Saratov State University

ZAVRS

Eremenko, Utaliev, Yanchenko

RUCODE FINAL

20.10.2024

RUCODE FINAL

1 Basic 1
- 1.1 Pragma optimization . . . 1
- 1.2 Hash Function . . . 1

2 Math 1
- 2.1 Integration . . . 1
- 2.2 Simulated Annealing . . . 1
- 2.3 Gauss . . . 2
- 2.4 Iteration . . . 2
- 2.5 Tridiagonal Matrix Algorithm . . . 2

3 Number Theory 2
- 3.1 FFT . . . 2
- 3.2 NTT . . . 2
- 3.3 FWHT . . . 3
- 3.4 Extended Euclid . . . 3
- 3.5 Burnside Lemma . . . 3
- 3.6 Phi and Mobius . . . 3
- 3.7 Pollard Rho . . . 3
- 3.8 Miller Rabin . . . 3
- 3.9 Primitive Root . . . 3
- 3.10 Discrete Root . . . 3
- 3.11 Discrete log . . . 3
- 3.12 Diophantine Equations . . . 4
- 3.13 Stern-Brocot Tree . . . 4
- 3.14 Lattice Points Below Line . . . 4

4 Graph Theory 4
- 4.1 Biconnected Components . . . 4
- 4.2 2-SAT . . . 4
- 4.3 Kirchoffs Theorem . . . 5
- 4.4 Tuttes Theorem . . . 5
- 4.5 Matching Duals . . . 5
- 4.6 LCA . . . 5
- 4.7 Kuhn . . . 5
- 4.8 1-K bfs . . . 5
- 4.9 Floyd-Warshall . . . 5
- 4.10 Ford-Bellman with negative cycle . . . 5
- 4.11 Eulerian path . . . 5
- 4.12 Boruvka . . . 6

5 Flows 6
- 5.1 Dinics Algorithm . . . 6
- 5.2 MCMF . . . 6
- 5.3 MCMF with Potentials . . . 7
- 5.4 L-R Flow . . . 7
- 5.5 Stoer Wagner Algorithm . . . 7

6 Data Structures 8
- 6.1 Centroid Decomposition . . . 8
- 6.2 HLD . . . 8
- 6.3 Explicit Treap . . . 8
- 6.4 Sparce Table . . . 8
- 6.5 Persistent Segment Tree . . . 9
- 6.6 Persistent Treap . . . 9
- 6.7 Ordered Set . . . 9

7 Strings 10
- 7.1 String Matching with FFT . . . 10
- 7.2 Suffix Array . . . 10
- 7.3 Prefix- and z-funtion . . . 10
- 7.4 Manachers Algorithm . . . 10
- 7.5 Prefix Automaton . . . 10
- 7.6 Aho-Corasick + Trie . . . 11
- 7.7 Hash String . . . 11

8 DP 11
- 8.1 Dynamic CHT . . . 11
- 8.2 Li Chao Tree . . . 11
- 8.3 D&C Optimization . . . 12
- 8.4 Knuth Optimization . . . 12
- 8.5 SOS DP . . . 12
- 8.6 Submasks . . . 12
- 8.7 LIS binary search . . . 12

9 Geometry 12
- 9.1 Base . . . 12
- 9.2 Intersections . . . 12
- 9.3 Angles . . . 13
- 9.4 Areas . . . 13
- 9.5 Distances . . . 13
- 9.6 Circle Circle Area . . . 13
- 9.7 Circle Line Intersection . . . 13
- 9.8 Circle Circle Intersection . . . 13
- 9.9 Tangent Lines of Two Circles . . . 13
- 9.10 Convex Hull . . . 13
- 9.11 Minkowski Sum . . . 13
- 9.12 Point in Convex Polygon . . . 14
- 9.13 SVG . . . 14

10 Miscellaneous 14
- 10.1 Josephus Problem . . . 14
- 10.2 Knight Moves in Infinity Grid . . . 14

11 Other 14
- 11.1 Комбинаторика . . . 14
- 11.2 Теория чисел . . . 15
- 11.3 Геометрия . . . 15
- 11.4 Графы . . . 16
- 11.5 Формулы . . . 16
- 11.6 Полезные числа . . . 17

1 Basic

1.1 Pragma optimization

```
#pragma GCC optimize("Ofast", "no-stack-protector", "no-math-
    errno", "unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,sse4.2,popcnt,abm,
    mmx,avx,tune=native,arch=core-avx2,tune=core-avx2")
#pragma GCC ivdep
```

1.2 Hash Function

```
struct custom_hash {
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }
  size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM = chrono::steady_clock::
        now().time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};
unordered_map<long long, int, custom_hash> safe_map;
```

2 Math

2.1 Integration

```
ld sq(ld l, ld r) {
  ld d = r - l;
  return d / 6 * (f(l) + 4 * f((l + r) / 2) + f(r));
//  return d / 2 * (f(l) + f(r));
}
ld integral(ld l, ld r) {
  ld mid = (l + r) / 2;
  ld sl = sq(l, mid);
  ld sr = sq(mid, r);
  ld sa = sq(l, r);
  if (fabsl((sa - (sl + sr)) / (sl + sr)) < EPS)
    return sl + sr;
  return integral(l, mid) + integral(mid, r);
}
```

2.2 Simulated Annealing

Для нахождения глобального минимума/максимума функции.

```
const ld t0 = 1e5; // or 1e9; init temp
const ld chg = 0.99999;
ld temp = t0;
int cur = random_state(); // any random valid state
int ans = cur;
// find minimum of function f
while (temp > END_TEMP && elapsed_time() <= time_limit) {
  // save the current state
  // make temp / t0 fraction of updates
  int next = neighbor(s);
  if (func(ans) > func(next)) {
    cur = tmp;
  }
  else{
    // Probability to go to worse state
    if (rnd_ld(0, 1) <= exp((cur - tmp) / temp))
      cur = tmp;
    else
      // restore the saved state
  }
  temp *= chg;
}
```

2.3 Gauss

```
template<class T>
int gauss(vector<valarray<T>> a, vector<T> &ans) {
  int n = sz(a);
  int m = sz(a[0]) - 1;
  vector<int> where(m, -1);
  for (int col = 0, row = 0; col < m && row < n; ++col) {
    int sel = row;
    fore(i, row, n) if (abs(a[i][col]) > abs(a[sel][col]))
      sel = i;
    if (fabsl(a[sel][col]) < EPS) continue;
    swap(a[sel], a[row]);
    where[col] = row;
    forn(i, n) if (i != row) {
      T c = a[i][col] / a[row][col];
      a[i] -= a[row] * c;
    }
    ++row;
  }
  ans.assign(m, 0);
  forn(i, m) if (where[i] != -1)
    ans[i] = a[where[i]][m] / a[where[i]][i];
  forn(i, n) {
    T sum = 0;
    forn(j, m) sum += ans[j] * a[i][j];
    if (fabsl(sum - a[i][m]) > EPS) return 0;
  }
  forn(i, m) if (where[i] == -1) return INF;
  return 1;
}
```

2.4 Iteration

```
ld dist(vector<ld>& a, vector<ld>& b) {
  ld res = 0;
  forn(i, sz(a)) res += (a[i] - b[i]) * (a[i] - b[i]);
  return res;
}
vector<ld> fixed_point(vector<vector<ld>> aa, vector<ld> bb) {
  int n = sz(aa);
  vector<vector<ld>> a(n, vector<ld>(n, 0));
  vector<ld> b(n, 0);
  forn(i, n) {
    if (fabsl(aa[i][i]) < EPS)
      return vector<ld>();
    forn(j, n) if (i != j)
      a[i][j] = -aa[i][j] / aa[i][i];
    b[i] = bb[i] / aa[i][i];
  }
  vector<ld> x(n, 0);
  forn(ITERATION, 200) {
    vector<ld> nx(n, 0);
    forn(i, n) forn(j, n) if (i != j)
      nx[i] += a[i][j] * x[j];
    forn(i, n) nx[i] += b[i];
    if (dist(x, nx) < EPS)
      break;
    x = nx;
  }
  return x;
}
```

2.5 Tridiagonal Matrix Algorithm

$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i$

```
vector<ld> prog(vector<vector<ld>> mat, vector<ld> d) {
  int n = sz(d);
  vector<ld> a(n), b(n), c(n);
  forn(i, n) {
    if (i - 1 >= 0) a[i] = mat[i][i - 1];
    b[i] = -mat[i][i];
    if (i + 1 < n) c[i] = mat[i][i + 1];
  }

  vector<ld> p(n), q(n);

  p[0] = c[0] / b[0];
  q[0] = -d[0] / b[0];

  fore(i, 1, n) {
    p[i] = c[i] / (b[i] - a[i] * p[i - 1]);
    q[i] = (a[i] * q[i - 1] - d[i]) / (b[i] - a[i] * p[i - 1]);
  }

  vector<ld> x(n);
  x[n - 1] = q[n - 1];
  for (int i = n - 2; i >= 0; i--) {
    x[i] = p[i] * x[i + 1] + q[i];
  }

  return x;
}
```

3 Number Theory

3.1 FFT

```
// comp - struc of complex number
vector<comp> w[LOGN];
vector<int> rv[LOGN];
void prepare() {
  forn(st, LOGN - 1) {
    w[st].resize(1 << st);
    forn(i, 1 << st) {
      ld ang = PI / (1 << st) * i;
      w[st][i] = comp(cosl(ang), sinl(ang));
    }}
  forn(st, LOGN) {
    rv[st].resize(1 << st);
    if (st > 0) {
      int h = (1 << (st - 1)) - 1;
      forn(i, 1 << st)
        rv[st][i] = (rv[st - 1][i & h] << 1) | (i > h);
}}}
void fft(vector<comp> &a, bool inv) {
  int n = sz(a);
  int ln = __builtin_ctz(n);
  forn(i, n) {
    int ni = rv[ln][i];
    if (i < ni) swap(a[i], a[ni]);
  }
  forn(st, ln) {
    int len = 1 << st;
    for (int k = 0; k < n; k += (len << 1)) {
      fore(pos, k, k + len) {
        comp l = a[pos];
        comp r = a[pos + len] * w[st][pos - k];
        a[pos] = l + r;
        a[pos + len] = l - r;
      }}}
  if (inv) {
    forn(i, n)
      a[i] = a[i] / n;
    reverse(a.begin() + 1, a.end());
}}
vector<ll> mul(vector<int> a, vector<int> b) {
  int cnt = 1 << (32 - __builtin_clz(sz(a) + sz(b) - 1));
  vector<comp> c(cnt), tmpa(cnt), tmpb(cnt);
  forn(i, cnt) c[i] = comp(i < sz(a) ? a[i] : 0, i < sz(b) ? b[
       i] : 0);
  fft(c, false);
  forn(i, cnt) {
    tmpa[i] = (c[i] + c[(cnt - i) % cnt].conj()) / comp(2, 0);
    tmpb[i] = (c[i] - c[(cnt - i) % cnt].conj()) / comp(0, 2);
  }
  forn(i, cnt) c[i] = tmpa[i] * tmpb[i];
  fft(c, true);
  vector<ll> ans(cnt);
  forn(i, cnt) ans[i] = llrint(c[i].x);
  return ans;
}
```

3.2 NTT

```
extern const int MOD = 998244353;
const int LOGN = 20; // for every logn find own root
const int g = 3; // calc find_root
vector<Mint> w[LOGN];
vector<int> rv[LOGN];
int find_root() {
  for (int x = 1; x < mod; ++x)
    if (binpow(x, 1 << POW) == 1
          && binpow(x, 1 << (POW - 1)) != 1)
      return x;
  throw;
}
void prepare() {
  Mint wb = Mint(g).pow((MOD - 1) / (1 << LOGN));
  forn(st, LOGN - 1) {
    w[st].assign(1 << st, 1);
    Mint bw = wb.pow(1 << (LOGN - st - 1));
    Mint cw = 1;
```

```
    forn(k, 1 << st) {
      w[st][k] = cw;
      cw *= bw;}}
  forn(st, LOGN) {
    rv[st].assign(1 << st, 0);
    if (st == 0) {
      rv[st][0] = 0;
      continue;
    }
    int h = (1 << (st - 1));
    forn(k, 1 << st)
      rv[st][k] = (rv[st - 1][k & (h - 1)] << 1) | (k >= h);}}
```

3.3 FWHT

$c_i = \sum_{j,k:\, j \oplus k = i} a_j \cdot b_k$

```
vector<int> hadamard_transform(vector<int>& a) {
  vector<int> dp = a;
  for (size_t bit = 1; bit < a.size(); bit <<= 1)
    for (size_t mask = 0; mask < a.size(); mask++) {
      if ((mask & bit) == 0) {
        int u = dp[mask], v = dp[mask ^ bit];
        dp[mask] = u + v;
        dp[mask ^ bit] = u - v;}
  } return dp;
}
vector<int> inverse_hadamard_transform(vector<int>& f) {
  vector<int> dp = f;
  for (size_t bit = 1; bit < f.size(); bit <<= 1)
    for (size_t mask = 0; mask < f.size(); mask++) {
      if ((mask & bit) == 0) {
        int x = dp[mask], y = dp[mask ^ bit];
        dp[mask] = (x + y) / 2;
        dp[mask ^ bit] = (x - y) / 2;}
  } return dp;
}
// a.size() == b.size() == 2^k
vector<int> xor_convolution(vector<int>& a, vector<int>& b) {
  vector<int> f = hadamard_transform(a);
  vector<int> g = hadamard_transform(b);
  vector<int> h(f.size());
  for (size_t i = 0; i < f.size(); i++) h[i] = f[i] * g[i];
  vector<int> c = inverse_hadamard_transform(h);
  return c;
}
```

3.4 Extended Euclid

```
ll exgcd(ll a, ll b, ll& x, ll& y) {
  if (b == 0) {
    x = 1, y = 0;
    return a;
  }
  ll x1, y1;
  ll d = exgcd(b, a % b, x1, y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return d;
}
```

