문제 7579 앱 DP + knapsack

1차원 DP로 풀기

메모리를 DP로 잡기에는 배열 용량이 너무 커진다. size = 10000001

⇒생각의 전환 비용을 DP로 잡는다. size = 10001

메모리 관점 : M 이상의 메모리를 제거했을 때 최소비용을 구한다.

⇓

비용 관점 : index 비용으로 제거할 수 있는 최대 메모리를 구한다.

→ M 이상의 메모리를 제거한 index 중 최소 index 를 구한다.

int m[100] : i 번째 앱의 메모리 저장 int c[100] : i 번째 앱의 비활성화 비용 저장

int cache[10001] : i 비용으로 제거한 메모리 저장

int size : 모든 앱을 비활성화 했을 때의 비용 (반복문 범위 지정 역할)

Knapsack 함수 구현

① size 계산 : 모든 앱의 비활성화 비용을 더해서 구한다.

② 점화식: cache[j] = max(cache[j], cache[j-c[i]] + m[i])

- 기존 계산된 메모리와 j-c[i] 비용에서 계산된 메모리에 m[i]를 더한 값 중, 더 큰 값 저장

j 비용으로 제거한 메모리 j 비용으로 제거한 메모리

③ 반복문 구현 :
```
for (i=0 ~ N-1 ; i++)
    for (j=size; j>=c[i]; j--)
```

③ 반복문 구현 : for (i=0 ~ N-1 ,;i++)
for (j=size; j≥ c[i]; j--)

- j는 반드시 내림차순으로 구현되어야 한다.

※ 오름차순 구현 시 이미 m[i] 이 더해진 계산 결과를 다음 점화식 호출 때 또 m[i]가 중복으로 더해진다

이미 비활성화 된 앱을 또 종료시키게 된다. ex) cache[1] = cache[0] + 2

⇒ cache[1]은 ; 앱이 종료된 상태

cache[2] = cache[1] + 2

⇒ 이미 ; 앱이 종료된 cache[1]에 또 +2를 한다.

내림차순 구현시 점화식 계산시에 아직 m[i] 가 더해지지 않은 값을 가져오기 때문에
중복으로 계산되지 않는다.

ex) cache[2] = cache[1] + 2

⇒ cache[2]는 ; 앱이 종료 됐지만, cache[1]에서는 종료 X

cache[1] = cache[0] + 2

⇒ 여기서 cache[1] 에 ; 앱이 종료된다

또는 Dp를 2차원 배열로 구현하는 방법이 있다: ⇒ 오름차순 가능

④ 정답 출력: M 이상의 메모리를 제거한 index 들 중 최솟값 출력