Contents

1. Thi cử

1.1. Checklists

1. Wrong answer:
   - [ ] Clear data structure sau mỗi test case chưa ?
   - [ ] Thuật có đúng trong giới hạn input không ?
   - [ ] Đọc lại đề
   - [ ] Xét trường hợp biên chưa ?
   - [ ] Hiểu đúng đề chưa ?
   - [ ] Có biến nào chưa khởi tạo không ?
   - [ ] Tràn số ?
   - [ ] Nhầm biến (N với M, i với j) ?
   - [ ] Có chắc thuật đúng không ?
   - [ ] Có case nào không ngờ đến không ?
   - [ ] Nếu dùng STL, các hàm STL có hoạt động như ý muốn không ?
   - [ ] Debug bằng assert.
   - [ ] Trao đổi với teammate / 2 người cùng code.
   - [ ] Output format đúng chưa ?
   - [ ] Đọc lại checklist.
2. Runtime error:
   - [ ] Test trường hợp biên chưa ?
   - [ ] Biến chưa khởi tạo ?
   - [ ] Tràn mảng ?
   - [ ] Fail assert nào đó ?
   - [ ] Chia/mod cho 0 ?
   - [ ] Đệ quy vô hạn ?
   - [ ] Con trỏ hoặc iterator ?
   - [ ] Dùng quá nhiều bộ nhớ ?
   - [ ] Spam sub để debug (e.g. remapped signals, see Various).
3. Time limit exceeded:
   - [ ] Lặp vô hạn ?
   - [ ] Độ phức tạp có đúng không ?
   - [ ] Tối ưu mod ?
   - [ ] Copy biến quá nhiều ?
   - [ ] Thay `vector, map` thành `array, unordered_map` ? Thay `int` thành `short` ?
4. Memory limit exceeded:
- [ ] Tối đa cần bao nhiêu bộ nhớ ?
- [ ] Clear data structure sau mỗi test case chưa ?

1.2. Advices

- Khi không còn bài gì để làm thì hẵng làm hình.
- Nếu không sure bất cứ điều gì (kể cả đọc đề), hãy thảo luận với teammate.
- Viết pseudocode trước khi code, không chỉ để tiết kiệm computer time, mà còn tự phản biện chính mình.
- Đừng debug code trên máy. In code và debug output rồi debug trên giấy.
- Nếu kẹt, hãy đi dạo hoặc đi vệ sinh. Có thể nghĩ ra gì đó đấy.
- Nếu bị WA liên tục, để tạm đấy và xem bài khác rồi quay lại.
- Đừng ngại viết lại hết code, thường chỉ mất khoảng 15 phút.
- Nếu có thể dễ sinh ra input lớn hoặc tricky test, hãy cố làm điều đó trước khi nộp.
- Làm xong bài nào thì ném và xoá mọi thứ liên quan đến nó (đề bài, giấy nháp, ...).
- Ghi lại xem ai đang làm bài nào.
- Cuối giờ, mọi người tập trung vào 1 bài thôi.

1.3. commands

```sh
# lệnh dịch và flag để debug
g++ -fdiagnostics-color=always -std=gnu++20 -O2 -g -static -Wall -Wextra -Warith-conversion -Wlogical-op -Wshift-overflow=2 -Wduplicated-cond -Wcast-qual -Wcast-align -D_GLIBCXX_DEBUG -D_FORTIFY_SOURCE=2 -DLOCAL -fstack-protector
# kiểm tra sha1 để đảm bảo gõ template đúng, 8 kí tự hex trong description là 8 kí tự đầu sha1sum của code sau khi đã bỏ hết dấu cách, comment, tab, xuống dòng,...
cpp -dD -P -fpreprocessed file_name | tr -d '[:space:]'| sha1sum | cut -c-8
```

1.4. macros

```h
#pragma GCC optimize("Ofast,unroll-loops")       // unroll long, simple loops
#pragma GCC target("avx2,fma")                  // vectorizing code
#pragma GCC target("lzcnt,popcnt,abm,bmi,bmi2")  // for fast bitset operation

#include <bits/extc++.h>  // bits/stdc++.h + extensions

#include <tr2/dynamic_bitset>

using namespace std;
using namespace __gnu_pbds;  // ordered_set, gp_hash_table
using namespace __gnu_cxx;   // rope, cut and insert subarray in O(logn)

// for templates to work
#define rep(i, a, b) for (int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
#define pb push_back
#define eb emplace_back
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

// fast map
const int RANDOM = chrono::high_resolution_clock::now().time_since_epoch()

struct chash {  // customize hash function for gp_hash_table
  int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<int, int, chash> table;

/* ordered set
    find_by_order(k): returns an iterator to the k-th element (0-based)
    order_of_key(k): returns the number of elements in the set that are strictly less than k
*/
template <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;

// dynamic bitset
using bs = tr2::dynamic_bitset<uint64_t>;

/*  rope
    rope <int> cur = v.substr(l, r - l + 1);
    v.erase(l, r - l + 1);
    v.insert(v.mutable_begin(), cur);
```

```
*/
```

2. Trick & Ghi chú

2.1. Sequences

2.1.1. Catalan

$$C_n = \frac{1}{n+1}\binom{2n}{n}, C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$

2.1.2. Lucas

Let $n = n_k p^k + n_{k-1} p^{k-1} + ... + n_0$ and $m = m_k p^k + m_{k-1} p^{k-1} + ... + m_0$ in base p.

$$\binom{n}{m} = \prod_{i=0}^{k} \binom{n_i}{m_i} \bmod p$$

.

2.1.3. Number of Derangements

$d(n)$ là số hoán vị n phần tử mà không có i sao cho $p_i = i$.

$$d(n) = (n-1)(d(n-1) + d(n-2))$$

.

2.1.4. Số Stirling loại 1

Số hoán vị n phần tử có đúng k chu trình.

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

$$\sum_{k=0}^{n} s(n,k)x^k = x(x+1)...(x+n-1)$$

2.1.5. Số Stirling loại 2

Số cách chia n phần tử vào đúng k nhóm.

$$S(n,k) = kS(n-1,k) + S(n-1,k-1)$$

$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j} j^n$$

2.2. Bổ đề Burnside

Đặt G là nhóm hữu hạn tác động lên tập X. Với mỗi $g \in G$, gọi X^g là tập các điểm bất định bởi g ($\{x \in X \mid g.x = x\}$). Số quỹ đạo có thể có là:

$$\left|\frac{X}{G}\right| = \frac{1}{|G|}\sum_{g \in G} |X^g|$$

2.3. Super interpretation of kth powers

The square of the size of a set is equal to the number of ordered pairs of elements in the set. So we iterate over pairs and for each we compute the contribution to the answer.

Similarly, the k-th power is equal to the number of sequences (tuples) of length k.

$$E(X^2) = E(\#\text{ordered pairs}), E(X^k) = E(\#\text{ordered tuples})$$

2.4. Power technique

If you want to maintain the sum of k-th powers, it might help to also maintain the sum of smaller powers. For example, if the sum of 0-th, 1-th and 2-nd powers is S_0, S_1 and S_2, and we increase all elements by x, the new sums are S_0, $S_1 + S_0 x$ and $S_2 + 2xS_1 + x^2 S_0$.

2.5. Định lý Pick

Cho một đa giác có các điểm nguyên. Gọi i là số điểm nguyên nằm trong đa giác, và b là số điểm nguyên nằm trên cạnh. Diện tích của đa giác là: $A = i + \frac{b}{2} - 1$.

2.6. Nhận xét

- Trong đồ thị 2 phía, MIS = N - cặp ghép cực đại.
- Cho 2 xâu S, T. Số xâu phân biệt của prefix(S) + suffix(T) = $|S| * |T|$ - số kí tự giống nhau của S và T, không tính S_0 và T_n.

3. Toán

3.1. MillerRabin

Kiểm tra số nguyên tố nhanh, **chắc chắn** đúng trong `unsigned long long`. (458fb286)

```
bool isPrime(ull n) {
  if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
  ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
      s = __builtin_ctzll(n - 1), d = n >> s;
  for (ull a : A) {  // ^ count trailing zeroes
    ull p = modpow(a % n, d, n), i = s;
    while (p != 1 && p != n - 1 && a % n && i--) p = modmul(p, p, n);
    if (p != n - 1 && i != s) return 0;
  }
  return 1;
}
```

3.2. Matrix

Ma trận vuông, hỗ trợ nhân và luỹ thừa. (9d7c19f0)

```
/*
  Matrix<int> A(3);
  A.d = {{{{1, 2, 3}}, {{4, 5, 6}}, {{7, 8, 9}}}};
  vector<int> vec = {1, 2, 3};
  vec = (A ^ N) * vec;
*/

template <class T>
struct Matrix {
  typedef Matrix M;
  int N;
  vector<vector<T>> d;
  Matrix(int n) : N(n), d(n, vector<T>(n, 0)) {}
  M operator*(const M& m) const {
    M a(N);
    rep(i, 0, N) rep(j, 0, N) rep(k, 0, N) a.d[i][j] += d[i][k] * m.d[k][j];
    return a;
  }
  vector<T> operator*(const vector<T>& vec) const {
    vector<T> ret(N);
    rep(i, 0, N) rep(j, 0, N) ret[i] += d[i][j] * vec[j];
    return ret;
  }
  M operator^(ll p) const {
    assert(p >= 0);
    M a(N), b(*this);
    rep(i, 0, N) a.d[i][i] = 1;
    while (p) {
      if (p & 1) a = a * b;
      b = b * b;
      p >>= 1;
    }
```

```
    return a;
  }
};
```

3.3. ModLog

Tìm $x > 0$ nhỏ nhất sao cho $a^x = b \bmod m$, hoặc -1. `modLog(a,1,m)` trả về order của a trong $\mathbb{Z}_m^*$. Độ phức tạp $O(\sqrt{m})$. (b9ee5a0a)

```
ll modLog(ll a, ll b, ll m) {
  ll n = (ll)sqrt(m) + 1, e = 1, f = 1, j = 1;
  unordered_map<ll, ll> A;
  while (j <= n && (e = f = e * a % m) != b % m) A[e * b % m] = j++;
  if (e == b % m) return j;
  if (gcd(m, e) == gcd(m, b))
    rep(i, 2, n + 2) if (A.count(e = e * f % m)) return n * i - A[e];
  return -1;
}
```

3.4. ModSQRT

Tìm căn bậc hai modulo p nguyên tố trong trung bình $O(\log p)$. (735ac7d8)

```
ll modsqrt(ll a, ll p) {
  a %= p;
  if (a < 0) a += p;
  if (a == 0) return 0;

  if (modpow(a, (p - 1) / 2, p) != 1) return -1;
  if (p % 4 == 3) return modpow(a, (p + 1) / 4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1, n = 2;
  int r = 0, m;
  while (s % 2 == 0) ++r, s /= 2;
  /// find a non-square mod p
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p), g = modpow(n, s, p);
  for (;; r = m) {
    ll t = b;
    for (m = 0; m < r && t != 1; ++m) t = t * t % p;
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

3.5. Factor

Tìm một ước của n nhanh trong $O(\sqrt[4]{n}\log n)$. Phân tích đệ quy n thành thừa số nguyên tố. (27e26e39)

