ICPC Team Reference Notebook

Nhóm: UMT.DuongNhuTruocKhiEmTonTai

Kỳ thi: ICPC Quốc Gia 2025

Thành viên: Đặng Phúc An Khang, Võ Ngọc Trâm Anh, Nguyễn Lê Đăng Khoa
Trường: Đại học Quản lý và Công nghệ Tp.HCM (UMT)
Liên hệ: ankhangluonvuituoi@gmail.com / 0967670770

Mục lục

1 Số học

1.1 Kiến thức cần nhớ

- $a \cdot b = \gcd(a,b) \cdot \text{lcm}(a,b)$
- $\gcd(a,b) = \gcd(b, a \bmod b)$ (Thuật toán Euclid)
- $a = bq + r,\ 0 \le r < |b|$ (Phép chia có dư)
- Luật đồng dư:
  - $a \equiv b \pmod m \Rightarrow a + c \equiv b + c \pmod m$
  - $a \equiv b \pmod m \Rightarrow ac \equiv bc \pmod m$
- Fermat nhỏ: $a^{p-1} \equiv 1 \pmod p$ (nếu p nguyên tố, $p \nmid a$)
- Nghịch đảo modulo: $a^{-1} \equiv a^{p-2} \pmod p$ (với p nguyên tố)
- Phân tích thừa số nguyên tố và số lượng ước: $d(n) = \prod(e_i + 1)$ nếu $n = \prod p_i^{e_i}$
- Số ước của $n = \prod p_i^{e_i}$: $d(n) = \prod(e_i + 1)$
- Số ước chung của a và b: $d(\gcd(a,b)) = \prod(\min(e_i, f_i) + 1)$
- Số ước của $n!$: $d(n!) = \prod_{p \le n}(\sum_{k=1}^{\infty} \lfloor \frac{n}{p^k} \rfloor + 1)$
- Tính divCount nhanh cho mọi số đến n: $O(n \log n)$ bằng sieve

1.2 Lũy thừa nhanh (powmod)

Tính $a^b \bmod m$ với logarit cơ số 2:

$$\text{pow}(a,b,m) = \begin{cases} 1 & b = 0 \\ \text{pow}(a^2 \bmod m, \lfloor b/2 \rfloor, m) & b \text{ chẵn} \\ a \cdot \text{pow}(a, b-1, m) \bmod m & b \text{ lẻ} \end{cases}$$

```
ll modpow(ll a, ll b, ll m){
    a %= m; ll r = 1;
    while(b){
        if(b & 1) r = (__int128)r * a % m;
        a = (__int128)a * a % m;
        b >>= 1;
    }
    return r;
}
```

1.3 BCNN của mảng

Cho mảng A gồm n phần tử nguyên. Tính: $LCM(A_1, A_2, \ldots, A_n) \mod 10^9 + 7$.

```
// Tính LCM(a1, a2, ..., an) theo modulo 1e9+7
// LCM = ∏p^{max_exp_p}
static const int MOD = 1'000'000'007;

long long mod_pow(long long a, long long e, long long mod = MOD) {
    long long r = 1 % mod;
    a %= mod;
    while (e > 0) {
        if (e & 1) r = (r * a) % mod;
        a = (a * a) % mod;
        e >>= 1;
    }
    return r;
}
// spf[x] = uoc nguyen to nho nhat cua x
vector<int> build_spf(int maxA) {
    vector<int> spf(maxA + 1);
    for (int i = 0; i <= maxA; ++i) spf[i] = i;
    for (int i = 2; 1LL * i * i <= maxA; ++i) {
        if (spf[i] == i) { // i is prime
            for (long long j = 1LL * i * i; j <= maxA; j += i) {
                if (spf[(int)j] == (int)j) spf[(int)j] = i;
            }
        }
    }
    return spf;
}
int main() {
    int n; cin >> n return 0;
    vector<int> a(n);
    int maxA = 0;
    for (int i = 0; i < n; ++i)
    { cin >> a[i]; maxA = max(maxA, a[i]); }

    if (maxA <= 1) { cout << (maxA == 0 ? 0 : 1) << '\n'; return 0; }
    vector<int> spf = build_spf(maxA);
    vector<int> maxExp(maxA + 1, 0);
    for (int x : a) {
        if (x <= 1) continue;
        while (x > 1) {
            int p = spf[x];
            int cnt = 0;
            while (x % p == 0) {
                x /= p;
                ++cnt;
            }
            if (cnt > maxExp[p]) maxExp[p] = cnt;
        }
    }
    // ans = ∏p^{maxExp[p]} (mod MOD)
    long long lcm_mod = 1;
    for (int p = 2; p <= maxA; ++p) {
        int e = maxExp[p];
        if (e > 0) {
            lcm_mod = (lcm_mod * mod_pow(p, e)) % MOD;
        }
    }
    cout << lcm_mod << '\n';
    return 0;
}
```

1.4 Tổng tổng tổng

Cho $f(i,j)$ là tổng tất cả các ước nguyên dương i, j. Tính tổng $S = \sum_{i=1}^{n} \sum_{j=1}^{m} f(i,j)$.

Phân tích

Gọi S là tổng trên mọi cặp $(i \le j \le N)$ của *tổng các ước số chung* của i và j. Ta đổi góc nhìn: với mỗi d, hãy đếm số cặp $(i \le j)$ đều là bội của d. Nếu đặt $t = \left\lfloor \frac{N}{d} \right\rfloor$ thì các bội dương của d trong $[1..N]$ là

$\{d, 2d, \ldots, td\}$ (có t phần tử). Số cặp $(i \le j)$ chọn từ t phần tử là $\binom{t+1}{2} = \frac{t(t+1)}{2}$.

Mỗi cặp như vậy đóng góp đúng d (vì d là một ước chung). Do đó $S = \sum_{d=1}^{N} d \cdot \left(\left\lfloor \frac{N}{d} \right\rfloor_2^{+1}\right)$.

Tối ưu hoá $O(\sqrt{N})$. Hàm $\left\lfloor \frac{N}{d} \right\rfloor$ nhận cùng một giá trị trên từng đoạn liên tiếp. Nếu tại L ta có $k = \left\lfloor \frac{N}{L} \right\rfloor$ thì giá trị này giữ nguyên cho mọi $d \in [L..R]$ với $R = \left\lfloor \frac{N}{k} \right\rfloor$.
Khi đó, đóng góp của đoạn là

$$\left(\sum_{d=L}^{R} d\right) \cdot \binom{k+1}{2} = \frac{(L+R)(R-L+1)}{2} \cdot \frac{k(k+1)}{2}.$$

Duyệt các đoạn bằng hai con trỏ $(L \leftarrow R+1)$ cho tới N. Tất cả phép tính thực hiện theo modulo 10^9+7.

```cpp
const int MOD = 1'000'000'007;
const int INV2 = (int(1e9) + 8) / 2; // = 500000004

// sum_{d=l}^r d (mod MOD)
static inline long long range_sum(long long l, long long r) {
    l %= MOD; if (l < 0) l += MOD;
    r %= MOD; if (r < 0) r += MOD;
    long long cnt = (r - l + 1) % MOD; if (cnt < 0) cnt += MOD;
    long long s = (l + r) % MOD;
    return ((s * cnt) % MOD) * INV2 % MOD;
}

int main() {
    long long N; cin >> N;

    long long ans = 0, at = 1;
    while (at <= N) {
        long long k = N / at; // k = floor(N / d)
        long long nxt = N / k; // d in [at, nxt] share same k

        // choose = C(k+1, 2) = k * (k + 1) / 2 mod MOD
        long long choose = ( (k % MOD) * ((k + 1) % MOD) ) % MOD;
        choose = (choose * INV2) % MOD;
        // sum d over [at..nxt]
        long long seg = range_sum(at, nxt);
        ans = (ans + seg * choose) % MOD;
        at = nxt + 1;
    }
    cout << ans << '\n';
}
```

2 Tổ hợp & Xác suất & Kỳ vọng

2.1 Kiến thức cơ bản

- **Giai thừa:** $n! = 1 \cdot 2 \cdot 3 \cdots n, \quad 0! = 1$
- **Chỉnh hợp (Permutation):** $A(n,k) = n \times (n-1) \times \cdots \times (n-k+1) = \frac{n!}{(n-k)!}$
- **Tổ hợp (Combination):** $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
- **Tổ hợp có lặp:** $\binom{n+k-1}{k}$ (chọn k phần tử từ n loại, cho phép trùng)
- **Công thức Pascal:** $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$
- **Nhị thức Newton:** $(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$
- **Tính modulo với số nguyên tố p:**

$$\binom{n}{k} \bmod p = n! \cdot (k!)^{-1} \cdot ((n-k)!)^{-1} \bmod p$$

$$a^{-1} \equiv a^{p-2} \pmod{p} \quad \text{(Fermat nhỏ)}$$

Kỳ vọng (Expected Value)

- **Định nghĩa tổng quát:** $E[X] = \sum_x x \cdot P(X = x)$
- **Nếu X là biến rời rạc nhận các giá trị $x_1, x_2, \ldots, x_k$:** $E[X] = x_1 P(X = x_1) + x_2 P(X = x_2) + \cdots + x_k P(X = x_k)$
- **Nếu X là tổng của nhiều biến độc lập $X = X_1 + X_2 + \cdots + X_n$:** $E[X] = E[X_1] + E[X_2] + \cdots + E[X_n]$
- **Tính tuyến tính (Linearity of Expectation) – rất quan trọng:** $E[X+Y] = E[X] + E[Y]$ (đúng không cần độc lập) $E[aX + b] = aE[X] + b$
- **Nếu X, Y độc lập:** $E[X \cdot Y] = E[X] \cdot E[Y]$
- **Biến Bernoulli ($X \in \{0,1\}$):** $P(X = 1) = p, \quad P(X = 0) = 1 - p, \quad E[X] = p$
- **Phân phối Nhị thức (Binomial):** $P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$ $E[X] = np$
- **Phân phối Hình học (Geometric) – số lần thử đến khi thành công:** $P(X = k) = (1-p)^{k-1} p$ $E[X] = \dfrac{1}{p}$
- **Biến chỉ thị (Indicator Variable):** Xét $X_i = 1$ nếu sự kiện A_i xảy ra, $X_i = 0$ nếu không. Khi đó $E[X_i] = P(A_i)$ và:

$$E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} P(A_i)$$

  (rất hay dùng để tính kỳ vọng số lần xảy ra của một sự kiện)
- **Ví dụ kinh điển trong thi đấu:**
  - Tung đồng xu n lần, kỳ vọng số lần xuất hiện mặt ngửa: $E = n \cdot \frac{1}{2}$
  - Một hoán vị ngẫu nhiên độ dài n, kỳ vọng số phần tử giữ nguyên vị trí (fixed points): $E = 1$
  - Bài toán thu thập đủ n món (Coupon Collector): $E = n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right) \approx n \ln n$

2.2 Tổng max của các dãy con độ dài k

Cho A gồm n phần tử. Giá trị của dãy con B là $\max(B)$. Tính $S = \sum_{B:\,|B|=k} \max(B) \pmod{10^9+7}$.

Ý tưởng. Sắp xếp A tăng dần: $a_1 \le a_2 \le \cdots \le a_n$. Phần tử a_i là *lớn nhất* của một dãy con độ dài k iff chọn $k-1$ phần tử bất kỳ trong $\{a_1, \ldots, a_{i-1}\}$.

$$\Rightarrow \text{ đóng góp của } a_i = a_i \cdot \binom{i-1}{k-1}.$$

Công thức.

$$S = \sum_{i=k}^{n} a_i \binom{i-1}{k-1} \pmod{10^9+7}.$$

2.3 Đường đi trong tam giác trên của lưới $n \times n$

Tóm tắt đề bài. Trên bảng $n \times n$, từ ô (x, y) $(x \le y)$, mỗi bước đi xuống hàng tiếp theo:

$$(i, j) \to (i+1, k) \quad \text{với } j \le k \le n.$$

Không được đi vào ô có $a > b$ (chỉ ở vùng $i \le j$). Hỏi có bao nhiêu cách đi đến (n, n). Có q truy vấn (x, y), in kết quả mod 998244353.

Phân tích. Mỗi đường đi gồm:

$$D = n - x \text{ bước xuống}, \qquad R = n - y \text{ bước sang phải}.$$

Điều kiện không đi vào $a > b$ tương đương **không được vượt xuống dưới đường chéo** $i = j$. Đây là bài toán đếm đường đi giới hạn bởi đường chéo → áp dụng *nguyên lý phản xạ (reflection principle)*.

Ký hiệu. Gọi $s = y - x \ge 0$. Tổng số đường đi không bị giới hạn:

$$\binom{D+R}{D}.$$

Số đường đi *vi phạm* (đi xuống dưới $i = j$) là:

$$\binom{D+R}{D-s-1}.$$

Công thức cuối: $\text{ans}(x, y) = \binom{D+R}{D} - \binom{D+R}{D-(y-x)-1} \pmod{998244353}$ với $D = n - x$, $R = n - y$. Nếu chỉ số $\binom{n}{k}$ âm hoặc $> n$ thì giá trị là 0.

2.4 Tổng thời gian ăn đào trên mọi cây (quy tắc mọc tuần tự)

Mô tả tóm tắt. Bắt đầu từ 1 quả (node gốc), mỗi bước gắn thêm 1 quả vào *một nhánh trống* bất kỳ dưới một quả đang có (mỗi quả tối đa 2 nhánh). Sau n bước thu được một cây nhị phân có hướng gốc (trái/phải phân biệt) với *thứ tự mọc* khác nhau được tính là các cây khác nhau. Thời gian ăn hết đào trên một cây bằng $\sum$ khoảng cách giữa mọi cặp quả (mỗi nhánh đi qua tốn 1). Yêu cầu: tổng thời gian này *trên tất cả các cây* modulo p.

Quan sát. - Số "lịch mọc" (cây phân biệt) là $n!$ (ở bước có t quả thì có $t+1$ nhánh trống để chọn). - Với một cây T, $\text{dist\_sum}(T) = \sum_e s_e(n - s_e)$ (cộng theo mỗi cạnh e, s_e là kích thước một phía khi cắt tại e). - Quy hoạch động không dùng chia (phù hợp modulo p bất kỳ): tách gốc với $|L|{=}i, |R|{=}j = n - 1 - i$. Số cách ghép: $\binom{n-1}{i} \cdot \text{Cnt}[i] \cdot \text{Cnt}[j]$, trong đó $\text{Cnt}[m]$ là tổng số lịch của kích thước m.

DP. Gọi:

$$\text{Cnt}[n] = \text{tổng số lịch}, \quad \text{SD}[n] = \sum(\text{tổng độ sâu các node}) \text{ trên mọi lịch}, \quad \text{T}[n] = \sum \text{dist\_sum}(T)$$

trên mọi lịch kích thước n. Với $\binom{\cdot}{\cdot}$ tính sẵn mod p, có:

$$\text{Cnt}[0] = 1,\ \text{SD}[0] = 0,\ \text{T}[0] = 0.$$

$$\text{Cnt}[n] = \sum_{i=0}^{n-1} \binom{n-1}{i} \text{Cnt}[i]\ \text{Cnt}[n-1-i],$$

$$\text{SD}[n] = \sum_{i=0}^{n-1} \binom{n-1}{i} \Big(\text{SD}[i]\ \text{Cnt}[j] + \text{SD}[j]\ \text{Cnt}[i] + (i{+}j)\ \text{Cnt}[i]\ \text{Cnt}[j]\Big),$$

$$\text{T}[n] = \sum_{i=0}^{n-1} \binom{n-1}{i} \Big(\text{T}[i]\ \text{Cnt}[j] + \text{T}[j]\ \text{Cnt}[i] + (j{+}1)\ (\text{SD}[i] + i\,\text{Cnt}[i])\ \text{Cnt}[j] + (i{+}1)\ (\text{SD}[j] + j\,\text{Cnt}[j])\ \text{Cnt}[i]\Big),$$

trong đó $j = n - 1 - i$. Đáp án cần in ra là $\boxed{\text{T}[n] \bmod p}$. Độ phức tạp $O(n^2)$.

