Team notebook

HCMUS-THONK

October 29, 2022

Contents

1 DP-Optimization

1.1 ConvexHullTrick

```
// solving DP[i] = Aj * xi + Bj + C
deque<ii> st;
ll calc(ii x, ll y) {
  return x.F * y + x.S;
```

```
}
bool ck(ii x, ii y, ii z) {
  return ll(y.S - x.S) * (y.F - z.F) <= ll(z.S - y.S) * (x.F - y.F); // for
      max: >=
}
// inside main()
  // lines must be sorted inc/dec by A (Ax + B)
  // FOR EACH x inc/dec
    while(sz(st) > 1 && calc(st.front(), x) >= calc(st[1], x)) st.pop_front();
        // for max: <=
    dp[i] = ... + calc(st.front(), x);
    ii t = mp(Ai, Bi);
    while(sz(st) > 1 && ck(t, st.back(), st[sz(st) - 2])) st.pop_back();
    st.pb(t);
```

1.2 DivideConquer

```
 // DIVIDE AND CONQUER OPTIMIZATION ( dp[i][k] = min j<k {dp[j][k-1] + C(j,i)}
     )
 // Description: searches for bounds to optimal point using the monotocity
     condition
 // Condition: L[i][k] <= L[i+1][k]
 // Time Complexity: O(K*N^2) becomes O(K*N*logN)
 // Notation: dp[i][k]: optimal solution using k positions, until position i
 // L[i][k]: optimal point, smallest j which minimizes dp[i][k]
 // C(i,j): cost for splitting range [j,i] to j and i
const int N = 1e3 + 5;
ll dp[N][N];
//Cost for using i and j
ll C(ll i, ll j);
void compute(ll l, ll r, ll k, ll optl, ll optr) {
  // stop condition
  if (l > r) return;
  ll mid = (l + r) / 2;
  //best : cost, pos
  pair<ll,ll> best = {LINF, -1};
  //searchs best: lower bound to right, upper bound to left
  for (ll i = optl; i <= min(mid, optr); i++) {
    best = min(best, {dp[i][k - 1] + C(i, mid),i});
  }
  dp[mid][k] = best.first;
  ll opt = best.second;
  compute(l, mid - 1, k, optl, opt);
  compute(mid + 1, r, k, opt, optr);
}
//Iterate over k to calculate
ll solve() {
  //dimensions of dp[N][K]
  int n, k;
  //Initialize DP
  for (ll i = 1; i <= n; i++) {
    //dp[i,1] = cost from 0 to i
    dp[i][1] = C(0, i);
  }
  for (ll l = 2; l <= k; l++) {
    compute(1, n, l, 1, n);
  }
  /*+ Iterate over i to get min{dp[i][k]}, don't forget cost from n to i
    for(ll i=1;i<=n;i++){
        ll rest = ;
        ans = min(ans,dp[i][k] + rest);
    }
  */
}
```

1.3 Knuth

```
// Optimize O(n^3) -> O(n^2)
// dp(i, j) = min[i <= k < j](dp(i, k) + dp(k + 1, j) + C(i, j))
// REQUIRE: opt(i, j - 1) <= opt(i, j) <= opt(i + 1, j)
int solve() {
  int N;
  ... // read N and input
  int dp[N][N], opt[N][N];
  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 = inf, 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;
  }}
  cout << dp[0][N-1] << endl;
}
```

1.4 SOS

```
// O(N * 2^N)
// A[i] = initial values
// Calculate F[i] = Sum of A[j] for j subset of i
for(int i = 0; i < (1 << N); i++) F[i] = A[i];
for(int i = 0; i < N; i++)
  for(int j = 0; j < (1 << N); j++)
      if(j & (1 << i)) F[j] += F[j ^ (1 << i)];
```

1.5 SteinerTree

```
// Steiner-Tree O(2^t*n^2 + n*3^t + APSP)
// N - number of nodes
// T - number of terminals
// dist[N][N] - Adjacency matrix
// steiner_tree() = min cost to connect first t nodes, 1-indexed
// dp[i][bit_mask] = min cost to connect nodes active in bitmask rooting in i
// min{dp[i][bit_mask]}, i <= n if root doesn't matter
int n, t, dp[N][(1 << T)], dist[N][N];
int steiner_tree() {
  for (int k = 1; k <= n; ++k)
    for (int i = 1; i <= n; ++i)
      for (int j = 1; j <= n; ++j)
        dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
  for(int i = 1; i <= n; i++)
    for(int j = 0; j < (1 << t); j++)
      dp[i][j] = INF;
  for(int i = 1; i <= t; i++) dp[i][1 << (i-1)] = 0;
  for(int msk = 0; msk < (1 << t); msk++) {
    for(int i = 1; i <= n; i++) {
      for(int ss = msk; ss > 0; ss = (ss - 1) & msk)
        dp[i][msk] = min(dp[i][msk], dp[i][ss] + dp[i][msk - ss]);
      if(dp[i][msk] != INF)
        for(int j = 1; j <= n; j++)
          dp[j][msk] = min(dp[j][msk], dp[i][msk] + dist[i][j]);
    }
  }
  int mn = INF;
  for(int i = 1; i <= n; i++) mn = min(mn, dp[i][(1 << t) - 1]);
  return mn;
}
```

2 DataStructure

2.1 DSURollback

```
// 0-based
struct Data {
  int time, u, par;
};
struct DSU {
  vi par;
  vt<Data> change;
  DSU(int n) : par(n + 5, -1) {}
  int getRoot(int x) {
    while (par[x] >= 0) x = par[x];
    return x;
  }
  bool join(int x, int y, int t) {
    x = getRoot(x);
    y = getRoot(y);
    if (x == y) return 0;
    if (par[x] < par[y]) swap(x, y);
    change.pb({t, y, par[y]});
    par[y] += par[x];
    change.pb({t, x, par[x]});
    par[x] = y;
    return 1;
  }
  void rollback(int t) {
```

```
    while (!change.empty() && change.back().time > t) {
      par[change.back().u] = change.back().par;
      change.pop_back();
    }
  }
};
```

2.2 Fenwick

```
/**
 * Source: folklore/TopCoder
 * Description: Computes partial sums a[0] + a[1] + ... + a[pos - 1], and
     updates single elements a[i],
 * taking the difference between the old and new value.
 * Time: Both operations are $O(\log N)$.
 */
struct FT {
  vt<ll> s;
  FT(int n) : s(n) {}
  void update(int pos, ll dif) { // a[pos] += dif
      for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
  }
  ll query(int pos) { // sum of values in [0, pos)
      ll res = 0;
      for (; pos > 0; pos &= pos - 1) res += s[pos-1];
      return res;
  }
  int lower_bound(ll sum) {// min pos st sum of [0, pos] >= sum
      // Returns n if no sum is >= sum, or -1 if empty sum is.
      if (sum <= 0) return -1;
      int pos = 0;
      for (int pw = 1 << 25; pw; pw >>= 1) {
        if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
            pos += pw, sum -= s[pos-1];
      }
      return pos;
  }
};
```

