Contents

1 Strings

1.1 Manacher

```cpp
vector<int> manacher_odd(string s) {
  int n = s.size();
  s = "$" + s + "^";
  vector<int> p(n + 2);
  int l = 1, r = 1;
  for(int i = 1; i <= n; i++) {
    p[i] = max(0, min(r - i, p[l + (r - i)]));
    while(s[i - p[i]] == s[i + p[i]]) p[i]++;
    if(i + p[i] > r) l = i - p[i], r = i + p[i];
  }
  return vector<int>(begin(p) + 1, end(p) - 1);
}
vector<int> manacher(string s) {
  string t;
  for(auto c: s) t += string("#") + c;
  auto res = manacher_odd(t + "#");
  return vector<int>(begin(res) + 1, end(res) - 1);
}
```

1.2 Hashing

```cpp
ll compute_hash(string const& s) {
  const ll p = 31, m = 1e9 + 9;
  ll hash_value = 0, p_pow = 1;
  for (char c : s) {
    hash_value = (hash_value + (c - 'a' + 1) *
        p_pow) % m;
    p_pow = (p_pow * p) % m;
  }
  return hash_value;
}
```

1.3 Rabin Karp

```cpp
vector<ll> rabin_karp(string const& s, string const
    & t) {
  const ll p = 31, m = 1e9 + 9;
  ll S = s.size(), T = t.size();
  vector<ll> p_pow(max(S, T));
  p_pow[0] = 1;
  for (ll i = 1; i < (ll) p_pow.size(); i++) p_pow[
      i] = (p_pow[i-1] * p) % m;
  vector<ll> h(T + 1, 0);
  for (ll i = 0; i < T; i++) h[i + 1] = (h[i] + (t[
      i] - 'a' + 1) * p_pow[i]) % m;
  ll h_s = 0;
  for (ll i = 0; i < S; i++) h_s = (h_s + (s[i] - '
      a' + 1) * p_pow[i]) % m;
  vector<ll> occurences;
  for (ll i = 0; i + S - 1 < T; i++) {
    ll cur_h = (h[i + S] + m - h[i]) % m;
    if (cur_h == h_s * p_pow[i] % m) occurences.
        push_back(i);
  }
  return occurences;
}
```

1.4 Z Function

```cpp
vector<int> z_function(string s) {
  int n = s.size();
  vector<int> z(n);
  for (int i = 1, l = 0, r = 0; i < n; i++) {
    if (i < r) z[i] = min(r - i, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]])
        z[i]++;
    if (i + z[i] > r) {
      l = i;
      r = i + z[i];
    }
  }
  return z;
}
```

1.5 Finding Repetitions

```cpp
vector<int> z_function(string const& s) {
  int n = s.size();
  vector<int> z(n);
  for (int i = 1, l = 0, r = 0; i < n; i++) {
    if (i <= r) z[i] = min(r - i + 1, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]])
        z[i]++;
```

```
    if (i + z[i] - 1 > r) {
      l = i;
      r = i + z[i] - 1;
    }
  }
  return z;
}
int get_z(vector<int> const& z, int i) {
  if (0 <= i && i < (int) z.size()) return z[i];
  else return 0;
}
vector<pair<int, int>> repetitions;
void convert_to_repetitions(int shift, bool left, int cntr, int l, int k1, int k2) {
  for (int l1 = max(1, l - k2); l1 <= min(l, k1); l1++) {
    if (left && l1 == l) break;
    int l2 = l - l1;
    int pos = shift + (left ? cntr - l1 : cntr - l - l1 + 1);
    repetitions.emplace_back(pos, pos + 2 * l - 1);
  }
}
void find_repetitions(string s, int shift = 0) {
  int n = s.size();
  if (n == 1) return;
  int nu = n / 2;
  int nv = n - nu;
  string u = s.substr(0, nu);
  string v = s.substr(nu);
  string ru(u.rbegin(), u.rend());
  string rv(v.rbegin(), v.rend());
  find_repetitions(u, shift);
  find_repetitions(v, shift + nu);
  vector<int> z1 = z_function(ru);
  vector<int> z2 = z_function(v + '#' + u);
  vector<int> z3 = z_function(ru + '#' + rv);
  vector<int> z4 = z_function(v);
  for (int cntr = 0; cntr < n; cntr++) {
    int l, k1, k2;
    if (cntr < nu) {
      l = nu - cntr;
      k1 = get_z(z1, nu - cntr);
      k2 = get_z(z2, nv + 1 + cntr);
    } else {
      l = cntr - nu + 1;
      k1 = get_z(z3, nu + 1 + nv - 1 - (cntr - nu));
      k2 = get_z(z4, (cntr - nu) + 1);
    }
    if (k1 + k2 >= l) convert_to_repetitions(shift, cntr < nu, cntr, l, k1, k2);
  }
}
```

1.6 Longest Common Prefix

```
vector<int> lcp_construction(string const& s, vector<int> const& p) {
  int n = s.size();
  vector<int> rank(n, 0);
  for (int i = 0; i < n; i++) rank[p[i]] = i;
  int k = 0;
  vector<int> lcp(n-1, 0);
  for (int i = 0; i < n; i++) {
    if (rank[i] == n - 1) {
      k = 0;
      continue;
    }
    int j = p[rank[i] + 1];
    while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
    lcp[rank[i]] = k;
    if (k) k--;
  }
  return lcp;
}
```

1.7 Suffix Array

```
vector<int> sort_cyclic_shifts(string const& s) {
  int n = s.size();
  const int alphabet = 256;
  vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
  for (int i = 0; i < n; i++) cnt[s[i]]++;
  for (int i = 1; i < alphabet; i++) cnt[i] += cnt[i - 1];
  for (int i = 0; i < n; i++) p[--cnt[s[i]]] = i;
  c[p[0]] = 0;
  int classes = 1;
  for (int i = 1; i < n; i++) {
    if (s[p[i]] != s[p[i-1]]) classes++;
    c[p[i]] = classes - 1;
  }
  vector<int> pn(n), cn(n);
  for (int h = 0; (1 << h) < n; ++h) {
    for (int i = 0; i < n; i++) {
      pn[i] = p[i] - (1 << h);
      if (pn[i] < 0)
        pn[i] += n;
    }
    fill(cnt.begin(), cnt.begin() + classes, 0);
    for (int i = 0; i < n; i++) cnt[c[pn[i]]]++;
    for (int i = 1; i < classes; i++) cnt[i] += cnt[i - 1];
    for (int i = n-1; i >= 0; i--) p[--cnt[c[pn[i]]]] = pn[i];
    cn[p[0]] = 0;
    classes = 1;
    for (int i = 1; i < n; i++) {
      pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n]};
      pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 << h)) % n]};
      if (cur != prev) ++classes;
      cn[p[i]] = classes - 1;
    }
    c.swap(cn);
  }
  return p;
}
vector<int> build_suff_arr(string s) {
  s += "$";
  vector<int> sorted_shifts = sort_cyclic_shifts(s);
  sorted_shifts.erase(sorted_shifts.begin());
  return sorted_shifts;
}
// compare two substrings
int compare(int i, int j, int l, int k) {
  pair<int, int> a = {c[k][i], c[k][(i + l - (1 << k)) % n]};
  pair<int, int> b = {c[k][j], c[k][(j + l - (1 << k)) % n]};
  return a == b ? 0 : a < b ? -1 : 1;
}
```

1.8 Count Unique Substrings

```
int count_unique_substrings(string const& s) {
  int n = s.size();
  const int p = 31;
  const int m = 1e9 + 9;
  vector<long long> p_pow(n);
  p_pow[0] = 1;
  for (int i = 1; i < n; i++) p_pow[i] = (p_pow[i - 1] * p) % m;
  vector<long long> h(n + 1, 0);
  for (int i = 0; i < n; i++) h[i + 1] = (h[i] + (s[i] - 'a' + 1) * p_pow[i]) % m;
  int cnt = 0;
  for (int l = 1; l <= n; l++) {
    unordered_set<long long> hs;
    for (int i = 0; i <= n - l; i++) {
      long long cur_h = (h[i + l] + m - h[i]) % m;
      cur_h = (cur_h * p_pow[n - i - 1]) % m;
      hs.insert(cur_h);
    }
    cnt += hs.size();
  }
  return cnt;
}
```

1.9 Knuth Morris Pratt

```
vector<ll> prefix_function(string s) {
  ll n = (ll) s.length();
  vector<ll> pi(n);
  for (ll i = 1; i < n; i++) {
    ll j = pi[i - 1];
    while (j > 0 && s[i] != s[j]) j = pi[j - 1];
    if (s[i] == s[j]) j++;
    pi[i] = j;
  }
  return pi;
}
// count occurences
vector<int> ans(n + 1);
for (int i = 0; i < n; i++)
  ans[pi[i]]++;
for (int i = n-1; i > 0; i--)
  ans[pi[i-1]] += ans[i];
for (int i = 0; i <= n; i++)
  ans[i]++;
```

1.10 Group Identical Substrings

```
vector<vector<int>> group_identical_strings(vector<string> const& s) {
  int n = s.size();
  vector<pair<long long, int>> hashes(n);
  for (int i = 0; i < n; i++) hashes[i] = {compute_hash(s[i]), i};
  sort(hashes.begin(), hashes.end());
  vector<vector<int>> groups;
  for (int i = 0; i < n; i++) {
```

```
    if (i == 0 || hashes[i].first != hashes[i - 1].first) groups.emplace_back();
    groups.back().push_back(hashes[i].second);
  }
  return groups;
}
```

2 Geometry

2.1 Nearest Points

```
struct pt {
  ll x, y, id;
};
struct cmp_x {
  bool operator()(const pt & a, const pt & b) const {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
  }
};
struct cmp_y {
  bool operator()(const pt & a, const pt & b) const { return a.y < b.y; }
};
ll n;
vector<pt> a;
double mindist;
pair<ll, ll> best_pair;
void upd_ans(const pt & a, const pt & b) {
  double dist = sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
  if (dist < mindist) {
    mindist = dist;
    best_pair = {a.id, b.id};
  }
}
vector<pt> t;
void rec(ll l, ll r) {
  if (r - l <= 3) {
    for (ll i = l; i < r; ++i)
      for (ll j = i + 1; j < r; ++j)
        upd_ans(a[i], a[j]);
    sort(a.begin() + l, a.begin() + r, cmp_y());
    return;
  }
  ll m = (l + r) >> 1, midx = a[m].x;
  rec(l, m);
  rec(m, r);
  merge(a.begin() + l, a.begin() + m, a.begin() + m, a.begin() + r, t.begin(), cmp_y());
  copy(t.begin(), t.begin() + r - l, a.begin() + l);
  ll tsz = 0;
  for (ll i = l; i < r; ++i) {
    if (abs(a[i].x - midx) < mindist) {
      for (ll j = tsz - 1; j >= 0 && a[i].y - t[j].y < mindist; --j)
        upd_ans(a[i], t[j]);
      t[tsz++] = a[i];
    }
  }
}
t.resize(n);
sort(a.begin(), a.end(), cmp_x());
mindist = 1E20;
rec(0, n);
```

2.2 Minkowski Sum

```
struct pt {
  ll x, y;
  pt operator + (const pt & p) const { return pt{x + p.x, y + p.y}; }
  pt operator - (const pt & p) const { return pt{x - p.x, y - p.y}; }
  ll cross(const pt & p) const { return x * p.y - y * p.x; }
};
void reorder_polygon(vector<pt> & P){
  size_t pos = 0;
  for (size_t i = 1; i < P.size(); i++){
    if (P[i].y < P[pos].y || (P[i].y == P[pos].y && P[i].x < P[pos].x)) pos = i;
  }
  rotate(P.begin(), P.begin() + pos, P.end());
}
vector<pt> minkowski(vector<pt> P, vector<pt> Q){
  // the first vertex must be the lowest
  reorder_polygon(P);
  reorder_polygon(Q);
  // we must ensure cyclic indexing
  P.push_back(P[0]);
  P.push_back(P[1]);
  Q.push_back(Q[0]);
  Q.push_back(Q[1]);
  // main part
  vector<pt> result;
  size_t 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]).cross(Q[j + 1] - Q[j]);
    if (cross >= 0 && i < P.size() - 2) ++i;
    if (cross <= 0 && j < Q.size() - 2) ++j;
  }
  return result;
}
```

2.3 Point In Convex

```
struct pt {
  long long x, y;
  pt() {}
  pt(long long _x, long long _y) : x(_x), y(_y) {}
  pt operator+(const pt &p) const { return pt(x + p.x, y + p.y); }
  pt operator-(const pt &p) const { return pt(x - p.x, y - p.y); }
  long long cross(const pt &p) const { return x * p.y - y * p.x; }
  long long dot(const pt &p) const { return x * p.x + y * p.y; }
  long long cross(const pt &a, const pt &b) const { return (a - *this).cross(b - *this); }
  long long dot(const pt &a, const pt &b) const { return (a - *this).dot(b - *this); }
  long long sqrLen() const { return this->dot(*this); }
};
bool lexComp(const pt &l, const pt &r) { return l.x < r.x || (l.x == r.x && l.y < r.y); }
int sgn(long long val) { return val > 0 ? 1 : (val == 0 ? 0 : -1); }
vector<pt> seq;
pt translation;
int n;
bool pointInTriangle(pt a, pt b, pt c, pt point) {
  long long s1 = abs(a.cross(b, c));
  long long s2 = abs(point.cross(a, b)) + abs(point.cross(b, c)) + abs(point.cross(c, a));
  return s1 == s2;
}
void prepare(vector<pt> &points) {
  n = points.size();
  int pos = 0;
  for (int i = 1; i < n; i++) {
    if (lexComp(points[i], points[pos])) pos = i;
  }
  rotate(points.begin(), points.begin() + pos, 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(pt point) {
  point = point - translation;
  if (seq[0].cross(point) != 0 && sgn(seq[0].cross(point)) != sgn(seq[0].cross(seq[n - 1])))
    return false;
  if (seq[n - 1].cross(point) != 0 && sgn(seq[n - 1].cross(point)) != sgn(seq[n - 1].cross(seq[0])))
    return false;
  if (seq[0].cross(point) == 0)
    return seq[0].sqrLen() >= point.sqrLen();
  int l = 0, r = n - 1;
  while (r - l > 1) {
    int mid = (l + r) / 2;
    int pos = mid;
    if (seq[pos].cross(point) >= 0) l = mid;
    else r = mid;
  }
  int pos = l;
  return pointInTriangle(seq[pos], seq[pos + 1], pt(0, 0), point);
}
```

