day04

December 4, 2023

1 Day 4

1.1 Scratchcards

The gondola takes you up. Strangely, though, the ground doesn't seem to be coming with you; you're not climbing a mountain. As the circle of Snow Island recedes below you, an entire new landmass suddenly appears above you! The gondola carries you to the surface of the new island and lurches into the station.

As you exit the gondola, the first thing you notice is that the air here is much **warmer** than it was on Snow Island. It's also quite **humid**. Is this where the water source is?

The next thing you notice is an Elf sitting on the floor across the station in what seems to be a pile of colorful square cards.

"Oh! Hello!" The Elf excitedly runs over to you. "How may I be of service?" You ask about water sources.

"I'm not sure; I just operate the gondola lift. That does sound like something we'd have, though - this is **Island Island**, after all! I bet the **gardener** would know. He's on a different island, though - er, the small kind surrounded by water, not the floating kind. We really need to come up with a better naming scheme. Tell you what: if you can help me with something quick, I'll let you **borrow my boat** and you can go visit the gardener. I got all these scratchcards as a gift, but I can't figure out what I've won."

The Elf leads you over to the pile of colorful cards. There, you discover dozens of scratchcards, all with their opaque covering already scratched off. Picking one up, it looks like each card has two lists of numbers separated by a vertical bar (|): a list of **winning numbers** and then a list of **numbers you have**. You organize the information into a table (your puzzle input).

As far as the Elf has been able to figure out, 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:

```
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 Elf's pile of scratchcards is worth `13` points.

Take a seat in the large pile of colorful cards. **How many points are they worth in total**?

1.2 Part One

I initially loaded each card line into a list with two `sets` of `ints`. I used sets because it would make calculating the number of matches striaghtforward, with `.intersection()`. Once the number of matching numbers is known, I used a tally table/formula (`score_card()`) to calculate the card score, to avoid repeated doubling of a number in place.

[1]:
```
from pathlib import Path
```

[2]:
```
def load_data(path):
    cards = []
    with path.open() as ifh:
        for line in (_ for _ in ifh.readlines()):
            win, have = tuple(line.split(": ")[-1].split(" | "))
            win = {int(_) for _ in win.split()}  # Use sets so that .
 ↪intersection() gives us matches
            have = {int(_) for _ in have.split()}
            cards.append((win, have))
    return cards

def score_card(win, have):
    matches = len(win.intersection(have))
    if matches == 0:  # Use tally table for 0 or 1 match
        return 0
    elif matches == 1:
        return 1
    else:  # Use formula for more matches
        return 2**(matches-1)

def score_cards(cards):
    scores = [score_card(win, have) for (win, have) in cards]
    return scores
```

[3]:
```
cards = load_data(Path("data/day04_test.txt"))
sum(score_cards(cards))
```

[3]: 13

[4]:
```
cards = load_data(Path("data/day04_data.txt"))
sum(score_cards(cards))
```

[4]: 28750

1.3 Part Two

Just as you're about to report your findings to the Elf, one of you realizes that the rules have actually been printed on the back of every card this whole time.

There's no such thing as "points". Instead, scratchcards only cause you to **win more scratchcards** equal to the number of winning numbers you have.

Specifically, you win **copies** of the scratchcards below the winning card equal to the number of matches. So, if card 10 were to have 5 matching numbers, you would win one copy each of cards 11, 12, 13, 14, and 15.

Copies of scratchcards are scored like normal scratchcards and have the **same card number** as the card they copied. So, if you win a copy of card 10 and it has 5 matching numbers, it would then win a copy of the same cards that the original card 10 won: cards 11, 12, 13, 14, and 15. This process repeats until none of the copies cause you to win any more cards. (Cards will never make you copy a card past the end of the table.)

This time, the above example goes differently:

```
‘text 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

Card 1 has four matching numbers, so you win one copy each of the next four cards: cards 2, 3,
Your original card 2 has two matching numbers, so you win one copy each of cards 3 and 4.
Your copy of card 2 also wins one copy each of cards 3 and 4.
Your four instances of card 3 (one original and three copies) have two matching numbers, so you
Your eight instances of card 4 (one original and seven copies) have one matching number, so you
Your fourteen instances of card 5 (one original and thirteen copies) have no matching numbers a
Your one instance of card 6 (one original) has no matching numbers and wins no more cards.
```

Once all of the originals and copies have been processed, you end up with 1 instance of card 1, 2 instances of card 2, 4 instances of card 3, 8 instances of card 4, 14 instances of card 5, and 1 instance of card 6. In total, this example pile of scratchcards causes you to ultimately have 30 scratchcards!

Process all of the original and copied scratchcards until no more scratchcards are won. Including the original set of scratchcards, **how many total scratchcards do you end up with**?

The key to understanding the problem for me was realising *where* the copied cards go. A casual skim might make it look as though they're added on to the end, in sequence, but they're treated as siblings of the inital card in the set. That simplifies things.

In `rescore_cards()` I took the cards I initially read in, and made a dictionary, keyed by card number, with value that's a list with the card and the count of cards with that card number.

Then, we can count the number of matches in the same way as before (I could have refactored to its own function for both solutions, but hey-ho!), and update the card count for the appropriate number of following cards. The critical change here is that we update the *following* card counts by the number of *current* cards.

[5]:
```
def rescore_cards(cards):
    # Change card representation. Use a dictionary, keyed by card ID, and
    # also carry a count of the number of each card
    card_counts = {idx + 1: [card, 1] for (idx, card) in enumerate(cards)}
    for card_id, card_data in card_counts.items():
        card, card_count = card_data
        matches = len(card[0].intersection(card[1]))  # How many matches for␣
↪this card?
        if matches != 0:  # Update the next $matches cards
            for update_id in range(card_id + 1, card_id + matches + 1):
                card_counts[update_id][1] += card_count  # Update the number␣
↪with current card count
    return [_[1] for _ in card_counts.values()]  # Return a list of counts, for␣
↪the puzzle
```

[6]:
```
cards = load_data(Path("data/day04_test.txt"))
sum(rescore_cards(cards))
```

[6]:
```
30
```

[7]:
```
cards = load_data(Path("data/day04_data.txt"))
sum(rescore_cards(cards))
```

[7]:
```
10212704
```