```
ull pollard(ull n) {
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  auto f = [&](ull x) { return modmul(x, x, n) + i; };
  while (t++ % 40 || __gcd(prd, n) == 1) {
    if (x == y) x = ++i, y = f(x);
    if ((q = modmul(prd, max(x, y) - min(x, y), n))) prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}
vector<ull> factor(ull n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), all(r));
  return l;
}
```

3.6. CRT

Duy trì hệ phương trình đồng dư. (545277d5)

```
template <typename T>
struct CRT {
  T res;
  CRT() { res = 0, prd = 1; }
  // Add condition: res % p == r
  void add(T p, T r) {
    res += mul(r - res % p + p, euclid(prd, p).first + p, p) * prd;
    prd *= p;
    if (res >= prd) res -= prd;
  }

 private:
  T prd;
  T mul(T a, T b, T p) {
    a %= p, b %= p;
    T q = (T)((long double)a * b / p);
    T r = a * b - q * p;
    while (r < 0) r += p;
    while (r >= p) r -= p;
    return r;
  }
  pair<T, T> euclid(T a, T b) {
    if (!b) return make_pair(1, 0);
    pair<T, T> r = euclid(b, a % b);
    return make_pair(r.second, r.first - a / b * r.second);
  }
};
```

3.7. DivModSum

Tính $\sum_{i=0}^{n-1} \frac{a + i \times d}{m}$ và $\sum_{i=0}^{n-1} (a + i \times d) \bmod m$. Độ phức tạp $O(\log N)$ (e0fe4359)

```
ll sumsq(ll to) { return to / 2 * ((to - 1) | 1); }

// sum( (a + d*i) / m ) for i in [0, n-1]
ll divsum(ll a, ll d, ll m, ll n) {
  ll res = d / m * sumsq(n) + a / m * n;
  d %= m, a %= m;
  if (!d) return res;
  ll to = (n * d + a) / m;
  return res + (n - 1) * to - divsum(m - 1 - a, m, d, to);
}
```

```
}
// sum( (a + d*i) % m ) for i in [0, n-1]
ll modsum(ll a, ll d, ll m, ll n) {
  a = ((a % m) + m) % m, d = ((d % m) + m) % m;
  return n * a + d * sumsq(n) - m * divsum(a, d, m, n);
}
```

3.8. FFT

FFT trên $\mathbb{R}$ (a762eff9)

```
#pragma once

typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vector<complex<long double>> R(2, 1);
  static vector<C> rt(2, 1);  // (^ 10% faster if double)
  for (static int k = 2; k < n; k *= 2) {
    R.resize(n);
    rt.resize(n);
    auto x = polar(1.0L, acos(-1.0L) / k);
    rep(i, k, 2 * k) rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
  }
  vi rev(n);
  rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
  rep(i, 0, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j, 0, k) {
        auto x = (double*)&rt[j + k],
             y = (double*)&a[i + j + k];
        C z(x[0] * y[0] - x[1] * y[1],
            x[0] * y[1] + x[1] * y[0]);
        a[i + j + k] = a[i + j] - z;
        a[i + j] += z;
      }
}
vd conv(const vd& a, const vd& b) {
  if (a.empty() || b.empty()) return {};
  vd res(sz(a) + sz(b) - 1);
  int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
  vector<C> in(n), out(n);
  copy(all(a), begin(in));
  rep(i, 0, sz(b)) in[i].imag(b[i]);
  fft(in);
  for (C& x : in) x *= x;
  rep(i, 0, n) out[i] = in[-i & (n - 1)] - conj(in[i]);
  fft(out);
  rep(i, 0, sz(res)) res[i] = imag(out[i]) / (4 * n);
  return res;
}
```

3.9. NTT

FFT modulo nguyên tố **bất kỳ** dựa trên FFT thực. (de1ee60d)

```
#include "FFT.h"

typedef vector<ll> vl;
template <int M>
vl convMod(const vl &a, const vl &b) {
  if (a.empty() || b.empty()) return {};
  vl res(sz(a) + sz(b) - 1);
  int B = 32 - __builtin_clz(sz(res)), n = 1 << B, cut = int(sqrt(M));
  vector<C> L(n), R(n), outs(n), outl(n);
  rep(i, 0, sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
  rep(i, 0, sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
  fft(L), fft(R);
  rep(i, 0, n) {
    int j = -i & (n - 1);
    outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
    outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
  }
  fft(outl), fft(outs);
  rep(i, 0, sz(res)) {
    ll av = ll(real(outl[i]) + .5), cv = ll(imag(outs[i]) + .5);
    ll bv = ll(imag(outl[i]) + .5) + ll(real(outs[i]) + .5);
    res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
  }
  return res;
}
```

3.10. FST

Tính tích chập AND, OR, XOR. (e376044a)

```
template <typename T>
void FST(vector<T>& a, bool inv, string type) {
  for (int n = sz(a), step = 1; step < n; step *= 2) {
    for (int i = 0; i < n; i += 2 * step) rep(j, i, i + step) {
        T &u = a[j], &v = a[j + step];
        if (type == "and") tie(u, v) = inv ? tuple{v - u, u} : tuple{v, u + v};
        else if (type == "or") tie(u, v) = inv ? tuple{v, u - v} : tuple{u + v, u};
        else if (type == "xor") tie(u, v) = tuple{u + v, u - v};
      }
  }
  if (inv && type == "xor")
    for (T& x : a) x /= sz(a);
}
template <typename T>
vector<T> conv(vector<T> a, vector<T> b, string type) {
  FST(a, 0, type);
  FST(b, 0, type);
  rep(i, 0, sz(a)) a[i] *= b[i];
  FST(a, 1, type);
  return a;
}
```

3.11. LinearRecurrence

Tìm số hạng thứ k của dãy truy hồi cấp n `S[i] = sum S[i-j-1]tr[j]` trong $O(n^2 \log k)$. (93f0e156)

```
// Usage: linearRec({0, 1}, {1, 1}, k) -> k'th
Fibonacci number

typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
  int n = sz(tr);

  auto combine = [&](Poly a, Poly b) {
    Poly res(n * 2 + 1);
    rep(i, 0, n + 1) rep(j, 0, n + 1) res[i + j] =
        (res[i + j] + a[i] * b[j]) % mod;
    for (int i = 2 * n; i > n; --i)
      rep(j, 0, n) res[i - 1 - j] = (res[i - 1 - j] +
res[i] * tr[j]) % mod;
    res.resize(n + 1);
    return res;
  };

  Poly pol(n + 1), e(pol);
  pol[0] = e[1] = 1;

  for (++k; k; k /= 2) {
    if (k % 2) pol = combine(pol, e);
    e = combine(e, e);
  }

  ll res = 0;
  rep(i, 0, n) res = (res + pol[i + 1] * S[i]) % mod;
  return res;
}
```

3.12. BerlekampMassey

Phục hồi một dãy truy hồi cấp n từ $2n$ số hạng đầu tiên trong $O(n^2)$. (76d26fc3)

```
vector<ll> berlekampMassey(vector<ll> s) {
  int n = sz(s), L = 0, m = 0;
  vector<ll> C(n), B(n), T;
  C[0] = B[0] = 1;
  ll b = 1;
  rep(i, 0, n) {
    ++m;
    ll d = s[i] % mod;
    rep(j, 1, L + 1) d = (d + C[j] * s[i - j]) % mod;
    if (!d) continue;
    T = C;
    ll coef = d * modpow(b, mod - 2) % mod;
    rep(j, m, n) C[j] = (C[j] - coef * B[j - m]) %
mod;
    if (2 * L > i) continue;
    L = i + 1 - L;
    B = T;
    b = d;
    m = 0;
  }
  C.resize(L + 1);
  C.erase(C.begin());
  for (ll& x : C) x = (mod - x) % mod;
  return C;
}
```

3.13. Lagrange

Tìm đa thức bậc $n - 1$ qua n điểm trong $O(n^2)$. Vẫn đúng trong trường modulo. (d12aad85)

```
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
  vd res(n), temp(n);
  rep(k, 0, n - 1) rep(i, k + 1, n) y[i] = (y[i] -
y[k]) / (x[i] - x[k]);
  double last = 0;
  temp[0] = 1;
  rep(k, 0, n) rep(i, 0, n) {
    res[i] += y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] -= last * x[k];
  }
  return res;
}
```

3.14. RowEchelon

Chuyển ma trận về dạng bậc thang trong $O(M^2N)$, với M là số hàng, N là số cột. (51f95f4f)

```
using ld = double;
using mat = vector<vector<ld>>;
const ld EPS = 1e-9;

pair<vector<int>, int> RowEchelon(mat& A) {
  int n = A.size(), m = A[0].size(), sgn = 1;
  vector<int> piv;
  for (int i = 0, rnk = 0; i < m && rnk < n; ++i) {
    for (int j = rnk + 1; j < n; ++j)
      if (abs(A[j][i]) > abs(A[rnk][i]))
        swap(A[j], A[rnk]), sgn = -sgn;
    if (abs(A[rnk][i]) < EPS) continue;
    for (int j = 0; j < n; ++j) {
      ld coef = A[j][i] / A[rnk][i];
      if (j == rnk || abs(coef) < EPS) continue;
      for (int k = 0; k < m; ++k)
        A[j][k] -= coef * A[rnk][k];
    }
    piv.push_back(i); ++rnk;
  }
  return {piv, sgn};
}
```

3.15. MatrixDet

Tính định thức ma trận vuông trong $O(n^3)$. (0a077691)

```
#include "RowEchelon.h"

ld Determinant(mat& A) {
  int n = A.size();
  ld det = RowEchelon(A).second;
  for (int i = 0; i < n; ++i)
    det *= A[i][i];
  return det;
}
```

3.16. MatrixInv

Tìm ma trận nghịch đảo trong $O(n^3)$. (e16cd43b)

```
#include "RowEchelon.h"
```

```
bool Invert(mat& A) {
  int n = A.size(); // assert(n == A[0].size());
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
      A[i].push_back(i == j);
  auto [piv, sgn] = RowEchelon(A);
  if ((int)piv.size() < n || piv.back() >= n) return
false;
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j)
      // For adjunct, do A[i][j + n] *= sgn instead.
      A[i][j + n] /= A[i][i];
    A[i].erase(A[i].begin(), A[i].begin() + n);
  }
  return true;
}
```

3.17. SolveLinear

Giải hệ phương trình tuyến tính sau khi chuyển về dạng bậc thang trong $O(M^2N)$. (1d083365)

```
#include "RowEchelon.h"

vector<ld> SolveLinear(mat& A) {
  int m = A[0].size() - 1;
  auto piv = RowEchelon(A).first;
  if (piv.empty() || piv.back() == m) return {};
  vector<ld> sol(m, 0.);
  for (int i = 0; i < (int)piv.size(); ++i)
    sol[piv[i]] = A[i][m] / A[i][piv[i]];
  return sol;
}
```

3.18. GaussBinary

Giải hệ phương trình tuyến tính modulo 2 trong $O\left(\frac{n^3}{64}\right)$ sử dụng bitset. (32593702)

