1. 더 맵게

문제 설명

매운 것을 좋아하는 Leo는 모든 음식의 스코빌 지수를 K 이상으로 만들고 싶습니다. 모든 음식의 스코빌 지수를 K 이상으로 만들기 위해 Leo는 스코빌 지수가 가장 낮은 두 개의 음식을 아래와 같이 특별한 방법으로 섞어 새로운 음식을 만듭니다.
섞은 음식의 스코빌 지수 = 가장 맵지 않은 음식의 스코빌 지수 + (두 번째로 맵지 않은 음식의 스코빌 지수 * 2)

Leo는 모든 음식의 스코빌 지수가 K 이상이 될 때까지 반복하여 섞습니다.
Leo가 가진 음식의 스코빌 지수를 담은 배열 scoville과 원하는 스코빌 지수 K가 주어질 때, 모든 음식의 스코빌 지수를 K 이상으로 만들기 위해 섞어야 하는 최소 횟수를 return 하도록 solution 함수를 작성해주세요.

제한 사항

- scoville의 길이는 2 이상 1,000,000 이하입니다.
- K는 0 이상 1,000,000,000 이하입니다.
- scoville의 원소는 각각 0 이상 1,000,000 이하입니다.
- 모든 음식의 스코빌 지수를 K 이상으로 만들 수 없는 경우에는 -1을 return 합니다.

입출력 예

scoville	K	return
[1, 2, 3, 9, 10, 12]	7	2

입출력 예 설명

1. 스코빌 지수가 1인 음식과 2인 음식을 섞으면 음식의 스코빌 지수가 아래와 같이 됩니다.
   새로운 음식의 스코빌 지수 = 1 + (2 * 2) = 5
   가진 음식의 스코빌 지수 = [5, 3, 9, 10, 12]
2. 스코빌 지수가 3인 음식과 5인 음식을 섞으면 음식의 스코빌 지수가 아래와 같이 됩니다.
   새로운 음식의 스코빌 지수 = 3 + (5 * 2) = 13
   가진 음식의 스코빌 지수 = [13, 9, 10, 12]

모든 음식의 스코빌 지수가 7 이상이 되었고 이때 섞은 횟수는 2회입니다.

```
import java.util.PriorityQueue;

class Solution {
   public int solution(int[] scoville, int K) {
```

```
        int answer = 0;
        PriorityQueue<Integer> heap = new PriorityQueue();

        for (int aScoville : scoville) {
            heap.offer(aScoville);
        }

        while (heap.peek() <= K) {
            if (heap.size() == 1) {
                return -1;
            }
            int a = heap.poll();
            int b = heap.poll();


            int result = a + (b * 2);

            heap.offer(result);
            answer ++;
        }
        return answer;
    }
}
```

https://jar100.tistory.com/18

2. 디스크 컨트롤러

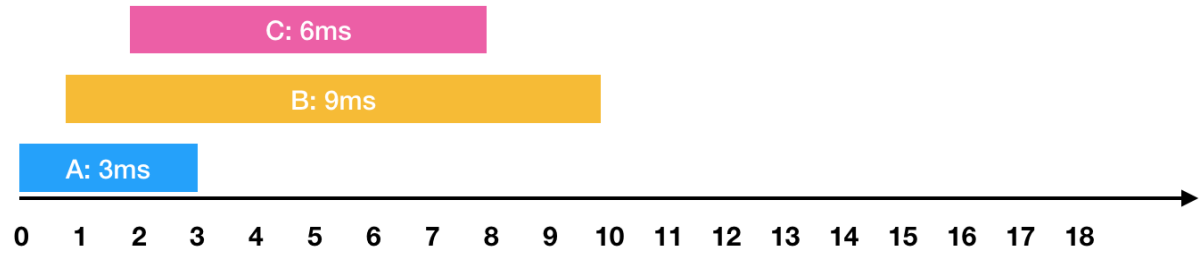
문제 설명

하드디스크는 한 번에 하나의 작업만 수행할 수 있습니다. 디스크 컨트롤러를 구현하는 방법은 여러 가지가 있습니다. 가장 일반적인 방법은 요청이 들어온 순서대로 처리하는 것입니다.

