Chuyên đề A4: Quy hoạch động đếm

Ban Chuyên môn Tin học – Tổ chức The Gifted Battlefield nhiệm kỳ 2025-2026

Xuất bản vào Ngày 5 tháng 3 năm 2026

Tóm tắt nội dung

Quy hoạch động là một kĩ thuật lập trình mạnh, quan trọng trong lập trình thi đấu và được dùng để giải nhiều dạng bài. Trong bài này, ta sẽ tìm hiểu quy hoạch động đếm cấu hình.

Mục lục

1 Tổ hợp **2**
- 1.1 Định nghĩa . . . 2
- 1.2 Các khái niệm cơ bản trong Tổ hợp học . . . 2
- 1.3 Bionomial Coefficient . . . 3

2 Quy nạp trong tổ hợp **5**
- 2.1 Định nghĩa . . . 5
- 2.2 Nguyên lý của quy nạp . . . 5
- 2.3 Vai trò của quy nạp trong tổ hợp . . . 6

3 Quy hoạch động trong tổ hợp **6**
- 3.1 Định nghĩa . . . 6
- 3.2 Số Stirling loại 2 . . . 6
- 3.3 Bài toán Ways to Sum . . . 7

4 Ứng dụng của Binomial Coefficient trong quy hoạch động đếm **8**
- 4.1 Code tham khảo . . . 8
- 4.2 Ứng dụng trong quy hoạch động đếm . . . 10

5 Ví dụ **12**

6 Tài liệu tham khảo **16**
- 6.1 Bài luyện tập . . . 16

1 Tổ hợp

1.1 Định nghĩa

Tổ hợp học (tên tiếng Anh: combinatorics) là một lĩnh vực của toán học chủ yếu liên quan đến các bài toán đếm, sắp xếp. Tổ hợp không chỉ xuất hiện trong toán học, mà còn được ứng dụng sâu rộng trong tin học, đặc biệt là lập trình thi đấu.

Tổ hợp học là một lĩnh vực rất rộng, và được chia thành nhiều nhóm nhỏ khác nhau. Dưới đây là một số nhánh quan trọng thường gặp:

- Tổ hợp liệt kê (Enumerative Combinatorics)
- Tổ hợp trên đồ thị (Graph Combinatorics)
- Tổ hợp cực trị (Extremal Combinatorics)
- $\cdots$

1.2 Các khái niệm cơ bản trong Tổ hợp học

2.1. Hoán vị (Permutation)

Định nghĩa. *Hoán vị* của n phần tử là một cách sắp xếp toàn bộ n phần tử đó theo một thứ tự nhất định.

Công thức.

$$P(n) = n! = n \times (n-1) \times (n-2) \times \cdots \times 1$$

Ví dụ. Có 3 học sinh A, B, C. Hỏi có bao nhiêu cách sắp xếp 3 bạn này thành một hàng?

$$P(3) = 3! = 3 \times 2 \times 1 = 6$$

Các hoán vị gồm:

$$\text{ABC, ACB, BAC, BCA, CAB, CBA.}$$

2.2. Chỉnh hợp (Arrangement)

Định nghĩa. *Chỉnh hợp* của n phần tử lấy k phần tử là số cách chọn và sắp xếp k phần tử trong n phần tử, có phân biệt thứ tự.

Công thức.

$$A(n,k) = \frac{n!}{(n-k)!}$$

Ví dụ. Từ 5 học sinh, chọn ra 3 em để xếp vào 3 chỗ ngồi. Hỏi có bao nhiêu cách xếp?

$$A(5,3) = \frac{5!}{(5-3)!} = 5 \times 4 \times 3 = 60$$

2.3. Quy tắc cộng (Addition Rule)

Định nghĩa. Nếu một công việc có thể được thực hiện theo một trong hai cách độc lập, và hai cách đó không thể xảy ra đồng thời, thì tổng số cách thực hiện công việc là tổng số cách của từng cách riêng.

Ví dụ. Một trường có 8 học sinh nam và 12 học sinh nữ. Hỏi có bao nhiêu cách chọn một người đại diện cho lớp?

$$8 + 12 = 20$$

Vậy có 20 cách chọn.