2.4 Line Sweep

```
const double EPS = 1E-9;
struct pt { double x, y; };
struct seg {
  pt p, q;
  ll id;
  double get_y(double x) const {
    if (abs(p.x - q.x) < EPS) return p.y;
    return p.y + (q.y - p.y) * (x - p.x) / (q.x - p.x);
  }
};
bool intersect1d(double l1, double r1, double l2, double r2) {
  if (l1 > r1) swap(l1, r1);
  if (l2 > r2) swap(l2, r2);
  return max(l1, l2) <= min(r1, r2) + EPS;
}
```

```
ll vec(const pt& a, const pt& b, const pt& c) {
  double s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
  return abs(s) < EPS ? 0 : s > 0 ? +1 : -1;
}
bool intersect(const seg& a, const seg& b) {
  return intersect1d(a.p.x, a.q.x, b.p.x, b.q.x) &&
         intersect1d(a.p.y, a.q.y, b.p.y, b.q.y) &&
         vec(a.p, a.q, b.p) * vec(a.p, a.q, b.q) <= 0 &&
         vec(b.p, b.q, a.p) * vec(b.p, b.q, a.q) <= 0;
}
bool operator<(const seg& a, const seg& b) {
  double x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.x));
  return a.get_y(x) < b.get_y(x) - EPS;
}
struct event {
  double x;
  ll tp, id;
  event() {}
  event(double x, ll tp, ll id) : x(x), tp(tp), id(id) {}
  bool operator<(const event& e) const {
    if (abs(x - e.x) > EPS) return x < e.x;
    return tp > e.tp;
  }
};
set<seg> s;
vector<set<seg>::iterator> where;
set<seg>::iterator prev(set<seg>::iterator it) {
  return it == s.begin() ? s.end() : --it;
}
set<seg>::iterator next(set<seg>::iterator it) {
  return ++it;
}
pair<ll, ll> solve(const vector<seg>& a) {
  ll n = (ll) a.size();
  vector<event> e;
  for (ll i = 0; i < n; ++i) {
    e.push_back(event(min(a[i].p.x, a[i].q.x), +1, i));
    e.push_back(event(max(a[i].p.x, a[i].q.x), -1, i));
  }
  sort(e.begin(), e.end());
  s.clear();
  where.resize(a.size());
  for (size_t i = 0; i < e.size(); ++i) {
    ll id = e[i].id;
    if (e[i].tp == +1) {
      set<seg>::iterator nxt = s.lower_bound(a[id]), prv = prev(nxt);
      if (nxt != s.end() && intersect(*nxt, a[id])) return make_pair(nxt->id, id);
      if (prv != s.end() && intersect(*prv, a[id])) return make_pair(prv->id, id);
      where[id] = s.insert(nxt, a[id]);
    } else {
      set<seg>::iterator nxt = next(where[id]), prv = prev(where[id]);
      if (nxt != s.end() && prv != s.end() && intersect(*nxt, *prv)) return make_pair(prv->id, nxt->id);
      s.erase(where[id]);
    }
  }
  return make_pair(-1, -1);
}
```

2.5 Line Intersection

```
struct pt { double x, y; };
struct line { double a, b, c; };
const double EPS = 1e-9;
double det(double a, double b, double c, double d) { return a*d - b*c; }
bool intersect(line m, line n, pt & res) {
  double zn = det(m.a, m.b, n.a, n.b);
  if (abs(zn) < EPS) return false;
  res.x = -det(m.c, m.b, n.c, n.b) / zn;
  res.y = -det(m.a, m.c, n.a, n.c) / zn;
  return true;
}
bool parallel(line m, line n) { return abs(det(m.a, m.b, n.a, n.b)) < EPS; }
bool equivalent(line m, line n) {
  return abs(det(m.a, m.b, n.a, n.b)) < EPS
      && abs(det(m.a, m.c, n.a, n.c)) < EPS
      && abs(det(m.b, m.c, n.b, n.c)) < EPS;
}
```

2.6 Basic Geometry

```
struct point2d {
  ftype x, y;
  point2d() {}
  point2d(ftype x, ftype y): x(x), y(y) {}
  point2d& operator+=(const point2d &t) {
    x += t.x;
    y += t.y;
    return *this;
  }
  point2d& operator-=(const point2d &t) {
    x -= t.x;
    y -= t.y;
    return *this;
  }
  point2d& operator*=(ftype t) {
    x *= t;
    y *= t;
    return *this;
  }
  point2d& operator/=(ftype t) {
    x /= t;
    y /= t;
    return *this;
  }
  point2d operator+(const point2d &t) const { return point2d(*this) += t; }
  point2d operator-(const point2d &t) const { return point2d(*this) -= t; }
  point2d operator*(ftype t) const { return point2d(*this) *= t; }
  point2d operator/(ftype t) const { return point2d(*this) /= t; }
};
point2d operator*(ftype a, point2d b) { return b * a; }
ftype dot(point2d a, point2d b) { return a.x * b.x + a.y * b.y; }
ftype dot(point3d a, point3d b) { return a.x * b.x + a.y * b.y + a.z * b.z; }
ftype norm(point2d a) { return dot(a, a); }
double abs(point2d a) { return sqrt(norm(a)); }
double proj(point2d a, point2d b) { return dot(a, b) / abs(b); }
double angle(point2d a, point2d b) { return acos(dot(a, b) / abs(a) / abs(b)); }
point3d cross(point3d a, point3d b) { return point3d(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x); }
ftype triple(point3d a, point3d b, point3d c) {
      return dot(a, cross(b, c)); }
ftype cross(point2d a, point2d b) { return a.x * b.y - a.y * b.x; }
point2d intersect(point2d a1, point2d d1, point2d a2, point2d d2) { return a1 + cross(a2 - a1, d2) / cross(d1, d2) * d1; }
point3d intersect(point3d a1, point3d n1, point3d a2, point3d n2, point3d a3, point3d n3) {
  point3d x(n1.x, n2.x, n3.x);
  point3d y(n1.y, n2.y, n3.y);
  point3d z(n1.z, n2.z, n3.z);
  point3d d(dot(a1, n1), dot(a2, n2), dot(a3, n3));
  return point3d(triple(d, y, z), triple(x, d, z), triple(x, y, d)) / triple(n1, n2, n3);
}
```

2.7 Circle Line Intersection

```
double r, a, b, c; // given as input
double x0 = -a * c / (a * a + b * b);
double y0 = -b * c / (a * a + b * b);
if (c * c > r * r * (a * a + b * b) + EPS) {
  puts ("no points");
} else if (abs (c  *c - r * r * (a * a + b * b)) < EPS) {
  puts ("1 point");
  cout << x0 << ' ' << y0 << '\n';
} else {
  double d = r * r - c * c / (a * a + b * b);
  double mult = sqrt (d / (a * a + b * b));
  double ax, ay, bx, by;
  ax = x0 + b * mult;
  bx = x0 - b * mult;
  ay = y0 - a * mult;
  by = y0 + a * mult;
  puts ("2 points");
  cout << ax << ' ' << ay << '\n' << bx << ' ' << by << '\n';
}
```

2.8 Convex Hull

```
struct pt {
  double x, y;
};
ll orientation(pt a, pt b, pt c) {
  double v = a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.y - b.y);
  if (v < 0) {
    return -1;
  } else if (v > 0) {
    return +1;
  }
  return 0;
}
```

```
bool cw(pt a, pt b, pt c, bool include_collinear) {
  ll o = orientation(a, b, c);
  return o < 0 || (include_collinear && o == 0);
}
bool collinear(pt a, pt b, pt c) {
  return orientation(a, b, c) == 0;
}
void convex_hull(vector<pt>& a, bool
     include_collinear = false) {
  pt p0 = *min_element(a.begin(), a.end(), [](pt a,
       pt b) {
    return make_pair(a.y, a.x) < make_pair(b.y, b.x
       );
  });
  sort(a.begin(), a.end(), [&p0](const pt& a, const
       pt& b) {
    ll o = orientation(p0, a, b);
    if (o == 0) {
      return (p0.x - a.x) * (p0.x - a.x) + (p0.y -
           a.y) * (p0.y - a.y)
           < (p0.x - b.x) * (p0.x - b.x) + (p0.y -
               b.y) * (p0.y - b.y);
    }
    return o < 0;
  });
  if (include_collinear) {
    ll i = (ll) a.size()-1;
    while (i >= 0 && collinear(p0, a[i], a.back()))
         i--;
    reverse(a.begin()+i+1, a.end());
  }
  vector<pt> st;
  for (ll i = 0; i < (ll) a.size(); i++) {
    while (st.size() > 1 && !cw(st[st.size() - 2],
         st.back(), a[i], include_collinear)) {
      st.pop_back();
    }
    st.push_back(a[i]);
  }
  a = st;
}
```

2.9 Count Lattices

```
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;
}
```

2.10 Segment Intersection

```
const double EPS = 1E-9;
struct pt {
  double x, y;
  bool operator<(const pt& p) const {
    return x < p.x - EPS || (abs(x - p.x) < EPS &&
         y < p.y - EPS);
  }
};
struct line {
  double a, b, c;
  line() {}
  line(pt p, pt q) {
    a = p.y - q.y;
    b = q.x - p.x;
    c = -a * p.x - b * p.y;
    norm();
  }
  void norm() {
    double z = sqrt(a * a + b * b);
    if (abs(z) > EPS) a /= z, b /= z, c /= z;
  }
  double dist(pt p) const { return a * p.x + b * p.
       y + c; }
};
double det(double a, double b, double c, double d)
     {
  return a * d - b * c;
}
inline bool betw(double l, double r, double x) {
  return min(l, r) <= x + EPS && x <= max(l, r) +
       EPS;
}
inline bool intersect_1d(double a, double b, double
     c, double d) {
  if (a > b) swap(a, b);
  if (c > d) swap(c, d);
  return max(a, c) <= min(b, d) + EPS;
}
bool intersect(pt a, pt b, pt c, pt d, pt& left, pt
     & right) {
  if (!intersect_1d(a.x, b.x, c.x, d.x) || !
       intersect_1d(a.y, b.y, c.y, d.y)) return
       false;
  line m(a, b);
  line n(c, d);
  double zn = det(m.a, m.b, n.a, n.b);
  if (abs(zn) < EPS) {
    if (abs(m.dist(c)) > EPS || abs(n.dist(a)) >
         EPS) return false;
    if (b < a) swap(a, b);
    if (d < c) swap(c, d);
    left = max(a, c);
    right = min(b, d);
    return true;
  } else {
    left.x = right.x = -det(m.c, m.b, n.c, n.b) /
         zn;
    left.y = right.y = -det(m.a, m.c, n.a, n.c) /
         zn;
    return betw(a.x, b.x, left.x) && betw(a.y, b.y,
          left.y) &&
           betw(c.x, d.x, left.x) && betw(c.y, d.y,
                left.y);
  }
}
```

2.11 Areas

```
int signed_area_parallelogram(point2d p1, point2d
     p2, point2d p3) {
  return cross(p2 - p1, p3 - p2);
}
double triangle_area(point2d p1, point2d p2,
     point2d p3) {
  return abs(signed_area_parallelogram(p1, p2, p3))
       / 2.0;
}
bool clockwise(point2d p1, point2d p2, point2d p3)
     {
  return signed_area_parallelogram(p1, p2, p3) < 0;
}
bool counter_clockwise(point2d p1, point2d p2,
     point2d p3) {
  return signed_area_parallelogram(p1, p2, p3) > 0;
}
double area(const vector<point>& fig) {
  double res = 0;
  for (unsigned i = 0; i < fig.size(); i++) {
    point p = i ? fig[i - 1] : fig.back();
    point q = fig[i];
    res += (p.x - q.x) * (p.y + q.y);
  }
  return fabs(res) / 2;
}
```

3 Dynamic Programming

3.1 Knuth Optimization

```
ll solve() {
  ll N;
  ... // Read input
  vector<vector<ll>> dp(N, vector<ll>(N)), opt(N,
       vector<ll>(N));
  auto C = [&](ll i, ll j) {
    ... // Implement cost function C.
  };
  for (ll i = 0; i < N; i++) {
    opt[i][i] = i;
    ... // Initialize dp[i][i] according to the
         problem
  }
  for (ll i = N - 2; i >= 0; i--) {
    for (ll j = i + 1; j < N; j++) {
      ll mn = ll_MAX, cost = C(i, j);
      for (ll 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] << '\n';
}
```

3.2 Knapsack

```
ll knapsack(ll W, vector<ll> &wt, vector<ll> &val,
     ll n) {
  vector<ll> dp(W + 1, 0);
```

```cpp
  for (ll i = 1; i <= n; i++) {
    for (ll w = W; w >= 0; w--) {
      if (wt[i - 1] <= w) {
        dp[w] = max(dp[w], dp[w - wt[i - 1]] + val[i - 1]);
      }
    }
  }
  return dp[W];
}
```

3.3 Divide And Conquer

```cpp
ll m, n;
vector<ll> dp_before(n), dp_cur(n);
ll C(ll i, ll j);
void compute(ll l, ll r, ll optl, ll optr) {
  if (l > r) return;
  ll mid = (l + r) >> 1;
  pair<ll, ll> best = {LLONG_MAX, -1};
  for (ll 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;
  ll opt = best.second;
  compute(l, mid - 1, optl, opt);
  compute(mid + 1, r, opt, optr);
}
ll solve() {
  for (ll i = 0; i < n; i++) dp_before[i] = C(0, i);
  for (ll i = 1; i < m; i++) {
    compute(0, n - 1, 0, n - 1);
    dp_before = dp_cur;
  }
  return dp_before[n - 1];
}
```

