Contents

Template

```
#include "bits/stdc++.h"
using namespace std;

typedef long long ll;
#define endl '\n'
#define int ll
//====================//
void magic() { }

signed main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0), cout.tie(0);
  int t = 1;
  while (t--) magic();
}
```

Pragma

```
#pragma GCC optimize("O3")
#pragma GCC optimize("Ofast,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
#pragma GCC target("avx,avx2,fma")
```

Custom Hash

```
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);
  }
};
```

Submask Loop

```
for (int s = mask; s; s = (s - 1) & mask) {


}
```

Mint

```
const int MOD = 1e9 + 7;
struct mint {
  int value;
  mint(int v = 0) { value = v % MOD; if(value < 0) value += MOD; }
  mint(int a, int b) : value(0) { *this += a; *this /= b; }

  mint& operator+=(mint const& b) { value += b.value; if(value >= MOD) value -= MOD; return *this; }
  mint& operator-=(mint const& b) { value -= b.value; if(value < 0) value += MOD; return *this; }
  mint& operator*=(mint const& b) { value = 1ll * value * b.value % MOD; return *this; }

  friend mint power(mint a, ll e) {
    mint res = 1; while(e) { if(e & 1) res *= a; a *= a; e >>= 1; }
    return res;
  }
  friend mint inv(mint a) { return power(a, MOD - 2); }
  mint &operator/=(mint const& b) { return *this *= inv(b); }
  friend mint operator+(mint a, mint const b) { return a += b; }
  friend mint operator-(mint a, mint const b) { return a -= b; }
  friend mint operator-(mint const a) { return 0 - a; }
  friend mint operator*(mint a, mint const b) { return a *= b; }
  friend mint operator/(mint a, mint const b) { return a /= b; }
  friend ostream& operator<<(ostream& os, mint const& a) { return os << a.value; }
  friend bool operator==(mint const &a, mint const& b) { return a.value == b.value; }
  friend bool operator!=(mint const& a, mint const& b) { return a.value != b.value; }
};
```

Comb

```
struct Comb {
  vector<mint> fact, invr;
  Comb(int n): fact(n+1, 1), invr(n+1, 1) {
    for (int i = 1; i <= n; ++i)
      fact[i]=fact[i-1]*i, invr[i]=inv(fact[i]);
  }
  mint nPr(int n, int r) {return n<r?0:fact[n]*invr[n - r];}
  mint nCr(int n, int r) {return nPr(n, r)*invr[r];}
  mint SAndBars(int n, int k) {return min(n, k)<0?0:nCr(n+k-1, k-1);}
  mint catalan(int n) {return nCr(n * 2, n) / (n + 1);}
};
```

Cycle Take All Edges

```
vector<int> res, vis;
vector<vector<pair<int, int>>> adj;

void dfs(int u) {
  while (adj[u].size()) {
    auto [v, i] = adj[u].back();
    adj[u].pop_back();
    if (vis[i]) continue;
    vis[i] = 1;
    dfs(v);
  }
  res.push_back(u);
}
```

Path Take All Edges

```
vector<int> res;
vector<vector<int>> adj;
void dfs(int s){
  while(adj[s].size()){
    int u = adj[s].back(); adj[s].pop_back();
    dfs(u);
  }
  res.push_back(s);
}

void check() {

  for(int i = 2; i < n; i++)
    if(in[i] != out[i])
      return "IMPOSSIBLE";

  if(in[1] != out[1] - 1 || out[n] != in[n] - 1)
    return "IMPOSSIBLE";

  dfs(1);
  reverse(res.begin(), res.end());
  if(res.size() != m + 1 || res.back() != n) {
    return "IMPOSSIBLE";

  for(auto v : res)
    cout << v << " ";
}
```

De Burijn

```
unordered_map<string, int> id;
string s[MX_SIZE];
int cnt = 1, n, k = 2;
vector<char> a{'0', '1'};

void rec(string cur){
  if(cur.size() == n){
    id[cur] = cnt;
    s[cnt++] = cur;
    return;
  }

  for(int i = 0; i < a.size(); i++)
    rec(cur + a[i]);
}

vector<int> adj[MX_SIZE];
bool vis[MX_SIZE];
string res;
void dfs(int u) {
  for (auto v: adj[u])
    if (!vis[v]) {
      vis[v] = 1;
      dfs(v);
      res.push_back(s[v].back());
    }
}

void makeAdj(){
  int mx = (1<<n);
  for(int i = 1; i <= mx; i++){
    string cur = s[i].substr(1, n - 1);
    for(int j = 0; j < a.size(); j++)
      adj[i].push_back(id[cur + a[j]]);
  }
}

string makeDeBruijn() {
  rec("");
  makeAdj();

  dfs(1);
  s[1].pop_back();
  return res + s[1];
}
```