2.4. Quy tắc nhân (Multiplication Rule)

Định nghĩa. Nếu một công việc gồm nhiều giai đoạn liên tiếp, trong đó:

- Giai đoạn 1 có a_1 cách thực hiện,
- Giai đoạn 2 có a_2 cách thực hiện,
- ...
- Giai đoạn k có a_k cách thực hiện,

và mọi lựa chọn ở giai đoạn này không ảnh hưởng đến lựa chọn ở giai đoạn khác, thì tổng số cách thực hiện toàn bộ công việc là:

$$a_1 \times a_2 \times \cdots \times a_k$$

Ví dụ. Một mật khẩu gồm 1 chữ cái (A–Z), 1 chữ số (0–9) và 1 chữ cái đặc biệt(gồm 9 loại). Hỏi có bao nhiêu mật khẩu khác nhau?

$$26 \times 10 \times 9 = 2{,}340$$

2.5. Nguyên lý bao hàm – loại trừ (Inclusion–Exclusion Principle)

Định nghĩa. Khi đếm tổng số phần tử của nhiều tập hợp có giao nhau, ta phải trừ đi phần bị đếm trùng. Với hai tập A và B:

$$|A \cup B| = |A| + |B| - |A \cap B|$$

Ví dụ. Trong một lớp có:

- 20 học sinh học Toán,
- 15 học sinh học Lý,
- 8 học sinh học cả hai môn.

Số học sinh học ít nhất một trong hai môn là:

$$|A \cup B| = 20 + 15 - 8 = 27$$

1.3 Bionomial Coefficient

Định nghĩa 1.1

Hệ số nhị thức $\binom{n}{k}$ (đọc là "n choose k") biểu thị **số cách chọn k phần tử từ n phần tử khác nhau**, không quan tâm đến thứ tự.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

Ví dụ. Xét $\binom{5}{3} = 10$. Điều này có nghĩa là ta có đúng 10 cách chọn 3 phần tử trong 5 phần tử **khi không quan tâm đến thứ tự**.

Giả sử ta có tập hợp: {1, 2, 3, 4, 5}.

Các cấu hình chọn 3 phần tử trong 5 phần tử (không phân biệt thứ tự) là:

1. (1, 2, 3)
2. (1, 2, 4)
3. (1, 2, 5)
4. (1, 3, 4)
5. (1, 3, 5)
6. (1, 4, 5)
7. (2, 3, 4)
8. (2, 3, 5)
9. (2, 4, 5)
10. (3, 4, 5)

Những cấu hình như $(3, 2, 1)$ bị loại bỏ vì trùng với $(1, 2, 3)$ nếu không xét thứ tự.

$$\Rightarrow \binom{5}{3} = 10.$$

Giới thiệu tam giác Pascal

Tam giác Pascal là một bảng số được xây dựng theo quy luật:

- Hai cạnh của tam giác luôn bằng 1.
- Mỗi phần tử bên trong bằng tổng hai phần tử phía trên nó.

$$P(n,k) = P(n-1,k-1) + P(n-1,k)$$

với:

$$P(n,0) = P(n,n) = 1$$

Một phần của tam giác Pascal:

$$\begin{array}{ccccc} 1 & & & & \\ 1 & 1 & & & \\ 1 & 2 & 1 & & \\ 1 & 3 & 3 & 1 & \\ 1 & 4 & 6 & 4 & 1 \end{array}$$

Liên hệ với hệ số nhị thức

Mỗi phần tử trong tam giác Pascal chính là một hệ số nhị thức:

$$P(n,k) = \binom{n}{k}$$

Hệ số nhị thức thỏa mãn công thức:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Ví dụ:

$$\binom{4}{2} = 6$$

Tương đương với việc có 6 cách chọn 2 phần tử từ 4 phần tử mà không xét thứ tự.
Ngoài ra:

- $\binom{n}{k}$ là hệ số của x^k trong khai triển $(1+x)^n$.
- Ứng dụng: đếm số cách chọn, tính xác suất, chia nhóm, đếm cấu hình, ...

Một số tính chất cơ bản

Tính chất	Công thức	Ý nghĩa
Cơ sở	$\binom{n}{0} = \binom{n}{n} = 1$	Chỉ có đúng 1 cách chọn 0 phần tử hoặc chọn tất cả n phần tử từ một tập n phần tử.
Đệ quy Pascal	$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$	Công thức được xây dựng dựa trên việc chọn hoặc không chọn phần tử thứ n.
Đối xứng	$\binom{n}{k} = \binom{n}{n-k},\ k \leq n$	Chọn k phần tử từ n phần tử tương đương với bỏ đi $n-k$ phần tử.
Hockey Stick	$\binom{r}{r} + \binom{r+1}{r} + \cdots + \binom{n}{r} = \binom{n+1}{r+1}$	Một đẳng thức đẹp trong tam giác Pascal, thường được vẽ giống hình cây gậy hockey. Xem thêm: Link.