```
vector<bs> solve_linear(int n, int m, vector<bs>
A, bs b) {
  int rk = 0;
  rep(j, 0, m) {
    if (rk == n) break;
    rep(i, rk + 1, n) if (A[i][j]) {
      swap(A[rk], A[i]);
      if (b[rk] != b[i]) b[rk] = !b[rk], b[i] = !
b[i];
      break;
    }
    if (!A[rk][j]) continue;
    rep(i, 0, n) if (i != rk) {
      if (A[i][j]) {
        b[i] = b[i] ^ b[rk], A[i] = A[i] ^ A[rk];
      }
    }
    ++rk;
  }
  rep(i, rk, n) if (b[i]) return {};
  vector<bs> res(1, bs(m));

  vi pivot(m, -1);
  int p = 0;
  rep(i, 0, rk) {
    while (!A[i][p]) ++p;
    res[0][p] = b[i], pivot[p] = i;
  }
  rep(j, 0, m) if (pivot[j] == -1) {
    bs x(m);
    x[j] = 1;
    rep(k, 0, j) if (pivot[k] != -1 && A[pivot[k]]
[j]) x[k] = 1;
    res.eb(x);
  }
  return res;
}
```

3.19. PolyRoots

Tìm các nghiệm phức của đa thức bậc n trong $O(n^2 \times T)$ với T là số lần lặp hội tụ. Nếu mò được nghiệm, hãy cho chúng vào phần khởi tạo. (c17e693c)

```
using C = complex<double>;

vector<C> PolyRoots(vector<C> p) {
  int n = p.size() - 1;
  vector<C> ret(n);
  for (int i = 0; i < n; ++i)
    ret[i] = pow(C{0.456, 0.976}, i); // các nghiệm
ban đầu để tăng khả năng hội tụ
  for (int it = 0; it < 1000; ++it) {
    for (int i = 0; i < n; ++i) {
      C up = 0, dw = 1;
      for (int j = n; j >= 0; --j) {
        up = up * ret[i] + p[j];
        if (j != i && j != n)
          dw = dw * (ret[i] - ret[j]);
      }
      ret[i] -= up / dw / p[n];
    }
  }
  return ret;
}
```

4. Cấu trúc dữ liệu

4.1. DSURollback

(8aba86ef)

```
struct DSURollback {
  vi e;
  vector<pii> st;
  DSURollback(int n) : e(n, -1) {}
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x :
find(e[x]); }
  int time() { return sz(st); }
  void rollback(int t) {
    for (int i = time(); i-- > t;) e[st[i].first] =
st[i].second;
    st.resize(t);
  }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.push_back({a, e[a]});
```

```
    st.push_back({b, e[b]});
    e[a] += e[b];
    e[b] = a;
    return true;
  }
};
```

4.2. LineContainer

Duy trì tập các đường thẳng dạng $y = kx + m$ và truy vấn giá trị lớn nhất tại điểm x. Nếu muốn tìm giá trị nhỏ nhất, đổi dấu `k`, `m` và kết quả truy vấn. (`0efebf77`)

```
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) {  // floored division
    return a / b - ((a ^ b) < 0 && a % b);
  }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k)
      x->p = x->m > y->m ? inf : -inf;
    else
      x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p) isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

4.3. Splay

Code Splay của anh Hạnh. (`a023974a`)

```
struct Node {
  Node *child[2], *parent;
  bool reverse;
  int value, size;
  long long sum;
};

Node *nil, *root;

void initTree() {
  nil = new Node();
  nil->child[0] = nil->child[1] = nil->parent = nil;
  nil->value = nil->size = nil->sum = 0;
  nil->reverse = false;
  root = nil;
}

void pushDown(Node *x) {
  if (x == nil) return;
  if (x->reverse) {
    swap(x->child[0], x->child[1]);
    x->child[0]->reverse = !x->child[0]->reverse;
    x->child[1]->reverse = !x->child[1]->reverse;
    x->reverse = false;
  }
}

void update(Node *x) {
  pushDown(x->child[0]);
  pushDown(x->child[1]);
  x->size = x->child[0]->size + x->child[1]->size + 1;
  x->sum = x->child[0]->sum + x->child[1]->sum + x->value;
}

void setLink(Node *x, Node *y, int d) {
  x->child[d] = y;
  y->parent = x;
}

int getDir(Node *x, Node *y) { return x->child[0] == y ? 0 : 1; }

void rotate(Node *x, int d) {
  Node *y = x->child[d], *z = x->parent;
  setLink(x, y->child[d ^ 1], d);
  setLink(y, x, d ^ 1);
  setLink(z, y, getDir(z, x));
  update(x);
  update(y);
}

void splay(Node *x) {
  while (x->parent != nil) {
    Node *y = x->parent, *z = y->parent;
    int dy = getDir(y, x), dz = getDir(z, y);
    if (z == nil)
      rotate(y, dy);
    else if (dy == dz)
      rotate(z, dz), rotate(y, dy);
    else
      rotate(y, dy), rotate(z, dz);
  }
}

Node *nodeAt(Node *x, int pos) {
  while (pushDown(x), x->child[0]->size != pos)
    if (pos < x->child[0]->size)
      x = x->child[0];
    else
      pos -= x->child[0]->size + 1, x = x->child[1];
  return splay(x), x;
}
```

```
void split(Node *x, int left, Node *&t1, Node *&t2) {
  if (left == 0)
    t1 = nil, t2 = x;
  else {
    t1 = nodeAt(x, left - 1);
    t2 = t1->child[1];
    t1->child[1] = t2->parent = nil;
    update(t1);
  }
}

Node *join(Node *x, Node *y) {
  if (x == nil) return y;
  x = nodeAt(x, x->size - 1);
  setLink(x, y, 1);
  update(x);
  return x;
}
```

4.4. PersistentIT

(c3e126da)

```
struct Node {
  int left, right;  // ID of left child & right child
  long long ln;     // Max value of node
  Node() {}
  Node(long long ln, int left, int right) : ln(ln),
left(left), right(right) {}
} it[11000111];  // Each node has a position in this
array, called ID
int nNode;

int ver[MN];  // ID of root in each version

// Update max value of a node
inline void refine(int cur) {
  it[cur].ln = max(it[it[cur].left].ln,
it[it[cur].right].ln);
}

// Update a range, and return new ID of node
int update(int l, int r, int u, int x, int oldId) {
  if (l == r) {
    ++nNode;
    it[nNode] = Node(x, 0, 0);
    return nNode;
  }

  int mid = (l + r) >> 1;
  int cur = ++nNode;

  if (u <= mid) {
    it[cur].left = update(l, mid, u, x,
it[oldId].left);
    it[cur].right = it[oldId].right;
    refine(cur);
  } else {
    it[cur].left = it[oldId].left;
    it[cur].right = update(mid + 1, r, u, x,
it[oldId].right);
    refine(cur);
  }

  return cur;
}

// Get max of range. Same as usual IT
int get(int nodeId, int l, int r, int u, int v) {
  if (v < l || r < u) return -1;
  if (u <= l && r <= v) return it[nodeId].ln;

  int mid = (l + r) >> 1;
  return max(get(it[nodeId].left, l, mid, u, v),
             get(it[nodeId].right, mid + 1, r, u,
v));
}

// When update:
++nVer;
ver[nVer] = update(1, n, u, x, ver[nVer - 1]);

// When query:
res = get(ver[t], 1, n, u, v);
```

4.5. WaveletTree

(ed6c801b)

```
struct Node {
  Node *l = 0, *r = 0;
  int lo, hi;
  vi C;  // C[i] = # of first i elements going left
  Node(const vi& A, int lo, int hi) : lo(lo), hi(hi),
C(1, 0) {
    if (lo + 1 == hi) return;
    int mid = (lo + hi) / 2;
    vi L, R;
    for (int a : A) {
      C.push_back(C.back());
      if (a < mid)
        L.push_back(a), C.back()++;
      else
        R.push_back(a);
    }
    l = new Node(L, lo, mid), r = new Node(R, mid,
hi);
  }
  // k'th (0-indexed) element in the sorted range [L,
R)
  int quantile(int k, int L, int R) {
    if (lo + 1 == hi) return lo;
    int c = C[R] - C[L];
    if (k < c) return l->quantile(k, C[L], C[R]);
    return r->quantile(k - c, L - C[L], R - C[R]);
  }
  // number of elements in range [0, R) equal to x
  int rank(int x, int R) {
    if (lo + 1 == hi) return R;
    if (x < l->hi) return l->rank(x, C[R]);
    return r->rank(x, R - C[R]);
  }
  // number of elements x in range [L, R) st. a <= x
< b
  int rectangle(int a, int b, int L, int R) {
    if (a <= lo && hi <= b) return R - L;
    if (a >= hi || b <= lo) return 0;
    return l->rectangle(a, b, C[L], C[R]) +
```

```
      r->rectangle(a, b, L - C[L], R - C[R]);
  }
};
```

5. Đồ thị

5.1. HopcroftKarp

Cặp ghép cực đại trên đồ thị 2 phía trong $O\left(E\sqrt{V}\right)$. 0-indexed.
Cách dùng: `vi btoa(m, -1); hopcroftKarp(g, btoa);`
(0584e70d)

```
bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi& B) {
  if (A[a] != L) return 0;
  A[a] = -1;
  for (int b : g[a])
    if (B[b] == L + 1) {
      B[b] = 0;
      if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
        return btoa[b] = a, 1;
    }
  return 0;
}

int hopcroftKarp(vector<vi>& g, vi& btoa) {
  int res = 0;
  vi A(g.size()), B(btoa.size()), cur, next;
  for (;;) {
    fill(all(A), 0);
    fill(all(B), 0);
    /// Find the starting nodes for BFS (i.e. layer 0).
    cur.clear();
    for (int a : btoa)
      if (a != -1) A[a] = -1;
    rep(a, 0, sz(g)) if (A[a] == 0) cur.push_back(a);
    /// Find all layers using bfs.
    for (int lay = 1;; lay++) {
      bool islast = 0;
      next.clear();
      for (int a : cur)
        for (int b : g[a]) {
          if (btoa[b] == -1) {
            B[b] = lay;
            islast = 1;
          } else if (btoa[b] != a && !B[b]) {
            B[b] = lay;
            next.push_back(btoa[b]);
          }
        }
      if (islast) break;
      if (next.empty()) return res;
      for (int a : next) A[a] = lay;
      cur.swap(next);
    }
    /// Use DFS to scan for augmenting paths.
    rep(a, 0, sz(g)) res += dfs(a, 0, g, btoa, A, B);
  }
}
```

5.2. GeneralMatching

Tìm cặp ghép cực đại trên đồ thị thường trong $O(V^3)$. 0-indexed.
(780e7569)

```
vector<int> GeneralMatching(vector<vector<int>>& graph) {
  int n = graph.size(), timer = -1;
  vector<int> mate(n, -1), label(n), parent(n),
              orig(n), aux(n, -1), q;
  auto lca = [&](int x, int y) {
    for (timer++; ; swap(x, y)) {
      if (x == -1) continue;
      if (aux[x] == timer) return x;
      aux[x] = timer;
      x = (mate[x] == -1 ? -1 : orig[parent[mate[x]]]);
    }
  };
  auto blossom = [&](int v, int w, int a) {
    while (orig[v] != a) {
      parent[v] = w; w = mate[v];
      if (label[w] == 1) label[w] = 0, q.push_back(w);
      orig[v] = orig[w] = a; v = parent[w];
    }
  };
  auto augment = [&](int v) {
    while (v != -1) {
      int pv = parent[v], nv = mate[pv];
      mate[v] = pv; mate[pv] = v; v = nv;
    }
  };
  auto bfs = [&](int root) {
    fill(label.begin(), label.end(), -1);
    iota(orig.begin(), orig.end(), 0);
    q.clear();
    label[root] = 0; q.push_back(root);
    for (int i = 0; i < (int)q.size(); ++i) {
      int v = q[i];
      for (auto x : graph[v]) {
        if (label[x] == -1) {
          label[x] = 1; parent[x] = v;
          if (mate[x] == -1)
            return augment(x), 1;
          label[mate[x]] = 0; q.push_back(mate[x]);
        } else if (label[x] == 0 && orig[v] != orig[x]) {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a); blossom(v, x, a);
        }
      }
    }
    return 0;
  };
  // Time halves if you start with (any) maximal matching.
  for (int i = 0; i < n; i++)
    if (mate[i] == -1)
      bfs(i);
  return mate;
}
```

