Please write a Python script to solve the problem below. It should read the input from a file input.txt, which has the same format as the example.

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?

Below is a Python script to solve the problem. The script reads the input from `input.txt`, parses the rules and updates, checks the ordering of the updates, and calculates the sum of the middle pages of correctly ordered updates.

python Copy code

```python
def parse_input(filename):
    with open(filename, 'r') as file:
        lines = file.read().strip().split('\n')

    # Split rules and updates
    rules = []
    updates = []
    parsing_rules = True
    for line in lines:
        if line == "":
            parsing_rules = False
            continue
        if parsing_rules:
            rules.append(tuple(map(int, line.split('|'))))
        else:
            updates.append(list(map(int, line.split(','))))

    return rules, updates


def is_update_ordered(update, rules):
    # Create a mapping of page indices
    page_index = {page: idx for idx, page in enumerate(update)}

    for x, y in rules:
        if x in page_index and y in page_index:
            if page_index[x] > page_index[y]:
                return False
    return True


def find_middle_page(update):
    mid_idx = len(update) // 2
    return update[mid_idx]


def calculate_sum_of_middle_pages(rules, updates):
    correctly_ordered_updates = []
    for update in updates:
        if is_update_ordered(update, rules):
            correctly_ordered_updates.append(update)

    middle_pages_sum = sum(find_middle_page(update) for update in
correctly_ordered_updates)
    return middle_pages_sum


def main():
    # Read and parse input
    rules, updates = parse_input('input.txt')

    # Calculate the sum of middle pages
    result = calculate_sum_of_middle_pages(rules, updates)

    print(f"Sum of middle pages of correctly ordered updates: {result}")


if __name__ == "__main__":
    main()
```

Explanation:

1. **Parsing Input**:
    - The input is split into rules and updates. The `parse_input` function processes these sections.
    - Rules are stored as tuples of integers, e.g., `(47, 53)`.
    - Updates are stored as lists of integers, e.g., `[75, 47, 61, 53, 29]`.
2. **Checking Update Order**:
    - The `is_update_ordered` function ensures that all applicable rules are respected within an update.
    - A dictionary maps page numbers to their indices for efficient lookups.
3. **Finding Middle Page**:
    - The `find_middle_page` function computes the middle page of an update using the middle index.
4. **Summing Middle Pages**:
    - Correctly ordered updates are identified, and the middle pages of these updates are summed up.

5. **Output**:
    - The final result is printed as the sum of middle pages of all correctly ordered updates.

Instructions:

1. Save the script in a Python file (e.g., `check_updates.py`).
2. Place the input data in a file named `input.txt` in the same directory as the script.
3. Run the script:

```bash
python check_updates.py
```

4. The output will display the sum of the middle pages of correctly ordered updates.

Let me know if you need any further clarifications!

Please update the script for part 2 of the problem:

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?

Here's the updated script that addresses **Part 2** of the problem. It identifies incorrectly ordered updates, fixes their order using the rules, and calculates the sum of the middle pages for these updates after they are corrected.

```python
from collections import defaultdict, deque

def parse_input(filename):
    with open(filename, 'r') as file:
        lines = file.read().strip().split('\n')

    # Split rules and updates
    rules = []
    updates = []
    parsing_rules = True
    for line in lines:
        if line == "":
            parsing_rules = False
            continue
        if parsing_rules:
            rules.append(tuple(map(int, line.split('|'))))
        else:
            updates.append(list(map(int, line.split(','))))

    return rules, updates


def is_update_ordered(update, rules):
    # Create a mapping of page indices
    page_index = {page: idx for idx, page in enumerate(update)}

    for x, y in rules:
        if x in page_index and y in page_index:
            if page_index[x] > page_index[y]:
                return False
    return True


def find_middle_page(update):
    mid_idx = len(update) // 2
    return update[mid_idx]


def fix_update_order(update, rules):
    # Build a graph based on rules
    graph = defaultdict(list)
    in_degree = defaultdict(int)
    update_set = set(update)

    # Add edges only for pages in this update
    for x, y in rules:
        if x in update_set and y in update_set:
            graph[x].append(y)
            in_degree[y] += 1
            if x not in in_degree:
                in_degree[x] = 0

    # Perform topological sort (Kahn's Algorithm)
    queue = deque([node for node in update if in_degree[node] == 0])
    sorted_update = []

    while queue:
        node = queue.popleft()
        sorted_update.append(node)
        for neighbor in graph[node]:
            in_degree[neighbor] -= 1
            if in_degree[neighbor] == 0:
                queue.append(neighbor)

    return sorted_update


def calculate_sum_of_middle_pages_part2(rules, updates):
    incorrectly_ordered_updates = []
    for update in updates:
        if not is_update_ordered(update, rules):
            incorrectly_ordered_updates.append(update)

    fixed_updates = [fix_update_order(update, rules) for update in incorrectly_ordered_updates]
    middle_pages_sum = sum(find_middle_page(update) for update in fixed_updates)

    return middle_pages_sum


def main():
    # Read and parse input
    rules, updates = parse_input('input.txt')

    # Calculate the sum of middle pages for incorrectly ordered updates after fixing them
    result = calculate_sum_of_middle_pages_part2(rules, updates)

    print(f"Sum of middle pages of fixed updates: {result}")


if __name__ == "__main__":
    main()
```

Explanation of Changes:

1. **Fixing the Order**:
    - The `fix_update_order` function uses a **topological sort** algorithm to reorder the updates based on the rules.
    - A directed graph is built using the rules applicable to the pages in the update.

   - **Kahn's Algorithm** is used to ensure the correct order of pages.
2. **Processing Incorrect Updates**:
   - Updates that are not in the correct order are identified using the `is_update_ordered` function.
   - These updates are passed to `fix_update_order` to be corrected.
3. **Sum of Middle Pages**:
   - The fixed updates are processed to find their middle pages, which are summed up.

Instructions:

1. Save the script in the same Python file or as a new file (e.g., `fix_updates.py`).
2. Place the input data in `input.txt`.
3. Run the script:

   bash Copy code

   ```bash
   python fix_updates.py
   ```
4. The script will output the sum of middle pages of the fixed updates.

Let me know if you have more questions!

Message ChatGPT