2.5 Đếm cặp (x, y) sao cho dãy Fibonacci mở rộng chứa k (vị trí ≥ 2)

Mô tả. Cho $f_0 = x$, $f_1 = y$ (đều dương), $f_i = f_{i-1} + f_{i-2}$ với $i > 1$. Đếm số cặp (x, y) sao cho tồn tại $n \ge 2$ với $f_n = k$ (và k *không* là f_0 hay f_1). In kết quả mod $10^9{+}7$.

Phân tích. Với dãy Fibonacci chuẩn $F_0 = 0, F_1 = 1$, ta có

$$f_n = F_{n-1}\,x + F_n\,y \quad (n \ge 2).$$

Cần số nghiệm nguyên dương (x, y) của

$$F_{n-1}\,x + F_n\,y = k \quad (x, y \ge 1).$$

Đặt $a = F_{n-1}, b = F_n, k' = k - a - b$ và $x' = x - 1 \ge 0, y' = y - 1 \ge 0$:

$$ax' + by' = k'.$$

Với $\gcd(a, b) = 1$, số nghiệm $(x', y') \in \mathbb{Z}^2_{\ge 0}$ đếm bằng số y' thỏa

$$0 \le y' \le \left\lfloor \tfrac{k'}{b} \right\rfloor, \quad b\,y' \equiv k' \pmod{a}.$$

Gọi b^{-1} là nghịch đảo của b theo modulo a (tồn tại do $\gcd(a, b) = 1$), nghiệm nhỏ nhất

$$y_0 \equiv b^{-1}\,(k' \bmod a) \pmod{a}, \quad 0 \le y_0 < a.$$

Nếu $y_0 > \left\lfloor \frac{k'}{b} \right\rfloor$ thì #nghiệm $= 0$, ngược lại

$$\#\text{nghiệm} \ = \ \left\lfloor \frac{\lfloor k'/b \rfloor - y_0}{a} \right\rfloor + 1.$$

Điều kiện cần: $k \ge F_{n+1}$ (vì $x, y \ge 1$). Số n cần xét chỉ $O(\log k)$.

Thuật toán. Liệt kê $n = 2, 3, \ldots$ cho đến khi $F_{n+1} > k$, với mỗi n tính đóng góp như trên và cộng modulo $10^9{+}7$. Độ phức tạp $O(\log k)$.

3 Quy hoạch động & Tham lam

3.1 Dãy con tăng dài nhất (LIS)

Bài toán. Cho mảng A gồm n phần tử, tìm độ dài lớn nhất của dãy con tăng chặt:

$$i_1 < i_2 < \cdots < i_k, \quad A_{i_1} < A_{i_2} < \cdots < A_{i_k}.$$

Ý tưởng. Duy trì mảng chỉ số $\texttt{f}[L]$ là *chỉ số phần tử kết thúc nhỏ nhất* trong mọi dãy tăng *độ dài đúng* L đã thấy. So sánh luôn qua giá trị $A[\texttt{f}[L]]$.

Bước xử lý phần tử $A[i]$**.** Tìm $\texttt{pos} = \max\{L \mid A[\texttt{f}[L]] < A[i]\}$ bằng nhị phân trên $L \in [0, \texttt{len}]$. Khi đó có thể kéo dài dãy độ dài pos thành pos+1 bằng $A[i]$: $\texttt{f[pos+1]} \leftarrow i$. Nếu $\texttt{pos+1} > \texttt{len}$ thì cập nhật len.

Kết quả. len chính là độ dài LIS. Độ phức tạp $O(n \log n)$, bộ nhớ $O(n)$.

```
int main() {
    int n; cin >> n;
    vector<int> a(n + 1), f(n + 1, -1);
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    f[0] = 0;
```

```
    a[0] = 0;
    int len = 0;
    // f[len] : chi so cua phan tu ket thuc co gia tri nho nhat trong tat ca nhung day con tang
         co do dai = len tinh den thoi diem i
    for (int i = 1; i <= n; i++) {
       int l = 0, r = len, pos = 0;
       while (l <= r) {
          int mid = l + r >> 1;
          if (a[i] > a[f[mid]]) {
             pos = mid;
             l = mid + 1;
          }
          else {
             r = mid - 1;
          }
       }
       f[pos + 1] = i;
       if (pos + 1 > len) len++;
    }
    cout << len;
}
```

3.2 Xoá tối đa k đoạn về 0 để tối đa hoá tổng

Bài toán. Cho mảng A gồm n số nguyên. Thực hiện không quá k lần thao tác: chọn đoạn liên tiếp $[l, r]$ và gán $A_l, \ldots, A_r := 0$. Tìm tổng lớn nhất có thể của mảng sau các thao tác.

Quy hoạch động (cuộn lớp). Ký hiệu:

$$\text{dp}[i][j][t] = \text{tổng lớn nhất xét đến } i,\ \text{đã dùng } j \text{ lần xoá},\ t \in \{0, 1\}$$

trong đó $t = 0$: *không xoá tại vị trí i* (giữ A_i), $t = 1$: *đang ở trong một đoạn xoá* (nên A_i bị không cộng).

Chuyển trạng thái.

$$\text{dp}[i][j][0] = \max\big(\text{dp}[i-1][j][0] + A_i,\ \text{dp}[i-1][j][1] + A_i\big),$$
$$\text{dp}[i][j][1] = \max\big(\text{dp}[i-1][j-1][0],\ \text{dp}[i-1][j][1]\big).$$

Giải thích: hoặc không xoá tại i (cộng A_i), hoặc xoá tại i (không cộng A_i); bắt đầu đoạn xoá mới làm j tăng 1.

Khởi tạo. $\text{dp}[0][0][0] = 0$, các trạng thái còn lại $= -\infty$ (không hợp lệ). Trả lời: $\max_{0 \le j \le k} \max(\text{dp}[n][j][0], \text{dp}[n][j][1])$.

Độ phức tạp. $O(nk)$ thời gian, $O(k)$ bộ nhớ với cuộn lớp theo i.

```
using int64 = long long;
const int64 NEG_INF = (int64)-4e18;

int main() {
   int n, k; if (!(cin >> n >> k)) return 0;
   vector<int64> a(n+1);
   for (int i = 1; i <= n; ++i) cin >> a[i];
   // dp[cur/prev][j][t], t in {0(not deleting), 1(deleting)}
   vector<array<int64,2>> dp_prev(k+1), dp_cur(k+1);
   // init: dp[0][0][0] = 0; others = -INF
   for (int j = 0; j <= k; ++j) dp_prev[j] = {NEG_INF, NEG_INF};
   dp_prev[0][0] = 0;
   for (int i = 1; i <= n; ++i) {
      for (int j = 0; j <= k; ++j) dp_cur[j] = {NEG_INF, NEG_INF};
      for (int j = 0; j <= k; ++j) {
         // t = 0: keep A[i]
         if (dp_prev[j][0] != NEG_INF)
            dp_cur[j][0] = max(dp_cur[j][0], dp_prev[j][0] + a[i]);
         if (dp_prev[j][1] != NEG_INF)
            dp_cur[j][0] = max(dp_cur[j][0], dp_prev[j][1] + a[i]);
         // t = 1: delete at i -> A[i] contributes 0
         // continue deleting
         if (dp_prev[j][1] != NEG_INF)
            dp_cur[j][1] = max(dp_cur[j][1], dp_prev[j][1]);
         // start a new deletion segment here (use one operation)
         if (j > 0 && dp_prev[j-1][0] != NEG_INF)
            dp_cur[j][1] = max(dp_cur[j][1], dp_prev[j-1][0]);
      }
      swap(dp_prev, dp_cur);
   }

   int64 ans = NEG_INF;
   for (int j = 0; j <= k; ++j) {
      ans = max(ans, dp_prev[j][0]);
      ans = max(ans, dp_prev[j][1]);
   }
   cout << ans << '\n';
   return 0;
}
```

3.3 Balo nhiều vật (bounded knapsack) – tách nhị phân

Bài toán. Có n loại, loại i có trọng lượng w_i, giá trị v_i, số lượng a_i. Chọn tổng trọng lượng $\le S$ để giá trị lớn nhất.

Mỗi loại i có trọng lượng w_i, giá trị v_i, và số lượng tối đa a_i. Không được vượt quá sức chứa S.

Vấn đề: Nếu duyệt từng món trong a_i thì tốn $O(n \cdot a_i \cdot S)$ (quá lớn).

Cách làm:

- Ta tách số lượng a_i thành các nhóm theo dạng nhị phân: $a_i = 1 + 2 + 4 + \cdots +$ (phần dư)
- Mỗi nhóm được coi như một món 0/1:$W = w_i \cdot$ nhóm, $\quad V = v_i \cdot$ nhóm
- Tổng số món sau khi tách chỉ khoảng $\sum \log_2 a_i \le 1400$.
- Sau đó giải như bài toán **balo 0/1**: $dp[s] = \max(dp[s],\ dp[s - W] + V)$

```
#include <bits/stdc++.h>
using namespace std;

int main() {
   ios::sync_with_stdio(false);
   cin.tie(nullptr);

   int n, S;
   if (!(cin >> n >> S)) return 0;

   struct Item { int w; long long v; };
   vector<Item> items;
   items.reserve(n * 15);

   for (int i = 0; i < n; ++i) {
      int w, v, a;
```

```
        cin >> w >> v >> a;
        for (int k = 1; a > 0; k <<= 1) {
            int take = min(k, a);
            a -= take;
            items.push_back({ w * take, 1LL * v * take });
        }
    }
    vector<long long> dp(S + 1, 0);
    for (auto &it : items) {
        int W = it.w; long long V = it.v;
        if (W > S) continue;
        for (int s = S; s >= W; --s) {
            dp[s] = max(dp[s], dp[s - W] + V);
        }
    }
    cout << *max_element(dp.begin(), dp.end()) << '\n';
    return 0;
}
```

3.4 Hai lộ trình ăn nấm: đi $(1,1) \to (n,n)$ rồi quay về tối đa hoá tổng

Tóm tắt. Từ $(1,1)$ đi đến (n,n) chỉ bằng bước `Down/Right`, rồi quay về $(1,1)$ chỉ bằng `Up/Left`. Mỗi ô chỉ được hái nấm **tối đa một lần**. Tính tổng nấm lớn nhất có thể hái.

Phân tích (quy về 2 lộ trình đồng thời). Lộ trình đi và lộ trình về có thể xem như *hai người* cùng đi từ $(1,1)$ đến (n,n), chỉ bước `Down/Right`, và tổng thưởng là:

$$\text{gain} = A_{i_1,j_1} + (i_1 \neq i_2 \ ? \ A_{i_2,j_2} : 0).$$

Tại "thời điểm" $k = i_1 + j_1 = i_2 + j_2$, đặt $j_1 = k - i_1$, $j_2 = k - i_2$. Quy hoạch động:

$$\text{dp}[i_1][i_2] = \max \begin{cases} \text{dp}[i_1-1][i_2-1], \\ \text{dp}[i_1-1][i_2], \\ \text{dp}[i_1][i_2-1], \\ \text{dp}[i_1][i_2] \end{cases} + \ A_{i_1,j_1} + \big(i_1 \neq i_2\big) \cdot A_{i_2,j_2}.$$

Duyệt $k = 2..2n$, mỗi k cập nhật dp theo các chỉ số hợp lệ ($1 \le i_1, i_2 \le n$, $1 \le j_1, j_2 \le n$).

Độ phức tạp. $O(n^3)$ thời gian, $O(n^2)$ bộ nhớ (cuộn theo k).

```
int main() {
    int n; if (!(cin >> n)) return 0;
    vector<vector<long long>> a(n+1, vector<long long>(n+1));
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= n; ++j)
            cin >> a[i][j];

    const long long NEG = -(1LL<<60);
    vector<vector<long long>> dp(n+1, vector<long long>(n+1, NEG));
    dp[1][1] = a[1][1]; // k = 2 (i+j)

    // k = i1 + j1 = i2 + j2
    for (int k = 3; k <= 2*n; ++k) {
        vector<vector<long long>> ndp(n+1, vector<long long>(n+1, NEG));
        for (int i1 = 1; i1 <= n; ++i1) {
            int j1 = k - i1;
            if (j1 < 1 || j1 > n) continue;
            for (int i2 = 1; i2 <= n; ++i2) {
                int j2 = k - i2;
                if (j2 < 1 || j2 > n) continue;

                long long best = NEG;
                // 4 ways to come: (i1-1 or i1) x (i2-1 or i2) from previous k-1
                if (i1 > 1 && i2 > 1) best = max(best, dp[i1-1][i2-1]);
                if (i1 > 1) best = max(best, dp[i1-1][i2]);
                if (i2 > 1) best = max(best, dp[i1][i2-1]);
                                    best = max(best, dp[i1][i2]); // both came from right

                if (best == NEG) continue;
                long long gain = a[i1][j1] + ( (i1==i2 && j1==j2) ? 0 : a[i2][j2] );
                ndp[i1][i2] = max(ndp[i1][i2], best + gain);
            }
        }
        dp.swap(ndp);
    }

    cout << dp[n][n] << '\n';
    return 0;
}
```

3.5 Đường đi đơn lớn nhất trên cây (node-weighted)

Bài toán. Cho cây n đỉnh, mỗi đỉnh i có trọng số A_i (có thể âm). Tìm tổng lớn nhất của một *đường đi đơn* bất kỳ.

Phân tích. Gọi $down[u]$ là *tổng lớn nhất của một đường đi đi xuống* từ u (chỉ chọn đúng một nhánh con), bao gồm u:

$$down[u] = A_u + \max\{0, \max_{v \in \text{con}(u)} down[v]\}.$$

Đường đi tốt nhất *đi qua* u là $A_u + \max(0, \text{best}_1) + \max(0, \text{best}_2)$, với $\text{best}_1, \text{best}_2$ là hai giá trị $down[v]$ lớn nhất từ các con của u (cắt âm về 0). Đáp án là max trên mọi u.

