Day 5

Part 1

Satisfied with their search on Ceres, the squadron of scholars suggests subsequently scanning the stationery stacks of sub-basement 17.

The North Pole printing department is busier than ever this close to Christmas, and while The Historians continue their search of this historically significant facility, an Elf operating a very familiar printer beckons you over.

The Elf must recognize you, because they waste no time explaining that the new sleigh launch safety manual updates won't print correctly. Failure to update the safety manuals would be dire indeed, so you offer your services.

Safety protocols clearly indicate that new pages for the safety manuals must be printed in a very specific order. The notation `X|Y` means that if both page number `X` and page number `Y` are to be produced as part of an update, page number `X` must be printed at some point before page number `Y`.

The Elf has for you both the page ordering rules and the pages to produce in each update (your puzzle input), but can't figure out whether each update has the pages in the right order.

For example:

```
47|53
97|13
97|61
97|47
75|29
61|13
75|53
29|13
97|29
53|29
61|53
97|53
61|29
47|13
75|47
97|75
47|61
75|61
47|29
75|13
53|13

75,47,61,53,29
97,61,53,29,13
75,29,13
75,97,47,61,53
61,13,29
97,13,75,29,47
```

The first section specifies the page ordering rules, one per line. The first rule, `47|53`, means that if an update includes both page number `47` and page number `53`, then page number `47` must be printed at some point before page number `53`. (`47` doesn't necessarily need to be immediately before `53`; other pages are allowed to be between them.)

The second section specifies the page numbers of each update. Because most safety manuals are different, the pages needed in the updates are different too. The first update, `75,47,61,53,29`, means that the update consists of page numbers `75`, `47`, `61`, `53`, and `29`.

To get the printers going as soon as possible, start by identifying which updates are already in the right order.

In the above example, the first update (`75,47,61,53,29`) is in the right order:

- `75` is correctly first because there are rules that put each other page after it: `75|47`, `75|61`, `75|53`, and `75|29`.
- `47` is correctly second because `75` must be before it (`75|47`) and every other page must be after it according to `47|61`, `47|53`, and `47|29`.
- `61` is correctly in the middle because `75` and `47` are before it (`75|61` and `47|61`) and `53` and `29` are after it (`61|53` and `61|29`).
- `53` is correctly fourth because it is before page number `29` (`53|29`).
- `29` is the only page left and so is correctly last.

Because the first update does not include some page numbers, the ordering rules involving those missing page numbers are ignored.

The second and third updates are also in the correct order according to the rules. Like the first update, they also do not include every page number, and so only some of the ordering rules apply - within each update, the ordering rules that involve missing page numbers are not used.

The fourth update, `75,97,47,61,53`, is not in the correct order: it would print `75` before `97`, which violates the rule `97|75`.

The fifth update, `61,13,29`, is also not in the correct order, since it breaks the rule `29|13`.

The last update, `97,13,75,29,47`, is not in the correct order due to breaking several rules.

For some reason, the Elves also need to know the middle page number of each update being printed. Because you are currently only printing the correctly-ordered updates, you will need to find the middle page number of each correctly-ordered update. In the above example, the correctly-ordered updates are:

```
75,47,61,53,29
97,61,53,29,13
75,29,13
```

These have middle page numbers of `61`, `53`, and `29` respectively. Adding these page numbers together gives `143`.

Of course, you'll need to be careful: the actual list of page ordering rules is bigger and more complicated than the above example.

Determine which updates are already in the correct order. What do you get if you add up the middle page number from those correctly-ordered updates?

