

# Quy hoạch động bitmask

## Quy hoạch động Bitmask

### Người viết:

- Nguyễn Đức Kiên, Trường Đại học Công nghệ, ĐHQGHN

### Reviewer:

- Phạm Công Minh - Trường THPT chuyên Khoa học Tự nhiên, ĐHQGHN
- Cao Thanh Hậu - Trường Đại học Khoa học Tự nhiên, ĐHQG-HCM

## Giới thiệu

Khi giải các bài toán về Quy hoạch động (QHD), chúng ta đã làm quen với các trạng thái là vị trí hoặc giá trị. Ví dụ, trong bài toán [Knapsack](#) [🔗](#), chúng ta đã gọi  $dp[i][j]$  là giá trị lớn nhất thu được khi đến vị trí thứ  $i$ , thu được khối lượng là  $j$ .

Với việc sử dụng biểu diễn nhị phân (bitmask), chúng ta hoàn toàn có thể sử dụng phương pháp QHD với trạng thái là các tập hợp.

## Kiến thức cần biết

Để đọc hiểu bài viết sau một cách hiệu quả, bạn đọc nên có sẵn các kiến thức về:

- [Quy hoạch động cơ bản](#) [🔗](#)
- [Các phép toán với bit](#) [🔗](#)

## Biểu diễn nhị phân và các phép toán trên bit

Đầu tiên, ta nhắc lại đôi chút về các [phép toán bit](#) [🔗](#).

Như chúng ta đã biết, mọi số nguyên trong hệ thập phân đều có thể biểu diễn được dưới dạng [nhị phân](#) [🔗](#). Với các số nhị phân này, ta có thêm một số phép toán bên cạnh các phép toán thông thường như **AND** ( `&` trong C++), **OR** ( `|` trong C++), **XOR** ( `^` trong C++), dịch trái ( `<<` trong C++), dịch phải ( `>>` trong C++), ... Khi sử dụng C++, ngôn ngữ này hỗ trợ chúng ta thêm một số hàm liên quan đến dãy bit như `std::builtin_popcount()` (đếm số bit 1), `std::builtin_ctz()` (đếm số bit 0 ở tận cùng bên phải), ...

Sử dụng các phép toán trên, chúng ta có thể dễ dàng thực hiện các thao tác:

- Xác định giá trị của bit thứ  $i$ : `(mask >> i) & 1`

- Lật ngược (flip) giá trị của bit thứ  $i$ :  $\text{mask} \wedge= (1 \ll i)$
- Đếm số bit 1 trong một số được biểu diễn nhị phân

Có thể thấy các thao tác này có những điểm tương đồng với việc thao tác trên các tập hợp: kiểm tra một phần tử có nằm trong tập hợp hay không, thêm/bớt một phần tử trong tập hợp, đếm số phần tử của tập hợp. Điều đó cho ta ý tưởng về việc dùng biểu diễn nhị phân để biểu diễn các tập hợp con của một tập hợp.

## Biểu diễn các tập hợp bằng bitmask

Ta có thể biểu diễn một tập con của một tập hợp bằng một xâu nhị phân, trong đó bit thứ  $i$  bằng 0/1 khi phần tử thứ  $i$  không/có nằm trong tập hợp. Xâu nhị phân này được gọi là **bitmask**.

Giả sử có một tập hợp  $S = \{a_0, a_1, a_2, \dots, a_{n-1}\}$ . Xét một tập con  $T = \{a_{i_1}, a_{i_2}, \dots, a_{i_k}\}$  của  $S$ . Khi đó ta có thể biểu diễn  $T$  bằng một xâu nhị phân độ dài  $n$ , với bit thứ  $i$  (ở đây là bit có giá trị cao thứ  $i$  hay bit thứ  $i$  từ phải sang) bằng 1 nếu  $a_i$  nằm trong tập  $T$ :

$$\text{mask}_T = 000 \dots 010 \dots 010 \dots 010 \dots 000$$

với các vị trí bằng 1 là  $i_1, i_2, \dots, i_k$ .