Cài đặt. Duyệt cây theo hậu tố (postorder) bằng duyệt không đệ quy: lấy thứ tự DFS, xử lý ngược lại để tính $down$ và cập nhật đáp án. Thời gian $O(n)$, bộ nhớ $O(n)$.

```
int main() {
    int n; if (!(cin >> n)) return 0;
    vector<long long> A(n+1);
    for (int i = 1; i <= n; ++i) cin >> A[i];

    vector<vector<int>> g(n+1);
    for (int i = 0; i < n-1; ++i) {
        int u, v; cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }

    // iterative DFS to avoid stack overflow
    vector<int> parent(n+1, 0), order;
    order.reserve(n);
    stack<int> st;
    st.push(1); parent[1] = 0;
    while (!st.empty()) {
        int u = st.top(); st.pop();
        order.push_back(u);
```

```
        for (int v : g[u]) if (v != parent[u]) {
            parent[v] = u;
            st.push(v);
        }
    }

    vector<long long> down(n+1, LLONG_MIN/4);
    long long ans = LLONG_MIN/4;

    // process nodes in reverse order = postorder
    for (int idx = (int)order.size()-1; idx >= 0; --idx) {
        int u = order[idx];
        long long best1 = 0, best2 = 0; // top two non-negative child downs
        for (int v : g[u]) if (v != parent[u]) {
            long long c = max(0LL, down[v]);
            if (c > best1) { best2 = best1; best1 = c; }
            else if (c > best2) { best2 = c; }
        }
        down[u] = A[u] + best1;
        long long through = A[u] + best1 + best2;
        ans = max(ans, through);
    }
}
```

3.6 Đếm số đồ thị con liên thông (subtree connected) của cây

Bài toán. Cho cây n đỉnh. Một *đồ thị con* ở đây là tập đỉnh liên thông (không rỗng). Đếm số đồ thị con, mod 10^9+7.

Phân tích. Gốc hoá cây tại 1. Gọi

$$\mathrm{dp}[u] = \text{ số tập đỉnh liên thông (không rỗng) nằm trong } \textit{duy nhất} \text{ nhánh con của } u \text{ và } \textit{chứa } u \ .$$

Khi đó với mỗi con v của u, ta có hai lựa chọn độc lập: *không* lấy nhánh v hoặc *lấy* một tập liên thông chứa v (có $\mathrm{dp}[v]$ cách). Do đó:

$$\boxed{\mathrm{dp}[u] \ = \prod_{v \in \mathrm{child}(u)} \big(1 + \mathrm{dp}[v]\big) \mod M}.$$

Mọi tập liên thông có *đỉnh gần gốc nhất* là duy nhất, nên tổng số đồ thị con chính là

$$\boxed{\mathrm{Ans} \ = \sum_{u=1}^{n} \mathrm{dp}[u] \mod M}, \quad M = 10^9+7.$$

Độ phức tạp. $O(n)$ thời gian, $O(n)$ bộ nhớ (duyệt hậu tố).

```
static const long long MOD = 1'000'000'007LL;
int main() {
    int n; if (!(cin >> n)) return 0;
    vector<vector<int>> g(n+1);
    for (int i = 0; i < n-1; ++i) {
        int u, v; cin >> u >> v;
        g[u].push_back(v); g[v].push_back(u);
    }

    // Iterative DFS to get parent and postorder
    vector<int> parent(n+1, 0), order; order.reserve(n);
    stack<int> st; st.push(1); parent[1] = -1;
    while (!st.empty()) {
        int u = st.top(); st.pop();
        order.push_back(u);
        for (int v : g[u]) if (v != parent[u]) {
            parent[v] = u;
            st.push(v);
        }
    }
    vector<long long> dp(n+1, 1); // initialize dp[u]=1 (leaf -> 1)
    // process in reverse (postorder)
    for (int idx = (int)order.size()-1; idx >= 0; --idx) {
        int u = order[idx];
        long long ways = 1;
        for (int v : g[u]) if (v != parent[u]) {
            ways = (ways * (1 + dp[v]) ) % MOD;
        }
        dp[u] = ways;
    }
    long long ans = 0;
    for (int u = 1; u <= n; ++u) {
        ans += dp[u];
        if (ans >= MOD) ans -= MOD;
    }
    cout << ans % MOD << '\n';
    return 0;
}
```

3.7 Đếm số đường đi độ dài k trên cây

Đề bài tóm tắt. Cho cây n đỉnh. Đường đi đơn là chuỗi các đỉnh phân biệt, mỗi cặp kề nhau nối bằng cạnh. Với mỗi truy vấn n, k, hãy đếm số lượng đường đi đơn có *đúng* k cạnh.

Ràng buộc. $1 \le n \le 10^5,\ 1 \le k \le 100$. Cây không có chu trình.

Ý tưởng (DP trên cây).

- Gốc cây bất kỳ, ví dụ tại đỉnh 1.
- Với mỗi đỉnh u, lưu mảng:

$$dp_u[d] = \text{số đỉnh trong subtree của } u \text{ cách } u \text{ đúng } d \text{ cạnh}$$

- Khởi tạo: $dp_u[0] = 1$ (tự bản thân nó).
- Khi gộp con v vào u:
  - **Đường đi qua u:** một đầu ở nhánh đã xử lý (dp_u), đầu còn lại trong subtree(v):

$$\mathrm{ans} \mathrel{+}= \sum_{j=0}^{k-1} dp_u[j] \cdot dp_v[k-1-j]$$

    (Đi từ u xuống v tốn thêm 1 cạnh, nên tổng là $j + 1 + t = k$).
  - **Cập nhật dp:** mọi nút trong subtree v sẽ xa u hơn 1 cạnh:

$$dp_u[d] \mathrel{+}= dp_v[d-1] \quad \forall d = 1..k$$

- Duyệt cây theo hậu tự (post-order). Đáp án là tổng ans.

```cpp
int main() {
    int n, K; if (!(cin >> n >> K)) return 0;
    vector<vector<int>> g(n+1);
    for (int i = 0; i < n-1; ++i) {
        int u, v; cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    vector<int> parent(n+1, 0), order;
    order.reserve(n);
    stack<int> st;
    st.push(1); parent[1] = -1;
    while (!st.empty()) {
        int u = st.top(); st.pop();
        order.push_back(u);
        for (int v : g[u]) if (v != parent[u]) {
            parent[v] = u;
            st.push(v);
        }
    }
    reverse(order.begin(), order.end());

    long long ans = 0;
    vector<vector<long long>> dp(n+1, vector<long long>(K+1, 0));
    for (int u : order) {
        dp[u][0] = 1; // chính u, ảkhong cách 0
        for (int v : g[u]) if (v != parent[u]) {
            for (int j = 0; j <= K-1; ++j) {
                int t = K-1-j; // độsâu trong dp[v] ướtrc khi ịdch
                ans += dp[u][j] * dp[v][t];
            }
            for (int d = 1; d <= K; ++d) {
                dp[u][d] += dp[v][d-1];
            }
        }
    }

    cout << ans << '\n';
    return 0;
}
```

3.8 Số thứ k có tổng chữ số là số nguyên tố

Bài toán. Liệt kê các số dương có tổng chữ số nguyên tố: $2, 3, 5, 7, 11, 12, 14, 16, \ldots$. Cho k, tìm *số thứ* k trong dãy (đếm từ 1).

Ý tưởng. Dùng *Digit DP* để đếm

$$F(X) = \#\{1 \le n \le X \mid \text{sumDigits}(n) \text{ là số nguyên tố}\}.$$

Sau đó *nhị phân* tìm X nhỏ nhất sao cho $F(X) \ge k$. Tổng chữ số tối đa $\le 9 \cdot 19 = 171$ (với $X < 10^{19}$), nên tiền xử lý prime đến 171.

Độ phức tạp. Mỗi lần đếm $O(\text{digits} \cdot \text{maxSum}) \approx 19 \cdot 171$, nhị phân ≈ 60 bước $\Rightarrow$ rất nhanh.

```cpp
const int MAX_SUM = 9 * 19; // 171
// Sieve primes up to 171
vector<int> isPrimeVec() {
    int N = MAX_SUM;
    vector<int> prime(N+1, 1);
    prime[0] = prime[1] = 0;
    for (int p = 2; p * p <= N; ++p)
        if (prime[p])
            for (int q = p*p; q <= N; q += p) prime[q] = 0;
    return prime;
}

struct DigitDP {
    vector<int> prime;
    // memo[pos][sum][tight?] is too big if 3D; we compress tight by separate calls
    long long memo[20][MAX_SUM+1];
    bool vis[20][MAX_SUM+1];
    vector<int> digs;

    DigitDP(): prime(isPrimeVec()) {}

    long long dfs(int pos, int sum, bool tight) {
        if (pos == (int)digs.size())
            return prime[sum]; // full number formed: count if sum is prime

        if (!tight && vis[pos][sum]) return memo[pos][sum];

        int limit = tight ? digs[pos] : 9;
        long long res = 0;
        for (int d = 0; d <= limit; ++d) {
            res += dfs(pos+1, sum + d, tight && (d == limit));
        }

        if (!tight) {
            vis[pos][sum] = true;
            memo[pos][sum] = res;
        }
        return res;
    }

    // count numbers in [0..x] whose digit-sum is prime
    long long count_upto(unsigned long long x) {
        digs.clear();
        if (x == 0) { // only 0
            return 0; // 0 is not counted (sequence starts from positive integers)
        }
        // build digits (most significant first)
        vector<int> tmp;
        while (x > 0) { tmp.push_back(int(x % 10)); x /= 10; }
        reverse(tmp.begin(), tmp.end());
        digs = move(tmp);
        // clear memo
        memset(vis, 0, sizeof(vis));
        return dfs(0, 0, true) - 0 /* exclude 0 explicitly */;
    }
};

int main() {
    unsigned long long k;
    if (!(cin >> k)) return 0;

```

```
    if (k == 0) {
        cout << 0 << '\n';
        return 0;
    }
    DigitDP dp;
    // exponential search to find high bound with count >= k
    unsigned long long lo = 1, hi = 1;
    while (dp.count_upto(hi) < (long long)k) {
        if (hi > (unsigned long long)4e18) break;
        hi <<= 1;
    }
    // binary search
    while (lo < hi) {
        unsigned long long mid = lo + ((hi - lo) >> 1);
        if (dp.count_upto(mid) >= (long long)k) hi = mid;
        else lo = mid + 1;
    }
    cout << lo << '\n';
    return 0;
}
```

3.9 Đếm số không có xâu con palindrome (độ dài > 1)

Tóm tắt. Đếm các số trong $[L, R]$ sao cho trong biểu diễn thập phân *không có* xâu con liên tiếp nào là palindrome độ dài ≥ 2.

Phân tích. Mọi palindrome độ dài ≥ 4 đều chứa một palindrome độ dài 2 hoặc 3 ở giữa. Vì thế **chỉ cần cấm** hai mẫu tối thiểu:

$$\text{(i) } d_i \neq d_{i+1} \quad \text{và} \quad \text{(ii) } d_i \neq d_{i+2}.$$

Đếm bằng **digit DP**:

$$\text{DP[pos][prev1][prev2][started][tight]}$$

với $prev1$ là chữ số liền trước, $prev2$ là chữ số cách 2 trước đó (giá trị đặc biệt "_" khi chưa bắt đầu). Khi chọn chữ số x ở vị trí hiện tại (nếu đã bắt đầu), ràng buộc: $x \neq prev1$ và $x \neq prev2$.

Đáp án: $F(R) - F(L-1)$.

```

struct Solver {
    vector<int> dig; // most significant -> least
    // dp[pos][prev1][prev2][started][tight]
    // prev1, prev2 in [0..9], or 10 = NONE
    long long dp[20][11][11][2][2];
    bool vis[20][11][11][2][2];

    long long dfs(int pos, int prev1, int prev2, int started, int tight) {
        if (pos == (int)dig.size()) {
            // valid number if we've started (or allow 0 as started==0 -> treat as number 0)
            return 1; // leading all zeros -> counts as 0
        }
        long long &memo = dp[pos][prev1][prev2][started][tight];
        if (vis[pos][prev1][prev2][started][tight]) return memo;
        vis[pos][prev1][prev2][started][tight] = true;
        long long res = 0;

        int up = tight ? dig[pos] : 9;
        for (int x = 0; x <= up; ++x) {
            int ntight = tight && (x == up);
            if (!started) {
                if (x == 0) {
                    // still not started: prevs remain NONE
                    res += dfs(pos + 1, 10, 10, 0, ntight);
                } else {
                    // start number with digit x: no constraints yet
                    res += dfs(pos + 1, x, 10, 1, ntight);
                }
            } else {
                // must satisfy x != prev1 and x != prev2
                if (x == prev1) continue;
                if (x == prev2) continue;
                res += dfs(pos + 1, x, prev1, 1, ntight);
            }
        }
        memo = res;
        return res;
    }

    long long count_upto(unsigned long long N) {
        dig.clear();
        if (N == 0) {
            // special: number 0 has no pal substrings (length >1), valid
            return 1;
        }
        while (N > 0) {
            dig.push_back(int(N % 10));
            N /= 10;
        }
        reverse(dig.begin(), dig.end());
        memset(vis, 0, sizeof(vis));
        return dfs(0, 10, 10, 0, 1);
    }
};

int main() {
    unsigned long long L, R;
    if (!(cin >> L >> R)) return 0;
    Solver sv;
    auto fR = sv.count_upto(R);
    auto fL = (L == 0) ? 0LL : sv.count_upto(L - 1);
    cout << (fR - fL) << '\n';
    return 0;
}
```

3.10 Đếm $y \in [L, R]$ sao cho $x \oplus y \leq S$ (digit DP theo bit)

Tóm tắt. Cho $x, L, R, S \leq 10^{18}$. Đếm số y thỏa $L \leq y \leq R$ và $x \oplus y \leq S$.

Ý tưởng. Dùng *bit-DP* trên 60 bit cao xuống thấp. Tính

$$F(N) = \#\{\, 0 \leq y \leq N : \; x \oplus y \leq S \,\}, \quad \text{đáp án } = F(R) - F(L-1).$$

Trạng thái:

$$\text{dp}[i][t_y][t_z]$$

với i là vị trí bit (từ cao xuống), t_y là cờ "đang chặt theo N", t_z là cờ "đang chặt theo S". Tại bit i, chọn $y_i \in \{0, 1\}$ không vượt N_i nếu $t_y = 1$. Khi đó $z_i = x_i \oplus y_i$ không vượt S_i nếu $t_z = 1$. Cập nhật cờ chặt mới theo so sánh bằng.

```cpp
struct BitDP {
    // PreLoaded bit arrays for N (upper bound on y) and S (upper bound on x^y)
    int NB[64], SB[64], XB[64];
    long long dp[64][2][2];
    bool vis[64][2][2];

    long long dfs(int i, int ty, int tz) {
        if (i < 0) return 1; // all bits assigned
        if (vis[i][ty][tz]) return dp[i][ty][tz];
        vis[i][ty][tz] = true;
        long long res = 0;

        int nbi = NB[i], sbi = SB[i], xbi = XB[i];
        for (int yi = 0; yi <= 1; ++yi) {
            if (ty && yi > nbi) continue; // exceed N at this bit
            int zi = yi ^ xbi;
            if (tz && zi > sbi) continue; // exceed S at this bit
            int nty = ty && (yi == nbi);
            int ntz = tz && (zi == sbi);
            res += dfs(i - 1, nty, ntz);
        }
        return dp[i][ty][tz] = res;
    }

    long long count_upto(unsigned long long x, unsigned long long N, unsigned long long S) {
        // fill bit arrays (LSB at index 0)
        for (int i = 0; i < 64; ++i) {
            NB[i] = (N >> i) & 1ULL;
            SB[i] = (S >> i) & 1ULL;
            XB[i] = (x >> i) & 1ULL;
        }
        memset(vis, 0, sizeof(vis));
        return dfs(63, 1, 1);
    }
};

int main() {
    unsigned long long x, L, R, S;
    if (!(cin >> x >> L >> R >> S)) return 0;

    BitDP solver;
    auto FR = solver.count_upto(x, R, S);
    auto FL = (L == 0 ? 0LL : solver.count_upto(x, L - 1, S));
    cout << (FR - FL) << '\n';
    return 0;
}
```

3.11 Sắp xếp đồ vào thùng (Bin Packing với $n \leq 20$)

Đề bài. Marisa có n món đồ, mỗi món nặng w_i. Mỗi thùng chịu tải tối đa W. Mỗi thùng có thể chứa nhiều món nhưng tổng khối lượng không vượt W. Hỏi cần ít nhất bao nhiêu thùng để chứa hết n món.

