#include <bits/stdc++.h>

using namespace std;

#define ll long long

#define ld long double

#define fi first

#define se second

#define int long long

int f[1001];

int pow(int n,int k)

{

int res=1;

for (int i=1; i<=k; i++)

{

res\*=n;

if (res>1e9) return 1e9;

// 1e9 ở đây tương ứng với

}

return res;

}

int32\_t main()

{

int n,k;

cin >> n >> k;

memset(f, 0, sizeof(f));

f[1]=1;

f[0]=1;

for (int i=2; i<=n; i++)

for (int j=n; j>=pow(i,k); j--) f[j]+=f[j-pow(i,k)];

cout << f[n];

}

Quy hoạch động (Dynamic Programming - DP) là một kỹ thuật trong lập trình giúp giải quyết các bài toán tối ưu hóa bằng cách chia nhỏ chúng thành các bài toán con nhỏ hơn và lưu trữ kết quả của các bài toán con này để tái sử dụng.

Đoạn code trên sử dụng phương pháp quy hoạch động để giải quyết bài toán phân tích một số thành tổng các lũy thừa của các số khác nhau.

Tư tưởng ở trong bài toán này là:

1. Giả sử ta đã tính được số cách để tách 1 số j nào đó là
2. Khi đó, với mỗi tổng j+ (giả sử tổng này đang có số cách phân tích là ) sẽ có số cách phân tích được cộng thêm x (hay y = y + x cho dễ hiểu). Điều này mọi người có thể tự nhận ra, không cần giải thích thêm.

Vậy nên, ta cần có 1 mảng để lưu lại số cách phân tích các số - gọi là f, với ý nghĩa: f[i] là số cách phân tích số i.

Sau đây là phần giải thích cho đoạn code ở phần đầu

1. Hàm pow(int n, int k) được sử dụng để tính lũy thừa bậc k của số n.
2. Trong hàm main(), chúng ta đọc n và k từ bàn phím. Mảng f được khởi tạo với f[1] và f[0] được gán là 1 (vì có một cách duy nhất để phân tích số 0 và 1, đó là và ).
3. Sau đó, chúng ta sẽ duyệt qua tất cả các số nguyên i từ 2 đến n-1. Với mỗi i như vậy, chúng ta cập nhật mảng f bằng cách cộng thêm f[j - ] vào f[j] với mọi j từ n đến . Điều này tương ứng với việc thêm số vào tổng j - đã có để tạo ra tổng j.
4. Cuối cùng, f[n] sẽ chứa số cách để phân tích n thành tổng các lũy thừa bậc k của các số nguyên dương khác nhau. Kết quả này được in ra màn hình.

Ví dụ với testcase 9 2:

1. Mảng f khi mới khởi tạo. Các số bắt đầu từ 0:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

1. Khi i = 2, ta sẽ cộng kết quả của các số j từ 9 cho tới với kết quả của số j - :

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

1. Khi i = 3, tương tự ta có:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |

1. Với i = 4 … 9, do điều kiện không được đảm bảo nên sẽ không thực hiện gì cả.

Từ mảng trên suy ra số 0 có 1 cách tách, số 1 có 1 cách tách …

Khi kiểm tra lại, ta thấy kết quả ở mảng trên là chính xác.

Khuyến khích các bạn tự code lại theo ý hiểu của mình. Pa sờ te code không giúp các bạn tư duy được gì cả.

Bài này có thể code theo đệ quy, nếu ai muốn được A+ có thể thử sức.