2.3 Fenwick2D

```
/**
 * Source: folklore
 * Description: Computes sums a[i,j] for all i<I, j<J, and increases single
     elements a[i,j].
 *  Requires that the elements to be updated are known in advance (call
     fakeUpdate() before init()).
 * Time: $O(\log^2 N)$. (Use persistent segment trees for $O(\log N)$.)
 */
struct FT2 {
  vvi ys; vt<FT> ft;
  FT2(int limx) : ys(limx) {}
  void fakeUpdate(int x, int y) {
      for (; x < sz(ys); x |= x + 1) ys[x].pb(y);
  }
  void init() {
      for (vi& v : ys) sort(all(v)), ft.eb(sz(v));
  }
  int ind(int x, int y) {
      return (int)(lower_bound(all(ys[x]), y) - ys[x].begin()); }
  void update(int x, int y, ll dif) {
      for (; x < sz(ys); x |= x + 1)
        ft[x].update(ind(x, y), dif);
  }
  ll query(int x, int y) {
      ll sum = 0;
      for (; x; x &= x - 1)
        sum += ft[x-1].query(ind(x-1, y));
      return sum;
  }
};
```

3 Geometry

3.1 basic

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

```
#define nd second
#define pb push_back
#define cl(x, v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<
typedef long long ll;
typedef long double ld;
typedef pair<int, int> pii;
typedef pair<int, pii> piii;
typedef pair<ll, ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector<vi> vii;
const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9 + 7;
const int N = 1e5 + 5;
typedef long double type;
// for big coordinates change to long long
bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }
int sign(type x) { return ge(x, 0) - le(x, 0); }
struct point {
  type x, y;
  point() : x(0), y(0) {}
  point(type x, type y) : x(x), y(y) {}
  point operator-() { return point(-x, -y); }
  point operator+(point p) { return point(x + p.x, y + p.y); }
  point operator-(point p) { return point(x - p.x, y - p.y); }
  point operator*(type k) { return point(k * x, k * y); }
  point operator/(type k) { return point(x / k, y / k); }
  // inner product
  type operator*(point p) { return x * p.x + y * p.y; }
  // cross product
  type operator%(point p) { return x * p.y - y * p.x; }
  bool operator==(const point &p) const { return x == p.x and y == p.y; }
  bool operator!=(const point &p) const { return x != p.x or y != p.y; }
  bool operator<(const point &p) const {
    return (x < p.x) or (x == p.x and y < p.y);
  }
  // 0 => same direction
  // 1 => p is on the left
  //-1 => p is on the right
  int dir(point o, point p) {
    type x = (*this - o) % (p - o);
    return ge(x, 0) - le(x, 0);
  }
  bool on_seg(point p, point q) {
    if (this->dir(p, q)) return 0;
    return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) &&
           ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y));
  }
  ld abs() { return sqrt(x * x + y * y); }
  type abs2() { return x * x + y * y; }
  ld dist(point q) { return (*this - q).abs(); }
  type dist2(point q) { return (*this - q).abs2(); }
  ld arg() { return atan2l(y, x); }
  // Project point on vector y
  point project(point y) { return y * ((*this * y) / (y * y)); }
  // Project point on line generated by points x and y
  point project(point x, point y) { return x + (*this - x).project(y - x); }
  ld dist_line(point x, point y) { return dist(project(x, y)); }
  ld dist_seg(point x, point y) {
    return project(x, y).on_seg(x, y) ? dist_line(x, y) : min(dist(x),
        dist(y));
  }
  point rotate(ld sin, ld cos) {
    return point(cos * x - sin * y, sin * x + cos * y);
  }
  point rotate(ld a) { return rotate(sin(a), cos(a)); }
  // rotate around the argument of vector p
  point rotate(point p) { return rotate(p.y / p.abs(), p.x / p.abs()); }
};
int direction(point o, point p, point q) { return p.dir(o, q); }
point rotate_ccw90(point p) { return point(-p.y, p.x); }
point rotate_cw90(point p) { return point(p.y, -p.x); }
// for reading purposes avoid using * and % operators, use the functions below:
type dot(point p, point q) { return p.x * q.x + p.y * q.y; }
type cross(point p, point q) { return p.x * q.y - p.y * q.x; }
// double area
type area_2(point a, point b, point c) {
  return cross(a, b) + cross(b, c) + cross(c, a);
}
int angle_less(const point &a1, const point &b1, const point &a2,
               const point &b2) {
  // angle between (a1 and b1) vs angle between (a2 and b2)
```

```cpp
  // 1 : bigger
  //-1 : smaller
  // 0 : equal
  point p1(dot(a1, b1), abs(cross(a1, b1)));
  point p2(dot(a2, b2), abs(cross(a2, b2)));
  if (cross(p1, p2) < 0) return 1;
  if (cross(p1, p2) > 0) return -1;
  return 0;
}
ostream &operator<<(ostream &os, const point &p) {
  os << "(" << p.x << "," << p.y << ")";
  return os;
}
```

