Exercise 1

The newly-improved calibration document consists of lines of text; each line originally contained a specific calibration value that the Elves now need to recover. On each line, the calibration value can be found by combining the first digit and the last digit (in that order) to form a single two-digit number.

For example:

```
1abc2
pqr3stu8vwx
a1b2c3d4e5f
Treb7uchet
```

In this example, the calibration values of these four lines are 12, 38, 15, and 77. Adding these together produces 142.
Make a .txt file with your lines like the example and make a program that takes the sum of all of the calibration values?

Exercise 2

The Elf explains that you've arrived at Snow Island and apologizes for the lack of snow. He'll be happy to explain the situation, but it's a bit of a walk, so you have some time. They don't get many visitors up here; would you like to play a game in the meantime?
As you walk, the Elf shows you a small bag and some cubes which are either red, green, or blue. Each time you play this game, he will hide a secret number of cubes of each color in the bag, and your goal is to figure out information about the number of cubes.
To get information, once a bag has been loaded with cubes, the Elf will reach into the bag, grab a handful of random cubes, show them to you, and then put them back in the bag. He'll do this a few times per game.
You play several games and record the information from each game (your puzzle input). Each game is listed with its ID number (like the 11 in Game 11: ...) followed by a semicolon-separated list of subsets of cubes that were revealed from the bag (like 3 red, 5 green, 4 blue).

For example, the record of a few games might look like this:

```
Game 1: 3 blue, 4 red; 1 red, 2 green, 6 blue; 2 green
Game 2: 1 blue, 2 green; 3 green, 4 blue, 1 red; 1 green, 1 blue
Game 3: 8 green, 6 blue, 20 red; 5 blue, 4 red, 13 green; 5 green, 1 red
Game 4: 1 green, 3 red, 6 blue; 3 green, 6 red; 3 green, 15 blue, 14 red
Game 5: 6 red, 1 blue, 3 green; 2 blue, 1 red, 2 green
```

In game 1, three sets of cubes are revealed from the bag (and then put back again). The first set is 3 blue cubes and 4 red cubes; the second set is 1 red cube, 2 green cubes, and 6 blue cubes; the third set is only 2 green cubes.
The Elf would first like to know which games would have been possible if the bag contained only 12 red cubes, 13 green cubes, and 14 blue cubes?

In the example above, games 1, 2, and 5 would have been possible if the bag had been loaded with that configuration. However, game 3 would have been impossible because at one point the Elf showed you 20 red cubes at once; similarly, game 4 would also have been impossible because the Elf showed you 15 blue cubes at once. If you add up the IDs of the games that would have been possible, you get 8. Determine which games would have been possible if the bag had been loaded with only 12 red cubes, 13 green cubes, and 14 blue cubes. What is the sum of the IDs of those games?

.txt file:

```
Game 1: 1 blue, 1 red; 10 red; 8 red, 1 blue, 1 green; 1 green, 5 blue
Game 2: 9 green, 11 red; 1 green, 7 red, 1 blue; 1 red, 1 blue, 1 green; 11 green, 3 red, 1 blue; 5 green, 12 red; 8 green, 1 blue, 7 red
Game 3: 16 blue, 2 red, 4 green; 8 red, 4 green; 7 green, 16 blue
Game 4: 3 green, 4 blue, 6 red; 7 red, 12 green, 5 blue; 5 green, 16 red, 8 blue
Game 5: 4 green, 4 blue, 3 red; 4 green, 7 red, 1 blue; 2 blue, 2 red, 4 green; 3 green, 7 red, 4 blue; 2 blue, 3 green, 8 red
```

Exercise 3

The engine schematic (your puzzle input) consists of a visual representation of the engine. There are lots of numbers and symbols you don't really understand, but apparently any number adjacent to a symbol, even diagonally, is a "part number" and should be included in your sum. (Periods (.) do not count as a symbol.)

Here is an example Exercise3.txt file:

```
467..114..
...*......
..35..633.
......#...
617*......
.....+.58.
..592.....
......755.
...$.*....
.664.598..
```

In this Exercise3.txt file, two numbers are not part numbers because they are not adjacent to a symbol: 114 (top right) and 58 (middle right). Every other number is adjacent to a symbol and so is a part number; their sum is 4361.

Make a program that gets the sum of the part numbers in the Exercise3.txtt file.

Exercise 4

You have to figure out which of the numbers you have appear in the list of winning numbers. The first match makes the card worth one point and each match after the first doubles the point value of that card.