Tree Isomorphim

```
int n;
int treeID = 0;
vector<vector<int>> adj[2];
vector<int> sz[2], cent[2], name[2];
map<vector<int>, int> mp;

void getCent(int t, int u, int p) {
  sz[t][u] = 1;
  bool isCent = true;
  for(int &v: adj[t][u]){
    if(v == p) continue;
    getCent(t, v, u);
    sz[t][u] += sz[t][v];
    if(sz[t][v] > n / 2) isCent = false;
  }

  if(n - sz[t][u] > n / 2) isCent = false;
  if(isCent) cent[t].push_back(u);
}

void dfs(int t, int u, int p){
  vector<int> ch;
  for(int v : adj[t][u]){
    if(v == p) continue;
    dfs(t, v, u);
    ch.push_back(name[t][v]);
  }

  sort(ch.begin(), ch.end());
  if(!mp[ch]) mp[ch] = ++treeID;
  name[t][u] = mp[ch];
}

bool isIso() {
  // assign sizes first
  // you only need the dfs if the tree is rooted
  getCent(0, 1, 0);
  getCent(1, 1, 0);

  for(auto &a: cent[0])
    for(int &b: cent[1]) {
      dfs(0, a, 0), dfs(1, b, 0);
      if(name[0][a] == name[1][b]) return true;
    }

  return false;
}
```

Binary Lifting

```
const int B = 21;
vector<vector<int>> adj;
vector<array<int, B>> up;
vector<int> lvl;

void dfs(int u, int p = 0, int d = 0) {
  lvl[u] = d;
  up[u][0] = p;
  for(int i = 1; i < B; ++i)
    up[u][i] = up[up[u][i - 1]][i - 1];

  for(int &v: adj[u])
    if(v != p) dfs(v, u, d + 1);
}

int kthAncestor(int u, int k) {
  for(int i = B - 1; ~i; --i)
    if(k >> i & 1) u = up[u][i];
  return u;
}

int getLca(int u, int v) {
  if(lvl[u] > lvl[v]) swap(u, v);
  v = kthAncestor(v, lvl[v] - lvl[u]);

  for(int i = B - 1; ~i; --i)
    if(up[u][i] != up[v][i])
      u = up[u][i], v = up[v][i];

  return u == v? u : up[v][0];
}

int getDist(int u, int v) {
  return lvl[u] + lvl[v] - 2 * lvl[getLca(u, v)];
}
```

DSU

```
struct DSU {
  int n, cnt;
  vector<int> p, sz;

  DSU(int n): n(n), cnt(n), p(n + 1), sz(n + 1, 1) {
    iota(p.begin(), p.end(), 0);
  }

  int find(int u) {
    if(u == p[u]) return u;
    return p[u] = find(p[u]);
  }

  bool same(int u, int v) { return find(u) == find(v); }

  bool merge(int u, int v) {
    int uP = find(u), vP = find(v);
    if(vP == uP) return false;

    if(sz[vP] > sz[uP]) swap(uP, vP);
    sz[uP] += sz[vP], p[vP] = uP, cnt--;

    return true;
  }
};
```

DSU Rollback

```
struct DSURollback {
  vector<int> parent, set_size;
  stack<array<int, 5>> st;
  int cc;

  void make_set(int u) {
    if(set_size[u]) return;
    parent[u] = u;
    set_size[u] = 1;
    ++cc;
  }

  DSURollback(int n) { // 1-indexed
    cc = 0;
    parent.assign(n + 1, 0);
    set_size.assign(n + 1, 0);
    for (int i = 1; i <= n; ++i) make_set(i);
  }

  int find(int u) {
    if(parent[u] == u) return u;
    return find(parent[u]);
  }

  bool union_set(int u, int v) {

    int a = find(u);
    int b = find(v);
    if(a == b) {
      st.push({-1, -1, -1, -1, -1});
      return 0;
    }

    if(set_size[a] < set_size[b])
      swap(a, b);

    st.push({b, parent[b], a, set_size[a], cc});
    parent[b] = a;
    set_size[a] += set_size[b];
    --cc;

    return 1;
  }

  void roll_back() {
    if(st.empty()) return;
    array<int, 5> tp = st.top();
    st.pop();
    if(tp[0] == -1) return;
    parent[tp[0]] = tp[1];
    set_size[tp[2]] = tp[3];
    cc = tp[4];
  }
};
```

