힙/우선순위 큐

Programmers

#디스크 컨트롤러

25.06.30

Problem

디스크 컨트롤러

~ 22:25 | 문제 풀기
~ 22:30| 힌트 공개
~ 22:55 | 2차 풀기
~ 23:00 | 풀이공개

코딩테스트 연습 〉 힙(Heap) 〉 **디스크 컨트롤러**

디스크 컨트롤러 제출 내역 dark

문제 설명

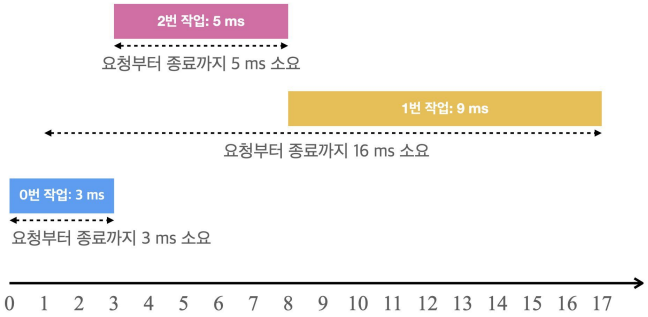
하드디스크는 한 번에 하나의 작업만 수행할 수 있습니다. 디스크 컨트롤러를 구현하는 방법은 여러 가지가 있습니다. 이 문제에서는 **우선순위 디스크 컨트롤러**라는 가상의 장치를 이용한다고 가정합니다. **우선순위 디스크 컨트롤러**는 다음과 같이 동작합니다.

1. 어떤 작업 요청이 들어왔을 때 작업의 번호, 작업의 요청 시각, 작업의 소요 시간을 저장해 두는 **대기 큐**가 있습니다. 처음에 이 큐는 비어있습니다.
2. 디스크 컨트롤러는 하드디스크가 작업을 하고 있지 않고 **대기 큐**가 비어있지 않다면 가장 우선순위가 높은 작업을 **대기 큐**에서 꺼내서 하드디스크에 그 작업을 시킵니다. 이때, 작업의 소요시간이 짧은 것, 작업의 요청 시각이 빠른 것, 작업의 번호가 작은 것 순으로 우선순위가 높습니다.
3. 하드디스크는 작업을 한 번 시작하면 작업을 마칠 때까지 그 작업만 수행합니다.
4. 하드디스크가 어떤 작업을 마치는 시점과 다른 작업 요청이 들어오는 시점이 겹친다면 하드디스크가 작업을 마치자마자 디스크 컨트롤러는 요청이 들어온 작업을 **대기 큐**에 저장한 뒤 우선순위가 높은 작업을 **대기 큐**에서 꺼내서 하드디스크에 그 작업을 시킵니다. 또, 하드디스크가 어떤 작업을 마치는 시점에 다른 작업이 들어오지 않더라도 그 작업을 마치자마자 또 다른 작업을 시작할 수 있습니다. 이 과정에서 걸리는 시간은 없다고 가정합니다.

예를 들어

```
- 0ms 시점에 3ms가 소요되는 0번 작업 요청
- 1ms 시점에 9ms가 소요되는 1번 작업 요청
- 3ms 시점에 5ms가 소요되는 2번 작업 요청
```

와 같은 요청이 들어왔습니다. 이를 그림으로 표현하면 다음과 같습니다.



작업 번호	요청 시각	작업 종료 시각	반환 시간
0번	0ms	3ms	3ms(= 3ms - 0ms)
1번	1ms	17ms	16ms(= 17ms - 1ms)
2번	3ms	8ms	5ms(= 8ms - 3ms)

우선순위 디스크 컨트롤러에서 모든 요청 작업의 반환 시간의 평균은 8ms(= (3ms + 16ms + 5ms) / 3)가 됩니다.

각 작업에 대해 [작업이 요청되는 시점, 작업의 소요시간]을 담은 2차원 정수 배열 `jobs` 가 매개변수로 주어질 때, **우선순위 디스크 컨트롤러**가 이 작업을 처리했을 때 모든 요청 작업의 반환 시간의 평균의 정수부분을 return 하는 solution 함수를 작성해 주세요.

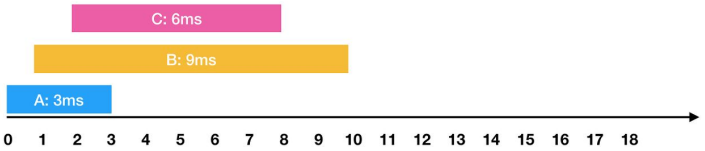
Hint

Step 1. 문제 분석

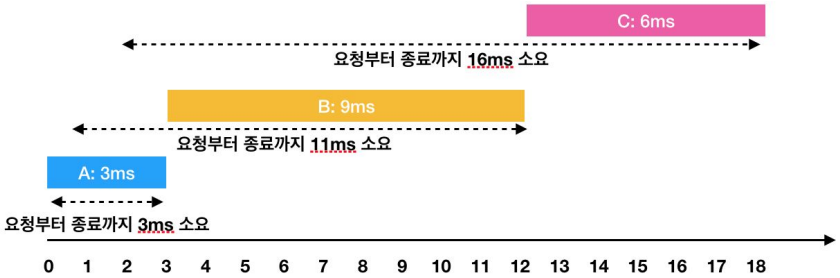
문제 요약

하드디스크는 한 번에 하나의 작업만 처리할 수 있다.
작업은 **[요청 시점, 소요 시간]**으로 주어지며, 디스크 컨트롤러는 다음 우선순위에 따라 작업을 수행한다.

