Day 02

Part 1

Fortunately, the first location The Historians want to search isn't a long walk from the Chief Historian's office.

While the Red-Nosed Reindeer nuclear fusion/fission plant appears to contain no sign of the Chief Historian, the engineers there run up to you as soon as they see you. Apparently, they still talk about the time Rudolph was saved through molecular synthesis from a single electron.

They're quick to add that - since you're already here - they'd really appreciate your help analyzing some unusual data from the Red-Nosed reactor. You turn to check if The Historians are waiting for you, but they seem to have already divided into groups that are currently searching every corner of the facility. You offer to help with the unusual data.

The unusual data (your puzzle input) consists of many reports, one report per line. Each report is a list of numbers called levels that are separated by spaces. For example:

```
7 6 4 2 1
1 2 7 8 9
9 7 6 2 1
1 3 2 4 5
8 6 4 4 1
1 3 6 7 9
```

This example data contains six reports each containing five levels.

The engineers are trying to figure out which reports are safe. The Red-Nosed reactor safety systems can only tolerate levels that are either gradually increasing or gradually decreasing. So, a report only counts as safe if both of the following are true:

- The levels are either all increasing or all decreasing.
- Any two adjacent levels differ by at least one and at most three.

In the example above, the reports can be found safe or unsafe by checking those rules:

- `7 6 4 2 1`: Safe because the levels are all decreasing by 1 or 2.
- `1 2 7 8 9`: Unsafe because 2 7 is an increase of 5.
- `9 7 6 2 1`: Unsafe because 6 2 is a decrease of 4.
- `1 3 2 4 5`: Unsafe because 1 3 is increasing but 3 2 is decreasing.
- `8 6 4 4 1`: Unsafe because 4 4 is neither an increase or a decrease.
- `1 3 6 7 9`: Safe because the levels are all increasing by 1, 2, or 3.

So, in this example, `2` reports are safe.

Analyze the unusual data from the engineers. How many reports are safe?

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

import numpy as np
```

In [2]:
```
test_path = Path("data/day02_test.txt")
```

```python
data_path = Path("data/day02_data.txt")

def load_data(fpath: Path) -> list[np.array]:
    """Return a list of reports read from file (as np.arrays)."""
    reports = []

    for line in fpath.open():  # Populate reports
        reports.append(np.array([int(_) for _ in line.split()]))

    return reports

def gradient(report: np.array) -> tuple[int, int, np.array, np.array]:
    """Returns differences between successive entries in the report.

    Elements in the tuple (in order):

    - -1/0/1 - decreasing/other/increasing
    - count of differences that are zero or go against trend
    - boolean mask of differences that are zero or go against trend
    - indices of differences that are zero or go against trend
    """
    diffs = np.diff(report)

    if sum(diffs) < 0:  # decreasing
        return (-1, sum(diffs >= 0), diffs >= 0, np.where(diffs >= 0))
    if sum(diffs) > 0:  # increasing
        return (1, sum(diffs <= 0), diffs <= 0, np.where(diffs <= 0))

    return (0, sum(diffs != 0), diffs != 0, diffs)


def is_monotonic(report: np.array) -> bool:
    """Returns True if the report is monotonic, False otherwise"""
    grad = gradient(report)

    if grad[0] != 0 and grad[1] == 0:
        return True
    return False


def bad_steps(report: np.array, min: int=1, max: int=3) -> tuple[int, np.array, np.ar
    """Returns bad steps (outwith range) between successive entries in the report.

