백준 1781번 - 컵라면

백준 1781번 - 컵라면

시간제한	메모리 제한	제출	정답	맞은 사람	정답 비율
2초	256MB	7845	2281	1707	30.829%

문제

상욱 조교는 동호에게 N개의 문제를 주고서, 각각의 문제를 풀었을 때 컵라면을 몇 개 줄 것인지 제시 하였다. 하지만 동호의 찌를듯한 자신감에 소심한 상욱 조교는 각각의 문제에 대해 데드라인을 정하였다.

문제 번호	1	2	3	4	5	6	7
데드라인	1	1	3	3	2	2	6
컵라면 수	6	7	2	1	4	5	1

위와 같은 상황에서 동호가 2, 6, 3, 1, 7, 5, 4 순으로 숙제를 한다면 2, 6, 3, 7번 문제를 시간 내에 풀어 총 15개의 컵라면을 받을 수 있다.

문제는 동호가 받을 수 있는 최대 컵라면 수를 구하는 것이다. 위의 예에서는 15가 최대이다.

문제를 푸는데는 단위 시간 1이 걸리며, 각 문제의 데드라인은 N이하의 자연수이다. 또, 각 문제를 풀 때 받을 수 있는 컵라면 수와 최대로 받을 수 있는 컵라면 수는 모두 231보다 작거나 같은 자연수이다.

입력

첫 줄에 숙제의 개수 N (1 ≤ N ≤ 200,000)이 들어온다. 다음 줄부터 N+1번째 줄까지 i+1번째 줄에 i번째 문제에 대한 데드라인과 풀면 받을 수 있는 컵라면 수가 공백으로 구분되어 입력된다.

출력

첫 줄에 동호가 받을 수 있는 최대 컵라면 수를 출력한다.

예제 입력1

```
7
1 6
1 7
3 2
3 1
2 4
2 5
6 1
```

예제 출력1

```
15
```

출처

출처

알고리즘 분류

- 자료 구조
- 그리디 알고리즘
- 우선순위 큐

접근 방법

현재 시간(t) 보다 크거나 같은 deadline을 갖는 작업 중 가장 가치가 큰 것을 선택하는 방법이 최적해임을 알 수 있다.

직관적으로 생각해보았을 때, t 시간에 처리할 수 있는 작업들($task_1, task_2.., task_k$) 에 대해서 최대 가치를 얻을 수 있는 $task_{max}$ 를 선택하지 않고

다른 작업 $task_i$ 를 선택했을 때, $task_{max}$의 deadline 때문에 $t+\alpha$ 시간에 $task_{max}$ 를 선택하지 못할 수 있다. 이 경우에는, $task_{max}$를 택하는 것이 최적해를 구성하는 것에 있어 무조건적으로 이득이기 때문에 선택한다. 이는 t 시간에 처리할 수 있는 작업들이 $t+\alpha$ ($\alpha > 0$) 시간에 처리할 수 있는 작업들보다 크거나 같기 때문에 성립한다.

그렇다면, 어떻게 이것을 구현할 것인가?

모든 작업들이 단위 시간 1에 처리가 된다는 것을 바탕으로 t = 1, ..., $2^{31}-1$ 까지 고려한다는 것은 시간 복잡도나 공간 복잡도 측면에서 불가능하다.

사실 숙제의 개수 N 은 최대 200,000 이므로 실제 고려해야하는 t는 최대 200,000 이다.

이를 바탕으로 현재 시간에 처리할 수 있는 작업들 중, 최대 가치를 가지는 컵라면 을 찾기 위해서 현재 시간 이후의 `deadline`을 가지는 작업들을 빠르게 찾을 수 있는 동적 쿼리를 구성하는 방법도 생각할 수 있겠다. parametric search로 이를 구현한다고 해도, 탐색된 결과가 최대 N개 일 수 있고, 여기서 최댓값을 찾기 위해서 $O(N)$ 시간이 소요되어 적절한 시간 복잡도내에 문제를 해결하는 것이 불가능하므로 다른 방법을 생각해보아야한다.

관점을 바꾸어 생각해보자.

deadline을 기준으로 작업들을 정렬하고, 1번째 작업부터 N번째 작업까지 순차적으로 고려해보자.

우리가 구하고자 하는 문제는 1번째 작업부터 N번째 작업까지 모두 고려하였을 때, 가치의 최댓값이다.

부분 문제를 이렇게 정의해보자.

`1번째 작업부터 i번째 작업까지 고려하였을 때, 가치의 최댓값`

우리는 $i = 1, \ldots, N$ 까지 해당 특성을 유지만 해준다면 전체 문제의 최적해를 찾을 수 있게된다.

$i = 1$ 일때는 첫번째 작업이 현재 단위시간(=초기값 1)보다 크거나 같다면 넣어준다.

$i = 2$ 일때는 두번째 작업이 현재 단위시간(t = 2) 보다 크거나 같다면 넣어주고, 그렇지 않다면 이전까지의 작업 중 최소 가치의 작업을 빼고 현재 작업을 넣어준다.

문제의 조건상 $O(NlogN)$ 의 시간에 해결하기 위하여 Min-Heap을 통해 구현하였다.

소스코드

```cpp
#define FASTIO cin.tie(0)->sync_with_stdio(false), cout.tie(0)
////////////////////////////////////////////////////////////////
#include <bits/stdc++.h>
typedef long long ll;
using namespace std;
int main(void){
    FASTIO;
////////////////////////////////////////////////////////////////
    int N;
    cin >> N;

    vector<pair<ll,ll>> v(N);
    for(auto &item : v) {
        cin >> item.first >> item.second;
    }
    sort(v.begin(), v.end());
    priority_queue<ll, vector<ll>, greater<>> PQ;
    for(auto [deadline, value] : v) {
        if(PQ.size() + 1 > deadline && PQ.top() < value) {
            PQ.pop();
        } else if (PQ.size() + 1 > deadline && PQ.top() >= value) {
            continue;
        }
        PQ.push(value);
    }

    ll ans = 0LL;
```

```
    while (!PQ.empty()) {
        ans += PQ.top();
        PQ.pop();
    }
    cout << ans << '\n';
    return 0;
}
```