3.5 Burnside Lemma

Количество классов эквивалетности равно сумме количеств неподвижных точек по всем перестановкам из группы G, делённой на размер этой группы:

$$|\text{Classes}| = \frac{1}{|G|} \sum_{\pi \in G} I(\pi)$$

3.6 Phi and Mobius

```
int lst[N], mu[N], phi[N];
void sieve() {
  forn(i, N) lst[i] = phi[i] = i;
  mu[1] = 1;
  for (int i = 2; i < N; ++i) {
    if (lst[i] == lst[i / lst[i]])
      mu[i] = 0;
    else
      mu[i] = -mu[i / lst[i]];
    if (lst[i] != i) continue;
    for (int j = i; j < N; j += i) {
      phi[j] -= phi[j] / i;
      lst[j] = min(lst[j], i);
    }
  }
}
```

3.7 Pollard Rho

Факторизация числа с малыми множителями в разложении. $O(N^{1/4})$

```
uli get_factor(uli n) {
  auto f = [n](uli x){ return mod_mul(x, x, n) + 1; };
  uli x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  while (t ++ % 40 || gcd(prd, n) == 1) {
    if (x == y) x = ++ i, y = f(x);
    if (q = mod_mul(prd, max(x, y) - min(x, y), n)) prd = q;
    x = f(x), y = f(f(y));
  }
  return gcd(prd, n);
}
vector<uli> factorize(uli n) {
  if (n == 1) return {};
  if (isprime(n)) return {n};
  uli x = get_factor(n);
  auto l = factorize(x), r = factorize(n / x);
  l.insert(l.end(), r.begin(), r.end());
  return l;
}
```

3.8 Miller Rabin

```
bool isPrime(uli n) {
  if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
  uli A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
    s = __builtin_ctzll(n-1), d = n >> s;
  for (uli a : A) {
    uli 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;
}
```

3.9 Primitive Root

```
int generator(int p) {
  auto fact = factorize(p - 1);
  for (int res = 2; res <= p; ++res) {
    bool ok = true;
    for (int i = 0; i < int(fact.size()) && ok; ++i)
      ok &= binpow(res, phi / fact[i], p) != 1;
    if (ok) return res;
  }
  return -1;
}
```

3.10 Discrete Root

Найти все x: $x^k \equiv a \pmod n$

```
vector<int> roots(int a, int k, int mod){
  if (a == 0) return {0};
  int g = generator(mod); // primitive root
  // Baby-step giant-step discrete logarithm algorithm
  int sq = (int) sqrt (mod + .0) + 1;
  vector<pair<int, int>> dec(sq);
  fore(i, 1, sq + 1)
    dec[i - 1] = {binpow(g, i * sq * k % (mod - 1), mod), i};
  sort(dec.begin(), dec.end());
  int any_ans = -1;
  forn(i, sq) {
    int my = binpow(g, i * k % (mod - 1), mod) * a % mod;
    auto it = lower_bound(all(dec), mp(my, 0));
    if (it != dec.end() && it->x == my) {
      any_ans = it->y * sq - i;
      break;
    }
  }
  if (any_ans == -1) return {};
  int delta = (mod - 1) / gcd(k, mod - 1);
  vector<int> ans;
  for (int cur = any_ans % delta; cur < mod - 1; cur += delta)
    ans.pb(binpow(g, cur, mod));
  sort(all(ans));
  return ans;
}
```

3.11 Discrete log

$a^x \equiv b \pmod m$

```
int discreteLog(int a, int b, int m) {
  a %= m, b %= m;
  int n = sqrt(m) + 1;
  map<int, int> vals;
```

```
  for (int p = 1; p <= n; ++p)
    vals[binpow(a, p * n, m)] = p;
  for (int q = 0; q <= n; ++q) {
    int cur = (binpow(a, q, m) * 1ll * b) % m;
    if (vals.count(cur)) {
      int ans = vals[cur] * n - q;
      return ans;
    }
  }
  return -1;
}
```

3.12 Diophantine Equations

solutions to $ax + by = c$ where $x \in [xlow, xhigh]$ and $y \in [ylow, yhigh]$; returns {cnt, leftsol, rightsol, gcd of a and b}

```
array<li, 6> solve_linear_diophantine(li a, li b, li c, li xlow
    , li xhigh, li ylow, li yhigh){
  li x, y, g = exgcd(a >= 0 ? a : -a, b >= 0 ? b : -b, x, y);
  array<li, 6> no_sol{0, 0, 0, 0, 0, g};
  if(c % g) return no_sol;
  x *= c / g, y *= c / g;
  if(a < 0) x = -x;
  if(b < 0) y = -y;
  a /= g, b /= g, c /= g;
  auto shift = [&](li &x, li &y, li a, li b, li cnt){ x += cnt
       * b, y -= cnt * a; };
  int sign_a = a > 0 ? 1 : -1, sign_b = b > 0 ? 1 : -1;
  shift(x, y, a, b, (xlow - x) / b);
  if(x < xlow) shift(x, y, a, b, sign_b);
  if(x > xhigh) return no_sol;
  li lx1 = x;
  shift(x, y, a, b, (xhigh - x) / b);
  if(x > xhigh) shift(x, y, a, b, -sign_b);
  li rx1 = x;
  shift(x, y, a, b, -(ylow - y) / a);
  if(y < ylow) shift(x, y, a, b, -sign_a);
  if(y > yhigh) return no_sol;
  li lx2 = x;
  shift(x, y, a, b, -(yhigh - y) / a);
  if(y > yhigh) shift(x, y, a, b, sign_a);
  li rx2 = x;
  if(lx2 > rx2) swap(lx2, rx2);
  li lx = max(lx1, lx2), rx = min(rx1, rx2);
  if(lx > rx) return no_sol;
  return {(rx - lx) / (b >= 0 ? b : -b) + 1, lx, (c - lx * a) /
        b, rx, (c - rx * a) / b, g};
}
```

3.13 Stern-Brocot Tree

Дерево, содержащее все неотриц. дроби. На каждом узле стоит медианта $\frac{a+b}{c+d}$, дробей $\frac{a}{b}$ и $\frac{c}{d}$, стоящих в ближайших к этому узлу левом и правом верхних узлах. Все дроби несократимы и появляются ровно 1 раз.

Нахождение ближайшей дроби к p/q за $O(log^2(p+q))$. Храним текущую левую и правую границы p_l/q_l и p_r/q_r. Бинпоиском ищем макс. a: $\frac{p_l+a\,p_r}{q_l+a\,q_r}$ (если идём вправо по дерево), который ничего не ломает, обновляем границы.

3.14 Lattice Points Below Line

Number of integer points $(x; y)$ such for $0 \le x < n$ and $0 < y \le \lfloor kx + b \rfloor$

```
int count_lattices(Fraction k, Fraction b, long long n) {
  auto fk = k.floor();
  auto fb = b.floor();
  auto cnt = 0LL;
  if (k >= 1 || b >= 1) {
    cnt += (fk * (n - 1) + 2 * fb) * n / 2;
    k -= fk;
    b -= fb;
  }
  auto t = k * n + b;
  auto ft = t.floor();
  if (ft >= 1) {
    cnt += count_lattices(1 / k, (t - t.floor()) / k, t.floor()
        );
  }
  return cnt;
}
```

4 Graph Theory

4.1 Biconnected Components

```
// bicone - edge, biconv - vertex
vector<vector<int>>g, h, comp;
vector<pair<int,int>> edg;
vector<int>used, lvl, f, st;
void add_comp(int sz) {
  comp.pb({});
 while(st.size() > sz){
  comp.back().pb(st.back());
  st.pop_back();
}}

void dfs(int v, int p = -1) {
  used[v] = true;
  up[v] = lvl[v];
 st.pb(v); //bicone
  for (int i : g[v]){
  int u = edg[i].first ^ edg[i].second ^ v;
  if (u == p) continue;
  if (!used[u]) { //bicone
        int sz = st.size();
    lvl[u] = lvl[v] + 1;
        dfs(u, v);
        up[v] = min(up[v], up[u]);
        if (fup[u] > tin[v])
          add_comp(sz);
      } else {
        up[v] = min(up[v], lvl[u]);
      }
  if (!used[u]){ // biconv
   int sz = st.size();
   lvl[u] = lvl[v] + 1;
   st.pb(i);
   dfs(u, v);
   if (up[u] >= lvl[v]) add_comp(sz);
   up[u] = min(up[u], up[v]);
  } else{
   if (lvl[u] < lvl[v]) st.pb(i);
   up[v] = min(up[v], lvl[u]);
  }
 }
 if (p == -1 && !st.empty()) add_comp(0);
}
```

4.2 2-SAT

```
vector<vector<int>>g, gt;
vector<int>used;
vector<int>order;
vector<int>color;
void add_or(int x, int y) {
  g[x ^ 1].pb(y);
  g[y ^ 1].pb(x);
  gt[y].pb(x ^ 1);
  gt[x].pb(y ^ 1);
}
void add_im(int x, int y) {
  add_or(x ^ 1, y);
}
void add_xor0(int u, int v) {
  add_or(2 * u, 2 * v + 1);
  add_or(2 * u + 1, 2 * v);
}
void add_xor1(int u, int v) {
  add_or(2 * u, 2 * v);
  add_or(2 * u + 1, 2 * v + 1);
}
void topsort(int x) {
  used[x] = true;
  for (auto y : g[x])
    if (!used[y]) topsort(y);
  order.pb(x);
}
void css(int x, int col) {
  color[x] = col;
  for (auto y : gt[x]) {
    if (!color[y]) css(y, col);
  }
}

void init(int n) {
  order.clear();
  g.assign(2 * n, {});
  gt.assign(2 * n, {});
  used.assign(2 * n, 0);
  color.assign(2 * n, 0);
}
```

```
int32_t main() {
  /// строим граф ///
  forn(i, 2 * n) if (!used[i]) {
    topsort(i);
  }
  reverse(all(order));
  int col = 1;
  for (auto x : order) if (!color[x]) {
    css(x, col++);
  }
  forn(i, n) {
    if (color[2 * i] == color[2 * i + 1]) {
      cout << "FALSE" << endl;
      return;
    }
    // нужно перебрать < и > //
  }
}
```

4.3 Kirchoffs Theorem

Возьмём матрицу смежности графа G, заменим каждый элемент этой матрицы на противоположный, а на диагонали вместо элемента $A_{i,i}$ поставим степень вершины i (если имеются кратные рёбра, то в степени вершины они учитываются со своей кратностью). Тогда, согласно матричной теореме Кирхгофа, все алгебраические дополнения этой матрицы равны между собой, и равны количеству остовных деревьев этого графа. Например, можно удалить последнюю строку и последний столбец этой матрицы, и модуль её определителя будет равен искомому количеству.

4.4 Tuttes Theorem

Матрицей Татта называется следующая матрица $n \times n$: $A_{ij} (1 \leq i < j \leq n)$ – это либо независимая переменная, соответствующая ребру между вершинами i и j, либо тождественный ноль, если ребра между этими вершинами нет. $A_{ji} = -A_{ij}$.

В графе G существует совершенное паросочетание тогда и только тогда, когда $\det(A) \neq 0$. На практике подставляем случайные числа, делаем несколько итераций.

4.5 Matching Duals

В произвольном двудольном графе мощность максимального паросочетания равна мощности минимального вершинного покрытия.

Дополнение минимального вершинного покрытия является максимальным независимым множеством.

Алгоритм построения минимального вершинного покрытия:

1. Построить максимальное паросочетание.
2. Ориентировать ребра:
   - Из паросочетания – из правой доли в левую.
   - Не из паросочетания – из левой доли в правую.
3. Запустить обход в глубину из всех свободных вершин левой доли, построить множества L^+, L^-, R^+, R^-.
4. В качестве результата взять $L^- \cup R^+$.

4.6 LCA

```
int dfs_time = 0;
vector<int> g[N];
int d[N], tin[N], tout[N];
int to[N][LOGN];
void lca_dfs(int v, int p) {
  tin[v] = dfs_time++;
  to[v][0] = p;
  for (int i = 1; i < LOGN; ++i)
    to[v][i] = to[to[v][i - 1]][i - 1];
  for (int u : g[v]) {
    if (u == p) continue;
    d[u] = d[v] + 1;
    lca_dfs(u, v);
  }
  tout[v] = dfs_time;}
bool isParent(int v, int u) { // v isParent u
  return tin[v] <= tin[u] && tout[v] >= tout[u];}
int lca(int v, int u) {
  if (isParent(v, u)) return v;
  if (isParent(u, v)) return u;
  for (int i = LOGN - 1; i >= 0; --i)
    if (!isParent(to[v][i], u))
      v = to[v][i];
  return to[v][0];}
```

4.7 Kuhn

```
// n1 - number of vertices of the first part, n2 - second part
// switch n1, n2 if n1 > n2!
vector<int> used(n1);
vector<vector<int>> g(n1);
vector<int> to(n2, -1); //-1 if no matching edge comes out of i
bool kuhn(int v){
 if(used[v]) return false;
 used[v] = 1;
 for (int u : g[v]){
  if (to[u] == -1){
   to[u] = v;
   return true;
  }}
 for (int u : g[v]){
  if (kuhn(to[u])){
   to[u] = v;
   return true;
  }}
 return false;}
//=====================
forn(i, n1){
 if (kuhn(i)){
  //...
  used.assign(n,0);}
}
```

4.8 1-K bfs

```
void bfs(int s, int k){
 vector<queue<int>> q(k);
 vector<int> d(n, INF);
 q[0].push(s);
 d[s] = 0;
 int cnt = 1, pos = 0;
 while (cnt > 0){
   while(q[pos].empty()) pos = (pos + 1) % k;
   int v = q[pos].front();
   q[pos].pop();
   --cnt;
   for (auto e: g[v]){
    int u = e.u, w = e.weight; // 1 <= w <= k
    if (d[u] > d[v] + w){
     d[u] = d[v] + w;
     q[w].push(u);
     ++cnt;
}}}}
```

4.9 Floyd-Warshall

```
vector<vector<int>> d(n, vector<int>(n));
forn(i,n) d[i][i] = 0;
forn(k,n) forn(i,n) forn(j, n)
      if (d[i][k] < INF && d[k][j] < INF)
        d[i][j] = min (d[i][j], d[i][k] + d[k][j]);
```

4.10 Ford-Bellman with negative cycle

```
vector<int> d (n, INF), p(n, -1);
d[v] = 0;
int x;
forn(i, n){
  x = -1;
  forn(j, m)
    if (d[e[j].v] < INF && d[e[j].u] > d[e[j].v] + e[j].cost){
        d[e[j].u] = max (-INF, d[e[j].v] + e[j].cost);
        p[e[j].u] = e[j].v;
        x = e[j].u;
  }}
if (x != -1) { // have negative cycle
  int y = x;
  for (int i = 0; i < n; ++i) y = p[y];
  vector<int> path; // path = negative cycle
  for (int cur=y; ; cur=p[cur]) {
    path.push_back (cur);
    if (cur == y && path.size() > 1) break;
  }
  reverse (path.begin(), path.end());
}
```

4.11 Eulerian path

Эйлеров цикл существует тогда и только тогда, когда степени всех вершин чётны. Эйлеров путь существует тогда и только тогда, когда количество вершин с нечётными степенями равно двум (или нулю). Граф должен быть достаточно связным (если удалить из него все изолированные вершины, то должен получиться связный граф).