3.2 circle

```cpp
#include "basics.cpp"
#include "lines.cpp"
struct circle {
  point c;
  ld r;
  circle() {c = point();r = 0;}
  circle(point _c, ld _r) : c(_c), r(_r) {}
  ld area() { return acos(-1.0) * r * r; }
  ld chord(ld rad) { return 2 * r * sin(rad / 2.0); }
  ld sector(ld rad) { return 0.5 * rad * area() / acos(-1.0); }
  bool intersects(circle other) { return le(c.dist(other.c), r + other.r); }
  bool contains(point p) { return le(c.dist(p), r); }
  pair<point, point> getTangentPoint(point p) {
    ld d1 = c.dist(p), theta = asin(r / d1);
    point p1 = (c - p).rotate(-theta);
    point p2 = (c - p).rotate(theta);
    p1 = p1 * (sqrt(d1 * d1 - r * r) / d1) + p;
    p2 = p2 * (sqrt(d1 * d1 - r * r) / d1) + p;
    return make_pair(p1, p2);
  }
};
circle circumcircle(point a, point b, point c) {
  circle ans;
  point u = point((b - a).y, -(b - a).x);
  point v = point((c - a).y, -(c - a).x);
  point n = (c - b) * 0.5;
  ld t = cross(u, n) / cross(v, u);
  ans.c = ((a + c) * 0.5) + (v * t);
  ans.r = ans.c.dist(a);
  return ans;
}
point compute_circle_center(point a, point b, point c) {
  // circumcenter
  b = (a + b) / 2;
  c = (a + c) / 2;
  return compute_line_intersection(b, b + rotate_cw90(a - b), c, c +
      rotate_cw90(a - c));
}
int inside_circle(point p, circle c) {
  if (fabs(p.dist(c.c) - c.r) < EPS) return 1;
  else if (p.dist(c.c) < c.r) return 0;
  else return 2;
} // 0 = inside/1 = border/2 = outside
circle incircle(point p1, point p2, point p3) {
  ld m1 = p2.dist(p3);
  ld m2 = p1.dist(p3);
  ld m3 = p1.dist(p2);
  point c = (p1 * m1 + p2 * m2 + p3 * m3) * (1 / (m1 + m2 + m3));
  ld s = 0.5 * (m1 + m2 + m3);
  ld r = sqrt(s * (s - m1) * (s - m2) * (s - m3)) / s;
  return circle(c, r);
}
circle minimum_circle(vt<point> p) {
  random_shuffle(p.begin(), p.end());
  circle C = circle(p[0], 0.0);
  for (int i = 0; i < (int)p.size(); i++) {
    if (C.contains(p[i])) continue;
    C = circle(p[i], 0.0);
    for (int j = 0; j < i; j++) {
      if (C.contains(p[j])) continue;
      C = circle((p[j] + p[i]) * 0.5, 0.5 * p[j].dist(p[i]));
      for (int k = 0; k < j; k++) {
        if (C.contains(p[k])) continue;
        C = circumcircle(p[j], p[i], p[k]);
      }
    }
  }
  return C;
}
// compute intersection of line through points a and b with
```

```cpp
// circle centered at c with radius r > 0
vt<point> circle_line_intersection(point a, point b, point c, ld r) {
  vt<point> ret;
  b = b - a;
  a = a - c;
  ld A = dot(b, b);
  ld B = dot(a, b);
  ld C = dot(a, a) - r * r;
  ld D = B * B - A * C;
  if (D < -EPS) return ret;
  ret.push_back(c + a + b * (sqrt(D + EPS) - B) / A);
  if (D > EPS) ret.push_back(c + a + b * (-B - sqrt(D)) / A);
  return ret;
}
vt<point> circle_circle_intersection(point a, point b, ld r, ld R) {
  vt<point> ret;
  ld d = sqrt(a.dist2(b));
  if (d > r + R || d + min(r, R) < max(r, R)) return ret;
  ld x = (d * d - R * R + r * r) / (2 * d);
  ld y = sqrt(r * r - x * x);
  point v = (b - a) / d;
  ret.push_back(a + v * x + rotate_ccw90(v) * y);
  if (y > 0) ret.push_back(a + v * x - rotate_ccw90(v) * y);
  return ret;
}
// GREAT CIRCLE
double gcTheta(double pLat, double pLong, double qLat, double qLong) {
  pLat *= acos(-1.0) / 180.0;
  pLong *= acos(-1.0) / 180.0; // convert degree to radian
  qLat *= acos(-1.0) / 180.0;
  qLong *= acos(-1.0) / 180.0;
  return acos(cos(pLat) * cos(pLong) * cos(qLat) * cos(qLong) +
              cos(pLat) * sin(pLong) * cos(qLat) * sin(qLong) +
              sin(pLat) * sin(qLat));
}
double gcDistance(double pLat, double pLong, double qLat, double qLong, double
    radius) {
  return radius * gcTheta(pLat, pLong, qLat, qLong);
}
```

3.3 lines

```cpp
#include "basics.cpp"
// WARNING: all distance functions are not realizing sqrt operation
// Suggestion: for line intersections check line_line_intersection and then use
// compute_line_intersection
point project_point_line(point c, point a, point b) {
  ld r = dot(b - a, b - a);
  if (fabs(r) < EPS) return a;
  return a + (b - a) * dot(c - a, b - a) / dot(b - a, b - a);
}
point project_point_ray(point c, point a, point b) {
  ld r = dot(b - a, b - a);
  if (fabs(r) < EPS) return a;
  r = dot(c - a, b - a) / r;
  if (le(r, 0)) return a;
  return a + (b - a) * r;
}
point project_point_segment(point c, point a, point b) {
  ld r = dot(b - a, b - a);
  if (fabs(r) < EPS) return a;
  r = dot(c - a, b - a) / r;
  if (le(r, 0)) return a;
  if (ge(r, 1)) return b;
  return a + (b - a) * r;
}
ld distance_point_line(point c, point a, point b) {
  return c.dist2(project_point_line(c, a, b));
}
ld distance_point_ray(point c, point a, point b) {
  return c.dist2(project_point_ray(c, a, b));
}
ld distance_point_segment(point c, point a, point b) {
  return c.dist2(project_point_segment(c, a, b));
}
// not tested
ld distance_point_plane(ld x, ld y, ld z, ld a, ld b, ld c, ld d) {
  return fabs(a * x + b * y + c * z - d) / sqrt(a * a + b * b + c * c);
}
bool lines_parallel(point a, point b, point c, point d) {
  return fabs(cross(b - a, d - c)) < EPS;
}
bool lines_collinear(point a, point b, point c, point d) {
  return lines_parallel(a, b, c, d) && fabs(cross(a - b, a - c)) < EPS &&
         fabs(cross(c - d, c - a)) < EPS;
```

```
}
point lines_intersect(point p, point q, point a, point b) {
  point r = q - p, s = b - a, c(p % q, a % b);
  if (eq(r % s, 0)) return point(LINF, LINF);
  return point(point(r.x, s.x) % c, point(r.y, s.y) % c) / (r % s);
}
// be careful: test line_line_intersection before using this function
point compute_line_intersection(point a, point b, point c, point d) {
  b = b - a;
  d = c - d;
  c = c - a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b * cross(c, d) / cross(b, d);
}
bool line_line_intersect(point a, point b, point c, point d) {
  if (!lines_parallel(a, b, c, d)) return 1;
  if (lines_collinear(a, b, c, d)) return 1;
  return 0;
}
// rays in direction a -> b, c -> d
bool ray_ray_intersect(point a, point b, point c, point d) {
  if (a.dist2(c) < EPS || a.dist2(d) < EPS || b.dist2(c) < EPS ||
      b.dist2(d) < EPS) return 1;
  if (lines_collinear(a, b, c, d)) {
    if (ge(dot(b - a, d - c), 0)) return 1;
    if (ge(dot(a - c, d - c), 0)) return 1;
    return 0;
  }
  if (!line_line_intersect(a, b, c, d)) return 0;
  point inters = lines_intersect(a, b, c, d);
  if (ge(dot(inters - c, d - c), 0) && ge(dot(inters - a, b - a), 0)) return 1;
  return 0;
}
bool segment_segment_intersect(point a, point b, point c, point d) {
  if (a.dist2(c) < EPS || a.dist2(d) < EPS || b.dist2(c) < EPS ||
      b.dist2(d) < EPS)
    return 1;
  int d1, d2, d3, d4;
  d1 = direction(a, b, c);
  d2 = direction(a, b, d);
  d3 = direction(c, d, a);
  d4 = direction(c, d, b);
  if (d1 * d2 < 0 and d3 * d4 < 0) return 1;
  return a.on_seg(c, d) or b.on_seg(c, d) or c.on_seg(a, b) or d.on_seg(a, b);
}
bool segment_line_intersect(point a, point b, point c, point d) {
  if (!line_line_intersect(a, b, c, d)) return 0;
  point inters = lines_intersect(a, b, c, d);
  if (inters.on_seg(a, b)) return 1;
  return 0;
}
// ray in direction c -> d
bool segment_ray_intersect(point a, point b, point c, point d) {
  if (a.dist2(c) < EPS || a.dist2(d) < EPS || b.dist2(c) < EPS ||
      b.dist2(d) < EPS)
    return 1;
  if (lines_collinear(a, b, c, d)) {
    if (c.on_seg(a, b)) return 1;
    if (ge(dot(d - c, a - c), 0)) return 1;
    return 0;
  }
  if (!line_line_intersect(a, b, c, d)) return 0;
  point inters = lines_intersect(a, b, c, d);
  if (!inters.on_seg(a, b)) return 0;
  if (ge(dot(inters - c, d - c), 0)) return 1;
  return 0;
}
// ray in direction a -> b
bool ray_line_intersect(point a, point b, point c, point d) {
  if (a.dist2(c) < EPS || a.dist2(d) < EPS || b.dist2(c) < EPS ||
      b.dist2(d) < EPS)
    return 1;
  if (!line_line_intersect(a, b, c, d)) return 0;
  point inters = lines_intersect(a, b, c, d);
  if (!line_line_intersect(a, b, c, d)) return 0;
  if (ge(dot(inters - a, b - a), 0)) return 1;
  return 0;
}
ld distance_segment_line(point a, point b, point c, point d) {
  if (segment_line_intersect(a, b, c, d)) return 0;
  return min(distance_point_line(a, c, d), distance_point_line(b, c, d));
}
ld distance_segment_ray(point a, point b, point c, point d) {
  if (segment_ray_intersect(a, b, c, d)) return 0;
  ld min1 = distance_point_segment(c, a, b);
  ld min2 = min(distance_point_ray(a, c, d), distance_point_ray(b, c, d));
  return min(min1, min2);
}
```

```
ld distance_segment_segment(point a, point b, point c, point d) {
  if (segment_segment_intersect(a, b, c, d)) return 0;
  ld min1 =
     min(distance_point_segment(c, a, b), distance_point_segment(d, a, b));
  ld min2 =
     min(distance_point_segment(a, c, d), distance_point_segment(b, c, d));
  return min(min1, min2);
}
ld distance_ray_line(point a, point b, point c, point d) {
  if (ray_line_intersect(a, b, c, d)) return 0;
  ld min1 = distance_point_line(a, c, d);
  return min1;
}
ld distance_ray_ray(point a, point b, point c, point d) {
  if (ray_ray_intersect(a, b, c, d)) return 0;
  ld min1 = min(distance_point_ray(c, a, b), distance_point_ray(a, c, d));
  return min1;
}
ld distance_line_line(point a, point b, point c, point d) {
  if (line_line_intersect(a, b, c, d)) return 0;
  return distance_point_line(a, c, d);
}
```

