Contents

1 Basic

1.1 createFile

```
// Linux
for i in {A..Z}; do cp tem.cpp $i.cpp; done
// Windows
'A'..'Z' | % { cp tem.cpp "$_.cpp" }
```

1.2 run

```
g++ -std=c++20 -DPEPPA -Wall -Wextra -Wshadow -O2 -fsanitize=
    address,undefined $1.cpp -o $1 && ./$1
```

1.3 tem

```
#include <bits/stdc++.h>
#pragma GCC optimize("Ofast,unroll-loops,no-stack-protector")
#pragma GCC target("sse3","sse2","sse")

using namespace std;
using i64 = long long;

#define int i64
#define all(a) a.begin(), a.end()
#define rep(a, b, c) for (int a = b; a < c; a++)

bool chmin(auto& a, auto b) { return (b < a and (a = b, true));
    }
bool chmax(auto& a, auto b) { return (a < b and (a = b, true));
    }

void solve() {
  //
}

int32_t main() {
  std::ios::sync_with_stdio(false);
  std::cin.tie(nullptr);

  int t = 1;
  std::cin >> t;

  while (t--) {
    solve();
  }

  return 0;
}
```

1.4 debug

```
#ifdef PEPPA
template <typename R>
concept I = ranges::range<R> && !std::same_as<ranges::
    range_value_t<R>, char>;
template <typename A, typename B>
std::ostream& operator<<(std::ostream& o, const std::pair<A, B
    >& p) {
  return o << "(" << p.first << ", " << p.second << ")";
}
template <I T>
std::ostream& operator<<(std::ostream& o, const T& v) {
  o << "{";
  int f = 0;
  for (auto &&i : v) o << (f++ ? " " : "") << i;
  return o << "}";
}
void debug__(int c, auto&&... a) {
  std::cerr << "\e[1;" << c << "m";
  (..., (std::cerr << a << " "));
  std::cerr << "\e[0m" << std::endl;
}
#define debug_(c, x...) debug__(c, __LINE__, "[" + std::string
    (#x) + "]", x)
#define debug(x...) debug_(93, x)
#else
#define debug(x...) void(0)
#endif
```

1.5 run.bat

```
@echo off
g++ -std=c++23 -DPEPPA -Wall -Wextra -Wshadow -O2 %1.cpp -o %1.
    exe
if "%2" == "" ("%1.exe") else ("%1.exe" < "%2")
```

1.6 random

```
std::mt19937_64 rng(std::chrono::steady_clock::now().
    time_since_epoch().count());
inline i64 rand(i64 l, i64 r) { return std::
    uniform_int_distribution<i64>(l, r)(rng); }
```

1.7 TempleHash

```
cat file.cpp | cpp -dD -P -fpreprocessed | tr -d "[:space:]" |
    md5sum | cut -c-6
```

2 Misc

2.1 FastIO

```
#include <unistd.h>
int OP;
char OB[65536];
inline char RC() {
  static char buf[65536], *p = buf, *q = buf;
  return p == q && (q = (p = buf) + read(0, buf, 65536)) == buf
      ? -1 : *p++;
}
inline int R() {
  static char c;
  while ((c = RC()) < '0');
  int a = c ^ '0';
  while ((c = RC()) >= '0') a *= 10, a += c ^ '0';
  return a;
}
inline void W(int n) {
  static char buf[12], p;
  if (n == 0) OB[OP++] = '0';
  p = 0;
  while (n) buf[p++] = '0' + (n % 10), n /= 10;
  for (--p; p >= 0; --p) OB[OP++] = buf[p];
  if (OP > 65520) write(1, OB, OP), OP = 0;
}

// another FastIO
char buf[1 << 21], *p1 = buf, *p2 = buf;
inline char getc() {
  return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 1 << 21,
      stdin), p1 == p2) ? 0 : *p1++;
}
template<typename T> void Cin(T &a) {
  T res = 0; int f = 1;
  char c = getc();
  for (; c < '0' || c > '9'; c = getc()) {
    if (c == '-') f = -1;
  }
  for (; c >= '0' && c <= '9'; c = getc()) {
    res = res * 10 + c - '0';
  }
  a = f * res;
}
template<typename T, typename... Args> void Cin(T &a, Args &...
    args) {
  Cin(a), Cin(args...);
}
template<typename T> void Cout(T x) { // there's no '\n' in
    output
  if (x < 0) putchar('-'), x = -x;
  if (x > 9) Cout(x / 10);
  putchar(x % 10 + '0');
}
```

2.2 stress.sh

```
#!/usr/bin/env bash
g++ $1.cpp -o $1
g++ $2.cpp -o $2
g++ $3.cpp -o $3
for i in {1..100} ; do
  ./$3 > input.txt
  # st=$(date +%s%N)
  ./$1 < input.txt > output1.txt
  # echo "$((($(date +%s%N) - $st)/1000000))ms"
  ./$2 < input.txt > output2.txt
  if cmp --silent -- "output1.txt" "output2.txt" ; then
    continue
  fi
  echo Input:
  cat input.txt
  echo Your Output:
  cat output1.txt
  echo Correct Output:
  cat output2.txt
  exit 1
done
echo OK!
./stress.sh main good gen
```

2.3 stress.bat

```
@echo off
setlocal EnableExtensions

g++ -std=c++20 -O3 "%1.cpp" -o "%1.exe"
g++ -std=c++20 -O3 "%2.cpp" -o "%2.exe"
g++ -std=c++20 -O3 "%3.cpp" -o "%3.exe"

for /l %%i in (1,1,100) do (
 "%3.exe" > input.txt
 "%1.exe" < input.txt > output1.txt
 "%2.exe" < input.txt > output2.txt

 fc /b output1.txt output2.txt >nul
 if errorlevel 1 (
  echo Input:
  type input.txt
  echo Your Output:
  type output1.txt
  echo Correct Output:
  type output2.txt
  exit /b 1
 )
)

@REM ./stress main good gen
```

2.4 Timer

```
struct Timer {
  int t;
  bool enable = false;

  void start() {
    enable = true;
    t = std::clock();
  }
  int msecs() {
    assert(enable);
    return (std::clock() - t) * 1000 / CLOCKS_PER_SEC;
  }
};
```

2.5 MinPlusConvolution

```
// a is convex a[i+1]-a[i] <= a[i+2]-a[i+1]
vector<int> min_plus_convolution(vector<int> &a, vector<int> &b
    ) {
 int n = ssize(a), m = ssize(b);
 vector<int> c(n + m - 1, INF);
 auto dc = [&](auto Y, int l, int r, int jl, int jr) {
  if (l > r) return;
  int mid = (l + r) / 2, from = -1, &best = c[mid];
  for (int j = jl; j <= jr; ++j)
   if (int i = mid - j; i >= 0 && i < n)
    if (best > a[i] + b[j])
     best = a[i] + b[j], from = j;
  Y(Y, l, mid - 1, jl, from), Y(Y, mid + 1, r, from, jr);
 };
 return dc(dc, 0, n - 1 + m - 1, 0, m - 1), c;
}
```

2.6 PyTrick

```
import sys
input = sys.stdin.readline

from itertools import permutations
op = ['+', '-', '*', '']
a, b, c, d = input().split()
ans = set()
for (x,y,z,w) in permutations([a, b, c, d]):
  for op1 in op:
    for op2 in op:
      for op3 in op:
```

```
        val = eval(f"{x}{op1}{y}{op2}{z}{op3}{w}")
        if (op1 == '' and op2 == '' and op3 == '') or val < 0:
          continue
        ans.add(val)
print(len(ans))

map(int,input().split())
arr2d = [ [ list(map(int,input().split())) ] for i in range(N)
     ] # N*M

from decimal import *
from fractions import *
s = input()
n = int(input())
f = Fraction(s)
g = Fraction(s).limit_denominator(n)
h = f * 2 - g
if h.numerator <= n and h.denominator <= n and h < g:
  g = h
print(g.numerator, g.denominator)

from fractions import Fraction
x = Fraction(1, 2), y = Fraction(1)
print(x.as_integer_ratio()) # print 1/2
print(x.is_integer())
print(x.__round__())
print(float(x))

r = Fraction(input())
N = int(input())
r2 = r - 1 / Fraction(N) ** 2
ans = r.limit_denominator(N)
ans2 = r2.limit_denominator(N)
if ans2 < ans and 0 <= ans2 <= 1 and abs(ans - r) >= abs(ans2 -
      r):
  ans = ans2
print(ans.numerator,ans.denominator)
```

3 Data Structure

3.1 Fenwick Tree

```
template<class T>
struct Fenwick {
  int n;
  vector<T> a;
  Fenwick(int _n) : n(_n), a(_n) {}
  void add(int p, T x) {
    for (int i = p; i < n; i = i | (i + 1)) {
      a[i] = a[i] + x;
    }
  }
  T qry(int p) { // sum [0, p]
    T s{};
    for (int i = p; i >= 0; i = (i & (i + 1)) - 1) {
      s = s + a[i];
    }
    return s;
  }
  T qry(int l, int r) { // sum [l, r)
    return qry(r - 1) - qry(l - 1);
  }
  pair<int, T> select(T k) { // [first position >= k, sum [0, p
    )
    T s{};
    int p = 0;
    for (int i = 1 << __lg(n); i; i >>= 1) {
      if (p + i <= n and s + a[p + i - 1] < k) {
        p += i;
        s = s + a[p - 1];
      }
    }
    return {p, s};
  }
};
```

3.2 Li Chao

```
struct Line {
  // y = ax + b
  i64 a{0}, b{-inf<i64>};
  i64 operator()(i64 x) {
    return a * x + b;
  }
};
// max LiChao
struct Seg {
  int l, r;
  Seg *ls{}, *rs{};
  Line f{};
  Seg(int l, int r) : l(l), r(r) {}
  void add(Line g) {
    int m = (l + r) / 2;
    if (g(m) > f(m)) {
      swap(g, f);
    }
    if (g.b == -inf<i64> or r - l == 1) {
      return;
    }
    if (g.a < f.a) {
      if (!ls) {
        ls = new Seg(l, m);
      }
      ls->add(g);
    } else {
      if (!rs) {
        rs = new Seg(m, r);
      }
      rs->add(g);
    }
  }
  i64 qry(i64 x) {
    if (f.b == -inf<i64>) {
      return -inf<i64>;
    }
    int m = (l + r) / 2;
    i64 y = f(x);
    if (x < m and ls) {
      chmax(y, ls->qry(x));
    } else if (x >= m and rs) {
      chmax(y, rs->qry(x));
    }
    return y;
  }
};
```

3.3 PBDS

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template<typename T> using RBT = tree<T, null_type, less<T>,
     rb_tree_tag, tree_order_statistics_node_update>;
/*
.find_by_order(k) 回傳第 k 小的值（based-0）
.order_of_key(k) 回傳有多少元素比 k 小
*/
struct custom_hash {
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }

  size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM = chrono::steady_clock::
     now().time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};
// gp_hash_table<int, int, custom_hash> ss;
```

3.4 ODT

```
map<int, int> odt;
// initialize edges odt[1] and odt[n + 1]
auto split = [&](const int &x) -> void {
  const auto it = prev(odt.upper_bound(x));
  odt[x] = it->second;
};
auto merge = [&](const int &l, const int &r) -> void {
  auto itl = odt.lower_bound(l), itr = odt.lower_bound(r + 1);
  for (; itl != itr; itl = odt.erase(itl)) {
    // do something
  }
  // assign value to odt[l]
};
```

3.5 Sparse Table

```
template<class T>
struct SparseTable{
  function<T(T, T)> F;
  vector<vector<T>> sp;
  SparseTable(vector<T> &a, const auto &f) {
    F = f;
    int n = a.size();
    sp.resize(n, vector<T>(__lg(n) + 1));
    for (int i = n - 1; i >= 0; i--) {
      sp[i][0] = a[i];
      for (int j = 1; i + (1 << j) <= n; j++) {
        sp[i][j] = F(sp[i][j - 1], sp[i + (1 << j - 1)][j - 1])
       ;
      }
    }
  }
  T query(int l, int r) { // [l, r)
    int k = __lg(r - l);
    return F(sp[l][k], sp[r - (1 << k)][k]);
  }
};
```

3.6 Splay

```
struct Node {
  Node *ch[2]{}, *p{};
  Info info{}, sum{};
  Tag tag{};
  int size{};
  bool rev{};
} pool[int(1E5 + 10)], *top = pool;
Node *newNode(Info a) {
  Node *t = top++;
  t->info = t->sum = a;
  t->size = 1;
  return t;
}
int size(const Node *x) { return x ? x->size : 0; }
Info get(const Node *x) { return x ? x->sum : Info{}; }
int dir(const Node *x) { return x->p->ch[1] == x; }
bool nroot(const Node *x) { return x->p and x->p->ch[dir(x)] ==
      x; }
void reverse(Node *x) { if (x) x->rev = !x->rev; }
void update(Node *x, const Tag &f) {
  if (!x) return;
  f(x->tag);
  f(x->info);
  f(x->sum);
}
void push(Node *x) {
  if (x->rev) {
    swap(x->ch[0], x->ch[1]);
    reverse(x->ch[0]);
    reverse(x->ch[1]);
    x->rev = false;
  }
  update(x->ch[0], x->tag);
  update(x->ch[1], x->tag);
  x->tag = Tag{};
}
void pull(Node *x) {
  x->size = size(x->ch[0]) + 1 + size(x->ch[1]);
  x->sum = get(x->ch[0]) + x->info + get(x->ch[1]);
}
void rotate(Node *x) {
  Node *y = x->p, *z = y->p;
  push(y);
  int d = dir(x);
  push(x);
  Node *w = x->ch[d ^ 1];
  if (nroot(y)) {
    z->ch[dir(y)] = x;
  }
  if (w) {
    w->p = y;
  }
  (x->ch[d ^ 1] = y)->ch[d] = w;
  (y->p = x)->p = z;
  pull(y);
  pull(x);
}
void splay(Node *x) {
  while (nroot(x)) {
    Node *y = x->p;
    if (nroot(y)) {
      rotate(dir(x) == dir(y) ? y : x);
    }
    rotate(x);
  }
}
Node *nth(Node *x, int k) {
  assert(size(x) > k);
  while (true) {
    push(x);
    int left = size(x->ch[0]);
    if (left > k) {
      x = x->ch[0];
    } else if (left < k) {
      k -= left + 1;
      x = x->ch[1];
    } else {
      break;
    }
  }
  splay(x);
  return x;
}
Node *split(Node *x) {
  assert(x);
  push(x);
  Node *l = x->ch[0];
  if (l) l->p = x->ch[0] = nullptr;
  pull(x);
  return l;
}
Node *join(Node *x, Node *y) {
  if (!x or !y) return x ? x : y;
  y = nth(y, 0);
  push(y);
  y->ch[0] = x;
  if (x) x->p = y;
  pull(y);
  return y;
}
Node *find_first(Node *x, auto &&pred) {
  Info pre{};
  while (true) {
    push(x);
    if (pred(pre + get(x->ch[0]))) {
      x = x->ch[0];
    } else if (pred(pre + get(x->ch[0]) + x->info) or !x->ch
     [1]) {
      break;
    } else {
      pre = pre + get(x->ch[0]) + x->info;
      x = x->ch[1];
    }
  }
  splay(x);
  return x;
}
```

3.7 Treap