2 Quy nạp trong tổ hợp

2.1 Định nghĩa

Quy nạp phương pháp cho phép ta chứng minh một mệnh đề đúng với mọi số tự nhiên n bằng cách chứng minh nó đúng với mọi trường hợp cơ bản và suy ra nó đúng cho các trường hợp tiếp theo. Ta có thể tưởng tượng chúng giống dãy domino, khi ta chứng minh được cho trường hợp cơ bản, hay ở đây là đẩy ngả viên domino đầu tiên. Thì theo đó những viên khác sẽ ngả theo, giống như việc ta giải quyết được cho các trường hợp với số lớn hơn.

2.2 Nguyên lý của quy nạp

Giả sử ta có mệnh đề $P(n)$ phụ thuộc vào số tự nhiên n. Một mệnh đề $P(n)$ được gọi là đúng với mọi $n \geq n_0$ nếu thỏa 2 điều kiện:

- Chứng minh rằng $P(n_0)$ đúng. Đây còn được gọi là bước cơ sở.
- Giả sử $P(k)$ đúng với một giá trị tùy ý $k \geq n_0$, sau đó chứng minh rằng $P(k+1)$ cũng đúng. Bước này được gọi là bước quy nạp.

Khi đó, theo nguyên lý quy nạp trong toán học, mệnh đề $P(n)$ đúng với mọi $n \geq n_0$.

2.3 Vai trò của quy nạp trong tổ hợp

Trong tổ hợp, ta thường gặp các công thức biểu diễn mối quan hệ giữa các số tổ hợp, các dãy đếm, hoặc các cấu hình có quy luật lặp lại.
Dưới đây là vai trò chính:

- **Xác nhận tính đúng đắn của các đẳng thức tổ hợp phức tạp.**

  Ví dụ, ta sẽ chứng minh rằng

$$S = \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{n} = \sum_{k=0}^{n} \binom{n}{k} = 2^n$$

  Chứng minh bằng quy nạp:

  1. **Bước cơ sở:** Với $n = 0$, ta có $\binom{0}{0} = 1$ và $2^0 = 1$. Mệnh đề đúng.
  2. **Giả thiết quy nạp:** Giả sử công thức đúng với n, tức là $\sum_{k=0}^{n} \binom{n}{k} = 2^n$.
  3. **Bước quy nạp:** Ta cần chứng minh công thức cũng đúng với $n+1$. Sử dụng tính chất của tam giác Pascal $\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$, ta có:

$$\sum_{k=0}^{n+1} \binom{n+1}{k} = \binom{n+1}{0} + \binom{n+1}{1} + \cdots + \binom{n+1}{n+1}$$

     Khai triển dựa trên công thức Pascal và gom nhóm lại, ta sẽ thấy tổng này gấp đôi tổng của hàng trước đó:

$$= 2 \times \sum_{k=0}^{n} \binom{n}{k} = 2 \times 2^n = 2^{n+1}.$$

  Kết luận: Qua các bước trên, ta khẳng định công thức $\sum_{k=0}^{n} \binom{n}{k} = 2^n$ đúng với mọi $n \geq 0$.

3 Quy hoạch động trong tổ hợp

3.1 Định nghĩa

Quy hoạch động (DP) giúp biến các bài toán đếm tổ hợp có số cấu hình lớn thành bài toán có số cấu hình có thể xử lý bằng cách lưu kết quả các bài toán con và tái sử dụng. Ở đây ta sẽ liệt kê một vài ví dụ về ứng dụng của DP trong các bài tổ hợp.

3.2 Số Stirling loại 2

Đề bài

Cho N phần tử **phân biệt**. Đếm số cách chia chúng thành đúng K tập con **khác rỗng**. Ký hiệu: $S(N, K)$.