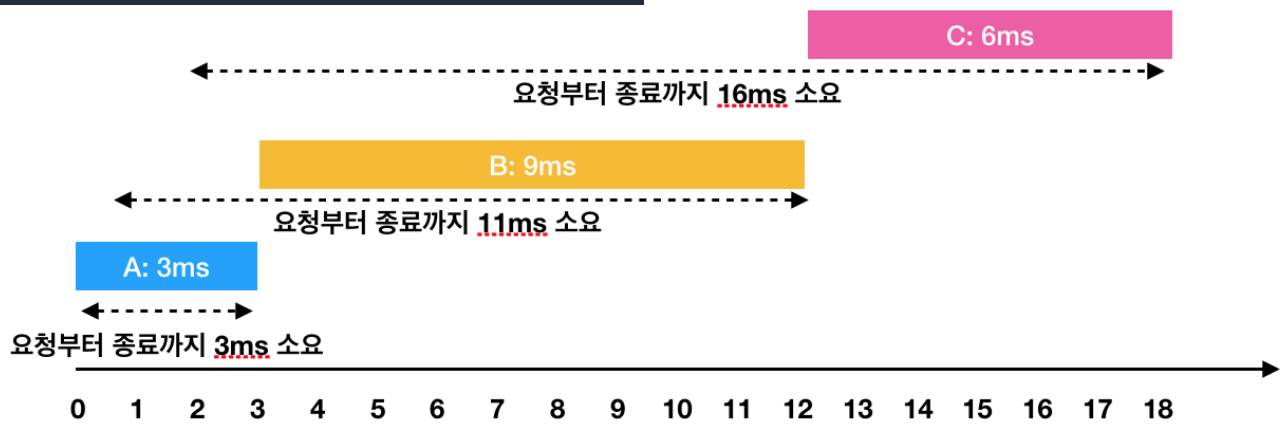
예를들어

- 0ms 시점에 3ms 가 소요되는 A 작업 요청
- 1ms 시점에 9ms 가 소요되는 B 작업 요청
- 2ms 시점에 6ms 가 소요되는 C 작업 요청

와 같은 요청이 들어왔습니다. 이를 그림으로 표현하면 아래와 같습니다.



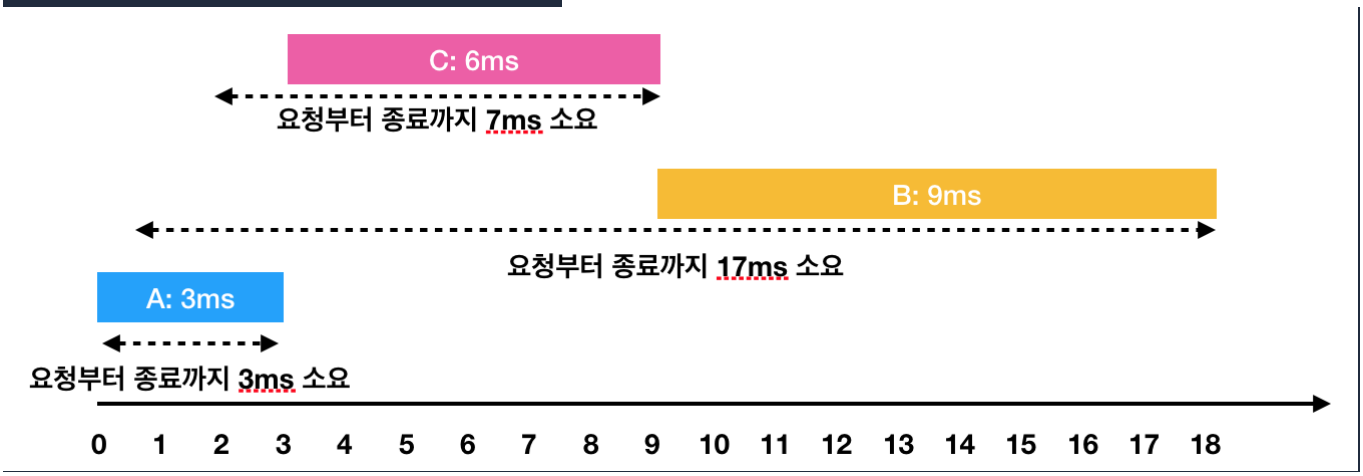
한 번에 하나의 요청만을 수행할 수 있기 때문에 각각의 작업을 요청받은 순서대로 처리하면 다음과 같이 처리 됩니다.