Если нечетных вершин ровно две, то соединим их ребром, построим эйлеров цикл, затем удалим это ребро из цикла. Если правильно сдвинуть этот цикл, получим эйлеров путь.

```
set<int> g[N]
void euler(int v) {
  while (!g[v].empty()) {
    int u = *g[v].begin();
    g[v].erase(u);
    g[u].erase(v);
    euler(u);}
  ans.push_back(v + 1);}
```

4.12 Boruvka

Алгоритм Борувки опирается на этот факт и заключается в следующем:

1. Для каждой вершины найдем минимальное инцидентное ей ребро.
2. Добавим все такие рёбра в остов и сожмем получившиеся компоненты, то есть объединим списки смежности вершин, которые эти рёбра соединяют.
3. Повторяем шаги 1-2, пока в графе не останется только одна вершина-компонента.

Алгоритм может работать неправильно, если в графе есть ребра, равные по весу. Пример: «треугольник» с одинаковыми весами рёбер. Избежать такую ситуацию можно, введя какой-то дополнительный порядок на рёбрах —например, сравнивая пары из веса и номера ребра.

Заметим, что на каждой итерации каждая оставшаяся вершина будет задействована в «мердже». Это значит, что количество вершин-компонент уменьшится хотя бы вдвое, а значит всего итераций будет не более $O(logn)$. Итоговая сложность: $O(mlogn)$

Алгоритм полезен на неявных графах, если мы можем быстро находить минимальное ребро вершины.

5 Flows

5.1 Dinics Algorithm

```
struct edge {
  int to, cap, f;
};
int s, t;
vector<edge>ed;
vector<vector<int>>g;
vector<int>d, lst;

void add_edge(int x, int y, int cap) {
  g[x].pb((int)ed.size());
  ed.pb({ y, cap, 0 });
  g[y].pb((int)ed.size());
  ed.pb({ x, 0, 0 });
}
int res(int i) {
  return ed[i].cap - ed[i].f;
}

bool bfs() {
  d.assign(sz(g), INF);
  d[s] = 0;
  queue<int> q;
  q.push(s);
  while (!q.empty()) {
    int v = q.front();
    q.pop();
    for (int e : g[v]) {
      if (res(e) == 0) continue;
      int u = ed[e].to;
      if (d[u] > d[v] + 1) {
        d[u] = d[v] + 1;
        q.push(u);
      }
    }
  }
  return d[t] < INF;
}

int dfs(int v, int mf) {
  if (v == t) return mf;
  int sum = 0;
  for (; lst[v] < g[v].size(); lst[v]++) {
    int e = g[v][lst[v]];
    int u = ed[e].to;
    if (res(e) == 0 || d[u] != d[v] + 1) continue;
    int push = dfs(u, min(mf - sum, res(e)));
    sum += push;
    ed[e].f += push;
    ed[e ^ 1].f -= push;
    if (sum == mf) break;
  }
  return sum;
}

int dinic() {
  int flow = 0;
  while (true)
  {
    if (!bfs()) break;
    lst.assign(sz(g), 0);
    while (true)
    {
      int f = dfs(s, INF);
      if (!f) break;
      flow += f;
    }
  }
  return flow;
}
```

5.2 MCMF

```
struct edge {
  int to, cap, f, cost;
};
int s, t;
vector<edge>ed;
vector<vector<int>>g;
vector<int>p, pe;
vector<int>d;

void add_edge(int x, int y, int cap, int cost) {
  g[x].pb((int)ed.size());
  ed.pb({ y, cap, 0, cost });
  g[y].pb((int)ed.size());
  ed.pb({ x, 0, 0, -cost });
}
int res(int i) {
  return ed[i].cap - ed[i].f;
}

bool spfa() {
  p.assign(sz(g), -1);
  pe.assign(sz(g), -1);
  queue<int>q;
  vector<bool>inq(sz(g));
  q.push(s);
  d[s] = 0;
  inq[s] = true;
  while (!q.empty()) {
    int x = q.front();
    q.pop();
    inq[x] = false;
    for (auto e : g[x]) {
      int y = ed[e].to;
      int w = ed[e].cost;
      if (!res(e)) continue;
      if (d[x] + w >= d[y]) continue;
      d[y] = d[x] + w;
      p[y] = x;
      pe[y] = e;
      if (!inq[y]) q.push(y), inq[y] = true;
    }
  }
  return (p[t] != -1);
}

int augment() {
  int mf = INF;
  int cur = t;
  while (cur != s) {
    mf = min(mf, res(pe[cur]));
    cur = p[cur];
  }
  cur = t;
  while (cur != s) {
    int i = pe[cur];
    ed[i].f += mf;
    ed[i ^ 1].f -= mf;
    cur = p[cur];
  }
  return mf;
}

int min_cost() {
```

```
  int flow = 0;
  while (spfa()) {
    d.assign(sz(g), INF);
    if (!spfa()) break;
    flow += augment();
  }
  return flow;
}
```

5.3 MCMF with Potentials

call init_dag() or init_fb() (depending on if your graph is dag or not) before running calc()

```
template<typename T, typename C> struct mincost {
  const C MAX_COST = numeric_limits<C>::max();

  struct edge {
    int u, rev;
    T cap, flow;
    C cost;
  };
  int n, s, t;
  T flow;
  C cost;

  vector<edge*> p;
  vector<C> d, add;
  vector<bool> inq;
  vector<vector<edge>> g;

  mincost() {}
  mincost(int n, int s, int t) : n(n), s(s), t(t) {
    g.resize(n);
    d.resize(n);
    add.resize(n);
    p.resize(n);
    inq.resize(n);
    flow = 0;
    cost = 0;
  }
  void add_edge(int v, int u, T cap, C cost) {
    g[v].pb({u, sz(g[u]), cap, 0, cost});
    g[u].pb({v, sz(g[v]) - 1, 0, 0, -cost});
  }
  void init_dag() {
    fill(all(add), MAX_COST);
    vector<int> ind(n);
    queue<int> q({s});
    forn(v, n) for (auto& e : g[v]) {
      if (e.cap > e.flow)
        ++ind[e.u];
    }
    add[s] = 0;
    while (!q.empty()) {
      int v = q.front(); q.pop();
      for (auto& e : g[v]) {
        int u = e.u;
        if (e.cap > e.flow) {
          add[u] = min(add[u], add[v] + e.cost);
          --ind[u];
          if (ind[u] == 0)
            q.push(u);
        }
      }
    }
  }
  void init_fb() {
    fill(all(add), MAX_COST);
    add[s] = 0;
    forn(_, n - 1) {
      forn(v, n) for (auto& e : g[v]) {
        add[e.u] = min(add[e.u], add[v] + e.cost);
      }
    }
  }
  T push(T lim) {
    fill(all(d), MAX_COST);
    d[s] = 0;
    priority_queue<pair<C, int>> q;
    q.push({-d[s], s});
    while (!q.empty()) {
      int v = q.top().y;
      C curd = -q.top().x;
      q.pop();
      if (curd > d[v])
        continue;
      for (auto& e : g[v]) {
        int u = e.u;
        C w = e.cost + add[v] - add[u];
        if (e.cap > e.flow && d[u] > d[v] + w) {
          d[u] = d[v] + w;
          p[u] = &e;
          q.push({-d[u], u});
        }
      }
    }
    forn(v, n) {
      add[v] += d[v];
    }
    if (d[t] == MAX_COST) {
      return 0;
    }
    T cur_flow = lim;
    int v = t;
    while (v != s) {
      auto e = *p[v];
      cur_flow = min(cur_flow, e.cap - e.flow);
      v = g[v][e.rev].u;
    }
    v = t;
    while (v != s) {
      auto e = *p[v];
      p[v]->flow += cur_flow;
      g[v][e.rev].flow -= cur_flow;
      v = g[v][e.rev].u;
    }
    return cur_flow;
  }
  void calc(T k = numeric_limits<T>::max()) {
    T add_flow = 0;
    while ((add_flow = push(k - flow)) > 0) {
      flow += add_flow;
      cost += (add[t] - add[s]) * add_flow;
    }
  }
};
```

5.4 L-R Flow

Добавляем новый исток и сток, старый исток и сток назовём S и T, новые — S' и T'. Если ищем именно поток, а не циркуляцию, то добавляем ребро $T \to S$, $cap = \infty$. Каждое ребро вида $x \to y$ с ограничениями $[l, r]$ делим на три ребра:

1. $x \to y, cap = r - l$;
2. $S' \to y, cap = l$;
3. $x \to T', cap = l$.

Как восстанавливать сертификат? Поток по такому ребру — это суммарный поток по первым двум рёбрам из разделения. При этом ответа нет, если в полученной сети макс. поток меньше суммы l в ограничениях.

5.5 Stoer Wagner Algorithm

Находит мин разрез в неориентированном взвешенном графе. Работает за $O(nm)$.

```
int w[kN][kN], g[kN], del[kN], v[kN];
void AddEdge(int x, int y, int c) {
  w[x][y] += c;
  w[y][x] += c;
}
pair<int, int> Phase(int n) {
  fill(v, v + n, 0), fill(g, g + n, 0);
  int s = -1, t = -1;
  while (true) {
    int c = -1;
    for (int i = 0; i < n; ++i) {
      if (del[i] || v[i]) continue;
      if (c == -1 || g[i] > g[c]) c = i;
    }
    if (c == -1) break;
    v[c] = 1, s = t, t = c;
    for (int i = 0; i < n; ++i) {
      if (del[i] || v[i]) continue;
      g[i] += w[c][i];
    }
  }
  return make_pair(s, t);
}
int GlobalMinCut(int n) {
  int cut = kInf;
  fill(del, 0, sizeof(del));
  for (int i = 0; i < n - 1; ++i) {
    int s, t; tie(s, t) = Phase(n);
```

```
    del[t] = 1, cut = min(cut, g[t]);
    for (int j = 0; j < n; ++j) {
      w[s][j] += w[t][j];
      w[j][s] += w[j][t];
    }
  }
  return cut;
}
```

6 Data Structures

6.1 Centroid Decomposition

```
int h[N], pcd[N];
int dfs(int v, int s, int &cd, int p = -1){
  int sum = 1;
  for (auto it : g[v]) if (h[it.u] == -1 && it.u != p)
    sum += dfs(it.u, s, cd, v);
  if (cd == -1 && (2 * sum >= s || p == -1))
    cd = v;
  return sum;
}
void build(int v, int s, int d, int p = -1){
  int cd = -1;
  dfs(v, s, cd);
  h[cd] = d;
  pcd[cd] = p;
  for (auto it : g[cd]) if (h[it.u] == -1)
    build(it.u, s / 2, d + 1, cd);
}
void solve() {
  memset(h, -1, sizeof(h));
  build(0, n, 0);
}
```

6.2 HLD

```
int n;
vector<int> g[N];
int p[N], siz[N], d[N], nxt[N];
int tin[N], T;
void dfs_sz(int v) {
  if (p[v] != -1) {
    auto it = find(g[v].begin(), g[v].end(), p[v]);
    if (it != g[v].end())
      g[v].erase(it);
  }
  siz[v] = 1;
  for (int &u : g[v]) {
    p[u] = v;
    d[u] = d[v] + 1;
    dfs_sz(u);
    siz[v] += siz[u];
    if (siz[u] > siz[g[v][0]])
      swap(u, g[v][0]);
  }
}
void dfs_hld(int v) {
  tin[v] = T++;
  for (int u : g[v]) {
    nxt[u] = (u == g[v][0] ? nxt[v] : u);
    dfs_hld(u);
  }
}
void update(int l, int r, int val) { } // [l; r] inclusive
int get(int l, int r) { } // [l; r] inclusive
void update_path(int v, int u, int val) {
  for (; nxt[v] != nxt[u]; u = p[nxt[u]]) {
    if (d[nxt[v]] > d[nxt[u]]) swap(v, u);
    update(tin[nxt[u]], tin[u], val);
  }
  if (d[v] > d[u]) swap(v, u);
  update(tin[v], tin[u], val);
}
int get_path(int v, int u) {
  int res;
  for (; nxt[v] != nxt[u]; u = p[nxt[u]]) {
    if (d[nxt[v]] > d[nxt[u]]) swap(v, u);
    // update res with the result of get()
    get(tin[nxt[u]], tin[u]);
  }
  if (d[v] > d[u]) swap(v, u);
  get(tin[v], tin[u]);
  return res;
}
```

```
void update_subtree(int v, int val) {
  update(tin[v], tin[v] + siz[v] - 1, val);
}
int get_subtree(int v) {
  return get(tin[v], tin[v] + siz[v] - 1);
}
void init_hld(int root = 0) {
  d[root] = 0;
  nxt[root] = root;
  p[root] = -1;
  T = 0;
  dfs_sz(root);
  dfs_hld(root);
}
```

6.3 Explicit Treap

```
struct node {
  int y, val, cnt = 0, rev = 0;
  node* l = 0;
  node* r = 0;
  node(int val) {
    this->val = val;
    y = mt();
    cnt = 1;
  }
  node() {}
};
using treap = node*;

const int N = 2e5 + 10;
node buf[N];
int siz = 0;
treap new_node(int val) {
  buf[siz] = node(val);
  return &buf[siz++];
}
node get(treap t) {
  if (!t) return node();
  return *t;
}
treap fix(treap t) {
  if (!t) return 0;
  t->cnt = get(t->l).cnt + get(t->r).cnt + 1;
  return t;
}
treap push(treap t) {
  if (!t) return 0;
  if (t->rev) {
    swap(t->l, t->r);
    if (t->l) t->l->rev ^= 1;
    if (t->r) t->r->rev ^= 1;
    t->rev = 0;
  }
  return t;
}
treap merge(treap a, treap b) {
  if (!a) return b;
  if (!b) return a;
  a = push(a); b = push(b);
  if (a->y > b->y) {
    a->r = merge(a->r, b);
    return fix(a);
  }
  else {
    b->l = merge(a, b->l);
    return fix(b);
  }
}
pair<treap, treap> split(treap t, int k) {
  if (!t) return { 0, 0 };
  t = push(t);
  int cntl = get(t->l).cnt;
  if (cntl < k) {
    auto p = split(t->r, k - cntl - 1);
    t->r = p.first;
    return { fix(t), p.second };
  }
  else {
    auto p = split(t->l, k);
    t->l = p.second;
    return { p.first, fix(t) };
  }
}
```