```
struct Treap {
  Treap *l, *r;
  int key, size;
  Treap(int k) : l(nullptr), r(nullptr), key(k), size(1) {}
  void pull();
  void push() {};
};
inline int SZ(Treap *p) {
  return p == nullptr ? 0 : p->size;
}
void Treap::pull() {
  size = 1 + SZ(l) + SZ(r);
}
Treap *merge(Treap *a, Treap *b) {
  if (!a || !b) return a ? a : b;
  if (rand() % (SZ(a) + SZ(b)) < SZ(a)) {
    return a->push(), a->r = merge(a->r, b), a->pull(), a;
  }
  return b->push(), b->l = merge(a, b->l), b->pull(), b;
}
// <= k, > k
void split(Treap *p, Treap *&a, Treap *&b, int k) { // by key
  if (!p) return a = b = nullptr, void();
```

```
  p->push();
  if (p->key <= k) {
    a = p, split(p->r, a->r, b, k), a->pull();
  } else {
    b = p, split(p->l, a, b->l, k), b->pull();
  }
}
// k, n - k
void split2(Treap *p, Treap *&a, Treap *&b, int k) { // by size
  if (!p) return a = b = nullptr, void();
  p->push();
  if (SZ(p->l) + 1 <= k) {
    a = p, split2(p->r, a->r, b, k - SZ(p->l) - 1);
  } else {
    b = p, split2(p->l, a, b->l, k);
  }
  p->pull();
}
void insert(Treap *&p, int k) {
  Treap *l, *r;
  p->push(), split(p, l, r, k);
  p = merge(merge(l, new Treap(k)), r);
  p->pull();
}
bool erase(Treap *&p, int k) {
  if (!p) return false;
  if (p->key == k) {
    Treap *t = p;
    p->push(), p = merge(p->l, p->r);
    delete t;
    return true;
  }
  Treap *&t = k < p->key ? p->l : p->r;
  return erase(t, k) ? p->pull(), true : false;
}
int Rank(Treap *p, int k) { // # of key < k
  if (!p) return 0;
  if (p->key < k) return SZ(p->l) + 1 + Rank(p->r, k);
  return Rank(p->l, k);
}
Treap *kth(Treap *p, int k) { // 1-base
  if (k <= SZ(p->l)) return kth(p->l, k);
  if (k == SZ(p->l) + 1) return p;
  return kth(p->r, k - SZ(p->l) - 1);
}
// pref: kth(Rank(x)), succ: kth(Rank(x+1)+1)
tuple<Treap*, Treap*, Treap*> interval(Treap *&o, int l, int r)
      { // 1-based
  Treap *a, *b, *c; // b: [l, r]
  split2(o, a, b, l - 1), split2(b, b, c, r - l + 1);
  return make_tuple(a, b, c);
}
```

4 Matching and Flow

4.1 Dinic

```
template <typename T>
struct Dinic {
  const T INF = numeric_limits<T>::max() / 2;
  struct edge {
    int v, r; T rc;
  };
  vector<vector<edge>> adj;
  vector<T> dis, it;
  Dinic(int n) : adj(n), dis(n), it(n) {}
  void add_edge(int u, int v, T c) {
    adj[u].pb({v, adj[v].size(), c});
    adj[v].pb({u, adj[u].size() - 1, 0});
  }
  bool bfs(int s, int t) {
    fill(all(dis), INF);
    queue<int> q;
    q.push(s);
    dis[s] = 0;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (const auto& [v, r, rc] : adj[u]) {
        if (dis[v] < INF || rc == 0) continue;
        dis[v] = dis[u] + 1;
        q.push(v);
      }
    }
    return dis[t] < INF;
  }

  T dfs(int u, int t, T cap) {
    if (u == t || cap == 0) return cap;
    for (int &i = it[u]; i < (int)adj[u].size(); ++i) {
      auto &[v, r, rc] = adj[u][i];
      if (dis[v] != dis[u] + 1) continue;
      T tmp = dfs(v, t, min(cap, rc));
      if (tmp > 0) {
        rc -= tmp;
        adj[v][r].rc += tmp;
        return tmp;
      }
    }
    return 0;
  }

  T flow(int s, int t) {
    T ans = 0, tmp;
    while (bfs(s, t)) {
      fill(all(it), 0);
      while ((tmp = dfs(s, t, INF)) > 0) {
        ans += tmp;
      }
    }
    return ans;
  }

  bool inScut(int u) { return dis[u] < INF; }
};
```

4.2 General Matching

```
struct GeneralMatching { // n <= 500
  const int BLOCK = 10;
  int n;
  vector<vector<int> > g;
  vector<int> hit, mat;
  priority_queue<pair<int, int>, vector<pair<int, int>>,
     greater<pair<int, int>>> unmat;
  GeneralMatching(int _n) : n(_n), g(_n), mat(n, -1), hit(n) {}
  void add_edge(int a, int b) { // 0 <= a != b < n
    g[a].push_back(b);
    g[b].push_back(a);
  }
  int get_match() {
    for (int i = 0; i < n; i++) if (!g[i].empty()) {
      unmat.emplace(0, i);
    }
    // If WA, increase this
    // there are some cases that need >=1.3*n^2 steps for BLOCK
     =1
    // no idea what the actual bound needed here is.
    const int MAX_STEPS = 10 + 2 * n + n * n / BLOCK / 2;
    mt19937 rng(random_device{}());
    for (int i = 0; i < MAX_STEPS; ++i) {
      if (unmat.empty()) break;
      int u = unmat.top().second;
      unmat.pop();
      if (mat[u] != -1) continue;
      for (int j = 0; j < BLOCK; j++) {
        ++hit[u];
        auto &e = g[u];
        const int v = e[rng() % e.size()];
        mat[u] = v;
        swap(u, mat[v]);
        if (u == -1) break;
      }
      if (u != -1) {
        mat[u] = -1;
        unmat.emplace(hit[u] * 100ULL / (g[u].size() + 1), u);
      }
    }
    int siz = 0;
    for (auto e : mat) siz += (e != -1);
    return siz / 2;
  }
};
```

4.3 KM

```
template<class T>
T KM(const vector<vector<T>> &w) {
  const T INF = numeric_limits<T>::max() / 2;
```

```
  const int n = w.size();
  vector<T> lx(n), ly(n);
  vector<int> mx(n, -1), my(n, -1), pa(n);
  auto augment = [&](int y) {
    for (int x, z; y != -1; y = z) {
      x = pa[y];
      z = mx[x];
      my[y] = x;
      mx[x] = y;
    }
  };
  auto bfs = [&](int s) {
    vector<T> sy(n, INF);
    vector<bool> vx(n), vy(n);
    queue<int> q;
    q.push(s);
    while (true) {
      while (q.size()) {
        int x = q.front();
        q.pop();
        vx[x] = 1;
        for (int y = 0; y < n; y++) {
          if (vy[y]) continue;
          T d = lx[x] + ly[y] - w[x][y];
          if (d == 0) {
            pa[y] = x;
            if (my[y] == -1) {
              augment(y);
              return;
            }
            vy[y] = 1;
            q.push(my[y]);
          } else if (chmin(sy[y], d)) {
            pa[y] = x;
          }
        }
      }
      T cut = INF;
      for (int y = 0; y < n; y++)
        if (!vy[y])
          chmin(cut, sy[y]);
      for (int j = 0; j < n; j++) {
        if (vx[j]) lx[j] -= cut;
        if (vy[j]) ly[j] += cut;
        else sy[j] -= cut;
      }
      for (int y = 0; y < n; y++)
        if (!vy[y] and sy[y] == 0) {
          if (my[y] == -1) {
            augment(y);
            return;
          }
          vy[y] = 1;
          q.push(my[y]);
        }
    }
  };
  for (int x = 0; x < n; x++)
    lx[x] = ranges::max(w[x]);
  for (int x = 0; x < n; x++)
    bfs(x);
  T ans = 0;
  for (int x = 0; x < n; x++)
    ans += w[x][mx[x]];
  return ans;
}
```

4.4 MCMF

```
template<class T>
struct MCMF {
  const T INF = numeric_limits<T>::max() / 2;
  struct edge { int v, r; T f, w; };
  vector<vector<edge>> adj;
  const int n;
  MCMF(int n) : n(n), adj(n) {}
  void addEdge(int u, int v, T f, T c) {
    adj[u].push_back({v, ssize(adj[v]), f, c});
    adj[v].push_back({u, ssize(adj[u]) - 1, 0, -c});
  }
  vector<T> dis;
  vector<bool> vis;
  bool spfa(int s, int t) {
    queue<int> que;
    dis.assign(n, INF);
    vis.assign(n, false);
    que.push(s);
    vis[s] = 1;
    dis[s] = 0;
    while (!que.empty()) {
      int u = que.front(); que.pop();
      vis[u] = 0;
      for (auto [v, _, f, w] : adj[u])
        if (f && chmin(dis[v], dis[u] + w))
          if (!vis[v]) {
            que.push(v);
            vis[v] = 1;
          }
    }
    return dis[t] != INF;
  }
  T dfs(int u, T in, int t) {
    if (u == t) return in;
    vis[u] = 1;
    T out = 0;
    for (auto &[v, rev, f, w] : adj[u])
      if (f && !vis[v] && dis[v] == dis[u] + w) {
        T x = dfs(v, min(in, f), t);
        in -= x;
        out += x;
        f -= x;
        adj[v][rev].f += x;
        if (!in) break;
      }
    if (in) dis[u] = INF;
    vis[u] = 0;
    return out;
  }
  pair<T, T> flow(int s, int t) { // {flow, cost}
    T a = 0, b = 0;
    while (spfa(s, t)) {
      T x = dfs(s, INF, t);
      a += x;
      b += x * dis[t];
    }
    return {a, b};
  }
};
```

4.5 Model

- Maximum/Minimum flow with lower bound / Circulation problem
  1. Construct super source S and sink T.
  2. For each edge (x, y, l, u), connect $x \to y$ with capacity $u - l$.
  3. For each vertex v, denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
  4. If $in(v) > 0$, connect $S \to v$ with capacity $in(v)$, otherwise, connect $v \to T$ with capacity $-in(v)$.
     - To maximize, connect $t \to s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T. If $f \neq \sum_{v \in V, in(v)>0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
     - To minimize, let f be the maximum flow from S to T. Connect $t \to s$ with capacity ∞ and let the flow from S to T be f'. If $f + f' \neq \sum_{v \in V, in(v)>0} in(v)$, there's no solution. Otherwise, f' is the answer.
  5. The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.

- Construct minimum vertex cover from maximum matching M on bipartite graph (X, Y)
  1. Redirect every edge: $y \to x$ if $(x, y) \in M$, $x \to y$ otherwise.
  2. DFS from unmatched vertices in X.
  3. $x \in X$ is chosen iff x is unvisited.
  4. $y \in Y$ is chosen iff y is visited.

- Minimum cost cyclic flow
  1. Consruct super source S and sink T
  2. For each edge (x, y, c), connect $x \to y$ with $(cost, cap) = (c, 1)$ if $c > 0$, otherwise connect $y \to x$ with $(cost, cap) = (-c, 1)$
  3. For each edge with $c < 0$, sum these cost as K, then increase $d(y)$ by 1, decrease $d(x)$ by 1
  4. For each vertex v with $d(v) > 0$, connect $S \to v$ with $(cost, cap) = (0, d(v))$
  5. For each vertex v with $d(v) < 0$, connect $v \to T$ with $(cost, cap) = (0, -d(v))$
  6. Flow from S to T, the answer is the cost of the flow $C + K$

- Maximum density induced subgraph
  1. Binary search on answer, suppose we're checking answer T
  2. Construct a max flow model, let K be the sum of all weights
  3. Connect source $s \to v$, $v \in G$ with capacity K
  4. For each edge (u, v, w) in G, connect $u \to v$ and $v \to u$ with capacity w

5. For $v \in G$, connect it with sink $v \to t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
6. T is a valid answer if the maximum flow $f < K|V|$

- Minimum weight edge cover
  1. Change the weight of each edge to $\mu(u) + \mu(v) - w(u, v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v.
  2. Let the maximum weight matching of the graph be x, the answer will be $\sum \mu(v) - x$.

5 Geometry

5.1 Point