For example, your Exercise4.txt file :

```
Card 1: 41 48 83 86 17 | 83 86  6 31 17  9 48 53
Card 2: 13 32 20 16 61 | 61 30 68 82 17 32 24 19
Card 3:  1 21 53 59 44 | 69 82 63 72 16 21 14  1
Card 4: 41 92 73 84 69 | 59 84 76 51 58  5 54 83
Card 5: 87 83 26 28 32 | 88 30 70 12 93 22 82 36
Card 6: 31 18 13 56 72 | 74 77 10 23 35 67 36 11
```

In the above example, card 1 has five winning numbers (41, 48, 83, 86, and 17) and eight numbers you have (83, 86, 6, 31, 17, 9, 48, and 53). Of the numbers you have, four of them (48, 83, 17, and 86) are winning numbers! That means card 1 is worth 8 points (1 for the first match, then doubled three times for each of the three matches after the first).

- Card 2 has two winning numbers (32 and 61), so it is worth 2 points.
- Card 3 has two winning numbers (1 and 21), so it is worth 2 points.
- Card 4 has one winning number (84), so it is worth 1 point.
- Card 5 has no winning numbers, so it is worth no points.
- Card 6 has no winning numbers, so it is worth no points.

So, in this example, the pile of scratchcards is worth 13 points.
Make a program that calculates the points of the pile of scratchcards which is in Exercise4.txt file

Exercise 6

The organizer brings you over to the area where the boat races are held. The boats are much smaller than you expected - they're actually toy boats, each with a big button on top. Holding down the button charges the boat, and releasing the button allows the boat to move. Boats move faster if their button was held longer, but time spent holding the button counts against the total race time. You can only hold the button at the start of the race, and boats don't move until the button is released.

For Exercise6.txt:

```
Time:       58     81     96     76
Distance:   434   1041   2219   1218
```

This document describes three races:

- The first race lasts 7 milliseconds. The record distance in this race is 9 millimeters.
- The second race lasts 15 milliseconds. The record distance in this race is 40 millimeters.
- The third race lasts 30 milliseconds. The record distance in this race is 200 millimeters.

Your toy boat has a starting speed of zero millimeters per millisecond. For each whole millisecond you spend at the beginning of the race holding down the button, the boat's speed increases by one millimeter per millisecond.
So, because the first race lasts 7 milliseconds, you only have a few options:

- Don't hold the button at all (that is, hold it for 0 milliseconds) at the start of the race. The boat won't move; it will have traveled 0 millimeters by the end of the race.
- Hold the button for 1 millisecond at the start of the race. Then, the boat will travel at a speed of 1 millimeter per millisecond for 6 milliseconds, reaching a total distance traveled of 6 millimeters.
- Hold the button for 2 milliseconds, giving the boat a speed of 2 millimeters per millisecond. It will then get 5 milliseconds to move, reaching a total distance of 10 millimeters.
- Hold the button for 3 milliseconds. After its remaining 4 milliseconds of travel time, the boat will have gone 12 millimeters.
- Hold the button for 4 milliseconds. After its remaining 3 milliseconds of travel time, the boat will have gone 12 millimeters.
- Hold the button for 5 milliseconds, causing the boat to travel a total of 10 millimeters.
- Hold the button for 6 milliseconds, causing the boat to travel a total of 6 millimeters.
- Hold the button for 7 milliseconds. That's the entire duration of the race. You never let go of the button. The boat can't move until you let go of the button. Please make sure you let go of the button so the boat gets to move. 0 millimeters.

Since the current record for this race is 9 millimeters, there are actually 4 different ways you could win: you could hold the button for 2, 3, 4, or 5 milliseconds at the start of the race.
In the second race, you could hold the button for at least 4 milliseconds and at most 11 milliseconds and beat the record, a total of 8 different ways to win.
In the third race, you could hold the button for at least 11 milliseconds and no more than 19 milliseconds and still beat the record, a total of 9 ways you could win.

To see how much margin of error you have, determine the number of ways you can beat the record in each race; in this example, if you multiply these values together, you get 288 (4 * 8 * 9).
Write a program that determines the number of ways you could beat the record in each race and returns the result of what you get if you multiply these numbers together

Exercise 7:

In Camel Cards, you get a list of hands, and your goal is to order them based on the strength of each hand. A hand consists of five cards labeled one of A, K, Q, J, T, 9, 8, 7, 6, 5, 4, 3, or 2. The relative strength of each card follows this order, where A is the highest and 2 is the lowest.
Every hand is exactly one type. From strongest to weakest, they are:

- Five of a kind, where all five cards have the same label: AAAAA
- Four of a kind, where four cards have the same label and one card has a different label: AA8AA
- Full house, where three cards have the same label, and the remaining two cards share a different label: 23332
- Three of a kind, where three cards have the same label, and the remaining two cards are each different from any other card in the hand: TTT98
- Two pair, where two cards share one label, two other cards share a second label, and the remaining card has a third label: 23432