Bellman

```
struct Edge {
  int from, to, w;
  Edge() : to(0), from(0), w(0) {};
  Edge(int f, int t, int ww) : from(f), to(t), w(ww) {};
};

struct Node {
  long long dist;
  int par;
  bool reachCycle;
  Node() : dist(oo), par(-1), reachCycle(0) {};
};

struct bellmanFord {
  int n, m;
  bool negative_cycle = 0;
  vector<Edge> edges;
  vector<Node> node;

  bellmanFord(int nn, int mm, vector<Edge> &e) {
    n = nn, m = mm;
    edges.resize(m);
    node.resize(n + 2);
    edges = e;
  }

  void build(int src, int startCost) {
    node[src].dist = startCost;
    node[src].par = src;

    for (int i = 1; i <= n; ++i) {
      bool exist = 0;
      for (long long j = 0; j < m; ++j) {
        auto [u, v, w] = edges[j];

        if (node[u].dist < oo && node[v].dist > node[u].dist + w) {
          if (i == n) negative_cycle = 1;
          node[v].dist = node[u].dist + w;
          node[v].par = u;
          exist = 1;
        }
      }
      if (!exist) break;
    }
  }

  void buildReachCycle(int src) {
    vector<long long> oldDist(n + 1);
    for (int i = 1; i <= n; i++)
      oldDist[i] = node[i].dist;

    build(src, oldDist[src]);
    for (int i = 1; i <= n; i++)
      node[i].reachCycle = (oldDist[i] != node[i].dist);
  }
};
```

Floyd

```
vector<vector<int>> d(n + 1, vector<int>(n + 1, oo));

for(int u = 1; u <= n; ++u) d[u][u] = 0;


for(int i = 0; i < m; ++i) {

  int u, v, c; cin >> u >> v >> c;

  d[u][v] = d[v][u] = min(d[u][v], c);

}


for(int k = 1; k <= n; ++k)

  for(int i = 1; i <= n; ++i)

    for(int j = 1; j <= n; ++j)

      d[i][j] = min(d[i][j], d[i][k] + d[k][j])
```

Tarjan

```
vector<int> in, low;
int timer = 1;

// In case of multi edges, skip parent only one time
void dfs(int u, int p = -1) {
  in[u] = low[u] = timer++;

  int ch = 0;
  for (int v : adj[u]) {
    if (v == p) continue;
    if (in[v]) low[u] = min(low[u], in[v]);
    else {
      dfs(v, u);
      low[u] = min(low[u], low[v]);

      if (low[v] > in[u]) IS_BRIDGE(u, v);
      if (low[v] >= in[u] && p!=-1) IS_ART(u);

      ++ch;
    }
  }

  //   if(p == -1 && ch > 1)
  //      IS_ART(v);
}
```

Dinic

```
// O(V^2 * E)
// O(E * Sqrt(V)) in maximum matching problem (Unit Networks)
static const int oo = 2e15;

struct Edge {
  int u, v, flow = 0, cap = 0; // keep the order

  Edge(int u, int v): u(u), v(v) {}
  Edge(int u, int v, int c): u(u), v(v), cap(c) {}

  int rem() { return cap - flow; }
};

struct Dinic {
  int n, s, t, id = 1, flow = 0;
  vector<Edge> edges;
  vector<vector<int>> adj;
  vector<int> lvl, ptr;

  Dinic(int n, int src, int sink): n(n), s(src), t(sink) {
    adj.assign(n + 1, {});
    ptr.assign(n + 1, {});
  }

  void addEdge(int u, int v, int w = oo, int undir = 0) {
    adj[u].push_back(edges.size());
    edges.push_back(Edge(u, v, w));
    adj[v].push_back(edges.size());
    edges.push_back(Edge(v, u, w * undir));
  }

  void move() {
    while(bfs()) {
      ptr.assign(n + 1, {});
      while (int f = dfs(s)) flow += f;
    }
  }

  bool bfs() {
    lvl.assign(n + 1, -1);
    queue<int> q;
    q.push(s), lvl[s] = 0;

    while(!q.empty()) {
      auto u = q.front();
      q.pop();

      for(auto &i: adj[u]) {
        auto &[_, v, f, c] = edges[i];
        if(~lvl[v] || f == c) continue;
        lvl[v] = lvl[u] + 1;
        q.push(v);
      }
    }

    return lvl[t] != -1;
  }

  int dfs(int u, int currFlow = oo) {
    if(u == t) return currFlow;
    if(!currFlow) return 0;

    for(; ptr[u] < adj[u].size(); ++ptr[u]) {
      int i = adj[u][ptr[u]];
      auto [_, v, f, c] = edges[i];
      if(f == c || (lvl[v] != lvl[u] + 1)) continue;

      int bottleNeck = dfs(v, min(currFlow, c - f));
      if(!bottleNeck) continue;

      edges[i].flow += bottleNeck;
      edges[i ^ 1].flow -= bottleNeck;

      return bottleNeck;
    }

    return 0;
  }
};
```