    Elements in the tuple (in order):

    - count of bad steps
    - boolean mask of bad steps
    - differences between successive elements
    - indices of bad steps
    """
    diffs = np.diff(report)
    bad_steps = (abs(diffs) > max) | (abs(diffs) < min)

    return (sum(bad_steps), bad_steps, diffs, np.where(bad_steps == True))


def is_within_step_size(report: np.array, min: int=1, max: int=3) -> bool:
    """Returns True if all steps are within step size limits, False otherwise."""
    steps = bad_steps(report, min, max)

    if steps[0] == 0:
        return True
    return False
```

```
def assess_reports(reports: list[np.array]) -> tuple[list[np.array], list[np.array]]:
    """Returns tuple of (safe reports, unsafe reports)."""
    safe_reports, unsafe_reports = [], []

    for report in reports:
        if is_monotonic(report) and is_within_step_size(report):
            safe_reports.append(report)
        else:
            unsafe_reports.append(report)


    return (safe_reports, unsafe_reports)

reports = load_data(test_path)
safe, unsafe = assess_reports(reports)
len(safe)
```

Out[2]: 2

In [3]:
```
reports = load_data(data_path)
safe, unsafe = assess_reports(reports)
len(safe)
```

Out[3]: 213

Part 2

The engineers are surprised by the low number of safe reports until they realize they forgot to tell you about the Problem Dampener.

The Problem Dampener is a reactor-mounted module that lets the reactor safety systems tolerate a single bad level in what would otherwise be a safe report. It's like the bad level never happened!

Now, the same rules apply as before, except if removing a single level from an unsafe report would make it safe, the report instead counts as safe.

More of the above example's reports are now safe:

- `7 6 4 2 1`: Safe without removing any level.
- `1 2 7 8 9`: Unsafe regardless of which level is removed.
- `9 7 6 2 1`: Unsafe regardless of which level is removed.
- `1 3 2 4 5`: Safe by removing the second level, 3.
- `8 6 4 4 1`: Safe by removing the third level, 4.
- `1 3 6 7 9`: Safe without removing any level.

Thanks to the Problem Dampener, `4` reports are actually safe!

Update your analysis by handling situations where the Problem Dampener can remove a single level from unsafe reports. How many reports are now safe?

In [4]:
```
def dampener_assess_reports(reports: list[np.array]) -> tuple[list[np.array, np.array
    """Returns tuple of (safe_reports, unsafe_reports)"""
    safe_reports, unsafe_reports = [], []

    for report in reports:

        # Report meets original criteria
        if is_monotonic(report) and is_within_step_size(report):
            safe_reports.append(report)
        elif gradient(report)[1] > 1:  # Two gradient issues, the report can't be fix
```

```
            unsafe_reports.append(report)
        else:  # There's a potential fix by deleting a single element

            # A single gradient issue - delete element to check
            if gradient(report)[1] == 1:
                newrep = np.delete(report, gradient(report)[3][0][0] + 1)
                if is_monotonic(newrep) and is_within_step_size(newrep):
                    safe_reports.append(report)
                    continue
                # If that doesn't work, does deleting the first element help?
                elif gradient(report)[3][0][0] == 0:
                    newrep = np.delete(report, gradient(report)[3][0][0])
                    if is_monotonic(newrep) and is_within_step_size(newrep):
                        safe_reports.append(report)
                        continue

            # A single step size issue - delete element to check
            if bad_steps(report)[0] == 1:
                newrep = np.delete(report, bad_steps(report)[3][0][0] + 1)
                if is_monotonic(newrep) and is_within_step_size(newrep):
                    safe_reports.append(report)
                    continue
                # Check if deleting the first element helps
                if bad_steps(report)[3][0][0] == 0:
                    newrep = np.delete(report, bad_steps(report)[3][0][0])
                    if is_monotonic(newrep) and is_within_step_size(newrep):
                        safe_reports.append(report)
                        continue

            # Couldn't fix - the report is unsafe
            unsafe_reports.append(report)

    return (safe_reports, unsafe_reports)

reports = load_data(test_path)
safe, unsafe = dampener_assess_reports(reports)
len(safe)
```

Out[4]: 4

In [5]:
```
reports = load_data(data_path)
safe, unsafe = dampener_assess_reports(reports)
len(safe)
```

Out[5]: 285

In [ ]:
```

```