Ràng buộc:

$$1 \leq n \leq 20, \quad 1 \leq w_i \leq W \leq 10^9.$$

Ý tưởng.

Vì $n \leq 20$, dùng *bitmask DP*. Với mỗi tập con các món đồ (mask), lưu:

$$\text{dp}[mask] = (b, s)$$

trong đó b là số thùng đã dùng, s là tổng khối lượng trong thùng đang mở cuối cùng. Trạng thái tối ưu là trạng thái có b nhỏ hơn, hoặc nếu b bằng nhau thì s nhỏ hơn.

Chuyển trạng thái: với mỗi món i thuộc $mask$, xét $prev = mask \setminus \{i\}$. Nếu $s + w_i \leq W$ thì bỏ vào thùng hiện tại; ngược lại phải mở thùng mới.

$$\text{dp}[mask] = \begin{cases} (b,\ s + w_i) & \text{nếu } s + w_i \leq W, \\ (b + 1,\ w_i) & \text{nếu vượt } W. \end{cases}$$

Khởi tạo:

$$\text{dp}[0] = (1,\ 0).$$

Kết quả là $\text{dp}[(1 \ll n) - 1].b$.

Độ phức tạp. $O(n \cdot 2^n)$, phù hợp với $n \leq 20$.

```cpp
// dp[mask] = {so thung da dung, khoi luong hien tai trong thung cuoi}
pair<int,ll> dp[1 << 20];
int main() {
    int n; ll W;
    cin >> n >> W;
    vector<ll> w(n);
    for(int i = 0; i < n; i++) cin >> w[i];

    int N = 1 << n;
    dp[0] = {1, 0};

    for(int mask = 1; mask < N; mask++){
        dp[mask] = {n + 1, 0};
        for(int i = 0; i < n; i++){
            if(mask & (1 << i)){
                int pm = mask ^ (1 << i);
                auto [bins, curWeight] = dp[pm];
                if(curWeight + w[i] <= W){
                    dp[mask] = min(dp[mask], {bins, curWeight + w[i]});
                }
                else {
                    dp[mask] = min(dp[mask], {bins + 1, w[i]});
                }
            }
        }
    }

    cout << dp[N - 1].first << "\n";
    return 0;
}
```

3.12 TSP – Quay lui (Backtracking) với cắt tỉa

Bài toán. Cho ma trận chi phí $A_{i,j}$ (có thể bất đối xứng), tìm chu trình chi phí nhỏ nhất đi qua mỗi thành phố đúng một lần và quay lại thành phố xuất phát (cố định thành phố 0).

Ý tưởng quay lui + cắt tỉa. Dùng DFS xây lộ trình từ 0, giữ:

- `mask`: tập các đỉnh đã đi,
- `u`: đỉnh hiện tại,
- `cost`: chi phí hiện tại,
- `best`: đáp án tốt nhất.

Cắt tỉa bằng *cận dưới* đơn giản: đặt $g = \min_{i \neq j} A_{i,j}$, số cạnh còn lại (kể cả cạnh về 0) là $r = (n - \text{cnt}) + 1$.

Nếu

$$\texttt{cost} + g \cdot r \geq \texttt{best}$$

thì bỏ nhánh. Duyệt các thành phố tiếp theo theo thứ tự chi phí tăng dần để sớm cập nhật `best`.

Độ phức tạp. Tệ nhất $O(n!)$, nhưng với $n \leq 10$ và cắt tỉa + sắp thứ tự, chạy tốt.

```
int main() {
    int n; if (!(cin >> n)) return 0;
    vector<vector<long long>> A(n, vector<long long>(n));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            cin >> A[i][j];

    // global minimum edge cost (exclude self-loops)
    long long gmin = LLONG_MAX;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (i != j) gmin = min(gmin, A[i][j]);

    // Pre-sort candidate next nodes for each u by cost A[u][v] ascending
    vector<vector<int>> order(n);
    for (int u = 0; u < n; ++u) {
        order[u].resize(n-1);
        int idx = 0;
        for (int v = 0; v < n; ++v) if (v != u) order[u][idx++] = v;
        sort(order[u].begin(), order[u].end(), [&](int x, int y){
            return A[u][x] < A[u][y];
        });
    }

    const int START = 0;
    long long best = (long long)4e18;

    function<void(int,int,long long)> dfs = [&](int u, int mask, long long cost) {
        int cnt = __builtin_popcount((unsigned)mask);
        // bound: remaining edges = (n - cnt) to visit all + 1 to return
        int remEdges = (n - cnt) + 1;
        if (cost + gmin * remEdges >= best) return;

        if (cnt == n) {
            best = min(best, cost + A[u][START]); // close the tour
            return;
        }
        // try next cities in ascending edge cost
        for (int v : order[u]) {
            if (mask & (1 << v)) continue;
            dfs(v, mask | (1 << v), cost + A[u][v]);
        }
    };

    dfs(START, 1 << START, 0LL);
    cout << best << '\n';
    return 0;
}
```

4 Cây phân đoạn (Segment Tree)

4.1 Đếm số mảng con có tổng không âm (Prefix Sum + Segment Tree)

Đề bài. Cho mảng A gồm n số nguyên. Một *mảng con* $A[l..r]$ hợp lệ nếu:

$$\sum_{i=l}^{r} A[i] \geq 0.$$

Hãy đếm số mảng con như vậy.

Ràng buộc:

$$1 \leq n \leq 10^5, \quad |A_i| \leq 10^9.$$

Ý tưởng. Dùng tổng tiền tố:

$$P_0 = 0, \quad P_i = A_1 + \cdots + A_i.$$

Khi đó:

$$\sum_{l}^{r} \geq 0 \iff P_r - P_{l-1} \geq 0 \iff P_{l-1} \leq P_r.$$

Với mỗi r, cần đếm bao nhiêu P_{l-1} trước đó thỏa $P_{l-1} \leq P_r$.

Do P_i có thể lớn, ta **nén giá trị** $P_0..P_n$ rồi dùng **Segment Tree (Fenwick cũng được)** để:

- Cộng 1 vào vị trí của P_r (đánh dấu đã xuất hiện).
- Truy vấn tổng đoạn $[1..\text{pos}(P_r)]$ để biết số $P_{l-1} \leq P_r$.

Độ phức tạp. $O(n \log n)$.

```
struct SegTree {
    int n;
    vector<long long> st;
    SegTree(int n=0): n(n), st(4*n+5, 0) {}
    void add(int p, long long v, int id, int l, int r) {
        if (l == r) { st[id] += v; return; }
        int m = (l + r) >> 1;
        if (p <= m) add(p, v, id<<1, l, m);
        else add(p, v, id<<1|1, m+1, r);
        st[id] = st[id<<1] + st[id<<1|1];
    }
    void add(int p, long long v){ add(p, v, 1, 1, n); }
    long long sum(int L, int R, int id, int l, int r) {
        if (R < l || r < L) return 0;
        if (L <= l && r <= R) return st[id];
        int m = (l + r) >> 1;
        return sum(L,R,id<<1,l,m) + sum(L,R,id<<1|1,m+1,r);
    }
    long long sum(int L, int R){ if(L>R) return 0; return sum(L,R,1,1,n); }
};

int main() {
    int n; if (!(cin >> n)) return 0;
    vector<long long> a(n+1,0);
```

```
    for (int i = 1; i <= n; ++i) cin >> a[i];
    // prefix sums
    vector<long long> pref(n+1, 0);
    for (int i = 1; i <= n; ++i) pref[i] = pref[i-1] + a[i];
    // coordinate compression of pref[0..n]
    vector<long long> all = pref;
    sort(all.begin(), all.end());
    all.erase(unique(all.begin(), all.end()), all.end());
    auto idx = [&](long long x){
        return int(lower_bound(all.begin(), all.end(), x) - all.begin()) + 1; // 1-based
    };
    SegTree st((int)all.size());
    long long ans = 0;
    for (int r = 0; r <= n; ++r) {
        int id = idx(pref[r]);
        // count previous prefixes <= pref[r]
        ans += st.sum(1, id);
        // add current prefix for future
        st.add(id, 1);
    }
    cout << ans << '\n';
    return 0;
}
```

4.2 Phủ đoạn tối thiểu bằng segtree (range-chmin & point query)

Bài toán. Có q gói (l_i, r_i, c_i), chọn tối thiểu chi phí để phủ đủ $[1..n]$; nếu không thể, in -1.

Ý tưởng với DP + Segment Tree. Gọi $dp[i]$ là chi phí nhỏ nhất để phủ đủ $[1..i]$. Khi xét vị trí i, mọi gói bắt đầu tại i (tức $l = i$) sẽ cho ta một *ứng viên* chi phí $dp[i-1]+c$ phủ được *toàn bộ* các vị trí từ i đến r. Vì vậy, với mỗi gói (i, r, c), ta **cập nhật** trên đoạn $[i, r]$ giá trị min-assign bằng $dp[i-1]+c$. Khi đó $dp[i]$ chính là giá trị nhỏ nhất đang "phủ" tại điểm i.

Cụ thể.

$$dp[0] = 0, \quad dp[i] = \min_{(l,r,c):\, l \le i \le r} \big(dp[l-1] + c\big).$$

Duyệt $i = 1..n$: với mỗi gói (i, r, c) thực hiện $[i, r]$ range-chmin bằng $(dp[i-1]+c)$; $dp[i] = \text{point\_query}(i)$.

Segment tree hỗ trợ **range-chmin** (gán $val := \min(val, x)$) và **point query**. Độ phức tạp $O((n+q)\log n)$.

```
const long long INF = (1LL<<62);
struct SegTree {
    int n;
    vector<long long> val, lz; // val: min at node, lz: pending range-chmin
    SegTree(int n=0): n(n) {
        val.assign(4*n+4, INF);
        lz.assign(4*n+4, INF);
    }
    void apply(int p, long long x) {
        val[p] = min(val[p], x);
        lz[p] = min(lz[p], x);
    }
    void push(int p) {
        if (lz[p] != INF) {
            apply(p<<1, lz[p]);
            apply(p<<1|1, lz[p]);
            lz[p] = INF;
        }
    }
    void range_chmin(int p, int L, int R, int i, int j, long long x) {
        if (j < L || R < i) return;
        if (i <= L && R <= j) { apply(p, x); return; }
        push(p);
        int M = (L+R)>>1;
        range_chmin(p<<1, L, M, i, j, x);
        range_chmin(p<<1|1, M+1, R, i, j, x);
        val[p] = min(val[p<<1], val[p<<1|1]);
    }
    void range_chmin(int l, int r, long long x) { if (l<=r) range_chmin(1,1,n,l,r,x); }

    long long point_query(int p, int L, int R, int idx) {
        if (L == R) return val[p];
        push(p);
        int M = (L+R)>>1;
        if (idx <= M) return point_query(p<<1, L, M, idx);
        else return point_query(p<<1|1, M+1, R, idx);
    }
    long long point_query(int idx){ return point_query(1,1,n,idx); }
};

int main() {
    int n, q; if (!(cin >> n >> q)) return 0;

    vector<vector<pair<int,long long>>> start(n+2); // start[l] -> list of (r, c)
    for (int i = 0; i < q; ++i) {
        int l, r; long long c;
        cin >> l >> r >> c;
        if (l > r) swap(l, r);
        l = max(l, 1);
        r = min(r, n);
        if (l <= r) start[l].push_back({r, c});
    }

    vector<long long> dp(n+1, INF);
    dp[0] = 0;

    SegTree st(n); // indices 1..n, initial +INF

    for (int i = 1; i <= n; ++i) {
        // kích ạhot các gói ắbt đầu ạti i
        for (auto [r, c] : start[i]) {
            if (dp[i-1] < INF) {
                long long cost = dp[i-1] + c;
                st.range_chmin(i, r, cost);
            }
        }
        dp[i] = st.point_query(i);
    }

    if (dp[n] >= INF/2) cout << -1 << '\n';
    else cout << dp[n] << '\n';
    return 0;
}
```

4.3 Số thao tác đổi chỗ cặp kề tối thiểu để sắp xếp hoán vị

Đề bài. Cho hoán vị A độ dài n. Mỗi thao tác được phép hoán vị hai phần tử kề nhau A_i, A_{i+1}. Hãy tìm số thao tác ít nhất để sắp A tăng dần.

Nhận xét. Số bước tối thiểu chính là **số nghịch thế**:

$$\text{inv}(A) = \#\{(i,j) : i < j,\ A_i > A_j\}.$$

Mỗi hoán vị kề làm giảm đúng 1 nghịch thế.

Tính bằng Segment Tree. Duyệt $i = 1 \to n$, với mỗi $x = A_i$: - # phần tử trước i mà lớn hơn x = query$(x{+}1, n)$. - Cập nhật đã thấy x: update$(x, +1)$.

```
struct SegTree {
   int n;
   vector<int> st;
   SegTree(int n=0): n(n), st(4*n+5, 0) {}
   void add(int p, int v, int id, int l, int r){
      if(l==r){ st[id]+=v; return; }
      int m=(l+r)>>1;
      if(p<=m) add(p,v,id<<1,l,m);
      else add(p,v,id<<1|1,m+1,r);
      st[id]=st[id<<1]+st[id<<1|1];
   }
   void add(int p,int v){ add(p,v,1,1,n); }
   int sum(int L,int R,int id,int l,int r){
      if(R<l||r<L) return 0;
      if(L<=l&&r<=R) return st[id];
      int m=(l+r)>>1;
      return sum(L,R,id<<1,l,m)+sum(L,R,id<<1|1,m+1,r);
   }
   int sum(int L,int R){ if(L>R) return 0; return sum(L,R,1,1,n); }
};

int main(){
   ios::sync_with_stdio(false);
   cin.tie(nullptr);

   int n;
   if(!(cin>>n)) return 0;
   vector<int> a(n);
   for(int i=0;i<n;i++) cin>>a[i]; // permutation 1..n

   SegTree st(n);
   long long inv = 0;
   for(int i=0;i<n;i++){
      int x = a[i];
      inv += st.sum(x+1, n); // ốs ầphn ửt ướtrc đó > x
      st.add(x, 1); // đánh ấdu đã ấthy x
   }
   cout << inv << '\n';
   return 0;
}
```

5 Đồ thị (Graph)

5.1 Đếm số vùng không bị ảnh hưởng bởi bão (L1)

Tóm tắt. Ma trận $n \times n$ gồm '.' (biển) và 'X' (tâm bão). Mỗi bão tại (i, j) ảnh hưởng mọi ô (i', j') với $|i - i'| + |j - j'| \le r$. Hỏi có bao nhiêu *vùng liên thông 4-hướng* chỉ gồm các ô **không bị ảnh hưởng**.