3.4 Digit Dp

```cpp
vector<vector<vector<vector<ll>>>> dp(K + 1, vector<vector<vector<ll>>>(9 * K + 1, vector<vector<ll>>(9 * K + 1, vector<ll>(9 * K, 0))));
for (ll n = 1; n <= 9 * K; n++) dp[0][n][0][0] = 1;
ll pow10 = 1;
for (ll k = 1; k <= K; k++) {
  for (ll n = 1; n <= 9 * K; n++) {
    for (ll s = 0; s <= 9 * K; s++) {
      for (ll m = 0; m < n; m++) {
        for (ll y = 0; y <= 9; y++) {
          if (s >= y) dp[k][n][s][m] += dp[k - 1][n][s - y][((m - y * pow10) % n + n) % n];
        }
      }
    }
  }
  pow10 *= 10;
}
string N;
cin >> N;
ll n = N.length(), ans = 0;
vector<ll> g(9 * K + 1, 0);
for (ll s = 1; s <= 9 * K; s++) {
  string substring = "";
  ll pow10 = 1;
  for (ll i = 0; i < n - 1; i++) pow10 *= 10;
  for (ll i = 0; i < n; i++) {
    substring += '0';
    for (ll j = 0; j < N[i] - '0'; j++) {
      ll digit_sum = j;
      for (ll k = 0; k < i; k++) digit_sum += substring[k] - '0';
      if (s >= digit_sum) g[s] += dp[n - 1 - i][s][s - digit_sum][((-pow10 * stoll(substring)) % s + s) % s];
      substring[i]++;
    }
    pow10 /= 10;
  }
  ans += g[s];
}
auto is_good = [&](string s) -> bool {
  ll digit_sum = 0;
  for (ll i = 0; i < (ll) s.length(); i++) digit_sum += s[i] - '0';
  return stoll(s) % digit_sum == 0;
};
if (is_good(N)) ans++;
cout << ans << "\n";
```

3.5 Subset Sum

```cpp
bool subset_sum(ll n, vector<ll> &arr, ll sum) {
  vector<vector<ll>> dp(n + 1, vector<ll>(sum + 1, false));
  dp[0][0] = true;
  for (ll i = 1; i <= n; i++) {
    for (ll j = 0; j <= sum; j++) {
      dp[i][j] = dp[i - 1][j];
      if (j >= arr[i]) {
        dp[i][j] |= dp[i - 1][j - arr[i]];
      }
    }
  }
  return dp[n][sum];
}
```

3.6 Longest Increasing Subsequence

```cpp
ll get_ceil_idx(vector<ll> &a, vector<ll> &T, ll l, ll r, ll x) {
  while (r - l > 1) {
    ll m = l + (r - l) / 2;
    if (a[T[m]] >= x) {
      r = m;
    } else {
      l = m;
    }
  }
  return r;
}
ll LIS(ll n, vector<ll> &a) {
  ll len = 1;
  vector<ll> T(n, 0), R(n, - 1);
  T[0] = 0;
  for (ll i = 1; i < n; i++) {
    if (a[i] < a[T[0]]) {
      T[0] = i;
    } else if (a[i] > a[T[len - 1]]) {
      R[i] = T[len - 1];
      T[len++] = i;
    } else {
      ll pos = get_ceil_idx(a, T, -1, len - 1, a[i]);
      R[i] = T[pos - 1];
      T[pos] = i;
    }
  }
  return len;
}
```

3.7 Longest Common Subsequence

```cpp
ll LCS(string x, string y, ll n, ll m) {
  vector<vector<ll>> dp(n + 1, vector<ll>(m + 1));
  for (ll i = 0; i <= n; i++) {
    for (ll j = 0; j <= m; j++) {
      if (i == 0 || j == 0) {
        dp[i][j] = 0;
      } else if (x[i - 1] == y[j - 1]) {
        dp[i][j] = dp[i - 1][j - 1] + 1;
      } else {
        dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
      }
    }
  }
  ll index = dp[n][m];
  vector<char> lcs(index + 1);
  lcs[index] = '\0';
  ll i = n, j = m;
  while (i > 0 && j > 0) {
    if (x[i - 1] == y[j - 1]) {
      lcs[index - 1] = x[i - 1];
      i--;
      j--;
      index--;
    } else if (dp[i - 1][j] > dp[i][j - 1]) {
      i--;
    } else {
      j--;
    }
  }
  return dp[n][m];
}
```

3.8 Max Sum

```cpp
int max_subarray_sum(vi arr) {
  int x = 0, s = 0;
  for (int k = 0; k < n; k++) {
  s = max(arr[k],s+arr[k]);
  x = max(x,s);
  }
  return x;
}
```

3.9 Bitmask Weights

```cpp
vector<pair<ll, ll>> dp(1 << n, {INF, 0});
dp[0] = {1, 0};
for (ll mask = 1; mask < (1 << n); mask++)
```

```
  for (ll i = 0; i < n; i++)
    if (mask & (1 << i)) {
      ll prev_mask = mask ^ (1 << i);
      auto [best, last_sum] = dp[prev_mask];
      if (last_sum + w[i] <= x) {
        dp[mask] = min(dp[mask], {best, last_sum + w[i]});
      } else {
        dp[mask] = min(dp[mask], {best + 1, w[i]});
      }
    }
cout << dp[(1 << n) - 1].first << "\n";
```

3.10 Edit Distance

```
ll edit_distance(string x, string y, ll n, ll m) {
  vector<vector<int>> dp(n + 1, vector<int>(m + 1, INF));
  dp[0][0] = 0;
  for (int i = 1; i <= n; i++) {
    dp[i][0] = i;
  }
  for (int j = 1; j <= m; j++) {
    dp[0][j] = j;
  }
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
      dp[i][j] = min({dp[i - 1][j] + 1, dp[i][j - 1] + 1, dp[i - 1][j - 1] + (x[i - 1] != y[j - 1])});
    }
  }
  return dp[n][m];
}
```

4 Math

4.1 Chinese Remainder Theorem

```
struct Congruence {
  ll a, m;
};

ll chinese_remainder_theorem(vector<Congruence> const& congruences) {
  ll M = 1;
  for (auto const& congruence : congruences) M *= congruence.m;
  ll solution = 0;
  for (auto const& congruence : congruences) {
    ll a_i = congruence.a;
    ll M_i = M / congruence.m;
    ll N_i = mod_inv(M_i, congruence.m);
    solution = (solution + a_i * M_i % M * N_i) % M;
  }
  return solution;
}
```

4.2 Extended Euclidean

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

4.3 Modulo Inverse

```
ll mod_inv(ll a, ll m) {
  if (m == 1) return 0;
  ll m0 = m, x = 1, y = 0;
  while (a > 1) {
    ll q = a / m, t = m;
    m = a % m;
    a = t;
    t = y;
    y = x - q * y;
    x = t;
  }
  if (x < 0) x += m0;
  return x;
}
```

4.4 Sum Of Divisors

```
ll sum_of_divisors(ll num) {
  ll total = 1;
  for (int i = 2; (ll)i * i <= num; i++) {
    if (num % i == 0) {
      int e = 0;
      do {
        e++;
        num /= i;
      } while (num % i == 0);
      ll sum = 0, pow = 1;
      do {
        sum += pow;
        pow *= i;
      } while (e-- > 0);
      total *= sum;
    }
  }
  if (num > 1) total *= (1 + num);
  return total;
}
```

4.5 Range Sieve

```
vector<bool> range_sieve(ll l, ll r) {
  ll n = sqrt(r);
  vector<bool> is_prime(n + 1, true);
  vector<ll> prime;
  is_prime[0] = is_prime[1] = false;
  prime.push_back(2);
  for (ll i = 4; i <= n; i += 2) is_prime[i] = false;
  for (ll i = 3; i <= n; i += 2) {
    if (is_prime[i]) {
      prime.push_back(i);
      for (ll j = i * i; j <= n; j += i) is_prime[j] = false;
    }
  }
  vector<bool> result(r - l + 1, true);
  for (ll i : prime)
    for (ll j = max(i * i, (l + i - 1) / i * i); j <= r; j += i)
      result[j - l] = false;
  if (l == 1) result[0] = false;
  return result;
}
```

4.6 Pollard Rho Brent

```
ll mult(ll a, ll b, ll mod) {
  return (__int128_t) a * b % mod;
}
ll f(ll x, ll c, ll mod) {
  return (mult(x, x, mod) + c) % mod;
}
ll pollard_rho_brent(ll n, ll x0 = 2, ll c = 1) {
  ll x = x0, g = 1, q = 1, xs, y, m = 128, l = 1;
  while (g == 1) {
    y = x;
    for (ll i = 1; i < l; i++) x = f(x, c, n);
    ll k = 0;
    while (k < l && g == 1) {
      xs = x;
      for (ll i = 0; i < m && i < l - k; i++) {
        x = f(x, c, n);
        q = mult(q, abs(y - x), n);
      }
      g = __gcd(q, n);
      k += m;
    }
    l *= 2;
  }
  if (g == n) {
    do {
      xs = f(xs, c, n);
      g = __gcd(abs(xs - y), n);
    } while (g == 1);
  }
  return g;
}
```

4.7 Factorial Modulo

```
int factmod(int n, int p) {
  vector<int> f(p);
  f[0] = 1;
  for (int i = 1; i < p; i++) f[i] = f[i - 1] * i % p;
  int res = 1;
  while (n > 1) {
    if ((n / p) % 2) res = p - res;
    res = res * f[n % p] % p;
    n /= p;
  }
  return res;
}
```

4.8 Matrix

```
/*
Matrix exponentation:
f[n] = af[n-1] + bf[n-2] + cf[n-3]
Use:
|f[n]  | |a b c||f[n-1]|
|f[n-1]|=|1 0 0||f[n-2]|
|f[n-2]| |0 1 0||f[n-3]|
To get:
|f[n]  | |a b c|^(n-2)|f[2]|
|f[n-1]|=|1 0 0|      |f[1]|
|f[n-2]| |0 1 0|      |f[0]|
*/
struct Matrix { int mat[MAX_N][MAX_N]; };
Matrix matrix_mul(Matrix a, Matrix b) {
  Matrix ans; int i, j, k;
  for (i = 0; i < MAX_N; i++)
  for (j = 0; j < MAX_N; j++)
  for (ans.mat[i][j] = k = 0; k < MAX_N; k++)
    ans.mat[i][j] += a.mat[i][k] * b.mat[k][j];
  return ans;
}
Matrix matrix_pow(Matrix base, int p) {
  Matrix ans; int i, j;
  for (i = 0; i < MAX_N; i++)
    for (j = 0; j < MAX_N; j++)
      ans.mat[i][j] = (i == j);
  while (p) {
    if (p & 1) ans = matrix_mul(ans, base);
    base = matrix_mul(base, base);
    p >>= 1;
  }
  return ans;
}
```

4.9 Find All Solutions

```
bool find_any_solution(ll a, ll b, ll c, ll &x0, ll &y0, ll &g) {
  g = gcd_extended(abs(a), abs(b), x0, y0);
  if (c % g) return false;
  x0 *= c / g;
  y0 *= c / g;
  if (a < 0) x0 = -x0;
  if (b < 0) y0 = -y0;
  return true;
}
void shift_solution(ll & x, ll & y, ll a, ll b, ll cnt) {
  x += cnt * b;
  y -= cnt * a;
}
ll find_all_solutions(ll a, ll b, ll c, ll minx, ll maxx, ll miny, ll maxy) {
  ll x, y, g;
  if (!find_any_solution(a, b, c, x, y, g)) return 0;
  a /= g;
  b /= g;
  ll sign_a = a > 0 ? +1 : -1;
  ll sign_b = b > 0 ? +1 : -1;
  shift_solution(x, y, a, b, (minx - x) / b);
  if (x < minx) shift_solution(x, y, a, b, sign_b);
  if (x > maxx) return 0;
  ll lx1 = x;
  shift_solution(x, y, a, b, (maxx - x) / b);
  if (x > maxx) shift_solution(x, y, a, b, -sign_b);
  ll rx1 = x;
  shift_solution(x, y, a, b, -(miny - y) / a);
  if (y < miny) shift_solution(x, y, a, b, -sign_a);
  if (y > maxy) return 0;
  ll lx2 = x;
  shift_solution(x, y, a, b, -(maxy - y) / a);
  if (y > maxy) shift_solution(x, y, a, b, sign_a);
  ll rx2 = x;
  if (lx2 > rx2) swap(lx2, rx2);
  ll lx = max(lx1, lx2), rx = min(rx1, rx2);
  if (lx > rx) return 0;
  return (rx - lx) / abs(b) + 1;
}
```

4.10 Miller Rabin

```
using u64 = uint64_t;
using u128 = __uint128_t;
u64 binpower(u64 base, u64 e, u64 mod) {
  u64 result = 1;
  base %= mod;
  while (e) {
    if (e & 1) result = (u128) result * base % mod;
    base = (u128) base * base % mod;
    e >>= 1;
  }
  return result;
}
bool check_composite(u64 n, u64 a, u64 d, ll s) {
  u64 x = binpower(a, d, n);
  if (x == 1 || x == n - 1) return false;
  for (ll r = 1; r < s; r++) {
    x = (u128) x * x % n;
    if (x == n - 1) return false;
  }
  return true;
}
bool miller_rabin(u64 n) {
  if (n < 2) return false;
  ll r = 0;
  u64 d = n - 1;
  while ((d & 1) == 0) {
    d >>= 1;
    r++;
  }
  for (ll a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
    if (n == a) return true;
    if (check_composite(n, a, d, r)) return false;
  }
  return true;
}
```