5.3. PushRelabel

Tìm maxflow bằng Push-relabel trong $O(V^2\sqrt{E})$ (nhanh hơn Dinic). (654d5e05)

```
struct PushRelabel {
  struct Edge {
    int dest, back;
    ll f, c;
  };
  vector<vector<Edge>> g;
  vector<ll> ec;
  vector<Edge*> cur;
  vector<vi> hs;
  vi H;
  PushRelabel(int n) : g(n), ec(n), cur(n), hs(2 *
n), H(n) {}

  void addEdge(int s, int t, ll cap, ll rcap = 0) {
    if (s == t) return;
    g[s].push_back({t, sz(g[t]), 0, cap});
    g[t].push_back({s, sz(g[s]) - 1, 0, rcap});
  }

  void addFlow(Edge& e, ll f) {
    Edge& back = g[e.dest][e.back];
    if (!ec[e.dest] && f)
hs[H[e.dest]].push_back(e.dest);
    e.f += f;
    e.c -= f;
    ec[e.dest] += f;
    back.f -= f;
    back.c += f;
    ec[back.dest] -= f;
  }
  ll calc(int s, int t) {
    int v = sz(g);
    H[s] = v;
    ec[t] = 1;
    vi co(2 * v);
    co[0] = v - 1;
    rep(i, 0, v) cur[i] = g[i].data();
    for (Edge& e : g[s]) addFlow(e, e.c);

    for (int hi = 0;;) {
      while (hs[hi].empty())
        if (!hi--) return -ec[s];
      int u = hs[hi].back();
      hs[hi].pop_back();
      while (ec[u] > 0)  // discharge u
        if (cur[u] == g[u].data() + sz(g[u])) {
          H[u] = 1e9;
          for (Edge& e : g[u])
            if (e.c && H[u] > H[e.dest] + 1) H[u] =
H[e.dest] + 1, cur[u] = &e;
          if (++co[H[u]], !--co[hi] && hi < v)
            rep(i, 0, v) if (hi < H[i] && H[i] < v)--
co[H[i]], H[i] = v + 1;
          hi = H[u];
        } else if (cur[u]->c && H[u] == H[cur[u]-
>dest] + 1)
          addFlow(*cur[u], min(ec[u], cur[u]->c));
        else
          ++cur[u];
    }
  }
  bool leftOfMinCut(int a) { return H[a] >= sz(g); }
};
```

5.4. MinAssignment

Nhanh hơn Hungarian nhiều. Muốn tìm max cost, đặt cost âm. 0-indexed.(a6a71796)

```
pair<ll, vector<int>> MinAssignment(const
vector<vector<ll>>& W) {
  int n = W.size(), m = W[0].size();  // assert(n <=
m);
  vector<ll> v(m), dist(m);            // v: potential
  vector<int> L(n, -1), R(m, -1);      // matching
pairs
  vector<int> idx(m), prev(m);
  iota(idx.begin(), idx.end(), 0);

  ll w, h;
  int j, l, s, t;
  auto reduce = [&]() {
    if (s == t) {
      l = s;
      w = dist[idx[t++]];
      for (int k = t; k < m; ++k) {
        j = idx[k];
        h = dist[j];
        if (h > w) continue;
        if (h < w) t = s, w = h;
        idx[k] = idx[t];
        idx[t++] = j;
      }
      for (int k = s; k < t; ++k) {
        j = idx[k];
        if (R[j] < 0) return 1;
      }
    }
    int q = idx[s++], p = R[q];
    for (int k = t; k < m; ++k) {
      j = idx[k];
      h = W[p][j] - W[p][q] + v[q] - v[j] + w;
      if (h < dist[j]) {
        dist[j] = h;
        prev[j] = p;
        if (h == w) {
          if (R[j] < 0) return 1;
          idx[k] = idx[t];
          idx[t++] = j;
        }
      }
    }
    return 0;
  };
  for (int i = 0; i < n; ++i) {
    for (int k = 0; k < m; ++k) dist[k] = W[i][k] -
v[k], prev[k] = i;
    s = t = 0;
    while (!reduce());
    for (int k = 0; k < l; ++k) v[idx[k]] +=
dist[idx[k]] - w;
    for (int k = -1; k != i;) R[j] = k = prev[j],
swap(j, L[k]);
  }
```

```
  ll ret = 0;
  for (int i = 0; i < n; ++i) ret += W[i][L[i]];  // (i, L[i]) is a solution
  return {ret, L};
}
```

5.5. Biconnected

Tìm tất cả thành phần song liên thông trong $O(E+V)$, và với mỗi thành phần chạy callback cho mỗi cạnh. (`f6883dc4`)

```
/**
 * Usage:
 *  int eid = 0; ed.resize(N);
 *  for each edge (a,b) {
 *    ed[a].emplace_back(b, eid);
 *    ed[b].emplace_back(a, eid++); }
 *  bicomps([\&](const vi\& edgelist) {...});
 */

vi num, st;
vector<vector<pii>> ed;
int Time;
template <class F>
int dfs(int at, int par, F& f) {
  int me = num[at] = ++Time, top = me;
  for (auto [y, e] : ed[at])
    if (e != par) {
      if (num[y]) {
        top = min(top, num[y]);
        if (num[y] < me) st.push_back(e);
      } else {
        int si = sz(st);
        int up = dfs(y, e, f);
        top = min(top, up);
        if (up == me) {
          st.push_back(e);
          f(vi(st.begin() + si, st.end()));
          st.resize(si);
        } else if (up < me)
          st.push_back(e);
        else { /* e is a bridge */
        }
      }
    }
  return top;
}

template <class F>
void bicomps(F f) {
  num.assign(sz(ed), 0);
  rep(i, 0, sz(ed)) if (!num[i]) dfs(i, -1, f);
}
```

5.6. 2SAT

(`8dbaa5bf`)

```
/**
 * Usage: TwoSat ts(number of boolean variables);
 *  ts.either(0, ~3); // Var 0 is true or var 3 is false
 *  ts.setValue(2); // Var 2 is true
 *  ts.atMostOne({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true
 *  ts.solve(); // Returns true iff it is solvable
 *  ts.values[0..N-1] holds the assigned values to the vars
 */
struct TwoSat {
  int N;
  vector<vi> gr;
  vi values;  // 0 = false, 1 = true

  TwoSat(int n = 0) : N(n), gr(2 * n) {}

  int addVar() {  // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++;
  }

  void either(int f, int j) {
    f = max(2 * f, -1 - 2 * f);
    j = max(2 * j, -1 - 2 * j);
    gr[f].push_back(j ^ 1);
    gr[j].push_back(f ^ 1);
  }
  void setValue(int x) { either(x, x); }

  void atMostOne(const vi& li) {  // (optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i, 2, sz(li)) {
      int next = addVar();
      either(cur, ~li[i]);
      either(cur, next);
      either(~li[i], next);
      cur = ~next;
    }
    either(cur, ~li[1]);
  }

  vi val, comp, z;
  int time = 0;
  int dfs(int i) {
    int low = val[i] = ++time, x;
    z.push_back(i);
    for (int e : gr[i])
      if (!comp[e]) low = min(low, val[e] ?: dfs(e));
    if (low == val[i]) do {
        x = z.back();
        z.pop_back();
        comp[x] = low;
        if (values[x >> 1] == -1) values[x >> 1] = x & 1;
      } while (x != i);
    return val[i] = low;
  }

  bool solve() {
    values.assign(N, -1);
    val.assign(2 * N, 0);
    comp = val;
    rep(i, 0, 2 * N) if (!comp[i]) dfs(i);
    rep(i, 0, N) if (comp[2 * i] == comp[2 * i + 1]) return 0;
    return 1;
```

```
  }
};
```

5.7. Dominator

Dựng Dominator Tree cho đồ thị có hướng khi đặt gốc là s. u là cha của v nếu mọi đường đi từ s đến v đều phải đi qua u. Độ phức tạp $O(M \log N)$ hằng số thấp. (f57e7195)

```
vector<int> DomTree(vector<vector<int>>& graph,
int src) {
  int n = graph.size();
  vector<vector<int>> tree(n), trans(n), buck(n);
  vector<int> semi(n), par(n), dom(n), label(n),
atob(n, -1), btoa(n, -1), link(n, -1);

  function<int(int, int)> find = [&](int u, int d) {
    if (link[u] == -1) return d ? -1 : u;
    int v = find(link[u], d + 1);
    if (v < 0) return u;
    if (semi[label[link[u]]] < semi[label[u]])
label[u] = label[link[u]];
    link[u] = v;
    return d ? v : label[u];
  };
  int t = 0;
  function<void(int)> dfs = [&](int u) {
    atob[u] = t;
    btoa[t] = u;
    label[t] = semi[t] = t;
    t++;
    for (auto v : graph[u]) {
      if (atob[v] == -1) dfs(v), par[atob[v]] =
atob[u];
      trans[atob[v]].push_back(atob[u]);
    }
  };
  dfs(src);
  for (int u = t - 1; u >= 0; --u) {
    for (auto v : trans[u]) semi[u] = min(semi[u],
semi[find(v, 0)]);
    if (u) buck[semi[u]].push_back(u);
    for (auto w : buck[u]) {
      int v = find(w, 0);
      dom[w] = semi[v] == semi[w] ? semi[w] : v;
    }
    if (u) link[u] = par[u];
  }
  vector<int> ret(n, -1);
  for (int u = 1; u < t; ++u) {
    if (dom[u] != semi[u]) dom[u] = dom[dom[u]];
    ret[btoa[u]] = btoa[dom[u]];
  }
  return ret;
}
```

5.8. GomoryHu

Dựng cây Gomory-Hu của đồ thị luồng trong $N - 1$ lần chạy luồng. Max flow/min cut giữa 2 đỉnh u, v trên đồ thị luồng là trọng số cạnh nhỏ nhất trên đường đi từ u đến v. (5d1daffc)

```
typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
  vector<Edge> tree;
  vi par(N);
  rep(i, 1, N) {
    PushRelabel D(N);  // Dinic also works
    for (Edge t : ed) D.addEdge(t[0], t[1], t[2],
t[2]);
    tree.push_back({i, par[i], D.calc(i, par[i])});
    rep(j, i + 1, N) if (par[j] == par[i] &&
D.leftOfMinCut(j)) par[j] = i;
  }
  return tree;
}
```

5.9. MinCostMaxFlow

Min-cost max-flow. If costs can be negative, call `setpi` before `maxflow`, not support negative cycle. To obtain the actual flow, look at positive values only.

