디스크 컨트롤러

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

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

제한 사항

jobs의 길이는 1 이상 500 이하입니다.
jobs의 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.

Key Point

1. 평균시간을 적게 만들수 있는 스케쥴링 알고리즘 숙지

2. 순서를 정하기 위한 sort, pq 활용시 우선순위 배치 방법

풀이

1. 모든 작업을 대기하는 큐를 선언하고, 요청 시간 순으로 정렬한다 (도착하는 순서)

2. 현재 처리가 가능한 작업 (현재 time 보다 낮은 reqTime) 을 모두 대기 큐에서 옮긴다.

3. 작업이 가능한 것들 중에 작업시간이 가장 짧게 끝나는 기준으로 경과 시간을 계산한다.

4. 작업이 남아 있는데 현재 시간에서 처리 하지 못할 경우 시간을 경과 시키며 작업을 반복 한다.

5. 모든 작업이 완료 된 경우 평균 값을 계산하여 리턴.

코드

```
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedList;
import java.util.PriorityQueue;

class Job {
  int reqTime;
  int workTime;

  Job(int reqTime, int workTime) {
    this.reqTime = reqTime;
    this.workTime = workTime;
  }
}

class Solution {
    public static int solution(int[][] jobs) {
        LinkedList<Job> wait = new LinkedList<>();

        PriorityQueue<Job> pq = new PriorityQueue<Job>(new Comparator<Job>() {

      @Override
      public int compare(Job o1, Job o2) {
        return o1.workTime - o2.workTime;
      }
    });

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

        Collections.sort(wait, new Comparator<Job>() {

      @Override
      public int compare(Job o1, Job o2) {
        return o1.reqTime - o2.reqTime;
      }
    });

      int answer = 0;
        int cnt = 0;
        int time = 0;

        while(cnt < jobs.length) {

          while(!wait.isEmpty() && wait.peek().reqTime <= time) {
            pq.offer(wait.pollFirst());
          }
          if(!pq.isEmpty()) {
            Job job = pq.poll();
            time += job.workTime;
            answer += time - job.reqTime;
            cnt++;
          }
          else {
            time++;
```

```
          }
        }
        0 , 1 , 2     0 , 1 , 0, 3   20, 5
          return answer / cnt;
    }
}
```

스케쥴링 기법

```
1. FCFS ( First Come , First Serve )

먼저 도착하는 프로세스를 먼저 처리 하는 스케쥴링 기법

간단하게 구현 할 수 있지만, 긴 프로세스가 먼저 도착하게 되면 나머지 프로세스는 대기하게 된다.

먼저 도착하는 프로세스의 크기에 따라 평균 완료시간의 차이의 편차가 크다.


2. SJF ( Shorted Job First )

최단 작업 우선 스케쥴링, 가장 작은 평균시간을 달성 할 수 있다.

가장 적은 평균 대기시간을 달성 할 수 있지만, 원래 스케쥴링 기법은 다음 프로세스의 작업시간을

계산하여야 하므로 획기적인 큰 차이를 보이지는 않는다고 한다.

하지만 이번 문제의 경우 모든 시간을 주어졌기 때문에 해당 알고리즘을 통해 최적화가 가능하다.
```

참고

```
https://oaksong.github.io/2018/02/12/cpu-scheduling/
```