6.4 Sparce Table

```
int lg[MAXN+1];
lg[1] = 0;
for (int i = 2; i <= MAXN; i++)
  lg[i] = lg[i/2] + 1;

int st[K + 1][MAXN];
copy(array.begin(), array.end(), st[0]);
for (int i = 1; i <= K; i++)
  for (int j = 0; j + (1 << i) <= N; j++)
    st[i][j] = f(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);

int i = lg[R - L + 1];
int minimum = min(st[i][L], st[i][R - (1 << i) + 1]);
```

6.5 Persistent Segment Tree

```
struct Segtree {
  int lb, rb;
  int s = 0;
  Segtree *l = 0, *r = 0;
  Segtree(int lb, int rb) : lb(lb), rb(rb) {
    if (lb != rb) {
      int t = (lb + rb) / 2;
      l = new Segtree(lb, t);
      r = new Segtree(t, rb);
    }
  }
  void copy() {
    if (l) {
      l = new Segtree(*l);
      r = new Segtree(*r);
    }
  }
  void add(int k, int x) {
    copy();
    s += x;
    if (l) {
      if (k < l->rb)
        l->add(k, x);
      else
        r->add(k, x);
    }
  }
  int sum(int lq, int rq) {
    // этот метод ничего не меняет -- он и так хороший
    if (lq <= lb && rb <= rq)
      return s;
    if (max(lb, lq) >= min(rb, rq))
      return 0;
    return l->sum(lq, rq) + r->sum(lq, rq);
  }
};

// Теперь осталось только создать список версий, и после каждой
      операции копировать туда новый корень:

vector<Segtree*> roots;
roots.push_back(new Segtree(0, n));
void add(int k, int x, int v) {
  Segtree *root = new Segtree(*roots[v]);
  root->add(k, x);
  roots.push_back(root);
}
```

6.6 Persistent Treap

```
mt19937 rnd(42);
struct node {
  node* l, * r;
  int siz, val;
  node() : l(NULL), r(NULL), val(0), siz(0) {}
  node(int c) : l(NULL), r(NULL), val(c), siz(1) {}
};

typedef node* treap;
typedef pair<treap, treap> ptt;

int getsiz(treap t) {
  return t ? t->siz : 0;
}

treap create(int val) {
  //можно заменить на буфер
  return new node(val);
}
treap create() {
  return new node();
}
treap ncopy(treap t) {
  if (!t) return t;
  treap nw = create();
  nw->l = t->l;
  nw->r = t->r;
  nw->val = t->val;
  nw->siz = t->siz;
  return nw;
}
treap fix(treap t) {
  if (!t) return t;
  t->siz = getsiz(t->l) + getsiz(t->r) + 1;
  return t;
}

treap merge(treap a, treap b) {
  a = fix(a);
  b = fix(b);
  if (!a || !b) return a ? ncopy(a) : ncopy(b);
  if (int(rnd() % (getsiz(a) + getsiz(b))) >= getsiz(a)) {
    treap nw = ncopy(b);
    nw->l = merge(a, b->l);
    return fix(nw);
  }
  else {
    treap nw = ncopy(a);
    nw->r = merge(a->r, b);
    return fix(nw);
  }
}

ptt split(treap t, int siz) {
  t = fix(t);
  if (!t) return ptt(NULL, NULL);
  if (getsiz(t->l) >= siz) {
    ptt p = split(t->l, siz);
    treap nw = ncopy(t);
    nw->l = p.second;
    return ptt(p.first, fix(nw));
  }
  else {
    ptt p = split(t->r, siz - getsiz(t->l) - 1);
    treap nw = ncopy(t);
    nw->r = p.first;
    return ptt(fix(nw), p.second);
  }
}

void dfs(treap t, vector<int>& a) {
  if (t->l) dfs(t->l, a);
  a.push_back(t->val);
  if (t->r) dfs(t->r, a);
}

treap build(int l, int r, const vector<int>& a) {
  if (l == r)
    return NULL;
  if (l == r - 1)
    return new node(a[l]);
  int m = (l + r) / 2;
  treap t = new node(a[m]);
  t->l = build(l, m, a);
  t->r = build(m + 1, r, a);
  return fix(t);
}

//нужно делать раз в несколько тысяч операций.
treap rebuild(treap t) {
  vector<int> a;
  dfs(t, a);
  //здесь можно обнулить buf_size
  return build(0, a.size(), a);
}
```

6.7 Ordered Set

```
#include "ext/pb_ds/assoc_container.hpp"
using namespace __gnu_pbds;

gp_hash_table<ll, int> h({},{},{},{}, {1 << 16});
template <typename T> using ordered_set = tree<T, null_type,
     less<T>, rb_tree_tag, tree_order_statistics_node_update>;
template <typename K, typename V> using ordered_map = tree<K, V
     , less<K>, rb_tree_tag, tree_order_statistics_node_update
     >;

// HOW TO USE ::
```

```
// -- order_of_key(10) returns the number of elements in set/
    map strictly less than 10
// -- *find_by_order(10) returns 10-th smallest element in set/
    map (0-based)
```

7 Strings

7.1 String Matching with FFT

Нам даны две строки, текст T и шаблон P, состоящие из строчных букв. Мы должны найти все вхождения шаблона в текст. Шаблон может содержать символ джокера *, который соответствует любому символу.

1. Преобразуйте текст T и шаблон P в полиномы: Для строки текста:

$$A(x) = a_0 x^0 + a_1 x^1 + \cdots + a_{n-1} x^{n-1}$$

где $a_i = \cos(\alpha_i) + i \sin(\alpha_i), \quad \alpha_i = \frac{2\pi T[i]}{26}$

Для шаблона:

$$B(x) = b_0 x^0 + b_1 x^1 + \cdots + b_{m-1} x^{m-1}$$

где $b_i = \cos(\beta_i) - i \sin(\beta_i), \quad \beta_i = \frac{2\pi P[m-i-1]}{26}$

(шаблон разворачивается для сопоставления).

2. Умножьте полиномы с помощью FFT, получив произведение:

$$C(x) = A(x) \cdot B(x)$$

Коэффициент c_{m-1+i} укажет, есть ли совпадение шаблона на позиции i. 3. Если в шаблоне есть джокеры (символы $*$), установите для них соответствующие коэффициенты полинома $b_i = 0$.

4. Совпадение шаблона с текстом на позиции i происходит, если:

$$c_{m-1+i} = m - x$$

где x —количество джокеров в шаблоне.

Таким образом, вы сможете искать строки и шаблоны с джокерами через умножение полиномов с FFT.

7.2 Suffix Array

Don't forget to modify the string to avoid cyclic comparisons if needed

```
struct suffix_array {
  vector<int> c, pos;
  vector<pair<pt, int>> p, nw;
  vector<int> cnt;
  int n;

  void radix_sort(int max_al) {
    cnt.assign(max_al, 0);
    forn(i, n) ++cnt[p[i].x.y];
    fore(i, 1, max_al) cnt[i] += cnt[i - 1];
    nw.resize(n);
    forn(i, n) nw[--cnt[p[i].x.y]] = p[i];
    cnt.assign(max_al, 0);
    forn(i, n) ++cnt[nw[i].x.x];
    fore(i, 1, max_al) cnt[i] += cnt[i - 1];
    for (int i = n - 1; i >= 0; --i) p[--cnt[nw[i].x.x]] = nw[i
        ];
  }

  vector<int> lcp;
  sparse_table st;

  int get_lcp(int l, int r) {
    l = c[l], r = c[r];
    if (l > r) swap(l, r);
    return st.get(l, r);
  }

  suffix_array(const string &s) {
    n = sz(s);
    c = vector<int>(all(s));
    int max_al = *max_element(all(c)) + 1;
    p.resize(n);
    for (int k = 1; k < n; k <<= 1) {
      for (int i = 0, j = k; i < n; ++i, j = (j + 1 == n ? 0 :
          j + 1))
        p[i] = mp(mp(c[i], c[j]), i);
      radix_sort(max_al);
      c[p[0].y] = 0;
      fore(i, 1, n) c[p[i].y] = c[p[i - 1].y] + (p[i].x != p[i
          - 1].x);
      max_al = c[p.back().y] + 1;
    }
    lcp.resize(n);
    int l = 0;
    forn(i, n) {
      l = max(0, l - 1);
      if (c[i] == n - 1)
        continue;
      while (i + l < n && p[c[i] + 1].y + l < n && s[i + l] ==
          s[p[c[i] + 1].y + l])
        ++l;
      lcp[c[i]] = l;
    }
    pos.resize(n);
    forn(i, n)
      pos[i] = p[i].y;
    st = sparse_table(lcp);
  }
};
```

7.3 Prefix- and z-funtion

```
vector<int> pref_func(string s) {
  int n = s.size();
  vector<int> a(n);
  for (int i = 1; i < n; i++) {
    int j = a[i - 1];
    while (j > 0 && s[i] != s[j])
      j = a[j - 1];
    if (s[i] == s[j]) {
      j++;
      a[i] = j;
    }
  }
  return a;
}

vector<int> z_func(string s) {
  int n = s.size();
  vector<int> z(n);
  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 and s[z[i]] == s[i + z[i]])
      ++z[i];
    if (i + z[i] > r)
      l = i, r = i + z[i] - 1;
  }
  return z;
}
```

7.4 Manachers Algorithm

```
vector<int> d1(n), d2(n);
for (int i = 0, l = 0, r = -1; i < n; i++) {
  int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
  while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
    k++;
  }
  d1[i] = k--;
  if (i + k > r) {
    l = i - k;
    r = i + k;
  }
}
for (int i = 0, l = 0, r = -1; i < n; i++) {
  int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
  while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k
      ]) {
    k++;
  }
  d2[i] = k--;
  if (i + k > r) {
    l = i - k - 1;
    r = i + k ;
  }
}
```

7.5 Prefix Automaton

```
void compute_automaton(string s, vector<vector<int>>& aut) {
  s += '#';
  int n = s.size();
  vector<int> pi = prefix_function(s);
  aut.assign(n, vector<int>(26));
  for (int i = 0; i < n; i++) {
    for (int c = 0; c < 26; c++) {
      if (i > 0 && 'a' + c != s[i])
        aut[i][c] = aut[pi[i-1]][c];
      else
        aut[i][c] = i + ('a' + c == s[i]);
```

```
    }
  }
}
```

7.6 Aho-Corasick + Trie

```
const int B = 2e5 + 10;
int sz = 0;
map<char, int> nxt[B], go[B];
int cnt[B], pr[B], slink[B];
char prc[B];
int dlink[B];

int create() {
  dlink[sz] = -1;
  slink[sz] = -1;
  pr[sz] = -1;
  cnt[sz] = 0;
  nxt[sz].clear();
  return sz++;
}

void add(string s, int i, int k = 1) {
  int v = 0;
  for (auto x : s) {
    if (!nxt[v].count(x))
      nxt[v][x] = create();
    pr[nxt[v][x]] = v;
    prc[nxt[v][x]] = x;
    v = nxt[v][x];
  }
  cnt[v] += k;
}

int get_slink(int x);

int get_go(int x, char k) {
  if (go[x].count(k)) return go[x][k];
  if (nxt[x].count(k)) return go[x][k] = nxt[x][k];
  if (!x) return go[x][k] = 0;
  return go[x][k] = get_go(get_slink(x), k);
}

int get_slink(int x) {
  if (slink[x] != -1) return slink[x];
  if (x == 0 or pr[x] == 0) return slink[x] = 0;
  return slink[x] = get_go(get_slink(pr[x]), prc[x]);
}

int get_dlink(int x) {
  if (dlink[x] != -1) return dlink[x];
  if (!x) return dlink[x] = 0;
  if (cnt[get_slink(x)]) return dlink[x] = get_slink(x);
  return dlink[x] = get_dlink(get_slink(x));
}
```

7.7 Hash String

```
const int K = 2;
using my_hash = array<int, K>;
mt19937 rnd(time(0));
my_hash A, M;
bool is_prime(int x) {
  for (int i = 2; i * 1ll * i <= x; i++)
    if (x % i == 0) return false;
  return true;
}
int next_prime(int x) {
  while (!is_prime(x)) ++x;
  return x;
}
int random_prime(int l, int r) {
  return next_prime(uniform_int_distribution<int>(l, r)(rnd));
}
void prepare_primes() {
  forn(i, K) A[i] = random_prime(30, 200), M[i] = random_prime
      (1e9, 1e9 + 1e5);
}
// operator +, -, *, inv(), zero(), one()

my_hash get_hash(char c) {
  my_hash res; forn(i, K) res[i] = (c - 'a') + 1;
  return res;
}
my_hash get_hash(const string& s) {
  my_hash res = zero(); forrn(i, sz(s)) res = (res * A) +
      get_hash(s[i]);
  return res;
}
```

```
struct pref_hash {
  vector<my_hash> p;
  vector<my_hash> inv_a;
  pref_hash(string s = "") {
    int n = sz(s) + 1;
    p.assign(n + 1, zero());
    inv_a.assign(n + 1, one());
    my_hash INV_A = inv(A);
    my_hash cur = one();
    forn(i, n) {
      p[i + 1] = p[i] + get_hash(s[i]) * cur;
      inv_a[i + 1] = inv_a[i] * INV_A;
      cur = cur * A;
    }
  }
  my_hash substr(int l, int r) { return (p[r] - p[l]) * inv_a[l
      ]; }
};
vector<my_hash> deg_a;
void prepare_degrees(int n) {
  deg_a.assign(n + 1, one());
  forn(i, n) deg_a[i + 1] = deg_a[i] * A;
}
my_hash concat(my_hash l, my_hash r, int len) { return l + r *
     deg_a[len]; }
```