Time: $O(FE \log(V))$ where F is max flow. $O(VE)$ for `setpi`. (dbccc874)

```
const ll INF = numeric_limits<ll>::max() / 4;

struct MCMF {
  struct edge {
    int from, to, rev;
    ll cap, cost, flow;
  };
  int N;
  vector<vector<edge>> ed;
  vi seen;
  vector<ll> dist, pi;
  vector<edge*> par;

  MCMF(int N) : N(N), ed(N), seen(N), dist(N), pi(N),
par(N) {}

  void addEdge(int from, int to, ll cap, ll cost) {
    if (from == to) return;
    ed[from].push_back(edge{from, to, sz(ed[to]),
cap, cost, 0});
    ed[to].push_back(edge{to, from, sz(ed[from]) - 1,
0, -cost, 0});
  }

  void path(int s) {
    fill(all(seen), 0);
    fill(all(dist), INF);
    dist[s] = 0;
    ll di;

    __gnu_pbds::priority_queue<pair<ll, int>> q;
    vector<decltype(q)::point_iterator> its(N);
    q.push({0, s});

    while (!q.empty()) {
      s = q.top().second;
      q.pop();
      seen[s] = 1;
      di = dist[s] + pi[s];
      for (edge& e : ed[s])
        if (!seen[e.to]) {
          ll val = di - pi[e.to] + e.cost;
          if (e.cap - e.flow > 0 && val < dist[e.to])
{
```

```
          dist[e.to] = val;
          par[e.to] = &e;
          if (its[e.to] == q.end())
            its[e.to] = q.push({-dist[e.to],
e.to});
          else
            q.modify(its[e.to], {-dist[e.to],
e.to});
        }
      }
    }
    rep(i, 0, N) pi[i] = min(pi[i] + dist[i], INF);
  }

  pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
      ll fl = INF;
      for (edge* x = par[t]; x; x = par[x->from])
        fl = min(fl, x->cap - x->flow);

      totflow += fl;
      for (edge* x = par[t]; x; x = par[x->from]) {
        x->flow += fl;
        ed[x->to][x->rev].flow -= fl;
      }
    }
    rep(i, 0, N) for (edge& e : ed[i]) totcost +=
e.cost * e.flow;
    return {totflow, totcost / 2};
  }

  // If some costs can be negative, call this before
maxflow:
  void setpi(int s) {  // (otherwise, leave this out)
    fill(all(pi), INF);
    pi[s] = 0;
    int it = N, ch = 1;
    ll v;
    while (ch-- && it--) {
      rep(i, 0, N) {
        if (pi[i] != INF)
          for (edge& e : ed[i])
            if (e.cap)
              if ((v = pi[i] + e.cost) < pi[e.to])
pi[e.to] = v, ch = 1;
      }
    }
    assert(it >= 0);  // negative cost cycle
  }
};
```

5.10. GlobalMinCut

Tìm lát cắt cực tiểu trong đồ thị vô hướng trong $O(V^3)$. (251586c7)

```
pair<int, vi> globalMinCut(vector<vi> mat) {
  pair<int, vi> best = {INT_MAX, {}};
  int n = sz(mat);
  vector<vi> co(n);
  rep(i, 0, n) co[i] = {i};
  rep(ph, 1, n) {
    vi w = mat[0];
    size_t s = 0, t = 0;
    rep(it, 0, n - ph) {  // O(V^2) -> O(E log V)
with prio. queue
      w[t] = INT_MIN;
      s = t, t = max_element(all(w)) - w.begin();
      rep(i, 0, n) w[i] += mat[t][i];
    }
    best = min(best, {w[t] - mat[t][t], co[t]});
    co[s].insert(co[s].end(), all(co[t]));
    rep(i, 0, n) mat[s][i] += mat[t][i];
    rep(i, 0, n) mat[i][s] = mat[s][i];
    mat[0][t] = INT_MIN;
  }
  return best;
}
```

5.11. DirectedMST

Trả về giá trị và các cạnh của cây khung nhỏ nhất trên đồ thị có hướng với đỉnh nguồn cho trước trong $O(E \log V)$. Nếu không tồn tại in ra -1. (84be161b)

```
struct DSU {
  vector<int> link;
  DSU(int n) : link(n, -1) {}
  int Find(int x) { return link[x] == -1 ? x :
link[x] = Find(link[x]); }
};

struct SkewHeap {
  struct Node { ll key, lazy = 0; int l = -1, r =
-1; };
  vector<Node> T;

  void push(int x) {
    if (x == -1 || !T[x].lazy) return;
    for (int y : {T[x].l, T[x].r}) if (y != -1)
      T[y].lazy += T[x].lazy;
    T[x].key += T[x].lazy, T[x].lazy = 0;
  }
  // Make new node. Returns its index. Indexes go 0,
1, ...
  int New(ll key) {
    T.push_back(Node{key});
    return (int)T.size() - 1;
  }
  // Increment all values in heap p by v
  void Add(int x, ll v) { if (~x) T[x].lazy += v,
push(x); }
  // Merge heaps a and b
  int Merge(int a, int b) {
    if (b == -1 || a == -1) return a + b + 1;
    if (T[a].key > T[b].key) swap(a, b);
    int &l = T[a].l, &r = T[a].r;
    push(r); swap(l, r); l = Merge(l, b);
    return a;
  }
  void Pop(int& x) { x = Merge(T[x].l, T[x].r); }
  ll Get(int x) { return T[x].key; }
};

struct Edge { int a, b; ll c; };

pair<ll, vector<int>> DMST(int n, int src,
vector<Edge> es) {
  // Compress graph - O(M logN)
```

```
  SkewHeap H; DSU D(2 * n); int x = 0;
  vector<int> par(2 * n, -1), ins(par), vis(par);
  for (auto e : es) ins[e.b] = H.Merge(ins[e.b], H.New(e.c));
  auto go = [&](int x) { return D.Find(es[ins[x]].a); };
  for (int i = n; ins[x] != -1; ++i) {
    for (; vis[x] == -1; x = go(x)) vis[x] = 0;
    for (; x != i; x = go(x)) {
      int rem = ins[x]; ll w = H.Get(rem); H.Pop(rem);
      H.Add(rem, -w); ins[i] = H.Merge(ins[i], rem);
      par[x] = i; D.link[x] = i;
    }
    for (; ins[x] != -1 && go(x) == x; H.Pop(ins[x]));
  }
  // Expand graph - O(N)
  ll cost = 0; vector<int> ans;
  for (int i = src; i != -1; i = par[i]) vis[i] = 1;
  for (int i = x; i >= 0; --i) {
    if (vis[i]) continue;
    cost += es[ins[i]].c; ans.push_back(ins[i]);
    for (int j = es[ins[i]].b; j != -1 && !vis[j]; j = par[j])
      vis[j] = 1;
  }
  return {cost, ans};
}
```

6. Xâu

6.1. Z

(791227fd)

```
vi Z(const string& S) {
  vi z(sz(S));
  int l = -1, r = -1;
  rep(i, 1, sz(S)) {
    z[i] = i >= r ? 0 : min(r - i, z[i - l]);
    while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]]) z[i]++;
    if (i + z[i] > r) l = i, r = i + z[i];
  }
  return z;
}
```

6.2. MinRotation

Tìm cyclic shift của xâu có thứ tự từ điển nhỏ nhất trong $O(n)$. (b7274ea5)

```
int minRotation(string s) {
  int a = 0, N = sz(s);
  s += s;
  rep(b, 0, N) rep(k, 0, N) {
    if (a + k == b || s[a + k] < s[b + k]) {
      b += max(0, k - 1);
      break;
    }
    if (s[a + k] > s[b + k]) {
      a = b;
      break;
    }
  }
  return a;
}
```

6.3. Manacher

(aac2e18e)

```
array<vi, 2> manacher(const string& s) {
  int n = sz(s);
  array<vi, 2> p = {vi(n + 1), vi(n)};
  rep(z, 0, 2) for (int i = 0, l = 0, r = 0; i < n; i++) {
    int t = r - i + !z;
    if (i < r) p[z][i] = min(t, p[z][l + t]);
    int L = i - p[z][i], R = i + p[z][i] - !z;
    while (L >= 1 && R + 1 < n && s[L - 1] == s[R + 1]) p[z][i]++, L--, R++;
    if (R > r) l = L, r = R;
  }
  return p;
}
```

6.4. AhoCorasick

(1d6ed2c7)

```
struct aho_corasick {
  struct node {
    int suffix_link = -1, exit_link = -1, cnt = 0, nxt[26];
    node() { fill(nxt, nxt + 26, -1); }
  };
  vector<node> g = {node()};
  void insert_string(const string &s) {
    int p = 0;
    for (char c : s) {
      if (g[p].nxt[c - 'a'] == -1) {
        g[p].nxt[c - 'a'] = g.size();
        g.emplace_back();
      }
      p = g[p].nxt[c - 'a'];
    }
    g[p].cnt++;
  }
  void build_automaton() {
    for (deque<int> q = {0}; q.size(); q.pop_front()) {
      int v = q.front(), suffix_link = g[v].suffix_link;
      if (v)
        g[v].exit_link =
            g[suffix_link].cnt ? suffix_link : g[suffix_link].exit_link;
      for (int i = 0; i < 26; i++) {
        int &nxt = g[v].nxt[i], nxt_sf = v ? g[suffix_link].nxt[i] : 0;
        if (nxt == -1)
          nxt = nxt_sf;
        else {
          g[nxt].suffix_link = nxt_sf;
          q.push_back(nxt);
        }
      }
    }
  }
```

```
};
```

6.5. SuffixArray

Suffix Array và LCP trong $O(n \log n)$.(9f6f8f7d)

```
struct SuffixArray {
  vi sa, lcp;
  SuffixArray(string& s, int lim = 256) {  // or basic_string<int>
    int n = sz(s) + 1, k = 0, a, b;
    vi x(all(s)), y(n), ws(max(n, lim));
    x.push_back(0), sa = lcp = y, iota(all(sa), 0);
    for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
      p = j, iota(all(y), n - j);
      rep(i, 0, n) if (sa[i] >= j) y[p++] = sa[i] - j;
      fill(all(ws), 0);
      rep(i, 0, n) ws[x[i]]++;
      rep(i, 1, lim) ws[i] += ws[i - 1];
      for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
      swap(x, y), p = 1, x[sa[0]] = 0;
      rep(i, 1, n) a = sa[i - 1], b = sa[i],
                   x[b] = (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
    }
    for (int i = 0, j; i < n - 1; lcp[x[i++]] = k)
      for (k &&k--, j = sa[x[i] - 1]; s[i + k] == s[j + k]; k++);
  }
};
```

6.6. PalindromeTree

Dựng Palindrome Tree. Nó có 2 root, root 0/1 cho xâu đối xứng chẵn/lẻ, mỗi node lưu độ dài xâu đối xứng, số lượng và link đến xâu đó. Xâu độ dài N **chỉ có tối đa N xâu con đối xứng phân biệt**. (52a0927f)

```
struct Node {
  map<char, int> leg;
  int link, len, cnt = 0;
};

vector<Node> PalTree(string str) {
  vector<Node> T(str.size() + 2);
  T[1].link = T[1].len = 0;
  T[0].link = T[0].len = -1;
  int last = 0, nodes = 2;

  for (int i = 0; i < (int)str.size(); ++i) {
    char now = str[i];
    int node = last;
    while (now != str[i - T[node].len - 1]) node = T[node].link;
    if (T[node].leg.count(now)) {
      node = T[node].leg[now];
      T[node].cnt += 1;
      last = node;
      continue;
    }
    int cur = nodes++;
    T[cur].len = T[node].len + 2;
    T[node].leg[now] = cur;
    int link = T[node].link;
    while (link != -1) {
      if (now == str[i - T[link].len - 1] && T[link].leg.count(now)) {
        link = T[link].leg[now];
        break;
      }
      link = T[link].link;
    }
    if (link <= 0) link = 1;
    T[cur].link = link;
    T[cur].cnt = 1;
    last = cur;
  }
  for (int node = nodes - 1; node > 0; --node)
    T[T[node].link].cnt += T[node].cnt;

  T.resize(nodes);
  return T;
}
```