4.11 Fibonacci

```
/*
Properties:
- Cassini's identity: f[n-1]f[n+1] - f[n]^2 = (-1)^n
- d'Ocagne's identity: f[m]f[n+1] - f[m+1]f[n] = (-1)^n f[m-n]
- Addition rule: f[n+k] = f[k]f[n+1] + f[k-1]f[n]
- k = n case: f[2n] = f[n](f[n+1] + f[n-1])
- f[n] | f[nk]
- f[n] | f[m] => n | m
- GCD rule: gcd(f[m], f[n]) = f[gcd(m, n)]
- [[1 1], [1 0]]^n = [[f[n+1] f[n]], [f[n], f[n-1]]]
- f[2k+1] = f[k+1]^2 + f[k]^2
- f[2k] = f[k](f[k+1] + f[k-1]) = f[k](2f[k+1] - f[k])
- Periodic sequence modulo p
- sum[i=1..n]f[i] = f[n+2] - 1
- sum[i=0..n-1]f[2i+1] = f[2n]
- sum[i=1..n]f[2i] = f[2n+1] - 1
- sum[i=1..n]f[i]^2 = f[n]f[n+1]
Fibonacci encoding:
1. Iterate through the Fibonacci numbers from the largest to the
smallest until you find one less than or equal to n.
2. Suppose this number was F_i. Subtract F_i from n and put a 1
in the i-2 position of the code word (indexing from 0 from the
leftmost to the rightmost bit).
3. Repeat until there is no remainder.
4. Add a final 1 to the codeword to indicate its end.
Closed-form: f[n] = (((1 + rt(5))/2)^n - ((1 - rt(5)) / 2)^n)/rt(5)
*/

struct matrix {
    ll mat[2][2];
    matrix friend operator *(const matrix &a, const matrix &b){
      matrix c;
      for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
          c.mat[i][j] = 0;
          for (int k = 0; k < 2; k++) c.mat[i][j] += a.mat[i][k] * b.mat[k][j];
        }
      }
      return c;
    }
  };
  matrix matpow(matrix base, ll n) {
    matrix ans{ {
      {1, 0},
      {0, 1}
    } };
    while (n) {
      if (n & 1) ans = ans * base;
      base = base * base;
      n >>= 1;
    }
    return ans;
  }
  ll fib(int n) {
    matrix base{ {
      {1, 1},
      {1, 0}
    } };
    return matpow(base, n).mat[0][1];
  }
  pair<int, int> fib (int n) {
```

```
    if (n == 0) return {0, 1};
    auto p = fib(n >> 1);
    int c = p.first * (2 * p.second - p.first);
    int d = p.first * p.first + p.second * p.second;
    if (n & 1) return {d, c + d};
    else return {c, d};
  }
}
```

4.12 Fast Fourier Transform

```
using cd = complex<double>;
const double PI = acos(-1);
void fft(vector<cd>& a, bool invert) {
  int n = a.size();
  if (n == 1) return;
  vector<cd> a0(n / 2), a1(n / 2);
  for (int i = 0; 2 * i < n; i++) {
    a0[i] = a[2 * i];
    a1[i] = a[2 * i + 1];
  }
  fft(a0, invert);
  fft(a1, invert);
  double ang = 2 * PI / n * (invert ? -1 : 1);
  cd w(1), wn(cos(ang), sin(ang));
  for (int i = 0; 2 * i < n; i++) {
    a[i] = a0[i] + w * a1[i];
    a[i + n / 2] = a0[i] - w * a1[i];
    if (invert) {
      a[i] /= 2;
      a[i + n / 2] /= 2;
    }
    w *= wn;
  }
}
vector<int> multiply(vector<int> const& a, vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size()) n <<= 1;
    fa.resize(n);
    fb.resize(n);
    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++) fa[i] *= fb[i];
    fft(fa, true);
    vector<int> result(n);
    for (int i = 0; i < n; i++) result[i] = round(fa[i].real());
    return result;
}
```

4.13 Segmented Sieve

```
vector<ll> segmented_sieve(ll n) {
  const ll S = 10000;
  ll nsqrt = sqrt(n);
  vector<char> is_prime(nsqrt + 1, true);
  vector<ll> prime;
  is_prime[0] = is_prime[1] = false;
  prime.push_back(2);
  for (ll i = 4; i <= nsqrt; i += 2) {
    is_prime[i] = false;
  }
  for (ll i = 3; i <= nsqrt; i += 2) {
    if (is_prime[i]) {
      prime.push_back(i);
      for (ll j = i * i; j <= nsqrt; j += i) {
        is_prime[j] = false;
      }
    }
  }
  vector<ll> result;
  vector<char> block(S);
  for (ll k = 0; k * S <= n; k++) {
    fill(block.begin(), block.end(), true);
    for (ll p : prime) {
      for (ll j = max((k * S + p - 1) / p, p) * p - k * S; j < S; j += p) {
        block[j] = false;
      }
    }
    if (k == 0) {
      block[0] = block[1] = false;
    }
    for (ll i = 0; i < S && k * S + i <= n; i++) {
      if (block[i]) {
        result.push_back(k * S + i);
      }
    }
  }
  return result;
}
```

4.14 Linear Sieve

```
void linear_sieve(ll N, vector<ll> &lowest_prime, vector<ll> &prime) {
  for (ll i = 2; i <= N; i++) {
    if (lowest_prime[i] == 0) {
      lowest_prime[i] = i;
      prime.push_back(i);
    }
    for (ll j = 0; i * prime[j] <= N; j++) {
      lowest_prime[i * prime[j]] = prime[j];
      if (prime[j] == lowest_prime[i]) break;
    }
  }
}
```

4.15 Tonelli Shanks

```
ll legendre(ll a, ll p) {
  return bin_pow_mod(a, (p - 1) / 2, p);
}
ll tonelli_shanks(ll n, ll p) {
  if (legendre(n, p) == p - 1) {
    return -1;
  }
  if (p % 4 == 3) {
    return bin_pow_mod(n, (p + 1) / 4, p);
  }
  ll Q = p - 1, S = 0;
  while (Q % 2 == 0) {
    Q /= 2;
    S++;
  }
  ll z = 2;
  for (; z < p; z++) {
    if (legendre(z, p) == p - 1) {
      break;
    }
  }
  ll M = S, c = bin_pow_mod(z, Q, p), t = bin_pow_mod(n, Q, p), R = bin_pow_mod(n, (Q + 1) / 2, p);
  while (t % p != 1) {
    if (t % p == 0) {
      return 0;
    }
    ll i = 1, t2 = t * t % p;
    for (; i < M; i++) {
      if (t2 % p == 1) {
        break;
      }
      t2 = t2 * t2 % p;
    }
    ll b = bin_pow_mod(c, bin_pow_mod(2, M - i - 1, p), p);
    M = i;
    c = b * b % p;
    t = t * c % p;
    R = R * b % p;
  }
  return R;
}
```

5 Miscellaneous

5.1 Techniques

```
/*
Dynamic Programming
- Bitmask
- Range
- Digit
- Knapsack
Graph Theory
- Tree diameter
- Reversing edges
- Tree re-rooting
- DP on trees
- DFS tree
- Euler tour
- Binary Jumping
- Centroid
- DAG
- Condense
Data Structures
- Multiple information
- Binary searching on the tree
- 2D range query
- SQRT decomposition
- Small-to-large
Sorting and searching
- Sliding window
- Two pointers
- Binary search on the answer
*/
```

5.2 Gauss

```cpp
const double EPS = 1e-9;
const ll INF = 2;
ll gauss(vector <vector<double>> a, vector<double> &ans) {
  ll n = (ll) a.size(), m = (ll) a[0].size() - 1;
  vector<ll> where (m, -1);
  for (ll col = 0, row = 0; col < m && row < n; ++col) {
    ll sel = row;
    for (ll i = row; i < n; ++i) {
      if (abs(a[i][col]) > abs(a[sel][col])) {
        sel = i;
      }
    }
    if (abs (a[sel][col]) < EPS) {
      continue;
    }
    for (ll i = col; i <= m; ++i) {
      swap(a[sel][i], a[row][i]);
    }
    where[col] = row;
    for (ll i = 0; i < n; ++i) {
      if (i != row) {
        double c = a[i][col] / a[row][col];
        for (ll j = col; j <= m; ++j) {
          a[i][j] -= a[row][j] * c;
        }
      }
    }
    ++row;
  }
  ans.assign(m, 0);
  for (ll i = 0; i < m; ++i) {
    if (where[i] != -1) {
      ans[i] = a[where[i]][m] / a[where[i]][i];
    }
  }
  for (ll i = 0; i < n; ++i) {
    double sum = 0;
    for (ll j = 0; j < m; ++j) {
      sum += ans[j] * a[i][j];
    }
    if (abs (sum - a[i][m]) > EPS) {
      return 0;
    }
  }
  for (ll i = 0; i < m; ++i) {
    if (where[i] == -1) {
      return INF;
    }
  }
  return 1;
}
```

5.3 Ternary Search

```cpp
double ternary_search(double l, double r) {
  double eps = 1e-9;
  while (r - l > eps) {
    double m1 = l + (r - l) / 3;
    double m2 = r - (r - l) / 3;
    double f1 = f(m1);
    double f2 = f(m2);
    if (f1 < f2) {
      l = m1;
    } else {
      r = m2;
    }
  }
  return f(l);
}
```

6 Data Structures

6.1 Segment Tree 2d

```cpp
template<typename T, typename InType = T>
class SegTree2dNode {
public:
  int i, j, tree_size;
  SegTree<T, InType>* seg_tree;
  SegTree2dNode<T, InType>* lc, * rc;
  SegTree2dNode() {}
  SegTree2dNode(const vector<vector<InType>>& a, int i, int j) : i(i), j(j) {
    tree_size = a[0].size();
    if (j - i == 1) {
      lc = rc = nullptr;
      seg_tree = new SegTree<T, InType>(a[i]);
      return;
    }
    int k = (i + j) / 2;
    lc = new SegTree2dNode<T, InType>(a, i, k);
    rc = new SegTree2dNode<T, InType>(a, k, j);
    seg_tree = new SegTree<T, InType>(vector<T>(tree_size));
    operation_2d(lc->seg_tree, rc->seg_tree);
  }
  ~SegTree2dNode() {
    delete lc;
    delete rc;
  }
  void set_2d(int kx, int ky, T x) {
    if (kx < i || j <= kx) return;
    if (j - i == 1) {
      seg_tree->set(ky, x);
      return;
    }
    lc->set_2d(kx, ky, x);
    rc->set_2d(kx, ky, x);
    operation_2d(lc->seg_tree, rc->seg_tree);
  }
  T range_query_2d(int lx, int rx, int ly, int ry) {
    if (lx <= i && j <= rx) return seg_tree->range_query(ly, ry);
    if (j <= lx || rx <= i) return -INF;
    return max(lc->range_query_2d(lx, rx, ly, ry), rc->range_query_2d(lx, rx, ly, ry));
  }
  void operation_2d(SegTree<T, InType>* x, SegTree<T, InType>* y) {
    for (int k = 0; k < tree_size; k++) {
      seg_tree->set(k, max(x->range_query(k, k + 1), y->range_query(k, k + 1)));
    }
  }
};
template<typename T, typename InType = T>
class SegTree2d {
public:
  SegTree2dNode<T, InType> root;
  SegTree2d() {}
  SegTree2d(const vector<vector<InType>>& mat) : root(mat, 0, mat.size()) {}
  void set_2d(int kx, int ky, T x) { root.set_2d(kx, ky, x); }
  T range_query_2d(int lx, int rx, int ly, int ry) { return root.range_query_2d(lx, rx, ly, ry); }
};
```

6.2 Range Add Point Query

```cpp
template<typename T, typename InType = T>
class SegTreeNode {
public:
  const T IDN = 0, DEF = 0;
  int i, j;
  T val;
  SegTreeNode<T, InType>* lc, * rc;
  SegTreeNode(int i, int j) : i(i), j(j) {
    if (j - i == 1) {
      lc = rc = nullptr;
      val = DEF;
      return;
    }
    int k = (i + j) / 2;
    lc = new SegTreeNode<T, InType>(i, k);
    rc = new SegTreeNode<T, InType>(k, j);
    val = 0;
  }
  SegTreeNode(const vector<InType>& a, int i, int j) : i(i), j(j) {
    if (j - i == 1) {
      lc = rc = nullptr;
      val = (T) a[i];
      return;
    }
    int k = (i + j) / 2;
    lc = new SegTreeNode<T, InType>(a, i, k);
    rc = new SegTreeNode<T, InType>(a, k, j);
    val = 0;
  }
  void range_add(int l, int r, T x) {
    if (r <= i || j <= l) return;
    if (l <= i && j <= r) {
      val += x;
      return;
    }
    lc->range_add(l, r, x);
    rc->range_add(l, r, x);
  }
  T point_query(int k) {
    if (k < i || j <= k) return IDN;
    if (j - i == 1) return val;
    return val + lc->point_query(k) + rc->point_query(k);
  }
};
template<typename T, typename InType = T>
class SegTree {
public:
  SegTreeNode<T, InType> root;
  SegTree(int n) : root(0, n) {}
  SegTree(const vector<InType>& a) : root(a, 0, a.size()) {}
  void range_add(int l, int r, T x) { root.range_add(l, r, x); }
```

```cpp
  T point_query(int k) { return root.point_query(k); }
};
```

6.3 Disjoint Set Union

```cpp
struct DSU {
  vector<int> parent, size;
  DSU(int n) {
    parent.resize(n);
    size.resize(n);
    for (int i = 0; i < n; i++) make_set(i);
  }
  void make_set(int v) {
    parent[v] = v;
    size[v] = 1;
  }
  bool is_same(int a, int b) { return find_set(a) == find_set(b); }
  int find_set(int v) { return v == parent[v] ? v : parent[v] = find_set(parent[v]); }
  void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
      if (size[a] < size[b]) swap(a, b);
      parent[b] = a;
      size[a] += size[b];
    }
  }
};
```