Ý tưởng

Xét phần tử thứ i, ta có hai lựa chọn:

- Tạo một tập mới chứa riêng phần tử i.
- Đưa phần tử i vào một trong j tập đã có.

$$dp[i][j] = \text{số cách chia } i \text{ phần tử thành } j \text{ tập}$$

$$dp[i][j] = dp[i-1][j-1] + j \cdot dp[i-1][j]$$

Base case:

$$dp[0][0] = 1$$

Độ phức tạp: $O(NK)$

Code

```
dp[0][0] = 1;
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= k; j++)
        dp[i][j] = (dp[i - 1][j - 1] + j * dp[i - 1][j]) % MOD;
```

3.3 Bài toán Ways to Sum

Ví dụ 3.1

Cho N loại đồng xu với giá trị:

$$a_1, a_2, \ldots, a_N$$

Mỗi loại coin có thể sử dụng **không giới hạn lần**. Hãy đếm số cách chọn các coin sao cho tổng bằng T

Hai cách được xem là **giống nhau** nếu chúng dùng cùng số lượng mỗi loại coin.

$$dp[x] = \text{số cách tạo tổng } x$$

$$dp[x] \mathrel{+}= dp[x - coin]$$

Độ phức tạp

$$O(N \times T)$$

```
dp[0] = 1;
for (int coin : coins)
    for (int x = coin; x <= T; x++)
        dp[x] = (dp[x] + dp[x - coin]) % MOD;
```

Ví dụ 3.2

Cho N loại đồng xu với giá trị:

$$a_1, a_2, \ldots, a_N$$

Mỗi loại coin có thể sử dụng **không giới hạn lần**. Hãy đếm số **dãy coin** có tổng bằng T

Hai dãy coin khác thứ tự được xem là **khác nhau**.

$$dp[x] = \text{số dãy coin tạo tổng } x$$

$$dp[x] \mathrel{+}= dp[x - coin]$$

Độ phức tạp

$$O(N \times T)$$

```
dp[0] = 1;
for (int x = 1; x <= T; x++)
    for (int coin : coins)
        if (x >= coin)
            dp[x] = (dp[x] + dp[x - coin]) % MOD;
```

Như vậy, thông qua phần này ta có thể thấy, DP có ứng dụng rất lớn và quan trọng, đặc biệt là trong các bài về đếm tổ hợp.

4 Ứng dụng của Binomial Coefficient trong quy hoạch động đếm

Như đã tìm hiểu ở phần Tổ hợp, **Binomial Coefficient** (Hệ số nhị thức, kí hiệu là $\binom{n}{k}$) biểu thị số cách chọn k phần tử từ n phần tử. Trong các bài toán quy hoạch động, khái niệm này không chỉ dùng để đếm cấu hình mà còn là công cụ đắc lực để tối ưu thời gian chạy.
Cụ thể, Binomial Coefficient được dùng để tăng tốc cho những phép tính liên quan tới đếm. Trong trường hợp thông thường, việc áp dụng công thức này kết hợp với luỹ thừa ma trận hoặc nghịch đảo modulo có thể cải thiện độ phức tạp từ $O(N)$ xuống $O(\log N)$. Đặc biệt, nếu số N và K nhỏ, ta có thể tính trước tất cả các Binomial Coefficient bằng tam giác Pascal, sau đó truy xuất kết quả chỉ trong $O(1)$.

4.1 Code tham khảo

Tính $\binom{n}{k}$ bằng giai thừa trong $O(n)$

Ta dùng công thức:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \pmod{\text{M}},$$

với $\text{M} = 10^9 + 7$.
Ý tưởng:

- Tiền xử lý mảng `fac[i]` $= i!$ (giai thừa).
- Tính nghịch đảo của giai thừa bằng luỹ thừa nhanh: $x^{-1} = x^{\text{mod}-2} \pmod{\text{M}}$.

Cài đặt

```
#include <bits/stdc++.h>

using namespace std;

const int mod = 1e9 + 7;
const int maxn = 1000000 + 5; // chọn số đủ lớn

int binpow(int a, int b) {
    int res = 1;
    while (b) {
```

```
        if (b & 1) res = 1LL * res * a % mod;
        a = 1LL * a * a % mod;
        b >>= 1;
    }
    return res;
}

int fac[maxn], inv[maxn];

int C(int n, int k) {
    if (k < 0 || k > n) return 0;
    return 1LL * fac[n] * inv[n - k] % mod * inv[k] % mod;
}

void build() {
    fac[0] = 1;
    for (int i = 1; i < maxn; i++)
        fac[i] = 1LL * fac[i - 1] * i % mod;

    inv[maxn - 1] = binpow(fac[maxn - 1], mod - 2);
    for (int i = maxn - 2; i >= 0; i--)
        inv[i] = 1LL * inv[i + 1] * (i + 1) % mod;
}

signed main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    build();
    return 0;
}
```