4 Graph

4.1 ArtPointsBridges

```
// Articulation points and Bridges O(V+E)
int par[N], art[N], low[N], num[N], ch[N], cnt;
void articulation(int u) {
  low[u] = num[u] = ++cnt;
  for (int v : adj[u]) {
    if (!num[v]) {
      par[v] = u; ch[u]++;
      articulation(v);
      if (low[v] >= num[u]) art[u] = 1;
      if (low[v] > num[u]) { /* u-v bridge */ }
      low[u] = min(low[u], low[v]);
    }
    else if (v != par[u]) low[u] = min(low[u], num[v]);
  }
}

for (int i = 0; i < n; ++i) if (!num[i])
  articulation(i), art[i] = ch[i]>1;
```

4.2 Dinic

```
// Maxflow & MinCut
// index from 0
struct Edge {
  int a,b,cap,flow;
  Edge(int _a, int _b, int _cap, int _flow) : a(_a), b(_b), cap(_cap),
      flow(_flow) {}
};
struct MaxFlow {
  int n, s, t;
  vi d, ptr, q;
  vt<Edge> e;
  vvi g;
  MaxFlow(int _n) : n(_n), d(_n), ptr(_n), q(_n), g(_n) {
    e.clear();
    for (int i = 0; i < n; i++) {
      g[i].clear();
      ptr[i] = 0;
  }}
  void addEdge(int a, int b, int cap) {
    g[a].pb(sz(e));
    e.eb(a,b,cap,0);
    g[b].pb(sz(e));
    e.eb(b,a,0,0);
  }
  int getMaxFlow(int _s, int _t) {
    s = _s; t = _t;
    int flow = 0;
    while(1) {
      if (!bfs()) break;
      fill(all(ptr), 0);
      while (int pushed = dfs(s, inf)) flow += pushed;
    }
    return flow;
  }
private:
```

```
  bool bfs() {
    int qh = 0, qt = 0;
    q[qt++] = s;
    fill(all(d), -1);
    d[s] = 0;
    while (qh < qt && d[t] == -1) {
      int v = q[qh++];
      for (int i = 0; i < sz(g[v]); i++) {
        int id = g[v][i], to = e[id].b;
        if (d[to] == -1 && e[id].flow < e[id].cap) {
          q[qt++] = to;
          d[to] = d[v] + 1;
    }}}
    return d[t] != -1;
  }
  int dfs (int v, int flow) {
    if (!flow) return 0;
    if (v == t) return flow;
    for (; ptr[v] < sz(g[v]); ++ptr[v]) {
      int id = g[v][ptr[v]], to = e[id].b;
      if (d[to] != d[v] + 1) continue;
      int pushed = dfs(to, min(flow, e[id].cap - e[id].flow));
      if (pushed) {
        e[id].flow += pushed;
        e[id^1].flow -= pushed;
        return pushed;
    }}
    return 0;
  }
};
```

4.3 Euler

```
// All even degrees vertices -> Euler cycle
// Exactly two odd degrees vertices -> Euler path
list<int> cyc;
void EulerTour(list<int>::iterator i, int u) {
  for (int j = 0; j < sz(AdjList[u]); j++) {
    ii v = AdjList[u][j];
    if (v.second) {
      v.second = 0;
      for (int k = 0; k < sz(AdjList[v.first]); k++) {
        ii uu = AdjList[v.first][k];
        if (uu.first == u && uu.second) {
          uu.second = 0;
          break;
      }}
      EulerTour(cyc.insert(i, u), v.first);
}}}
// inside main()
//   cyc.clear();
//   EulerTour(cyc.begin(), start);
```

