2022-2 HI-ARC 중급스터디

2주차. DP

이지은 (leeju1013)

목차

피보나치 수

성공

브론즈 III

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초 (추가 시간 없음)	128 MB	50788	23816	19438	48.230%

문제

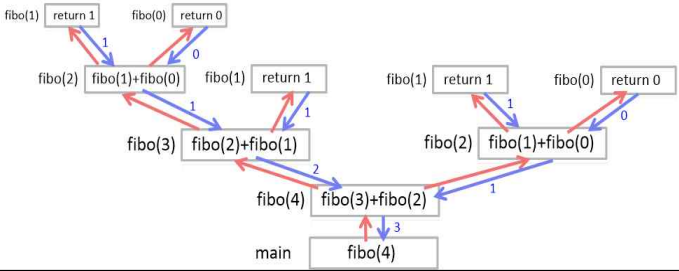
피보나치 수는 0과 1로 시작한다. 0번째 피보나치 수는 0이고, 1번째 피보나치 수는 1이다. 그 다음 2번째 부터는 바로 앞 두 피보나치 수의 합이 된다.

이를 식으로 써보면 $F_n = F_{n-1} + F_{n-2}$ ($n \geq 2$)가 된다.

n=17일때 까지 피보나치 수를 써보면 다음과 같다.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597

n이 주어졌을 때, n번째 피보나치 수를 구하는 프로그램을 작성하시오.



입력

첫째 줄에 n이 주어진다. n은 45보다 작거나 같은 자연수이다.

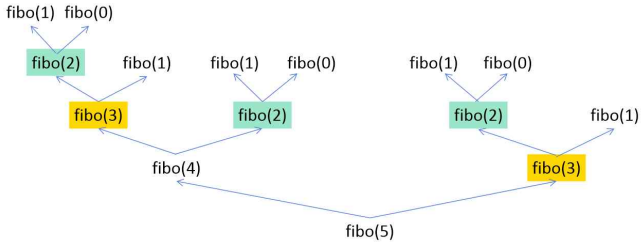
출력

첫째 줄에 n번째 피보나치 수를 출력한다.

```cpp
#include <iostream>
using namespace std;

int fibo(int n){
    if(n==0) return 0;
    else if(n==1) return 1;
    else return fibo(n-1)+fibo(n-2);
}
int main(){
    ios_base::sync_with_stdio(false);

    int n; cin>>n;
    cout<<fibo(n);
    return 0;
}
```



제출 번호	아이디	문제	문제 제목	결과	메모리	시간	언어	코드 길이
40355570	leeju1013	2747	피보나치 수	시간 초과			C++14	362 B

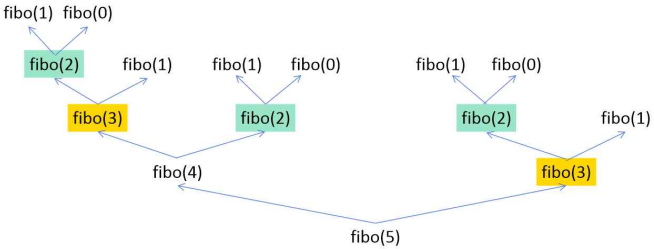
```cpp
#include <iostream>
using namespace std;
int dp[46];
int fibo(int n){
    if(n==0) return 0; //또는 return dp[0];
    if(n==1) return 1; //또는 return dp[1] ;
    if(dp[n]!=-1) return dp[n]; //계산 한 적이 있다면
    return dp[n] = fibo(n-1)+fibo(n-2);
}
int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL);

    int n; cin>>n;
    fill(dp, dp+n+1, -1); //배열 dp를 -1로 초기화
    dp[0]=0, dp[1]=1;

    cout<<fibo(n);
    return 0;
}
```

idx	0	1	2	3	4	5
dp	0	1	1	2	3	5

idx	0	1	2	3	4	5
dp	0	1	-1	-1	-1	-1



```cpp
#include <iostream>
using namespace std;
int dp[46];
int fibo(int n){
    if(n==0) return 0;
    if(n==1) return 1;
    if(dp[n]) return dp[n];
    return dp[n] = fibo(n-1)+fibo(n-2);
}
int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL);

    int n; cin>>n;
    cout<<fibo(n);
    return 0;
}
```