Hàm **binpow** là một hàm để tính số a^b nhanh trong $O(logN)$. Với việc các bài toán thường có giá trị MOD là số nguyên tố lớn nên ta có thể dùng Fermat nhỏ để tính a^{-1}. Bạn có thể tìm hiểu thêm về hàm này tại đây: Link

Tính mọi $\binom{n}{k}$ đến n bằng tam giác Pascal trong $O(n^2)$

Dựa trên công thức đệ quy:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}.$$

Ta có thể xây dựng bảng $f[i][j] = \binom{i}{j}$

Cài đặt

```
const int maxn = 2005;
const int mod = 1e9 + 7;

int f[maxn][maxn];

void build_pascal(int n) {
    f[0][0] = 1;
    for (int i = 1; i <= n; i++) {
        f[i][0] = 1;
        for (int j = 1; j <= i; j++) {
```

```
            f[i][j] = f[i - 1][j - 1] + f[i - 1][j];
            if (f[i][j] >= mod) f[i][j] -= mod;
        }
    }
}
```

4.2 Ứng dụng trong quy hoạch động đếm

Xét bài: Distributing Apples (CSES 1716).

Đề bài (diễn giải):
Có n học sinh và m quả táo. Hỏi có bao nhiêu cách chia m quả táo cho n học sinh, sao cho:

- Sau khi chia không có dư bất kì quả táo nào (phát hết m quả táo cho n bạn).
- Một học sinh có thể nhận 0 quả.
- Kết quả lấy modulo $10^9 + 7$.

4.1. Cách 1: Quy hoạch động

Gọi dp$[i][j]$ là **số cách** chia j quả táo cho i học sinh đầu tiên.
Giả sử học sinh thứ i nhận k quả táo ($0 \le k \le j$), thì phần còn lại là chia $j-k$ quả táo cho $i-1$ học sinh trước đó.
Ta có:

$$\text{dp}[i][j] = \sum_{k=0}^{j} \text{dp}[i-1][j-k].$$

Khởi tạo:

$$\text{dp}[0][0] = 1, \quad \text{dp}[0][j] = 0 \text{ nếu } j > 0.$$

Cài đặt

```
const int mod = 1e9 + 7;
const int maxn = 505; // tuy bai toan ma chon cho phu hop

int dp[maxn][maxn];

void solve_dp(int n, int m) {
    dp[0][0] = 1;
    for (int i = 1; i <= n; i++) {
        for (int j = 0; j <= m; j++) {
            dp[i][j] = 0;
            for (int k = 0; k <= j; k++) {
                dp[i][j] += dp[i - 1][j - k];
                if (dp[i][j] >= mod) dp[i][j] -= mod;
            }
        }
    }
    cout << dp[n][m] << "\n";
}
```

4.2. Cách 2: Dùng Binomial Coefficient

Ta có bài toán **kinh điển** "Bài toán chia kẹo euler":

- Mỗi quả táo là một dấu $*$.
- Ta đặt $n-1$ vạch $|$ để chia chuỗi $*$ thành n đoạn (mỗi đoạn tương ứng với số táo của một học sinh).

Tổng số ký tự trong chuỗi là:

$$m \text{ (dấu sao) } + (n-1) \text{ (vạch) } = m+n-1.$$

Ta cần chọn vị trí cho $n-1$ vạch trong $m+n-1$ vị trí:

$$f(n,m) = \binom{m+n-1}{n-1}.$$

Ví dụ nhỏ: Với $n=3$ học sinh, $m=2$ quả táo, ta có 6 cấu hình:

1. (0, 0, 2)
2. (0, 1, 1)
3. (0, 2, 0)
4. (1, 0, 1)
5. (1, 1, 0)
6. (2, 0, 0)

Biểu diễn bằng $*$ và $|$: [0, 2, 0] tương ứng: $|\ *\ *\ |$. Một ví dụ khác: $n=4$, $m=6$.
Cấu hình [1, 0, 3, 2] tương ứng $*\ |\ |\ ***\ |\ **$.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
#define fi first
#define se second
#define pii pair<int,int>
#define arr3 array<int,3>

using namespace std;

const int mod = 1e9 + 7;
const int maxn = 10000000 + 7; // chọn số lớn hơn m*n

int binpow(int a, int b) {
    int res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

int fac[maxn], inv[maxn];

int C(int n, int k) {
    if (k < 0 || k > n) return 0;
    return fac[n] * inv[n - k] % mod * inv[k] % mod;
}
```

