백준 1911번 - 흙길 보수하기

- 문제
- 입력
- 출력
- 예제 입력1
- 예제 출력1
- 출처
- 알고리즘 분류
- 접근 방법
- 소스코드

백준 1911번 - 흙길 보수하기

시간제한	메모리 제한	제출	정답	맞은 사람	정답 비율
2초	128MB	4472	1637	1268	37.593%

문제

어젯밤 겨울 캠프 장소에서 월드 본원까지 이어지는, 흙으로 된 비밀길 위에 폭우가 내려서 N (1 <= N <= 10,000) 개의 물웅덩이가 생겼다. 월드학원은 물웅덩이를 덮을 수 있는 길이 L (L은 양의 정수) 짜리 널빤지들을 충분히 가지고 있어서, 이들로 다리를 만들어 물웅덩이들을 모두 덮으려고 한다. 물웅덩이들의 위치와 크기에 대한 정보가 주어질 때, 모든 물웅덩이들을 덮기 위해 필요한 널빤지들의 최소 개수를 구하여라.

입력

첫째 줄에 N과 L이 들어온다.

둘째 줄부터 N+1번째 줄까지 총 N개의 줄에 각각의 웅덩이들의 정보가 주어진다. 웅덩이의 정보는 웅덩이의 시작 위치와 끝 위치로 이루어진다. 각 위치는 0이상 1,000,000,000이하의 정수이다.

출력

첫째 줄에 모든 물웅덩이들을 덮기 위해 필요한 널빤지들의 최소 개수를 출력한다.

예제 입력1

```
3 3
1 6
13 17
8 12
```

예제 출력1

```
5
```

출처

출처

알고리즘 분류

- 정렬
- 스위핑

접근 방법

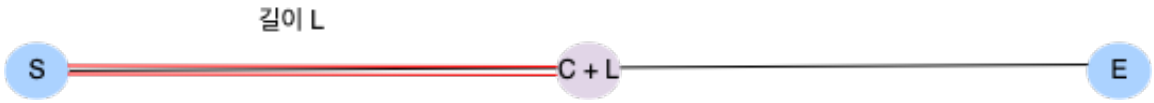
웅덩이의 위치의 범위는 $0 \leq x \leq 10^9$ 이므로, 배열으로 표현하여 마킹하는 방식으로 문제를 해결할 수 없다.

웅덩이를 덮는 순서는 문제의 해를 찾는 것에 영향을 주지 않으므로, 0부터 M 까지 단방향으로 스캔하는 방식으로 문제를 해결해보자.

0 부터 M 까지 단방향으로 지나가면서 특정 웅덩이를 마주친다면 무조건 해당 웅덩이를 덮어야한다. (그렇지 않는다면, 해당 웅덩이는 추후에 덮을 수 없다)

웅덩이의 시작 위치(s) 와 마지막 위치(e) 에 대해서 널판지가 어떠한 방식으로 웅덩이를 덮을 수 있는지 고려하자.

이전까지 c 이하의 모든 웅덩이를 덮은 상황을 가정한다.



첫번째 케이스는 `널판지가 웅덩이의 일부를 덮은` 경우이다. 이 경우에는 추후에 $C + L + 1$ 부터 E 까지의 나머지 웅덩이를 덮어야한다.

두번째 케이스는 `널판지가 웅덩이 전체를 덮고 나머지 영역을 추가적으로 덮은` 경우이다. 이 경우에는 E ~ $C + L$ 까지의 웅덩이도 덮을 수 있다.

첫번째 케이스에서 C 가 새로운 웅덩이의 시작 위치 S 이전이라면 (즉, $C < S$)

$C + 1$ ~ $S - 1$ 까지는 웅덩이가 없으니, C는 S 부터 시작하는 것이 적절하다.

이후, 해당 웅덩이를 덮기 위해서는 $\lceil (e - s + 1)/L \rceil$ 만큼의 널빤지가 필요하다.

그렇다면, $C + L * \lceil (e - s + 1)/L \rceil$ 의 위치까지 스캔한 것이 되고, 이 범위는 두번째 케이스로 수렴한다. (단, 나머지 영역을 추가적으로 덮지 않을 수 있음)

이후에는 C 값과 다음 웅덩이의 시작 위치 S 를 비교한 다음, 널판지로 덮히지 않은 위치($C + 1$) 과 비교하여 최댓값으로 설정한 다음, 탐색을 진행한다.

소스코드

```
#define FASTIO cin.tie(0)->sync_with_stdio(false), cout.tie(0)
////////////////////////////////////////////////////////////////////
#include <bits/stdc++.h>
```

```cpp
using namespace std;
int main(void){
    FASTIO;
////////////////////////////////////////////////////////////////

    int N, L;
    cin >> N >> L;

    priority_queue<pair<int, int>, vector<pair<int,int>>, greater<pair<int,int>>>
PQ;
    for (int i = 0; i < N; i++) {
        int s, e;
        cin >> s >> e;
        e--; // [s, e]
        PQ.push({s, e});
    }

    int c = 0, ans = 0;
    while (!PQ.empty()) {
        auto[s, e] = PQ.top();
        PQ.pop();

        if (s <= c) s = c + 1;
        if (e < s || e <= c) continue;

        // 해당 구간을 덮기 위해서 얼마나 많은 널빤지가 필요?
        // factor = ceil((e - s + 1) / L) --> c = s + factor * L - 1
        int factor = ceil((e - s + 1) / (double)L);
        c = s + factor * L - 1;
        ans += factor;
    }
    cout << ans;
    return 0;
}

```