7. Khác

7.1. FracBinarySearch

Tìm phân số $\frac{p}{q}$ nhỏ nhất trong đoạn $[0, 1]$ sao cho $f\left(\frac{p}{q}\right)$ là đúng, với $p \leq m_p, q \leq m_q$ (adce8eda)

```
using ll = long long;
struct Frac { ll p, q; };

template <class Func>
Frac FracLowerBound(Func f, ll maxq, ll maxp) {
  Frac lo{0, 1}, hi{1, 1}, mid;  // Set hi to 1/0 for (0, inf)
  if (f(lo)) return lo;
  assert(f(hi));
  for (int it = 0, dir = 1; it < 3 || !dir; ++it) {
    // invariant: f(lo) == !dir, f(hi) == dir
    for (ll step = 1, adv = 1, now = 0;; adv ? step *= 2 : step /= 2) {
      now += step;
      mid = {lo.p * now + hi.p, lo.q * now + hi.q};
      if (abs(mid.p) > maxp || mid.q > maxq || f(mid) != dir)
        now -= step, adv = 0;
      if (!step) break;
    }
    if (mid.p != hi.p) it = 0;
    hi = lo;
    lo = mid;
    dir = !dir;
  }
  // (lo, hi) are consecutive with f(lo) == 0, f(hi) == 1
  return hi;
}
```

7.2. ContinuedFraction

Cho N và số thực $x > 0$, tính xấp xỉ hữu tỉ $\frac{p}{q}$ của x với $p, q \leq N$ trong $O(\log N)$. Đảm bảo $\left|\frac{p}{q} - x\right| < \frac{1}{q}$.

(8d63f564)

```h
typedef double d;  // for N ~ 1e7; long double for N ~ 1e9
pair<ll, ll> approximate(d x, ll N) {
  ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX;
  d y = x;
  for (;;) {
    ll lim = min(P ? (N - LP) / P : inf, Q ? (N - LQ) / Q : inf),
       a = (ll)floor(y), b = min(a, lim), NP = b * P + LP, NQ = b * Q + LQ;
    if (a > b) {
      // If b > a/2, we have a semi-convergent that gives us a
      // better approximation; if b = a/2, we *may* have one.
      // Return {P, Q} here for a more canonical approximation.
      return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (d)Q)) ? make_pair(NP, NQ)
                                                              : make_pair(P, Q);
    }
    if (abs(y = 1 / (y - (d)a)) > 3 * N) {
      return {NP, NQ};
    }
    LP = P;
    P = NP;
    LQ = Q;
    Q = NQ;
  }
}
```

7.3. 1D1D

Nếu hàm $w(i, j)$ thoả mãn bất đẳng thức tứ giác: $w(a, c) + w(b, d) \le w(a, d) + w(b, c)$ với mọi $a < b < c < d$, thì ta có thể tính hàm DP 1 chiều: $f(i) = \min_{0 \le j < i} f(j) + w(j, i)$ trong $O(n \log n)$.

```cpp
struct item {
  int l, r, p;
};

const int N = 1e5 + 3;
int n;
long long f[N];

long long w(int j, int i) {
  // một hàm cost bất kì thỏa mãn bất đẳng thức tứ giác
}

void solve() {
  deque<item> dq;
  dq.push_back({1, n, 0});
  for (int i = 1; i <= n; ++i) {
    f[i] = f[dq.front().p] + w(dq.front().p, i);
    // deque chỉ lưu giá trị từ h[i + 1]
    // tới h[n]
    ++dq.front().l;

    // nếu l > r, ta loại đoạn này khỏi deque
    if (dq.front().l > dq.front().r) {
      dq.pop_front();
    }

    while (!dq.empty()) {
      auto [l, r, p] = dq.back();
      if (f[i] + w(i, l) < f[p] + w(p, l)) {
        dq.pop_back();
        // p không còn là giá trị của
        // h[l], h[l + 1], ..., h[r]
        // lúc này, h[l]=h[l+1]=...=h[r]=i.
      } else
        break;
    }

    if (dq.empty()) {
      dq.push_back({i + 1, n, i});
      // h[i+1]=h[i+2]=...=h[n]=i
    } else {
      // tìm nhị phân vị trí pos nhỏ nhất
      // thỏa mãn h[pos] = i
      auto& [l, r, p] = dq.back();
      int low = l, high = r;
      int pos = r + 1, mid;
      while (low <= high) {
        mid = (low + high) / 2;
        if (f[i] + w(i, mid) < f[p] + w(p, mid)) {
          pos = mid, high = mid - 1;
        } else {
          low = mid + 1;
        }
      }

      // cập nhật đoạn (l,r,p) thành (l,pos-1,p)
      r = pos - 1;
      if (pos <= n) {
        dq.push_back({pos, n, i});
        // h[pos]=h[pos+1]=...=h[n]=i
      }
    }
  }
}
```

7.4. SOSDP

Toàn bộ implementation SOS DP của VNOI.

```cpp
// SOS xuôi - z(f(s))
for (int k = 0; k < n; k++)
  for (int mask = 0; mask < (1 << n); mask++)
    if (mask & (1 << k)) dp[mask] += dp[mask ^ (1 << k)];

// Hàm đan dấu - o(f(s))
for (int mask = 0; mask < (1 << n); mask++)
  sos[mask] = f[mask] * (__builtin_parity(mask) ? -1 : 1);

// SOS ngược - mu(f(s)) = ozo(f(s))
// z(mu(f)) = mu(z(f)) = f

// SOS truy hồi chính nó: f(S) = h(sum f(T), T subset S) + a(S)
for (int mask = 0; mask < (1 << n); mask++) {
  dp[mask][0] = 0;  // trường hợp cơ sở
  for (int k = 1; k <= n; k++) {
    // tính Sum over Proper Subset
```

```
    if (mask & (1 << (k - 1))) {
      int sub = mask ^ (1 << (k - 1));
      dp[mask][k] = dp[mask][k - 1] + dp[sub][k - 1] + f[sub];
    } else
      dp[mask][k] = dp[mask][k - 1];
  }
  f[mask] = h(dp[mask][n]) + a[mask];  // tính f
}

/*
  SOS 2 hàm gọi chéo nhau
  f(S) = h1(sum g(T), T subset S) + a(S)
  g(S) = h2(sum f(T), T subseteq S) + b(S)
*/
for (int mask = 0; mask < (1 << n); mask++) {
  // tính dpG và f
  for (int k = 1; k <= n; k++) {
    if (mask & (1 << (k - 1))) {  // truy hồi theo kiểu proper subset
      int sub = mask ^ (1 << (k - 1));
      dpG[mask][k] = dpG[mask][k - 1] + dpG[sub][k - 1] + g[sub];
    } else
      dpG[mask][k] = dpG[mask][k - 1];
  }
  f[mask] = h1(dpG[mask][n]) + a[mask];

  // tính dpF và g
  dpF[mask][0] = f[mask];
  for (int k = 1; k <= n; k++) {
    if (mask & (1 << (k - 1)))  // truy hồi theo kiểu subset
      dpF[mask][k] = dpF[mask][k - 1] + dpF[mask ^ (1 << (k - 1))][k - 1];
    else
      dpF[mask][k] = dpF[mask][k - 1];
  }
  g[mask] = h2(dpF[mask][n]) + b[mask];
}

// Tính tích chập SOS của 2 hàm f, g
// Make fhat[][] = {0} and ghat[][] = {0}
for (int mask = 0; mask < (1 << N); mask++) {
  fhat[__builtin_popcount(mask)][mask] = f[mask];
  ghat[__builtin_popcount(mask)][mask] = g[mask];
}

// Apply zeta transform on fhat[][] and ghat[][]
for (int i = 0; i < N; i++) {
  for (int j = 0; j < N; j++) {
    for (int mask = 0; mask < (1 << N); mask++) {
      if ((mask & (1 << j)) != 0) {
        fhat[i][mask] += fhat[i][mask ^ (1 << j)];
        ghat[i][mask] += ghat[i][mask ^ (1 << j)];
      }
    }
  }
}

// Do the convolution and store into h[][] = {0}
for (int mask = 0; mask < (1 << N); mask++) {
  for (int i = 0; i < N; i++) {
    for (int j = 0; j <= i; j++) {
      h[i][mask] += fhat[j][mask] * ghat[i - j][mask];
    }
  }
}

// Apply inverse SOS dp on h[][]
for (int i = 0; i < N; i++) {
  for (int j = 0; j < N; j++) {
    for (int mask = 0; mask < (1 << N); mask++) {
      if ((mask & (1 << j)) != 0) {
        h[i][mask] -= h[i][mask ^ (1 << j)];
      }
    }
  }
}
for (int mask = 0; mask < (1 << N); mask++)
  fog[mask] = h[__builtin_popcount(mask)][mask];
```

7.5. Knuth

Nếu hàm $w(i,j)$ thoả mãn bất đẳng thức tứ giác: $w(a,c) + w(b,d) \le w(a,d) + w(b,c)$ với mọi $a < b < c < d$, thì ta có thể tính hàm DP: $f(i,j) = \min_{i \le k < j} f(i,k) + f(k+1,j) + w(j,i)$ trong $O(n^2)$.

```
auto C = [&](int i, int j) {
  ... // Implement cost function C.
};

for (int i = 0; i < N; i++) {
  opt[i][i] = i;
  ... // Initialize dp[i][i] according to the problem
}

for (int i = N - 2; i >= 0; i--) {
  for (int j = i + 1; j < N; j++) {
    int mn = INT_MAX;
    int cost = C(i, j);
    for (int k = opt[i][j - 1]; k <= min(j - 1, opt[i + 1][j]); k++) {
      if (mn >= dp[i][k] + dp[k + 1][j] + cost) {
        opt[i][j] = k;
        mn = dp[i][k] + dp[k + 1][j] + cost;
      }
    }
    dp[i][j] = mn;
  }
}

return dp[0][N - 1];
```

7.6. HexGrid

```
int roundCount(int round) { return (6 * round); }
int roundSum(int round) { return (6 * round * (round + 1) / 2); }
int findRound(int n) {
  int res = 1;
  while (roundSum(res) < n) res++;
  return (res);
```

```cpp
}
pair<int, int> cord(int n) {
  if (n == 0) return (make_pair(0, 0));
  int c = findRound(n);
  int prev = roundSum(c - 1);
  if (n <= prev + c) return (make_pair(c, n - prev));
  if (n <= prev + 2 * c) return (make_pair(prev + 2 * c - n, c));
  if (n <= prev + 3 * c) return (make_pair(prev + 2 * c - n, prev + 3 * c - n));
  if (n <= prev + 4 * c) return (make_pair(-c, prev + 3 * c - n));
  if (n <= prev + 5 * c) return (make_pair(n - prev - 5 * c, -c));
  return (make_pair(n - prev - 5 * c, n - prev - 6 * c));
}
bool inRound(int x, int y, int c) {
  if (0 <= y && y <= c && x == c) return (true);
  if (0 <= x && x <= c && y == c) return (true);
  if (0 <= y && y <= c && y - x == c) return (true);
  if (-c <= y && y <= 0 && x == -c) return (true);
  if (-c <= x && x <= 0 && y == -c) return (true);
  if (0 <= x && x <= c && x - y == c) return (true);
  return (false);
}
int findRound(int x, int y) {
  int res = 1;
  while (!inRound(x, y, res)) res++;
  return (res);
}
int number(int x, int y) {
  if (x == 0 && y == 0) return (0);
  int c = findRound(x, y);
  int prev = roundSum(c - 1);
  if (1 <= y && y <= c && x == c) return (prev + y);
  if (0 <= x && x <= c && y == c) return (prev + 2 * c - x);
  if (0 <= y && y <= c && y - x == c) return (prev + 2 * c - x);
  if (-c <= y && y <= 0 && x == -c) return (prev + 3 * c - y);
  if (-c <= x && x <= 0 && y == -c) return (prev + 5 * c + x);
  return (prev + 5 * c + x);
}
```

