1202. 보석 도둑

출처	BoJ
실습/과제/자율	자율
문제유형	그리디
날짜(처음 풀이)	@2024년 10월 10일
풀이 언어	Java

https://www.acmicpc.net/problem/1202

https://www.acmicpc.net/problem/1202

1. 문제 요약

상덕이가 훔칠 수 있는 보석의 **최대 가격**을 구하는 프로그램을 작성하시오.

- 상덕이가 털 보석점에는 **보석**이 총 N개 있다.
  - 각 보석은 무게 Mi와 가격 Vi를 가지고 있다.
- 상덕이는 **가방**을 K개 가지고 있다.
  - 각 가방에 담을 수 있는 최대 무게는 Ci이다.
  - 가방에는 최대 한 개의 보석만 넣을 수 있다.

2. 문제 입출력

입력

- 첫째 줄에 N과 K가 주어진다.
- 다음 N개 줄에는 각 보석의 정보 Mi와 Vi가 주어진다.
- 다음 K개 줄에는 가방에 담을 수 있는 최대 무게 Ci가 주어진다.

입력예시

```
2 1
5 10
100 100
11
```

출력

- 첫째 줄에 상덕이가 훔칠 수 있는 보석 가격의 합의 최댓값을 출력한다.

출력예시

```
10
```

📣 3. 제한사항

- **시간** : 1초
- **메모리** : 256MB

- 1 ≤ N, K ≤ 300,000
- 0 ≤ Mi, Vi ≤ 1,000,000
- 1 ≤ Ci ≤ 100,000,000
- 모든 숫자는 양의 정수이다.

🧐 4. 접근법

- 가방의 개수와 담을 수 있는 최대 무게가 정해져 있다.
- 각 가방에는 최대 1개의 보석만 담을 수 있다.
- 각 보석은 무게와 가격을 가진다.
- 최대 가격을 만들어야 한다.

⇒ 그리디

⏱️ 5. 시간복잡도

우선순위 큐의 삽입, 삭제 시간 복잡도 ⇒ O(logN)

⇒ 총 시간복잡도 : O(NlogN)

💻 6. 코드

```
import java.io.*;
import java.util.*;
```

```java
public class Main {
    static class Gem implements Comparable<Gem> {
        int weight;
        int value;

        Gem(int weight, int value) {
            this.weight = weight;
            this.value = value;
        }

        public int compareTo(Gem other) {
            return this.weight - other.weight;
        }
    }

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        StringTokenizer st = new StringTokenizer(br.readLine());

        int N = Integer.parseInt(st.nextToken()); // 보석 수
        int K = Integer.parseInt(st.nextToken()); // 가방 수

        Gem[] gems = new Gem[N];
        for (int i = 0; i < N; i++) {
            st = new StringTokenizer(br.readLine());
            int Mi = Integer.parseInt(st.nextToken()); // 보석 무게
            int Vi = Integer.parseInt(st.nextToken()); // 보석 가격
            gems[i] = new Gem(Mi, Vi);
        }

        int[] bags = new int[K];
        for (int i = 0; i < K; i++) {
            bags[i] = Integer.parseInt(br.readLine()); // 가방 최대 무게
        }

        // 보석&가방 무게 순으로 정렬
        Arrays.sort(gems);
        Arrays.sort(bags);

        PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());
        long totalPrice = 0;

        for (int i = 0; i < K; i++) {
            int bagCapacity = bags[i];

            for (idx=0; idx < N; idx++) {
                if (gems[idx].weight <= bagCapacity) {
                    pq.offer(gems[idx].value);
                } else {
                    break;
                }
            }

            if (!pq.isEmpty()) {
                totalPrice += pq.poll(); // 가방에 넣을 수 있는 보석 중 가장 큰 금액 더하기
            }
        }
```

```
        System.out.println(totalPrice);
    }
}
```