# Bài 1: Số cách chia N phần tử khác nhau vào K nhóm

##### **Đề bài**

Cho N phần tử khác nhau, hỏi có bao nhiêu cách chia N phần tử đó thành K nhóm, mà mỗi nhóm có ít nhất 1 phần tử (các hoán vị của nhóm được xem là 1 cách).

**Giới hạn:**

* 1 ≤ K ≤ N ≤ 25

##### **Định dạng test**

**Input:**

* Dòng đầu tiên chứa số T là số test.
* T dòng tiếp theo mỗi dòng chứa 2 số N và K.

**Output:**

* T dòng, mỗi dòng là số cách với test tương ứng.

Ví dụ:

input:

1

4 2

---

output:

7

Giải thích : 7 cách chia đó là (ABC)(D) , (ABD)(C) , (ADC)(B) , (DBC)(A) , (AB)(CD) , (AC)(BD) , (BC)(AD).

##### **Thuật toán**

Gọi f[i][j] là số cách chia i phần tử vào j nhóm, công thức truy hồi:

F[0,0]=0;

For(i=1..25)

For(j=1..i)

f[i][j] = f[i-1][j-1] + f[i-1][j] \* j

Giải thích:

* Nếu đã chia được i-1 phần tử trước đó thành j-1 nhóm, thì phần tử thứ i sẽ chỉ có thể được chia vào nhóm thứ j.
* Nếu đã chia được i-1 phần tử trước đó thành j nhóm, thì sẽ có j cách để chia phần tử thứ i vào j nhóm đó.

*Bài 2: Có N gói kẹo, gói thứ i có A*i *cái kẹo. Không được bóc bất kỳ một gói kẹo nào, cần chia N gói kẹo thành hai phần sao cho độ chênh lệch số kẹo giữa hai gói là ít nhất.*

*Dữ liệu vào trong file “****chiakeo.inp****” có dạng*:

*– Dòng đầu tiên là số N(N<=100);*

*– Dòng thứ hai là N số A*i*(i=1, 2,.., N; A*i*<=100).*

*Kết quả ra file “****chiakeo.out****” có dạng*:

*– Dòng đầu là độ chênh lệch nhỏ nhất giữa hai phần có thể được.*

*– 2 dòng tiếp theo là 2 đoạn đã được chia ra sao cho chênh lệch là ít nhất*

Chiakeo.Inp chiakeo.out

7 1

7 9 14 18 15 3 7 9 18 3 7

7 14 15

Công thức này là một phần của thuật toán quy hoạch động để giải bài toán phân chia các gói kẹo sao cho độ chênh lệch nhỏ nhất giữa hai phần. Để hiểu rõ hơn, ta sẽ phân tích từng bước của thuật toán.

### Ý tưởng chính của thuật toán

Bài toán có thể được coi là bài toán con của bài toán "Subset Sum Problem", nơi mà ta muốn kiểm tra xem liệu có thể chia tập hợp các phần tử thành hai phần có tổng gần nhau nhất.

Chúng ta sử dụng mảng f để biểu diễn trạng thái của bài toán. f[j] sẽ lưu trữ chỉ số của gói kẹo cuối cùng được sử dụng để đạt được tổng j. Nếu f[j] == -1, điều đó có nghĩa là không thể đạt được tổng j với các gói kẹo hiện tại.

### Bước 1: Khởi tạo

cpp

Copy code

f[0] = 0;

Điều này có nghĩa là chúng ta có thể đạt được tổng 0 mà không cần sử dụng bất kỳ gói kẹo nào.

### Bước 2: Cập nhật mảng f

cpp

Copy code

for (int i = 1; i <= n; i++) {

for (int j = t; j >= a[i]; j--) {

if (f[j - a[i]] != -1 && f[j] == -1) {

f[j] = i;

}

}

}

Ở đây, ta lặp qua từng gói kẹo a[i] và cố gắng cập nhật mảng f để bao gồm gói kẹo a[i]. Việc cập nhật từ phải sang trái (for (int j = t; j >= a[i]; j--)) đảm bảo rằng mỗi gói kẹo chỉ được sử dụng một lần.

#### Chi tiết từng bước:

* for (int i = 1; i <= n; i++): Lặp qua từng gói kẹo.
* for (int j = t; j >= a[i]; j--): Lặp qua từng tổng có thể từ t đến a[i].
* if (f[j - a[i]] != -1 && f[j] == -1): Kiểm tra nếu có thể đạt được tổng j - a[i] với các gói kẹo hiện tại (f[j - a[i]] != -1) và tổng j chưa được đạt được (f[j] == -1).
* f[j] = i;: Nếu điều kiện trên thỏa mãn, cập nhật f[j] bằng chỉ số của gói kẹo a[i].

### Tại sao công thức này hoạt động

Công thức này hoạt động dựa trên nguyên lý quy hoạch động, nơi mà chúng ta xây dựng nghiệm của bài toán lớn từ nghiệm của các bài toán con nhỏ hơn.

1. **Khởi tạo**: f[0] = 0 vì chúng ta luôn có thể đạt được tổng 0 mà không cần sử dụng bất kỳ gói kẹo nào.
2. **Cập nhật**: Khi duyệt qua từng gói kẹo, chúng ta kiểm tra tất cả các tổng có thể đạt được từ t đến a[i]. Nếu có thể đạt được tổng j - a[i], thì có thể đạt được tổng j bằng cách thêm gói kẹo a[i] vào. Chúng ta chỉ cập nhật f[j] nếu f[j] chưa được cập nhật trước đó để đảm bảo rằng mỗi gói kẹo chỉ được sử dụng một lần.
3. **Truy vết**: Sau khi cập nhật mảng f, chúng ta có thể truy vết lại để tìm các gói kẹo đã được sử dụng để đạt được tổng gần nhất với t.