백준 2294번 - 동전 2

백준 2294번 - 동전 2

시간제한	메모리 제한	제출	정답	맞은 사람	정답 비율
1초	128MB	49861	14658	10251	28.741%

문제

n가지 종류의 동전이 있다. 이 동전들을 적당히 사용해서, 그 가치의 합이 k원이 되도록 하고 싶다. 그러면서 동전의 개수가 최소가 되도록 하려고 한다. 각각의 동전은 몇 개라도 사용할 수 있다.

사용한 동전의 구성이 같은데, 순서만 다른 것은 같은 경우이다.

입력

첫째 줄에 n, k가 주어진다. (1 ≤ n ≤ 100, 1 ≤ k ≤ 10,000) 다음 n개의 줄에는 각각의 동전의 가치가 주어진다. 동전의 가치는 100,000보다 작거나 같은 자연수이다. 가치가 같은 동전이 여러 번 주어질 수도 있다.

출력

첫째 줄에 사용한 동전의 최소 개수를 출력한다. 불가능한 경우에는 -1을 출력한다.

예제 입력1

```
3 15
1
5
12
```

예제 출력1

```
3
```

출처

출처

알고리즘 분류

- 다이나믹 프로그래밍

접근 방법

동전($v_1, ..., v_n$) 을 각각 $a_1, ..., a_n$ 개 쓴다고 할 때, $a_1 * v_1 + ... a_n * v_n = K$ 를 구성하는 $a_1 + ... + a_n$ 의 최솟값을 구하는 문제이다.

잘 살펴보면, $a_1 + ... + a_n$ 가 최솟값이 될 경우, $a_1 + ... + a_{n-1}$ 또한 최솟값이 되어야한다.

이는 $a_1 * v_1 + ... + a_{n-1} * v_{n-1} = K - \alpha$ 라는 가치를 형성한다고 하였을 때, $a_1 + ... a_n$ 이 최솟값이 아닐 경우 발생하는 모순을 보이므로서 쉽게 증명할 수 있다.

즉, 해당 문제는 최적 부분 구조를 가진다고 볼 수 있다.

이러한 특성을 활용하여 문제를 다음과 같이 분할해보자.

$Prob_i :=\ [0, i]$ 까지의 동전을 사용하여 β 라는 가치를 얻기위해 필요한 동전의 최소 개수

전체문제는 $Prob_{n-1}$ 이 되고, 이러한 상태를 2차원 배열로 표현한다면 아래와 같이 표현할 수 있다.

$dp[i][j] :=$ `i번째 동전까지 고려하였을 때, j라는 가치를 얻기 위해 필요한 동전의 최소 개수`

$dp[i]$ 의 각각의 가치를 구성하는 부분해가 $dp[i+1]$ 의 부분해에 포함될 수 있으므로(최적 부분 구조) Recurrence Relation을 고려할 수 있다.

Case1. i번째 동전만으로 최적해를 구성하는 경우

- $dp[i][j] =$ 사용한 i번째 동전의 개수(count) $(if, j = count * value[i])$

Case2. 이전(0~i-1)번째 동전으로 구성된 가치에서 현재 동전을 추가하여 최적해를 구성하는 경우

- $dp[i][j] = dp[i-1][j-\alpha] +$ 사용한 i번째 동전의 개수(count) $(if, count * value[i] = \alpha)$

이 2가지 경우를 하나로 합친 점화식은 아래와 같이 정의할 수 있다.

```
// i번째 동전까지 고려하였을 때, 가치 j에 대해 고려
cnt = 0;
while(j - cnt * value[i] >= 0){
    dp[i][j] = min(dp[i][j], dp[i-1][j - cnt * value[i]] + cnt);
    cnt++;
}
// cnt는 i번째 동전을 사용한 횟수.
```

이렇게 점화식을 정의한다면, base condition은 첫번째 동전에 대해서 고려한 아래가 된다.

```
cnt = 0;
while(value[0] * cnt <= K) {
    dp[i][value[0]*cnt] = cnt;
    cnt++;
}
```

위와 같이 구현할 경우 시간 복잡도는 $O(N * K * \beta)$ 가 된다. (β 는 $j - val[i] * cnt$ 에 비례)

최악의 경우는 $val[i] = 1$ 일 경우일 것이다. 이 경우, 시간 초과를 받을 수 있는데 중복되는 $val[i]$값에 대해 중복을 제거하는 것으로 실제 수행 시간을 매우 크게 줄일 수 있다.

아래의 소스코드는 위 테크닉을 활용한 것이다.

(사실, 모든 가치 j=[0, k] 까지 고려하여 값을 채우는 것 보단, 현재 동전의 가치의 배수로 테이블을 채우는 방법이 depth를 1단계 낮추기 때문에 더 좋은 방법이다..)

소스코드

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

    int n, k;
    cin >> n >> k;

    vector<int> val(n);
    for(auto &elem : val) {
        cin >> elem;
    }
    sort(val.begin(), val.end());
    val.erase( unique(val.begin(), val.end()), val.end());

    int dp[105][10005], cnt = 0;
    fill(&dp[0][0], &dp[104][10004], 1e9);
    while (val[0] * cnt <= k) {
        dp[0][val[0] * cnt] = cnt;
        cnt++;
    }
    n = val.size();
    for (int i = 1; i < n; i++) {
        // 가치 j에 대해
        for (int j = 0; j <= k; j++) {
            cnt = 0;
            while(j - val[i] * cnt >= 0) {
```

```
                dp[i][j] = min(dp[i][j], dp[i - 1][j - val[i] * cnt] + cnt);
                cnt++;
            }
            // 또는 현재 동전만으로 구성
            cnt = 0;
            while (val[i] * cnt <= k) {
                dp[i][val[i] * cnt] = min(dp[i][val[i] * cnt], cnt);
                cnt++;
            }
        }
    }
    if (dp[n - 1][k] == 1e9) {
        dp[n - 1][k] = -1;
    }
    cout << dp[n-1][k] << '\n';

    return 0;
}
```