Ý tưởng.

1. Tính khoảng cách Manhattan tới *tâm bão gần nhất* cho mọi ô bằng **BFS đa nguồn** (đưa tất cả ô 'X' vào hàng đợi với khoảng cách 0). Với lưới 4-neighbors, BFS cho đúng khoảng cách L_1.
2. Một ô bị ảnh hưởng nếu dist $\le r$ (hoặc nó là 'X'); còn lại là **an toàn**.
3. Đếm số **thành phần liên thông 4-hướng** trên các ô an toàn bằng duyệt BFS/DFS.

Mỗi ô được thăm hữu hạn lần $\Rightarrow O(n^2)$ thời gian, $O(n^2)$ bộ nhớ. Phù hợp với $n \le 5000$.

```
int main() {
   int n; int r;
   if (!(cin >> n >> r)) return 0;
   vector<string> a(n);
   for (int i = 0; i < n; ++i) cin >> a[i];

   const int INF = 1e9;
   vector<vector<int>> dist(n, vector<int>(n, INF));
   deque<pair<int,int>> dq;

   // Multi-source BFS from all storms 'X'
   for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
         if (a[i][j] == 'X') {
            dist[i][j] = 0;
            dq.emplace_back(i, j);
         }
      }
   }

   int di[4] = {1,-1,0,0};
   int dj[4] = {0,0,1,-1};

   while (!dq.empty()) {
      auto [x, y] = dq.front(); dq.pop_front();
      int nd = dist[x][y] + 1;
      if (nd > r) continue; // farther than r is irrelevant for marking
      for (int t = 0; t < 4; ++t) {
         int nx = x + di[t], ny = y + dj[t];
         if (nx < 0 || nx >= n || ny < 0 || ny >= n) continue;
         if (dist[nx][ny] <= nd) continue;
         dist[nx][ny] = nd;
         dq.emplace_back(nx, ny);
      }
   }

   // Build safe mask: safe if originally '.' and dist > r (or no storm -> INF)
   vector<vector<char>> safe(n, vector<char>(n, 0));
   for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j)
         if (a[i][j] == '.' && dist[i][j] > r) safe[i][j] = 1;

```

```
    // Count connected components (4-neighbors) on safe cells
    vector<vector<char>> vis(n, vector<char>(n, 0));
    auto bfs = [&](int si, int sj) {
        queue<pair<int,int>> q;
        vis[si][sj] = 1;
        q.emplace(si, sj);
        while (!q.empty()) {
            auto [x, y] = q.front(); q.pop();
            for (int t = 0; t < 4; ++t) {
                int nx = x + di[t], ny = y + dj[t];
                if (nx < 0 || nx >= n || ny < 0 || ny >= n) continue;
                if (!safe[nx][ny] || vis[nx][ny]) continue;
                vis[nx][ny] = 1;
                q.emplace(nx, ny);
            }
        }
    };

    long long components = 0;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (safe[i][j] && !vis[i][j]) {
                ++components;
                bfs(i, j);
            }

    cout << components << '\n';
    return 0;
}
```

5.2 Băng tan & hai điểm gặp nhau (Swan Lake)

Đề bài. Cho ma trận $n \times m$ với ‘X‘ là băng, ‘.‘ là nước, hai ô ‘L‘ là vị trí hai điểm. Mỗi ngày, *mọi* ô băng kề **cạnh** (4 hướng) với nước sẽ tan (trở thành nước). Hỏi sau bao nhiêu ngày thì có đường đi chỉ qua ô nước giữa hai ‘L‘.

Ý tưởng chuẩn (2 pha + nhị phân ngày).

1. **Pha 1 – Ngày tan của từng ô:** gán tất cả ô nước và ‘L‘ có *ngày tan* = 0. Chạy BFS đa nguồn: khi từ một ô có ngày d đẩy sang ô băng chưa gán, gán ngày = $d+1$. Kết quả: day$[i][j]$ là ngày sớm nhất ô đó trở thành nước.
2. **Pha 2 – Tìm ngày nhỏ nhất:** nhị phân trên D (từ 0 đến max day). Kiểm tra với BFS/DFS từ ‘L_1‘ chỉ đi qua các ô có day $\le D$. Nếu tới được ‘L_2‘ thì khả thi, thu hẹp khoảng; ngược lại tăng D.

Độ phức tạp: tính ngày $O(nm)$; mỗi lần kiểm tra $O(nm)$; nhị phân $\approx \lceil \log_2(nm) \rceil$ lần □ tổng $O(nm \log(nm))$, đủ cho $n, m \le 1500$.

```
static const int INF = 1e9;
int main() {
    int n, m; if (!(cin >> n >> m)) return 0;
    vector<string> g(n);
    for (int i = 0; i < n; ++i) cin >> g[i];

    vector<pair<int,int>> L;
    deque<pair<int,int>> q;
    vector<vector<int>> day(n, vector<int>(m, INF));

    // init: water and L have day 0
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j) {
            if (g[i][j] == 'L') L.push_back({i,j});
            if (g[i][j] != 'X') { // '.' or 'L'
                day[i][j] = 0;
                q.push_back({i,j});
            }
        }

    auto inside = [&](int x,int y){return 0<=x && x<n && 0<=y && y<m;};
    const int dx[4]={-1,1,0,0}, dy[4]={0,0,-1,1};

    // BFS to compute earliest melt day
    int maxDay = 0;
    while(!q.empty()){
        auto [x,y]=q.front(); q.pop_front();
        for(int dir=0; dir<4; ++dir){
            int nx=x+dx[dir], ny=y+dy[dir];
            if(!inside(nx,ny)) continue;
            if(day[nx][ny] != INF) continue;
            // if neighbor is ice, it melts one day after current water reaches border
            day[nx][ny] = day[x][y] + 1;
            maxDay = max(maxDay, day[nx][ny]);
            q.push_back({nx,ny});
        }
    }

    // binary search minimal D
    auto can = [&](int D)->bool{
        // BFS from L[0], only through cells with day <= D
        queue<pair<int,int>> qb;
        vector<vector<char>> vis(n, vector<char>(m, 0));
        qb.push(L[0]); vis[L[0].first][L[0].second]=1;
        while(!qb.empty()){
            auto [x,y]=qb.front(); qb.pop();
            if (x==L[1].first && y==L[1].second) return true;
            for(int dir=0; dir<4; ++dir){
                int nx=x+dx[dir], ny=y+dy[dir];
                if(!inside(nx,ny) || vis[nx][ny]) continue;
                if(day[nx][ny] <= D){
                    vis[nx][ny]=1;
                    qb.push({nx,ny});
                }
            }
        }
        return false;
    };

    int lo=0, hi=maxDay, ans=maxDay;
    while(lo<=hi){
        int mid=(lo+hi)/2;
        if(can(mid)){ ans=mid; hi=mid-1; }
        else lo=mid+1;
    }
    cout << ans << '\n';
    return 0;
}
```

5.3 Đếm số đường đi ngắn nhất (Dijkstra + đếm cách)

Tóm tắt. Đồ thị vô hướng có trọng số. Đếm số lượng *đường đi ngắn nhất* từ 1 đến n, modulo 10^9+7.

Ý tưởng. Chạy Dijkstra từ 1:

- $dist[u]$ là độ dài ngắn nhất tới u.
- $ways[u]$ là số đường đi đạt $dist[u]$ (mod 10^9+7).

Khi relax cạnh (u, v, w):

$$\begin{cases} \text{Nếu } dist[u]+w < dist[v]: & dist[v] \leftarrow dist[u]+w,\ \ ways[v] \leftarrow ways[u] \\ \text{Nếu } dist[u]+w = dist[v]: & ways[v] \leftarrow (ways[v]+ways[u]) \bmod M \end{cases}$$

Kết quả là $ways[n]$ nếu $dist[n] < \infty$, ngược lại in 0 (không có đường).

Độ phức tạp. $O(m \log n)$.

```cpp
const int MOD = 1'000'000'007;
int main() {
   int n, m; if (!(cin >> n >> m)) return 0;

   vector<vector<pair<int,long long>>> g(n+1);
   for (int i = 0; i < m; ++i) {
      int u, v; long long w;
      cin >> u >> v >> w;
      g[u].push_back({v, w});
      g[v].push_back({u, w});
   }

   vector<long long> dist(n+1, INF);
   vector<int> ways(n+1, 0);
   priority_queue<pair<long long,int>, vector<pair<long long,int>>, greater<pair<long
        long,int>>> pq;

   dist[1] = 0;
   ways[1] = 1;
   pq.push({0, 1});

   while (!pq.empty()) {
      auto [du, u] = pq.top(); pq.pop();
      if (du != dist[u]) continue;

      for (auto [v, w] : g[u]) {
         long long nd = du + w;
         if (nd < dist[v]) {
            dist[v] = nd;
            ways[v] = ways[u];
            pq.push({dist[v], v});
         } else if (nd == dist[v]) {
            ways[v] += ways[u];
            if (ways[v] >= MOD) ways[v] -= MOD;
         }
      }
   }

   if (dist[n] >= INF/2) cout << 0 << '\n'; // không có đường đi
   else cout << ways[n] << '\n';
   return 0;
}
```

5.4 Thang máy: đếm số tầng có thể tới (Dijkstra trên lớp dư)

Đề bài. Từ tầng 1, mỗi lần bấm có thể:

- quay về tầng 1;
- lên A tầng; lên B tầng; lên C tầng.

Với n tầng, đếm số tầng $\le n$ có thể đi tới từ tầng 1.

Quy đổi. Một tầng f tới được $\iff$ $f = 1 + t$ với t biểu diễn được dạng

$$t = aA + bB + cC \quad (a, b, c \in \mathbb{Z}_{\ge 0}), \quad t \le N := n - 1.$$

Bài toán trở thành: đếm số $t \in [0, N]$ là tổng không âm của $\{A, B, C\}$.

Thuật giải (Dijkstra trên lớp dư mod m). Đặt $m = \min(A, B, C)$. Xét đồ thị có m đỉnh là các lớp dư $0, 1, \ldots, m-1$; với mỗi $x \in \{A, B, C\}$ có cạnh

$$r \to (r + x) \bmod m \text{ với trọng số } x.$$

Chạy Dijkstra từ 0 để thu được dist$[r]$ là *giá trị nhỏ nhất* có tổng $\equiv r \pmod m$ biểu diễn được bởi $\{A, B, C\}$. Khi đó t biểu diễn được $\iff$ $t \ge \text{dist}[\, t \bmod m\,]$.

Đếm. Với mỗi $r \in [0, m-1]$, nếu dist$[r] \le N$ thì số $t \le N$ có $t \equiv r \pmod m$ là

$$\left\lfloor \frac{N - \text{dist}[r]}{m} \right\rfloor + 1.$$

Tổng đáp án:

$$\boxed{\text{Ans} \;=\; \sum_{r=0}^{m-1} \max\left(0,\ \left\lfloor \frac{N - \text{dist}[r]}{m} \right\rfloor + 1\right)}.$$

Độ phức tạp. Dijkstra trên $m \le 10^5$ đỉnh với $3m$ cạnh: $O(m \log m)$. Đếm: $O(m)$. Phù hợp với $n \le 10^{18}$.

```cpp
using ll = long long;
const ll INF = LLONG_MAX / 4;
int main() {
   ll n; ll A, B, C;
   cin >> n >> A >> B >> C;
   ll N = n - 1;
   ll m = min({A, B, C});
   vector<ll> dist(m, INF);
   dist[0] = 0;

   priority_queue<pair<ll,int>, vector<pair<ll,int>>, greater<pair<ll,int>>> pq;
   pq.push({0, 0});

   auto relax = [&](int r, ll x) {
      ll newt = dist[r] + x;
      int nr = newt % m;
      if (newt < dist[nr]) {
         dist[nr] = newt;
         pq.push({newt, nr});
      }
   };

   while (!pq.empty()) {
      auto [cd, r] = pq.top();
```

```
        pq.pop();
        if (cd > dist[r]) continue;
        relax(r, A);
        relax(r, B);
        relax(r, C);
    }
    ll ans = 0;
    for (int r = 0; r < m; r++) if (dist[r] <= N) ans += (N - dist[r]) / m + 1;
    cout << ans << "\n";
    return 0;
}
```

5.5 Đỉnh nhận được từ mọi đỉnh (reachable-from-all)

Đề bài. Cho đồ thị có hướng n đỉnh, m cạnh. Đếm số đỉnh v sao cho *từ mọi đỉnh u đều tồn tại đường đi $u \to v$*.

Nhận xét. Trong đồ thị ngưng tụ (SCC DAG), nếu tồn tại một thành phần liên thông mạnh T mà *mọi SCC khác đều đi tới T*, thì *mọi đỉnh trong T* đều thỏa mãn. Nếu không tồn tại, đáp án là 0. Thành phần T (nếu có) là *duy nhất*.

Thuật toán $O(n+m)$:

1. Xét đồ thị đảo cạnh G^R. Tìm *ứng viên mẹ* c trong G^R: đỉnh có thời điểm kết thúc DFS cuối cùng (giống bài *mother vertex*).
2. Kiểm chứng: duyệt từ c trên G^R. Nếu không tới được *tất cả* n đỉnh $\Rightarrow$ không có đỉnh nào thỏa $\Rightarrow$ in 0.
3. Kích thước SCC chứa c chính là số đỉnh cần đếm. Ta lấy giao:

$$\text{scc}(c) = \{\text{đỉnh tới được từ } c \text{ trên } G\} \cap \{\text{đỉnh tới được từ } c \text{ trên } G^R\}.$$

Kết quả. Nếu bước 2 qua, in $|\text{scc}(c)|$, ngược lại in 0.

```
int main() {
    int n, m; if (!(cin >> n >> m)) return 0;
    vector<vector<int>> g(n), gr(n);
    for (int i = 0; i < m; ++i) {
        int u, v; cin >> u >> v;
        --u; --v;
        g[u].push_back(v);
        gr[v].push_back(u); // reverse edge
    }
    vector<char> vis(n, 0);
    vector<int> order; order.reserve(n);
    for (int s = 0; s < n; ++s) if (!vis[s]) {
        vector<pair<int,int>> st; st.emplace_back(s, 0);
        vis[s] = 1;
        while (!st.empty()) {
            int u = st.back().first;
            int &it = st.back().second;
            if (it < (int)gr[u].size()) {
                int v = gr[u][it++];
                if (!vis[v]) { vis[v] = 1; st.emplace_back(v, 0); }
            } else {
                order.push_back(u);
                st.pop_back();
            }
        }
    }
    int c = order.back();
    auto bfs = [&](const vector<vector<int>>& GG, int start) {
        vector<char> seen(n, 0);
        queue<int> q; q.push(start); seen[start] = 1;
        int cnt = 1;
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (int v : GG[u]) if (!seen[v]) {
                seen[v] = 1; q.push(v); ++cnt;
            }
        }
        return pair<vector<char>,int>(move(seen), cnt);
    };

    auto [reach_in_GR, cnt_all] = bfs(gr, c);
    if (cnt_all != n) {
        cout << 0 << '\n';
        return 0;
    }
    auto [reach_from_c_in_G, _1] = bfs(g, c);
    long long scc_size = 0;
    for (int i = 0; i < n; ++i)
        if (reach_from_c_in_G[i] && reach_in_GR[i]) ++scc_size;

    cout << scc_size << '\n';
    return 0;
}
```

5.6 LCA (Lowest Common Ancestor) bằng Binary Lifting

Đề bài. Cây n đỉnh gốc tại 1. Trả lời q truy vấn (u, v), tìm tổ tiên chung gần nhất (LCA) của u và v.