4.4 GomoryHu

```
// - When i -> j is not connected, answer[i][j] = INF
// - Used together with MaxFlowDinic.h
/*
 * Find min cut between every pair of vertices using N max_flow call (instead
    of N^2)
 * Not tested with directed graph
 * Index start from 0
 */
const int MN = /* n_max */;
struct GomoryHu {
  int ok[MN],cap[MN][MN],answer[MN][MN],parent[MN],n;
  MaxFlow flow;
  GomoryHu(int _n) : n(_n), flow(_n) {
    for(int i = 0; i < n; ++i) ok[i] = parent[i] = 0;
    for(int i = 0; i < n; ++i)
      for(int j = 0; j < n; ++j)
        cap[i][j] = 0, answer[i][j] = inf;
  }
  void addEdge(int u, int v, int c) {
    cap[u][v] += c;
  }
  void calc() {
    for(int i = 1; i <= n-1; ++i) {
      flow = MaxFlow(n);
      for(int u = 0; u < n; u++) {
        for(int v = 0; v < n; v++) {
          if (cap[u][v]) flow.addEdge(u, v, cap[u][v]);
      }}
      int f = flow.getMaxFlow(i, parent[i]);
```

```
    bfs(i);
    for(int j = i + 1; j < n; ++j) {
      if (ok[j] && parent[j]==parent[i]) parent[j]=i;
    }
    answer[i][parent[i]] = answer[parent[i]][i] = f;
    for(int j = 0; j < i; ++j)
      answer[i][j]=answer[j][i]=min(f,answer[parent[i]][j]);
  }}
  void bfs(int start) {
    memset(ok,0,sizeof ok);
    queue<int> qu;
    qu.push(start);
    while (!qu.empty()) {
      int u=qu.front(); qu.pop();
      for(int xid = 0; xid < sz(flow.g[u]); ++xid) {
        int id = flow.g[u][xid], v = flow.e[id].b, fl = flow.e[id].flow, c =
            flow.e[id].cap;
        if (!ok[v] && fl < c) {
          ok[v] = 1;
          qu.push(v);
  }}}}
};
//   g.calc()
//   --> g.answer[i][j] = min cut i-j
```

4.5 LCA

```
// L[i] = level
// L[root] = -1
// LCA[0][root] = -1
const int MN = 100111;
int V, LCA[22][MN], L[MN];
ll Rmax[22][MN];
void initLCA () {
  for(int lg = 1; i <= 19;i ++) {
    for(int i = 0; i < V; i++) {
      if (LCA[lg - 1][i] == -1) continue;
      LCA[lg][i] = LCA[lg - 1][LCA[lg - 1][i]];
      Rmax[lg][i] = max (Rmax[lg - 1][LCA[lg - 1][i]], Rmax[lg - 1][i]);
    }
  }
}
ll query (ll a, ll b) {
  ll ret = 0;
  if (L[a] < L[b]) swap (a, b);
  for(int lg = 19; lg >= 0; lg--) {
    if (LCA[lg][a] != -1 && L[LCA[lg][a]] >= L[b]) {
      ret = max(ret, Rmax[lg][a]);
      a = LCA[lg][a];
    }
  }
  if (a == b) return ret;
  for(int lg = 19; lg >= 0; lg--) {
    if (LCA[lg][a] != LCA[lg][b]) {
      ret = max(ret, Rmax[lg][a]);
      ret = max(ret, Rmax[lg][b]);
      a = LCA[lg][a];
      b = LCA[lg][b];
    }
  }
  ret = max(ret, Rmax[0][a]);
  ret = max(ret, Rmax[0][b]);
  return ret;
}
```

4.6 MCBM

```
// Max Cardinality Bipartite Matching (MCBM)
// Max Independent Set (V - MCBM)
// Min Vertex Cover (MCBM)
// - If TLE --> try shuffle edges
// - It should be quite fast, can AC 10^5 vertices
struct Matching {
  int n,iteration;
  vvi ke;
  vi seen,matchL,matchR;
  Matching(int _n) : n(_n), ke(_n), seen(_n, 0), matchL(_n, -1), matchR(_n,
      -1), iteration{0} {}
  void addEdge(int u, int v) {
    ke[u].push_back(v);
  }
  bool dfs(int u) {
    seen[u] = iteration;
    for (int v : ke[u]) {
```

```
      if (matchR[v] < 0) {
        matchR[v] = u;
        matchL[u] = v;
        return 1;
    }}
    for (int v : ke[u]) {
      if (seen[matchR[v]] != iteration && dfs(matchR[v])) {
        matchR[v] = u;
        matchL[u] = v;
        return 1;
    }}
    return 1;
  }
  int match() {
    int res = 0,newMatches = 0;
    do {
      iteration++;
      newMatches = 0;
      for (int u = 0; u < n; u++) {
        if (matchL[u] < 0 && dfs(u)) ++newMatches;
      }
      res += newMatches;
    } while (newMatches > 0);
    return res;
  }
};
// inside main()
// Matching mat(max(left, right));
// mat.addEdge(u, v);
// REP(i,left) { shuffle(mat.ke[i].begin(), mat.ke[i].end(), rng); }
// cout << mat.match() << '\n';
// if (mat.matchL[i] >= 0) {
//   cout << i << ' ' << mat.matchL[i] << '\n';
// }
```

4.7 Tarjan

```
// Tarjan for SCC and Edge Biconnected Componentes - O(n + m)
vi adj[N];
stack<int> st;
bool inSt[N];
int id[N], cmp[N];
int cnt, cmpCnt;
void clear() {
  memset(id, 0, sizeof id);
  cnt = cmpCnt = 0;
}
int tarjan(int n) {
  int low;
  id[n] = low = ++cnt;
  st.push(n), inSt[n] = 1;
  for(auto x : adj[n]) {
    if(id[x] && inSt[x]) low = min(low, id[x]);
    else if(!id[x]) {
      int lowx = tarjan(x);
      if(inSt[x]) low = min(low, lowx);
    }
  }
  if(low == id[n]) {
    while(st.size()) {
      int x = st.top();
      inSt[x] = 0;
      cmp[x] = cmpCnt;
      st.pop();
      if(x == n) break;
    }
    cmpCnt++;
  }
  return low;
}
```

4.8 TreeDiameter

```
// Tree diameter (weighted): farthest u->v
// Index from 0
// Return {length, path}
using ll = long long;
pair<ll, vi> get_diameter(const vt<vii>& g) {
  int n = sz(g);
  vector<ll> dist(n);
  vi parent(n);
  function<void(int, int, ll)> dfs = [&] (int u, int fu, ll cur_dist) {
    dist[u] = cur_dist;
    parent[u] = fu;
```

```
    for (auto [v, cost] : g[u]) if (v != fu) {
      dfs(v, u, cur_dist + cost);
    }
  };
  dfs(0, -1, 0);
  // r = furthest node from root
  int r = max_element(dist.begin(), dist.end()) - dist.begin();
  dfs(r, -1, 0);
  // r->s = longest path
  int s = max_element(dist.begin(), dist.end()) - dist.begin();
  vi path;
  for (int x = s; x >= 0; x = parent[x]) path.pb(x);
  return {dist[s], path};
}
```