탑다운(top-down)

: 재귀 호출 사용

```cpp
#include <iostream>
using namespace std;
int dp[46];
int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL);

    int n; cin>>n;
    dp[0]=0, dp[1]=1;
    for(int i=2; i<=n; i++){
        dp[i] = dp[i-1] + dp[i-2];
    }
    cout<< dp[n];
    return 0;
}
```

바텀업(bottom-up)

: 반복문 사용

3 2747번 제출

피보나치 수

3 브론즈 III

시간 제한

1 초 (추가 시간 없음)

입력

첫째 줄에 n이 주어진다. n은 45보다 작거나 같은 자연수이다.

```cpp
#include <iostream>
using namespace std;

int fibo(int n){
    if(n==0) return 0;
    else if(n==1) return 1;
    else return fibo(n-1)+fibo(n-2);
}
int main(){
    ios_base::sync_with_stdio(false);

    int n; cin>>n;
    cout<<fibo(n);
    return 0;
}
```

대략 $O(2^N)$

```cpp
#include <iostream>
using namespace std;
int dp[46];
int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL);

    int n; cin>>n;
    dp[0]=0, dp[1]=1;
    for(int i=2; i<=n; i++){
        dp[i] = dp[i-1] + dp[i-2];
    }
    cout<< dp[n];
    return 0;
}
```

$O(N)$

1. DP (Dynamic Programming)

- What?
  - 전체 문제를 작은 부분 문제로 나누어 푸는 방법
  - 중복되는 부분 문제는 한 번만 계산

- When?
  - 최적 부분 구조

- How?
  - 메모이제이션
  - 탑다운 / 바텀업

1. DP

```cpp
#include <iostream>
using namespace std;
int dp[46];
int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL);

    int n; cin>>n;
    dp[0]=0, dp[1]=1;
    for(int i=2; i<=n; i++){
        dp[i] = dp[i-1] + dp[i-2];
    }
    cout<< dp[n];
    return 0;
}
```

// DP 테이블 정의. dp[i] = i번째 피보나치 수

// 초기값 설정

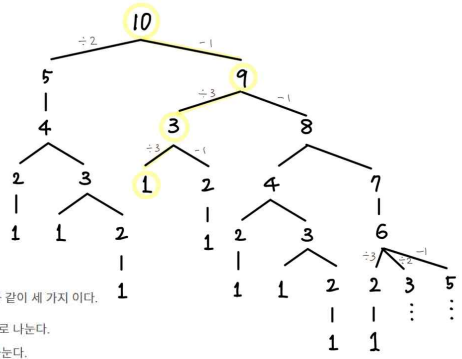
// 점화식 찾기★

3 1463번 제출 맞힌 사람

1로 만들기

실버 III

시간 제한	메모리 제한
0.15 초 (하단 참고)	128 MB



문제

정수 X에 사용할 수 있는 연산은 다음과 같이 세 가지 이다.

1. X가 3으로 나누어 떨어지면, 3으로 나눈다.
2. X가 2로 나누어 떨어지면, 2로 나눈다.
3. 1을 뺀다.

정수 N이 주어졌을 때, 위와 같은 연산 세 개를 적절히 사용해서 1을 만들려고 한다. 연산을 사용하는 횟수의 최솟값을 출력하시오.

입력

첫째 줄에 1보다 크거나 같고, 10^6보다 작거나 같은 정수 N이 주어진다.

출력

첫째 줄에 연산을 하는 횟수의 최솟값을 출력한다.

예제 입력 1 복사

```
2
```

예제 출력 1 복사

```
1
```

예제 입력 2 복사

```
10
```

예제 출력 2 복사

```
3
```

힌트

10의 경우에 10 -> 9 -> 3 -> 1 로 3번 만에 만들 수 있다.

정수 X에 사용할 수 있는 연산은 다음과 같이 세 가지 이다.