Ý tưởng. Tiền xử lý $\text{up}[k][u]$ = tổ tiên thứ 2^k của u và $\text{depth}[u]$.

- Nâng đỉnh sâu hơn lên cùng độ sâu với đỉnh kia bằng các bước 2^k.
- Nếu hai đỉnh khác nhau, thử từ k lớn xuống 0: khi $\text{up}[k]$ khác nhau thì cùng nhảy lên.
- LCA là $\text{up}[0]$ của một trong hai đỉnh sau bước trên.

Tiền xử lý $O(n \log n)$, mỗi truy vấn $O(\log n)$.

```
int main() {
    int n, q; if (!(cin >> n >> q)) return 0;
    vector<vector<int>> g(n+1);
    for (int i = 0; i < n-1; ++i) {
        int u, v; cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }

    int LOG = 1;
    while ((1 << LOG) <= n) ++LOG;

    vector<int> depth(n+1, -1);
    vector<vector<int>> up(LOG, vector<int>(n+1, 1)); // up[k][1] = 1

    // BFS node 1 to get depth and up[0]
    queue<int> qu;
    depth[1] = 0;
    up[0][1] = 1;
```

```
    qu.push(1);
    while (!qu.empty()) {
        int u = qu.front(); qu.pop();
        for (int v : g[u]) if (depth[v] == -1) {
            depth[v] = depth[u] + 1;
            up[0][v] = u;
            qu.push(v);
        }
    }
    for (int k = 1; k < LOG; ++k) {
        for (int v = 1; v <= n; ++v) {
            up[k][v] = up[k-1][ up[k-1][v] ];
        }
    }

    auto lift = [&](int u, int d) {
        for (int k = 0; k < LOG; ++k) {
            if (d & (1 << k)) u = up[k][u];
        }
        return u;
    };

    auto lca = [&](int u, int v) {
        if (depth[u] < depth[v]) swap(u, v);
        u = lift(u, depth[u] - depth[v]);
        if (u == v) return u;
        for (int k = LOG-1; k >= 0; --k) {
            if (up[k][u] != up[k][v]) {
                u = up[k][u];
                v = up[k][v];
            }
        }
        return up[0][u];
    };
    for (int i = 0; i < q; ++i) {
        int u, v; cin >> u >> v;
        int ans = lca(u, v);
        cout << ans << (i+1==q ? '\n' : ' ');
    }
    return 0;
}
```

5.7 Đếm cặp có *đường đi duy nhất* trong đồ thị vô hướng

Ý tưởng. Trong đồ thị vô hướng, giữa u, v có *duy nhất một* đường đi đơn $\iff$ đường đi đó chỉ dùng **các cạnh là cầu (bridge)**. Do đó, nếu chỉ giữ các *cầu*, ta được một **rừng** G'; mọi cặp đỉnh trong cùng một cây của G' có đúng một đường đi (và ngược lại).

$$\Rightarrow \text{ đáp án} = \sum_{\text{thành phần } C \text{ của } G'} \binom{|C|}{2}.$$

Thuật toán.

1. Dùng Tarjan để tìm tất cả **cầu** trong $O(n+m)$.
2. Duyệt chỉ theo các cạnh là cầu để lấy kích thước từng thành phần liên thông $|C|$.
3. Cộng $\frac{|C|(|C|-1)}{2}$ vào kết quả (dùng `long long`).

```
int main() {
    int n, m;
    if (!(cin >> n >> m)) return 0;

    vector<pair<int,int>> edges(m);
    vector<vector<pair<int,int>>> g(n+1); // (to, edge_id)
    for (int i = 0; i < m; ++i) {
        int u, v; cin >> u >> v;
        edges[i] = {u, v};
        g[u].push_back({v, i});
        g[v].push_back({u, i});
    }

    // Tarjan to find bridges
    vector<int> tin(n+1, -1), low(n+1, -1);
    vector<char> is_bridge(m, 0);
    int timer = 0;

    function<void(int,int)> dfs = [&](int u, int pe) {
        tin[u] = low[u] = timer++;
        for (auto [v, id] : g[u]) {
            if (id == pe) continue;
            if (tin[v] != -1) {
                // back-edge
                low[u] = min(low[u], tin[v]);
            } else {
                dfs(v, id);
                low[u] = min(low[u], low[v]);
                if (low[v] > tin[u]) is_bridge[id] = 1;
            }
        }
    };

    // Graph is connected per statement, but keep general
    for (int i = 1; i <= n; ++i)
        if (tin[i] == -1) dfs(i, -1);

    // Count sizes in the forest formed by only bridge edges
    vector<char> vis(n+1, 0);
    function<int(int)> dfs2 = [&](int u) {
        vis[u] = 1;
        int sz = 1;
        for (auto [v, id] : g[u]) {
            if (!is_bridge[id]) continue; // only traverse bridges
            if (!vis[v]) sz += dfs2(v);
        }
        return sz;
    };

    ll ans = 0;
    for (int i = 1; i <= n; ++i) if (!vis[i]) {
        int sz = dfs2(i); // component size in bridge-only forest
        ans += 1LL * sz * (sz - 1) / 2; // choose any pair inside
    }

    cout << ans << '\n';
    return 0;
}
```

5.8 Đếm số đỉnh x sao cho $f(u,x) = f(v,x)$ trên cây

Đề bài. Với mỗi truy vấn (u, v) trên cây n đỉnh, đếm số đỉnh x có khoảng cách từ x tới u và v bằng nhau.

Ý tưởng then chốt. Gọi $d = \text{dist}(u, v)$.

- Nếu $u = v \Rightarrow$ mọi đỉnh đều thỏa: đáp án $= n$.
- Nếu d lẻ $\Rightarrow$ *không có* đỉnh nguyên giữa đường đi $u \leftrightarrow v \Rightarrow$ đáp án $= 0$.
- Nếu d chẵn: điểm giữa (midpoint) là một *đỉnh* m trên đường đi $u \leftrightarrow v$.
  - Nếu m **trùng LCA** của (u, v): gọi a là đỉnh con của m nằm trên đường $m \to u$, b là đỉnh con của m nằm trên đường $m \to v$. Khi đó đáp án:

$$\boxed{n - \text{sz}[a] - \text{sz}[b]}$$

    (loại bỏ hai "nhánh" đi xa khỏi m về phía u và v).
  - Nếu m **nằm giữa** u và v nhưng *khác* LCA: giả sử u nằm sâu hơn phía m ($\text{depth}[u] > \text{depth}[v]$). Gọi c là đỉnh *con* của m trên đường $m \to u$ (tức là từ u đi lên $d/2 - 1$ bước). Khi đó đáp án:

$$\boxed{\text{sz}[m] - \text{sz}[c]}$$

    (chỉ tính phần "phía m không đi vào nhánh c").

Tiền xử lý. Chọn gốc 1, tính depth, up$[k][v]$ (binary lifting), sz$[v]$ bằng DFS/BFS. Trả lời mỗi truy vấn trong $O(\log n)$.

```cpp
const int MAXN = 100000;
int n, q;
vector<int> g[MAXN+1];

int LOG;
int up[20][MAXN+1];
int depth_[MAXN+1];
int sz[MAXN+1];

void dfs(int u, int p){
    up[0][u] = (p ? p : u);
    depth_[u] = p ? depth_[p] + 1 : 0;
    sz[u] = 1;
    for (int v : g[u]) if (v != p){
        dfs(v, u);
        sz[u] += sz[v];
    }
}

int lift(int u, int k){
    for (int i = 0; i < LOG; ++i)
        if (k & (1<<i)) u = up[i][u];
    return u;
}

int lca(int u, int v){
    if (depth_[u] < depth_[v]) swap(u, v);
    u = lift(u, depth_[u] - depth_[v]);
    if (u == v) return u;
    for (int i = LOG-1; i >= 0; --i){
        if (up[i][u] != up[i][v]){
            u = up[i][u];
            v = up[i][v];
        }
    }
    return up[0][u];
}

int dist(int u, int v){
    int w = lca(u, v);
    return depth_[u] + depth_[v] - 2*depth_[w];
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    if(!(cin >> n >> q)) return 0;
    for(int i=1;i<=n;i++) g[i].clear();
    for(int i=0;i<n-1;i++){
        int u,v; cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }

    LOG = 1;
    while ((1<<LOG) <= n) ++LOG;

    dfs(1, 0);
    for (int k = 1; k < LOG; ++k){
        for (int v = 1; v <= n; ++v){
            up[k][v] = up[k-1][ up[k-1][v] ];
        }
    }

    while(q--){
        int u, v; cin >> u >> v;
        if (u == v){
            cout << n << '\n';
            continue;
        }
        int d = dist(u, v);
        if (d & 1){
            cout << 0 << '\n';
            continue;
        }

        int w = lca(u, v);
        int du = depth_[u] - depth_[w];
        int dv = depth_[v] - depth_[w];
        int half = d / 2;

        if (du == dv){
            // midpoint is w
            // a: child of w towards u; b: child of w towards v
            int a = lift(u, du - 1);
            int b = lift(v, dv - 1);
            long long ans = (long long)n - sz[a] - sz[b];
            cout << ans << '\n';
        } else {
```

```
        // midpoint m lies on the deeper side
        if (du < dv) {
            // make u the deeper one
            swap(u, v);
            swap(du, dv);
        }
        // u deeper: go up half steps to reach m
        int m = lift(u, half);
        // child c from m towards u: go up (half - 1) from u
        int c = lift(u, half - 1);
        long long ans = (long long)sz[m] - sz[c];
        cout << ans << '\n';
      }
   }
   return 0;
}
```

5.9 Cộng vào cây con và truy vấn giá trị tại đỉnh

Đề bài. Cho cây n đỉnh, gốc là 1. Ban đầu mỗi đỉnh có giá trị 0. Có q truy vấn, gồm hai loại:

- `1 u v`: tăng tất cả các đỉnh trong **cây con gốc** u thêm v.
- `2 u`: in ra giá trị hiện tại tại đỉnh u.

Yêu cầu: Trả lời các truy vấn loại 2. $1 \le n, q \le 10^5$.

Ý tưởng lời giải.

1. Dùng **Euler Tour** để đánh số thời gian vào cây:

$$\text{tin}[u], \text{tout}[u] \Rightarrow \text{toàn bộ cây con của } u \text{ là đoạn } [\text{tin}[u], \text{tout}[u]].$$

2. Gom toàn bộ đỉnh cây vào mảng theo thứ tự Euler.
3. Khi gặp truy vấn:

$$\texttt{1 u v} \Rightarrow \text{cộng } v \text{ vào đoạn } [\text{tin}[u], \text{tout}[u]].$$

$$\texttt{2 u} \Rightarrow \text{giá trị tại } u = \text{giá trị tại vị trí } \text{tin}[u].$$

4. Sử dụng **Segment Tree + Lazy Propagation** (hoặc Fenwick Tree) để hỗ trợ:

$$\text{range add} + \text{point query}, \text{ độ phức tạp } O(q \log n).$$

Cấu trúc dữ liệu dùng Segtree:

- Mỗi node lưu `lazy` (giá trị cần cộng cho cả đoạn).
- range_add(l,r,val): cộng giá trị cho đoạn.
- point_query(pos): cộng dồn lazy từ gốc xuống để lấy giá trị tại vị trí pos.

```
struct SegTree {
   int n;
   vector<ll> lazy;
   SegTree(int n=0): n(n), lazy(4*n+4, 0) {}
   void range_add(int p, int L, int R, int i, int j, ll v) {
      if (j < L || R < i) return;
      if (i <= L && R <= j) { lazy[p] += v; return; }
      int M = (L + R) >> 1;
      range_add(p<<1, L, M, i, j, v);
      range_add(p<<1|1, M+1, R, i, j, v);
   }
   void range_add(int l, int r, ll v){ if(l<=r) range_add(1,1,n,l,r,v); }
   ll point_query(int p, int L, int R, int idx) {
      if (L == R) return lazy[p];
      int M = (L + R) >> 1;
      if (idx <= M) return lazy[p] + point_query(p<<1, L, M, idx);
      else return lazy[p] + point_query(p<<1|1, M+1, R, idx);
   }
   ll point_query(int idx){ return point_query(1,1,n,idx); }
};

int main(){
   int n, q; if(!(cin >> n >> q)) return 0;
   vector<vector<int>> g(n+1);
   for(int i=0;i<n-1;i++){
      int u,v; cin >> u >> v;
      g[u].push_back(v);
      g[v].push_back(u);
   }
   // Euler tour: tin[u]..tout[u] là đạon subtree(u)
   vector<int> tin(n+1,0), tout(n+1,0), parent(n+1,0), it(n+1,0), order; order.reserve(n);
   vector<int> st; st.reserve(2*n);
   int timer = 0;
   st.push_back(1); parent[1]=0;
   while(!st.empty()){
      int u = st.back();
      if(tin[u]==0) tin[u] = ++timer;
      if(it[u] < (int)g[u].size()){
         int v = g[u][it[u]++];
         if(v==parent[u]) continue;
         parent[v]=u;
         st.push_back(v);
      }else{
         tout[u] = timer;
         st.pop_back();
      }
   }

   SegTree stree(n);

   while(q--){
      int type; cin >> type;
      if(type==1){
         int u; ll v; cin >> u >> v;
         stree.range_add(tin[u], tout[u], v);
      }else{
         int u; cin >> u;
         cout << stree.point_query(tin[u]) << '\n';
      }
   }
   return 0;
}
```

6 Bitmask

6.1 Kiến thức cần nhớ

- **Biểu diễn tập hợp bằng bitmask:** Một số nguyên không âm dùng để biểu diễn tập các phần tử từ $0 \to n-1$. Bit thứ i bằng 1 nếu phần tử i thuộc tập.

$$\text{VD: } S = \{0, 2, 3\} \Rightarrow \text{mask} = (1101)_2 = 13$$

- **Các phép toán cơ bản:**
  - Thêm phần tử i: `mask` = (mask | (1 « i))
  - Xóa phần tử i: `mask &= ~(1 << i)`
  - Kiểm tra phần tử i: `(mask >> i) & 1`
  - Đảo bit i: `mask ^= (1 << i)`
- **Duyệt các tập con của một mask:**

```
for (int sub = mask; sub > 0; sub = (sub - 1) & mask) {
    // sub là một tập con của mask
}
```

- **Đếm số bit bằng 1 (popcount):**
  - `__builtin_popcount(x)` (int)
  - `__builtin_popcountll(x)` (long long)
- **Duyệt tất cả mask từ 0 đến $2^n - 1$:**

```
for (int mask = 0; mask < (1 << n); mask++) {
    // xử lý
}
```

- **Lưu ý về độ phức tạp:**

$$2^{20} \approx 10^6 \Rightarrow \text{có thể duyệt được} \quad \text{nhưng} \quad 2^{25} \approx 3 \times 10^7 \Rightarrow \text{chậm}$$

6.2 Đếm cặp $(i < j)$ thoả $A_i \oplus j = A_j \oplus i$

Ý tưởng. Từ $A_i \oplus j = A_j \oplus i \iff (A_i \oplus i) = (A_j \oplus j)$. Đặt $B_i = A_i \oplus i$ (chỉ số i tính **1-based**). Khi đó số cặp cần đếm là

$$\sum_x \binom{\text{freq}[x]}{2},$$

với $\text{freq}[x]$ là số lần x xuất hiện trong dãy B.