```cpp
using numbers::pi;
template<class T> inline constexpr T eps = numeric_limits<T>::
    epsilon() * 1E6;
using Real = long double;
struct Pt {
  Real x{}, y{};
  Pt operator+(Pt a) const { return {x + a.x, y + a.y}; }
  Pt operator-(Pt a) const { return {x - a.x, y - a.y}; }
  Pt operator*(Real k) const { return {x * k, y * k}; }
  Pt operator/(Real k) const { return {x / k, y / k}; }
  Real operator*(Pt a) const { return x * a.x + y * a.y; }
  Real operator^(Pt a) const { return x * a.y - y * a.x; }
  auto operator<=>(const Pt&) const = default;
  bool operator==(const Pt&) const = default;
};
int sgn(Real x) { return (x > -eps<Real>) - (x < eps<Real>); }
Real ori(Pt a, Pt b, Pt c) { return (b - a) ^ (c - a); }
bool argcmp(const Pt &a, const Pt &b) { // arg(a) < arg(b)
  int f = (Pt{a.y, -a.x} > Pt{} ? 1 : -1) * (a != Pt{});
  int g = (Pt{b.y, -b.x} > Pt{} ? 1 : -1) * (b != Pt{});
  return f == g ? (a ^ b) > 0 : f < g;
}
Pt rotate(Pt u) { return {-u.y, u.x}; }
Real abs2(Pt a) { return a * a; }
// floating point only
Pt rotate(Pt u, Real a) {
  Pt v{sinl(a), cosl(a)};
  return {u ^ v, u * v};
}
Real abs(Pt a) { return sqrtl(a * a); }
Real arg(Pt x) { return atan2l(x.y, x.x); }
Pt unit(Pt x) { return x / abs(x); }
```

5.2 Line

```cpp
struct Line {
  Pt a, b;
  Pt dir() const { return b - a; }
};
int PtSide(Pt p, Line L) {
  return sgn(ori(L.a, L.b, p)); // for int
  return sgn(ori(L.a, L.b, p) / abs(L.a - L.b));
}
bool PtOnSeg(Pt p, Line L) {
  return PtSide(p, L) == 0 and sgn((p - L.a) * (p - L.b)) <= 0;
}
Pt proj(Pt p, Line l) {
  Pt dir = unit(l.b - l.a);
  return l.a + dir * (dir * (p - l.a));
}
```

5.3 Circle

```cpp
struct Cir {
  Pt o;
  double r;
};
bool disjunct(const Cir &a, const Cir &b) {
  return sgn(abs(a.o - b.o) - a.r - b.r) >= 0;
}
bool contain(const Cir &a, const Cir &b) {
  return sgn(a.r - b.r - abs(a.o - b.o)) >= 0;
}
```

5.4 Point to Segment Distance

```cpp
double PtSegDist(Pt p, Line l) {
  double ans = min(abs(p - l.a), abs(p - l.b));
  if (sgn(abs(l.a - l.b)) == 0) return ans;
  if (sgn((l.a - l.b) * (p - l.b)) < 0) return ans;
  if (sgn((l.b - l.a) * (p - l.a)) < 0) return ans;
  return min(ans, abs(ori(p, l.a, l.b)) / abs(l.a - l.b));
}
double SegDist(Line l, Line m) {
  return PtSegDist({0, 0}, {l.a - m.a, l.b - m.b});
}
```

5.5 Point In Polygon

```cpp
int inPoly(Pt p, const vector<Pt> &P) {
  const int n = P.size();
  int cnt = 0;
  for (int i = 0; i < n; i++) {
    Pt a = P[i], b = P[(i + 1) % n];
    if (PtOnSeg(p, {a, b})) return 1; // on edge
    if ((sgn(a.y - p.y) == 1) ^ (sgn(b.y - p.y) == 1))
      cnt += sgn(ori(a, b, p));
  }
  return cnt == 0 ? 0 : 2; // out, in
}
```

5.6 Intersection of Line

```cpp
bool isInter(Line l, Line m) {
  if (PtOnSeg(m.a, l) or PtOnSeg(m.b, l) or
    PtOnSeg(l.a, m) or PtOnSeg(l.b, m))
    return true;
  return PtSide(m.a, l) * PtSide(m.b, l) < 0 and
      PtSide(l.a, m) * PtSide(l.b, m) < 0;
}
Pt LineInter(Line l, Line m) {
  double s = ori(m.a, m.b, l.a), t = ori(m.a, m.b, l.b);
  return (l.b * s - l.a * t) / (s - t);
}
bool strictInter(Line l, Line m) {
  int la = PtSide(m.a, l);
  int lb = PtSide(m.b, l);
  int ma = PtSide(l.a, m);
  int mb = PtSide(l.b, m);
  if (la == 0 and lb == 0) return false;
  return la * lb < 0 and ma * mb < 0;
}
```

5.7 Intersection of Circles

```cpp
vector<Pt> CircleInter(Cir a, Cir b) {
  double d2 = abs2(a.o - b.o), d = sqrt(d2);
  if (d < max(a.r, b.r) - min(a.r, b.r) || d > a.r + b.r)
    return {};
  Pt u = (a.o + b.o) / 2 + (a.o - b.o) * ((b.r * b.r - a.r * a.
    r) / (2 * d2));
  double A = sqrt((a.r + b.r + d) * (a.r - b.r + d) * (a.r + b.
    r - d) * (-a.r + b.r + d));
  Pt v = rotate(b.o - a.o) * A / (2 * d2);
  if (sgn(v.x) == 0 and sgn(v.y) == 0) return {u};
  return {u - v, u + v}; // counter clockwise of a
}
```

5.8 Intersection of Circle and Line

```cpp
vector<Pt> CircleLineInter(Cir c, Line l) {
  Pt H = proj(c.o, l);
  Pt dir = unit(l.b - l.a);
  double h = abs(H - c.o);
  if (sgn(h - c.r) > 0) return {};
  double d = sqrt(max((double)0., c.r * c.r - h * h));
  if (sgn(d) == 0) return {H};
  return {H - dir *d, H + dir * d};
  // Counterclockwise
}
```

5.9 Area of Circle Polygon

```cpp
double CirclePoly(Cir C, const vector<Pt> &P) {
  auto arg = [&](Pt p, Pt q) { return atan2(p ^ q, p * q); };
  double r2 = C.r * C.r / 2;
  auto tri = [&](Pt p, Pt q) {
    Pt d = q - p;
    auto a = (d * p) / abs2(d), b = (abs2(p) - C.r * C.r)/ abs2
    (d);
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a - sqrt(det)), t = min(1., -a + sqrt(det
    ));
    if (t < 0 or 1 <= s) return arg(p, q) * r2;
    Pt u = p + d * s, v = p + d * t;
    return arg(p, u) * r2 + (u ^ v) / 2 + arg(v, q) * r2;
  };
  double sum = 0.0;
  for (int i = 0; i < P.size(); i++)
```

```
  sum += tri(P[i] - C.o, P[(i + 1) % P.size()] - C.o);
  return sum;
}
```

5.10 Convex Hull

```
vector<Pt> BuildHull(vector<Pt> pt) {
  sort(all(pt));
  pt.erase(unique(all(pt)), pt.end());
  if (pt.size() <= 2) return pt;
  vector<Pt> hull;
  int sz = 1;
  rep (t, 0, 2) {
    rep (i, t, ssize(pt)) {
      while (ssize(hull) > sz && ori(hull.end()[-2], pt[i],
       hull.back()) >= 0)
        hull.pop_back();
      hull.pb(pt[i]);
    }
    sz = ssize(hull);
    reverse(all(pt));
  }
  hull.pop_back();
  return hull;
}
```

5.11 Convex Trick

```
struct Convex {
  int n;
  vector<Pt> A, V, L, U;
  Convex(const vector<Pt> &_A) : A(_A), n(_A.size()) { // n >=
     3
    auto it = max_element(all(A));
    L.assign(A.begin(), it + 1);
    U.assign(it, A.end()), U.push_back(A[0]);
    rep (i, 0, n) {
      V.push_back(A[(i + 1) % n] - A[i]);
    }
  }
  int inside(Pt p, const vector<Pt> &h, auto f) {
    auto it = lower_bound(all(h), p, f);
    if (it == h.end()) return 0;
    if (it == h.begin()) return p == *it;
    return 1 - sgn(ori(*prev(it), p, *it));
  }
  // 0: out, 1: on, 2: in
  int inside(Pt p) {
    return min(inside(p, L, less{}), inside(p, U, greater{}));
  }
  static bool cmp(Pt a, Pt b) { return sgn(a ^ b) > 0; }
  // A[i] is a far/closer tangent point
  int tangent(Pt v, bool close = true) {
    assert(v != Pt{});
    auto l = V.begin(), r = V.begin() + L.size() - 1;
    if (v < Pt{}) l = r, r = V.end();
    if (close) return (lower_bound(l, r, v, cmp) - V.begin()) %
     n;
    return (upper_bound(l, r, v, cmp) - V.begin()) % n;
  }
  // closer tangent point
  array<int, 2> tangent2(Pt p) {
    array<int, 2> t{-1, -1};
    if (inside(p) == 2) return t;
    if (auto it = lower_bound(all(L), p); it != L.end() and p
     == *it) {
      int s = it - L.begin();
      return {(s + 1) % n, (s - 1 + n) % n};
    }
    if (auto it = lower_bound(all(U), p, greater{}); it != U.
     end() and p == *it) {
      int s = it - U.begin() + L.size() - 1;
      return {(s + 1) % n, (s - 1 + n) % n};
    }
    for (int i = 0; i != t[0]; i = tangent((A[t[0] = i] - p),
     0));
    for (int i = 0; i != t[1]; i = tangent((p - A[t[1] = i]),
     1));
    return t;
  }
  int find(int l, int r, Line L) {
    if (r < l) r += n;
    int s = PtSide(A[l % n], L);
    return *ranges::partition_point(views::iota(l, r),
      [&](int m) {
        return PtSide(A[m % n], L) == s;
      }) - 1;
  };
  // Line A_x A_x+1 interset with L
  vector<int> intersect(Line L) {
    int l = tangent(L.a - L.b), r = tangent(L.b - L.a);
    if (PtSide(A[l], L) * PtSide(A[r], L) >= 0) return {};
    return {find(l, r, L) % n, find(r, l, L) % n};
  }
};
```

5.12 Half Plane Intersection

```
bool cover(Line L, Line P, Line Q) {
  // for double, i128 => Real
  i128 u = (Q.a - P.a) ^ Q.dir();
  i128 v = P.dir() ^ Q.dir();
  i128 x = P.dir().x * u + (P.a - L.a).x * v;
  i128 y = P.dir().y * u + (P.a - L.a).y * v;
  return sgn(x * L.dir().y - y * L.dir().x) * sgn(v) >= 0;
}
vector<Line> HPI(vector<Line> P) {
  sort(all(P), [&](Line l, Line m) {
    if (argcmp(l.dir(), m.dir())) return true;
    if (argcmp(m.dir(), l.dir())) return false;
    return ori(m.a, m.b, l.a) > 0;
  });
  int n = P.size(), l = 0, r = -1;
  for (int i = 0; i < n; i++) {
    if (i and !argcmp(P[i - 1].dir(), P[i].dir())) continue;
    while (l < r and cover(P[i], P[r - 1], P[r])) r--;
    while (l < r and cover(P[i], P[l], P[l + 1])) l++;
    P[++r] = P[i];
  }
  while (l < r and cover(P[l], P[r - 1], P[r])) r--;
  while (l < r and cover(P[r], P[l], P[l + 1])) l++;
  if (r - l <= 1 or !argcmp(P[l].dir(), P[r].dir()))
    return {}; // empty
  if (cover(P[l + 1], P[l], P[r]))
    return {}; // infinity
  return vector(P.begin() + l, P.begin() + r + 1);
}
```

5.13 Minimal Enclosing Circle

```
struct Cir {
  Pt o;
  double r;
  bool inside(Pt p) {
    return sgn(r - abs(p - o)) >= 0;
  }
};

Pt Center(Pt a, Pt b, Pt c) {
  Pt x = (a + b) / 2;
  Pt y = (b + c) / 2;
  return LineInter({x, x + rotate(b - a)}, {y, y + rotate(c - b
    )});
}
Cir MEC(vector<Pt> P) {
  mt19937 rng(time(0));
  shuffle(all(P), rng);
  Cir C{};
  for (int i = 0; i < P.size(); i++) {
    if (C.inside(P[i])) continue;
    C = {P[i], 0};
    for (int j = 0; j < i; j++) {
      if (C.inside(P[j])) continue;
      C = {(P[i] + P[j]) / 2, abs(P[i] - P[j]) / 2};
      for (int k = 0; k < j; k++) {
        if (C.inside(P[k])) continue;
        C.o = Center(P[i], P[j], P[k]);
        C.r = abs(C.o - P[i]);
      }
    }
  }
  return C;
}
```

5.14 Minkowski

```
// P, Q, R(return) are counterclockwise order convex polygon
vector<Pt> Minkowski(vector<Pt> P, vector<Pt> Q) {
  assert(P.size() >= 2 && Q.size() >= 2);
  auto cmp = [&](Pt a, Pt b) {
```

```
    return Pt{a.y, a.x} < Pt{b.y, b.x};
  };
  auto reorder = [&](auto &R) {
    rotate(R.begin(), min_element(all(R), cmp), R.end());
    R.push_back(R[0]), R.push_back(R[1]);
  };
  const int n = P.size(), m = Q.size();
  reorder(P), reorder(Q);
  vector<Pt> R;
  for (int i = 0, j = 0, s; i < n || j < m; ) {
    R.push_back(P[i] + Q[j]);
    s = sgn((P[i + 1] - P[i]) ^ (Q[j + 1] - Q[j]));
    if (s >= 0) i++;
    if (s <= 0) j++;
  }
  return R; // May not be a strict convexhull
}
```

5.15 Point In Circumcircle

```
// p[0], p[1], p[2] should be counterclockwise order
int inCC(const array<Pt, 3> &p, Pt a) {
  i128 det = 0;
  for (int i = 0; i < 3; i++)
    det += i128(abs2(p[i]) - abs2(a)) * ori(a, p[(i + 1) % 3],
     p[(i + 2) % 3]);
  return (det > 0) - (det < 0); // in:1, on:0, out:-1
}
```

5.16 Tangent Lines of Circle and Point

```
vector<Line> CircleTangent(Cir c, Pt p) {
  vector<Line> z;
  double d = abs(p - c.o);
  if (sgn(d - c.r) == 0) {
    Pt i = rotate(p - c.o);
    z.push_back({p, p + i});
  } else if (d > c.r) {
    double o = acos(c.r / d);
    Pt i = unit(p - c.o);
    Pt j = rotate(i, o) * c.r;
    Pt k = rotate(i, -o) * c.r;
    z.push_back({c.o + j, p});
    z.push_back({c.o + k, p});
  }
  return z;
}
```

5.17 Tangent Lines of Circles