4.9 Triangles

```
// Find all cycles of length 3 (a.k.a. triangles)
// Complexity: O(N + M*sqrt(M))
// Index from 0
vt<tuple<int,int,int>> find_all_triangles(int n, vii edges) {
  uniq(edges);
  vi deg(n, 0);
  for (const auto& [u, v] : edges) {
    if (u == v) continue;
    ++deg[u], ++deg[v];
  }
  vvi adj(n);
  for (auto [u, v] : edges) {
    if (u == v) continue;
    if (deg[u] > deg[v] || (deg[u] == deg[v] && u > v)) swap(u, v);
    adj[u].pb(v);
  }
  // If it's too slow, remove vector res and compute answer directly
  vt<tuple<int,int,int>> res;
  vt<bool> good(n, 0);
  for (int i = 0; i < n; i++) {
    for (auto j : adj[i]) good[j] = 1;
    for (auto j : adj[i]) {
      for (auto k : adj[j]) {
        if (good[k]) res.eb(i, j, k);
      }
    }
    for (auto j : adj[i]) good[j] = 0;
  }
  return res;
}
```

5 Math

5.1 Advanced

```
/* Line integral = integral(sqrt(1 + (dy/dx)^2)) dx */

/* Multiplicative Inverse over MOD for all 1..N - 1 < MOD in O(N)
 Only works for prime MOD. If all 1..MOD - 1 needed, use N = MOD */
ll inv[N];
inv[1] = 1;
for(int i = 2; i < N; ++i)
        inv[i] = MOD - (MOD / i) * inv[MOD % i] % MOD;

/* Catalan
 f(n) = sum(f(i) * f(n - i - 1)), i in [0, n - 1] = (2n)! / ((n+1)! * n!) = ...
 If you have any function f(n) (there are many) that follows this sequence
      (0-indexed):
 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900,
      2674440
 than it's the Catalan function */
// Number of correct bracket sequence consisting of n opening and n closing
     brackets.
// The number of rooted full binary trees with n + 1 leaves (vertices are not
     numbered). A rooted binary tree is full if every vertex has either two
     children or no children.
// The number of ways to completely parenthesize n + 1 factors.
// The number of triangulations of a convex polygon with n + 2 sides (i.e. the
     number of partitions of polygon into disjoint triangles by using the
     diagonals).
// The number of ways to connect the 2n points on a circle to form n disjoint
     chords.
// The number of non-isomorphic full binary trees with internal nodes (i.e.
     nodes having at least one son).
// The number of monotonic lattice paths from point (0,0) to point (n,n) in a
     square lattice of size nxn, which do not pass above the main diagonal
```

```
    (i.e. connecting (0,0) to (n,n)).
// Number of permutations of length that can be stack sorted (i.e. it can be
    shown that the rearrangement is stack sorted if and only if there is no
    such index i < j < k, such that ak < ai < aj).
// The number of non-crossing partitions of a set of elements.
// The number of ways to cover the ladder 1..n using n rectangles (The ladder
    consists of n columns, where i column has a height i).

ll cat[N];
cat[0] = 1;
for(int i = 1; i + 1 < N; i++) // needs inv[i + 1] till inv[N - 1]
       cat[i] = 2ll * (2ll * i - 1) * inv[i + 1] % MOD * cat[i - 1] % MOD;

/* Floor(n / i), i = [1, n], has <= 2 * sqrt(n) diff values.
 Proof: i = [1, sqrt(n)] has sqrt(n) diff values.
 For i = [sqrt(n), n] we have that 1 <= n / i <= sqrt(n)
 and thus has <= sqrt(n) diff values.
*/
/* l = first number that has floor(N / l) = x
 r = last number that has floor(N / r) = x
 N / r >= floor(N / l)
 r <= N / floor(N / l)*/
for(int l = 1, r; l <= n; l = r + 1){
       r = n / (n / l);
       // floor(n / i) has the same value for l <= i <= r
}

/* Recurrence using matriz
 h[i + 2] = a1 * h[i + 1] + a0 * h[i]
 [h[i] h[i-1]] = [h[1] h[0]] * [a1 1] ^ (i - 1)
                               [a0 0]         */

/* Fibonacci in O(log(N)) with memoization
 f(0) = f(1) = 1
 f(2*k) = f(k)^2 + f(k - 1)^2
 f(2*k + 1) = f(k)*[f(k) + 2*f(k - 1)] */

/* Wilson's Theorem Extension
 B = b1 * b2 * ... * bm (mod n) = +-1, all bi <= n such that gcd(bi, n) = 1
 if(n <= 4 or n = (odd prime)^k or n = 2 * (odd prime)^k) B = -1; for any k
 else B = 1; */

/* Stirling numbers of the second kind
 S(n, k) = Number of ways to split n numbers into k non-empty sets
 S(n, 1) = S(n, n) = 1
 S(n, k) = k * S(n - 1, k) + S(n - 1, k - 1)
 Sr(n, k) = S(n, k) with at least r numbers in each set
 Sr(n, k) = k * Sr(n - 1, k) + (n - 1) * Sr(n - r, k - 1)
                               (r - 1)
 S(n - d + 1, k - d + 1) = S(n, k) where if indexes i, j belong to the same
     set, then |i - j| >= d */

/* Burnside's Lemma
 |Classes| = 1 / |G| * sum(K ^ C(g)) for each g in G
 G = Different permutations possible
 C(g) = Number of cycles on the permutation g
 K = Number of states for each element
 Different ways to paint a necklace with N beads and K colors:
 G = {(1, 2, ... N), (2, 3, ... N, 1), ... (N, 1, ... N - 1)}
 gi = (i, i + 1, ... i + N), (taking mod N to get it right) i = 1 ... N
 i -> 2i -> 3i ..., Cycles in gi all have size n / gcd(i, n), so C(gi) =
     gcd(i, n)
 Ans = 1 / N * sum(K ^ gcd(i, n)), i = 1 ... N
 (For the brave, you can get to Ans = 1 / N * sum(euler_phi(N / d) * K ^ d), d
     | N) */

/* Mobius Inversion
 Sum of gcd(i, j), 1 <= i, j <= N?
 sum(k->N) k * sum(i->N) sum(j->N) [gcd(i, j) == k], i = a * k, j = b * k
 = sum(k->N) k * sum(a->N/k) sum(b->N/k) [gcd(a, b) == 1]
 = sum(k->N) k * sum(a->N/k) sum(b->N/k) sum(d->N/k) [d | a] * [d | b] * mi(d)
 = sum(k->N) k * sum(d->N/k) mi(d) * floor(N / kd)^2, l = kd, l <= N, k | l, d
     = l / k
 = sum(l->N) floor(N / l)^2 * sum(k|l) k * mi(l / k)
 If f(n) = sum(x|n)(g(x) * h(x)) with g(x) and h(x) multiplicative, than f(n)
     is multiplicative
 Hence, g(l) = sum(k|l) k * mi(l / k) is multiplicative
 = sum(l->N) floor(N / l)^2 * g(l) */
```