```cpp

void build() {
    fac[0] = 1;
    for (int i = 1; i < maxn; i++)
        fac[i] = fac[i - 1] * i % mod;

    inv[maxn - 1] = binpow(fac[maxn - 1], mod - 2);
    for (int i = maxn - 2; i >= 0; i--)
        inv[i] = inv[i + 1] * (i + 1) % mod;
}

int n, m;

void solve() {
    cin >> n >> m;
    cout << C(m + n - 1, n - 1) << '\n';
}

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    build();
    solve();
    return 0;
}
```

5 Ví dụ

Ví dụ 5.1

Cho một mảng C độ dài 26 ($\sum C_i \leq 5 \times 10^5$). Bạn phải tạo một xâu S, sao cho với mỗi $i = 1, 2, ..., 26$, kí tự thứ i của bảng chữ cái xuất hiện đúng C_i lần trong S. Xâu S (đánh chỉ số 0) là *đẹp* nếu: với mọi (i, j) mà $S_i = S_j$ thì $|i - j| \mod 2 = 0$. Đếm số xâu S phân biệt và đẹp tạo được, chia lấy phần dư cho 998244353.
Link đề bài

Ý tưởng

Đặt $n = \sum C_i$. Ta thấy rằng xâu S chia làm 2 phần: các vị trí chỉ số chẵn và chỉ số lẻ (gọi nhanh là *phần chẵn* và *phần lẻ*). Một chữ cái phải xuất hiện hoàn toàn trong 1 trong 2 phần, không thể cả hai, và số kí tự ở hai phần phải đúng bằng lần lượt là $\lceil \frac{n}{2} \rceil$ và $\lfloor \frac{n}{2} \rfloor$.
Đến đây, ta có thể nghĩ đến việc sử dụng quy hoạch động Knapsack. Định nghĩa trạng thái như sau: dp$[i][s]$ là số cách chia i chữ cái đầu vào 2 phần, sao cho phần chẵn có s kí tự (phần lẻ sẽ có $(C_1 + C_2 + \cdots + C_i) - s$ kí tự). Khi đó, kết quả bài toán chính là dp$[26][\lceil \frac{n}{2} \rceil]$. Ta chuyển trạng thái dựa vào việc chữ cái thứ i đặt vào phần chẵn hay phần lẻ:

- Phần chẵn: Nếu $C_i \leq s$, ta có s vị trí trống, trong đó có $\binom{s}{C_i}$ cách để đặt các chữ cái thứ i. $s - C_i$ vị trí còn lại dùng để đặt $i - 1$ chữ cái trước. Kết hợp lại, ta có $\binom{s}{C_i} \times \text{dp}[i - 1][s - C_i]$.
- Phần lẻ: Tương tự trên, nếu $s \leq C_1 + C_2 + \cdots + C_{i-1}$, ta có $(C_1 + C_2 + \cdots + C_i) - s$ vị trí trống, trong đó C_i vị trí dành cho chữ cái thứ i. Ta chuyển về dp$[i - 1][s]$, do s lưu số kí tự thuộc phần chẵn, trong khi kí tự i đang đặt vào phần lẻ. Tóm lại, ta có $\binom{(C_1+C_2+\cdots+C_i)-s}{C_i} \times \text{dp}[i - 1][s]$.

Kết hợp hai trường hợp bằng tổ hợp cộng, ta rút ra được công thức truy hồi:

$$\text{dp}[i][s] = \binom{s}{C_i} \times \text{dp}[i-1][s-C_i] + \binom{(C_1 + C_2 + \cdots + C_i) - s}{C_i} \times \text{dp}[i-1][s]$$

Nếu tính trước mảng giai thừa và nghịch đảo modulo của chúng, chuyển trạng thái sẽ mất $O(1)$. Có $O(n \times 26)$ trạng thái, nên độ phức tạp thời gian và bộ nhớ tổng cộng là $O(n \times 26)$. Lưu ý khi cài đặt, ta phải bỏ qua các i mà $C_i = 0$ (khi đó, ta chỉ cần gán $\text{dp}[i][s] = \text{dp}[i-1][s]$ và tiếp tục).

Cài đặt