1. X가 3으로 나누어 떨어지면, 3으로 나눈다.
2. X가 2로 나누어 떨어지면, 2로 나눈다.
3. 1을 뺀다.

정수 N이 주어졌을 때, 위와 같은 연산 세 개를 적절히 사용해서 1을 만들려고 한다. 연산을 사용하는 횟수의 최솟값을 출력하시오.

1. DP 테이블 정의

   dp[x] = x를 1로 만드는 연산 횟수의 최솟값

2. 점화식 찾기

   dp[x] = min(dp[x/3], dp[x/2], dp[x-1]) + 1

   x가 3으로 나누어 떨어질 때. x가 2로 나누어 떨어질 때.

3. 초기값 설정

   dp[1] = 0

예제 입력 2 복사

```
10
```

예제 출력 2 복사

```
3
```

힌트

10의 경우에 10 -> 9 -> 3 -> 1 로 3번 만에 만들 수 있다.

```cpp
#include <iostream>
using namespace std;
int dp[1000001];
int func(int x){
    if(x==1) return 0; //base case : dp[1]=0
    if(dp[x]!=-1) return dp[x]; //이미 계산했던거

    int result = func(x-1)+1;
    if(x%3==0) result = min(result, func(x/3)+1);
    if(x%2==0) result = min(result, func(x/2)+1);
    return dp[x]=result;
}
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);

    int n; cin>>n;
    fill(dp, dp+n+1, -1);
    cout<<func(n);

    return 0;
}
```

```cpp
#include <iostream>
using namespace std;
int dp[1000001];

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);

    int n; cin>>n;
    dp[1]=0;
    for(int i=2; i<=n; i++){
        dp[i]=dp[i-1]+1;
        if(i%3==0) dp[i]=min(dp[i], dp[i/3]+1);
        if(i%2==0) dp[i]=min(dp[i], dp[i/2]+1);
    }
    cout<<dp[n];
    return 0;
}
```

1. DP 테이블 정의

   dp[x] = x를 1로 만드는 연산 횟수의 최솟값

2. 점화식 찾기

   dp[x] = min(dp[x/3]+1, dp[x/2]+1, dp[x-1]+1)

   x가 3으로 나누어 떨어질 때. x가 2로 나누어 떨어질 때.

3. 초기값 설정

   dp[1] = 0

탑다운(top-down)
: 재귀 호출 사용

바텀업(bottom-up)
: 반복문 사용

가장 긴 증가하는 부분 수열

성공

실버 II

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	98771	38738	25401	37.177%

문제

수열 A가 주어졌을 때, 가장 긴 증가하는 부분 수열을 구하는 프로그램을 작성하시오.

예를 들어, 수열 A = {10, 20, 10, 30, 20, 50} 인 경우에 가장 긴 증가하는 부분 수열은 A = {**10**, **20**, 10, **30**, 20, **50**} 이고, 길이는 4이다.

입력

첫째 줄에 수열 A의 크기 N (1 ≤ N ≤ 1,000)이 주어진다.

둘째 줄에는 수열 A를 이루고 있는 A_i가 주어진다. (1 ≤ A_i ≤ 1,000)

출력

첫째 줄에 수열 A의 가장 긴 증가하는 부분 수열의 길이를 출력한다.

예제 입력 1 복사

```
6
10 20 10 30 20 50
```

예제 출력 1 복사

```
4
```

수열 A가 주어졌을 때, 가장 긴 증가하는 부분 수열을 구하는 프로그램을 작성하시오.

예를 들어, 수열 A = {10, 20, 10, 30, 20, 50} 인 경우에 가장 긴 증가하는 부분 수열은 A = {**10**, **20**, 10, **30**, 20, **50**} 이고, 길이는 4이다.

