. For each assignment, we can do constant time arithmetic to verify if the number of syllables on each line is correct. Thus, our overall runtime is $O(4^N)$.

Bonus Test Set

With 1000 words, exponential time is out of the question. We need something much faster.

The Reduction is Left as an Exercise to the Reader *Solution available in Python*

Some of you may have noticed that this problem is quite similar to the subset sum problem (SSP), which is a well-known NP-complete decision problem in computer science. It's not too difficult to see that since our syllable totals per line are small, we might be able to take some inspiration from the dynamic programming solution to SSP.

The difference, however, is that we need to consider additional states in order to represent the different number of syllables in each line. One way to do this is to represent states as quadruplets in the form of (i, a, b, c) where i is the index of the current word being considered, a is the number of syllables already in the first line, b is the syllables in the second, and c is the syllables in the third. For each state, we can either add word i to any of the lines or skip it.

If some state $(i, 5, 7, 5)$ is achieved, then we know it must be possible to construct a haiku. We can follow the transitions to recover which words were assigned to each line to construct our final answer. If $(i, 5, 7, 5)$ is never achieved, we know it must be impossible.

Since we have $N \bullet 6 \bullet 8 \bullet 6$ states to consider and computing the transitions from each state can be done in $O(1)$, our overall runtime is $O(N)$. By default, our space complexity is also $O(N)$ but this can be reduced to $O(1)$ by performing a bottom-up optimization similar to SSP.

Cutting the Crap

However, with $\mathbf{N}$ going up to 1000, considering $1000 \bullet 6 \bullet 8 \bullet 6 = 288000$ states may be too slow despite being asymptotically optimal depending on your language and other constant factors. We can get around this by realizing that most words in the list are actually useless! We'll never have a use for our 8th 2-syllable word because if we used all 7 prior, we would have run out of syllables to use the 8th. We'll never have a use for our 18th 1-syllable word, 5th 3-syllable word, 4th 4-syllable word, 4th 5-syllable word, 2nd 6-syllable word, 2nd 7-syllable word, or any words with more syllables for the same reason. Thus, we really only need to consider at most $17 + 7 + 4 + 3 + 3 + 1 + 1 = 36$ words or at most $36 \bullet 6 \bullet 8 \bullet 6 = 10368$ states. This drastically cuts down our run time in practice.

Pristine Partitions *Solutions available in Java, Python*

Another approach we could also take is to observe that there just aren't a lot of ways to make a 5 or 7 syllable line using words! Since we've already established the word order in each line is irrelevant, it's only the number of words with each number of syllables that are relevant to each line. The syllable structures are given by the integer partitions of 5 and 7.

There are only 7 ways to make a 5-syllable line. These syllable structures are:

```
[5]                 [4, 1]              [3, 2]
[3, 1, 1]           [2, 2, 1]           [2, 1, 1, 1]
[1, 1, 1, 1, 1]
```

Likewise, there are only 15 ways to make a 7-syllable line. These syllable structures are:

```
[7]                 [6, 1]              [5, 2]
[5, 1, 1]           [4, 3]              [4, 2, 1]
[4, 1, 1, 1]        [3, 3, 1]           [3, 2, 2]
[3, 2, 1, 1]        [3, 1, 1, 1, 1]     [2, 2, 2, 1]
[2, 2, 1, 1, 1]     [2, 1, 1, 1, 1, 1]  [1, 1, 1, 1, 1, 1, 1]
```

These syllable structures could be programmatically generated, or even found by hand.

We can choose any 5-structure, 7-structure, and another 5-structure to create a full haiku structure. There are only $7 \bullet 15 \bullet 7 = 735$ haiku structures we need to check.

With this in mind, we can check if a haiku can be constructed by checking if it's possible to use our words to fulfill any haiku structure! This can be done very cleanly by using a table to count how many words of each syllable we have, then checking each structure to see if our table has at least as many words for each syllable in the structure. If the structure can be filled, we construct a haiku using that structure and any words with the corresponding syllable counts.

It should be easy to see that this simple algorithm runs in $O(N)$ time and uses $O(1)$ space.

Cutting the Crap… Again *Solution available in Python*

Although it wasn't necessary for this test set, it's possible to cut down the number of structures by only considering haiku structures with unique total syllable counts. For example, consider `[[4, 1], [5, 2], [3, 1, 1]]` and `[[5], [4, 3], [2, 1, 1, 1]]`. By ignoring these symmetries, we are left with a mere 155 unique structures that need to be checked.