In [1]:
```
from collections import defaultdict
```

```
from math import floor
from pathlib import Path
```

In [2]:

```
test_path = Path("data/day05_test.txt")
data_path = Path("data/day05_data.txt")

def load_data(fpath: Path) -> tuple[dict[str: list[str]], list[list[str]]]:
    """Returns a dictionary of rules and list of updates.

    Rules are keyed by a page number, with the value a list of page
    numbers that must only be found after the keying page.

    Updates are a list of page numbers.
    """
    rules = []
    updates = []

    # Read rules and updates
    target = rules
    for line in [_.strip() for _ in fpath.open().readlines()]:
        if len(line):
            target.append(line)
        else:
            target = updates

    # Process updates
    updates = [_.split(",") for _ in updates]

    # Process rules into dictionary, keyed by page
    rules = [_.split("|") for _ in rules]
    ruledict = defaultdict(list)
    for p1, p2 in rules:  # values in the list must come after the key
        ruledict[p1].append(p2)

    return ruledict, updates


def check_updates(updates: list[list[str]], rules: dict[str: list[str]]) -> tuple[lis
    """Returns a tuple of good updates and bad updates.

    Good updates are those where all pages are already in order, as
    per the passed rule dictionary
    """
    good = []  # holds complying updates
    bad = []  # holds non-complying updates

    for update in updates:
        update_good = True  # Flag to indicate whether update complies
        for key in update:  # Check each update against the rules
            idx_k = update.index(key)  # Position of current key
            for page in rules[key]:    # Check following pages
                if (page in update) and (update.index(page) < idx_k):  # Following pa
                    update_good = False
        # Filter page into compliant/non-compliant
        if not update_good:
            bad.append(update)
        else:
            good.append(update)

    return good, bad


def middle_page_sum(updates: list[list[str]]) -> int:
    """Returns the sum of the middle page numbers of the passed updates"""
    total = 0
```

```
    # Sum the middle pages of each update
    for update in updates:
        idx = floor(len(update) / 2)
        total += int(update[idx])

    return total


rules, updates = load_data(test_path)
good, bad = check_updates(updates, rules)
middle_page_sum(good)
```

Out[2]: 143

In [3]:
```
rules, updates = load_data(data_path)
good, bad = check_updates(updates, rules)
middle_page_sum(good)
```

Out[3]: 7365

Part 2

While the Elves get to work printing the correctly-ordered updates, you have a little time to fix the rest of them.

For each of the incorrectly-ordered updates, use the page ordering rules to put the page numbers in the right order. For the above example, here are the three incorrectly-ordered updates and their correct orderings:

- `75,97,47,61,53` becomes 97,75,47,61,53`.
- `61,13,29` becomes `61,29,13`.
- `97,13,75,29,47` becomes 97,75,47,29,13`.

After taking only the incorrectly-ordered updates and ordering them correctly, their middle page numbers are `47`, `29`, and `47`. Adding these together produces `123`.

Find the updates which are not in the correct order. What do you get if you add up the middle page numbers after correctly ordering just those updates?

In [4]:
```
def reorder_update(update: list[str], rules: dict[str: list[str]]) -> list[str]:
    """Returns the passed update, reordered to comply with the passed rules."""
    # Check each rule against the update
    for key in rules:
        # If the rule is present and has any following pages in the update...
        if key in update and set(update).intersection(set(rules[key])):
            # ...find the earliest following page...
            idx_min_rule = min([update.index(_) for _ in rules[key] if _ in update])
            # ...and, if it's found before the rule key, move the rule key just ahead
            if update.index(key) > idx_min_rule:
                update.remove(key)
                update = update[:idx_min_rule] + [key] + update[idx_min_rule:]

    return update


def reorder_bad(updates: list[list[str]], rules: dict[str: list[str]]) -> list[list[s
    """Return the passed updates, with each reordered to comply with the passed rules
    return [reorder_update(_, rules) for _ in updates]

rules, updates = load_data(test_path)
```

```
good, bad = check_updates(updates, rules)
bad = reorder_bad(bad, rules)
middle_page_sum(bad)
```

Out[4]: 123

In [5]:
```
rules, updates = load_data(data_path)
good, bad = check_updates(updates, rules)
bad = reorder_bad(bad, rules)
middle_page_sum(bad)
```

Out[5]: 5770

In [ ]:
```

```