- One pair, where two cards share one label, and the other three cards have a different label from the pair and each other: A23A4
- High card, where all cards' labels are distinct: 23456

Hands are primarily ordered based on type; for example, every full house is stronger than any three of a kind.
If two hands have the same type, a second ordering rule takes effect. Start by comparing the first card in each hand. If these cards are different, the hand with the stronger first card is considered stronger. If the first card in each hand has the same label, however, then move on to considering the second card in each hand. If they differ, the hand with the higher second card wins; otherwise, continue with the third card in each hand, then the fourth, then the fifth.
So, 33332 and 2AAAA are both four of a kind hands, but 33332 is stronger because its first card is stronger. Similarly, 77888 and 77788 are both full houses, but 77888 is stronger because its third card is stronger (and both hands have the same first and second card).

To play Camel Cards, you are given a list of hands and their corresponding bid in a .txt file.

Exercise7.txt:

```
32T3K 765
T55J5 684
KK677 28
KTJJT 220
QQQJA 483
```

This example shows five hands; each hand is followed by its bid amount. Each hand wins an amount equal to its bid multiplied by its rank, where the weakest hand gets rank 1, the second-weakest hand gets rank 2, and so on up to the strongest hand. Because there are five hands in this example, the strongest hand will have rank 5 and its bid will be multiplied by 5.
So, the first step is to put the hands in order of strength:

- 32T3K is the only one pair and the other hands are all a stronger type, so it gets rank 1.
- KK677 and KTJJT are both two pair. Their first cards both have the same label, but the second card of KK677 is stronger (K vs T), so KTJJT gets rank 2 and KK677 gets rank 3.
- T55J5 and QQQJA are both three of a kind. QQQJA has a stronger first card, so it gets rank 5 and T55J5 gets rank 4.

Now, you can determine the total winnings of this set of hands by adding up the result of multiplying each hand's bid with its rank (765 * 1 + 220 * 2 + 28 * 3 + 684 * 4 + 483 * 5). So the total winnings in this example are 6440.
Using Exercise7.txt, Write a program Find the rank of every hand in your set and returns the total winnings?

Exercise 9:

Exercise9.txt:

```
0 3 6 9 12 15
1 3 6 10 15 21
10 13 16 21 30 45
```

To best protect the oasis, your environmental report should include a prediction of the next value in each history. To do this, start by making a new sequence from the difference at each step of your history. If that sequence is not all zeroes, repeat this process, using the sequence you just generated as the input sequence. Once all of the values in your latest sequence are zeroes, you can extrapolate what the next value of the original history should be.

In the above dataset, the first history is 0 3 6 9 12 15. Because the values increase by 3 each step, the first sequence of differences that you generate will be 3 3 3 3 3. Note that this sequence has one fewer value than the input sequence because at each step it considers two numbers from the input. Since these values aren't all zero, repeat the process: the values differ by 0 at each step, so the next sequence is 0 0 0 0. This means you have enough information to extrapolate the history! Visually, these sequences can be arranged like this:

```
0   3   6   9  12  15
  3   3   3   3   3
    0   0   0   0
```

To extrapolate, start by adding a new zero to the end of your list of zeroes; because the zeroes represent differences between the two values above them, this also means there is now a placeholder in every sequence above it:

```
0   3   6   9  12  15   B
  3   3   3   3   3   A
    0   0   0   0   0
```

You can then start filling in placeholders from the bottom up. A needs to be the result of increasing 3 (the value to its left) by 0 (the value below it); this means A must be 3:

```
0   3   6   9  12  15   B
  3   3   3   3   3   3
    0   0   0   0   0
```

Finally, you can fill in B, which needs to be the result of increasing 15 (the value to its left) by 3 (the value below it), or 18:

```
0   3   6   9  12  15  18
  3   3   3   3   3   3
    0   0   0   0   0
```

So, the next value of the first history is 18.
Finding all-zero differences for the second history requires an additional sequence:

```
1   3   6  10  15  21
  2   3   4   5   6
    1   1   1   1
      0   0   0
```

Then, following the same process as before, work out the next value in each sequence from the bottom up:

```
1   3   6  10  15  21  28
  2   3   4   5   6   7
    1   1   1   1   1
      0   0   0   0
```

So, the next value of the second history is 28.
The third history requires even more sequences, but its next value can be found the same way:

```
10  13  16  21  30  45  68
   3   3   5   9  15  23
     0   2   4   6   8
       2   2   2   2
         0   0   0
```

So, the next value of the third history is 68.
If you find the next value for each history in this example and add them together, you get 114.

Write a program that extrapolates the values from Exercise9.txt and gets their sum