```
vector<Line> CircleTangent(Cir c1, Cir c2, int sign1) {
  // sign1 = 1 for outer tang, -1 for inter tang
  vector<Line> ret;
  double d_sq = abs2(c1.o - c2.o);
  if (sgn(d_sq) == 0) return ret;
  double d = sqrt(d_sq);
  Pt v = (c2.o - c1.o) / d;
  double c = (c1.r - sign1 * c2.r) / d;
  if (c * c > 1) return ret;
  double h = sqrt(max(0.0, 1.0 - c * c));
  for (int sign2 = 1; sign2 >= -1; sign2 -= 2) {
    Pt n = Pt(v.x * c - sign2 * h * v.y, v.y * c + sign2 * h *
     v.x);
    Pt p1 = c1.o + n * c1.r;
    Pt p2 = c2.o + n * (c2.r * sign1);
    if (sgn(p1.x - p2.x) == 0 && sgn(p1.y - p2.y) == 0)
      p2 = p1 + rotate(c2.o - c1.o);
    ret.push_back({p1, p2});
  }
 return ret;
}
```

5.18 Triangle Center

```
Pt TriangleCircumCenter(Pt a, Pt b, Pt c) {
 Pt res;
 double a1 = atan2(b.y - a.y, b.x - a.x) + pi / 2;
 double a2 = atan2(c.y - b.y, c.x - b.x) + pi / 2;
 double ax = (a.x + b.x) / 2;
 double ay = (a.y + b.y) / 2;
 double bx = (c.x + b.x) / 2;
 double by = (c.y + b.y) / 2;
 double r1 = (sin(a2) * (ax - bx) + cos(a2) * (by - ay)) / (sin
    (a1) * cos(a2) - sin(a2) * cos(a1));
 return Pt(ax + r1 * cos(a1), ay + r1 * sin(a1));
}
Pt TriangleMassCenter(Pt a, Pt b, Pt c) {
 return (a + b + c) / 3.0;
}
Pt TriangleOrthoCenter(Pt a, Pt b, Pt c) {
 return TriangleMassCenter(a, b, c) * 3.0 -
     TriangleCircumCenter(a, b, c) * 2.0;
}
Pt TriangleInnerCenter(Pt a, Pt b, Pt c) {
 Pt res;
 double la = abs(b - c);
 double lb = abs(a - c);
 double lc = abs(a - b);
 res.x = (la * a.x + lb * b.x + lc * c.x) / (la + lb + lc);
 res.y = (la * a.y + lb * b.y + lc * c.y) / (la + lb + lc);
 return res;
}
```

5.19 Union of Circles

```
// Area[i] : area covered by at least i circle
vector<double> CircleUnion(const vector<Cir> &C) {
  const int n = C.size();
  vector<double> Area(n + 1);
  auto check = [&](int i, int j) {
    if (!contain(C[i], C[j]))
      return false;
    return sgn(C[i].r - C[j].r) > 0 or (sgn(C[i].r - C[j].r) ==
     0 and i < j);
  };
  struct Teve {
    double ang; int add; Pt p;
    bool operator<(const Teve &b) { return ang < b.ang; }
  };
  auto ang = [&](Pt p) { return atan2(p.y, p.x); };
  for (int i = 0; i < n; i++) {
    int cov = 1;
    vector<Teve> event;
    for (int j = 0; j < n; j++) if (i != j) {
      if (check(j, i)) cov++;
      else if (!check(i, j) and !disjunct(C[i], C[j])) {
        auto I = CircleInter(C[i], C[j]);
        assert(I.size() == 2);
        double a1 = ang(I[0] - C[i].o), a2 = ang(I[1] - C[i].o)
     ;
        event.push_back({a1, 1, I[0]});
        event.push_back({a2, -1, I[1]});
        if (a1 > a2) cov++;
      }
    }
    if (event.empty()) {
      Area[cov] += pi * C[i].r * C[i].r;
      continue;
    }
    sort(all(event));
    event.push_back(event[0]);
    for (int j = 0; j + 1 < event.size(); j++) {
      cov += event[j].add;
      Area[cov] += (event[j].p ^ event[j + 1].p) / 2.;
      double theta = event[j + 1].ang - event[j].ang;
      if (theta < 0) theta += 2 * pi;
      Area[cov] += (theta - sin(theta)) * C[i].r * C[i].r / 2.;
    }
  }
  return Area;
}
```

6 Graph

6.1 Block Cut Tree

```
struct BlockCutTree {
  int n;
  vector<vector<int>> adj;
  BlockCutTree(int _n) : n(_n), adj(_n) {}
  void addEdge(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  pair<int, vector<pair<int, int>>> work() {
    vector<int> dfn(n, -1), low(n), stk;
    vector<pair<int, int>> edg;
    int cnt = 0, cur = 0;
    function<void(int)> dfs = [&](int x) {
      stk.push_back(x);
      dfn[x] = low[x] = cur++;
```

```
      for (auto y : adj[x]) {
        if (dfn[y] == -1) {
          dfs(y);
          low[x] = min(low[x], low[y]);
          if (low[y] == dfn[x]) {
            int v;
            do {
              v = stk.back();
              stk.pop_back();
              edg.emplace_back(n + cnt, v);
            } while (v != y);
            edg.emplace_back(x, n + cnt);
            cnt++;
          }
        } else {
          low[x] = min(low[x], dfn[y]);
        }
      }
    };
    for (int i = 0; i < n; i++) {
      if (dfn[i] == -1) {
        stk.clear();
        dfs(i);
      }
    }
    return {cnt, edg};
  }
};
```

6.2 Count Cycles

```
// ord = sort by deg decreasing, rk[ord[i]] = i
// D: undirected to directed edge from rk small to rk big
vector<int> vis(n, 0);
int c3 = 0, c4 = 0;
for (int x : ord) { // c3
  for (int y : D[x]) vis[y] = 1;
  for (int y : D[x]) for (int z : D[y]) c3 += vis[z];
  for (int y : D[x]) vis[y] = 0;
}
for (int x : ord) { // c4
  for (int y : D[x]) for (int z : adj[y])
    if (rk[z] > rk[x]) c4 += vis[z]++;
  for (int y : D[x]) for (int z : adj[y])
    if (rk[z] > rk[x]) --vis[z];
}
```

6.3 Dominator Tree

```
vector<int> BuildDomTree(vector<vector<int>> adj, int rt) {
  int n = adj.size();

  // buckets: list of vertices y with sdom(y) = x
  vector<vector<int>> buckets(n), radj(n);

  // rev[dfn[x]] = x
  vector<int> dfn(n, -1), rev(n, -1), pa(n, -1);
  vector<int> sdom(n, -1), dom(n, -1);
  vector<int> fa(n, -1), val(n, -1);

  int stamp = 0;

  // re-number in DFS order
  auto dfs = [&](auto self, int u) -> void {
    rev[dfn[u] = stamp] = u;
    fa[stamp] = sdom[stamp] = val[stamp] = stamp;
    stamp++;
    for (int v : adj[u]) {
      if (dfn[v] == -1) {
        self(self, v);
        pa[dfn[v]] = dfn[u];
      }
      radj[dfn[v]].pb(dfn[u]);
    }
  };

  function<int(int, bool)> Eval = [&](int x, bool fir) {
    if (x == fa[x]) return fir ? x : -1;
    int p = Eval(fa[x], false);
    // x is one step away from the root
    if (p == -1) return x;
    if (sdom[val[x]] > sdom[val[fa[x]]]) val[x] = val[fa[x]];
    fa[x] = p;
    return fir ? val[x] : p;
  };

  auto Link = [&](int x, int y) -> void { fa[x] = y; };

  dfs(dfs, rt);

  // compute sdom in reversed DFS order
  for (int x = stamp - 1; x >= 0; --x) {
    for (int y : radj[x]) {
      // sdom[x] = min({y | (y, x) in E(G), y < x}, {sdom[z] |
      (y, x) in E(G), z > x && z is y's ancestor})
      chmin(sdom[x], sdom[Eval(y, true)]);
    }
    if (x > 0) buckets[sdom[x]].pb(x);
    for (int u : buckets[x]) {
      int p = Eval(u, true);
      if (sdom[p] == x) dom[u] = x;
      else dom[u] = p;
    }
    if (x > 0) Link(x, pa[x]);
  }
  // idom[x] = -1 if x is unreachable from rt
  vector<int> idom(n, -1);
  idom[rt] = rt;
  rep (x, 1, stamp) {
    if (sdom[x] != dom[x]) dom[x] = dom[dom[x]];
  }
  rep (i, 1, stamp) idom[rev[i]] = rev[dom[i]];
  return idom;
}
```

6.4 Enumerate Planar Face

```
// 0-based
struct PlanarGraph{
  int n, m, id;
  vector<Pt<int>> v;
  vector<vector<pair<int, int>>> adj;
  vector<int> conv, nxt, vis;

  PlanarGraph(int n, int m, vector<Pt<int>> _v) :
  n(n), m(m), id(0),
  v(_v), adj(n),
  conv(m << 1), nxt(m << 1), vis(m << 1) {}

  void add_edge(int x, int y) {
    adj[x].push_back({y, id << 1});
    adj[y].push_back({x, id << 1 | 1});
    conv[id << 1] = x;
    conv[id << 1 | 1] = y;
    id++;
  }

  vector<int> enumerate_face() {
    for (int i = 0; i < n; i++) {
      sort(all(adj[i]), [&](const auto &a, const auto & b) {
        return (v[a.first] - v[i]) < (v[b.first] - v[i]);
      });
      int sz = adj[i].size(), pre = sz - 1;
      for (int j = 0; j < sz; j++) {
        nxt[adj[i][pre].second] = adj[i][j].second ^ 1;
        pre = j;
      }
    }

    vector<int> ret;
    for (int i = 0; i < m * 2; i++) {
      if (!vis[i]) {
        int area = 0, now = i;
        vector<int> pt;
        while (!vis[now]) {
          vis[now] = true;
          pt.push_back(conv[now]);
          now = nxt[now];
        }
        pt.push_back(pt.front());
        for (int i = 0; i + 1 < ssize(pt); i++) {
          area -= (v[pt[i]] ^ v[pt[i + 1]]);
        }
        // pt = face boundary
        if (area > 0) {
          ret.push_back(area);
        } else {
          // pt is outer face
        }
      }
```

```
    }
    return ret;
  }
};
```

6.5 Manhattan MST

```
// {w, u, v}
vector<tuple<int, int, int>> ManhattanMST(vector<Pt> P) {
  vector<int> id(P.size());
  iota(all(id), 0);
  vector<tuple<int, int, int>> edg;
  for (int k = 0; k < 4; k++) {
    sort(all(id), [&](int i, int j) {
        return (P[i] - P[j]).ff < (P[j] - P[i]).ss;
      });
    map<int, int> sweep;
    for (int i : id) {
      auto it = sweep.lower_bound(-P[i].ss);
      while (it != sweep.end()) {
        int j = it->ss;
        Pt d = P[i] - P[j];
        if (d.ss > d.ff) {
          break;
        }
        edg.emplace_back(d.ff + d.ss, i, j);
        it = sweep.erase(it);
      }
      sweep[-P[i].ss] = i;
    }
    for (Pt &p : P) {
      if (k % 2) {
        p.ff = -p.ff;
      } else {
        swap(p.ff, p.ss);
      }
    }
  }
  return edg;
}
```

6.6 Matroid Intersection

```
/*
M1 = xx matroid, M2 = xx matroid
y<-s if I+y  satisfies M1
y->t if I+y  satisfies M2
x<-y if I-x+y satisfies M2
x->y if I-x+y satisfies M1
交換圖點權：
-w[e] if e \in I
w[e] otherwise
*/
vector<int> I(, 0);
while (true) {
  vector<vector<int>> adj();
  int s = , t = s + 1;
  auto M1 = [&]() -> void { // xx matroid
    {  // y<-s

    }
    {  // x->y

    }
  };
  auto M2 = [&]() -> void { // xx matroid
    {  // y->t

    }
    {  // x<-y

    }
  };
  auto augment = [&]() -> bool { // 註解掉的是帶權版
    vector<int> vis( + 2, 0), dis( + 2, IINF), from( + 2, -1);
    queue<int> q;
    vis[s] = 1;
    dis[s] = 0;
    q.push(s);
    while (!q.empty()) {
      int u = q.front(); q.pop();
      // vis[u] = 0;
      for (int v : adj[u]) {
        int w = ; // no weight -> 1
        if (chmin(dis[v], dis[u] + w)) {
          from[v] = u;
          // if (!vis[v]) {
            // vis[v] = 1;
            q.push(v);
          // }
        }
      }
    }
    if (from[t] == -1) return false;
    for (int cur = from[t];; cur = from[cur]) {
      if (cur == -1 || cur == s) break;
      I[cur] ^= 1;
    }
    return true;
  };
  M1(), M2();
  if (!augment()) break;
}
```

6.7 Maximum Clique

```
constexpr size_t kN = 150;
using bits = bitset<kN>;
struct MaxClique {
  bits G[kN], cs[kN];
  int ans, sol[kN], q, cur[kN], d[kN], n;
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i) G[i].reset();
  }
  void addEdge(int u, int v) {
    G[u][v] = G[v][u] = 1;
  }
  void preDfs(vector<int> &v, int i, bits mask) {
    if (i < 4) {
      for (int x : v) d[x] = (G[x] & mask).count();
      sort(all(v), [&](int x, int y) {
        return d[x] > d[y];
      });
    }
    vector<int> c(v.size());
    cs[1].reset(), cs[2].reset();
    int l = max(ans - q + 1, 1), r = 2, tp = 0, k;
    for (int p : v) {
      for (k = 1;
        (cs[k] & G[p]).any(); ++k);
      if (k >= r) cs[++r].reset();
      cs[k][p] = 1;
      if (k < l) v[tp++] = p;
    }
    for (k = l; k < r; ++k)
      for (auto p = cs[k]._Find_first(); p < kN; p = cs[k].
     _Find_next(p))
        v[tp] = p, c[tp] = k, ++tp;
    dfs(v, c, i + 1, mask);
  }
  void dfs(vector<int> &v, vector<int> &c, int i, bits mask) {
    while (!v.empty()) {
      int p = v.back();
      v.pop_back();
      mask[p] = 0;
      if (q + c.back() <= ans) return;
      cur[q++] = p;
      vector<int> nr;
      for (int x : v)
        if (G[p][x]) nr.push_back(x);
      if (!nr.empty()) preDfs(nr, i, mask & G[p]);
      else if (q > ans) ans = q, copy_n(cur, q, sol);
      c.pop_back();
      --q;
    }
  }
  int solve() {
    vector<int> v(n);
    iota(all(v), 0);
    ans = q = 0;
    preDfs(v, 0, bits(string(n, '1')));
    return ans;
  }
} cliq;
```

6.8 Tree Hash