7.7. MaximalCliques

Chạy một hàm nào đó duyệt qua tất cả các clique của một đồ thị trong $O(3^{\frac{n}{3}})$.(8da55bf5)

```cpp
// Usage: cliques(g, [&](const bs &clique) { callback }, ~bs(n), bs(n), bs(n));

template <class F>
void cliques(vector<bs>& eds, F f, bs P, bs X, bs R) {
  f(R);
  if (!P.any() && !X.any()) return;
  // if only need to find all maximal cliques
  // auto q = (P | X).find_first();
  // auto cands = P & ~eds[q];
  rep(i, 0, sz(eds)) if (P[i]) {
    R[i] = 1;
    cliques(eds, f, P & eds[i], X & eds[i], R);
    R[i] = P[i] = 0, X[i] = 1;
  }
}
```

7.8. MaximumClique

Tìm nhanh một clique lớn nhất. Dùng để giải Maximum Independent Set bằng cách tính maximum clique của phần bù. (91b80023)

```cpp
struct Maxclique {
  double limit = 0.025, pk = 0;
  struct Vertex {
    int i, d = 0;
  };
  typedef vector<Vertex> vv;
  vector<bs> e;
  vv V;
  vector<vi> C;
  vi qmax, q, S, old; // qmax = vertices in maximum clique, q = current clique
  void init(vv& r) {
    for (auto& v : r) v.d = 0;
    for (auto& v : r)
      for (auto j : r) v.d += e[v.i][j.i];
    sort(all(r), [](auto a, auto b) { return a.d > b.d; });
    int mxD = r[0].d;
    rep(i, 0, sz(r)) r[i].d = min(i, mxD) + 1;
  }
  void expand(vv& R, int lev = 1) {
    S[lev] += S[lev - 1] - old[lev];
    old[lev] = S[lev - 1];
    while (sz(R)) {
      if (sz(q) + R.back().d <= sz(qmax)) return;
      q.push_back(R.back().i);
      vv T;
      for (auto v : R)
        if (e[R.back().i][v.i]) T.push_back({v.i});
      if (sz(T)) {
        if (S[lev]++ / ++pk < limit) init(T);
        int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1, 1);
        C[1].clear(), C[2].clear();
        for (auto v : T) {
          int k = 1;
          auto f = [&](int i) { return e[v.i][i]; };
          while (any_of(all(C[k]), f)) k++;
          if (k > mxk) mxk = k, C[mxk + 1].clear();
          if (k < mnk) T[j++].i = v.i;
          C[k].push_back(v.i);
        }
        if (j > 0) T[j - 1].d = 0;
        rep(k, mnk, mxk + 1) for (int i : C[k]) T[j].i = i, T[j++].d = k;
        expand(T, lev + 1);
      } else if (sz(q) > sz(qmax))
        qmax = q;
      q.pop_back(), R.pop_back();
    }
  }
  vi maxClique() {
```

```
    init(V), expand(V);
    return qmax;
  }
  Maxclique(vector<bs> conn) : e(conn), C(sz(e) + 1),
S(sz(C)), old(S) {
    rep(i, 0, sz(e)) V.push_back({i});
  }
};
```

7.9. Frievalds

Kiểm tra xác suất tích ma trận $AB = C$ trong $O(Tn^2)$. Xác suất sai là 2^{-T}.(3600a0c4)

```cpp
int Freivalds(Mat a, Mat b, Mat c) {
  int n = a.n, iteration = 40;
  Mat zero(n, 1), r(n, 1);
  while (iteration--) {
    for (int i = 0; i < n; i++) r.a[i][0] = rnd() %
2;
    Mat ans = (a * (b * r)) - (c * r);
    if (ans != zero) return 0;
  }
  return 1;
}
```

7.10. XorBasis

(7bb7d19f)

```h
template <typename T = int, int B = 31>
struct Basis {
  T a[B];
  Basis() { memset(a, 0, sizeof a); }
  void insert(T x) { // insert x to the basis
    for (int i = B - 1; i >= 0; i--) {
      if (x >> i & 1) {
        if (a[i])
          x ^= a[i];
        else {
          a[i] = x;
          break;
        }
      }
    }
  }
  bool can(T x) { // can x be represent using the
basis
    for (int i = B - 1; i >= 0; i--) {
      x = min(x, x ^ a[i]);
    }
    return x == 0;
  }
  T max_xor(T ans = 0) { // maximum xor combination
in the basis
    for (int i = B - 1; i >= 0; i--) {
      ans = max(ans, ans ^ a[i]);
    }
    return ans;
  }
};
```

8. Hình

Các thuật toán hình có đa giác, nếu không chú thích gì, thì hoạt động với mọi loại đa giác (lồi, lõm, tự cắt). Khi không còn bài gì để làm nữa thì hẵng làm hình.

8.1. Point

(388cadd3)

```h
template <class T>
int sgn(T x) {
  return (x > 0) - (x < 0);
}
template <class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x = 0, T y = 0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x, y) <
tie(p.x, p.y); }
  bool operator==(P p) const { return tie(x, y) ==
tie(p.x, p.y); }
  P operator+(P p) const { return P(x + p.x, y +
p.y); }
  P operator-(P p) const { return P(x - p.x, y -
p.y); }
  P operator*(T d) const { return P(x * d, y * d); }
  P operator/(T d) const { return P(x / d, y / d); }
  T dot(P p) const { return x * p.x + y * p.y; }
  T cross(P p) const { return x * p.y - y * p.x; }
  T cross(P a, P b) const { return (a -
*this).cross(b - *this); }
  T dist2() const { return x * x + y * y; }
  long double dist() const { return sqrt((long
double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  long double angle() const { return atan2l(y, x); }
  P unit() const { return *this / dist(); }  // makes
dist()=1
  P perp() const { return P(-y, x); }        //
rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the
origin
  P rotate(double a) const {
    return P(x * cos(a) - y * sin(a), x * sin(a) + y
* cos(a));
  }
  friend ostream& operator<<(ostream& os, P p) {
    return os << "(" << p.x << "," << p.y << ")";
  }
};
```

8.2. SideOf

(dbcd89bc)

```h
#include "Point.h"

template <class P>
int sideOf(P s, P e, P p) {
  return sgn(s.cross(e, p));
}

template <class P>
int sideOf(const P& s, const P& e, const P& p, double
eps) {
  auto a = (e - s).cross(p - s);
```

```
  double l = (e - s).dist() * eps;
  return (a > l) - (a < -l);
}
```

8.3. ClosestPair

(d3f10d25)

```
#include "Point.h"

typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
  assert(sz(v) > 1);
  set<P> S;
  sort(all(v), [](P a, P b) { return a.y < b.y; });
  pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
  int j = 0;
  for (P p : v) {
    P d{1 + (ll)sqrt(ret.first), 0};
    while (v[j].y <= p.y - d.x) S.erase(v[j++]);
    auto lo = S.lower_bound(p - d), hi =
S.upper_bound(p + d);
    for (; lo != hi; ++lo) ret = min(ret, {(*lo -
p).dist2(), {*lo, p}});
    S.insert(p);
  }
  return ret.second;
}
```

8.4. ConvexHull

Trả về bao lồi của tập điểm theo CCW. Nếu muốn tính cả điểm nằm trên biên, sửa <= thành <.(57ee170a)

```
#include "Point.h"

typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
  if (sz(pts) <= 1) return pts;
  sort(all(pts));
  vector<P> h(sz(pts) + 1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(all(pts)))
    for (P p : pts) {
      while (t >= s + 2 && h[t - 2].cross(h[t - 1],
p) <= 0) t--;
      h[t++] = p;
    }
  return {h.begin(), h.begin() + t - (t == 2 && h[0]
== h[1])};
}
```

8.5. OnSegment

(a128e475)

```
#include "Point.h"

template <class P>
bool onSegment(P s, P e, P p) {
  return p.cross(s, e) == 0 && (s - p).dot(e - p) <=
0;
}
```

8.6. LineDistance

(25522bd1)

```
#include "Point.h"

template <class P>
double lineDist(const P& a, const P& b, const P& p) {
  return (double)(b - a).cross(p - a) / (b -
a).dist();
}
```

8.7. LineIntersection

(b8e9fffa)

```
#include "Point.h"
#include "Line.h"

template <class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
  auto d = (e1 - s1).cross(e2 - s2);
  if (d == 0)  // if parallel
    return {-(s1.cross(e1, s2) == 0), P(0, 0)};
  auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
  return {1, (s1 * p + e1 * q) / d};
}

tuple<T4, T4, T2> LineIntersection(Line m, Line n) {
  T2 d = (T2)m.a * n.b - (T2)m.b * n.a; // assert(d);
  T4 x = (T4)m.c * n.b - (T4)m.b * n.c;
  T4 y = (T4)m.a * n.c - (T4)m.c * n.a;
  return {x, y, d}; // (x/d, y/d) is intersection.
}
```

8.8. LineProjectionReflection

Trả về chân đường vuông góc/điểm đối xứng (tuỳ vào `refl=false/true`) của điểm p qua đường ab. Các điểm phải là số thực, cẩn thận tràn số.(a25456e3)

```
#include "Point.h"

template <class P>
P lineProj(P a, P b, P p, bool refl = false) {
  P v = b - a;
  return p - v.perp() * (1 + refl) * v.cross(p - a) /
v.dist2();
}
```

8.9. CircleLine

Định nghĩa của đường thẳng dạng $ax + by = c$ với $a, b, c \in \frac{\mathbb{Z}}{\mathbb{R}}$ (2965aaea)

```
#include "Point.h"

template <class P>
vector<P> circleLine(P c, double r, P a, P b) {
  P ab = b - a, p = a + ab * (c - a).dot(ab) /
ab.dist2();
  double s = a.cross(b, c), h2 = r * r - s * s /
ab.dist2();
  if (h2 < 0) return {};
  if (h2 == 0) return {p};
  P h = ab.unit() * sqrt(h2);
  return {p - h, p + h};
}
```

8.10. CircleIntersection

(b7e381bf)

```
#include "Point.h"

typedef Point<double> P;
bool circleInter(P a, P b, double r1, double r2,
pair<P, P>* out) {
  if (a == b) {
    assert(r1 != r2);
    return false;
  }
  P vec = b - a;
  double d2 = vec.dist2(), sum = r1 + r2, dif = r1 -
r2,
         p = (d2 + r1 * r1 - r2 * r2) / (d2 * 2), h2
= r1 * r1 - p * p * d2;
  if (sum * sum < d2 || dif * dif > d2) return false;
  P mid = a + vec * p, per = vec.perp() *
sqrt(fmax(0, h2) / d2);
  *out = {mid + per, mid - per};
  return true;
}
```

8.11. CircleTangents

Tìm các tiếp tuyến ngoài của hai hình tròn, hoặc các tiếp tuyến trong nếu `r2` âm.