6.4 Sparse Table 2d

```cpp
const int N = 100;
int matrix[N][N];
int table[N][N][(int)(log2(N) + 1)][(int)(log2(N) + 1)];
void build_sparse_table(int n, int m) {
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
      table[i][j][0][0] = matrix[i][j];
  for (int k = 1; k <= (int)(log2(n)); k++)
    for (int i = 0; i + (1 << k) - 1 < n; i++)
      for (int j = 0; j + (1 << k) - 1 < m; j++)
        table[i][j][k][0] = min(table[i][j][k - 1][0], table[i + (1 << (k - 1))][j][k - 1][0]);
  for (int k = 1; k <= (int)(log2(m)); k++)
    for (int i = 0; i < n; i++)
      for (int j = 0; j + (1 << k) - 1 < m; j++)
        table[i][j][0][k] = min(table[i][j][0][k - 1], table[i][j + (1 << (k - 1))][0][k - 1]);
  for (int k = 1; k <= (int)(log2(n)); k++)
    for (int l = 1; l <= (int)(log2(m)); l++)
      for (int i = 0; i + (1 << k) - 1 < n; i++)
        for (int j = 0; j + (1 << l) - 1 < m; j++)
          table[i][j][k][l] = min(
            min(table[i][j][k - 1][l - 1], table[i + (1 << (k - 1))][j][k - 1][l - 1]),
            min(table[i][j + (1 << (l - 1))][k - 1][l - 1], table[i + (1 << (k - 1))][j + (1 << (l - 1))][k - 1][l - 1])
          );
}
int rmq(int x1, int y1, int x2, int y2) {
  int k = log2(x2 - x1 + 1), l = log2(y2 - y1 + 1);
  return max(
    max(table[x1][y1][k][l], table[x2 - (1 << k) + 1][y1][k][l]),
    max(table[x1][y2 - (1 << l) + 1][k][l], table[x2 - (1 << k) + 1][y2 - (1 << l) + 1][k][l])
  );
}
```

6.5 Mo

```cpp
void remove(idx);  // TODO: remove value at idx from data structure
void add(idx);     // TODO: add value at idx from data structure
int get_answer();  // TODO: extract the current answer of the data structure
int block_size;
struct Query {
  int l, r, idx;
  bool operator<(Query other) const {
    return make_pair(l / block_size, r) < make_pair(other.l / block_size, other.r);
  }
};
vector<int> mo_s_algorithm(vector<Query> queries) {
  vector<int> answers(queries.size());
  sort(queries.begin(), queries.end());
  // TODO: initialize data structure
  int cur_l = 0, cur_r = -1;
  // invariant: data structure will always reflect the range [cur_l, cur_r]
  for (Query q : queries) {
    while (cur_l > q.l) {
      cur_l--;
      add(cur_l);
    }
    while (cur_r < q.r) {
      cur_r++;
      add(cur_r);
    }
    while (cur_l < q.l) {
      remove(cur_l);
      cur_l++;
    }
    while (cur_r > q.r) {
      remove(cur_r);
      cur_r--;
    }
    answers[q.idx] = get_answer();
  }
  return answers;
}
```

6.6 Sparse Table

```cpp
ll log2_floor(ll i) {
  return i ? __builtin_clzll(1) - __builtin_clzll(i) : -1;
}
vector<vector<ll>> build_sum(ll N, ll K, vector<ll> &array) {
  vector<vector<ll>> st(K + 1, vector<ll>(N + 1));
  for (ll i = 0; i < N; i++) st[0][i] = array[i];
  for (ll i = 1; i <= K; i++)
    for (ll j = 0; j + (1 << i) <= N; j++)
      st[i][j] = st[i - 1][j] + st[i - 1][j + (1 << (i - 1))];
  return st;
}
ll sum_query(ll L, ll R, ll K, vector<vector<ll>> &st) {
  ll sum = 0;
  for (ll i = K; i >= 0; i--) {
    if ((1 << i) <= R - L + 1) {
      sum += st[i][L];
      L += 1 << i;
    }
  }
  return sum;
}
vector<vector<ll>> build_min(ll N, ll K, vector<ll> &array) {
  vector<vector<ll>> st(K + 1, vector<ll>(N + 1));
  for (ll i = 0; i < N; i++) st[0][i] = array[i];
  for (ll i = 1; i <= K; i++)
    for (ll j = 0; j + (1 << i) <= N; j++)
      st[i][j] = min(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
  return st;
}
ll min_query(ll L, ll R, vector<vector<ll>> &st) {
  ll i = log2_floor(R - L + 1);
  return min(st[i][L], st[i][R - (1 << i) + 1]);
}
```

6.7 Binary Trie

```cpp
struct Node { struct Node* parent, child[2]; };
struct BinaryTrie {
  Node* root;
  BinaryTrie() {
    root = new Node();
    root->parent = NULL;
    root->child[0] = NULL;
    root->child[1] = NULL;
  }
  void insert_node(int x) {
    Node* cur = root;
    for (int place = 29; place >= 0; place--) {
      int bit = x >> place & 1;
      if (cur->child[bit] != NULL) cur = cur->child[bit];
      else {
        cur->child[bit] = new Node();
        cur->child[bit]->parent = cur;
        cur = cur->child[bit];
        cur->child[0] = NULL;
        cur->child[1] = NULL;
      }
    }
  }
  void remove_node(int x) {
    Node* cur = root;
    for (int place = 29; place >= 0; place--) {
      int bit = x >> place & 1;
```

```
      if (cur->child[bit] == NULL) return;
      cur = cur->child[bit];
    }
    while (cur->parent != NULL && cur->child[0] == NULL && cur->child[1] == NULL) {
      Node* temp = cur;
      cur = cur->parent;
      if (temp == cur->child[0]) cur->child[0] = NULL;
      else cur->child[1] = NULL;
      delete temp;
    }
  }
  int get_min_xor(int x) {
    Node* cur = root;
    int minXor = 0;
    for (int place = 29; place >= 0; place--) {
      int bit = x >> place & 1;
      if (cur->child[bit] != NULL) cur = cur->child[bit];
      else {
        minXor ^= 1 << place;
        cur = cur->child[1 ^ bit];
      }
    }
    return minXor;
  }
};
```

6.8 Segment Tree

```
template<typename T, typename InType = T>
class SegTreeNode {
public:
  const T IDN = 0, DEF = 0;
  int i, j;
  T val;
  SegTreeNode<T, InType>* lc, * rc;
  SegTreeNode(int i, int j) : i(i), j(j) {
    if (j - i == 1) {
      lc = rc = nullptr;
      val = DEF;
      return;
    }
    int k = (i + j) / 2;
    lc = new SegTreeNode<T, InType>(i, k);
    rc = new SegTreeNode<T, InType>(k, j);
    val = op(lc->val, rc->val);
  }
  SegTreeNode(const vector<InType>& a, int i, int j) : i(i), j(j) {
    if (j - i == 1) {
      lc = rc = nullptr;
      val = (T) a[i];
      return;
    }
    int k = (i + j) / 2;
    lc = new SegTreeNode<T, InType>(a, i, k);
    rc = new SegTreeNode<T, InType>(a, k, j);
    val = op(lc->val, rc->val);
  }
  void set(int k, T x) {
    if (k < i || j <= k) return;
    if (j - i == 1) {
      val = x;
      return;
    }
    lc->set(k, x);
    rc->set(k, x);
    val = op(lc->val, rc->val);
  }
  T range_query(int l, int r) {
    if (l <= i && j <= r) return val;
    if (j <= l || r <= i) return IDN;
    return op(lc->range_query(l, r), rc->range_query(l, r));
  }
  T op(T x, T y) {}
};
template<typename T, typename InType = T>
class SegTree {
public:
  SegTreeNode<T, InType> root;
  SegTree(int n) : root(0, n) {}
  SegTree(const vector<InType>& a) : root(a, 0, a.size()) {}
  void set(int k, T x) { root.set(k, x); }
  T range_query(int l, int r) { return root.range_query(l, r); }
};
```

6.9 Sqrt Decomposition

```
int n;
vector<int> a (n);
int len = (int) sqrt (n + .0) + 1; // size of the block and the number of blocks
vector<int> b (len);
for (int i = 0; i<n; ++i) b[i / len] += a[i];
for (;;) {
  int l, r;
  // read input data for the next query
  int sum = 0;
  for (int i = l; i <= r; )
    if (i % len == 0 && i + len - 1 <= r) {
      // if the whole block starting at i belongs to [l, r]
      sum += b[i / len];
      i += len;
    } else {
      sum += a[i];
      ++i;
    }
  // or
  /*
  int sum = 0;
  int c_l = l / len,   c_r = r / len;
  if (c_l == c_r)
      for (int i=l; i<=r; ++i)
          sum += a[i];
  else {
      for (int i=l, end=(c_l+1)*len-1; i<=end; ++i)
          sum += a[i];
      for (int i=c_l+1; i<=c_r-1; ++i)
          sum += b[i];
      for (int i=c_r*len; i<=r; ++i)
          sum += a[i];
  }
  */
}
```

6.10 Minimum Queue

```
ll get_minimum(stack<pair<ll, ll>> &s1, stack<pair<ll, ll>> &s2) {
  if (s1.empty() || s2.empty()) {
    return s1.empty() ? s2.top().second : s1.top().second;
  } else {
    return min(s1.top().second, s2.top().second);
  }
}
void add_element(ll new_element, stack<pair<ll, ll>> &s1) {
  ll minimum = s1.empty() ? new_element : min(new_element, s1.top().second);
  s1.push({new_element, minimum});
}
ll remove_element(stack<pair<ll, ll>> &s1, stack<pair<ll, ll>> &s2) {
  if (s2.empty()) {
    while (!s1.empty()) {
      ll element = s1.top().first;
      s1.pop();
      ll minimum = s2.empty() ? element : min(element, s2.top().second);
      s2.push({element, minimum});
    }
  }
  ll removed_element = s2.top().first;
  s2.pop();
  return removed_element;
}
```

6.11 Range Add Range Query

```
template<typename T, typename InType = T>
class SegTreeNode {
public:
  const T IDN = 0, DEF = 0;
  int i, j;
  T val, to_add = 0;
  SegTreeNode<T, InType>* lc, * rc;
  SegTreeNode(int i, int j) : i(i), j(j) {
    if (j - i == 1) {
      lc = rc = nullptr;
      val = DEF;
      return;
    }
    int k = (i + j) / 2;
    lc = new SegTreeNode<T, InType>(i, k);
    rc = new SegTreeNode<T, InType>(k, j);
    val = operation(lc->val, rc->val);
  }
  SegTreeNode(const vector<InType>& a, int i, int j) : i(i), j(j) {
    if (j - i == 1) {
      lc = rc = nullptr;
      val = (T) a[i];
      return;
    }
    int k = (i + j) / 2;
    lc = new SegTreeNode<T, InType>(a, i, k);
    rc = new SegTreeNode<T, InType>(a, k, j);
    val = operation(lc->val, rc->val);
  }
  void propagate() {
    if (to_add == 0) return;
    val += to_add;
    if (j - i > 1) {
```

```
      lc->to_add += to_add;
      rc->to_add += to_add;
    }
    to_add = 0;
  }
  void range_add(int l, int r, T delta) {
    propagate();
    if (r <= i || j <= l) return;
    if (l <= i && j <= r) {
      to_add += delta;
      propagate();
    } else {
      lc->range_add(l, r, delta);
      rc->range_add(l, r, delta);
      val = operation(lc->val, rc->val);
    }
  }
  T range_query(int l, int r) {
    propagate();
    if (l <= i && j <= r) return val;
    if (j <= l || r <= i) return IDN;
    return operation(lc->range_query(l, r), rc->range_query(l, r));
  }
  T operation(T x, T y) {}
};
template<typename T, typename InType = T>
class SegTree {
public:
  SegTreeNode<T, InType> root;
  SegTree(int n) : root(0, n) {}
  SegTree(const vector<InType>& a) : root(a, 0, a.size()) {}
  void range_add(int l, int r, T delta) { root.range_add(l, r, delta); }
  T range_query(int l, int r) { return root.range_query(l, r); }
};
```

7 Graph Theory

7.1 Bridge

```
int n;
vector<vector<int>> adj;
vector<bool> visited;
vector<int> tin, low;
int timer;
void dfs(int v, int p = -1) {
  visited[v] = true;
  tin[v] = low[v] = timer++;
  for (int to : adj[v]) {
    if (to == p) continue;
    if (visited[to]) {
      low[v] = min(low[v], tin[to]);
    } else {
      dfs(to, v);
      low[v] = min(low[v], low[to]);
      if (low[to] > tin[v]) IS_BRIDGE(v, to);
    }
  }
}
void find_bridges() {
  timer = 0;
  visited.assign(n, false);
  tin.assign(n, -1);
  low.assign(n, -1);
  for (int i = 0; i < n; ++i) {
    if (!visited[i]) dfs(i);
  }
}
```

7.2 Dijkstra

```
const int INF = 1000000000;
vector<vector<pair<int, int>>> adj;
void dijkstra(int s, vector<int> & d, vector<int> & p) {
  int n = adj.size();
  d.assign(n, INF);
  p.assign(n, -1);
  d[s] = 0;
  using pii = pair<int, int>;
  priority_queue<pii, vector<pii>, greater<pii>> q;
  q.push({0, s});
  while (!q.empty()) {
    int v = q.top().second, d_v = q.top().first;
    q.pop();
    if (d_v != d[v]) continue;
    for (auto edge : adj[v]) {
      int to = edge.first, len = edge.second;
      if (d[v] + len < d[to]) {
          d[to] = d[v] + len;
          p[to] = v;
          q.push({d[to], to});
      }
    }
  }
}
```

7.3 Zero One Bfs

```
vector<int> d(n, INF);
d[s] = 0;
deque<int> q;
q.push_front(s);
while (!q.empty()) {
  int v = q.front();
  q.pop_front();
  for (auto edge : adj[v]) {
    int u = edge.first, w = edge.second;
    if (d[v] + w < d[u]) {
      d[u] = d[v] + w;
      if (w == 1) q.push_back(u);
      else q.push_front(u);
    }
  }
}
```