```
map<vector<int>, int> id;
vector<vector<int>> sub;
vector<int> siz;
```

```cpp
int getid(const vector<int> &T) {
  if (id.count(T)) return id[T];
  int s = 1;
  for (int x : T) {
    s += siz[x];
  }
  sub.push_back(T);
  siz.push_back(s);
  return id[T] = id.size();
}
int dfs(int u, int f) {
  vector<int> S;
  for (int v : G[u]) if (v != f) {
    S.push_back(dfs(v, u));
  }
  sort(all(S));
  return getid(S);
}
```

6.9 Two-SAT

```cpp
struct TwoSat {
  int n;
  vector<vector<int>> G;
  vector<bool> ans;
  vector<int> id, dfn, low, stk;
  TwoSat(int n) : n(n), G(2 * n) {}
  void addClause(int u, bool f, int v, bool g) { // (u = f) or
     (v = g)
    G[2 * u + !f].push_back(2 * v + g);
    G[2 * v + !g].push_back(2 * u + f);
  }
  void addImply(int u, bool f, int v, bool g) { // (u = f) -> (
     v = g)
    G[2 * u + f].push_back(2 * v + g);
    G[2 * v + !g].push_back(2 * u + !f);
  }
  int addVar() {
    G.emplace_back();
    G.emplace_back();
    return n++;
  }
  void addAtMostOne(const vector<pair<int, bool>> &li) {
    if (ssize(li) <= 1) return;
    int pu; bool pf; tie(pu, pf) = li[0];
    for (int i = 2; i < ssize(li); i++) {
      const auto &[u, f] = li[i];
      int nxt = addVar();
      addClause(pu, !pf, u, !f);
      addClause(pu, !pf, nxt, true);
      addClause(u, !f, nxt, true);
      tie(pu, pf) = make_pair(nxt, true);
    }
    addClause(pu, !pf, li[1].first, !li[1].second);
  }
  int cur = 0, scc = 0;
  void dfs(int u) {
    stk.push_back(u);
    dfn[u] = low[u] = cur++;
    for (int v : G[u]) {
      if (dfn[v] == -1) {
        dfs(v);
        chmin(low[u], low[v]);
      } else if (id[v] == -1) {
        chmin(low[u], dfn[v]);
      }
    }
    if (dfn[u] == low[u]) {
      int x;
      do {
        x = stk.back();
        stk.pop_back();
        id[x] = scc;
      } while (x != u);
      scc++;
    }
  }
  bool satisfiable() {
    ans.assign(n, 0);
    id.assign(2 * n, -1);
    dfn.assign(2 * n, -1);
    low.assign(2 * n, -1);
    for (int i = 0; i < n * 2; i++)
      if (dfn[i] == -1) {
        dfs(i);
      }
    for (int i = 0; i < n; ++i) {
      if (id[2 * i] == id[2 * i + 1]) {
        return false;
      }
      ans[i] = id[2 * i] > id[2 * i + 1];
    }
    return true;
  }
};
```

6.10 Virtual Tree

```cpp
// need LCA
vector<vector<int>> vir(n);
auto clear = [&](auto self, int u) -> void {
  for (int v : vir[u]) self(self, v);
  vir[u].clear();
};
auto build = [&](vector<int> &v) -> void { // be careful of the
     changes to the array
  // maybe dont need to sort when do it while dfs
  sort(all(v), [&](int a, int b) {
    return dfn[a] < dfn[b];
  });
  clear(clear, 0);
  if (v[0] != 0) v.insert(v.begin(), 0);
  int k = v.size();
  vector<int> st;
  rep (i, 0, k) {
    if (st.empty()) {
      st.push_back(v[i]);
      continue;
    }
    int p = lca(v[i], st.back());
    if (p == st.back()) {
      st.push_back(v[i]);
      continue;
    }
    while (st.size() >= 2 && dep[st.end()[-2]] >= dep[p]) {
      vir[st.end()[-2]].push_back(st.back());
      st.pop_back();
    }
    if (st.back() != p) {
      vir[p].push_back(st.back());
      st.pop_back();
      st.push_back(p);
    }
    st.push_back(v[i]);
  }
  while (st.size() >= 2) {
    vir[st.end()[-2]].push_back(st.back());
    st.pop_back();
  }
};
```

7 Math

7.1 Combinatoric

```cpp
vector<mint> fac, inv;

inline void init (int n) {
  fac.resize(n + 1);
  inv.resize(n + 1);
  fac[0] = inv[0] = 1;
  rep (i, 1, n + 1) fac[i] = fac[i - 1] * i;
  inv[n] = fac[n].inv();
  for (int i = n; i > 0; --i) inv[i - 1] = inv[i] * i;
}

inline mint Comb(int n, int k) {
  if (k > n || k < 0) return 0;
  return fac[n] * inv[k] * inv[n - k];
}

inline mint H(int n, int m) {
  return Comb(n + m - 1, m);
}

inline mint catalan(int n){
  return fac[2 * n] * inv[n + 1] * inv[n];
}
```

7.2 Discrete Log

```
int power(int a, int b, int p, int res = 1) {
  for (; b; b /= 2, a = 1LL * a * a % p) {
    if (b & 1) {
      res = 1LL * res * a % p;
    }
  }
  return res;
}

int exbsgs(int a, int b, int p) {
  a %= p;
  b %= p;
  if (b == 1 || p == 1) {
    return 0;
  }
  if (a == 0) {
    return b == 0 ? 1 : -1;
  }

  i64 g, k = 0, t = 1; // t : a ^ k / sum{d}
  while ((g = std::gcd(a, p)) > 1) {
    if (b % g) {
      return -1;
    }
    b /= g;
    p /= g;
    k++;
    t = t * (a / g) % p;
    if (t == b) {
      return k;
    }
  }

  const int n = std::sqrt(p) + 1;
  std::unordered_map<int, int> mp;
  mp[b] = 0;

  int x = b, y = t;
  int mi = power(a, n, p);
  for (int i = 1; i < n; i++) {
    x = 1LL * x * a % p;
    mp[x] = i;
  }

  for (int i = 1; i <= n; i++) {
    t = 1LL * t * mi % p;
    if (mp.contains(t)) {
      return 1LL * i * n - mp[t] + k;
    }
  }

  return -1; // no solution
}
```

7.3 Div Floor Ceil

```
int CEIL(int a, int b) {
 return (a >= 0 ? (a + b - 1) / b : a / b);
}

int FLOOR(int a, int b) {
 return (a >= 0 ? a / b : (a - b + 1) / b);
}
```

7.4 exCRT

```
i64 exgcd(i64 a, i64 b, i64 &x, i64 &y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
  }
  i64 g = exgcd(b, a % b, y, x);
  y -= a / b * x;
  return g;
}

// return {x, T}
// a: moduli, b: remainders
// x: first non-negative solution, T: minimum period
std::pair<i64, i64> exCRT(auto &a, auto &b) {
  auto [m1, r1] = std::tie(a[0], b[0]);
  for (int i = 1; i < std::ssize(a); i++) {
    auto [m2, r2] = std::tie(a[i], b[i]);
    i64 x, y;
    i64 g = exgcd(m1, m2, x, y);
    if ((r2 - r1) % g) { // no solution
      return {-1, -1};
    }
    x = (i128(x) * (r2 - r1) / g) % (m2 / g);
    if (x < 0) {
      x += (m2 / g);
    }
    r1 = m1 * x + r1;
    m1 = std::lcm(m1, m2);
  }
  r1 %= m1;
  if (r1 < 0) {
    r1 += m1;
  }
  return {r1, m1};
};
```

7.5 Factorization

```
ull modmul(ull a, ull b, ull M) {
  i64 ret = a * b - M * ull(1.L / M * a * b);
  return ret + M * (ret < 0) - M * (ret >= (i64)M);
}

ull modpow(ull b, ull e, ull mod) {
  ull ans = 1;
  for (; e; b = modmul(b, b, mod), e /= 2)
    if (e & 1) ans = modmul(ans, b, mod);
  return ans;
}

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

ull pollard(ull n) {
  uniform_int_distribution<ull> unif(0, n - 1);
  ull c = 1;
  auto f = [n, &c](ull x) { return modmul(x, x, n) + c % n; };
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  while (t++ % 40 || __gcd(prd, n) == 1) {
    if (x == y) c = unif(rng), x = ++i, y = f(x);
    if ((q = modmul(prd, max(x, y) - min(x, y), n))) prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}

vector<ull> factor(ull n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), r.begin(), r.end());
  return l;
}
```

7.6 Floor Sum

```
// \sum_0^n floor((a * x + b) / c)) in log(n + m + a + b)
int floor_sum(int a, int b, int c, int n) { // add mod if
    needed
  int m = (a * n + b) / c;
  if (a >= c || b >= c)
    return (a / c) * (n * (n + 1) / 2) + (b / c) * (n + 1) +
     floor_sum(a % c, b % c, c, n);
  if (n < 0 || a == 0)
    return 0;
  return n * m - floor_sum(c, c - b - 1, a, m - 1);
}
```

7.7 FWT

```
void fwt(vector<ll> &f, bool inv = false) { // xor-convolution
  const int N = 31 - __builtin_clz(ssize(f)),
       inv2 = (MOD + 1) / 2;
  rep (i, 0, N) rep (j, 0, 1 << N) {
    if (j >> i & 1 ^ 1) {
```

```
      ll a = f[j], b = f[j | (1 << i)];
      if (inv) {
        f[j] = (a + b) * inv2 % MOD;
        f[j | (1 << i)] = (a - b + MOD) * inv2 % MOD;
      } else {
        f[j] = (a + b) % MOD;
        f[j | (1 << i)] = (a - b + MOD) % MOD;
      }
    }
  }
}
```

7.8 Gauss Elimination

```
using Z = ModInt<998244353>;
// using F = long double;

using Matrix = std::vector<std::vector<Z>>;
// using Matrix = std::vector<std::vector<F>>; (double)
// using Matrix = std::vector<std::bitset<5000>>; (mod 2)

template <typename T>
auto gauss(Matrix &A, std::vector<T> &b, int n, int m) {
  assert(std::ssize(b) == n);
  int r = 0;
  std::vector<int> where(m, -1);

  for (int i = 0; i < m && r < n; i++) {
    int p = r; // pivot
    while (p < n && A[p][i] == T(0)) {
      p++;
    }
    if (p == n) {
      continue;
    }

    std::swap(A[r], A[p]);
    std::swap(b[r], b[p]);
    where[i] = r;

    // coef: mod 2 don't need this
    T inv = T(1) / A[r][i];
    for (int j = i; j < m; j++) {
      A[r][j] *= inv;
    }
    b[r] *= inv;

    for (int j = 0; j < n; j++) { // deduct: mod 2 don't need
     this
      if (j != r) {
        T x = A[j][i];
        for (int k = i; k < m; k++) {
          A[j][k] -= x * A[r][k];
        }
        b[j] -= x * b[r];
      }
    }

    // for (int j = 0; j < n; ++j) { // (mod 2) -> coef and
     deduct
    //   if (j != r && A[j][i]) {
    //     A[j] ^= A[r];
    //     b[j] ^= b[r];
    //   }
    // }

    r++;
  }

  for (int i = r; i < n; i++) {
    if (ranges::all_of(A[i] | views::take(m), [](auto x) {
     return x == 0; }) && b[i] != T(0)) {
      return std::vector<T>(); // no solution
    }
    // if (A[i].none() && b[i]) { // (mod 2)
    //   return std::vector<T>();
    // }
  }

  // if (r < m) { // infinite solution
  //   return std::vector<T>();
  // }

  std::vector<T> res(m);
  for (int i = 0; i < m; i++) {
    if (where[i] != -1) {
      res[i] = b[where[i]];
    }
  }
  return res;
};
```

7.9 Lagrange Interpolation

```
struct Lagrange {
  int deg{};
  vector<int> C;
  Lagrange(const vector<int> &P) {
    deg = P.size() - 1;
    C.assign(deg + 1, 0);
    for (int i = 0; i <= deg; i++) {
      int q = inv[i] * inv[i - deg] % mod;
      if ((deg - i) % 2 == 1) {
        q = mod - q;
      }
      C[i] = P[i] * q % mod;
    }
  }
  int operator()(int x) { // 0 <= x < mod
    if (0 <= x and x <= deg) {
      int ans = fac[x] * fac[deg - x] % mod;
      if ((deg - x) % 2 == 1) {
        ans = (mod - ans);
      }
      return ans * C[x] % mod;
    }
    vector<int> pre(deg + 1), suf(deg + 1);
    for (int i = 0; i <= deg; i++) {
      pre[i] = (x - i);
      if (i) {
        pre[i] = pre[i] * pre[i - 1] % mod;
      }
    }
    for (int i = deg; i >= 0; i--) {
      suf[i] = (x - i);
      if (i < deg) {
        suf[i] = suf[i] * suf[i + 1] % mod;
      }
    }
    int ans = 0;
    for (int i = 0; i <= deg; i++) {
      ans += (i == 0 ? 1 : pre[i - 1]) * (i == deg ? 1 : suf[i
     + 1]) % mod * C[i];
      ans %= mod;
    }
    if (ans < 0) ans += mod;
    return ans;
  }
};
```

7.10 Linear Sieve