- Có thể trả về `0`, `1` hoặc `2` tiếp tuyến:
- `0` nếu một hình tròn chứa (hoặc chồng lên nhau, trong trường hợp nội tiếp, hoặc nếu hai hình tròn giống hệt nhau) hình tròn kia.
- `1` nếu hai hình tròn tiếp xúc với nhau (trong trường hợp này `first` = `second` và đường tiếp tuyến vuông góc với đường nối giữa tâm).
- `first` và `second` tương ứng cho biết các điểm tiếp xúc tại hình tròn 1 và hình tròn 2.
- Để tìm các tiếp tuyến của một hình tròn với một điểm, hãy đặt `r2 = 0`.

(2422578b)

```
#include "Point.h"

template <class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2,
double r2) {
  P d = c2 - c1;
  double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr *
dr;
  if (d2 == 0 || h2 < 0) return {};
  vector<pair<P, P>> out;
  for (double sign : {-1, 1}) {
    P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
    out.push_back({c1 + v * r1, c2 + v * r2});
  }
  if (h2 == 0) out.pop_back();
  return out;
}
```

8.12. Circumcircle

(f358ca56)

```
#include "Point.h"

typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
  return (B - A).dist() * (C - B).dist() * (A -
C).dist() /
         abs((B - A).cross(C - A)) / 2;
}
P ccCenter(const P& A, const P& B, const P& C) {
  P b = C - A, c = B - A;
  return A + (b * c.dist2() - c * b.dist2()).perp() /
b.cross(c) / 2;
}
```

8.13. MinimumEnclosingCircle

(d02ef5ff)

```
#include "Circumcircle.h"

pair<P, double> mec(vector<P> ps) {
  shuffle(all(ps), mt19937(time(0)));
  P o = ps[0];
  double r = 0, EPS = 1 + 1e-8;
  rep(i, 0, sz(ps)) if ((o - ps[i]).dist() > r * EPS)
{
    o = ps[i], r = 0;
    rep(j, 0, i) if ((o - ps[j]).dist() > r * EPS) {
      o = (ps[i] + ps[j]) / 2;
      r = (o - ps[i]).dist();
      rep(k, 0, j) if ((o - ps[k]).dist() > r * EPS)
{
        o = ccCenter(ps[i], ps[j], ps[k]);
        r = (o - ps[i]).dist();
      }
    }
  }
  return {o, r};
}
```

8.14. CirclePolygonIntersection

Trả về diện tích phần giao của đường tròn với đa giác trong $O(n)$

(18024d73)

```
#include "Point.h"

typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
  auto tri = [&](P p, P q) {
    auto r2 = r * r / 2;
    P d = q - p;
    auto a = d.dot(p) / d.dist2(), b = (p.dist2() - r
* r) / d.dist2();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a - sqrt(det)), t = min(1., -a
+ sqrt(det));
    if (t < 0 || 1 <= s) return arg(p, q) * r2;
    P u = p + d * s, v = p + d * t;
    return arg(p, u) * r2 + u.cross(v) / 2 + arg(v,
q) * r2;
  };
  auto sum = 0.0;
  rep(i, 0, sz(ps)) sum += tri(ps[i] - c, ps[(i + 1)
% sz(ps)] - c);
  return sum;
```

```
}
```

8.15. InsidePolygon

(82d8c704)

```
#include "OnSegment.h"
#include "Point.h"
#include "SegmentDistance.h"

template <class P>
bool inPolygon(vector<P> &p, P a, bool strict = true)
{
  int cnt = 0, n = sz(p);
  rep(i, 0, n) {
    P q = p[(i + 1) % n];
    if (onSegment(p[i], q, a)) return !strict;
    // or: if (segDist(p[i], q, a) <= eps) return !
strict;
    cnt ^= ((a.y < p[i].y) - (a.y < q.y)) *
a.cross(p[i], q) > 0;
  }
  return cnt;
}
```

8.16. PolygonCenter

(052f9d99)

```
#include "Point.h"

typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
  P res(0, 0);
  double A = 0;
  for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++)
{
    res = res + (v[i] + v[j]) * v[j].cross(v[i]);
    A += v[j].cross(v[i]);
  }
  return res / A / 3;
}
```

8.17. PolygonArea

Trả về 2 lần diện tích có dấu của đa giác.(1d364aa9)

```
#include "Point.h"

template <class T>
T polygonArea2(vector<Point<T>>& v) {
  T a = v.back().cross(v[0]);
  rep(i, 0, sz(v) - 1) a += v[i].cross(v[i + 1]);
  return a;
}
```

8.18. PolygonUnion

Trả về diện tích giao nhau của n đa giác trong $O(N^2)$ với N là tổng số điểm. (59ab1357)

```
#include "Point.h"
#include "SideOf.h"

typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x / b.x :
a.y / b.y; }
double polyUnion(vector<vector<P>>& poly) {
  double ret = 0;
  rep(i, 0, sz(poly)) rep(v, 0, sz(poly[i])) {
    P A = poly[i][v], B = poly[i][(v + 1) %
sz(poly[i])];
    vector<pair<double, int>> segs = {{0, 0}, {1,
0}};
    rep(j, 0, sz(poly)) if (i != j) {
      rep(u, 0, sz(poly[j])) {
        P C = poly[j][u], D = poly[j][(u + 1) %
sz(poly[j])];
        int sc = sideOf(A, B, C), sd = sideOf(A, B,
D);
        if (sc != sd) {
          double sa = C.cross(D, A), sb = C.cross(D,
B);
          if (min(sc, sd) < 0) segs.emplace_back(sa /
(sa - sb), sgn(sc - sd));
        } else if (!sc && !sd && j < i && sgn((B -
A).dot(D - C)) > 0) {
          segs.emplace_back(rat(C - A, B - A), 1);
          segs.emplace_back(rat(D - A, B - A), -1);
        }
      }
    }
    sort(all(segs));
    for (auto& s : segs) s.first = min(max(s.first,
0.0), 1.0);
    double sum = 0;
    int cnt = segs[0].second;
    rep(j, 1, sz(segs)) {
      if (!cnt) sum += segs[j].first - segs[j -
1].first;
      cnt += segs[j].second;
    }
    ret += A.cross(B) * sum;
  }
  return ret / 2;
}
```

8.19. PointInsideHull

(8ea9510d)

```
#include "OnSegment.h"
#include "Point.h"
#include "SideOf.h"

typedef Point<ll> P;

bool inHull(const vector<P>& l, P p, bool strict =
true) {
  int a = 1, b = sz(l) - 1, r = !strict;
  if (sz(l) < 3) return r && onSegment(l[0],
l.back(), p);
  if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
  if (sideOf(l[0], l[a], p) >= r || sideOf(l[0],
l[b], p) <= -r) return false;
  while (abs(a - b) > 1) {
    int c = (a + b) / 2;
    (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
  }
  return sgn(l[a].cross(l[b], p)) < r;
}
```

8.20. HullDiameter

(7eae8192)

```
#include "Point.h"

typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
  int n = sz(S), j = n < 2 ? 0 : 1;
  pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
  rep(i, 0, j) for (;; j = (j + 1) % n) {
    res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
    if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0) break;
  }
  return res.second;
}
```

8.21. Minkowski

Tính tổng của 2 bao lồi trong $O(n + m)$.(e1781ee4)

```
#include "Point.h"

vector<Point> MinkowskiSum(vector<Point> P, vector<Point> Q) {
  int n = P.size(), m = Q.size();
  vector<Point> R = {P[0] + Q[0]};
  for (int i = 1, j = 1; i < n || j < m; ) {
    if (i < n && (j == m || cross(P[i] - P[i - 1], Q[j] - Q[j - 1]) > 0)) {
      R.push_back(R.back() + P[i] - P[i - 1]);
      ++i;
    } else {
      R.push_back(R.back() + Q[j] - Q[j - 1]);
      ++j;
    }
  }
  return R;
}
```

8.22. Line

Định nghĩa của đường thẳng dạng $y = kx + m$ với $k, m \in \mathbb{Z}$ hoặc $\mathbb{R}$(fec7e0fc)

```
using T = int;
using T2 = long long;
using T4 = __int128_t;
const T2 INF = 4e18;

struct Line { T a, b; T2 c; };

bool half(Line m) { return m.a < 0 || m.a == 0 && m.b < 0; };
void normalize(Line& m) {
  T2 g =gcd((T2)gcd(abs(m.a), abs(m.b)), abs(m.c));
  if (half(m)) g *= -1;
  m.a /= g, m.b /= g, m.c /= g;
}
// Sorts halfplanes in clockwise order.
// To sort lines, normalize first (gcd logic not needed).
bool operator<(Line m, Line n) {
  return make_pair(half(m), (T2)m.b * n.a) <
         make_pair(half(n), (T2)m.a * n.b);
}
Line LineFromPoints(T x1, T y1, T x2, T y2) {
  T a = y1 - y2, b = x2 - x1;
  T2 c = (T2)a * x1 + (T2)b * y1;
  return {a, b, c}; // halfplane points to the left of vec.
}
```

8.23. HalfplaneSet

Tìm bao lồi giao của nửa mặt phẳng trong $O(n \log n)$. Nửa mặt phẳng được định nghĩa bằng $ax + by \leq c$(9cd1778e)

```
#include "Line.h"
#include "LineIntersection.h"

struct HalfplaneSet : multiset<Line> {
  HalfplaneSet() {
    insert({+1, 0, INF});
    insert({0, +1, INF});
    insert({-1, 0, INF});
    insert({0, -1, INF});
  };

  auto adv(auto it, int z) {  // z = {-1, +1}
    return (z == -1 ? --(it == begin() ? end() : it)
                    : (++it == end() ? begin() : it));
  }
  bool chk(auto it) {
    Line l = *it, pl = *adv(it, -1), nl = *adv(it, +1);
    auto [x, y, d] = LineIntersection(pl, nl);
    T4 sat = l.a * x + l.b * y - (T4)l.c * d;
    if (d < 0 && sat < 0) return clear(), 0;  // unsat
    if ((d > 0 && sat <= 0) || (d == 0 && sat < 0)) return erase(it), 1;
    return 0;
  }
  void Cut(Line l) {  // add ax + by <= c
    if (empty()) return;
    auto it = insert(l);
    if (chk(it)) return;
    for (int z : {-1, +1})
      while (size() && chk(adv(it, z)));
  }
  double Maximize(T a, T b) {  // max ax + by
    if (empty()) return -1 / 0.;
    auto it = lower_bound({a, b});
    if (it == end()) it = begin();
    auto [x, y, d] = LineIntersection(*adv(it, -1), *it);
    return (1.0 * a * x + 1.0 * b * y) / d;
  }
  double Area() { // half-plane intersection area
    double total = 0.;
    for (auto it = begin(); it != end(); ++it) {
      auto [x1, y1, d1] = LineIntersection(*adv(it, -1), *it);
      auto [x2, y2, d2] = LineIntersection(*it, *adv(it, +1));
```

```
      total += (1.0 * x1 * y2 - 1.0 * x2 * y1) / d1 /
d2;
    }
    return total * 0.5;
  }
};
```