```
// tính trước cnk

for (int i = 1; i <= 26; ++i)
    cin >> c[i];

int n = 0;
dp[0][0] = 1;
for (int i = 1; i <= 26; ++i){
    if (!c[i]){
        for (int s = 0; s <= n; ++s)
            dp[i][s] = dp[i - 1][s];
        continue;
    }

    n += c[i];
    for (int s = 0; s <= n; ++s){
        if (c[i] <= s) add(dp[i][s], 1ll * dp[i - 1][s - c[i]] * C(s, c[i]) % mod);
        if (s + c[i] <= n) add(dp[i][s], 1ll * dp[i - 1][s] * C(n - s, c[i]) % mod);
    }
}

cout << dp[26][(n + 1) / 2] << '\n';
```

Ví dụ 5.2

Cho n ($1 \le n \le 5 \cdot 10^3$) đỉnh được gán nhãn bằng các số nguyên phân biệt từ 1 đến n. Hãy xác định số lượng cây nhị phân (cây mà mỗi đỉnh có tối đa 2 con) n đỉnh có thể tạo ra. In ra đáp án *mod* $10^9 + 7$.

2 cây được gọi là khác nhau nếu thỏa 1 trong 2 điều kiện:

- 2 cây có hình dạng khác nhau.
- 2 cây có hình dạng cấu trúc giống nhau, nhưng tồn tại một đỉnh mà nhãn của nó ở 2 cây là khác nhau.

Ý tưởng

Tư tưởng quy hoạch động:

1. Chọn 1 đỉnh làm gốc của cây.
2. Chia các đỉnh còn lại thành 2 tập cho bên trái và phải.
3. Giải tiếp bài toán con cho 2 cây con trái và phải.

Gọi dp_i là đáp án cho bài toán cây có i đỉnh.

- Để chọn 1 đỉnh làm gốc cho cây, có i cách chọn.
- $i-1$ đỉnh còn lại chia làm 2 cây con với số lượng đỉnh là k và $i-1-k$:
  - Duyệt qua k từ 0 đến $i-1$ để xác định số đỉnh trong 2 cây con.
  - Khi đã có k, cần chọn ra k đỉnh từ $i-1$ đỉnh làm cây bên trái, số còn lại qua bên phải. Vậy số cách chọn là $\binom{i-1}{k}$.
  - Lúc này đã xác định được tập đỉnh ở 2 bên, vậy số cách sắp xếp cho 2 cây đó là $dp_k \cdot dp_{i-1-k}$.

Cuối cùng, rút ra được công thức: $dp_i = i \cdot \sum_{k=0}^{i-1} \binom{i-1}{k} \cdot dp_k \cdot dp_{i-1-k}$.
Đáp án cuối cùng là dp_n.

Độ phức tạp: $O(n^2)$

Cài đặt

```cpp
// tính trước cnk

long long dp[N];

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

    int n;
    cin >> n;
    dp[1] = 1;
    for(int i = 2; i <= n; ++i) {
        for(int k = 0; k < i; ++k)
        {
            dp[i] += nCk(i - 1, k) * dp[k] % mod * dp[i - 1 - k] % mod;
            dp[i] %= mod;
        }
        dp[i] *= i;
        dp[i] %= mod;
    }
    cout << dp[n];

