```
In [4]: def read_input():
            try:
                n, d = map(int, input().strip().split())
                teacher_data = []
                for _ in range(n):
                    arrival_day, lectures, curse_level = map(int, input().strip().split(
                    teacher_data.append([arrival_day, lectures, curse_level])
                return n, d, teacher_data
            except ValueError:
                return None

        def evaluate_lecture_schedule(n, d, teacher_data):
            total_curse = 0
            lecture_schedule = []

            for day in range(1, d + 1):
                for teacher in teacher_data:
                    if teacher[0] == day:
                        lecture_schedule.append(teacher)

                lecture_schedule.sort(key=lambda x: x[2], reverse=True)

                if lecture_schedule:
                    lecture_schedule[0][1] -= 1

                lecture_schedule += [[1, 0, 0] for _ in range(len(lecture_schedule))]
                lecture_schedule = [teacher for teacher in lecture_schedule if teacher[1

            for teacher in lecture_schedule:
                if teacher[1] > 0:
                    total_curse += teacher[2] * teacher[1]

            print(total_curse)

        def lectures():
            input_data = read_input()

            if input_data is None:
                print("Invalid input format")
                return

            n, d, teacher_data = input_data
            evaluate_lecture_schedule(n, d, teacher_data)

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

```
2 3
1 2 300
2 2 100
100
```

```
In [10]: import heapq

         def find_optimal_vaccine_time(n, m, portals, demon_patrols):
             min_time = [float('inf')] * n
             min_time[0] = 0

             heap = [(0, 0)]
```

```python
    while heap:
        time, node = heapq.heappop(heap)

        for next_node, travel_time in enumerate(portals[node]):
            if travel_time == -1:
                continue

            demon_interact = 1 if time + travel_time in demon_patrols[next_node]
            new_time = time + travel_time + demon_interact

            if new_time < min_time[next_node]:
                min_time[next_node] = new_time
                heapq.heappush(heap, (new_time, next_node))

    if min_time[n - 1] != float('inf'):
        return min_time[n - 1]
    else:
        return -1

if __name__ == "__main__": v
    n, m = map(int, input().split())
    portals = [[-1] * n for _ in range(n)]

    for _ in range(m):
        start, end, time = map(int, input().split())
        portals[start - 1][end - 1] = time

    demon_patrols = []

    for _ in range(n):
        patrol_info = set(map(int, input().split()))
        demon_patrols.append(patrol_info)

    result = find_optimal_vaccine_time(n, m, portals, demon_patrols)
    print(result)
```

```
4 4
1 2 3
1 3 2
2 4 2
3 4 3
0
1 4
2 2 3
0
5
```