```
const int C = 1e6 + 5;

int mo[C], lp[C], phi[C], isp[C];
vector<int> prime;

void sieve() {
  mo[1] = phi[1] = 1;
  rep (i, 1, C) lp[i] = 1;
  rep (i, 2, C) {
    if (lp[i] == 1) {
      lp[i] = i;
      prime.pb(i);
      isp[i] = 1;
      mo[i] = -1;
      phi[i] = i - 1;
    }
    for (int p : prime) {
      if (i * p >= C) break;
      lp[i * p] = p;
      if (i % p == 0) {
        phi[p * i] = phi[i] * p;
        break;
      }
      phi[i * p] = phi[i] * (p - 1);
      mo[i * p] = mo[i] * mo[p];
    }
  }
}
```

7.11 Lucas

```
// comb(n, m) % M, M = p^k
// O(M)-O(log(n))
struct Lucas {
  const int p, M;
  vector<int> f;
  Lucas(int p, int M) : p(p), M(M), f(M + 1) {
    f[0] = 1;
    for (int i = 1; i <= M; i++) {
      f[i] = f[i - 1] * (i % p == 0 ? 1 : i) % M;
    }
  }
  int CountFact(int n) {
    int c = 0;
    while (n) c += (n /= p);
    return c;
  }
  // (n! without factor p) % p^k
  int ModFact(int n) {
    int r = 1;
    while (n) {
      r = r * power(f[M], n / M % 2, M) % M * f[n % M] % M;
      n /= p;
    }
    return r;
  }
  int ModComb(int n, int m) {
    if (m < 0 or n < m) return 0;
    int c = CountFact(n) - CountFact(m) - CountFact(n - m);
    int r = ModFact(n) * power(ModFact(m), M / p * (p - 1) - 1,
     M) % M
            * power(ModFact(n - m), M / p * (p - 1) - 1, M) %
     M;
    return r * power(p, c, M) % M;
  }
};
```

7.12 Mod Int

```
using u32 = unsigned int;
using u64 = unsigned long long;
template <class T>
constexpr T power(T a, u64 b, T res = 1) {
  for (; b != 0; b /= 2, a *= a) {
    if (b & 1) {
      res *= a;
    }
  }
  return res;
}

template <u32 P>
struct ModInt {
  u32 v;
  const static ModInt G;

  constexpr ModInt &norm(u32 x) {
    v = x < P ? x : x - P;
    return *this;
  }
  constexpr ModInt(i64 x = 0) { norm(x % P + P); }
  constexpr ModInt inv() const { return power(*this, P - 2); }
  constexpr ModInt operator-() const { return ModInt() - *this;
    }
  constexpr ModInt operator+(const ModInt &r) const { return
     ModInt().norm(v + r.v); }
  constexpr ModInt operator-(const ModInt &r) const { return
     ModInt().norm(v + P - r.v); }
  constexpr ModInt operator*(const ModInt &r) const { return
     ModInt().norm(u64(v) * r.v % P); }
  constexpr ModInt operator/(const ModInt &r) const { return *
     this * r.inv(); }
  constexpr ModInt &operator+=(const ModInt &r) { return *this
     = *this + r; }
  constexpr ModInt &operator-=(const ModInt &r) { return *this
     = *this - r; }
  constexpr ModInt &operator*=(const ModInt &r) { return *this
     = *this * r; }
  constexpr ModInt &operator/=(const ModInt &r) { return *this
     = *this / r; }
  constexpr bool operator==(const ModInt &r) const { return v
     == r.v; }
  constexpr bool operator!=(const ModInt &r) const { return v
     != r.v; }
  explicit constexpr operator bool() const { return v != 0; }
  friend std::ostream &operator<<(std::ostream &os, const
     ModInt &r) {
    return os << r.v;
  }
};
using mint = ModInt<998244353>;
template <> const mint mint::G = mint(3);
```

7.13 Primitive Root

```
ull primitiveRoot(ull p) {
  auto fac = factor(p - 1);
  sort(all(fac));
  fac.erase(unique(all(fac)), fac.end());
  auto test = [p, fac](ull x) {
    for(ull d : fac)
    if (modpow(x, (p - 1) / d, p) == 1)
      return false;
    return true;
  };
  uniform_int_distribution<ull> unif(1, p - 1);
  ull root;
  while(!test(root = unif(rng)));
  return root;
}
```

7.14 Simplex

```
// max{cx} subject to {Ax<=b, x>=0}
// n: constraints, m: vars !!!
// x[] is the optimal solution vector
// usage :
// x = simplex(A, b, c); (A <= 100 x 100)
vector<double> simplex(
    const vector<vector<double>> &a,
    const vector<double> &b,
    const vector<double> &c) {

  int n = (int)a.size(), m = (int)a[0].size() + 1;
  vector val(n + 2, vector<double>(m + 1));
  vector<int> idx(n + m);
  iota(all(idx), 0);
  int r = n, s = m - 1;
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m - 1; ++j)
      val[i][j] = -a[i][j];
    val[i][m - 1] = 1;
    val[i][m] = b[i];
    if (val[r][m] > val[i][m])
      r = i;
  }
  copy(all(c), val[n].begin());
  val[n + 1][m - 1] = -1;
  for (double num; ; ) {
    if (r < n) {
      swap(idx[s], idx[r + m]);
      val[r][s] = 1 / val[r][s];
      for (int j = 0; j <= m; ++j) if (j != s)
        val[r][j] *= -val[r][s];
      for (int i = 0; i <= n + 1; ++i) if (i != r) {
        for (int j = 0; j <= m; ++j) if (j != s)
          val[i][j] += val[r][j] * val[i][s];
        val[i][s] *= val[r][s];
      }
    }
    r = s = -1;
    for (int j = 0; j < m; ++j)
      if (s < 0 || idx[s] > idx[j])
        if (val[n + 1][j] > eps || val[n + 1][j] > -eps && val[
    n][j] > eps)
          s = j;
    if (s < 0) break;
    for (int i = 0; i < n; ++i) if (val[i][s] < -eps) {
      if (r < 0
        || (num = val[r][m] / val[r][s] - val[i][m] / val[i][s
    ]) < -eps
        || num < eps && idx[r + m] > idx[i + m])
        r = i;
    }
    if (r < 0) {
      //  Solution is unbounded.
      return vector<double>{};
    }
  }
  if (val[n + 1][m] < -eps) {
    //  No solution.
```

```
    return vector<double>{};
  }
  vector<double> x(m - 1);
  for (int i = m; i < n + m; ++i)
    if (idx[i] < m - 1)
      x[idx[i]] = val[i - m][m];
  return x;
}
```

7.15 Sqrt Mod

```
// the Jacobi symbol is a generalization of the Legendre symbol
    ,
// such that the bottom doesn't need to be prime.
// (n|p) -> same as legendre
// (n|ab) = (n|a)(n|b)
// work with long long
int Jacobi(int a, int m) {
  int s = 1;
  for (; m > 1; ) {
    a %= m;
    if (a == 0) return 0;
    const int r = __builtin_ctz(a);
    if ((r & 1) && ((m + 2) & 4)) s = -s;
    a >>= r;
    if (a & m & 2) s = -s;
    swap(a, m);
  }
  return s;
}

// 0: a == 0
// -1: a isn't a quad res of p
// else: return X with X^2 % p == a
// doesn't work with long long
int QuadraticResidue(int a, int p) {
  if (p == 2) return a & 1;
  if (int jc = Jacobi(a, p); jc <= 0) return jc;
  int b, d;
  for (; ; ) {
    b = rand() % p;
    d = (1LL * b * b + p - a) % p;
    if (Jacobi(d, p) == -1) break;
  }
  int f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
  for (int e = (1LL + p) >> 1; e; e >>= 1) {
    if (e & 1) {
      tmp = (1LL * g0 * f0 + 1LL * d * (1LL * g1 * f1 % p)) % p
          ;
      g1 = (1LL * g0 * f1 + 1LL * g1 * f0) % p;
      g0 = tmp;
    }
    tmp = (1LL * f0 * f0 + 1LL * d * (1LL * f1 * f1 % p)) % p;
    f1 = (2LL * f0 * f1) % p;
    f0 = tmp;
  }
  return g0;
}
```

7.16 PiCount

```
i64 PrimeCount(i64 n) { // n ~ 10^13 => < 2s
  if (n <= 1) return 0;
  int v = sqrt(n), s = (v + 1) / 2, pc = 0;
  vector<int> smalls(v + 1), skip(v + 1), roughs(s);
  vector<i64> larges(s);
  for (int i = 2; i <= v; ++i) smalls[i] = (i + 1) / 2;
  for (int i = 0; i < s; ++i) {
    roughs[i] = 2 * i + 1;
    larges[i] = (n / (2 * i + 1) + 1) / 2;
  }
  for (int p = 3; p <= v; ++p) {
    if (smalls[p] > smalls[p - 1]) {
      int q = p * p;
      ++pc;
      if (1LL * q * q > n) break;
      skip[p] = 1;
      for (int i = q; i <= v; i += 2 * p) skip[i] = 1;
      int ns = 0;
      for (int k = 0; k < s; ++k) {
        int i = roughs[k];
        if (skip[i]) continue;
        i64 d = 1LL * i * p;
        larges[ns] = larges[k] - (d <= v ? larges[smalls[d] -
   pc] : smalls[n / d]) + pc;
        roughs[ns++] = i;
      }
      s = ns;
      for (int j = v / p; j >= p; --j) {
        int c = smalls[j] - pc, e = min(j * p + p, v + 1);
        for (int i = j * p; i < e; ++i) smalls[i] -= c;
      }
    }
  }
  for (int k = 1; k < s; ++k) {
    const i64 m = n / roughs[k];
    i64 t = larges[k] - (pc + k - 1);
    for (int l = 1; l < k; ++l) {
      int p = roughs[l];
      if (1LL * p * p > m) break;
      t -= smalls[m / p] - (pc + l - 1);
    }
    larges[0] -= t;
  }
  return larges[0];
}
```

7.17 ModMin

```
// min{k | l <= ((ak) mod m) <= r}, no solution -> -1
int mod_min(int a, int m, int l, int r) {
 if (a == 0) return l ? -1 : 0;
 if (int k = (l + a - 1) / a; k * a <= r)
  return k;
 int b = m / a, c = m % a;
 if (int y = mod_min(c, a, a - r % a, a - l % a))
  return (l + y * c + a - 1) / a + y * b;
 return -1;
}
```

7.18 FFT

```
template<typename C = complex<double>>
void FFT(vector<C> &P, C w, bool inv = 0) {
  int n = P.size(), lg = __builtin_ctz(n);
  assert(__builtin_popcount(n) == 1);

  for (int j = 1, i = 0; j < n - 1; ++j) {
    for (int k = n >> 1; k > (i ^= k); k >>= 1);
    if (j < i) swap(P[i], P[j]);
  }

  vector<C> ws = {inv ? C{1} / w : w};

  rep (i, 1, lg) ws.pb(ws[i - 1] * ws[i - 1]);
  reverse(all(ws));

  rep (i, 0, lg) {
    for (int k = 0; k < n; k += 2 << i) {
      C base = C{1};
      rep (j, k, k + (1 << i)) {
        auto t = base * P[j + (1 << i)];
        auto u = P[j];
        P[j] = u + t;
        P[j + (1 << i)] = u - t;
        base = base * ws[i];
      }
    }
  }

  if (inv) rep (i, 0, n) P[i] = P[i] / C(n);
}

const int N = 1 << 21;
const double PI = acos(-1);
const auto w = exp(-complex<double>(0, 2.0 * PI / N));
```

7.19 NTT prime

- P: 7681, Rt: 17
- P: 12289, Rt: 11
- P: 40961, Rt: 3
- P: 65537, Rt: 3
- P: 786433, Rt: 10
- P: 5767169, Rt: 3
- P: 7340033, Rt: 3
- P: 23068673, Rt: 3
- P: 469762049, Rt: 3
- P: 2061584302081, Rt: 7
- P: 2748779069441, Rt: 3
- P: 167772161, Rt: 3
- P: 104857601, Rt: 3
- P: 985661441, Rt: 3
- P: 998244353, Rt: 3
- P: 1107296257, Rt: 10
- P: 2013265921, Rt: 31
- P: 2810183681, Rt: 11
- P: 2885681153, Rt: 3
- P: 605028353, Rt: 3
- P: 1945555039024054273, Rt: 5
- P: 9223372036737335297, Rt: 3

7.20 Polynomial