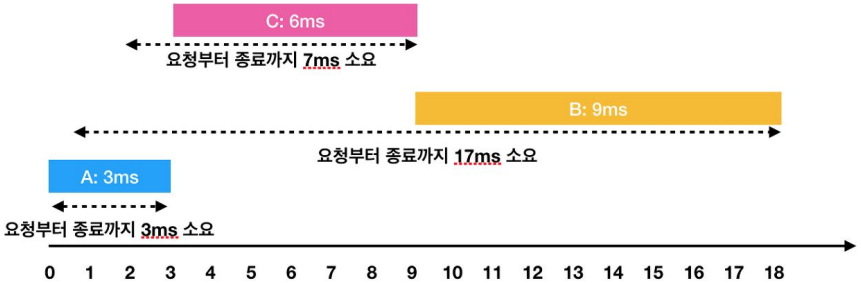
1. **소요 시간이 짧은** 작업 우선
2. **요청 시점이 빠른** 작업 우선
3. **작업 번호가 작은** 작업 우선



모든 작업이 완료될 때까지 각 작업의 **반환 시간(작업 종료 - 요청 시각)**을 구하고,
그 **평균의 정수 부분**을 반환하면 된다.



A →B → C 순서: 평균 10ms((3+11+16)/3)



A → C → B 순서: 평균 9ms((3+7+17)/3)

💡 Hint

Step 1. 문제 분석

💡 문제 유형

- "언제 어떤 작업을 꺼내야 하나?"가 핵심이다.
- **현재 시간(end_time)을 기준**으로, 요청된 작업들 중에서 **소요 시간이 가장 짧은 작업**을 선택해야 한다.
- 이를 위해 **힙(우선순위 큐)** 을 활용한다.
- 대기 큐에 넣는 시점과 꺼내는 시점을 정확히 다루는 것이 중요하다.

💡 Hint

Step 2. 접근 방식

💡 풀이 아이디어

- 최소 힙을 사용하여 **(소요 시간, 요청 시간, 작업 번호)** 순서로 정렬한다.

  ```
  heapq.heappush(heap, (소요시간,  요청시간, 작업번호))
  ```

- 현재 시간(end_time)보다 작거나 같은 요청만 힙에 넣는다.
- 힙이 비어 있으면, 다음 요청 시간으로 end_time을 갱신하여 대기한다.
- 각 작업의 반환 시간 = 종료 시간 - 요청 시간을 누적하여 평균을 구한다.

Hint

참고) https://school.programmers.co.kr/learn/courses/14743/lessons/118893

Step 2. 주의할 점

💡 jobs 정렬 여부

- 배열 `jobs`는 정렬되어 있지 않을 수 있다. 문제에서 따로 정렬되어있다고 언급이 없기 때문에 정렬되어 있다 가정해서는 안 된다!
- 즉, `jobs`를 따로 정렬해주어야 한다.

Hint

참고) https://school.programmers.co.kr/learn/courses/14743/lessons/118893

Step 2. 주의할 점

> 💡 **현재 작업이 끝난 후, 요청된 작업들은 기다릴 필요가 있다!**

- 매 작업을 완료할 때마다 **종료 시점이 계속 연장될 수 있음**에 유의해야 한다.
- 아래 예시처럼 더 먼저 들어온 작업은 0번이지만, 작업 소요 시간이 더 낮은 2번이 먼저 수행될 수 있으며, 2번이 수행되고 나면 작업 종료 시점이 연장되게 된다.
- 따라서 매 작업이 끝나는 시점마다 끝나기 전에 요청된 작업 리스트를 갱신해주어야 한다.

jobs	result
[[7, 8], [3, 5], [9, 6]]	9

작업 번호	요청 시각	작업 종료 시각	반환 시간
1번	3ms	8ms	5ms(=8ms - 3ms)
2번	9 ms	15ms	6ms(=15ms - 9ms)
0번	7ms	23ms	16ms(= 23ms - 7ms)

(5 + 6+ 16) / 3 = 9ms

💡 Hint

Step 2. 접근 방식

💡 코드 플로우

- jobs를 요청 시각 기준으로 정렬한다.
- 반복하면서 다음을 수행한다:
  - 현재 시간보다 먼저 들어온 요청을 모두 힙에 넣는다.
  - 힙에서 소요 시간이 가장 짧은 작업을 꺼낸다.
    - 종료 시간(end_time)을 갱신하고, 반환 시간을 누적한다.
  - 힙이 비었다면, 다음 요청이 도착할 때까지 기다린다 (end_time = jobs[i][0]).