    return 0;
}
```

Ví dụ 5.3

Cho một bảng $n * m$ ($n, m \leq 1e5$), có k điểm là những chướng ngại vật ($k \leq 2000$). IHFF muốn đi từ vị trí $(1,1)$ tới (n,m), do là người khá khỏe, nên khi gặp chướng ngại vật, IHFF sẽ cố gắng vượt qua, nhưng sau i lần ($i \leq 20$), IHFF hết sức nên sẽ không thể vượt qua thêm 1 chướng ngại vật nào nữa. Hãy giúp IHFF tìm số đường đi từ $(1,1)$ tới (n,m) mà IHFF không phải đi qua quá i chướng ngại vật.

Ý tưởng

Bài toán yêu cầu đếm số đường đi từ $(1,1)$ đến (n,m) trên một bảng $n \times m$ chỉ được đi sang phải hoặc đi xuống, sao cho IHFF không đi qua quá i chướng ngại vật. Vì n, m có thể lên tới 10^5, việc duyệt hoặc quy hoạch động trên

toàn bộ bảng là không khả thi. Tuy nhiên, số lượng chướng ngại vật $k \leq 2000$ và số lần có thể vượt qua $i \leq 20$ là khá nhỏ, nên ta chỉ cần xét đến các điểm đặc biệt này.

Ta sẽ xem mỗi chướng ngại vật như một điểm $p_x = (x_i, y_i)$, đồng thời tính cả điểm bắt đầu $(1, 1)$ và điểm kết thúc (n, m). Khi đó, ta chỉ cần quan tâm đến số đường đi giữa các điểm này. Số đường đi từ $a = (x_a, y_a)$ đến $b = (x_b, y_b)$ (chỉ được đi xuống hoặc sang phải) là:

$$\text{path}(a, b) = \begin{cases} \binom{b_x - a_x + b_y - a_y}{b_x - a_x}, & \text{nếu } a_x \leq b_x, a_y \leq b_y, \\ 0, & \text{ngược lại.} \end{cases}$$

Sau khi sắp xếp các chướng ngại vật theo tọa độ tăng dần, ta có thể xây dựng công thức quy hoạch động như sau:

$$dp[x][y] = \text{path}(p_x, (n, m)) - \sum_{z<x} \text{path}(p_x, p_z) \cdot dp[z][y] - \sum_{z<y} dp[x][z].$$

Công thức này loại bỏ các đường đi bị đếm trùng khi đi qua các chướng ngại vật trước đó (vế thứ hai) và chỉ giữ lại những đường có đúng y chướng ngại vật bị vượt qua (vế thứ ba).

Cuối cùng, kết quả là tổng số đường đi từ $(1, 1)$ đến (n, m) mà vượt qua không quá i chướng ngại vật:

$$\text{Kết quả} = \sum_{y=0}^{i} dp[0][y].$$

Độ phức tạp của thuật toán là $O(k^2 \cdot i)$, hoàn toàn phù hợp với giới hạn của đề.

Cài đặt

```cpp
// tính trước cnk

int n, m, k, i;
pair<int, int> points[N];
long long dp[M][LG], ans;

long long path(pair<int, int> a, pair<int, int> b) {
    if (a.first > b.first || a.second > b.second)
        return 0;
    return cnk[b.first - a.first + b.second - a.second][b.first - a.first];
}

void solve() {
    cin >> n >> m >> k >> i;
    points[0] = {1, 1};
    for (int x = 1; x <= k; x++) cin >> points[x].first >> points[x].second;
    sort(points, points + k + 1);
    reverse(points, points + k + 1);

    for (int x = 0; x <= k; x++) {
        for (int y = 0; y <= i; y++) {
            dp[x][y] = path(points[x], {n, m});
            for (int z = 0; z < x; z++) {
                dp[x][y] -= path(points[x], points[z]) * dp[z][y] % MOD;
                dp[x][y] = (dp[x][y] + MOD) % MOD;
            }
            for (int z = 0; z < y; z++) {
                dp[x][y] -= dp[x][z];
```

```
                dp[x][y] = (dp[x][y] + MOD) % MOD;
            }
        }
    }
    for (int y = 0; y <= i; y++) ans = (ans + dp[k][y]) % MOD;
    cout << ans << "\n";
}
```

6 Tài liệu tham khảo

6.1 Bài luyện tập

- CF 1946E - Girl Permutation
- CF 1929D - Sasha and a Walk in the City
- CF 1764D - Doremy's Pegging Game
- CF 1954D - Colored Balls
- CF 1749D - Counting Arrays
- CF 1984C2 - Magnitude (Hard)

Tài liệu

[1] VNOI Wiki. (n.d.). *Các kiến thức cơ bản về tổ hợp*. Retrieved January 15, 2026, from `https://wiki.vnoi.info/translate/he/Number-Theory-5`

[2] USACO Guide. (n.d.). *Combinatorics*. Retrieved January 15, 2026, from `https://usaco.guide/gold/combo?lang=cpp`

[3] VNOI Wiki. (n.d.). *Binomial coefficient*. Retrieved January 15, 2026, from `https://wiki.vnoi.info/algo/algebra/nCk`

[4] CP-Algorithms. (n.d.). *Stars and bars*. Retrieved January 15, 2026, from `https://cp-algorithms.com/combinatorics/stars_and_bars.html`