* 가장 긴 증가하는 부분 수열 (Longest Increasing Subsequence)
: 부분 수열(연속하지 않아도 됨) 중 오름차순으로 정렬된 가장 긴 수열

1. DP 테이블 정의
   dp[i] = arr[i]로 끝나는 LIS 길이

idx	0	1	2	3	4	5
arr	**10**	**20**	10	**30**	20	**50**
dp	1	2	1	3	2	4

2. 점화식 찾기
   dp[i] = arr[j] (0<=j<i) 중에서 arr[j]<arr[i]인 수들 중 dp[j]의 최댓값 +1

3. 초기값 설정
   dp[0] = 1

```cpp
#include <iostream>
using namespace std;

int arr[1001]; //수열A 저장할 배열 선언
int dp[1001]; //dp[i]는 arr[i]로 끝나는 LIS길이

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    //변수 n 선언, ans 선언, n 입력받기
    //배열 arr 입력받기

    //dp배열 초기값 설정
    for(int i=1; i<n; i++){ //dp배열 바텀업 방식으로 채우기
        //dp[i] 초기화
        for(int j=0; j<i; j++){ //앞에있는 수들 보기
            //arr[j]<arr[i]이고 dp[i]<dp[j]+1이면 dp[i]값 갱신
        }
        //ans에 dp[i] 최대값 갱신
    }
    //ans 출력
    return 0;
}
```

idx	0	1	2	3	4	5
arr	**10**	**20**	10	**30**	20	**50**
dp	1	2	1	3	2	4

```cpp
#include <iostream>
using namespace std;

int arr[1001]; //수열A 저장
int dp[1001]; //dp[i]는 arr[i]로 끝나는 LIS길이

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    int n,ans=1; cin>>n;
    for(int i=0; i<n; i++) cin>>arr[i];

    dp[0]=1;
    for(int i=1; i<n; i++){
        dp[i]=1;
        for(int j=0; j<i; j++){
            if(arr[j]<arr[i]) dp[i]=max(dp[i],dp[j]+1);
        }
        ans=max(ans,dp[i]);
    }
    cout<<ans;
    return 0;
}
```

쉬는 시간

1. DP
   - 2747. 피보나치 수
   - 1463. 1로 만들기
   - 11053. 가장 긴 증가하는 부분 수열
   - 9251. LCS
   - 2294. 동전2

LCS 성공

골드 V

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
0.1 초 (하단 참고)	256 MB	55332	22364	16446	40.265%

문제

LCS(Longest Common Subsequence, 최장 공통 부분 수열)문제는 두 수열이 주어졌을 때, 모두의 부분 수열이 되는 수열 중 가장 긴 것을 찾는 문제이다.

예를 들어, ACAYKP와 CAPCAK의 LCS는 ACAK가 된다.

입력

첫째 줄과 둘째 줄에 두 문자열이 주어진다. 문자열은 알파벳 대문자로만 이루어져 있으며, 최대 1000글자로 이루어져 있다.

출력

첫째 줄에 입력으로 주어진 두 문자열의 LCS의 길이를 출력한다.

예제 입력 1 복사

```
ACAYKP
CAPCAK
```

예제 출력 1

```
4
```

LCS(Longest Common Subsequence, 최장 공통 부분 수열)문제는 두 수열이 주어졌을 때, 모두의 부분 수열이 되는 수열 중 가장 긴 것을 찾는 문제이다.

예를 들어, ACAYKP와 CAPCAK의 LCS는 ACAK가 된다.

s1 = “A”
s2 = “CAP”
LCS = 1

s1 = “AC”
s2 = “CAPC”
LCS = 2

s1 = “ACAY”
s2 = “CAPCA”
LCS = 3

s1 = “ACAYK”
s2 = “CAPCAK”
LCS = 4

dp[i][j]	A	C	A	Y	K	P
C	0	1	1	1	1	1
A	**1**	1	2	2	2	2
P	1	1	2	2	2	3
C	1	**2**	2	2	2	3
A	1	2	**3**	3	3	3
K	1	2	3	3	**4**	4

LCS(Longest Common Subsequence, 최장 공통 부분 수열)문제는 두 수열이 주어졌을 때, 모두의 부분 수열이 되는 수열 중 가장 긴 것을 찾는 문제이다.