8 DP

8.1 Dynamic CHT

Хранит линейные функции и умеет узнавать мин/макс значение от всех функций в точке. $O(logN)$

```
struct Line {
  li k, m;
  mutable li p;
  bool operator<(const Line& o) const {
    return k < o.k;
  }
  bool operator<(const li&x) const {
    return p < x;
  }
};
template<bool GET_MAX = true>
struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  const li inf = numeric_limits<li>::max();
  li div(li a, li b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b);
  }
  bool isect(iterator x, iterator y) {
    if (y == end()) { x->p = inf; return false; }
    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(li k, li m) {
    if(!GET_MAX) k = -k, m = -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));
  }
  li query(li x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return (l.k * x + l.m) * (GET_MAX ? 1 : -1);
  }
};
```

8.2 Li Chao Tree

Хранит функции, которые могут пересекаться друг с другом максимум 1 раз (напр. прямые, параболы). Узнает мин/макс значение от всех функции в точке. $O(logN)$.

```
template<typename T>
struct lichao_tree {
  int n;
  vector<pair<T, T>> lines;
  lichao_tree(int n): n(n) {
    lines.resize(4 * n, {0, -numeric_limits<T>::max()});
  }
  inline T eval(pair<T, T> line, int x) {
    return line.x * x + line.y;
  }
```

```
  void add_line(int v, int l, int r, pair<T, T> cur) {
    int m = (l + r) >> 1;
    bool lft = eval(cur, l) > eval(lines[v], l);
    bool mid = eval(cur, m) > eval(lines[v], m);
    if (mid) swap(lines[v], cur);
    if (l + 1 == r) return;
    else if (lft != mid) add_line(v * 2 + 1, l, m, cur);
    else add_line(v * 2 + 2, m, r, cur);
  }
  void add_line(pair<T, T> cur) {
    add_line(0, 0, n, cur);
  }
  T get(int v, int l, int r, int x) {
    T res = eval(lines[v], x);
    if (l + 1 == r) return res;
    int m = (l + r) >> 1;
    if (x < m) return max(res, get(v * 2 + 1, l, m, x));
    else return max(res, get(v * 2 + 2, m, r, x));
  }
  T get(int x) {
    return get(0, 0, n, x);
  }
};
```

8.3 D&C Optimization

$dp[i][j] = \min_{0 \le k \le j} dp[i-1][k-1] + w(k,j)$.

Должно выполнятся условие $C(a,c) + C(b,d) \le C(a,d) + C(b,c)$ для всех $a \le b \le c \le d$

```
int m, n;
vector<long long> dp_before(n), dp_cur(n);
long long C(int i, int j);

// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr) {
  if (l > r)
    return;
  int mid = (l + r) >> 1;
  pair<long long, int> best = {LLONG_MAX, -1};
  for (int k = optl; k <= min(mid, optr); k++) {
    best = min(best, {(k ? dp_before[k - 1] : 0) + C(k, mid), k
        });
  }
  dp_cur[mid] = best.first;
  int opt = best.second;
  compute(l, mid - 1, optl, opt);
  compute(mid + 1, r, opt, optr);
}

int solve() {
  for (int i = 0; i < n; i++)
    dp_before[i] = C(0, i);
  for (int i = 1; i < m; i++) {
    compute(0, n - 1, 0, n - 1);
    dp_before = dp_cur;
  }
  return dp_before[n - 1];
}
```

8.4 Knuth Optimization

$dp[i][j] = \min_{i \le k \le j} [dp[i][k] + dp[k+1][j] + C(i,j)]$

Достаточное условие: 1. $C_{ac} + C_{bd} \le C_{ad} + C_{bc}$, 2. $C_{bc} \le C_{ad}$, при всех $a \le b \le c \le d$.

```
li calc(int l, int r) {
  dp[l][r] = INF64;
  fore(m, opt[l][r - 1], opt[l + 1][r] + 1) {
    li nres = calc(l, m) + calc(m, r) + C(l, r);
    if (dp[l][r] > nres) {
      dp[l][r] = nres;
      opt[l][r] = m;
    }
  }
  return dp[l][r];
}
```

8.5 SOS DP

or-свертка: $c_i = \sum_{j,k:\, j|k=i} a_j \cdot b_k$. Нужно взять сумму по подмножествам пос-тей a и b, перемножить поэлементно, а потом обратить суммы по подмножествам для этого произведения.

and-свертка: $c_i = \sum_{j,k:\, j\&k=i} a_j \cdot b_k$. Нужно взять сумму по **над**множествам пос-тей a и b, перемножить поэлементно, а потом обратить суммы по **над**множествам для этого произведения.

```
forn(i, LOGN) forn(j, N) if ((j & (1 << i)) == 0)
    dp[j | (1 << i)] += dp[j];
```

8.6 Submasks

```
for (int m=0; m<(1<<n); ++m)
  for (int s=m; s; s=(s-1)&m)
    // ... usage s and m ...
```

8.7 LIS binary search

$dp[i]$ - это число, на которое оканчивается возрастающая последовательность длины i (если таких несколько, то наименьшее из них).

Изначально полагаем $dp[0] = -\inf$, $dp[i] = \inf$.

По такой динамике тоже можно восстановить ответ, для чего надо хранить массив "предков" $p[i]$ —то, на элементе с каким индексом оканчивается оптимальная подпоследовательность длины i. Кроме того, для каждого элемента массива $a[i]$ надо будет хранить его "предка" —т.е. индекс того элемента, который должен стоять перед $a[i]$ в оптимальной подпоследовательности.

```
vector<int>dp(n + 1, INF);
dp[0] = -INF;
for (int i = 0; i < n; i++) {
  int j = upper_bound(dp.begin(), dp.end(), a[i]) - dp.begin();
  if (dp[j - 1] < a[i] && a[i] < dp[j])
    dp[j] = a[i];
}
```

9 Geometry

9.1 Base

```
//ll or ld !!!
struct vec {
  ll x, y;
  vec operator+(const vec& p) const { return { x + p.x, y + p.y
       }; }
  vec operator-(const vec& p) const { return { x - p.x, y - p.y
       }; }
  ll operator*(const vec& p) const { return x * p.x + y * p.y;
    }
  ll operator%(const vec& p) const { return x * p.y - y * p.x;
    }

  bool operator<(const vec& p) const {
    return x < p.x || (x == p.x && y < p.y);
  }
};

struct line {
  ll a, b, c;
  line() {};
  line(const vec& p, const vec& q) {
    a = p.y - q.y;
    b = q.x - p.x;
    c = -(a * p.x + b * p.y);
  }
  line(ll a, ll b, ll c) : a(a), b(b), c(c) {}
};

struct circle : vec {
  ll r;
};
```

9.2 Intersections

```
//Пересечение отрезков
bool intersec(vec a, vec b, vec c, vec d) {
  if (sign((d - c) % (a - c)) == sign((d - c) % (b - c)))
    return false;
  if (sign((b - a) % (c - a)) == sign((b - a) % (d - a)))
    return false;
  return true;
}
ll det(ll a, ll b, ll c, ll d) {
  return a * d - b * c;
}
//Пересечение прямых
bool intersec(const line& l1, const line& l2, vec& p) {
  ll D = det(l1.a, l1.a, l2.b, l2.b);
  if (D == 0) return false;
  ll Dx = det(l1.c, l1.b, l2.c, l2.b);
  ll Dy = det(l1.a, l1.c, l2.a, l2.c);
  p = { -Dx / D, -Dy / D };
  return true;
}
```

9.3 Angles

```
ld angle(vec a) { //Полярный угол точки [0, 2pi)
  ld ang = atan2l(a.y, a.x);
  if (ang < 0) ang += 2 * PI;
  return ang;
}
ld angle(vec a, vec b) { //Угол между векторами [0, pi]
  return acosl((a * b) / sqrtl(a * a) / sqrtl(b * b));
}
```

9.4 Areas

```
ld polygon_area(vector<vec>a) { //Площадь многоугольника
  ll res = 0;
  vec last = a.back();
  for (auto cur : a) {
    res += (cur.y + last.y) * (cur.x - last.x);
    last = cur;
  }
  return abs(res) / 2.l;
}
ld triangle_area(vec a, vec b, vec c) { //Площадь треугольника
  return abs((b - a) % (c - a)) / 2.l;
}
```

9.5 Distances

```
ld distance_to_the_line(vec p, line l) {
  return abs((l.a * p.x + l.b * p.y + l.c)) / sqrtl(l.c * l.a +
       l.b * l.b);
}
ld distance_to_the_beam(vec p, vec s, vec f) {
  if ((f - s) * (p - s) >= 0)
    return abs((p - s) % (f - s)) / dist(f - s);
  return dist(p - s);
}
ld distance_to_the_segment(vec p, vec s, vec f) {
  if ((s - f) * (p - f) >= 0 && (f - s) * (p - s) >= 0)
    return abs((p - s) % (f - s)) / dist(f - s);
  return min(dist(p - f), dist(p - s));
}
ld distance_between_the_segments(vec a, vec b, vec c, vec d) {
  if (intersec(a, b, c, d))
    return 0;
  auto dist = distance_to_the_segment;
  return min({ dist(a, c, d), dist(b, c, d), dist(c, a, b),
      dist(d, a, b) });
}
// 1 - по одну сторону относительно прямой, 0 - по разные
    стороны
bool position_of_the_points(vec p, vec q, line l) {
  return sign(l.a * p.x + l.b * p.y + l.c) == sign(l.a * q.x +
      l.b * q.y + l.c);
}
```

9.6 Circle Circle Area

```
ld circle_circle_area(vec a, int r1, vec b, int r2) {
  ld d = sqrtl(sqr(a.x - b.x) + sqr(a.y - b.y));
  if (r1 + r2 < d + EPS) return 0;
  if (r1 + d < r2 + EPS) return PI * r1 * r1;
  if (r2 + d < r1 + EPS) return PI * r2 * r2;
  ld theta_1 = acos((r1 * r1 + d * d - r2 * r2) / (2 * r1 * d))
      ,
    theta_2 = acos((r2 * r2 + d * d - r1 * r1) / (2 * r2 * d));
  return r1 * r1 * (theta_1 - sin(2 * theta_1) / 2.) + r2 * r2
      * (theta_2 - sin(2 * theta_2) / 2.);
}
```

9.7 Circle Line Intersection

```
// circle centered at (0, 0)
vector<vec> inter(const circle& c, const line& l) {
  ld x0 = -l.a * l.c / (l.a * l.a + l.b * l.b);
  ld y0 = -l.b * l.c / (l.a * l.a + l.b * l.b);
  if (l.c * l.c > c.r * c.r * (l.a * l.a + l.b * l.b) + EPS)
    return {};
  else if (abs(l.c * l.c - c.r * c.r * (l.a * l.a + l.b * l.b))
      < EPS) {
    return { {x0, y0} };
  }
  else {
    ld d = c.r * c.r - l.c * l.c / (l.a * l.a + l.b * l.b);
    ld mult = sqrt(d / (l.a * l.a + l.b * l.b));
    ld ax, ay, bx, by;
    ax = x0 + l.b * mult;
    bx = x0 - l.b * mult;
    ay = y0 - l.a * mult;
    by = y0 + l.a * mult;
    return { {ax, ay}, {bx, by} };
  }
}
```

9.8 Circle Circle Intersection

```
// circle A centered at (0, 0)
vector<vec> inter(circle a, circle b) {
  b.x -= a.x;
  b.y -= a.y;
  line l(-2 * b.x, -2 * b.y, sqr(b.x) + sqr(b.y) + sqr(a.r) -
      sqr(b.r));
  auto p = inter(a, l);
  for (auto& it : p) it.x += a.x, it.y += a.y;
  return p;
}
```

9.9 Tangent Lines of Two Circles

```
void tangents(vec c, ld r1, ld r2, vector<line>& ans) {
  ld r = r2 - r1;
  ld z = sqr(c.x) + sqr(c.y);
  ld d = z - sqr(r);
  if (d < -EPS) return;
  d = sqrtl(abs(d));
  line l;
  l.a = (c.x * r + c.y * d) / z;
  l.b = (c.y * r - c.x * d) / z;
  l.c = r1;
  ans.pb(l);
}
vector<line> tangents(circle a, circle b) {
  vector<line> ans;
  for (int i = -1; i <= 1; i += 2)
    for (int j = -1; j <= 1; j += 2)
      tangents(b - a, a.r * i, b.r * j, ans);
  forn(i, sz(ans))
    ans[i].c -= ans[i].a * a.x + ans[i].b * a.y;
  return ans;
}
```

9.10 Convex Hull

```
// (-1) - clockwise, (1) - counterclockwise, (0) - collinear
int orient(vec a, vec b, vec c) {
  return sign(a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.
      y - b.y));
}

vector<vec> convex_hull(vector<vec> a) {
  if (a.size() == 1) return;
  sort(a.begin(), a.end());
  vec p1 = a[0], p2 = a.back();
  vector<vec> up, down;
  up.push_back(p1);
  down.push_back(p1);
  for (size_t i = 1; i < a.size(); ++i) {
    if (i == a.size() - 1 || orient(p1, a[i], p2) < 0) {
      while (up.size() >= 2 && orient(up[up.size() - 2], up[up.
          size() - 1], a[i]) >= 0)
        up.pop_back();
      up.push_back(a[i]);
    }
    if (i == a.size() - 1 || orient(p1, a[i], p2) > 0) {
      while (down.size() >= 2 && orient(down[down.size() - 2],
          down[down.size() - 1], a[i]) <= 0)
        down.pop_back();
      down.push_back(a[i]);
    }
  }
  a.clear();
  forn(i, sz(up))
    a.push_back(up[i]);
  forn(i, sz(down))
    a.push_back(down[i]);
  return a;
}
```

9.11 Minkowski Sum

Рассмотрим два множества A и B точек на плоскости. Сумма Минковского $A + B$ определяется как $\{a + b \mid a \in A, b \in B\}$. Будем рассматривать случай, когда A и B состоят из выпуклых многоугольников P и Q с их внутренностями. Сумма выпуклых многоугольников P и Q является выпуклым многоугольником с не более чем $|P| + |Q|$ вершинами.

```
void reorder_polygon(vector<vec>& P) {
  auto mn = min_element(all(P), [](auto l, auto r) {
    return l.y < r.y or (l.y == r.y and l.x < r.x);
  });
  rotate(P.begin(), mn, P.end());
}

vector<vec> minkowski(vector<vec> P, vector<vec> Q) {
  reorder_polygon(P); reorder_polygon(Q);
  P.push_back(P[0]); P.push_back(P[1]);
  Q.push_back(Q[0]); Q.push_back(Q[1]);
  vector<vec> result;
  int i = 0, j = 0;
  while (i < P.size() - 2 || j < Q.size() - 2) {
    result.push_back(P[i] + Q[j]);
    auto cross = (P[i + 1] - P[i]) % (Q[j + 1] - Q[j]);
    if (cross >= 0 && i < P.size() - 2)
      ++i;
    if (cross <= 0 && j < Q.size() - 2)
      ++j;
  }
  return result;
}
```

