P8.

이원빈

문제분석 및 아이디어.

한 불도저로 하나의 task를 하루만에 끝낼 수 있기에, task의 early due date가 곧 task의 시작일이다.
더불어 S_i + d(duration) ≤ n이므로, Si의 범위를 지정하여 시작일이 S_i (1 ≤ i ≤ n-d) 인 task Ti가 총 m개 있다고 할 수 있다.
이러한 T_i들을 각각의 시작일 Si로 모아, 각 시작일에 대한 task의 수로 나타낼 수 있다. 즉, **각 Si에 '몇 개의 task가 있는지'**로 바꿔 볼 수 있다.

이 때, S_1에 있는 task부터 S_(max(s_i))까지 현재의 불도저 개수로, **S_i의 task를 주어진 기간(d+1일) 내에 처리할 수 있는지 확인한다.**
예를 들어 S_1의 task가 4개이고 d가 2, 불도저의 수가 1이라면, 4 / 3 = 1.3이고 이는 곧 하루 평균 1.3개의 task를 수행해야함을 의미한다.
불도저 한 개로는 부족하므로 이렇게 되면 불도저의 수를 하나 늘리고, 처음부터 다시 확인해본다.

또한 위 조건은 만족하지만 만약 현재의 불도저 개수보다 S_i의 task 수가 더 많다면 그 차이만큼을 다음 날로 이월시켜서 계산해야한다.

해결방법.

1. n, d, m을 입력받고, 이후 m개의 task를 입력받아 1차원 배열 startDates에 저장한다. 이 때, latestDate를 설정하여 task를 입력 받을 때 마다 해당 Si가 latestDate보다 큰지 확인한다. 만약 크다면 latestDate를 그 task의 Si로 설정한다.
2. 각 Si마다의 task수를 저장할 1차원 배열 tasksPerDay를 latestDate+1의 크기만큼 만들어주고, 0으로 초기화 한다. 편의를 위해 latestDate+1로 설정한 것이다.
3. startDates[i] (1 ≤ i ≤ m)를 순차적으로 확인하며, tasksPerDay[startDates[i]]를 하나씩 늘려준다.
4. 전 날에 수행하지 못한 task의 수인 remainTask로 설정하고, 0으로 초기화한다. 또한 불도저의 수는 1로 초기화한다.
5. tasksPerDay[i] (1 ≤ i ≤ latestDate)에 대해 순차적으로 다음의 과정을 반복한다.
    a. (tasksPerDay[i] + remainTask)를 (d+1)로 나눈 값을 불도저의 수와 비교한다.
        i. 만약 더 크다면 불도저가 더 필요하다는 것이므로, 불도저의 수를 하나 늘리고 첫째날로 다시 돌아간다.
        ii. 더 크지 않다면, remainTask를 (tasksPerDay[i] + remainTask) - 불도저의 수로 만든다. 만약 음수라면 0으로 설정한다.
6. 5번의 과정을 통해 구한 불도저의 수를 출력한다.

시간복잡도.

1. m개의 task를 1차원 배열에 저장 -> O(m)
2. 5번의 과정에서,1부터 latestDate까지 반복 -> 최악의 경우 (1 * 2 * … (latestDate-1) * latesDate)번을 봐야한다. 하지만 상수값은 시간 복잡도에 포함할 수 없다.

따라서 시간 복잡도는 O(m)이다.