5.2 Bignum

```
const int BASE = 10000;
void fix(vi &a) {
  a.pb(0);
  for (int i = 0; i < sz(a) - 1; ++i) {
    a[i + 1] += a[i] / BASE;
```

```
    a[i] %= BASE;
    if (a[i] < 0) {
      a[i] += BASE;
      a[i + 1]--;
  }}
  while (sz(a) >= 2 && a.back() == 0) a.pop_back();
}
vi operator*(const vi &a, const vi &b) {
  vi c(sz(a) + sz(b) + 1);
  for (int i = 0; i < sz(a); ++i)
    for (int j = 0; j < sz(b); ++j) {
      c[i + j] += a[i] * b[j];
      c[i + j + 1] += c[i + j] / BASE;
      c[i + j] %= BASE;
  }
  return fix(c), c;
}
vi to_vi(int x) { // x < Base
  assert(x < BASE);
  return vi(1, x);
}
vi operator+(vi a, const vi &b) {
  a.resize(max(sz(a), sz(b)));
  for (int i = 0; i < sz(b); ++i) a[i] += b[i];
  return fix(a), a;
}
vi operator-(vi a, const vi &b) {
  for (int i = 0; i < sz(b); ++i) a[i] -= b[i];
  return fix(a), a;
}
vi operator*(vi a, int x) { // x < BASE
  assert(x < BASE);
  for (int i = 0; i < sz(a); ++i) a[i] *= x;
  return fix(a), a;
}
bool operator<(const vi &a, const vi &b) {
  if (sz(a) != sz(b)) return sz(a) < sz(b);
  for (int i = sz(a) - 1; i >= 0; i--)
    if (a[i] != b[i]) return a[i] < b[i];
  return false;
}
vi operator/(vi a, int x) { // x < BASE
  assert(x < BASE);
  for (int i = sz(a) - 1, r = 0; i >= 0; --i, r %= x) {
    r = r * BASE + a[i];
    a[i] = r / x;
  }
  return fix(a), a;
}
int operator%(const vi &a, int x) { //x < BASE
  int r = 0;
  for (int i = sz(a) - 1; i >= 0; --i)
    r = (r * BASE + a[i]) % x;
  return r;
}
istream &operator>>(istream &cin, vi &a) {
  string s; cin >> s;
  a.assign(sz(s) / 4 + 1, 0);
  for (int i = 0; i < sz(s); ++i) {
    int x = (sz(s) - 1 - i) / 4; // <- log10(BASE)=4
    a[x] = a[x] * 10 + (s[i] - '0');
  }
  return fix(a), cin;
}
ostream &operator<<(ostream &cout, const vi &a) {
  cout << a.back();
  for (int i = sz(a) - 2; i >= 0; i--)
    cout << setfill('0') << setw(4) << a[i];
  return cout;
}
// vi a, b; cin >> a >> b;
// a = to_vi(x);
// a + b; a * b; a / x; a % x;
// if(a < b) cout << '-' << b - a;
// else cout << a - b;
```

5.3 CRT

```
// return 1:
//   M = LCM(n,m); 0 <= x < M
//   x % n = a, x % m = b
// 0 if no solution
template<typename T>
bool linearCongruences(const vt<T> &a,const vt<T> &b,const vt<T> &m,T &x,T &M)
    {
  int n = sz(a);
```

```
  x = 0; M = 1;
  for (int i = 0; i < n; i++) {
    T a_ = a[i] * M, b_ = b[i] - a[i] * x, m_ = m[i];
    T y, t, g = extgcd<T>(a_, m_, y, t);
    if (b_ % g) return 0;
    b_ /= g; m_ /= g;
    x += M * (y * b_ % m_);
    M *= m_;
  }
  x = (x + M) % M;
  return 1;
}
```

5.4 DiscreteLog

```
// O(sqrt(m))
// Solve c * a^x = b mod(m) for integer x >= 0.
// Return the smallest x possible, or -1 if there is no solution
// If all solutions needed, solve c * a^x = b mod(m) and (a*b) * a^y = b mod(m)
// x + k * (y + 1) for k >= 0 are all solutions
// Works for any integer values of c, a, b and positive m
// Corner Cases:
// 0^x = 1 mod(m) returns x = 0, so you may want to change it to -1
// You also may want to change for 0^x = 0 mod(1) to return x = 1 instead
// We leave it like it is because you might be actually checking for m^x = 0^x
   mod(m)
// which would have x = 0 as the actual solution.
ll discrete_log(ll c, ll a, ll b, ll m){
  c = ((c % m) + m) % m, a = ((a % m) + m) % m, b = ((b % m) + m) % m;
  if(c == b) return 0;
  ll g = __gcd(a, m);
  if(b % g) return -1;
  if(g > 1) {
      ll r = discrete_log(c * a / g, a, b / g, m / g);
      return r + (r >= 0);
  }
  unordered_map<ll, ll> babystep;
  ll n = 1, an = a % m;
  // set n to the ceil of sqrt(m):
  while(n * n < m) n++, an = (an * a) % m;
  // babysteps:
  ll bstep = b;
  for(ll i = 0; i <= n; i++) {
      babystep[bstep] = i;
      bstep = (bstep * a) % m;
  }
  // giantsteps:
  ll gstep = c * an % m;
  for(ll i = 1; i <= n; i++) {
      if(babystep.find(gstep) != babystep.end())
        return n * i - babystep[gstep];
      gstep = (gstep * an) % m;
  }
  return -1;
}
```

5.5 ExtendedEuclid

```
// return x,y and gcd(a,b) such that ax + by = gcd(a,b)
int gcd(int a, int b, int& x, int& y) {
  x = 1, y = 0;
  int x1 = 0, y1 = 1, a1 = a, b1 = b;
  while (b1) {
    int q = a1 / b1;
    tie(x, x1) = make_tuple(x1, x - q * x1);
    tie(y, y1) = make_tuple(y1, y - q * y1);
    tie(a1, b1) = make_tuple(b1, a1 - q * b1);
  }
  return a1;
}
```