9.12 Point in Convex Polygon

```
vector<vec> seq;
vec translation;
int n;

bool pointInTriangle(vec a, vec b, vec c, vec point) {
  ll s1 = abs((b - a) % (c - a));
  ll s2 = abs((a - point) % (b - point)) + abs((b - point) % (c
      - point)) + abs((c - point) % (a - point));
  return s1 == s2;
}

void prepare(vector<vec>& points) {
  n = points.size();
  auto mn = min_element(all(points));
  rotate(points.begin(), mn, points.end());
  n--;
  seq.resize(n);
  for (int i = 0; i < n; i++)
    seq[i] = points[i + 1] - points[0];
  translation = points[0];
}

bool pointInConvexPolygon(vec point) {
  point = point - translation;
  if (seq[0] % point and sign(seq[0] % point != sign(seq[0] %
     seq[n - 1])))
    return false;
  if (seq[n - 1] % point and sign(seq[n - 1] % point) != sign(
     seq[n - 1] % seq[0]))
    return false;
  if (seq[0] % point == 0)
    return seq[0] * seq[0] >= point * point;

  int l = 0, r = n - 1;
  while (r - l > 1) {
    int mid = (l + r) / 2;
    int pos = mid;
    if (seq[pos] % point >= 0)
      l = mid;
    else
      r = mid;
  }
  int pos = l;
  return pointInTriangle(seq[pos], seq[pos + 1], vec(0, 0),
     point);
}
```

9.13 SVG

```
struct SVG {
  FILE* out;
  ld sc = 50;
  void open() {
    out = fopen("image.svg", "w");
    fprintf(out, "<svg xmlns='http://www.w3.org/2000/svg'
        viewBox='-1000 -1000 2000 2000'>\n");
  }
  void line(vec a, vec b) {
    a = a * sc, b = b * sc;
    fprintf(out, "<line x1='%Lf' y1='%Lf' x2='%Lf' y2='%Lf'
        stroke='black'/>\n", a.x, -a.y, b.x, -b.y);
  }
  void circle(vec a, ld r = -1, string fill_col = "none") {
    r = (r == -1 ? 10 : sc * r);
    a = a * sc;
    fprintf(out, "<circle cx='%Lf' cy='%Lf' r='%Lf' fill='%s'
        stroke='black' stroke-width='1'/>\n", a.x, -a.y, r,
        fill_col.c_str());
  }
  void text(vec a, string s) {
    a = a * sc;
    fprintf(out, "<text x='%Lf' y='%Lf' font-size='10px'>%s</
        text>\n", a.x, -a.y, s.c_str());
  }
  void close() {
    fprintf(out, "</svg>\n");
    fclose(out);
    out = 0;
  }
  ~SVG() {
    if (out)
      close();
  }
} svg;
```

10 Miscellaneous

10.1 Josephus Problem

Группа из n людей стоит в круге, и каждый k-й человек по счёту исключается из круга до тех пор, пока не останется один человек. Задача заключается в определении позиции этого последнего оставшегося человека.

```
int josephus(int n, int k) {
  if (n == 1) return 0;
  if (k == 1) return n-1;
  if (k > n) return (josephus(n-1, k) + k) % n;
  int cnt = n / k;
  int res = josephus(n - cnt, k);
  res -= n % k;
  if (res < 0) res += n;
  else res += res / (k - 1);
  return res;
}
```

10.2 Knight Moves in Infinity Grid

Задача заключается в определении минимального количества ходов, которые необходимы коню для достижения заданной позиции на бесконечной шахматной доске, начиная с определённой исходной позиции.

```
li get_dist(li dx, li dy) {
  if (++(dx = abs(dx)) > ++(dy = abs(dy))) swap(dx, dy);
  if (dx == 1 && dy == 2) return 3;
  if (dx == 3 && dy == 3) return 4;
  li lb = max(dy / 2, (dx + dy) / 3);
  return ((dx ^ dy ^ lb) & 1) ? ++lb : lb;
}
```

11 Other

11.1 Комбинаторика

Разбиение числа n на k слагаемых:
$C_{n+k-1}^{k-1} = C_{n+k-1}^{n}$ – для $a_i \geq 0$, C_{n-1}^{k-1} – для $a_i \geq 1$, C_{n-k-1}^{k-1} – для $a_i \geq 2$.
Биномиальные коэффициенты:
Способы подсчёта $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$:

1. $\binom{n}{k}$ можно получить из $\binom{n}{0}$ за $O(k)$, с помощью формулы $\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$.

2. По простому (маленькому) модулю можно считать с помощью теоремы Люка:

   $\binom{m}{n} \equiv \prod_{i=0}^{k-1}\binom{m_i}{n_i} \pmod p$, p – простое. $m = (m_{k-1}, \ldots, m_0)_p, n = (n_{k-1}, \ldots, n_0)_p$ – представление чисел m и n в p-ичной системе счисления.

3. Предподсчитать факториалы за $O(n+m)$. При $n, k \sim 10^9$ можно предподсчитать в коде каждый 10^6-й факториал.

4. Можно для каждого простого найти степень, с которой оно входит в $\binom{n}{k}$ и далее получить само число.

5. Если $n \sim 10^{18}$, а $k \sim 10^6$ – маленькое, то $\binom{n}{k}$ по простому модулю (в случае составного надо использовать КТО) можно искать с помощью факториального представления: рекурсивная функция считает произведение по всем некратным p, по кратным вычисляет степень p и далее рекурсивно вызывает себя (поскольку кратные без коэффициента p образуют новые факториалы).

$\overline{C}_n^k = C_{n+k-1}^k$ – число сочетаний с повторениями из n по k,
$A_n^k = \frac{n!}{(n-k)!} = C_n^k\, k!$ – количество размещений из n по k,
$\sum\limits_{k=0}^{n} kC_n^k = n2^{n-1}$, $\sum\limits_{k=0}^{n} k^2C_n^k = (n+n^2)2^{n-2}$, $\sum\limits_{k=0}^{n} (C_n^k)^2 = C_{2n}^n$,
$\sum\limits_{k=0}^{\lfloor \frac{n}{2} \rfloor} C_{n-k}^k = F(n+1)$, где $F(n)$ – n-ое число Фибоначчи,
$\sum\limits_{k \le n} \binom{r+k}{k} = \binom{r+n+1}{n}, n \in \mathbb{Z}$ – параллельное суммирование,
$\sum\limits_{0 \le k \le n} \binom{k}{m} = \binom{n+1}{m+1}, m, n \in \mathbb{Z}, m, n \ge 0$ – верхнее суммирование,
$\sum\limits_{k} \binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}, n \in \mathbb{Z}$ – свёртка Вандермонда.

Числа Каталана: $C_n = \frac{1}{n+1}\binom{2n}{n}$.

Числа Стирлинга:
1-го рода: $Z_n^k = (n-1)Z_{n-1}^k + Z_{n-1}^{k-1}$ – число способов разбиения множества из n элементов на k циклов (циклы нельзя переворачивать, только прокручивать).

2-го рода: $S_n^k = kS_{n-1}^k + S_{n-1}^{k-1} = \frac{1}{k!}\sum\limits_{j=0}^{k}(-1)^{k+j}\binom{k}{j}j^n$ – число способов разбиения множества из n элементов на k непустых подмножеств.
Полагаем, что $Z_n^0 = S_n^0 = [n=0], Z_0^k = S_0^k = [k=0]$. Дуальность: $S_n^k = Z_{-k}^{-n}$.
$x^n = \sum\limits_k S_n^k x^{\underline{k}}$, $x^{\overline{n}} = \sum\limits_k Z_n^k x^k$, $x^n = \sum\limits_k S_n^k (-1)^{n-k} x^{\overline{k}}$, $x^{\underline{n}} = \sum\limits_k Z_n^k (-1)^{n-k} x^k, n \in \mathbb{Z}, n \ge 0$.

Количество неубывающих последовательностей из n элементов от 0 до a равно $\binom{n+a}{n}$.

Перестановки без неподвижных точек: $cnt = n!\sum\limits_{i=0}^{n}\frac{(-1)^n}{i!} \approx \frac{n!}{e}$

Число Непера: $e = \sum\limits_{n=0}^{\infty}\frac{1}{n!}, \frac{1}{e} = \sum\limits_{n=2}^{\infty}\frac{(-1)^n}{n!}$

Количество диаграмм Юнга
Крюк клетки – она сама, а также клетки, расположенные справа от нее, и клетки, расположенные снизу.
Количество заполнений таблицы равно факториалу количества ее клеток, деленному на произведение длин всех крюков.

Теорема Каммера:
$n,\ m \in Z,\ n \ge m \ge 0$ и p – простое число, тогда максимальная степень k, что $p^k | C_n^m$ равна количеству переносов при сложении чисел m и $n-m$ в системе счисления p.

11.2 Теория чисел

Обращение Мебиуса
$g(m) = \sum\limits_{d|m} f(d) \leftrightarrow f(m) = \sum\limits_{d|m}\mu(d)\cdot g(\frac{m}{d})$, где $\mu(d) =$
$\begin{cases} 0, \text{если d не свободно от квадратов;} \\ (-1)^k, \text{k – количество простых в разложении d.} \end{cases}$

Сумма меньших взаимнопростых: $\varphi_1(n) = \frac{n\varphi(n)}{2} = \frac{\varphi(n^2)}{2}$
Китайская теорема об остатках (КТО):
$x \equiv rm_i\ (mod\ m_i)\ (i = \overline{0, n-1})$
Ответ ищем в виде: $x = x_0 + x_1m_0 + \ldots + x_{n-1}m_0m_1 \cdot \ldots \cdot m_{n-2}$
$x_i \leftarrow rm_i$, for $j < i$ do $x_i \leftarrow (x_i - x_j)m_j^{-1}\ (mod\ m_i)$

Вычисление за линейное время: $x \equiv \sum\limits_{i=0}^{n-1} rm_i\frac{M}{m_i}y_i (mod\ M)$, где $M = \prod\limits_{i=0}^{n-1} m_i$ и $y_i \equiv \left(\frac{M}{m_i}\right)^{-1}(mod\ m_i)$.

Расширенный КТО: Заданы числа n, m, a, b, нужно найти число $x \equiv a\ (mod\ n), x \equiv b\ (mod\ m)$. Пусть $g = gcd(n,m)$. Если $a \not\equiv b\ (mod\ g)$, то решения не существует. Пусть $nx_0 + my_0 = g$, тогда $x = \left(\frac{n}{g}x_0a + \frac{m}{g}y_0b\right)\ mod\ lcm(n,m)$ является наименьшим решением.

Решение $x \equiv a^N (mod\ m)$, если $\gcd(a,m) > 1$: Пусть $g = gcd(a^N, m)$, найдем наименьшее k, что $g | a^k$, так что $a^k = a_1 g$, $m = m_1 g$. Тогда $x \equiv a^{N-k}a_1 g (mod\ m_1 g)$, и, следовательно, $x = x_1 g$, где $x_1 \equiv a^{N-k}a_1 (mod\ m_1)$.
Сравнение $ax \equiv b\ (mod\ m)$ имеет либо 0, либо $g = gcd(a,m)$ решений
Функция Кармайкла: $\lambda(n)$ – равна наименьшему показателю m, что $a^m \equiv 1 \pmod n, \forall (a,n) = 1$.
$\lambda(p^\alpha) = \varphi(p^\alpha), \forall p > 2, p \in \mathbb{P}$ или $p^\alpha \in \{2, 4\}$,
$\lambda(2^\alpha) = \frac{\varphi(2^\alpha)}{2}, \forall \alpha > 2$,
$\lambda(p_1^{\alpha_1}p_2^{\alpha_2}\ldots p^{\alpha_k}) = lcm(\lambda(p_1^{\alpha_1}), \lambda(p_2^{\alpha_2}), \ldots, \lambda(p_k^{\alpha_k}))$.
Теорема Вильсона: Натуральное число $p > 1$ является простым тогда и только тогда, когда $(p-1)! + 1 \equiv 0\ (mod\ p)$.
Критерий Эйлера: Пусть $p > 2$ - простое число. Число a, взаимно простое с p, является квадратичным вычетом по модулю p тогда и только тогда, когда $a^{\frac{p-1}{2}} \equiv 1 (mod\ p)$ и является квадратичным невычетом по модулю p тогда и только тогда, когда $a^{\frac{p-1}{2}} \equiv -1 (mod\ p)$.
Свойства чисел Фибоначчи: $F_{n+1}F_{n-1} - F_n^2 = (-1)^n$, $F_{n+k} = F_kF_{n+1} + F_{k-1}F_n$, $F_{2n} = F_n(F_{n+1} + F_{n-1})$.

Цепные дроби
$p_n = a_n \cdot p_{n-1} + p_{n-2}$, $q_n = a_n \cdot q_{n-1} + q_{n-2}$.
Цифровой корень:
Последовательность наименьших целых положительных чисел, которые требуют ровно n итераций извлечения цифрового корня в системе счисления по основанию b: $a_0 = 0$, $a_1 = b$, $a_n = 2b^{\frac{a_{n-1}}{b-1}} - 1$ при $n > 1$.
Числа Гаусса: Это комплексные числа, у которых и действительная, и мнимая части целые. Норма числа $a+ib$ определяется как a^2+b^2. Если $a+ib = (c+id)(e+if)$, то $a^2+b^2 = (c^2+d^2)(e^2+f^2)$. Число $a+ib$ делится на число $c+id$ тогда и только тогда, когда c^2+d^2 делит $ac+bd$ и $bc-ad$.
Гауссово число $a+ib$ является простым тогда и только тогда, когда: 1) либо одно из чисел a, b нулевое, а другое - целое простое число вида $\pm(4k+3)$, 2) либо a, b оба отличны от нуля и норма a^2+b^2 - простое натуральное число.