7.4 Hungarian

```
vector<int> u (n+1), v (m+1), p (m+1), way (m+1);
for (int i=1; i<=n; ++i) {
  p[0] = i;
  int j0 = 0;
  vector<int> minv (m+1, INF);
  vector<bool> used (m+1, false);
  do {
    used[j0] = true;
    int i0 = p[j0],  delta = INF,  j1;
    for (int j=1; j<=m; ++j)
      if (!used[j]) {
        int cur = A[i0][j]-u[i0]-v[j];
        if (cur < minv[j]) minv[j] = cur,  way[j] = j0;
        if (minv[j] < delta) delta = minv[j],  j1 = j;
      }
    for (int j=0; j<=m; ++j)
      if (used[j]) u[p[j]] += delta,  v[j] -= delta;
      else minv[j] -= delta;
    j0 = j1;
  } while (p[j0] != 0);
  do {
    int j1 = way[j0];
    p[j0] = p[j1];
    j0 = j1;
  } while (j0);
}
vector<int> ans (n+1);
for (int j=1; j<=m; ++j)
  ans[p[j]] = j;
int cost = -v[0];
```

7.5 Ford Fulkerson

```
bool bfs(ll n, vector<vector<ll>> &r_graph, ll s, ll t, vector<ll> &parent) {
  vector<bool> visited(n, false);
  queue<ll> q;
  q.push(s);
  visited[s] = true;
  parent[s] = -1;
  while (!q.empty()) {
    ll u = q.front();
    q.pop();
    for (ll v = 0; v < n; v++) {
      if (!visited[v] && r_graph[u][v] > 0) {
        if (v == t) {
          parent[v] = u;
          return true;
        }
        q.push(v);
        parent[v] = u;
        visited[v] = true;
      }
    }
  }
  return false;
}
ll ford_fulkerson(ll n, vector<vector<ll>> graph, ll s, ll t) {
  ll u, v;
  vector<vector<ll>> r_graph;
  for (u = 0; u < n; u++)
    for (v = 0; v < n; v++)
      r_graph[u][v] = graph[u][v];
  vector<ll> parent;
  ll max_flow = 0;
  while (bfs(n, r_graph, s, t, parent)) {
    ll path_flow = INF;
    for (v = t; v != s; v = parent[v]) {
      u = parent[v];
      path_flow = min(path_flow, r_graph[u][v]);
```

```
    }
    for (v = t; v != s; v = parent[v]) {
      u = parent[v];
      r_graph[u][v] -= path_flow;
      r_graph[v][u] += path_flow;
    }
    max_flow += path_flow;
  }
  return max_flow;
}
```

7.6 Prim

```
const int INF = 1000000000;
struct Edge {
  int w = INF, to = -1;
  bool operator<(Edge const& other) const {
    return make_pair(w, to) < make_pair(other.w, other.to);
  }
};
int n;
vector<vector<Edge>> adj;
void prim() {
  int total_weight = 0;
  vector<Edge> min_e(n);
  min_e[0].w = 0;
  set<Edge> q;
  q.insert({0, 0});
  vector<bool> selected(n, false);
  for (int i = 0; i < n; ++i) {
    if (q.empty()) {
      cout << "No MST!" << endl;
      exit(0);
    }
    int v = q.begin()->to;
    selected[v] = true;
    total_weight += q.begin()->w;
    q.erase(q.begin());
    if (min_e[v].to != -1) cout << v << " " << min_e[v].to << endl;
    for (Edge e : adj[v]) {
      if (!selected[e.to] && e.w < min_e[e.to].w) {
        q.erase({min_e[e.to].w, e.to});
        min_e[e.to] = {e.w, v};
        q.insert({e.w, e.to});
      }
    }
  }
  cout << total_weight << endl;
}
```

7.7 Centroid Decomposition

```
vector<vector<int>> adj;
vector<bool> is_removed;
vector<int> subtree_size;
int get_subtree_size(int node, int parent = -1) {
        subtree_size[node] = 1;
        for (int child : adj[node]) {
                if (child == parent || is_removed[child]) continue;
                subtree_size[node] += get_subtree_size(child, node);
        }
        return subtree_size[node];
}
int get_centroid(int node, int tree_size, int parent = -1) {
        for (int child : adj[node]) {
                if (child == parent || is_removed[child]) continue;
                if (subtree_size[child] * 2 > tree_size) return get_centroid(child, tree_size, node);
        }
        return node;
}
void build_centroid_decomp(int node = 0) {
        int centroid = get_centroid(node, get_subtree_size(node));
        // do something
        is_removed[centroid] = true;
        for (int child : adj[centroid]) {
                if (is_removed[child]) continue;
                build_centroid_decomp(child);
        }
}
```

7.8 Kahn

```
void kahn(vector<vector<ll>> &adj) {
  ll n = adj.size();
  vector<ll> in_degree(n, 0);
  for (ll u = 0; u < n; u++)
    for (ll v: adj[u]) in_degree[v]++;
  queue<ll> q;
  for (ll i = 0; i < n; i++)
    if (in_degree[i] == 0)
      q.push(i);
  ll cnt = 0;
  vector<ll> top_order;
  while (!q.empty()) {
    ll u = q.front();
    q.pop();
    top_order.push_back(u);
    for (ll v : adj[u])
      if (--in_degree[v] == 0) q.push(v);
    cnt++;
  }
  if (cnt != n) {
    cout << -1 << '\n';
    return;
  }
  // print top_order
}
```

7.9 Dinics

```
struct FlowEdge {
  int v, u;
  ll cap, flow = 0;
  FlowEdge(int v, int u, ll cap) : v(v), u(u), cap(cap) {}
};
struct Dinic {
  const ll flow_inf = 1e18;
  vector<FlowEdge> edges;
  vector<vector<int>> adj;
  int n, m = 0, s, t;
  vector<int> level, ptr;
  queue<int> q;
  Dinic(int n, int s, int t) : n(n), s(s), t(t) {
    adj.resize(n);
    level.resize(n);
    ptr.resize(n);
  }
  void add_edge(int v, int u, ll cap) {
    edges.emplace_back(v, u, cap);
    edges.emplace_back(u, v, 0);
    adj[v].push_back(m);
    adj[u].push_back(m + 1);
    m += 2;
  }
  bool bfs() {
    while (!q.empty()) {
      int v = q.front();
      q.pop();
      for (int id : adj[v]) {
        if (edges[id].cap - edges[id].flow < 1) continue;
        if (level[edges[id].u] != -1) continue;
        level[edges[id].u] = level[v] + 1;
        q.push(edges[id].u);
      }
    }
    return level[t] != -1;
  }
  ll dfs(int v, ll pushed) {
    if (pushed == 0) return 0;
    if (v == t) return pushed;
    for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
      int id = adj[v][cid], u = edges[id].u;
      if (level[v] + 1 != level[u] || edges[id].cap - edges[id].flow < 1) continue;
      ll tr = dfs(u, min(pushed, edges[id].cap - edges[id].flow));
      if (tr == 0) continue;
      edges[id].flow += tr;
      edges[id ^ 1].flow -= tr;
      return tr;
    }
    return 0;
  }
  ll flow() {
    ll f = 0;
    while (true) {
      fill(level.begin(), level.end(), -1);
      level[s] = 0;
      q.push(s);
      if (!bfs()) break;
      fill(ptr.begin(), ptr.end(), 0);
      while (ll pushed = dfs(s, flow_inf)) f += pushed;
    }
    return f;
  }
};
```

7.10 Floyd Warshall

```
void floyd_warshall(vector<vector<ll>> &dis, ll n) {
  for (ll k = 0; k < n; k++)
    for (ll i = 0; i < n; i++)
      for (ll j = 0; j < n; j++)
```

```
        if (dis[i][k] < INF && dis[k][j] < INF)
          dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
  for (ll i = 0; i < n; i++)
    for (ll j = 0; j < n; j++)
      for (ll k = 0; k < n; k++)
        if (dis[k][k] < 0 && dis[i][k] < INF && dis[k][j] < INF)
          dis[i][j] = -INF;
}
```

7.11 Kosaraju

```
void topo_sort(int u, vector<vector<int>>& adj, vector<bool>& vis, stack<int>& stk) {
  vis[u] = true;
  for (int v : adj[u]) {
    if (!vis[v]) {
      topo_sort(v, adj, vis, stk);
    }
  }
  stk.push(u);
}

vector<vector<int>> transpose(int n, vector<vector<int>>& adj) {
  vector<vector<int>> adj_t(n);
  for (int u = 0; u < n; u++) {
    for (int v : adj[u]) {
      adj_t[v].push_back(u);
    }
  }
  return adj_t;
}

void get_scc(int u, vector<vector<int>>& adj_t, vector<bool>& vis, vector<int>& scc) {
  vis[u] = true;
  scc.push_back(u);
  for (int v : adj_t[u]) {
    if (!vis[v]) {
      get_scc(v, adj_t, vis, scc);
    }
  }
}

void kosaraju(int n, vector<vector<int>>& adj, vector<vector<int>>& sccs) {
  vector<bool> vis(n, false);
  stack<int> stk;
  for (int u = 0; u < n; u++) {
    if (!vis[u]) {
      topo_sort(u, adj, vis, stk);
    }
  }
  vector<vector<int>> adj_t = transpose(n, adj);
  for (int u = 0; u < n; u++) {
    vis[u] = false;
  }
  while (!stk.empty()) {
    int u = stk.top();
    stk.pop();
    if (!vis[u]) {
      vector<int> scc;
      get_scc(u, adj_t, vis, scc);
      sccs.push_back(scc);
    }
  }
}
```

7.12 Maximum Bipartite Matching

```
bool bpm(ll n, ll m, vector<vector<bool>> &bpGraph, ll u, vector<bool> &seen, vector<ll> &matchR) {
  for (ll v = 0; v < m; v++) {
    if (bpGraph[u][v] && !seen[v]) {
      seen[v] = true;
      if (matchR[v] < 0 || bpm(n, m, bpGraph, matchR[v], seen, matchR)) {
        matchR[v] = u;
        return true;
      }
    }
  }
  return false;
}
ll maxBPM(ll n, ll m, vector<vector<bool>> &bpGraph) {
  vector<ll> matchR(m, -1);
  ll result = 0;
  for (ll u = 0; u < n; u++) {
    vector<bool> seen(m, false);
    if (bpm(n, m, bpGraph, u, seen, matchR)) {
      result++;
    }
  }
  return result;
}
```

7.13 Kruskals

```
struct Edge {
  int u, v, weight;
  bool operator<(Edge const& other) {
    return weight < other.weight;
  }
};
int n;
vector<Edge> edges;
int cost = 0;
vector<Edge> result;
DSU dsu = DSU(n);
sort(edges.begin(), edges.end());
for (Edge e : edges) {
  if (dsu.find_set(e.u) != dsu.find_set(e.v)) {
    cost += e.weight;
    result.push_back(e);
    dsu.union_sets(e.u, e.v);
  }
}
```

7.14 Is Cyclic

```
bool is_cyclic_util(int u, vector<vector<int>> &adj, vector<bool> &vis, vector<bool> &rec) {
  vis[u] = true;
  rec[u] = true;
  for(auto v : adj[u]) {
    if (!vis[v] && is_cyclic_util(v, adj, vis, rec)) return true;
    else if (rec[v]) return true;
  }
  rec[u] = false;
  return false;
}
bool is_cyclic(int n, vector<vector<int>> &adj) {
  vector<bool> vis(n, false), rec(n, false);
  for (int i = 0; i < n; i++)
    if (!vis[i] && is_cyclic_util(i, adj, vis, rec)) return true;
  return false;
}
```

7.15 Find Cycle

```
bool dfs(ll v) {
  color[v] = 1;
  for (ll u : adj[v]) {
    if (color[u] == 0) {
      parent[u] = v;
      if (dfs(u)) {
        return true;
      }
    } else if (color[u] == 1) {
      cycle_end = v;
      cycle_start = u;
      return true;
    }
  }
  color[v] = 2;
  return false;
}
void find_cycle() {
  color.assign(n, 0);
  parent.assign(n, -1);
  cycle_start = -1;
  for (ll v = 0; v < n; v++) {
    if (color[v] == 0 && dfs(v)) {
      break;
    }
  }
  if (cycle_start == -1) {
    cout << "Acyclic" << endl;
  } else {
    vector<ll> cycle;
    cycle.push_back(cycle_start);
    for (ll v = cycle_end; v != cycle_start; v = parent[v]) {
      cycle.push_back(v);
    }
    cycle.push_back(cycle_start);
    reverse(cycle.begin(), cycle.end());
    cout << "Cycle found: ";
    for (ll v : cycle) {
      cout << v << ' ';
    }
    cout << '\n';
  }
}
```

7.16 Topological Sort

```
void dfs(ll v) {
```

```
  visited[v] = true;
  for (ll u : adj[v]) {
    if (!visited[u]) {
      dfs(u);
    }
  }
  ans.push_back(v);
}
void topological_sort() {
  visited.assign(n, false);
  ans.clear();
  for (ll i = 0; i < n; ++i) {
    if (!visited[i]) {
      dfs(i);
    }
  }
  reverse(ans.begin(), ans.end());
}
```

7.17 Min Cost Flow

```
struct Edge {
  int from, to, capacity, cost;
};
vector<vector<int>> adj, cost, capacity;
const int INF = 1e9;
void shortest_paths(int n, int v0, vector<int>& d, vector<int>& p) {
  d.assign(n, INF);
  d[v0] = 0;
  vector<bool> inq(n, false);
  queue<int> q;
  q.push(v0);
  p.assign(n, -1);
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    inq[u] = false;
    for (int v : adj[u]) {
      if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v]) {
        d[v] = d[u] + cost[u][v];
        p[v] = u;
        if (!inq[v]) {
          inq[v] = true;
          q.push(v);
        }
      }
    }
  }
}
int min_cost_flow(int N, vector<Edge> edges, int K, int s, int t) {
  adj.assign(N, vector<int>());
  cost.assign(N, vector<int>(N, 0));
  capacity.assign(N, vector<int>(N, 0));
  for (Edge e : edges) {
    adj[e.from].push_back(e.to);
    adj[e.to].push_back(e.from);
    cost[e.from][e.to] = e.cost;
    cost[e.to][e.from] = -e.cost;
    capacity[e.from][e.to] = e.capacity;
  }
  int flow = 0;
  int cost = 0;
  vector<int> d, p;
  while (flow < K) {
    shortest_paths(N, s, d, p);
    if (d[t] == INF) break;
    int f = K - flow, cur = t;
    while (cur != s) {
      f = min(f, capacity[p[cur]][cur]);
      cur = p[cur];
    }
    flow += f;
    cost += f * d[t];
    cur = t;
    while (cur != s) {
      capacity[p[cur]][cur] -= f;
      capacity[cur][p[cur]] += f;
      cur = p[cur];
    }
  }
  if (flow < K) return -1;
  else return cost;
}
```

