



#1202. 보석 도둑

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

25.02.03



Problem 시간: 1시간 15분

보석 도둑 성공 다국어

#1202. 보석 도둑

2 골드 II

☆ 한국어 ▾

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	84733	20796	14392	22.748%

문제

세계적인 도둑 상덕이는 보석점을 털기로 결심했다.

상덕이가 털 보석점에는 보석이 총 N 개 있다. 각 보석은 무게 M_i 와 가격 V_i 를 가지고 있다. 상덕이는 가방을 K 개 가지고 있고, 각 가방에 담을 수 있는 최대 무게는 C_i 이다. 가방에는 최대 한 개의 보석만 넣을 수 있다.

상덕이가 훔칠 수 있는 보석의 최대 가격을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N 과 K 가 주어진다. ($1 \leq N, K \leq 300,000$)

다음 N 개 줄에는 각 보석의 정보 M_i 와 V_i 가 주어진다. ($0 \leq M_i, V_i \leq 1,000,000$)

다음 K 개 줄에는 가방에 담을 수 있는 최대 무게 C_i 가 주어진다. ($1 \leq C_i \leq 100,000,000$)

모든 숫자는 양의 정수이다.

출력

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

상덕이는 보석점을 털기로 했다.

보석은 총 N 개가 있고, 각각 무게 M_i 와 가격 V_i 를 가진다.

상덕이는 최대 K 개의 가방을 사용할 수 있으며, 각 가방의 최대 수용 무게는 C_i 이다.

각 가방에는 최대 한 개의 보석만 넣을 수 있다.

상덕이가 훔칠 수 있는 보석의 가격 합의 최댓값을 구하라.

예제 입력 1 복사

예제 출력 1 복사

2 1 N, K

5 10
100 100

10

N개의 줄
무게 M , 가격 V

11

K개의 줄
가방에 담을 수 있는
최대 무게 C

예제 입력 2 복사

3 2

1 65
5 23
2 99

예제 출력 2 복사

164

10
2

최대 무게 C



Problem

Step 1. 우선순위 큐 (Priority Queue) 개념 정리

① 우선순위 큐란?



우선순위 큐

값의 크기에 따라 먼저 나오는 순서가 결정되는 자료구조

- 일반적인 큐(Queue)는 **FIFO**(First In, First Out, 선입선출) 방식
- **우선순위 큐(Priority Queue)** 는 **값이 큰(혹은 작은) 순서**대로 먼저 처리



활용 예시

- 응급실 대기 시스템 (위급한 환자 먼저 치료)
- 프린터 작업 스케줄링 (긴급 문서 먼저 출력)
- 네트워크 패킷 전송 (높은 우선순위 데이터 먼저 전송)
- 알고리즘 문제 해결 (가장 유리한 값 선택)
- 다익스트라 알고리즘(최단 경로)
- 프림 알고리즘(최소 신장 트리)
- 탑 K 문제(가장 큰/작은 K개의 값 찾기)

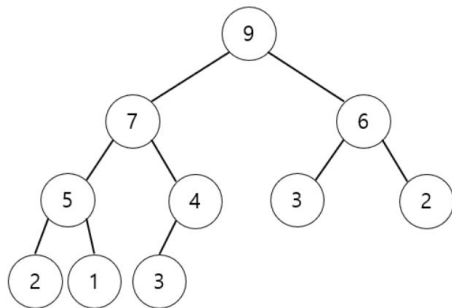


Problem

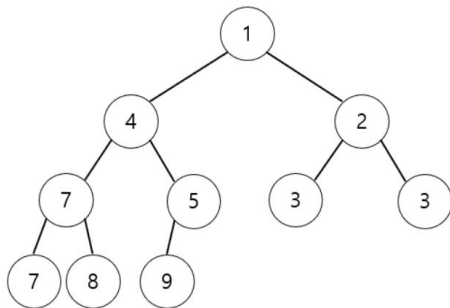
Step 1. 우선순위 큐 (Priority Queue) 개념 정리

② Heap (힙) 이란?

이진 트리 형태의 자료구조로 우선순위 큐로 구현할 때 주로 활용



최대 힙(max heap)



최소 힙(min heap)

힙 종류	설명	루트 위치
최대 힙 (Max Heap)	부모 노드가 자식 노드보다 크거나 같음	가장 큰 값이 루트에 위치
최소 힙 (Min Heap)	부모 노드가 자식 노드보다 작거나 같음	가장 작은 값이 루트에 위치



③ 왜 우선순위 큐에서 힙을 사용할까?

힉스의 장점

- [illegible]



Hint

Step 2. Python `heapq` 모듈 사용

📌 Python에서는 기본적으로 `heapq`는 “최소 힙”을 지원

📌 최대 힙을 만들려면 음수를 활용해야 함

연산	설명
<code>heapq.heapify(iterable)</code>	리스트를 힙으로 변환 ($O(N)$)
<code>heapq.heappush(heap, item)</code>	힙에 원소 추가 ($O(\log N)$)
<code>heapq.heappop(heap)</code>	힙에서 가장 작은 원소 제거 및 반환 ($O(\log N)$)
<code>heapq.heapreplace(heap, item)</code>	가장 작은 원소 제거 후, 새로운 원소 추가 ($O(\log N)$)
<code>heapq.heappushpop(heap, item)</code>	새로운 원소 추가 후, 가장 작은 원소 제거 ($O(\log N)$)