11.3 Геометрия

Теорема синусов: $\frac{a}{\sin\alpha} = \frac{b}{\sin\beta} = \frac{c}{\sin\gamma} = 2R$, R – радиус описанной окружности.
Теорема косинусов: $a^2 = b^2 + c^2 - 2bc\cos\widehat{bc}$.
Теорема тангенсов: $\frac{a-b}{a+b} = \frac{\tan\frac{1}{2}(\alpha-\beta)}{\tan\frac{1}{2}(\alpha+\beta)}$.

Геометрия на сфере:
1. Большой круг задаётся нормалью.
2. Два неравных круга пересекаются в двух противоположных точках: $\pm normal(cross(n_1, n_2), r)$.
3. Площадь многоугольника равна: $S = (\sum\limits_i \alpha_i - \pi * (n-2)) * r^2$, объем конуса (сектора) равен: $V = S * r/3$ (α_i – внутренний двугранный угол = угол между касательными векторами).
4. $\alpha_i = \pi - ang(n_i, n_{i+1})$, где n_j – нормаль к j-й дуге многоугольника направленная внутрь многоугольника.
5. Для сортировки точек на круге C удобно перейти к $2D$ (введя базис в плоскости круга: $e_1 = p_0, e_2 = cross(e_1, n_c)), p_0$ – любая из точек на круге, n_c – нормаль к кругу.
6. Для сортировки по кругу отрезков исходящих из точки C, надо свести задачу к 5.

Треугольники и их свойства: a, b, c – стороны, треугольника; α, β, γ – соответствующие сторонам a, b, c углы, r, R – радиусы вписанной и описанной окружностей, d – расстояние между центрами вписанной и описанной окружностей.

1. $S = \frac{1}{2}ab\sin\gamma = \frac{a^2\sin\beta\sin\gamma}{2\sin\alpha} = 2R^2\sin\alpha\sin\beta\sin\gamma$,
2. $S = \frac{ab}{2} = r^2 + 2rR, r = \frac{ab}{a+b+c} = \frac{a+b-c}{2}$ – для прямоугольного,
3. $S = \frac{a^2\sqrt{3}}{4}$ – для равностороннего,
4. $R = \frac{abc}{4S}$ – центр в точке пересечения серединных перпендикуляров, $r = \frac{S}{p}$ – центр в точке пересечения биссектрис, $d^2 = R^2 - 2Rr$.

Свойства правильных многоугольников:
Обозначим n – количество сторон многоугольника, a – длину стороны, r – радиус вписанной окружности, R – радиус описанной окружности, S – площадь, p – полупериметр.

1. Площадь правильного n-угольника: $S = \frac{na^2}{4\tan\left(\frac{\pi}{n}\right)}$
2. Радиус вписанной окружности: $r = \frac{S}{p} = \frac{a}{2\tan\left(\frac{\pi}{n}\right)}$
3. Радиус описанной окружности: $R = \frac{a}{2\sin\left(\frac{\pi}{n}\right)}$
4. Связь радиусов: $R = \frac{r}{\cos\left(\frac{\pi}{n}\right)}$
5. Угол при вершине: $\alpha = \frac{(n-2)\pi}{n}$
6. Сумма углов многоугольника: $S = (n-2)\cdot 180^\circ$
7. Связь между площадью и радиусами: $S = \frac{1}{2}nRr$
8. Длина стороны правильного n-угольника через радиус описанной окружности: $a = 2R\sin\left(\frac{\pi}{n}\right)$

Геометрическая инверсия
Инверсия точки P относительно окружности с центром в точке O и радиусом R – это точка P', которая лежит на луче OP, и $OP \cdot OP' = R^2$.
Прямая, проходящая через O, не меняется. Прямая, не проходящая через O, перейдет в окружность, проходящую через O, и наоборот. Окружность, не проходящая через O, перейдет в окружность, по-прежнему не проходящую через O.
Если после инверсии точки P и Q переходят в P' и Q', то $\angle PQO = \angle Q'P'O$, $\angle QPO = \angle P'Q'O$ и треугольники $\triangle PQO$ и $\triangle Q'P'O$ подобны.
Преобразование инверсии сохраняет углы в точках пересечения кривых (ориентация меняется на противоположную).
Обобщённая окружность при преобразовании инверсии сохраняется тогда и только тогда, когда она ортогональна окружности, относительно которой производится инверсия.
Чтобы найти окружность, получившуюся в результате инверсии прямой, нужно найти ближайшую к центру инверсии точку Q прямой, применить к ней инверсию, и тогда искомая окружность будет иметь диаметр OQ'.

Чтобы найти окружность, получившуюся в результате инверсии другой окружности, нужно провести через центр инверсии и центр старой окружности прямую, и посмотреть ее точки пересечения S и T со старой окружностью. Отрезок ST после инверсии будет образовывать диаметр, следовательно, центр новой окружности это среднее арифметическое точек S' и T'.
Окружность с центром в точке (x, y) и радиусом r после инверсии относительно окружности с центром в точке (x_0, y_0) и радиусом r_0 перейдет в окружность с центром в точке (x', y') и радиусом r', где
$x' = x_0 + s \cdot (x - x_0),\ y' = y_0 + s \cdot (y - y_0),\ r' = |s| \cdot r,\ s = \frac{r_0^2}{(x-x_0)^2+(y-y_0)^2-r^2}$.
Шар:
$S = 4\pi R^2,\ V = \frac{4}{3}\pi R^3\ V = V_n R^n,\ V_n = \frac{\pi^{\lfloor n/2 \rfloor}}{\Gamma(\frac{n}{2}+1)},\ V_{2k} = \frac{\pi^k}{k!},\ V_{2k+1} = \frac{2^{k+1}\pi^k}{(2k+1)!!}$
$S = S_n R^n,\ S_0 = 2,\ S_n = 2\pi V_{n-1}$
$V = \frac{1}{3}\pi h^2(3R-h),\ S = 2\pi Rh$ для шарового сегмента, h – высота сегмента
$V = \frac{2}{3}\pi R^2 h$ для шарового сектора, h – высота соответствующего шарового сегмента
Тор:
$S = 4\pi^2 Rr,\ V = 2\pi^2 Rr^2$, R – расстояние от центра образующей окружности до оси вращения, r – радиус образующей окружности
Тетраэдр:
$V = \frac{\sqrt{2}a^3}{12},\ h = \frac{\sqrt{6}a}{3},\ r = \frac{\sqrt{6}a}{12},\ R = \frac{\sqrt{6}a}{4}$,
$288V^2 = \begin{vmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & d_{12}^2 & d_{13}^2 & d_{14}^2 \\ 1 & d_{12}^2 & 0 & d_{23}^2 & d_{24}^2 \\ 1 & d_{13}^2 & d_{23}^2 & 0 & d_{34}^2 \\ 1 & d_{14}^2 & d_{24}^2 & d_{34}^2 & 0 \end{vmatrix}$. При фиксированных попарных расстояниях тетраэдр построить нельзя, если определитель < 0 или хотя бы на одной грани не выполняется неравенство треугольника.
Конус:
$V = \frac{1}{3}\pi R^2 h,\ S = \pi Rl$ – площадь боковой поверхности, где l – образующая.
Круг:
$S = \frac{R^2}{2}\theta$ для сектора круга, $S = \frac{R^2}{2}(\theta - \sin\theta)$ для сегмента круга.
Формула Эйлера для числа граней в планарном графе:
$V - E + F = 1 + C$, где V - число вершин, E - ребер, F - граней, C - компонент связности графа.
Теорема о секущих:
Если из точки, лежащей вне окружности, провести две секущие, то произведение одной секущей на её внешнюю часть равно произведению другой секущей на её внешнюю часть: $AB \cdot AC = AD \cdot AE$
Пересечение плоскости и прямой:
Дана плоскость $Ax+By+Cz+D = 0$ и прямая в виде $a+vt$, тогда точке пересечения будет соответствовать параметр
$t = -\frac{A\cdot a.x + B\cdot a.y + C\cdot a.z + D}{A\cdot v.x + B\cdot v.y + C\cdot v.z}$
Сферические координаты:
Если $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ – широта, а $\varphi \in [0, 2\pi)$ – долгота, то
$x = r\cos\theta\cos\varphi,\ y = r\cos\theta\sin\varphi,\ z = r\sin\theta$.
Пересечение окружности и прямой: Сдвинем систему координат в центр окружности. Для сдвига прямой на вектор (dx, dy) нужно сделать $C- = A \cdot dx + B \cdot dy$, в данном случае $dx = -x_0, dy = -y_0$. Теперь если $d > r$ $(d = \frac{|C|}{\sqrt{A^2+B^2}})$, то точек пересечения нет. Обозначим $l = \sqrt{r^2 - d^2}, c = (\frac{-AC}{A^2+B^2},\ \frac{-BC}{A^2+B^2})$. Тогда точки пересечения имеют вид: $c \pm normal(pt(-B, A), l)$.
Центры масс: Везде далее будем выражать центры масс с помощью радиус-векторов.
Центр масс системы материальных точек: $\vec{r_c} = \frac{\sum\limits_i \vec{r_i}\ m_i}{\sum\limits_i m_i}$,
Центр масс однородного каркаса, как многоугольника, так и многогранника (заменяем каждое ребро точкой в его середине с массой, равной длине ребра): $\vec{r_c} = \frac{\sum\limits_i \vec{r_i^{mid}}\ l_i}{P}$,
Центр масс сплошных фигур: центр масс произвольного сплошного треугольника или тетраэдра - среднее арифметическое его координат (обобщается и на симплексы больших размерностей). Центр масс произвольного сплошного многоугольника/многогранника считается следующим образом: выбирается произвольная точка p из нее проводятся треугольники/ тетраэдры к последовательным вершинам фигуры (или к треугольникам из триангуляции грани) и вычисляется взвешенное среднее центров масс треугольников/ тетраэдров с весами, равными знаковым площадям/объемам.
Центр масс поверхности многогранника - это взвешенное среднее центров масс граней с весами, равными площадям граней.
Окружности Мальфатти:
$r_1 = \frac{r}{2(p-a)}(p+d-r-e-f),\ r_2 = \frac{r}{2(p-b)}(p+e-r-d-f),\ r_3 = \frac{r}{2(p-c)}(p+f-r-d-e)$, где d, e, f – расстояния от инцентра до углов A, B, C соответственно

11.4 Графы

Теорема Холла: Если в двудольном графе произвольно выбранное множество вершин первой доли покрывает не меньшее по размеру множество вершин второй доли, то в графе существует совершенный парсоч.
Аксиомы частично упорядоченных множеств (ЧУМ):

$$1.\ a < b \wedge b < c \Rightarrow a < c;\ \ 2.\ a < b \Rightarrow a \neq b;\ \ 3.\ a < b \Rightarrow \overline{b < a}.$$

Цепь – множество попарносравнимых элементов.
Антицепь – множество попарнонесравнимых элементов.
Теорема Дилуорса: Размер максимальной антицепи равен размеру минимального покрытия ЧУМа цепями.
Следствие: Наибольшее количество вершин в орграфе таких, что никакая вершина недостижима из другой равно минимальному покрытию замыкания этого графа путями (циклы не мешают).
DAG minimum covering: Необходимо покрыть DAG наименьшим количеством вершинно-непересекающихся путей. Для этого построим двудольный граф вершины есть которого раздвоенные вершины исходного графа. Дугу (x, y) исходного графа заменяем на дугу (x_1, y_2) нового. Находим макс. парсоч. Теперь для восстановления ответа достаточно сказать, что вершины соединенные ребром парсоча являются соседними в одном из путей ответа.
Дейкстра с потенциалами: Введем потенциалы φ_i, новые веса ребер будут иметь вид: $\overline{c_{ij}} = c_{ij} + (\varphi_i - \varphi_j) \geq 0$. В качестве потенциалов можно выбрать $\varphi_i = d_i$, где d_i – кратчайшие расстояния из истока (специально добавленной в граф вершины из которой есть дуги веса 0 во все остальные вершины) вычисленные с помощью алгоритма Форда-Беллмана. В частности в задаче $minCostFlow$ можно положить исходно $\varphi_i = 0$ и после каждого шага увеличивать все φ_i на величину d_i. φ_i могут переполниться у недостижимых вершин.
Паросочетания:
В произвольном графе: $|MIVS| + |MVC| = |V|, |MM| + |MEC| = |V|$.
В двудольном графе: $|MVC| = |MM|$.
Для нахождения MVC в двудольном графе нужно запустить Куна из ненасыщенных вершин первой доли, тогда вершины по которым мы не прошли в первой доле и прошли во второй образуют MVC ($MIVS = V \backslash MVC$).
Для нахождения MEC построим MM. Теперь из каждой непокрытой (неизолированной) вершины все ребра ведут в насыщенные вершины. Выберем для каждой ненасыщенной вершины любое ребро и добавим эти ребра в MM получим MEC.
MEC представляет собой лес (с деревьями диаметра не более 2), поэтому для получения MM нужно из каждого дерева взять по одному ребру.
Наибольшее доминирование: X – множество вершин первой доли, Y – множество вершин второй доли которые покрыты вершинами из X. Необходимо найти X, чтобы величина $|X| - |Y|$ была максимальна. Для этого построим MM и запустим Куна из ненасыщенных вершин первой доли, тогда вершины по которым мы прошли в первой доле образуют множество X, а вершины по которым мы прошли во второй доле образуют множество Y и при этом значение $|X| - |Y|$ будет максимально.
2-SAT:
1. Задачу всегда НЕОБХОДИМО сводить к каноническому виду $2-CNF$: $(a_i \vee b_i) \wedge (a_{i+1} \vee b_{i+1})$. Все бинарные операции легко выражаются в этой форме. В частности импликация $x \to y$ выражается, как $(!x \vee y)$.
2. Если заранее известно чему равно значение x_i (т.е. x_i константа), то достаточно ввести либо ребро $!x_i \to x_i$, если $x_i = 1$, либо $x_i \to !x_i$.
3. Ответ восстанавливается очень просто среди значений x_0, x_1 выбирается то, компонента сильной связности которой стоит позже в порядке топологической сортировки (именно компонента, а не вершина x_i).
4. От произвольной таблицы истинности легко перейти к форме $2-CNF$: для этого нужно выбрать все строчки в которых значение функции равно 0 и добавить дизъюнкцию в которой переменные равные 1 взяты с отрицанием, а равные 0 – без отрицания.
Покраска подпути в дереве на min (offline $O(nlogn)$):
Для покраски подпути из вершины x в вершину y нужно предварительно подвесить дерево и заменить покраску (x, y) на 2 покраски (x, l), (y, l) $(l = lca(x, y))$. Для покраски (v, p) нужно аналогично двоичному подъему разбить запрос по степеням и сделать "ленивую" операцию. В конце можно просто "пропушить" все операции в порядке уменьшения степеней двойки.
MVC **в произвольном графе за** $O(\varphi^n)$
MVC ищется перебором. Предварительно для каждой вершины степени 1 нужно взять её соседа (если он тоже степени 1, то нужно взять только одного из них). Далее в переборе каждый раз ищем вершину с наибольшим количеством непокрытых рёбер. Теперь нам нужно взять либо её, либо всех её соседей ребро к которым ещё не покрыто.
Матричная теорема Кирхгофа: Пусть задан неориентированный связный граф с кратными ребрами. Если в матрице смежности графа заменить каждый элемент на противоположный по знаку, а элемент a_{ii} заменить на степень вершины i (с учетом кратности ребер), то все алгебраические дополнения этой матрицы равны между собой и равны количеству остовных деревьев этого графа.
Матрица Татта: Пусть в графе существует совершенное паросочетание, тогда его матрица Татта невырождена. Если алгебраическое дополнение элемента, соответствующего ребру (i, j), т.е. элемент $A^{-1}_{j,i}$, отличен от нуля, то это ребро может входить в совершенное паросочетание.