7.18 Kuhn

```
int n, k;
vector<vector<int>> g;
vector<int> mt;
vector<bool> used;
bool try_kuhn(int v) {
  if (used[v]) return false;
  used[v] = true;
  for (int to : g[v]) {
    if (mt[to] == -1 || try_kuhn(mt[to])) {
      mt[to] = v;
      return true;
    }
  }
  return false;
}
int main() {
  mt.assign(k, -1);
    vector<bool> used1(n, false);
    for (int v = 0; v < n; ++v) {
      for (int to : g[v]) {
        if (mt[to] == -1) {
          mt[to] = v;
          used1[v] = true;
          break;
        }
      }
    }
    for (int v = 0; v < n; ++v) {
      if (used1[v]) continue;
      used.assign(n, false);
      try_kuhn(v);
    }
    for (int i = 0; i < k; ++i)
      if (mt[i] != -1)
        printf("%d %d\n", mt[i] + 1, i + 1);
}
```

7.19 Articulation Point

```
void APUtil(vector<vector<ll>> &adj, ll u, vector<bool> &visited,
vector<ll> &disc, vector<ll> &low, ll &time, ll parent, vector<bool> &isAP) {
  ll children = 0;
  visited[u] = true;
  disc[u] = low[u] = ++time;
  for (auto v : adj[u]) {
    if (!visited[v]) {
      children++;
      APUtil(adj, v, visited, disc, low, time, u, isAP);
      low[u] = min(low[u], low[v]);
      if (parent != -1 && low[v] >= disc[u]) {
        isAP[u] = true;
      }
    } else if (v != parent) {
      low[u] = min(low[u], disc[v]);
    }
  }
  if (parent == -1 && children > 1) {
    isAP[u] = true;
  }
}
void AP(vector<vector<ll>> &adj, ll n) {
  vector<ll> disc(n), low(n);
  vector<bool> visited(n), isAP(n);
  ll time = 0, par = -1;
  for (ll u = 0; u < n; u++) {
    if (!visited[u]) {
      APUtil(adj, u, visited, disc, low, time, par, isAP);
    }
  }
  for (ll u = 0; u < n; u++) {
    if (isAP[u]) {
      cout << u << " ";
    }
  }
}
```

7.20 Hierholzer

```
void print_circuit(vector<vector<ll>> &adj) {
  map<ll, ll> edge_count;
  for (ll i = 0; i< adj.size(); i++) {
    edge_count[i] = adj[i].size();
  }
  if (!adj.size()) {
    return;
  }
  stack<ll> curr_path;
  vector<ll> circuit;
  curr_path.push(0);
  ll curr_v = 0;
  while (!curr_path.empty()) {
    if (edge_count[curr_v]) {
      curr_path.push(curr_v);
      ll next_v = adj[curr_v].back();
      edge_count[curr_v]--;
      adj[curr_v].pop_back();
      curr_v = next_v;
    } else {
      circuit.push_back(curr_v);
      curr_v = curr_path.top();
      curr_path.pop();
    }
  }
  for (ll i = circuit.size() - 1; i >= 0; i--) {
    cout << circuit[i] << ' ';
  }
```

```
}
```

7.21 Lowest Common Ancestor

```
struct LCA {
  vector<ll> height, euler, first, segtree;
  vector<bool> visited;
  ll n;
  LCA(vector<vector<ll>> &adj, ll root = 0) {
    n = adj.size();
    height.resize(n);
    first.resize(n);
    euler.reserve(n * 2);
    visited.assign(n, false);
    dfs(adj, root);
    ll m = euler.size();
    segtree.resize(m * 4);
    build(1, 0, m - 1);
  }
  void dfs(vector<vector<ll>> &adj, ll node, ll h = 0) {
    visited[node] = true;
    height[node] = h;
    first[node] = euler.size();
    euler.push_back(node);
    for (auto to : adj[node]) {
      if (!visited[to]) {
        dfs(adj, to, h + 1);
        euler.push_back(node);
      }
    }
  }
  void build(ll node, ll b, ll e) {
    if (b == e) segtree[node] = euler[b];
    else {
      ll mid = (b + e) / 2;
      build(node << 1, b, mid);
      build(node << 1 | 1, mid + 1, e);
      ll l = segtree[node << 1], r = segtree[node << 1 | 1];
      segtree[node] = (height[l] < height[r]) ? l : r;
    }
  }
  ll query(ll node, ll b, ll e, ll L, ll R) {
    if (b > R || e < L) return -1;
    if (b >= L && e <= R) return segtree[node];
    ll mid = (b + e) >> 1;
    ll left = query(node << 1, b, mid, L, R);
    ll right = query(node << 1 | 1, mid + 1, e, L, R);
    if (left == -1) return right;
    if (right == -1) return left;
    return height[left] < height[right] ? left : right;
  }
  ll lca(ll u, ll v) {
    ll left = first[u], right = first[v];
    if (left > right) swap(left, right);
    return query(1, 0, euler.size() - 1, left, right);
  }
};
```

7.22 Bellman Ford

```
struct Edge {
  int a, b, cost;
};
int n, m, v;
vector<Edge> edges;
const int INF = 1000000000;
void solve() {
  vector<int> d(n, INF);
  d[v] = 0;
  vector<int> p(n, -1);
  int x;
  for (int i = 0; i < n; ++i) {
    x = -1;
    for (Edge e : edges)
      if (d[e.a] < INF)
        if (d[e.b] > d[e.a] + e.cost) {
          d[e.b] = max(-INF, d[e.a] + e.cost);
          p[e.b] = e.a;
          x = e.b;
        }
  }
  if (x == -1) cout << "No negative cycle from " << v;
  else {
    int y = x;
    for (int i = 0; i < n; ++i) y = p[y];
    vector<int> path;
    for (int cur = y;; cur = p[cur]) {
      path.push_back(cur);
      if (cur == y && path.size() > 1) break;
    }
    reverse(path.begin(), path.end());
    cout << "Negative cycle: ";
    for (int u : path) cout << u << ' ';
  }
}
```

7.23 Edmonds Karp

```
int n;
vector<vector<int>> capacity;
vector<vector<int>> adj;
int bfs(int s, int t, vector<int>& parent) {
  fill(parent.begin(), parent.end(), -1);
  parent[s] = -2;
  queue<pair<int, int>> q;
  q.push({s, INF});
  while (!q.empty()) {
    int cur = q.front().first, flow = q.front().second;
    q.pop();
    for (int next : adj[cur]) {
      if (parent[next] == -1 && capacity[cur][next]) {
        parent[next] = cur;
        int new_flow = min(flow, capacity[cur][next]);
        if (next == t) return new_flow;
        q.push({next, new_flow});
      }
    }
  }
  return 0;
}
int maxflow(int s, int t) {
  int flow = 0;
  vector<int> parent(n);
  int new_flow;
  while (new_flow = bfs(s, t, parent)) {
    flow += new_flow;
    int cur = t;
    while (cur != s) {
      int prev = parent[cur];
      capacity[prev][cur] -= new_flow;
      capacity[cur][prev] += new_flow;
      cur = prev;
    }
  }
  return flow;
}
```

7.24 Is Bipartite

```
bool is_bipartite(vector<ll> &col, vector<vector<ll>> &adj, ll n) {
  queue<pair<ll, ll>> q;
  for (ll i = 0; i < n; i++) {
    if (col[i] == -1) {
      q.push({i, 0});
      col[i] = 0;
      while (!q.empty()) {
        pair<ll, ll> p = q.front();
        q.pop();
        ll v = p.first, c = p.second;
        for (ll j : adj[v]) {
          if (col[j] == c) {
            return false;
          }
          if (col[j] == -1) {
            col[j] = (c ? 0 : 1);
            q.push({j, col[j]});
          }
        }
      }
    }
  }
  return true;
}
```

7.25 Fast Second Mst

```
struct edge {
    int s, e, w, id;
    bool operator<(const struct edge& other) {
        return w < other.w; }
};
typedef struct edge Edge;
const int N = 2e5 + 5;
long long res = 0, ans = 1e18;
int n, m, a, b, w, id, l = 21;
vector<Edge> edges;
vector<int> h(N, 0), parent(N, -1), size(N, 0), present(N, 0);
vector<vector<pair<int, int>>> adj(N), dp(N, vector<pair<int, int>>(l));
vector<vector<int>> up(N, vector<int>(l, -1));
pair<int, int> combine(pair<int, int> a, pair<int, int> b) {
  vector<int> v = {a.first, a.second, b.first, b.second};
  int topTwo = -3, topOne = -2;
  for (int c : v) {
```

```cpp
    if (c > topOne) {
      topTwo = topOne;
      topOne = c;
    } else if (c > topTwo && c < topOne) topTwo = c;
  }
  return {topOne, topTwo};
}
void dfs(int u, int par, int d) {
  h[u] = 1 + h[par];
  up[u][0] = par;
  dp[u][0] = {d, -1};
  for (auto v : adj[u]) {
    if (v.first != par) dfs(v.first, u, v.second);
  }
}
pair<int, int> lca(int u, int v) {
  pair<int, int> ans = {-2, -3};
  if (h[u] < h[v]) swap(u, v);
  for (int i = l - 1; i >= 0; i--) {
    if (h[u] - h[v] >= (1 << i)) {
      ans = combine(ans, dp[u][i]);
      u = up[u][i];
    }
  }
  if (u == v) return ans;
  for (int i = l - 1; i >= 0; i--) {
    if (up[u][i] != -1 && up[v][i] != -1 && up[u][i] != up[v][i]) {
      ans = combine(ans, combine(dp[u][i], dp[v][i]));
      u = up[u][i];
      v = up[v][i];
    }
  }
  ans = combine(ans, combine(dp[u][0], dp[v][0]));
  return ans;
}

int main(void) {
  cin >> n >> m;
  for (int i = 1; i <= n; i++) {
    parent[i] = i;
    size[i] = 1;
  }
  for (int i = 1; i <= m; i++) {
    cin >> a >> b >> w; // 1-indexed
    edges.push_back({a, b, w, i - 1});
  }
  sort(edges.begin(), edges.end());
  for (int i = 0; i <= m - 1; i++) {
    a = edges[i].s;
    b = edges[i].e;
    w = edges[i].w;
    id = edges[i].id;
    if (unite_set(a, b)) {
      adj[a].emplace_back(b, w);
      adj[b].emplace_back(a, w);
      present[id] = 1;
      res += w;
    }
  }
  dfs(1, 0, 0);
  for (int i = 1; i <= l - 1; i++) {
    for (int j = 1; j <= n; ++j) {
      if (up[j][i - 1] != -1) {
        int v = up[j][i - 1];
        up[j][i] = up[v][i - 1];
        dp[j][i] = combine(dp[j][i - 1], dp[v][i - 1]);
      }
    }
  }
  for (int i = 0; i <= m - 1; i++) {
    id = edges[i].id;
    w = edges[i].w;
    if (!present[id]) {
      auto rem = lca(edges[i].s, edges[i].e);
      if (rem.first != w) {
        if (ans > res + w - rem.first) ans = res + w - rem.first;
      } else if (rem.second != -1) {
        if (ans > res + w - rem.second) ans = res + w - rem.second;
      }
    }
  }
  cout << ans << "\n";
  return 0;
}
```

8 References

8.1 Stack

```cpp
// declaration
stack<T> stk;
// functions
stk.empty();
stk.size();
stk.top();
stk.push(x);
stk.pop();
```

8.2 Queue

```cpp
// declaration
queue<T> q;
// functions
q.empty();
q.size();
q.front();
q.back();
q.push(x);
q.pop();
```

8.3 Syntax

```cpp
// set
st.insert(x)
st.begin()
st.find(x)
st.count(x)
st.erase(x)
st.end()
st.size()
st.empty()
// multiset
ms.insert(x)
ms.begin()
ms.end()
ms.clear()
ms.erase(x)
ms.size()
ms.empty()
// map
begin()
end()
size()
max_size()
empty()
pair insert(keyvalue, mapvalue)
erase(iterator position)
erase(const g)
clear()
// ordered_set
find_by_order(k)
order_of_key(k)
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

#define ordered_set                                      \
    tree<int, null_type, less<int>, rb_tree_tag, \
         tree_order_statistics_node_update>
// tuple
get<i>(tuple)
make_tuple(a1, a2, ...)
tuple_size<decltype(tuple)>::value
tuple1.swap(tuple2)
tie(a1, a2, ...) = tuple
tuple_cat(tuple1, tuple2)
// iterator
for (auto it = s.begin(); it != s.end(); it++) cout << *it << "\n";
begin()
end()
advance(ptr, k)
next(ptr, k)
prev(ptr, k)
// permutations
do {} while (next_permutation(nums.begin(), nums.end()));
// bitset
int num = 27; // Binary representation: 11011
bitset<10> s(string("0010011010")); // from right to left
bitset<sizeof(int) * 8> bits(num);
int set_bits = bits.count();
bits.set(index, val);
bits.reset();
bits.flip();
bits.all();
bits.any();
bits.none();
bits.test();
to_string();
to_ulong();
to_ullong();
[], &, |, !, >>=, <<=, &=, |=, ^=, ~;
// hamming distance
int hamming(int a, int b) {
  return __builtin_popcount(a ^ b);
}
// gcc compiler
__builtin_popcount(x)
__builtin_parity(x)
__builtin_clz(x) // leading
__builtin_ctz(x) // trailing
```

8.4 Priority Queue

```cpp
// declaration
priority_queue<T> pq;
priority_queue<T> pq(v.begin(), v.end());
priority_queue<T, vector<T>, greater<T>> pq;
// custom comparator
class Compare {
public:
  bool operator() (T a, T b) {
    if (cond) {
        return true; // do not swap
    }
    return false;
  }
};
priority_queue<T, vector<T>, Compare> pq;
// functions
pq.empty();
pq.size();
pq.top();
pq.push(x);
pq.pop();
```