예를 들어, ACAYKP와 CAPCAK의 LCS는 ACAK가 된다.

s1 = "A"
s2 = "CAPCAK"
LCS = 1

s1 = "AC"
s2 = "CAPCAK"
LCS = 2

s1 = "ACAY"
s2 = "CAPCAK"
LCS = 3

s1 = "ACAYKP"
s2 = "CAPCAK"
LCS = 4

dp[i][j]	A	C	A	Y	K	P
C	0	1	1	1	1	1
A	1	1	2	2	2	2
P	1	1	2	2	2	3
C	1	2	2	2	2	3
A	1	2	3	3	3	3
K	1	2	3	3	4	4

'A' != 'P'
'K' != 'C'
'K' == 'K'

1. DP 테이블 정의

dp[i][j] = s1의 i까지, s2의 j까지 문자들의 LCS길이

2. 점화식 찾기

$$dp[i][j] = \begin{cases} s1[i] == s2[j]\text{일때,} \\ \quad dp[i-1][j-1]+1 \\ s1[i] \; != s2[j]\text{일 때,} \\ \quad \max(dp[i][j-1], dp[j-1][i]) \end{cases}$$

dp[i][j]	A	C	A	Y	K	P
C	0	1	1	1	1	1
A	1	1	2	2	2	2
P	1	1	2	2	2	3
C	1	2	2	2	2	3
A	1	2	3	3	3	3
K	1	2	3	3	4	4

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

int dp[1001][1001];
// dp[i][j]: s1의 i번째, s2의 j번째 문자까지로 이루어진 문자열들의 LCS길이
string s1, s2; //입력받는 두 문자열

int func(int i, int j){
    if(i==-1||j==-1) return 0; // 기저조건
    if(dp[i][j]!=-1) return dp[i][j]; // 메모이제이션
    if(s1[i]==s2[j]) return dp[i][j]= func(i - 1, j - 1) + 1;
    return dp[i][j]=max(func(i - 1, j), func(i, j - 1));
}
int main() {
    ios_base::sync_with_stdio(0); cin.tie(0);

    //dp배열 초기화
    for(int i=0; i<1001; i++){
        for(int j=0; j<1001; j++)
            dp[i][j]=-1;
    }
    //문자열 입력
    cin>>s1>>s2;

    //LCS 출력
    cout << func(s1.length() - 1, s2.length() - 1);
    return 0;
}
```

탑다운(top-down)

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

int dp[1001][1001];
string s1, s2; //입력받는 두 문자열

int main() {
    ios_base::sync_with_stdio(0); cin.tie(0);

    cin>>s1>>s2;
    for(int i=1; i<=s1.length(); i++){
        for(int j=1; j<=s2.length(); j++){
            if(s1[i-1]==s2[j-1])
                dp[i][j]=dp[i-1][j-1]+1;
            else
                dp[i][j]=max(dp[i][j-1], dp[i-1][j]);
        }
    }
    //LCS 출력
    cout << dp[s1.length()][s2.length()];
    return 0;
}
```

바텀업(bottom-up)

동전 2

성공

골드 V

시간 제한	메모리 제한	제출	정답	맞힌 사람
1 초 (추가 시간 없음)	128 MB	51355	15189	10634

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 입력 1 복사

```
3 15
1
5
12
```

예제 출력 1 복사

```
3
```

11047번 제출

동전 0 성공

실버 IV

문제

준규가 가지고 있는 동전은 총 N종류이고, 각각의 동전을 매우 많이 가지고 있다.

동전을 적절히 사용해서 그 가치의 합을 K로 만들려고 한다. 이때 필요한 동전 개수의 최솟값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N과 K가 주어진다. ($1 \le N \le 10$, $1 \le K \le 100,000,000$)

둘째 줄부터 N개의 줄에 동전의 가치 A_i가 오름차순으로 주어진다. ($1 \le A_i \le 1,000,000$, $A_1 = 1$, $i \ge 2$인 경우에 A_i는 A_{i-1}의 배수)

출력

첫째 줄에 K원을 만드는데 필요한 동전 개수의 최솟값을 출력한다.

예제 입력 1

```
10 4200
1
5
10
50
100
500
1000
5000
10000
50000
```

예제 출력 1

```
6
```

2294번 제출

동전 2 성공

골드 V

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. ($1 \le n \le 100$, $1 \le k \le 10,000$) 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 입력 1

```
3 15
1
5
12
```

예제 출력 1

```
3
```

2294번 재출

동전 2

성공

골드 V

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.