5.6 Josephus

```
/* Josephus Problem - It returns the position to be, in order to not die.
   O(n)*/
/* With k=2, for instance, the game begins with 2 being killed and then n+2,
   n+4, ... */
ll josephus(ll n, ll k) {
  if(n==1) return 1;
  else return (josephus(n-1, k)+k-1)%n+1;
}
/* Another Way to compute the last position to be killed - O(d * log n) */
```

```
ll josephus(ll n, ll d) {
  ll K = 1;
  while (K <= (d - 1)*n) K = (d * K + d - 2) / (d - 1);
  return d * n + 1 - K;
}
```

5.7 Mod

```
const int MOD = 1e9 + 7;
ll modadd(ll a, const ll &b) {
    a += b;
    if(a >= MOD) a -= MOD;
    return a;
}
ll modsubtr(const ll &a, const ll &b) {
    return add(a, MOD - b);
}
ll modmul(ll a, const ll &b) {
    ll ret = a * b - M * ll(1.L / MOD * a * b);
    return ret + MOD * (ret < 0) - MOD * (ret >= (ll)MOD);
}
ll moddiv(const ll &a, const ll &b) {
    return mul(a, fpow(b, MOD - 2, MOD));
}
```

5.8 PrimeFactors

```
// Prime factors (up to 9*10^13. For greater see Pollard Rho)
vi factors;
int ind=0, pf = primes[0];
while (pf*pf <= n) {
  while (n%pf == 0) n /= pf, factors.pb(pf);
  pf = primes[++ind];
}
if (n != 1) factors.pb(n);
```

5.9 Sieve

```
// Sieve of Erasthotenes
int p[N]; vi primes;

for (ll i = 2; i < N; ++i) if (!p[i]) {
  for (ll j = i*i; j < N; j+=i) p[j]=1;
  primes.pb(i);
}
```

6 Misc

6.1 Checker

```
mt19937 rd(chrono::steady_clock::now().time_since_epoch().count());
const string NAME = "";
const int NTEST = 100;
ll Rand(ll l, ll r) {
  ll res = 0;
  for(int i = 0; i < 4; i++) res = (res << 15) ^ (rd() & ((1 << 15) - 1));
  return l + res % (r - l + 1)
}
signed main() {
  srand(time(NULL));
  for(int tc = 1; tc <= NTEST; tc++) {
    ofstream inp((NAME + ".txt").c_str());
    // gen code (inp << n << ...)
    inp.close();
    system((NAME + ".exe").c_str());
    system((NAME + "_trau.exe").c_str());
    if(system(("fc " + NAME + ".out " + NAME + ".ans").c_str())) {
      cout << "Test " << tc << ": WRONG\n"; return;
    }
    cout << "Test " << tc << ": CORRECT\n";
  }
}
```

6.2 Template

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

```cpp
using namespace std;
using ll = long long;
template<typename T> using vt = vector<T>;
using vi = vt<int>;
using vvi = vt<vi>;
using ii = pair<int, int>;
using vii = vt<ii>;
template<typename T> using mipq = priority_queue<T, vt<T>, greater<T>>;
    template<typename T> using mapq = priority_queue<T>;
#define sqr(x) ((x)*(x))
#define all(x) begin(x), end(x)
#define rall(x) (x).rbegin(), (x).rend()
#define sz(x) (int)(x).size()
#define debug(x) cerr << #x << " -> " << x << '\n'
#define F first
#define S second
#define fi first
#define se second
#define mp make_pair
#define pb push_back
#define eb emplace_back
const int inf=1e9+7;
const ll infll = 1e18 + 10;
template <typename T> T fgcd(T a, T b) {while(b) swap(b, a %= b); return a;}
ll fpow(ll a, ll b) {ll o = 1; for(;b;b >>= 1) {if(b & 1) o = o * a;a = a *
    a;} return o;}
ll fpow(ll a, ll b, const ll &m) {ll o = 1; a %= m; for(;b;b >>= 1) {if(b & 1)
    o = o * a % m;a = a * a % m;} return o;}
int flog(const int &x) {return 31 - __builtin_clz(x);}
ll flog(const ll &x) {return 63 - __builtin_clzll(x);}
template<class T> void uniq(vt<T> &a) {sort(all(a));a.resize(unique(all(a)) -
    a.begin());}
void setIO(string name) {
  cin.tie(0)->sync_with_stdio(0);
  // if(fopen((name+".txt").c_str(), "r")) freopen((name+".txt").c_str(), "r",
      stdin);
}
signed main() {
  setIO("");
}
```

7 String

7.1 Booths

```cpp
// Booth's Algorithm - Find the lexicographically least rotation of a string
    in O(n)
// rotation: abc|de -> de|abc
string least_rotation(string s) {
  s += s;
  vi f(sz(s), -1);
  int k = 0;
  for (int j = 1; j < sz(s); j++) {
    int i = f[j - k - 1];
    while (i != -1 && s[j] != s[k + i + 1]) {
      if (s[j] < s[k + i + 1]) k = j - i - 1;
      i = f[i];
    }
    if (s[j] != s[k + i + 1]) {
      if (s[j] < s[k]) k = j;
      f[j - k] = -1;
    } else f[j - k] = i + 1;
  }
  return s.substr(k, sz(s) / 2);
}
```

7.2 Hash

```cpp
void hbuild(char a[], int n, ll H[]) {
  for (int i = 1; i <= n; i++)
    H[i] = (H[i - 1] * M[1] + a[i]) % BASE;
}
ll hash_range(ll H[], int L, int R) {
  return (H[R] - H[L - 1] * M[R - L + 1] + 1LL * inf * inf) % inf;
}
// main
M[0] = 1;
M[1] = 2309;
for (int i = 2; i < N; i++)
  M[i] = M[i - 1] * M[1] % BASE;
```

7.3 Manacher

```
// Manacher (Longest Palindromic String) - O(n)
int lps[2*N+5];
char s[N];
int manacher() {
  int n = strlen(s);
  string p (2*n+3, '#');
  p[0] = '^';
  for (int i = 0; i < n; i++) p[2*(i+1)] = s[i];
  p[2*n+2] = '$';
  int k = 0, r = 0, m = 0;
  int l = p.length();
  for (int i = 1; i < l; i++) {
    int o = 2*k - i;
    lps[i] = (r > i) ? min(r-i, lps[o]) : 0;
    while (p[i + 1 + lps[i]] == p[i - 1 - lps[i]]) lps[i]++;
    if (i + lps[i] > r) k = i, r = i + lps[i];
    m = max(m, lps[i]);
  }
  return m;
}
```

7.4 Zfunction

```
// Z-Function - O(n)
// z[i] = max prefix from i
vi zfunction(const string& s){
  vi z (sz(s));
  for (int i = 1, l = 0, r = 0, n = sz(s); i < n; i++) {
    if (i <= r) z[i] = min(z[i-l], r - i + 1);
    while (i + z[i] < n && s[z[i]] == s[z[i] + i]) z[i]++;
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
  return z;
}
```