✨Solution

[풀이1](), [풀이2]()

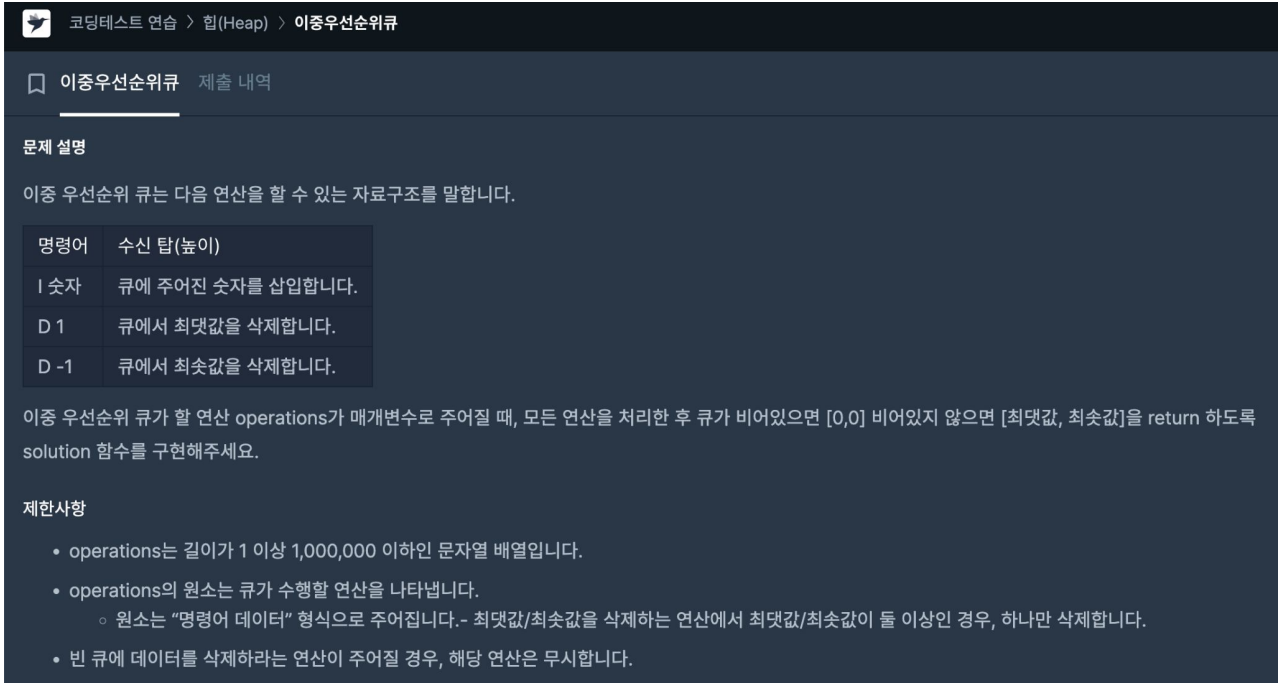
```python
import heapq

def solution(jobs):
    jobs.sort()     # 요청시간 기준 정렬
    job_len = len(jobs)
    i = 0           # jobs 인덱스
    end_time = 0    # 현재 시간
    return_time = 0 # 작업 반환 시간
    count = 0       # 작업 처리한 개수

    heap = []

    while count < job_len:
        # 현재 시간에 요청된 작업 처리
        while i < job_len and jobs[i][0] <= end_time:
            heapq.heappush(heap, (jobs[i][1], jobs[i][0], i))  # 소요시간, 요청시간, 작업번호 순서
            i += 1

        # 대기 큐에 작업이 있다면, 시간을 업데이트한다.
        if len(heap) > 0:
            work_time, start_time, num = heapq.heappop(heap)
            end_time += work_time
            return_time += end_time - start_time
            count += 1
        else:
            # 대기 큐가 비었다면, 다음 작업이 올 때까지 기다려야 한다.
            end_time = jobs[i][0]

    return return_time // job_len
```

```python
import heapq


def solution(jobs):
    answer = 0
    now = 0  # 현재시간
    i = 0  # 처리개수
    start = -1  # 마지막 완료시간
    heap = []

    while i < len(jobs):
        for job in jobs:
            if start < job[0] <= now:
                heapq.heappush(heap, [job[1], job[0]])

        if heap:
            current = heapq.heappop(heap)
            start = now
            now += current[0]
            answer += now - current[1]  # 요청으로부터 처리시간
            i += 1
        else:
            now += 1

    return answer // len(jobs)
```

💻 Assignment

프로그래머스 #이중우선순위큐

힙/우선순위 큐 문제. 최소 힙과 최대 힙을 이용해야 한다.



코딩테스트 연습 〉 힙(Heap) 〉 이중우선순위큐

이중우선순위큐 제출 내역

문제 설명

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 operations가 매개변수로 주어질 때, 모든 연산을 처리한 후 큐가 비어있으면 [0,0] 비어있지 않으면 [최댓값, 최솟값]을 return 하도록 solution 함수를 구현해주세요.

제한사항

- operations는 길이가 1 이상 1,000,000 이하인 문자열 배열입니다.
- operations의 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.