- A: 3ms 시점에 작업 완료 (요청에서 종료까지 : 3ms)
- B: 1ms 부터 대기하다가, 3ms 시점에 작업을 시작해서 12ms 시점에 작업 완료(요청에서 종료까지 : 11ms)
- C: 2ms 부터 대기하다가, 12ms 시점에 작업을 시작해서 18ms 시점에 작업 완료(요청에서 종료까지 : 16ms)

이 때 각 작업의 요청부터 종료까지 걸린 시간의 평균은 10ms(= (3 + 11 + 16) / 3)가 됩니다.

하지만 A → C → B 순서대로 처리하면



- A: 3ms 시점에 작업 완료(요청에서 종료까지 : 3ms)
- C: 2ms 부터 대기하다가, 3ms 시점에 작업을 시작해서 9ms 시점에 작업 완료(요청에서 종료까지 : 7ms)
- B: 1ms 부터 대기하다가, 9ms 시점에 작업을 시작해서 18ms 시점에 작업 완료(요청에서 종료까지 : 17ms)

이렇게 A → C → B 의 순서로 처리하면 각 작업의 요청부터 종료까지 걸린 시간의 평균은 9ms(= (3 + 7 + 17) / 3)가 됩니다.

각 작업에 대해 [작업이 요청되는 시점, 작업의 소요시간]을 담은 2 차원 배열 jobs 가 매개변수로 주어질 때, 작업의 요청부터 종료까지 걸린 시간의 평균을 가장 줄이는 방법으로 처리하면 평균이 얼마가 되는지 return 하도록 solution 함수를 작성해주세요. (단, 소수점 이하의 수는 버립니다)

제한 사항

- jobs 의 길이는 1 이상 500 이하입니다.
- jobs 의 각 행은 하나의 작업에 대한 [작업이 요청되는 시점, 작업의 소요시간] 입니다.
- 각 작업에 대해 작업이 요청되는 시간은 0 이상 1,000 이하입니다.
- 각 작업에 대해 작업의 소요시간은 1 이상 1,000 이하입니다.
- 하드디스크가 작업을 수행하고 있지 않을 때에는 먼저 요청이 들어온 작업부터 처리합니다.

입출력 예

jobs	return
[[0, 3], [1, 9], [2, 6]]	9

입출력 예 설명

문제에 주어진 예와 같습니다.

