Мамедова Эльмира Ровшановна
Логин Яндекс. Контеста: если подразумевается почта от аккаунт, на котором проходила тестирование: mamedovaelmira7777777@yandex.ru
Если почта с которой регистрировалась: mamedovaelmira9584@gmail.com

Задача 1. Fail2ban на коленке
По примеру лога:
[2024-04-13 15:31:17] WARNING Request from 56.185.32.17 ("GET /api/geojson?url=file:///etc/hosts HTTP/1.1") is malicious
Нам необходимо найти такие строки, которые содержат: IP-адрес клиента, HTTP-метод, путь с параметрами запроса.

По примеру видно, что вредоносные строки всегда содержат: Request from <IP> ("<METHOD> <PATH> HTTP/...") is malicious. Соответственно мы можем ориентироваться на ключевые слова: Request from, »METHOD PATH HTTP/…», is malicious

Попробуем выделить IP-адрес. Наблюдаем, что IP всегда идёт после Request from, например: Request from 56.185.32.17

Формат IP-адреса: четыре числа от 0 до 255, разделённые точками, упрощённо можно взять 1–3 цифры в каждом блоке: \d{1,3}.
Регулярка для IP: (?P<ip>\d{1,3}(?:\.\d{1,3}){3})
Разбор: (?P<ip>…) — именованная группа ip, \d{1,3} — от 1 до 3 цифр, (?:\.\d{1,3}){3} — три раза: точка и ещё блок чисел.

Пробуем выделить HTTP-метод
Сразу после IP идёт строка запроса в кавычках
("GET /api/geojson?... HTTP/1.1")
Регулярка для метода: (?P<method>[A-Z]+)

Выделяем путь и параметры запроса
После метода идёт путь, включая query string:
/api/geojson?url=file:///etc/hosts
Он идёт до первого пробела (в котором начинается HTTP/1.1), значит, можно взять:
(?P<path>[^]+)
Разбор:
(?P<path>...) — именованная группа path,
[^]+ — всё, кроме пробела (до HTTP/...).

Прописываем всё остальное дословно
Теперь берём фиксированные части строки и просто добавляем их:
"Request from" — перед IP;
(" — перед методом;
HTTP/...") — после пути;
is malicious — конец строки.

Собираем финальную регулярку
r'Request from (?P<ip>\d{1,3}(?:\.\d{1,3}){3}) \("(?P<method>[A-Z]+) (?P<path>[^]+) HTTP/[\d.]+"\) is malicious'
Разбор целиком:
Request from — маркер начала;
(?P<ip>...) — IP-адрес;
\ (" — начало строки запроса в кавычках;
(?P<method>...) — HTTP-метод;
(?P<path>...) — путь с параметрами;
HTTP/[\d.]+ — версия HTTP;
"\) is malicious — завершение вредоносной строки.

Задача 6. Преобразование метрик

По условию задачи нам необходимо за минимальное количество операций превратить число 1 в заданное число n.
Нам доступны 4 операции:

1. Умножить на A
2. Умножить на B
3. Прибавить 1
4. Прибавить C

Нужно определить минимальное количество таких шагов, чтобы из 1 получить n.

По ограничениям мы имеем, что число n может быть до 10^6, поэтому мы не можем перебрать все варианты в лоб(рекурсией или bcktracing), потому что не пройдем TL.

Рассмотрим пример из условия:

Пример:
1
1000 4 5 2
Ответ:
5
То есть из 1 мы должны получить 1000, используя операции с коэффициентами A = 4, B = 5, C = 2
И можем сделать это всего за 5 шагов!
Это навевает мысль: при правильной последовательности действий можно дойти до n довольно быстро, особенно за счёт умножений.

Придумаем свой пример и возьмем значения поменьше, чтобы можно было проверить своими руками.
Например: n = 10, A = 2, B = 3, C = 4.
Шаг 1: 1 * 2 = 2 или 1 * 3 = 3, 1 + 1 = 2, 1 + 4 = 5
Шаг 2: из 2 -> 4, 3; из 3 -> 6, 4, 7; из 5 -> 6, 9, 10
Мы получили 10 за 2 шага: 1 -> 5 -> 10

Подумаем, можем ли мы за DFS получить верный ответ и пройти ограничения по времени? Тогда мы будем перебирать все варианты, что очень медленно и трудно отследить уже посещенные состояния.

Можно попробовать через жадный подход: давайте будем сначала все умножать, а затем прибавлять. Но тогда на примере: 1 -> * 2 = 2 -> *2 = 4 -> *2 = 8 -> +1 = 9 -> +1 = 10 (5 шагов), но 1 -> +4 = 5 -> +5 = 10 (2 шага). Жадный алгоритм дает неоптимальный вариант(неверный ответ).

В целом мы можем рассмотреть задачу с помощью рюкзака(динамического программирования). Представим, что мы создаём dp[i] = минимальное количество шагов, чтобы дойти до i (итерируем от 1 до n, для каждой позиции пробуем: dp[i + 1] = min(dp[i + 1], dp[i] + 1) и так далее). Но тогда будет трудно сказать, когда именно мы попадаем в i.

Тогда можно рассмотреть BFS. Рассмотрим все возможные значения чисел как вершины графа, каждое применение операции - переход в новое число(ребро), ищем минимальное количество шагов - этот результат и будет кратчайшим путем и соответственно результатом.
Мы будем строить неявный граф, где из вершины идем в +1, +C, *A, *B и запустим BFS от 1.
Код представлен к самой задаче(результат - OK).

Операции имеют одинаковый вес(то есть выполняются за одинаковую «стоимость», в ином случае прибегли бы к алгоритму Дейкстры)

Оценка сложности: в худшем случае мы рассмотрим все n чисел, для каждого числа 4 перехода.
Time: O(n)
Memory: O(n) - создание 2 доп массивов.