11.5 Формулы

Элементарная тригонометрия:
$\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta$; $\cos(\alpha \pm \beta) = \cos\alpha\cos\beta \mp \sin\alpha\sin\beta$;
$\tan(\alpha \pm \beta) = \frac{\tan\alpha \pm \tan\beta}{1 \mp \tan\alpha\tan\beta}$;
$\sin\alpha\cos\beta = \frac{1}{2}(\sin(\alpha+\beta) + sin(\alpha-\beta))$; $\sin\alpha\sin\beta = \frac{1}{2}(\cos(\alpha-\beta) - cos(\alpha+\beta))$;
$\cos\alpha\cos\beta = \frac{1}{2}(\cos(\alpha-\beta) + cos(\alpha+\beta))$; $\sin\alpha \pm \sin\beta = 2\sin\frac{\alpha\pm\beta}{2}\cos\frac{\alpha\mp\beta}{2}$;
$\cos\alpha - \cos\beta = -2\sin\frac{\alpha+\beta}{2}\sin\frac{\alpha-\beta}{2}$;
$\cos\alpha + \cos\beta = 2\cos\frac{\alpha+\beta}{2}\cos\frac{\alpha-\beta}{2}$; $\tan\alpha \pm \tan\beta = \frac{\sin(\alpha\pm\beta)}{\cos\alpha\cos\beta}$; $\cot\alpha \pm \cot\beta = \frac{\sin(\beta\pm\alpha)}{\sin\alpha\sin\beta}$
Волшебная сумма: $\sum\limits_{0 \leq k < m} \lfloor \frac{nk+x}{m} \rfloor = \sum\limits_{0 \leq k < n} \lfloor \frac{mk+x}{n} \rfloor = d\lfloor \frac{x}{d} \rfloor + \frac{(m-1)(n-1)}{2} + \frac{d-1}{2}$, $d = (n, m)$.
Суммирование по частям:
$\Delta f(x) = f(x+1) - f(x), Ef(x) = f(x+1) \Rightarrow \sum u\Delta v = uv - \sum Ev\Delta u$,
$\Delta x^{\underline{m}} = mx^{\underline{m-1}}, \Delta c^x = (c-1)c^x, \Delta(af+bg) = a\Delta f + b\Delta g, \Delta fg = f\Delta g + Eg\Delta f$.
Формулы округлений:

$\lfloor x \rfloor = n \Leftrightarrow n \le x < n+1 \Leftrightarrow x-1 < n \le x$, $\lceil x \rceil = n \Leftrightarrow n-1 < x \le n \Leftrightarrow x \le n < x+1$,
$x < n \Leftrightarrow \lfloor x \rfloor < n$, $n < x \Leftrightarrow n < \lceil x \rceil$, $x \le n \Leftrightarrow \lceil x \rceil \le n$, $n \le x \Leftrightarrow n \le \lfloor x \rfloor$.

Теорема Пика: Для многоугольника без самопересечений с целочисленными вершинами имеем: $S = I + \frac{B}{2} - 1$, где S – площадь, I – количество целочисленных точек внутри, B – количество целочисленных точек на границе.

Интерполяционный многочлен Лагранжа:
Заданы пары значений (x_i, y_i) $(i = \overline{0,n})$ – узел и значение функции в узле. Тогда существует единственный многочлен степени не более n принимающий значения y_i в узлах x_i: $f(x) = \sum\limits_{i=0}^{n} y_i \prod\limits_{j=0, j\neq i}^{n} \frac{x-x_j}{x_i-x_j}$

Интерполяционный многочлен Ньютона:
Заданы пары значений (x_i, y_i) $(i = \overline{0,n})$ – узел и значение функции в узле. Определим разделенные разности вперед: $[y_\nu] := y_\nu (\nu = \overline{0,n})$;
$[y_\nu, \dots, y_{\nu+j}] := \frac{[y_{\nu+1}, \dots, y_{\nu+j}] - [y_\nu, \dots, y_{\nu+j-1}]}{x_{\nu+j} - x_\nu} (\nu = \overline{0, \dots, n-j}, j = \overline{1, \dots, n})$.
$f(x) = \sum\limits_{i=0}^{n} [y_0, \dots, y_i] \prod\limits_{j=0}^{i-1} (x - x_j)$.

Если пары значений (i, f_i), то можно записать как $P_n(x) = \sum\limits_{m=0}^{n} x^{\underline{m}} d_m$.
Где $d_m = \sum\limits_{k=0}^{m} \frac{f_k}{k!} \cdot \frac{(-1)^{m-k}}{(m-k)!}$. Тогда $A = \sum\limits_{m=0}^{n} \frac{f_m}{m!}$, $B = \sum\limits_{m=0}^{n} \frac{(-1)^m}{m!}$ и $[d_m] = A \times B$.
Нужны только $n+1$ первых элементов, остальные занулить. Значение в точке: $P(k) = \sum\limits_{m=0}^{n} k^{\underline{m}} d_m = \sum\limits_{m=0}^{k} \frac{k!}{(k-m)!} \cdot d_m$. Если $C = \sum\limits_{m=0}^{k} \frac{1}{m!}$, то $P(k) = k! \cdot [[d_m] \times C]_k$

Криволинейный интеграл первого рода:
Пусть l – гладкая, спрямляемая (имеет конечную длину) кривая, заданная параметрически: $x = x(t), y = y(t), z = z(t)$. Пусть $f(x,y,z)$ определена и интегрируема вдоль кривой l. Тогда:

$$\int_l f(x,y,z)dl = \int_a^b f(x(t),y(t),z(t))\sqrt{\dot{x}^2+\dot{y}^2+\dot{z}^2}dt$$

Здесь точка – это производная по t. На плоскости удобно вводить параметризацию: $x = t, y = y(t)$. Для вычисления длины кривой нужно положить $f(x,y,z) \equiv 1$.

Поверхностный интеграл первого рода:
Пусть на поверхности Φ можно ввести единую параметризацию посредством функций $x = x(u,v), y = y(u,v), z = z(u,v)$, заданных в ограниченной области Ω плоскости (u,v) и принадлежащих классу C^1 (непрерывнодифференцируемых) в этой области. Если функция $f(M) = f(x,y,z)$ непрерывна на поверхности Φ, то поверхностный интеграл первого рода от этой функции по поверхности Φ существует и может быть вычислен по формуле:

$$\iint\limits_\Phi f(M)d\sigma = \iint\limits_\Omega f(x(u,v),y(u,v),z(u,v))\sqrt{EG-F^2}dudv$$

где $E = (x'_u)^2 + (y'_u)^2 + (z'_u)^2, F = x'_u x'_v + y'_u y'_v + z'_u z'_v, G = (x'_v)^2 + (y'_v)^2 + (z'_v)^2$.

Теорема Безу: Остаток от деления многочлена $P(x)$ на двучлен $(x-a)$ равен $P(a)$.

Теорема Байеса:
$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$, где $P(A|B)$ - вероятность наступления A, если уже наступило B

Неприводимые многочлены:
Количество неприводимых многочленов (неразложимых на произведение) степени n в поле по простому модулю p равно: $cnt = \frac{1}{n} \sum\limits_{d|n} \mu(d) p^{n/d}$.

Коды Грея:
Код Грея – это такая перестановка битовых строк длины n, что каждая следующая отличается от предыдущей ровно в одном бите. n-й код Грея соответствует гамильтонову циклу вдоль вершин n-го куба.

```
int g (int n) { return n ^ (n >> 1); }
```

НОД многочлена и его производной
$deg(p) = deg(GCD(p(x), p'(x))) + k(p)$, где $k(p)$ – количество различных корней.
$p(x) = c \cdot (x-a_0)^{n_0} \cdot (x-a_1)^{n_1} \cdot \dots \cdot (x-a_k)^{n_k}$
$p'(x) = c \cdot [(x-a_0)^{n_0-1} \cdot (x-a_1)^{n_1} \cdot \dots \cdot (x-a_k)^{n_k} + (x-a_0)^{n_0} \cdot (x-a_1)^{n_1-1} \cdot \dots \cdot (x-a_k)^{n_k} + (x-a_0)^{n_0} \cdot (x-a_1)^{n_1} \cdot \dots \cdot (x-a_k)^{n_k-1}]$
$GCD(p(x), p'(x)) = c \cdot (x-a_0)^{n_0-1} \cdot (x-a_1)^{n_1-1} \cdot \dots \cdot (x-a_k)^{n_k-1}$

Метод касательных Ньютона:
Для решения уравнения $f(x) = 0$ будем проводить итерации: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$, где x_n – n-е приближение решения. В качестве x_0 можно выбрать любое значение, но для улучшения скорости сходимости лучше выбрать x_0 близким к искомому решению.

Метод простой итерации (в матричном виде): $x = (A+E)x - b$.

Метод наименьших квадратов: Решает для n точек задачу вида $\sum\limits_{i=1}^{n} (ax_i + b - y_i)^2 \to min$. Решением задачи является $a = \frac{n\sum\limits_{i=1}^{n} x_i y_i - \sum\limits_{i=1}^{n} x_i \sum\limits_{i=1}^{n} y_i}{n\sum\limits_{i=1}^{n} x_i^2 - \left(\sum\limits_{i=1}^{n} x_i\right)^2}$, $b = \frac{\sum\limits_{i=1}^{n} y_i - a\sum\limits_{i=1}^{n} x_i}{n}$.

Обратная польская нотация: Левоассоциативные операции (т.е. те, которые в случае равенства приоритета выполняются слева направо, например, $+$, $-$, $*$) выталкивают из стека операции с $\ge$ приоритетом. Правоассоциативные (например, возведение в степень) выталкивают операции с $>$ приоритетом. Унарные операции удобно предварительно заменить специальными символами, они имеют наибольший приоритет и зачастую считаются правоассоциативными. Открывающаяся скобка просто помещается в стек, а закрывающаяся выталкивает все операции, пока не встретит открывающуюся.

Формула Райзера для перманента: $Per(A) = (-1)^n \sum\limits_{S \subseteq \{1,\dots,n\}} (-1)^{|S|} \prod\limits_{i=1}^{n} \sum\limits_{j \in S} a_{ij}$.

Рюкзак на маленьких весах: Задан набор чисел A_i, а также набор чисел x_i. Для каждого x_i необходимо определить можно ли его набрать числами из набора A_i, причем каждое A_i можно брать более одного раза. Для решения этой задачи зафиксируем произвольное число из набора, например можно зафиксировать минимальное среди них $c := A_0$. Теперь посчитаем величины d_i – наименьшая сумма, которую можно набрать числами из набора A_i, такая что остаток этой суммы по модулю c равен i (это можно сделать например алгоритмом Дейкстры, хотя граф весьма специфичен и можно это делать быстрее). Теперь проверка числа x_i сводится к проверке неравенства $d[x_i \mod c] \le x_i$.

Пятнашки
Пусть a_i – перестановка чисел от 0 до 15 (в порядке сверху-вниз, слева-направо), N – количество инверсий в a, $K = \lfloor \frac{z-1}{4} \rfloor + 1$ – номер строки с нулем, где z – 1-индексированный номер позиции нуля в a. Тогда решение существует тогда и только тогда, когда $N + K$ четно.

11.6 Полезные числа

n	3^n	$n!$	B_n	C_n
0	1	1	1	1
1	3	1	1	1
2	9	2	2	2
3	27	6	5	5
4	81	24	15	14
5	243	120	52	42
6	729	720	203	132
7	2’187	5’040	877	429
8	6’561	40’320	4’140	1’430
9	19’683	362’880	21’147	4’862
10	59’049	3’628’800	115’975	16’796
11	177’147	39’916’800	678’570	58’786
12	531’441	479’001’600	4’213’597	208’012
13	1’594’323	6’227’020’800	27’644’437	742’900
14	4’782’969	-	190’899’322	2’674’440
15	14’348’907	-	1’382’958’545	9’694’845
16	43’046’721	-	-	35’357’670
17	129’140’163	-	-	129’644’790
18	387’420’489	-	-	477’638’700
19	1’162’261’467	-	-	1’767’263’190

S_n^k	0	1	2	3	4	5	6	7	8	9	10	11
0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	0	0	0
3	0	1	3	1	0	0	0	0	0	0	0	0
4	0	1	7	6	1	0	0	0	0	0	0	0
5	0	1	15	25	10	1	0	0	0	0	0	0
6	0	1	31	90	65	15	1	0	0	0	0	0
7	0	1	63	301	350	140	21	1	0	0	0	0
8	0	1	127	966	1’701	1’050	266	28	1	0	0	0
9	0	1	255	3’025	7’770	6’951	2’646	462	36	1	0	0
10	0	1	511	9’330	34’105	42’525	22’827	5’880	750	45	1	0
11	0	1	1’023	28’501	145’750	246’730	179’487	63’987	11’880	1’155	55	1