Do  $\text{mask}_T$  là một xâu nhị phân, ta có thể biểu diễn nó bằng một số ở dạng thập phân. Ví dụ:

$12 = (1100)_2$  biểu diễn tập con  $T = \{a_2, a_3\}$  của  $S = \{a_0, a_1, a_2, a_3\}$

$17 = (010001)_2$  biểu diễn tập con  $T = \{a_0, a_4\}$  của  $S = \{a_0, a_1, a_2, a_3, a_4, a_5\}$

$0 = (000)_2$  biểu diễn tập con  $T = \emptyset$  của  $S = \{a_0, a_1, a_2\}$

Khi duyệt tất cả các số từ 0 đến  $2^n - 1$ , ta sẽ duyệt qua mọi xâu nhị phân độ dài  $n$ , và cũng là duyệt qua tất cả các tập con của  $S = \{a_0, a_1, \dots, a_n\}$ . Công việc duyệt này có độ phức tạp thời gian là  $\mathcal{O}(2^n)$ .

Bạn đọc có thể làm thử [bài này](#) để hiểu rõ hơn về biểu diễn tập hợp bằng bitmask.

## Phương pháp QHĐ Bitmask

### Bài toán

**Đề bài:** [Codeforces - 100589G](#)

Cho hai số nguyên dương  $N$  và số tự nhiên  $K$  ( $N \leq 15, K \leq N$ ). Một hoán vị của dãy  $[a_1, \dots, a_n]$  là một cách sắp xếp lại dãy  $a$  sao cho mọi phần tử của dãy đều xuất hiện chính xác một lần. Đếm số hoán vị độ dài  $N$  của dãy  $[1, 2, \dots, N]$  mà trong hoán vị đó, hai phần tử liên tiếp chênh lệch nhau không quá  $K$ .

Ví dụ, với  $N = 4, K = 2$ ,  $[1, 3, 2, 4]$  là một hoán vị thoả mãn, còn  $[4, 1, 2, 3]$  là một hoán vị không thoả mãn do  $4 - 1 = 3 > 2$ .

### Phân tích

Cách làm tự nhiên nhất đối với bài toán trên là sinh ra mọi hoán vị có thể của dãy  $[1, 2, \dots, N]$  (tạm gọi là dãy  $a$ ) bằng cách sử dụng thuật toán quay lui. Độ phức tạp thời gian là  $\mathcal{O}(N!)$ ; với  $N \leq 15$  thì ta không thể sử dụng cách này.

Ta thử áp dụng tư tưởng quy hoạch động. Nếu xét theo vị trí, ta thấy phần tử ở vị trí  $i$  phải sai khác phần tử ở  $i - 1$  của  $a$  một khoảng không quá  $K$ . Cách này đưa ta tới cách gọi  $dp[i][p]$  là số lượng dãy hoán vị sao cho phần tử thứ  $i$  là  $p$  và dễ dàng biểu diễn nó theo  $dp[i - 1][p]$ . Tuy nhiên, cách gọi này không đảm bảo được tính chất quan trọng của hoán vị: tất cả các phần tử trong dãy ban đầu phải xuất hiện chính xác một lần.

Để có được tính chất này, thay vì chỉ dùng vị trí, ta dùng hẳn một tập hợp để lưu lại tất cả các vị trí đã được thêm vào dãy trước đó. Gọi  $dp[X][p]$  là số lượng dãy là hoán vị của  $X \subset A$  và có phần tử cuối cùng được thêm vào  $X$  là  $p$ . Khi thêm một phần tử  $q$  vào  $X$ , ta cần chắc chắn rằng  $q \notin X$ , và  $|q - p| \leq K$ . Như vậy:

$$\forall q \notin X, dp[X \cup \{q\}][q] = \sum_{\substack{p \in X, \\ |q-p| \leq K}} dp[X][p].$$

Với trường hợp cơ sở, ta có thể chọn  $dp[\emptyset][\infty] = 1$ . Dĩ nhiên, ta không thể biểu diễn  $\infty$  trong mảng khi cài đặt được; trong trường hợp này ta có thể thay thế nó bằng 0 và chấp nhận mọi trường hợp thêm  $q$  vào  $X$  nếu xuất hiện  $p = 0$ . Kết quả của bài toán là tổng số dãy được tạo ra từ dãy  $a$  ban đầu với đầy đủ phần tử ở mọi trường hợp phần tử cuối cùng, tức  $\sum_{p=1}^N dp[a][p]$ .

Còn để biểu diễn tập hợp  $X$  khi cài đặt, ta sử dụng bitmask như đã nói ở trên.