Hint

Step 2. Python `heapq` 모듈 사용



`heapq` 모듈 기본 사용법(최소 힙)

```
import heapq

heap = []
heapq.heappush(heap, 10)
heapq.heappush(heap, 5)
heapq.heappush(heap, 30)

print(heapq.heappop(heap)) # 5
print(heapq.heappop(heap)) # 10
print(heapq.heappop(heap)) # 30
```



최대 힙 구현 (음수 변환)

```
import heapq

max_heap = []
heapq.heappush(max_heap, -10)
heapq.heappush(max_heap, -5)
heapq.heappush(max_heap, -30)

print(-heapq.heappop(max_heap)) # 30
print(-heapq.heappop(max_heap)) # 10
print(-heapq.heappop(max_heap)) # 5
```



Hint

Step 3. 보석 도둑 문제에서의 힙 활용



문제 핵심 아이디어

- 보석과 가방을 정렬
 - 보석: 무게 기준 오름차순 정렬
 - 가방: 무게 기준 오름차순 정렬
- 우선순위 큐(최대 힙) 사용
 - 가방의 크기가 작은 순서대로 보석을 넣을 수 있는지 확인.
 - 현재 가방에 넣을 수 있는 보석을 **max-heap(최대 힙)** 으로 관리하여 **가장 비싼 보석을 선택**.

예제 입력 2 복사

예제 출력 2 복사

3 2
1 65
5 23
2 99
10
2

164

[보석 목록]: (무게, 가격)

[(1, 65), (2, 99), (5, 23)] → 무게 기준 정렬

[가방 목록]: [2, 10] → 무게 기준 정렬



가방이 작은 순서대로 처리되므로,
현재 가방에 넣을 수 있는 보석 중 가장 비싼 것을 선택

단계	가방 무게	추가된 보석들	최대 힙 상태	선택된 보석
①	2kg	(1, 65), (2, 99)	[-99, -65]	99
②	10kg	기존 보석 (1, 65), 새로운 보석 (5, 23) 추가	[-65, -23]	65



Hint

Step 4. 코드 플로우

1. 입력받기

- N : 보석 개수, K : 가방 개수
- 보석 리스트 (M_i, V_i) 입력받기 → 무게 기준 정렬
- 가방 리스트 C_i 입력받기 → 무게 기준 정렬

2. 우선순위 큐(최대 힙) 사용

- 각 가방을 검사하면서, 넣을 수 있는 보석들을 **최대 힙**에 저장
- 현재 가방에 넣을 수 있는 **가장 비싼 보석**을 선택하여 담음

3. 결과 출력

- 상덕이가 훔칠 수 있는 보석 가격의 최댓값을 출력

Solution

<https://bio-info.tistory.com/195>

📌 코드 흐름

1. 보석과 가방을 정렬
 - 보석을 (무게, 가격) 기준으로 무게 오름차순 정렬.
 - 가방을 무게 오름차순 정렬.
2. 각 가방에 대해 가능한 보석을 최대 힙에 추가
 - 현재 가방(bag)에 넣을 수 있는 보석을 최대 힙에 `heapq.heappush(max_heap, -가격)` 형태로 넣음.
 - `heapq`는 최소 힙이므로, 가격을 넣어 최대 힙처럼 사용.
3. 가장 가치가 높은 보석을 선택
 - `heapq.heappop(max_heap)`을 사용해 현재 가방에 넣을 수 있는 가장 비싼 보석을 선택.

📌 주요 연산 및 시간 복잡도

- 정렬: $O(K \log K)$
- 가방을 처리하면서 힙에 보석 추가 & 제거
 - `heappush` → 최악의 경우 $O(N \log N)$
 - `heappop` → $O(K \log N)$
- 최종적으로 $O((N + K) \log N)$ 가 됨.

```
import heapq
```

```
# Step 1: 입력 받기
```

```
N, K = map(int, input().split()) # N: 보석 개수, K: 가방 개수
```

```
jewelry = [] # (무게, 가격)
```

```
bags = [] # 가방의 최대 무게
```

```
for _ in range(N):
```

```
    M, V = map(int, input().split())
```

```
    jewelry.append((M, V))
```

```
for _ in range(K):
```

```
    bags.append(int(input()))
```

```
# Step 2: 보석과 가방 정렬
```

```
jewelry.sort() # 보석을 무게 기준으로 정렬
```

```
bags.sort() # 가방을 무게 기준으로 정렬
```

```
# Step 3: 우선순위 큐 (최대 힙) 사용
```

```
max_heap = []
```

```
result = 0
```

```
idx = 0
```

```
# 가방을 하나씩 처리
```

```
for bag in bags:
```

```
    # 현재 가방이 수용할 수 있는 보석을 모두 추가 (무게 기준 정렬된 상태)
```

```
    while idx < N and jewelry[idx][0] <= bag:
```

```
        heapq.heappush(max_heap, -jewelry[idx][1]) # 최대 힙을 위해 음수 저장
```

```
        idx += 1
```

```
# 가장 가치가 높은 보석을 선택
```

```
if max_heap:
```

```
    result += -heapq.heappop(max_heap)
```

```
print(result)
```



Assignment

백준 #19638. 센티와 마법의 뽕망치(실버1)

자료구조, 우선순위 큐

발제 문제에서 사용한 우선순위 큐를 활용하여 연습하기 좋은 문제

1 19638번 제출 맞힌 사람 소트킹 재채점 결과 채점 현황 내 제출 질문 게시판

센티와 마법의 뽕망치



1 실버 I

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	1024 MB	5554	1871	1464	33.243%