Độ phức tạp. $O(n)$ với `unordered_map`.

```
int main() {
    int n; if (!(cin >> n)) return 0;
    vector<long long> A(n+1);
    for (int i = 1; i <= n; ++i) cin >> A[i];
    unordered_map<long long, long long> freq;
    freq.reserve(n * 2);
    freq.max_load_factor(0.7);

    for (int i = 1; i <= n; ++i) {
        long long key = A[i] ^ i;
        ++freq[key];
    }
    long long ans = 0;
    for (auto &kv : freq) {
        long long f = kv.second;
        ans += f * (f - 1) / 2;
    }
    cout << ans << '\n';
    return 0;
}
```

6.3 Bật/Tắt/Đảo bit trên số 32-bit

Đề bài. Ban đầu có một số nguyên không âm 32-bit $x = 0$ (mọi bit đều bằng 0). Có q truy vấn:

- `1 k`: bật (set) bit thứ $k \Rightarrow$ gán bit k bằng 1.
- `2 k`: tắt (clear) bit thứ $k \Rightarrow$ gán bit k bằng 0.
- `3 k`: đảo (toggle) bit thứ $k \Rightarrow 0 \leftrightarrow 1$.

Sau mỗi truy vấn, in ra giá trị thập phân hiện tại của x.

Ràng buộc.

$$1 \le q \le 10^5, \quad 0 \le k \le 31.$$

Lời giải. Dùng toán tử bit trên `uint32_t`:

$$\text{mask} = (1 \ll k),$$
$$\text{set: } x := x \mid \text{mask}, \qquad \text{clear: } x := x \,\&\, \sim \text{mask}, \qquad \text{toggle: } x := x \oplus \text{mask}.$$

Mỗi truy vấn $O(1)$, tổng $O(q)$.

```
int main() {
    int q; if (!(cin >> q)) return 0;
    uint32_t x = 0; // 32-bit unsigned, initially 0
    while (q--) {
        int t; unsigned k;
        cin >> t >> k; // 0 <= k <= 31
        uint32_t mask = (1u << k);
        if (t == 1) x |= mask; // set k-th bit
        else if (t == 2) x &= ~mask; // clear k-th bit
        else x ^= mask; // toggle k-th bit
        cout << x << '\n';
    }
    return 0;
}
```

6.4 XOR nhỏ nhất của hai số khác nhau (cập nhật online)

Bài toán. Ban đầu dãy rỗng. Truy vấn:

- `1 x`: chèn x;
- `2 x`: xoá một x đang có;
- `3`: in $\min_{i \neq j}(a_i \oplus a_j)$ (đảm bảo lúc này có ≥ 2 số).

Ý tưởng. Tối thiểu XOR trong một đa tập bằng:

$$\min\Big(0 \text{ nếu tồn tại phần tử có tần suất} \ge 2, \min_{\text{cặp kề nhau trong dãy sắp xếp}}(b_i \oplus b_{i+1})\Big).$$

Vì với các số *phân biệt*, giá trị nhỏ nhất luôn đạt ở hai số **kề nhau** theo thứ tự tăng dần. Do đó, duy trì:

- `cnt[x]`: tần suất $\Rightarrow$ nếu có phần tử đạt ≥ 2 thì đáp án là 0;
- `set` các giá trị **phân biệt** (sắp xếp tăng);
- `multiset` các XOR của **cặp kề** trong `set`.

Chèn/xoá một giá trị phân biệt x chỉ ảnh hưởng tới hai láng giềng `pred` và `succ` trong `set`:

- **Insert mới x:** nếu cả `pred`, `succ` tồn tại, xoá $(pred \oplus succ)$; thêm $(pred \oplus x)$ và $(x \oplus succ)$.
- **Erase cuối x:** xoá $(pred \oplus x)$, $(x \oplus succ)$; nếu cả hai tồn tại, thêm $(pred \oplus succ)$.

Truy vấn 3: nếu có phần tử lặp $\Rightarrow 0$, ngược lại là min trong `multiset` XOR kề.

Độ phức tạp. Mỗi thao tác $O(\log n)$.

```
int main() {
   int q; if (!(cin >> q)) return 0;

   unordered_map<int,int> cnt;
   cnt.reserve(q*2); cnt.max_load_factor(0.7);

   set<int> st;
   multiset<int> adjXor;
   long long dupKinds = 0;

   auto get_pred = [&](int x)->optional<int>{
      auto it = st.lower_bound(x);
      if (it == st.begin()) return {};
      --it; return *it;
   };
   auto get_succ = [&](int x)->optional<int>{
      auto it = st.upper_bound(x);
      if (it == st.end()) return {};
      return *it;
   };
   auto erase_one_adj = [&](int val){
      auto it = adjXor.find(val);
      if (it != adjXor.end()) adjXor.erase(it);
   };

   auto insert_value = [&](int x){
      int c = cnt[x]++;
      if (c == 1) {
         ++dupKinds;
         return;
      }
      if (c >= 2) return;
      auto p = get_pred(x);
      auto s = get_succ(x);
      if (p && s) erase_one_adj((*p) ^ (*s));
      if (p) adjXor.insert((*p) ^ x);
      if (s) adjXor.insert(x ^ (*s));
      st.insert(x);
   };

   auto erase_value = [&](int x){
      int c = cnt[x];
      if (c == 2) {
         --dupKinds;
         cnt[x] = 1;
         return;
      }
      if (c > 2) {
         cnt[x] = c - 1;
         return;
      }
      cnt.erase(x);
      auto p = get_pred(x);
      auto s = get_succ(x);
      if (p) erase_one_adj((*p) ^ x);
      if (s) erase_one_adj(x ^ (*s));
      if (p && s) adjXor.insert((*p) ^ (*s));
      st.erase(x);
   };

   while (q--) {
      int t; cin >> t;
      if (t == 1) {
         int x; cin >> x;
         insert_value(x);
      } else if (t == 2) {
         int x; cin >> x;
         erase_value(x);
      } else {
         if (dupKinds > 0) {
            cout << 0 << '\n';
         } else {
            cout << *adjXor.begin() << '\n';
         }
      }
   }
   return 0;
}
```

7 Hash & String

7.1 Đếm số lần xuất hiện của T trong S (Z-algorithm, $O(|S|+|T|)$)

Ý tưởng. Tạo xâu $X = T + '\#' + S$ (ký tự ngăn cách không xuất hiện trong S, T), tính mảng Z của X. Mỗi vị trí i trong phần S có một khớp T nếu $Z[i] \geq |T|$. Tổng số vị trí như vậy là đáp án.

```
int main() {
   string S, T;
   if (!getline(cin, S)) return 0;
   if (!getline(cin, T)) return 0;

   const char SEP = '#';
   // Ensure SEP not in S or T; if present, choose another (rare). For safety:
   if (S.find(SEP) != string::npos || T.find(SEP) != string::npos) {
      // find a separator not present
      for (char c = 1; c < 127; ++c) {
         if (S.find(c) == string::npos && T.find(c) == string::npos) {
            // rebuild after choosing new sep
            // We'll just set SEP via a small hack: rebuild below using string ops
            string X = T; X.push_back(c); X += S;
            int n = (int)X.size();
            vector<int> Z(n, 0);
```

```
            int l = 0, r = 0;
            for (int i = 1; i < n; ++i) {
                if (i <= r) Z[i] = min(r - i + 1, Z[i - l]);
                while (i + Z[i] < n && X[Z[i]] == X[i + Z[i]]) ++Z[i];
                if (i + Z[i] - 1 > r) l = i, r = i + Z[i] - 1;
            }
            long long ans = 0;
            int m = (int)T.size();
            for (int i = m + 1; i < n; ++i) if (Z[i] >= m) ++ans;
            cout << ans << '\n';
            return 0;
        }
    }
}

// Normal path with '#'
string X;
X.reserve(T.size() + 1 + S.size());
X += T; X.push_back(SEP); X += S;

int n = (int)X.size();
vector<int> Z(n, 0);
int l = 0, r = 0;
for (int i = 1; i < n; ++i) {
    if (i <= r) Z[i] = min(r - i + 1, Z[i - l]);
    while (i + Z[i] < n && X[Z[i]] == X[i + Z[i]]) ++Z[i];
    if (i + Z[i] - 1 > r) l = i, r = i + Z[i] - 1;
}

long long ans = 0;
int m = (int)T.size();
for (int i = m + 1; i < n; ++i) if (Z[i] >= m) ++ans;
cout << ans << '\n';
return 0;
}
```

7.2 Chu kỳ ngắn nhất của xâu (minimal period)

Bài toán. Tìm xâu T ngắn nhất sao cho lặp T đủ n ký tự thì được S.

Ý tưởng (KMP prefix function). Tính $\pi[i]$ = độ dài tiền tố dài nhất cũng là hậu tố của $S[0..i]$. Gọi $p = n - \pi[n-1]$. Khi đó:

$$\text{Nếu } n \bmod p = 0 \Rightarrow T = S[0..p-1], \quad \text{ngược lại } T = S.$$

```
int main() {
    string S; if (!(cin >> S)) return 0;
    int n = (int)S.size();
    vector<int> pi(n, 0);
    for (int i = 1; i < n; ++i) {
        int j = pi[i-1];
        while (j > 0 && S[i] != S[j]) j = pi[j-1];
        if (S[i] == S[j]) ++j;
        pi[i] = j;
    }
    int p = n - pi[n-1];
    if (n % p == 0) cout << S.substr(0, p) << '\n';
    else cout << S << '\n';
    return 0;
}
```

7.3 So sánh tập của các tiền tố hai mảng A và B

Đề bài. Cho hai mảng A, B độ dài n. Với mỗi truy vấn (i, j), xét hai *tập*:

$$S_A(i) = \{A_1, \ldots, A_i\}, \qquad S_B(j) = \{B_1, \ldots, B_j\}.$$

Hỏi $S_A(i)$ và $S_B(j)$ có bằng nhau không?

Ý tưởng. Tiền xử lý hai dãy:

$$\text{reachA}[i] = \min\{\, j \mid S_A(i) \subseteq S_B(j) \,\}, \quad \text{reachB}[j] = \min\{\, i \mid S_B(j) \subseteq S_A(i) \,\}.$$

Tính bằng hai con trỏ với *tập* đã thấy (chỉ quan tâm sự xuất hiện, không đếm số lần):

- Duyệt $i = 1..n$, mỗi khi gặp phần tử mới trong A mà chưa có trong $B[1..j]$ thì tăng biến `need`. Tăng j và "thêm" các phần tử trong B cho tới khi `need`=0. Khi đó `reachA[i]=j` (nếu không đạt, đặt ∞).
- Tương tự, tráo vai $A \leftrightarrow B$ để có `reachB`.

Với truy vấn (i, j), $S_A(i) = S_B(j)$ **iff** $\text{reachA}[i] \le j$ *và* $\text{reachB}[j] \le i$ (bao hàm 2 chiều).

Độ phức tạp tiền xử lý $O(n)$ trung bình (unordered_set), mỗi truy vấn $O(1)$.

```
vector<int> buildReachSubset(const vector<int>& A, const vector<int>& B) {
    int n = (int)A.size();
    vector<int> reach(n + 1, INF); // 1..n; reach[0] = 0 (empty set is subset of empty)
    reach[0] = 0;

    unordered_set<int> inA, inB;
    inA.reserve(n * 2); inA.max_load_factor(0.7f);
    inB.reserve(n * 2); inB.max_load_factor(0.7f);

    int j = 0; // current prefix length on B
    int need = 0; // number of distinct elements in A[1..i] not yet in B[1..j]

    for (int i = 1; i <= n; ++i) {
        if (!inA.count(A[i-1])) { // A[i] is new in set SA(i)
            inA.insert(A[i-1]);
            if (!inB.count(A[i-1])) ++need; // this element not covered by B yet
        }
        while (j < n && need > 0) {
            int b = B[j++];
            if (!inB.count(b)) {
                inB.insert(b);
                if (inA.count(b)) --need; // b fulfills one needed element
            }
        }
        if (need == 0) reach[i] = j; // minimal j that covers SA(i)
        // else keep INF (impossible with given prefixes)
    }
    return reach;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
```

```
    int n, q;
    if (!(cin >> n >> q)) return 0;
    vector<int> A(n), B(n);
    for (int i = 0; i < n; ++i) cin >> A[i];
    for (int i = 0; i < n; ++i) cin >> B[i];

    // Precompute minimal covering prefixes
    vector<int> reachA = buildReachSubset(A, B); // SA(i) ⊆ SB(reachA[i])
    vector<int> reachB = buildReachSubset(B, A); // SB(j) ⊆ SA(reachB[j])

    while (q--) {
        int i, j; cin >> i >> j;
        bool ok = (reachA[i] <= j) && (reachB[j] <= i);
        cout << (ok ? "YES" : "NO") << '\n';
    }
    return 0;
}
```

7.4 Xâu xoay từ điển nhỏ nhất (Lexicographically Minimal Rotation)

Đề bài. Một **xâu xoay (cyclic shift)** được tạo bằng cách chuyển ký tự cuối của xâu lên đầu (hoặc ngược lại) nhiều lần. Ví dụ với `S = "marisa"`:

$$\text{marisa},\ \text{amaris},\ \text{samari},\ \text{isamar},\ \text{risama},\ \text{arisam}$$

Yêu cầu: Tìm **xâu xoay nhỏ nhất theo thứ tự từ điển** của xâu S.

Ràng buộc.

$$1 \le |S| \le 10^6$$

Ý tưởng. - Nếu duyệt tất cả các xoay và so sánh $\rightarrow O(n^2)$, quá chậm. - Dùng **Thuật toán Booth** để tìm vị trí bắt đầu của xoay nhỏ nhất, độ phức tạp $O(n)$.

Thuật toán Booth (tóm tắt):

- Gộp xâu với chính nó: $T = S + S$
- Dùng hai con trỏ i, j để tìm vị trí bắt đầu tốt nhất.
- Nếu $T[i+k] = T[j+k]$ thì tăng k.
- Nếu $T[i+k] > T[j+k]$ thì bỏ toàn bộ từ i đến $i+k$, cập nhật $i = i+k+1$.
- Nếu $T[i+k] < T[j+k] \rightarrow$ tương tự cho j.
- Kết quả là $\min(i, j)$.

Độ phức tạp: $O(n)$ thời gian, $O(1)$ bộ nhớ.