8.5 Vector

```cpp
// declaration
vector<T> v;
vector<T> v = {v0, v1, v2, ...};
vector<T> v(size, initial);
// functions
v.begin();
v.end();
v.size();
v.empty();
v.push_back(x);
v.pop_back();
v.insert();
v.erase(x);
v.clear();
// algorithms
lower_bound(v.begin(), v.end(), x);
upper_bound(v.begin(), v.end(), x);
binary_search(v.begin(), v.end(), x);
// sort
sort(v.begin(), v.end());
sort(v.rbegin(), v.rend()); // reverse iterators
sort(v.begin(), v.end(), greater<T>); // using functor
bool comp(T a, T b) {
  if (cond) {
    return true; // do not swap
  }
  return false;
}
sort(v.begin(), v.end(), comp); // using custom sort
sort(v.begin(), v.end(), [](const T a, const T b) {
  if (cond) {
    return true;
  }
  return false;
}); // using lambda function
```

$f(n) = O(g(n))$	iff $\exists$ positive c, n_0 such that $0 \le f(n) \le cg(n)$ $\forall n \ge n_0$.
$f(n) = \Omega(g(n))$	iff $\exists$ positive c, n_0 such that $f(n) \ge cg(n) \ge 0$ $\forall n \ge n_0$.
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
$f(n) = o(g(n))$	iff $\lim_{n\to\infty} f(n)/g(n) = 0$.
$\lim\limits_{n\to\infty} a_n = a$	iff $\forall \epsilon > 0$, $\exists n_0$ such that $\lvert a_n - a\rvert < \epsilon$, $\forall n \ge n_0$.
$\sup S$	least $b \in \mathbb{R}$ such that $b \ge s$, $\forall s \in S$.
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \le s$, $\forall s \in S$.
$\liminf\limits_{n\to\infty} a_n$	$\lim\limits_{n\to\infty} \inf\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\limsup\limits_{n\to\infty} a_n$	$\lim\limits_{n\to\infty} \sup\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\binom{n}{k}$	Combinations: Size k subsets of a size n set.
$\genfrac[]{0pt}{}{n}{k}$	Stirling numbers (1st kind): Arrangements of an n element set into k cycles.
$\genfrac\{\}{0pt}{}{n}{k}$	Stirling numbers (2nd kind): Partitions of an n element set into k non-empty sets.
$\genfrac\langle\rangle{0pt}{}{n}{k}$	1st order Eulerian numbers: Permutations $\pi_1\pi_2\ldots\pi_n$ on $\{1, 2, \ldots, n\}$ with k ascents.
$\left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle$	2nd order Eulerian numbers.
C_n	Catalan Numbers: Binary trees with $n+1$ vertices.

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \qquad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \qquad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}.$$

In general:

$$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\left[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}\left((i+1)^{m+1} - i^{m+1} - (m+1)i^m\right)\right]$$

$$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k} B_k n^{m+1-k}.$$

Geometric series:

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, \quad c \ne 1, \qquad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \qquad \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, \quad |c| < 1,$$

$$\sum_{i=0}^{n} ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \ne 1, \qquad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad |c| < 1.$$

Harmonic series:

$$H_n = \sum_{i=1}^{n} \frac{1}{i}, \qquad \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$$

$$\sum_{i=1}^{n} H_i = (n+1)H_n - n, \qquad \sum_{i=1}^{n}\binom{i}{m}H_i = \binom{n+1}{m+1}\left(H_{n+1} - \frac{1}{m+1}\right).$$

1. $\displaystyle \binom{n}{k} = \frac{n!}{(n-k)!k!}$, **2.** $\displaystyle \sum_{k=0}^{n}\binom{n}{k} = 2^n$, **3.** $\displaystyle \binom{n}{k} = \binom{n}{n-k}$,

4. $\displaystyle \binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$, **5.** $\displaystyle \binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$,

6. $\displaystyle \binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$, **7.** $\displaystyle \sum_{k=0}^{n}\binom{r+k}{k} = \binom{r+n+1}{n}$,

8. $\displaystyle \sum_{k=0}^{n}\binom{k}{m} = \binom{n+1}{m+1}$, **9.** $\displaystyle \sum_{k=0}^{n}\binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}$,

10. $\displaystyle \binom{n}{k} = (-1)^k\binom{k-n-1}{k}$, **11.** $\displaystyle \genfrac\{\}{0pt}{}{n}{1} = \genfrac\{\}{0pt}{}{n}{n} = 1$,

12. $\displaystyle \genfrac\{\}{0pt}{}{n}{2} = 2^{n-1} - 1$, **13.** $\displaystyle \genfrac\{\}{0pt}{}{n}{k} = k\genfrac\{\}{0pt}{}{n-1}{k} + \genfrac\{\}{0pt}{}{n-1}{k-1}$,

14. $\displaystyle \genfrac[]{0pt}{}{n}{1} = (n-1)!$, **15.** $\displaystyle \genfrac[]{0pt}{}{n}{2} = (n-1)!H_{n-1}$, **16.** $\displaystyle \genfrac[]{0pt}{}{n}{n} = 1$, **17.** $\displaystyle \genfrac[]{0pt}{}{n}{k} \ge \genfrac\{\}{0pt}{}{n}{k}$,

18. $\displaystyle \genfrac[]{0pt}{}{n}{k} = (n-1)\genfrac[]{0pt}{}{n-1}{k} + \genfrac[]{0pt}{}{n-1}{k-1}$, **19.** $\displaystyle \genfrac\{\}{0pt}{}{n}{n-1} = \genfrac[]{0pt}{}{n}{n-1} = \binom{n}{2}$, **20.** $\displaystyle \sum_{k=0}^{n}\genfrac[]{0pt}{}{n}{k} = n!$, **21.** $\displaystyle C_n = \frac{1}{n+1}\binom{2n}{n}$,

22. $\displaystyle \genfrac\langle\rangle{0pt}{}{n}{0} = \genfrac\langle\rangle{0pt}{}{n}{n-1} = 1$, **23.** $\displaystyle \genfrac\langle\rangle{0pt}{}{n}{k} = \genfrac\langle\rangle{0pt}{}{n}{n-1-k}$, **24.** $\displaystyle \genfrac\langle\rangle{0pt}{}{n}{k} = (k+1)\genfrac\langle\rangle{0pt}{}{n-1}{k} + (n-k)\genfrac\langle\rangle{0pt}{}{n-1}{k-1}$,

25. $\displaystyle \genfrac\langle\rangle{0pt}{}{0}{k} = \begin{cases} 1 & \text{if } k = 0, \\ 0 & \text{otherwise} \end{cases}$ **26.** $\displaystyle \genfrac\langle\rangle{0pt}{}{n}{1} = 2^n - n - 1$, **27.** $\displaystyle \genfrac\langle\rangle{0pt}{}{n}{2} = 3^n - (n+1)2^n + \binom{n+1}{2}$,

28. $\displaystyle x^n = \sum_{k=0}^{n}\genfrac\langle\rangle{0pt}{}{n}{k}\binom{x+k}{n}$, **29.** $\displaystyle \genfrac\langle\rangle{0pt}{}{n}{m} = \sum_{k=0}^{m}\binom{n+1}{k}(m+1-k)^n(-1)^k$, **30.** $\displaystyle m!\genfrac\{\}{0pt}{}{n}{m} = \sum_{k=0}^{n}\genfrac\langle\rangle{0pt}{}{n}{k}\binom{k}{n-m}$,

31. $\displaystyle \genfrac\langle\rangle{0pt}{}{n}{m} = \sum_{k=0}^{n}\genfrac\{\}{0pt}{}{n}{k}\binom{n-k}{m}(-1)^{n-k-m}k!$, **32.** $\displaystyle \left\langle\!\!\left\langle {n \atop 0} \right\rangle\!\!\right\rangle = 1$, **33.** $\displaystyle \left\langle\!\!\left\langle {n \atop n} \right\rangle\!\!\right\rangle = 0$ for $n \ne 0$,

34. $\displaystyle \left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle = (k+1)\left\langle\!\!\left\langle {n-1 \atop k} \right\rangle\!\!\right\rangle + (2n-1-k)\left\langle\!\!\left\langle {n-1 \atop k-1} \right\rangle\!\!\right\rangle$, **35.** $\displaystyle \sum_{k=0}^{n}\left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle = \frac{(2n)^{\underline{n}}}{2^n}$,

36. $\displaystyle \genfrac\{\}{0pt}{}{x}{x-n} = \sum_{k=0}^{n}\left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle\binom{x+n-1-k}{2n}$, **37.** $\displaystyle \genfrac\{\}{0pt}{}{n+1}{m+1} = \sum_{k}\binom{n}{k}\genfrac\{\}{0pt}{}{k}{m} = \sum_{k=0}^{n}\genfrac\{\}{0pt}{}{k}{m}(m+1)^{n-k}$, The Chinese remainder theorem: There exists a number C such that:

$$\begin{aligned} C &\equiv r_1 \bmod m_1 \\ \vdots\ &\ \vdots\ \ \vdots \\ C &\equiv r_n \bmod m_n \end{aligned}$$

if m_i and m_j are relatively prime for $i \neq j$.

Euler's function: $\phi(x)$ is the number of positive integers less than x relatively prime to x. If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of x then

$$\phi(x) = \prod_{i=1}^n p_i^{e_i-1}(p_i - 1).$$

Euler's theorem: If a and b are relatively prime then

$$1 \equiv a^{\phi(b)} \bmod b.$$

Fermat's theorem:

$$1 \equiv a^{p-1} \bmod p.$$

The Euclidean algorithm: if $a > b$ are integers then

$$\gcd(a, b) = \gcd(a \bmod b, b).$$

If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of x then

$$S(x) = \sum_{d|x} d = \prod_{i=1}^n \frac{p_i^{e_i+1} - 1}{p_i - 1}.$$

Perfect Numbers: x is an even perfect number iff $x = 2^{n-1}(2^n - 1)$ and $2^n - 1$ is prime.
Wilson's theorem: n is a prime iff

$$(n-1)! \equiv -1 \bmod n.$$

Möbius inversion:

$$\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of} \\ & r \text{ distinct primes.} \end{cases}$$

If

$$G(a) = \sum_{d|a} F(d),$$

then

$$F(a) = \sum_{d|a} \mu(d) G\Big(\frac{a}{d}\Big).$$

Prime numbers:

$$p_n = n \ln n + n \ln\ln n - n + n\frac{\ln\ln n}{\ln n} + O\Big(\frac{n}{\ln n}\Big),$$

$$\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3} + O\Big(\frac{n}{(\ln n)^4}\Big).$$

Definitions:

Loop	An edge connecting a vertex to itself.
Directed	Each edge has a direction.
Simple	Graph with no loops or multi-edges.
Walk	A sequence $v_0 e_1 v_1 \ldots e_\ell v_\ell$.
Trail	A walk with distinct edges.
Path	A trail with distinct vertices.
Connected	A graph where there exists a path between any two vertices.
Component	A maximal connected subgraph.
Tree	A connected acyclic graph.
Free tree	A tree with no root.
DAG	Directed acyclic graph.
Eulerian	Graph with a trail visiting each edge exactly once.
Hamiltonian	Graph with a cycle visiting each vertex exactly once.
Cut	A set of edges whose removal increases the number of components.
Cut-set	A minimal cut.
Cut edge	A size 1 cut.
k-Connected	A graph connected with the removal of any $k-1$ vertices.
k-Tough	$\forall S \subseteq V, S \neq \emptyset$ we have $k \cdot c(G - S) \leq \|S\|$.
k-Regular	A graph where all vertices have degree k.
k-Factor	A k-regular spanning subgraph.
Matching	A set of edges, no two of which are adjacent.
Clique	A set of vertices, all of which are adjacent.
Ind. set	A set of vertices, none of which are adjacent.
Vertex cover	A set of vertices which cover all edges.
Planar graph	A graph which can be embeded in the plane.
Plane graph	An embedding of a planar graph.

$$\sum_{v \in V} \deg(v) = 2m.$$

If G is planar then $n - m + f = 2$, so

$$f \leq 2n - 4, \quad m \leq 3n - 6.$$

Any planar graph has a vertex with degree ≤ 5.

Notation:

$E(G)$	Edge set
$V(G)$	Vertex set
$c(G)$	Number of components
$G[S]$	Induced subgraph
$\deg(v)$	Degree of v
$\Delta(G)$	Maximum degree
$\delta(G)$	Minimum degree
$\chi(G)$	Chromatic number
$\chi_E(G)$	Edge chromatic number
G^c	Complement graph
K_n	Complete graph
K_{n_1,n_2}	Complete bipartite graph
$\mathrm{r}(k,\ell)$	Ramsey number

Geometry

Projective coordinates: triples (x, y, z), not all x, y and z zero.

$$(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$$

Cartesian	Projective
(x, y)	$(x, y, 1)$
$y = mx + b$	$(m, -1, b)$
$x = c$	$(1, 0, -c)$

Distance formula, L_p and L_∞ metric:

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$

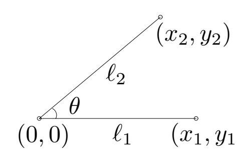
$$\big[|x_1 - x_0|^p + |y_1 - y_0|^p\big]^{1/p},$$

$$\lim_{p \to \infty} \big[|x_1 - x_0|^p + |y_1 - y_0|^p\big]^{1/p}.$$

Area of triangle (x_0, y_0), (x_1, y_1) and (x_2, y_2):

$$\tfrac{1}{2} \operatorname{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$$

Angle formed by three points:



$$\cos\theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{\ell_1 \ell_2}.$$

Line through two points (x_0, y_0) and (x_1, y_1):

$$\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$$

Area of circle, volume of sphere:

$$A = \pi r^2, \qquad V = \tfrac{4}{3}\pi r^3.$$

If I have seen farther than others, it is because I have stood on the shoulders of giants.
– Issac Newton