- ▶ Mọi tập con  $X$  của  $a$  đều có thể được biểu diễn bằng một dãy nhị phân độ dài  $N$ , tương đương một số nguyên dương. Chẳng hạn, với  $N = 4$ ,  $dp[\{0, 2\}][1]$  được biểu diễn thành  $dp[5][1]$ , do  $(0101)_2 = 5$ . Gọi biểu diễn nhị phân này là **mask**.
- ▶ Biểu diễn nhị phân của  $X$  khi  $X = a$  là **mask** =  $(1 \ll N) - 1$
- ▶ Biểu diễn của  $q \notin X$  là  $((\text{mask} \gg q) \& 1) \neq 1$  (tức là bit thứ  $q$  của **mask** khác 1)
- ▶ Biểu diễn của  $X \cup \{q\}$  là **mask** |  $(1 \ll q)$ .
- ▶ Để duyệt qua mọi trạng thái của tập  $X$ , ta duyệt mọi số nguyên tương ứng từ 0 đến  $2^N - 1$  cho giá trị của **mask**.

Trong bài toán này, hoán vị phải được tạo thành bởi các số từ 1 đến  $N$ . Nếu sử dụng cách biểu diễn trên, bit ở vị trí thứ 0 không biểu diễn cái gì cả, hơn nữa phải dùng  $N + 1$  bit mới biểu diễn hết được  $N$  số này. Vì vậy, thay vì dùng bit thứ  $q$  để biểu diễn việc  $q$  có thuộc  $X$  hay không, ta dùng bit thứ  $q - 1$ . Bằng cách này, ta chỉ cần đúng  $N$  bit để biểu diễn tập hợp mà không lãng phí bit nào.

## Cài đặt

```

1  const int MAXN = (1 << 15) + 1;
2
3  int n, k;
4  long long dp[MAXN][16];
5
6  //Lấy bit thứ k của số x
7  int getBit(int x, int k)
8  {
9      return (x >> k) & 1;
10 }
11
12 int solve(int n, int k)
13 {
14     for (int mask = 0; mask < (1 << n); mask++)
15         for (int k = 0; k <= n; k++)
16             dp[mask][k] = 0;
17
18 
```

```

19 //base case
20 dp[0][0] = 1;
21
22 for (int mask = 0; mask < (1 << n); mask++)
23     for (int q = 1; q <= n; q++)
24     {
25         //check q nằm trong tập hợp (biểu diễn bằng mask)
26         if (getBit(mask, q - 1)) continue;
27         for (int p = 0; p <= n; p++)
28         {
29             //check chênh lệch của q và p
30             if (p != 0 && abs(q - p) > k) continue;
31
32             //thêm q vào tập hợp
33             int newMask = mask | (1 << (q - 1));
34
35             dp[newMask][q] += dp[mask][p];
36         }
37     }
38
39 long long res = 0;
40 int fullMask = (1 << n) - 1;
41 for (int k = 1; k <= n; k++)
42     res += dp[fullMask][k];
43 return res;
}

```






Độ phức tạp về thời gian của cách cài đặt trên là  $\mathcal{O}(N^2 \times 2^N)$ , còn độ phức tạp về không gian là  $\mathcal{O}(N \times 2^N)$ .



## Chú ý thêm

Ta thấy rằng nếu  $m$  lớn, thuật toán cũng sẽ mất rất nhiều thời gian để chạy. Vì vậy, để thực hiện được phương pháp này thì  $m$  cần phải đủ nhỏ, thường là  $m \leq 20$  nếu thuật toán chỉ yêu cầu ta phải duyệt qua các tập hợp.

Việc sử dụng biểu diễn bitmask của tập hợp và QHĐ bitmask giúp chúng ta kiểm tra được mọi tập hợp con của một tập hợp lớn hơn. Do vậy, trong nhiều trường hợp, đây là phương pháp đơn giản và đáng để thử khi muốn kiểm điểm một cách hiệu quả từ các subtask bé, cũng như có một code chắc chắn để làm tiếp với các subtask lớn hơn

## Bài tập tương tự

- ▶ [CSES - Counting Tilings](#) 
- ▶ [VNOJ - Atcoder Educational DP Contest U - Grouping](#) 
- ▶ [Codeforces 453B - Little Pony and Harmony Chest](#) 
- ▶ [Codeforces 165E - Compatible Numbers](#) 
- ▶ [VOI 06 Bài 1 - QBSELECT](#) 

- [VOI 23 Bài 4 - WHOME](#)  (Subtask 1)
- [Bài tập trên USACO](#) 

Được cung cấp bởi [Wiki.js](#)