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    string s; cin >> s;
    int n = s.size();
    string t = s + s;

    int i = 0, j = 1, k = 0;
    while (i < n && j < n && k < n) {
        if (t[i + k] == t[j + k]) {
            k++;
        } else if (t[i + k] > t[j + k]) {
            i = i + k + 1;
            if (i == j) i++;
            k = 0;
        } else {
            j = j + k + 1;
            if (j == i) j++;
            k = 0;
        }
    }
    int start = min(i, j);
    cout << t.substr(start, n);
    return 0;
}
```

7.5 Xâu dài nhất xuất hiện $\ge k$ lần

Ý tưởng (Suffix Array + LCP). Gọi SA là mảng suffix, $LCP[i] = \text{lcp}(\text{suffix } SA[i-1], SA[i])$ (Kasai). Một xâu xuất hiện $\ge k$ lần tương ứng với **một cửa sổ** k suffix liên tiếp trong SA; độ dài lớn nhất trong cửa sổ đó là

$$\min(LCP[i-k+2],\ LCP[i-k+3],\ \ldots,\ LCP[i]) \quad (i \text{ là rìa phải cửa sổ}).$$

Quét rìa phải $i = 1..n-1$, duy trì **deque** giá trị nhỏ nhất (RMQ trượt) trên LCP với kích thước hiệu dụng $k-1$. Đáp án là max của các minimum này. Trường hợp $k = 1$ trả ngay $|S|$.

Độ phức tạp. Xây SA bằng doubling $O(n \log n)$, Kasai $O(n)$, cửa sổ trượt $O(n)$.

```
static vector<int> build_sa(const string &s) {
    int n = (int)s.size();
    vector<int> sa(n), rnk(n), tmp(n);
    iota(sa.begin(), sa.end(), 0);
    for (int i = 0; i < n; ++i) rnk[i] = s[i];

    for (int k = 1;; k <<= 1) {
        auto cmp = [&](int i, int j) {
            if (rnk[i] != rnk[j]) return rnk[i] < rnk[j];
            int ri = (i + k < n) ? rnk[i + k] : -1;
            int rj = (j + k < n) ? rnk[j + k] : -1;
            return ri < rj;
        };
        sort(sa.begin(), sa.end(), cmp);
        tmp[sa[0]] = 0;
        for (int i = 1; i < n; ++i)
            tmp[sa[i]] = tmp[sa[i-1]] + (cmp(sa[i-1], sa[i]) ? 1 : 0);
        for (int i = 0; i < n; ++i) rnk[i] = tmp[i];
        if (rnk[sa.back()] == n - 1) break; // all ranks unique
    }
    return sa;
}

static vector<int> build_lcp(const string &s, const vector<int> &sa) {
    int n = (int)s.size();
    vector<int> rank(n, 0), lcp(n, 0);
    for (int i = 0; i < n; ++i) rank[sa[i]] = i;
    int h = 0;
    for (int i = 0; i < n; ++i) {
        int r = rank[i];
        if (r == 0) { lcp[r] = 0; continue; }
        int j = sa[r - 1];
```

```
        while (i + h < n && j + h < n && s[i + h] == s[j + h]) ++h;
        lcp[r] = h;
        if (h) --h;
    }
    return lcp; // lcp[0] = 0, lcp[i] = LCP(SA[i-1], SA[i])
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    string S;
    if (!(cin >> S)) return 0;
    int k;
    if (!(cin >> k)) return 0;

    int n = (int)S.size();
    if (k <= 1) { cout << n << '\n'; return 0; }

    auto sa = build_sa(S);
    auto lcp = build_lcp(S, sa);

    // Sliding window over LCP to get min over each block of size (k-1)
    // Window r runs over LCP indices 1..n-1; for a k-suffix window ending at SA[r],
    // we need min(LCP[r-k+2..r]) length = k-1.
    deque<int> dq; // store indices of lcp with increasing values
    int best = 0;

    for (int r = 1; r <= n - 1; ++r) {
        // push lcp[r]
        while (!dq.empty() && lcp[dq.back()] >= lcp[r]) dq.pop_back();
        dq.push_back(r);

        // leftmost lcp index in current k-window
        int left = r - (k - 1) + 1; // equals r-k+2
        if (left >= 1) {
            // pop indices < left
            while (!dq.empty() && dq.front() < left) dq.pop_front();
            best = max(best, lcp[dq.front()]);
        }
    }

    cout << best << '\n';
    return 0;
}
```

7.6 Xâu con dài nhất ghép được từ bộ từ điển

Bài toán. Cho xâu S ($|S| \le 2000$) và n mẫu $T_1, \ldots, T_n$ (tổng độ dài $\le 10^5$). Tìm độ dài *xâu con liên tiếp* dài nhất của S có thể biểu diễn như phép **nối liên tiếp** các mẫu (mỗi mẫu dùng nhiều lần được).

Ý tưởng.

1. Dùng **Aho–Corasick** để tìm tất cả kết quả khớp mẫu trong S. Với mỗi vị trí bắt đầu i, lưu danh sách các điểm kết thúc j sao cho $S[i..j]$ là một mẫu. Ta có đồ thị DAG trên các vị trí $0..|S|$: cạnh $i \to j{+}1$ nếu tồn tại mẫu khớp $S[i..j]$.
2. Với mỗi điểm bắt đầu l (0-based), duyệt tuyến tính các vị trí từ l tới $|S|$:

$$\text{reachable}[l] = \text{true}, \quad \text{nếu reachable}[i] \Rightarrow \forall (i \to t)\ \ \text{reachable}[t] = \text{true}.$$

   Giá trị xa nhất r với $\text{reachable}[r] = \text{true}$ cho ta một xâu con hợp lệ $S[l..r-1]$ có độ dài $r - l$. Lấy max trên mọi l.

Độ phức tạp. Xây Aho $O(\sum |T_i| + \Sigma)$, quét S sinh cạnh $O(|S| + \#matches)$, và với $|S| \le 2000$ ta làm $|S|$ lần duyệt tuyến tính $\Rightarrow$ tổng chạy tốt.

```
struct Aho {
    struct Node {
        int next[26];
        int link = -1;
        vector<int> out; // lengths of patterns ending here
        Node() { memset(next, -1, sizeof(next)); }
    };
    vector<Node> tr;
    Aho() { tr.emplace_back(); }

    void add(const string& s) {
        int u = 0;
        for (char ch : s) {
            int c = ch - 'a';
            if (tr[u].next[c] == -1) {
                tr[u].next[c] = (int)tr.size();
                tr.emplace_back();
            }
            u = tr[u].next[c];
        }
        tr[u].out.push_back((int)s.size());
    }

    void build() {
        queue<int> q;
        tr[0].link = 0;
        for (int c = 0; c < 26; ++c) {
            int v = tr[0].next[c];
            if (v != -1) {
                tr[v].link = 0;
                q.push(v);
            } else {
                tr[0].next[c] = 0;
            }
        }
        while (!q.empty()) {
            int v = q.front(); q.pop();
            int f = tr[v].link;
            // merge outputs
            if (!tr[f].out.empty()) {
                // append shorter outputs
                tr[v].out.insert(tr[v].out.end(), tr[f].out.begin(), tr[f].out.end());
            }
            for (int c = 0; c < 26; ++c) {
                int u = tr[v].next[c];
                if (u != -1) {
                    tr[u].link = tr[f].next[c];
                    q.push(u);
```

```
            } else {
                tr[v].next[c] = tr[f].next[c];
            }
        }
    }
};

int main() {
    int n;
    if (!(cin >> n)) return 0;
    string S;
    cin >> S;
    vector<string> T(n);
    for (int i = 0; i < n; ++i) cin >> T[i];

    int N = (int)S.size();

    Aho aho;
    for (auto &t : T) {
        if (!t.empty()) aho.add(t);
    }
    aho.build();

    // Collect edges: from start i to end+1 for each match S[i..end]
    vector<vector<int>> nexts(N + 1); // edges from position i (0..N-1) to t (1..N)
    int state = 0;
    for (int i = 0; i < N; ++i) {
        int c = S[i] - 'a';
        if (c < 0 || c >= 26) { // defensive (though problem states lowercase)
            state = 0;
        } else {
            state = aho.tr[state].next[c];
        }
        if (!aho.tr[state].out.empty()) {
            for (int len : aho.tr[state].out) {
                int start = i - len + 1;
                if (start >= 0) {
                    nexts[start].push_back(i + 1); // edge start -> i+1
                }
            }
        }
    }

    // For each start l, propagate reachability along edges
    int best = 0;
    vector<char> reach(N + 1, 0);
    for (int l = 0; l < N; ++l) {
        // reset reach and propagate in one pass
        fill(reach.begin() + l, reach.end(), 0);
        reach[l] = 1;
        int farthest = l;
        for (int i = l; i <= N; ++i) {
            if (!reach[i]) continue;
            farthest = max(farthest, i);
            if (i == N) continue;
            for (int t : nexts[i]) {
                if (!reach[t]) reach[t] = 1;
            }
        }
        best = max(best, farthest - l);
        // early exit if remaining length can't beat best
        if (N - (l + 1) <= best) continue;
    }

    cout << best << '\n';
    return 0;
}
```

8 Hình học

- **Vector:** $\vec{AB} = (x_B - x_A,\ y_B - y_A)$
- **Tích vô hướng (Dot):** $\vec{a} \cdot \vec{b} = a_x b_x + a_y b_y$ $|\vec{a}| = \sqrt{\vec{a} \cdot \vec{a}}$ $\cos\theta = \dfrac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|}$
- **Tích có hướng (Cross):** $\vec{a} \times \vec{b} = a_x b_y - a_y b_x$ Ý nghĩa:
  - > 0: $\vec{a} \to \vec{b}$ quay ngược chiều kim đồng hồ (CCW)
  - < 0: quay thuận chiều kim đồng hồ (CW)
  - $= 0$: thẳng hàng
- **Diện tích tam giác** ABC**:** $S_{\triangle ABC} = \dfrac{|\vec{AB} \times \vec{AC}|}{2}$
- **Diện tích đa giác (Shoelace):** Với các điểm $P_1, \ldots, P_n$:

$$S = \frac{1}{2}\left|\sum_{i=1}^{n}(x_i y_{i+1} - y_i x_{i+1})\right| \quad \text{với } P_{n+1} = P_1$$

- **Khoảng cách từ** P **đến đường thẳng** (AB)**:** $d(P, AB) = \dfrac{|\vec{AP} \times \vec{AB}|}{|\vec{AB}|}$
- **Khoảng cách từ** P **đến đoạn** AB**:**
  - Nếu hình chiếu H của P nằm trên AB: $d = \dfrac{|\vec{AP} \times \vec{AB}|}{|\vec{AB}|}$
  - Ngược lại: $d = \min(|\vec{AP}|,\ |\vec{BP}|)$
- **Điểm thuộc đoạn thẳng:** P thuộc $AB \iff \vec{AP} \times \vec{AB} = 0$ và $\vec{AP} \cdot \vec{BP} \le 0$
- **Hai đoạn** AB **và** CD **cắt nhau nếu:** $ccw(A,B,C) \cdot ccw(A,B,D) \le 0$ và $ccw(C,D,A) \cdot ccw(C,D,B) \le 0$
- **Góc giữa hai vector:** $\cos\theta = \dfrac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|}$, $\sin\theta = \dfrac{|\vec{a} \times \vec{b}|}{|\vec{a}||\vec{b}|}$
- **Đường tròn:** Chu vi $= 2\pi r$, Diện tích $= \pi r^2$
- **Diện tích tam giác theo 3 cạnh (Heron):** $p = \dfrac{a+b+c}{2}$, $S = \sqrt{p(p-a)(p-b)(p-c)}$
- **Đường tròn nội tiếp:** Tâm là giao của 3 đường phân giác, $r = \dfrac{2S}{a+b+c}$
- **Đường tròn ngoại tiếp:** Tâm là giao của 3 trung trực, $R = \dfrac{abc}{4S}$

9 Ghi chú

```
#ifndef DEBUG_TEMPLATE_CPP
```

```cpp
#define DEBUG_TEMPLATE_CPP
namespace __DEBUG_UTIL__
{
    using namespace std;
    template <typename T>
    concept is_iterable = requires(T &&x) { begin(x); } &&
                          !is_same_v<remove_cvref_t<T>, string>;
    void print(const char *x) { cerr << x; }
    void print(char x) { cerr << "\'" << x << "\'"; }
    void print(bool x) { cerr << (x ? "T" : "F"); }
    void print(string x) { cerr << "\"" << x << "\""; }
    void print(vector<bool> &v)
    { /* Overloaded this because stl optimizes vector<bool> by using
         _Bit_reference instead of bool to conserve space. */
        int f = 0;
        cerr << '{';
        for (auto &&i : v)
            cerr << (f++ ? "," : "") << (i ? "T" : "F");
        cerr << "}";
    }
    template <typename T>
    void print(T &&x)
    {
        if constexpr (is_iterable<T>)
            if (size(x) && is_iterable<decltype(*(begin(x)))>)
            { /* Iterable inside Iterable */
                int f = 0;
                cerr << "\n~~~~~\n";
                for (auto &&i : x)
                {
                    cerr << setw(2) << left << f++, print(i), cerr << "\n";
                }
                cerr << "~~~~~\n";
            }
            else
            { /* Normal Iterable */
                int f = 0;
                cerr << "{";
                for (auto &&i : x)
                    cerr << (f++ ? "," : ""), print(i);
                cerr << "}";
            }
        else if constexpr (requires { x.pop(); }) /* Stacks, Priority Queues, Queues */
        {
            auto temp = x;
            int f = 0;
            cerr << "{";
            if constexpr (requires { x.top(); })
                while (!temp.empty())
                    cerr << (f++ ? "," : ""), print(temp.top()), temp.pop();
            else
                while (!temp.empty())
                    cerr << (f++ ? "," : ""), print(temp.front()), temp.pop();
            cerr << "}";
        }
        else if constexpr (requires { x.first; x.second; }) /* Pair */
        {
            cerr << '(', print(x.first), cerr << ',', print(x.second), cerr << ')';
        }
        else if constexpr (requires { get<0>(x); }) /* Tuple */
        {
            int f = 0;
            cerr << '(', apply([&f](auto... args)
                               { ((cerr << (f++ ? "," : ""), print(args)), ...); },
                               x);
            cerr << ')';
        }
        else
            cerr << x;
    }
    template <typename T, typename... V>
    void printer(const char *names, T &&head, V &&...tail)
    {
        int i = 0;
        for (size_t bracket = 0; names[i] != '\0' and (names[i] != ',' or bracket != 0); i++)
            if (names[i] == '(' or names[i] == '<' or names[i] == '{')
                bracket++;
            else if (names[i] == ')' or names[i] == '>' or names[i] == '}')
                bracket--;
        cerr.write(names, i) << " = ";
        print(head);
        if constexpr (sizeof...(tail))
            cerr << " ||", printer(names + i + 1, tail...);
        else
            cerr << "]\n";
    }
    template <typename T, typename... V>
    void printerArr(const char *names, T arr[], size_t N, V... tail)
    {
        size_t i = 0;
        for (; names[i] and names[i] != ','; i++)
            cerr << names[i];
        for (i++; names[i] and names[i] != ','; i++)
;
        cerr << " = {";
        for (size_t ind = 0; ind < N; ind++)
            cerr << (ind ? "," : ""), print(arr[ind]);
        cerr << "}";
        if constexpr (sizeof...(tail))
            cerr << " ||", printerArr(names + i + 1, tail...);
        else
            cerr << "]\n";
    }
}
#ifndef ONLINE_JUDGE
#define debug(...) std::cerr << __LINE__ << ": [", __DEBUG_UTIL__::printer(#__VA_ARGS__,
    __VA_ARGS__)
#define debugArr(...) std::cerr << __LINE__ << ": [", __DEBUG_UTIL__::printerArr(#__VA_ARGS__,
    __VA_ARGS__)
#else
#define debug(...)
#define debugArr(...)
#endif
#endif
```