Mathematical Derivation of formula seen in Day 10's code.

Suppose you have this as your adapter set:

0, 1, 2, 3, 4, 5, 6, 9, 10, 11

If we organize it into runs, we get

[0, 1, 2, 3, 4, 5, 6], [9, 10, 11]

These runs are independent from each other. What we remove or add to get a valid chain in one run does not affect the next.

This means that if the first run had 12 valid combos & the 2nd had two, then the total # of combos comes out to $12 \cdot 2 = 24$.

In order for a run to be valid, there mustn't be a jump greater than 3. This can be further broken down to two rules:

1. The first and last elements of the run cannot be removed.

Ex 1: If the 6 from above or the 9 were removed, there's a jump greater than 3, thus it's invalid.

Ex 2: Even the edge cases of zero & the last # of the last run are subject to this rule, as it either leaves out certain combos or makes the whole sequence invalid just like Ex 1, respectively.

2. No more than two consecutive adapters from the run can be removed.

Eg. 0, 1, 2, 3, 4, 5, 6

removing these two is fine, as 3 is the max jump.

These you cannot remove together, as it jumps from 0 → 4. Can't do that.

Now we have everything we need to tackle this problem.*

*Note that this why a jump of 2 makes these rules null & void, since removing a two then one-jolt jump is invalid. So that means 2 consecutive adapters cannot be removed, but only sometimes. It's not as neat as this solution.

Ex: 0, 1, 2, 3, 4, 5

1 1 0 0 1 1

0 1 4 5

Let us represent this problem of combos as a binary number, where 0 means exclude from the combo & 1 means include in the combo.

The problem now becomes 'How many valid binary numbers are there that satisfy the two rules' that follow?

1. The first & last digits are 1
(From the first & last elements of the run cannot be excluded)

2. No more than two zeros in a row exist
(From no more than two consecutive #s can be excluded)

These problems are equivalent!

Let's take the example of the 6-run from earlier, and transfer it into binary-world

0, 1, 2, 3, 4, 5 ⟶ 1 ? ? ? ? 1

The easiest way to solve this problem is to calculate the total # of binary numbers, then subtract out the ones that violate rule 1 or two.

In this 6-run, to satisfy rule 1, the first & last digits are set-in-store to be 1.

In order to violate rule 2, one of two cases has to happen:

1 0 0 0 ? 1 or 1 ? 0 0 0 1

where '?' can be 0 or 1.

But you get a 'copy' from two different possibilities.

$$\begin{array}{lll}
1000?1 & \to & 100011 \\
 & \to & 100001 \\
1?0001 & \to & 100001 \\
 & \to & 110001
\end{array}$$

Double-count.

This can be remedied by counting out runs of 4. There's only one, so you subtract that out.

So you are left w/ 2^4 combos that don't violate rule 1 are 3 that violate 2. The total is thus

$$2^4 - 3 = \boxed{13}.$$

So to generalize this to n-digits, you have this equation:

$$\underbrace{2^{n-2}}_{\text{combos that don't violate rule 1}} - \underbrace{2^{n-5}}_{\text{variation in other digits}} \underbrace{(n-4)}_{\text{possible combos of 3 zeroes}} + \underbrace{2^{n-6}(n-5)}_{\text{Double-counting of 4+ zero runs accounted for by adding back in}}$$

And this equation is what is found in my code.

The round() function is there since w/ $n \leq 6$, the terms become fractions w/ little impact on the # of calculated combos

And that is a mighty fine closed-form solution!