- 그리디 적용
->가치가 높은 동전부터, 최대한 많이 사용하기

결과

틀렸습니다

- 반례

예제 입력 1	예제 출력 1
3 15 1 5 12	3

오답 : 12*1 + 5*0 + 1*3 = 4개

정답 : 12*0 + 5*3 + 1*0 = 3개

11047번 제출

동전 0 성공

실버 IV

문제

준규가 가지고 있는 동전은 총 N종류이고, 각각의 동전을 매우 많이 가지고 있다.

동전을 적절히 사용해서 그 가치의 합을 K로 만들려고 한다. 이때 필요한 동전 개수의 최솟값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N과 K가 주어진다. (1 ≤ N ≤ 10, 1 ≤ K ≤ 100,000,000)

둘째 줄부터 N개의 줄에 동전의 가치 A_i가 오름차순으로 주어진다. (1 ≤ A_i ≤ 1,000,000, A_1 = 1, i ≥ 2인 경우에 A_i는 A_{i-1}의 배수)

출력

첫째 줄에 K원을 만드는데 필요한 동전 개수의 최솟값을 출력한다.

예제 입력 1	예제 출력 1
10 4200 1 5 10 50 100 500 1000 5000 10000 50000	6

2294번 제출

동전 2 성공

골드 V

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 입력 1	예제 출력 1
3 15 1 5 12	3

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 입력 1	예제 출력 1
3 15 1 5 12	3

1. DP 테이블 정의

dp[i] = i원을 만드는 데 필요한 동전의 최소 개수

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp	0	**1**	2	3	4	**1**	2	3	4	5	**2**	3	**1**	2	3	**3**

정답 : dp[15] = 3

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp	0	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp	0	1	2	3	4	1	2	3	4	5	2	3	4	5	6	3

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp	0	1	2	3	4	1	2	3	4	5	2	3	1	2	3	3

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp	0	1	2	3	4	1	2	3	4	5	2	3	1	2	3	3

예제 입력 1

```
3 15
1
5
12
```

예제 출력 1

```
3
```

i	0	1	2	3	4	5	6	7	8
dp	0	1	2	3	4	1	2	3	4

9	10	11	12	13	14	15
5	2	3	1	2	3	3

1. DP 테이블 정의

   dp[i] = i원을 만드는 데 필요한 동전의 최소 개수

2. 점화식 찾기

   dp[j] = min (dp[j] , dp[j - coin[i]] + 1)

3. 초기값 설정

   dp[0] = 0, dp[i] = KMAX+1

```cpp
#include <iostream>
using namespace std;
#define NMAX 101
#define KMAX 10000
int coin[NMAX]; //동전종류
int dp[KMAX+1]; //i원을 만드는 데 필요한 동전의 최소 개수

int main() {
    ios_base::sync_with_stdio(false); cin.tie(NULL);

    int n,k; //동전종류, 가치의 합
    cin>> n>>k;
    for(int i=0; i<n; i++){
        cin>>coin[i]; //각 동전의 가치
    }
    fill(dp+1, dp+k+1, KMAX+1); //index 1부터 k까지 KMAX+1로 초기화
    //dp[0]=0 이어야 하므로 index 0은 초기화안함.

    for(int i=0; i<n; i++){ //i번째동전 coin[i]를 써서
        for(int j=coin[i]; j<=k; j++){ //j원부터 환전시작
            dp[j]=min(dp[j],dp[j-coin[i]]+1);
            //j-coin[i]원의 동전 개수에서 1번째동전(+1개)를 사용한것이
            //원래 arr[j]보다 적다면 그것을 선택
        }
    }
    if(dp[k]==KMAX+1) cout<<"-1";
    else cout << dp[k];
    return 0;
}
```

감사합니다

- 필수 문제
  - 2747. 피보나치수
  - 1463. 1로만들기
  - 11053. 가장 긴 증가하는 부분수열
  - 9251. LCS
  - 2294. 동전2
- 연습 문제
  - 2748. 피보나치 수 2
  - 9461. 파도반 수열
  - 1912. 연속합
  - 11048. 이동하기
  - 14462. 소가 길을 건너간 이유 8
  - 14002. 가장 긴 증가하는 부분 수열 4
- 10월 5일(수요일) 저녁 6시 T702