```cpp
std::mt19937_64 rng(std::chrono::steady_clock::now().
     time_since_epoch().count());

template <class mint>
void nft(bool type, std::vector<mint> &a) {
  int n = int(a.size()), s = 0;
  while ((1 << s) < n) {
    s++;
  }
  assert(1 << s == n);
  static std::vector<mint> ep, iep;
  while (int(ep.size()) <= s) {
    ep.push_back(power(mint::G, mint(-1).v / (1 << int(ep.size
     ()))));
    iep.push_back(ep.back().inv());
  }
  std::vector<mint> b(n);
  for (int i = 1; i <= s; i++) {
    int w = 1 << (s - i);
    mint base = type ? iep[i] : ep[i], now = 1;
    for (int y = 0; y < n / 2; y += w) {
      for (int x = 0; x < w; x++) {
        auto l = a[y << 1 | x];
        auto r = now * a[y << 1 | x | w];
        b[y | x] = l + r;
        b[y | x | n >> 1] = l - r;
      }
      now *= base;
    }
    std::swap(a, b);
  }
}
template <class mint>
std::vector<mint> multiply(const std::vector<mint> &a, const
     std::vector<mint> &b) {
  int n = int(a.size()), m = int(b.size());
  if (!n || !m) return {};
  if (std::min(n, m) <= 8) {
    std::vector<mint> ans(n + m - 1);
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        ans[i + j] += a[i] * b[j];
      }
    }
    return ans;
  }
  int lg = 0;
  while ((1 << lg) < n + m - 1) {
    lg++;
  }
  int z = 1 << lg;
  auto a2 = a, b2 = b;
  a2.resize(z);
  b2.resize(z);
  nft(false, a2);
  nft(false, b2);
  for (int i = 0; i < z; i++) {
    a2[i] *= b2[i];
  }
  nft(true, a2);
  a2.resize(n + m - 1);
  mint iz = mint(z).inv();
  for (int i = 0; i < n + m - 1; i++) {
    a2[i] *= iz;
  }
  return a2;
}

template <class D>
struct Poly {
  std::vector<D> v;
  Poly(const std::vector<D> &v_ = {}) : v(v_) { shrink(); }
  void shrink() {
    while (v.size() > 1 && !v.back()) {
      v.pop_back();
    }
  }
  int size() const { return int(v.size()); }
  D freq(int p) const { return (p < size()) ? v[p] : D(0); }
  Poly operator+(const Poly &r) const {
    auto n = std::max(size(), r.size());
    std::vector<D> res(n);
    for (int i = 0; i < n; i++) {
      res[i] = freq(i) + r.freq(i);
    }
    return res;
  }
  Poly operator-(const Poly &r) const {
    int n = std::max(size(), r.size());
    std::vector<D> res(n);
    for (int i = 0; i < n; i++) {
      res[i] = freq(i) - r.freq(i);
    }
    return res;
  }
  Poly operator*(const Poly &r) const { return {multiply(v, r.v
     )}; }
  Poly operator*(const D &r) const {
    int n = size();
    std::vector<D> res(n);
    for (int i = 0; i < n; i++) {
      res[i] = v[i] * r;
    }
    return res;
  }
  Poly operator/(const D &r) const { return *this * r.inv(); }
  Poly operator/(const Poly &r) const {
    if (size() < r.size()) return {{}};
    int n = size() - r.size() + 1;
    return (rev().pre(n) * r.rev().inv(n)).pre(n).rev();
  }
  Poly operator%(const Poly &r) const { return *this - *this /
     r * r; }
  Poly operator<<(int s) const {
    std::vector<D> res(size() + s);
    for (int i = 0; i < size(); i++) {
      res[i + s] = v[i];
    }
    return res;
  }
  Poly operator>>(int s) const {
    if (size() <= s) {
      return Poly();
    }
    std::vector<D> res(size() - s);
    for (int i = 0; i < size() - s; i++) {
      res[i] = v[i + s];
    }
    return res;
  }
  Poly &operator+=(const Poly &r) { return *this = *this + r; }
  Poly &operator-=(const Poly &r) { return *this = *this - r; }
  Poly &operator*=(const Poly &r) { return *this = *this * r; }
  Poly &operator*=(const D &r) { return *this = *this * r; }
  Poly &operator/=(const Poly &r) { return *this = *this / r; }
  Poly &operator/=(const D &r) { return *this = *this / r; }
  Poly &operator%=(const Poly &r) { return *this = *this % r; }
  Poly &operator<<=(const size_t &n) { return *this = *this <<
     n; }
  Poly &operator>>=(const size_t &n) { return *this = *this >>
     n; }
  Poly pre(int le) const {
    return {{v.begin(), v.begin() + std::min(size(), le)}};
  }
  Poly rev(int n = -1) const {
    std::vector<D> res = v;
    if (n != -1) {
      res.resize(n);
    }
    std::reverse(res.begin(), res.end());
    return res;
  }
  Poly diff() const {
    std::vector<D> res(std::max(0, size() - 1));
    for (int i = 1; i < size(); i++) {
      res[i - 1] = freq(i) * i;
    }
    return res;
  }
  Poly inte() const {
    std::vector<D> res(size() + 1);
    for (int i = 0; i < size(); i++) {
      res[i + 1] = freq(i) / (i + 1);
    }
    return res;
  }
```

```
  // f * f.inv() = 1 + g(x)x^m
  Poly inv(int m) const {
    Poly res = Poly({D(1) / freq(0)});
    for (int i = 1; i < m; i *= 2) {
      res = (res * D(2) - res * res * pre(2 * i)).pre(2 * i);
    }
    return res.pre(m);
  }
  Poly exp(int n) const {
    assert(freq(0) == 0);
    Poly f({1}), g({1});
    for (int i = 1; i < n; i *= 2) {
      g = (g * 2 - f * g * g).pre(i);
      Poly q = diff().pre(i - 1);
      Poly w = (q + g * (f.diff() - f * q)).pre(2 * i - 1);
      f = (f + f * (*this - w.inte()).pre(2 * i)).pre(2 * i);
    }
    return f.pre(n);
  }
  Poly log(int n) const {
    assert(freq(0) == 1);
    auto f = pre(n);
    return (f.diff() * f.inv(n - 1)).pre(n - 1).inte();
  }
  Poly pow(int n, i64 k) const {
    int m = 0;
    while (m < n && freq(m) == 0) m++;
    Poly f(std::vector<D>(n, 0));
    if (k && m && (k >= n || k * m >= n)) return f;
    f.v.resize(n);
    if (m == n) return f.v[0] = 1, f;
    int le = m * k;
    Poly g({v.begin() + m, v.end()});
    D base = power<D>(g.freq(0), k), inv = g.freq(0).inv();
    g = ((g * inv).log(n - m) * D(k)).exp(n - m);
    for (int i = le; i < n; i++) f.v[i] = g.freq(i - le) * base
     ;
    return f;
  }
  Poly Getsqrt(int n) const {
    if (size() == 0) return {{0}};
    int z = QuadraticResidue(freq(0).v, 998244353);
    if (z == -1) return Poly{};
    Poly f = pre(n + 1);
    Poly g({z});
    for (int i = 1; i < n; i *= 2) {
      g = (g + f.pre(2 * i) * g.inv(2 * i)) / 2;
    }
    return g.pre(n + 1);
  }
  Poly sqrt(int n) const {
    int m = 0;
    while (m < n && freq(m) == 0) m++;
    if (m == n) return {{0}};
    if (m & 1) return Poly{};
    Poly s = Poly(std::vector<D>(v.begin() + m, v.end())).
     Getsqrt(n);
    if (s.size() == 0) return Poly{};
    std::vector<D> res(n);
    for (int i = 0; i + m / 2 < n; i++) res[i + m / 2] = s.freq
     (i);
    return Poly(res);
  }
  Poly modpower(u64 n, const Poly &mod) {
    Poly x = *this, res = {{1}};
    for (; n; n /= 2, x = x * x % mod) {
      if (n & 1) {
        res = res * x % mod;
      }
    }
    return res;
  }
  friend std::ostream &operator<<(std::ostream &os, const Poly
     &p) {
    if (p.size() == 0) {
      return os << "0";
    }
    for (auto i = 0; i < p.size(); i++) {
      if (p.v[i]) {
        os << p.v[i] << "x^" << i;
        if (i != p.size() - 1) {
          os << "+";
        }
      }
    }
    return os;
  }
};
template <class mint>
struct MultiEval {
  using NP = MultiEval *;
  NP l, r;
  int sz;
  Poly<mint> mul;
  std::vector<mint> que;
  MultiEval(const std::vector<mint> &que_, int off, int sz_) :
     sz(sz_) {
    if (sz <= 100) {
      que = {que_.begin() + off, que_.begin() + off + sz};
      mul = {{1}};
      for (auto x : que) {
        mul *= {{-x, 1}};
      }
      return;
    }
    l = new MultiEval(que_, off, sz / 2);
    r = new MultiEval(que_, off + sz / 2, sz - sz / 2);
    mul = l->mul * r->mul;
  }
  MultiEval(const std::vector<mint> &que_) : MultiEval(que_, 0,
      int(que_.size())) {}
  void query(const Poly<mint> &pol_, std::vector<mint> &res)
     const {
    if (sz <= 100) {
      for (auto x : que) {
        mint sm = 0, base = 1;
        for (int i = 0; i < pol_.size(); i++) {
          sm += base * pol_.freq(i);
          base *= x;
        }
        res.push_back(sm);
      }
      return;
    }
    auto pol = pol_ % mul;
    l->query(pol, res);
    r->query(pol, res);
  }
  std::vector<mint> query(const Poly<mint> &pol) const {
    std::vector<mint> res;
    query(pol, res);
    return res;
  }
};
template <class mint>
Poly<mint> berlekampMassey(const std::vector<mint> &s) {
  int n = int(s.size());
  std::vector<mint> b = {mint(-1)}, c = {mint(-1)};
  mint y = mint(1);
  for (int ed = 1; ed <= n; ed++) {
    int l = int(c.size()), m = int(b.size());
    mint x = 0;
    for (int i = 0; i < l; i++) {
      x += c[i] * s[ed - l + i];
    }
    b.push_back(0);
    m++;
    if (!x) {
      continue;
    }
    mint freq = x / y;
    if (l < m) {
      // use b
      auto tmp = c;
      c.insert(begin(c), m - l, mint(0));
      for (int i = 0; i < m; i++) {
        c[m - 1 - i] -= freq * b[m - 1 - i];
      }
      b = tmp;
      y = x;
    } else {
      // use c
      for (int i = 0; i < m; i++) {
        c[l - 1 - i] -= freq * b[m - 1 - i];
      }
    }
  }
  return c;
```