- 0ms 시점에 3ms 걸리는 작업 요청이 들어옵니다.
- 1ms 시점에 9ms 걸리는 작업 요청이 들어옵니다.
- 2ms 시점에 6ms 걸리는 작업 요청이 들어옵니다.

```
import java.util.*;

class Solution {
    class Job {
        int requestTime;
        int workingTime;

        Job(int requestTime, int workingTime){
            this.requestTime = requestTime;
            this.workingTime = workingTime;
        }
    }

    public int solution(int[][] jobs) {
        LinkedList<Job> waiting = new LinkedList<>();
        PriorityQueue<Job> pq = new PriorityQueue<>(new
Comparator<Job>() {
                @Override
                public int compare(Job j1, Job j2) {
                        return j1.workingTime - j2.workingTime;
                }
        });

        for(int[] job : jobs) {
                waiting.offer(new Job(job[0], job[1]));
        }

        Collections.sort(waiting, new Comparator<Job>() {
                        @Override
                        public int compare(Job j1, Job j2) {
                                return j1.requestTime - j2.requestTime;
                        }
        });

        int answer = 0;
        int cnt = 0;
        int time = waiting.peek().requestTime;

        while(cnt < jobs.length) {
```

```
            while(!waiting.isEmpty() &&
waiting.peek().requestTime <= time) {
                  pq.offer(waiting.pollFirst());
            }

            if(!pq.isEmpty()) {
                  Job job = pq.poll();
                  time += job.workingTime;
                  answer += time - job.requestTime;
                  cnt++;
            } else {
                  time++;
            }
      }

      return answer / cnt;
   }
}
```

https://velog.io/@hyeon930/%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%A8%B8%EC%8A%A4-%EB%94%94%EC%8A%A4%ED%81%AC-%EC%BB%A8%ED%8A%B8%EB%A1%A4%EB%9F%AC-Java

3. 이중우선순위큐

이중 우선순위 큐는 다음 연산을 할 수 있는 자료구조를 말합니다.

명령어	수신 탑(높이)
I 숫자	큐에 주어진 숫자를 삽입합니다.
D 1	큐에서 최댓값을 삭제합니다.
D -1	큐에서 최솟값을 삭제합니다.

이중 우선순위 큐가 할 연산 operations 가 매개변수로 주어질 때, 모든 연산을 처리한 후 큐가 비어있으면 [0,0] 비어있지 않으면 [최댓값, 최솟값]을 return 하도록 solution 함수를 구현해주세요.

제한사항

- operations 는 길이가 1 이상 1,000,000 이하인 문자열 배열입니다.
- operations 의 원소는 큐가 수행할 연산을 나타냅니다.
  - 원소는 “명령어 데이터” 형식으로 주어집니다.- 최댓값/최솟값을 삭제하는 연산에서 최댓값/최솟값이 둘 이상인 경우, 하나만 삭제합니다.
- 빈 큐에 데이터를 삭제하라는 연산이 주어질 경우, 해당 연산은 무시합니다.

입출력 예

operations	return
[I 16,D 1]	[0,0]
[I 7,I 5,I -5,D -1]	[7,5]

입출력 예 설명

16 을 삽입 후 최댓값을 삭제합니다. 비어있으므로 [0,0]을 반환합니다.
7,5,-5 를 삽입 후 최솟값을 삭제합니다. 최대값 7, 최소값 5 를 반환합니다.

```
import java.util.*;

class Solution {

   public int[] solution(String[] operations) {
        int[] answer = new int[2];
        Queue<integer> maxQueue = new
PriorityQueue<>(Collections.reverseOrder());
```

```
        Queue<integer> minQueue = new PriorityQueue<>();

        for(String operation : operations) {
           String[] cmd = operation.split(" ");
           if(cmd[0].equals("I")) {
              maxQueue.add(Integer.parseInt(cmd[1]));
              minQueue.add(Integer.parseInt(cmd[1]));
           } else if(!maxQueue.isEmpty()){
              if(cmd[1].equals("1")) deleteElement(minQueue,
maxQueue.poll());
              else deleteElement(maxQueue, minQueue.poll());
           }
        }

        answer[0] = maxQueue.isEmpty() ? 0 : maxQueue.poll();
        answer[1] = minQueue.isEmpty() ? 0 : minQueue.poll();
        return answer;
    }

   public void deleteElement(Queue<integer> queue, int num) {
      List<integer> temp = new ArrayList<>();
      while(!queue.isEmpty()) {
         int extract = queue.poll();
         if(extract == num) break;
         temp.add(extract);
      }
      queue.addAll(temp);
   }
}
```

https://lkhlkh23.tistory.com/115