```
}
template <class E, class mint = decltype(E().f)>
mint sparseDet(const std::vector<std::vector<E>> &g) {
  int n = int(g.size());
  if (n == 0) {
    return 1;
  }
  auto randV = [&]() {
    std::vector<mint> res(n);
    for (int i = 0; i < n; i++) {
      res[i] = mint(std::uniform_int_distribution<i64>(1, mint
      (-1).v)(rng)); // need rng
    }
    return res;
  };
  std::vector<mint> c = randV(), l = randV(), r = randV();
  // l * mat * r
  std::vector<mint> buf(2 * n);
  for (int fe = 0; fe < 2 * n; fe++) {
    for (int i = 0; i < n; i++) {
      buf[fe] += l[i] * r[i];
    }
    for (int i = 0; i < n; i++) {
      r[i] *= c[i];
    }
    std::vector<mint> tmp(n);
    for (int i = 0; i < n; i++) {
      for (auto e : g[i]) {
        tmp[i] += r[e.to] * e.f;
      }
    }
    r = tmp;
  }
  auto u = berlekampMassey(buf);
  if (u.size() != n + 1) {
    return sparseDet(g);
  }
  auto acdet = u.freq(0) * mint(-1);
  if (n % 2) {
    acdet *= mint(-1);
  }
  if (!acdet) {
    return 0;
  }
  mint cdet = 1;
  for (int i = 0; i < n; i++) {
    cdet *= c[i];
  }
  return acdet / cdet;
}
```

7.21 Theorem

- Pick's Theorem
  $A = i + \frac{b}{2} - 1$
  A: Area、i: grid number in the inner、b: grid number on the side

- Matrix-Tree theorem
  undirected graph
  $D_{ii}(G) = \text{deg}(i), D_{ij} = 0, i \neq j$
  $A_{ij}(G) = A_{ji}(G) = \#e(i,j), i \neq j$
  $L(G) = D(G) - A(G)$
  $t(G) = \det L(G)\binom{1,2,\cdots,i-1,i+1,\cdots,n}{1,2,\cdots,i-1,i+1,\cdots,n}$
  leaf to root
  $D^{out}_{ii}(G) = \text{deg}^{\text{out}}(i), D^{out}_{ij} = 0, i \neq j$
  $A_{ij}(G) = \#e(i,j), i \neq j$
  $L^{out}(G) = D^{out}(G) - A(G)$
  $t^{root}(G,k) = \det L^{out}(G)\binom{1,2,\cdots,k-1,k+1,\cdots,n}{1,2,\cdots,k-1,k+1,\cdots,n}$
  root to leaf
  $L^{in}(G) = D^{in}(G) - A(G)$
  $t^{leaf}(G,k) = \det L^{in}(G)\binom{1,2,\cdots,k-1,k+1,\cdots,n}{1,2,\cdots,k-1,k+1,\cdots,n}$

- Derangement
  $D_n = (n-1)(D_{n-1} + D_{n-2}) = nD(n-1) + (-1)^n$

- Möbius Inversion
  $f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu(\frac{n}{d})f(d)$

- Euler Inversion
  $\sum_{i|n} \varphi(i) = n$

- Binomial Inversion
  $f(n) = \sum_{i=0}^{n} \binom{n}{i} g(i) \Leftrightarrow \ g(n) = \sum_{i=0}^{n} (-1)^{n-i}\binom{n}{i} f(i)$

- Subset Inversion
  $f(S) = \sum_{T \subseteq S} g(T) \Leftrightarrow g(S) = \sum_{T \subseteq S} (-1)^{|S|-|T|} f(T)$

- Min-Max Inversion
  $\max_{i \in S} x_i = \sum_{T \subseteq S} (-1)^{|T|-1} \min_{j \in T} x_j$

- Ex Min-Max Inversion
  $\text{kthmax}_{i \in S}\, x_i = \sum_{T \subseteq S} (-1)^{|T|-k} \binom{|T|-1}{k-1} \min_{j \in T} x_j$

- Lcm-Gcd Inversion
  $\text{lcm}_{i \in S} x_i = \prod_{T \subseteq S} \left(\text{gcd}_{j \in T} x_j\right)^{(-1)^{|T|-1}}$

- Sum of powers
  $\sum_{k=1}^{n} k^m = \frac{1}{m+1} \sum_{k=0}^{m} \binom{m+1}{k} B_k^+ \, n^{m+1-k}$
  $\sum_{j=0}^{m} \binom{m+1}{j} B_j^- = 0$
  note: $B_1^+ = -B_1^-, B_i^+ = B_i^-$

- Cayley's formula
  number of trees on n labeled vertices: n^{n-2}
  Let $T_{n,k}$ be the number of labelled forests on n vertices with k connected components, such that vertices 1, 2, ..., k all belong to different connected components. Then $T_{n,k} = kn^{n-k-1}$.

- High order residue
  $[d^{\frac{p-1}{(n,p-1)}} \equiv 1]$

- Packing and Covering
  |maximum independent set| + |minimum vertex cover| = $|V|$

- Kőnig's theorem
  |maximum matching| = |minimum vertex cover|

- Dilworth's theorem
  width = |largest antichain| = |smallest chain decomposition|

- Mirsky's theorem
  height = |longest chain| = |smallest antichain decomposition| = |minimum anticlique partition|

- Lucas'Theorem
  For $n, m \in \mathbb{Z}^*$ and prime P, $\binom{m}{n}$ mod $P = \Pi\binom{m_i}{n_i}$ where m_i is the i-th digit of m in base P.

- Stirling approximation
  $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n e^{\frac{1}{12n}}$

- 1st Stirling Numbers(permutation $|P| = n$ with k cycles)
  $S(n,k)$ = coefficient of x^k in $\Pi_{i=0}^{n-1}(x+i)$
  $S(n+1,k) = nS(n,k) + S(n,k-1)$

- 2nd Stirling Numbers(Partition n elements into k non-empty set)
  $S(n,k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n$
  $S(n+1,k) = kS(n,k) + S(n,k-1)$

- Catalan number
  $C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n-1}$
  $\binom{n+m}{n} - \binom{n+m}{n+1} = (m+n)!\frac{n-m+1}{n+1}$ for $n \geq m$
  $C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$
  $C_0 = 1$ and $C_{n+1} = 2(\frac{2n+1}{n+2})C_n$
  $C_0 = 1$ and $C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$ for $n \geq 0$

- Extended Catalan number
  $\frac{1}{(k-1)n+1}\binom{kn}{n}$

- Calculate $c[i-j] += a[i] \times b[j]$ for $a[n], b[m]$
  1. a=reverse(a); c=mul(a,b); c=reverse(c[:n]);
  2. b=reverse(b); c=mul(a,b); c=rshift(c,m-1);

- Eulerian number (permutation $1 \sim n$ with m $a[i] > a[i-1]$)
  $A(n,m) = \sum_{i=0}^{m} (-1)^i \binom{n+1}{i} (m+1-i)^n$
  $A(n,m) = (n-m)A(n-1,m-1) + (m+1)A(n-1,m)$

- Hall's theorem
  Let $G = (X+Y, E)$ be a bipartite graph. For $W \subseteq X$, let $N(W) \subseteq Y$ denotes the adjacent vertices set of W. Then, G has a X'-perfect matching (matching contains $X' \subseteq X$) iff $\forall W \subseteq X', |W| \leq |N(W)|$.

- Tutte Matrix:
  For a graph $G = (V, E)$, its maximum matching $= \frac{rank(A)}{2}$ where $A_{ij} = ((i,j) \in E ? (i < j ? x_{ij} : -x_{ji}) : 0)$ and x_{ij} are random numbers.

- Erdoš–Gallai theorem
  There exists a simple graph with degree sequence $d_1 \geq \cdots \geq d_n$ iff $\sum_{i=1}^{n} d_i$ is even and $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k), \forall 1 \leq k \leq n$

- Euler Characteristic
  planar graph: $V - E + F - C = 1$
  convex polyhedron: $V - E + F = 2$
  V, E, F, C: number of vertices, edges, faces(regions), and components

- Burnside Lemma
  $|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$

- Polya theorem
  $|Y^x/G| = \frac{1}{|G|}\sum_{g\in G} m^{c(g)}$
  $m = |Y|$: num of colors, c(g) : num of cycle

- Cayley's Formula
  Given a degree sequence $d_1, \ldots, d_n$ of a labeled tree, there are $\frac{(n-2)!}{(d_1-1)!\cdots(d_n-1)!}$ spanning trees.

- Find a Primitive Root of n:
  n has primitive roots iff $n = 2, 4, p^k, 2p^k$ where p is an odd prime.
  1. Find $\phi(n)$ and all prime factors of $\phi(n)$, says $P = \{p_1, ..., p_m\}$
  2. $\forall g \in [2, n)$, if $g^{\frac{\phi(n)}{p_i}} \neq 1, \forall p_i \in P$, then g is a primitive root.
  3. Since the smallest one isn't too big, the algorithm runs fast.
  4. n has exactly $\phi(\phi(n))$ primitive roots.

- Taylor series
  $f(x) = f(c) + f'(c)(x-c) + \frac{f^{(2)}(c)}{2!}(x-c)^2 + \frac{f^{(3)}(c)}{3!}(x-c)^3 + \cdots$

- Lagrange Multiplier
  min $f(x, y)$, subject to $g(x, y) = 0$
  $\frac{\partial f}{\partial x} + \lambda\frac{\partial g}{\partial x} = 0$
  $\frac{\partial f}{\partial y} + \lambda\frac{\partial g}{\partial y} = 0$
  $g(x, y) = 0$

- Calculate $f(x+n)$ where $f(x) = \sum_{i=0}^{n-1} a_i x^i$
  $f(x+n) = \sum_{i=0}^{n-1} a_i (x+n)^i = \sum_{i=0}^{n-1} x^i \cdot \frac{1}{i!}\sum_{j=i}^{n-1}\frac{a_j}{j!}\cdot\frac{n^{j-i}}{(j-i)!}$

- Bell 數 (有 n 個人, 把他們拆組的方法總數)
  $B_0 = 1$
  $B_n = \sum_{k=0}^{n} s(n,k) \quad (second - stirling)$
  $B_{n+1} = \sum_{k=0}^{n}\binom{n}{k}B_k$

- Wilson's theorem
  $(p-1)! \equiv -1 \pmod{p}$
  $(p^q!)_p \equiv \begin{cases} 1, & (p=2)\wedge(q\geq 3), \\ -1, & \text{otherwise.} \end{cases} \pmod{p^q}$

- Fermat's little theorem
  $a^p \equiv a \pmod{p}$

- Euler's theorem
  $a^b \equiv \begin{cases} a^{b \bmod \varphi(m)}, & \gcd(a,m) = 1, \\ a^b, & \gcd(a,m) \neq 1, b < \varphi(m), \\ a^{(b \bmod \varphi(m)) + \varphi(m)}, & \gcd(a,m) \neq 1, b \geq \varphi(m). \end{cases} \pmod{m}$

- 環狀著色 (相鄰塗異色)
  $(k-1)(-1)^n + (k-1)^n$

8 Stringology

8.1 Aho–Corasick AM

```
struct ACM {
  int idx = 0;
  vector<array<int, 26>> tr;
  vector<int> cnt, fail;

  void clear() {
    tr.resize(1, array<int, 26>{});
    cnt.resize(1, 0);
    fail.resize(1, 0);
  }

  ACM() {
    clear();
  }

  int newnode() {
    tr.push_back(array<int, 26>{});
    cnt.push_back(0);
    fail.push_back(0);
    return ++idx;
  }

  void insert(string &s) {
    int u = 0;
    for (char c : s) {
      c -= 'a';
      if (tr[u][c] == 0) tr[u][c] = newnode();
      u = tr[u][c];
    }
    cnt[u]++;
  }

  void build() {
    queue<int> q;
    rep (i, 0, 26) if (tr[0][i]) q.push(tr[0][i]);
    while (!q.empty()) {
      int u = q.front(); q.pop();
      rep (i, 0, 26) {
        if (tr[u][i]) {
          fail[tr[u][i]] = tr[fail[u]][i];
          cnt[tr[u][i]] += cnt[fail[tr[u][i]]];
          q.push(tr[u][i]);
        } else {
          tr[u][i] = tr[fail[u]][i];
        }
      }
    }
  }

  int query(string &s) {
    int u = 0, res = 0;
    for (char c : s) {
      c -= 'a';
      u = tr[u][c];
      res += cnt[u];
    }
    return res;
  }
};
```

8.2 Double String

```
// need zvalue
int ans = 0;
auto dc = [&](auto self, string cur) -> void {
  int m = cur.size();
  if (m <= 1) return;
  string _s = cur.substr(0, m / 2), _t = cur.substr(m / 2, m);
  self(self, _s);
  self(self, _t);
  rep (T, 0, 2) {
    int m1 = _s.size(), m2 = _t.size();
    string s = _t + "$" + _s, t = _s;
    reverse(all(t));
    zvalue z1(s), z2(t);
    auto get_z = [&](zvalue &z, int x) -> int {
      if (0 <= x && x < z.z.size()) return z[x];
      return 0;
    };
    rep (i, 0, m1) if (_s[i] == _t[0]) {
      int len = m1 - i;
      int L = m1 - min(get_z(z2, m1 - i), len - 1),
        R = get_z(z1, m2 + 1 + i);
      if (T == 0) R = min(R, len - 1);
      R = i + R;
      ans += max(0, R - L + 1);
    }
    swap(_s, _t);
    reverse(all(_s));
    reverse(all(_t));
  }
};
dc(dc, str);
```

8.3 Lyndon Factorization

```
// partition s = w[0] + w[1] + ... + w[k-1],
// w[0] >= w[1] >= ... >= w[k-1]
// each w[i] strictly smaller than all its suffix
// min rotate: last < n of duval_min(s + s)
// max rotate: last < n of duval_max(s + s)
// min suffix: last of duval_min(s)
// max suffix: last of duval_max(s + -1)
vector<int> duval(const auto &s) {
  int n = s.size(), i = 0;
  vector<int> pos;
  while (i < n) {
    int j = i + 1, k = i;
    while (j < n and s[k] <= s[j]) { // >=
      if (s[k] < s[j]) k = i; // >
      else k++;
      j++;
    }
    while (i <= k) {
      pos.push_back(i);
      i += j - k;
    }
  }
  pos.push_back(n);
```

```
  return pos;
}
```

8.4 Manacher

```
/* center i: radius z[i * 2 + 1] / 2
  center i, i + 1: radius z[i * 2 + 2] / 2
  both aba, abba have radius 2 */
vector<int> manacher(const string &tmp) { // 0-based
  string s = "%";
  int l = 0, r = 0;
  for (char c : tmp) s += c, s += '%';
  vector<int> z(ssize(s));
  for (int i = 0; i < ssize(s); i++) {
    z[i] = r > i ? min(z[2 * l - i], r - i) : 1;
    while (i - z[i] >= 0 && i + z[i] < ssize(s) && s[i + z[i]]
     == s[i - z[i]])
    ++z[i];
    if (z[i] + i > r) r = z[i] + i, l = i;
  }
  return z;
}
```

8.5 SA-IS

```
auto sais(const auto &s) {
  const int n = (int)s.size(), z = ranges::max(s) + 1;
  if (n == 1) return vector{0};
  vector<int> c(z); for (int x : s) ++c[x];
  partial_sum(all(c), begin(c));
  vector<int> sa(n); auto I = views::iota(0, n);
  vector<bool> t(n); t[n - 1] = true;
  for (int i = n - 2; i >= 0; i--)
    t[i] = (s[i] == s[i + 1] ? t[i + 1] : s[i] < s[i + 1]);
  auto is_lms = views::filter([&t](int x) {
    return x && t[x] & !t[x - 1];
  });
  auto induce = [&] {
    for (auto x = c; int y : sa)
      if (y-- and !t[y]) sa[x[s[y] - 1]++] = y;
    for (auto x = c; int y : sa | views::reverse)
      if (y-- and t[y]) sa[--x[s[y]]] = y;
  };
  vector<int> lms, q(n); lms.reserve(n);
  for (auto x = c; int i : I | is_lms) {
    q[i] = int(lms.size());
    lms.push_back(sa[--x[s[i]]] = i);
  }
  induce(); vector<int> ns(lms.size());
  for (int j = -1, nz = 0; int i : sa | is_lms) {
    if (j >= 0) {
      int len = min({n - i, n - j, lms[q[i] + 1] - i});
      ns[q[i]] = nz += lexicographical_compare(
        s.begin() + j, s.begin() + j + len,
        s.begin() + i, s.begin() + i + len
      );
    }
    j = i;
  }
  ranges::fill(sa, 0); auto nsa = sais(ns);
  for (auto x = c; int y : nsa | views::reverse)
    y = lms[y], sa[--x[s[y]]] = y;
  return induce(), sa;
}
// sa[i]: sa[i]-th suffix is the
// i-th lexicographically smallest suffix.
// lcp[i]: LCP of suffix sa[i] and suffix sa[i + 1].
struct Suffix {
  int n;
  vector<int> sa, rk, lcp;
  Suffix(const auto &s) : n(s.size()),
    lcp(n - 1), rk(n) {
    vector<int> t(n + 1); // t[n] = 0
    copy(all(s), t.begin()); // s shouldn't contain 0
    sa = sais(t); sa.erase(sa.begin());
    for (int i = 0; i < n; i++) rk[sa[i]] = i;
    for (int i = 0, h = 0; i < n; i++) {
      if (!rk[i]) { h = 0; continue; }
      for (int j = sa[rk[i] - 1];
          i + h < n and j + h < n
          and s[i + h] == s[j + h];) ++h;
      lcp[rk[i] - 1] = h ? h-- : 0;
    }
  }
};
```

8.6 Suffix Array

```
struct SuffixArray {
  int n;
  vector<int> suf, rk, S;
  SuffixArray(vector<int> _S) : S(_S) {
    n = S.size();
    suf.assign(n, 0);
    rk.assign(n * 2, -1);
    iota(all(suf), 0);
    for (int i = 0; i < n; i++) rk[i] = S[i];
    for (int k = 2; k < n + n; k *= 2) {
      auto cmp = [&](int a, int b) -> bool {
        return rk[a] == rk[b] ? (rk[a + k / 2] < rk[b + k / 2])
            : (rk[a] < rk[b]);
      };
      sort(all(suf), cmp);
      auto tmp = rk;
      tmp[suf[0]] = 0;
      for (int i = 1; i < n; i++) {
        tmp[suf[i]] = tmp[suf[i - 1]] + cmp(suf[i - 1], suf[i])
      ;
      }
      rk.swap(tmp);
    }
  }
};
```

8.7 Z-value

```
struct zvalue {
  vector<int> z;
  int operator[] (const int &x) const {
    return z[x];
  }
  zvalue(string s) {
    int n = s.size();
    z.resize(n);
    z[0] = 0;
    for (int i = 1, l = 1, r = 0; i < n; i++) {
      z[i] = min(z[i - l], max<int>(0, r - i));
      while (i + z[i] < n && s[i + z[i]] == s[z[i]]) z[i]++;
      if (i + z[i] > r) l = i, r = i + z[i];
    }
  }
};
```