Min Cost Max Flow

```
static const int oo = 2e15;

struct Edge {
  int u, v, flow = 0, cap = 0, cost; // keep the order
  Edge(int u, int v, int c, int cost): u(u), v(v), cap(c), cost(cost) {}
  int rem() { return cap - flow; }
};


struct MCMF {
  int n, s, t, cost = 0, flow = 0;
  vector<Edge> edges;
  vector<vector<int>> adj;
  vector<int> from;

  MCMF(int n, int s, int t): n(n), s(s), t(t) {
    adj.assign(n + 1, {});
  }

  void addEdge(int u, int v, int w = oo, int cost = 0, int undir = 0) {
    adj[u].push_back(edges.size());
    edges.push_back(Edge(u, v, w, cost));
    adj[v].push_back(edges.size());
    edges.push_back(Edge(v, u, w * undir, -cost));
  }

  void move() {
    while (bfs()) {
      int u = t, addflow = oo;
      while (u != s) {
        Edge& e = edges[from[u]];
        addflow = min(addflow, e.rem());
        u = e.u;
      }

      u = t;
      while (u != s) {
        int i = from[u];
        edges[i].flow += addflow;
        edges[i ^ 1].flow -= addflow;
        cost += edges[i].cost * addflow;
        u = edges[i].u;
      }
      flow += addflow;
    }
  }

  bool bfs() {
    from.assign(n + 1, -1);
    vector<int> d(n + 1, oo), state(n + 1, 2);
    deque<int> q;

    state[s] = 1, d[s] = 0;
    q.clear();
    q.push_back(s);

    while (!q.empty()) {
      int u = q.front();
      q.pop_front();
      state[u] = 0;
      for (auto& i : adj[u]) {
        auto& [_, v, f, c, cost] = edges[i];

        if (f >= c || d[v] <= d[u] + cost) continue;

        d[v] = d[u] + cost;

        from[v] = i;
        if (state[v] == 1) continue;
        if (!state[v] || (!q.empty() && d[q.front()] > d[v]))
          q.push_front(v);
        else q.push_back(v);
        state[v] = 1;
      }
    }

    return ~from[t];
  }
};
```

SCC

```
struct SCC {
  int n, timer = 1, sz;
  vector<vector<int>> adj, comp, adjCC;
  vector<int> in, low, id, st;
  vector<bool> stacked;

  SCC(int n): n(n), sz(0) {};

  void build(vector<vector<int>>& _adj) {
    adj = _adj;
    in.assign(n + 1, 0);
    low.assign(n + 1, 0);
    id.assign(n + 1, 0);
    stacked.assign(n + 1, 0);

    for (int u = 1; u <= n; ++u) if (!in[u]) dfs(u);

    condens();
  }

  void dfs(int u) {
    in[u] = low[u] = timer++;
    stacked[u] = 1;
    st.push_back(u);

    for (int& v : adj[u]) {
      if (!in[v]) dfs(v), low[u] = min(low[u], low[v]);
      else if (stacked[v]) low[u] = min(low[u], in[v]);
    }

    if (low[u] == in[u]) {
      comp.push_back({});
      int v = -1;
      while (v != u) {
        v = st.back(), st.pop_back(), stacked[v] = 0;
        id[v] = sz;
        comp.back().push_back(v);
      }


      ++sz;
    }
  }
```

```
    void condens() {
      // new graph is zero indexed
      // BE CAREFUL OF MUTIPLE EDGES
      adjCC.assign(sz, {});
      for (int u = 1; u <= n; ++u)
        for (int& v : adj[u])
          if (id[u] != id[v])
            adjCC[id[u]].push_back(id[v]);
    }
};
```

Bridges

```
struct Bridges {
  int n, timer = 1, sz = 0;
  vector<vector<int>> adj, tree, BCC;
  vector<int> in, low, st, root;
  vector<array<int, 2>> brdgs;

  Bridges(int n): n(n) {};

  void build(auto& _adj) {
    adj = _adj;
    in.assign(n + 1, 0);
    low.assign(n + 1, 0);
    root.assign(n + 1, 0);

    for (int u = 1; u <= n; ++u)
      if (!in[u]) dfs(u);

    sz = BCC.size();
    tree.assign(n + 1, {});
    for (int u = 1; u <= n; u++)
      for (int v: adj[u])
        if (root[u] != root[v])
          tree[root[u]].push_back(root[v]);
  }

  void dfs(int u, int p = 0) {
    st.push_back(u);
    in[u] = low[u] = timer++;
    bool pFound = 0;
    for (int &v : adj[u]) {
      if(!pFound && v == p) {
        pFound = 1;
        continue;
      }

      if(!in[v]) dfs(v, u);
      low[u] = min(low[u], low[v]);
      if(low[v] > in[u]) brdgs.push_back({u, v});
    }

    if (low[u] == in[u]) {
      vector<int> c;
      while (st.back() != u)
        c.push_back(st.back()), st.pop_back();
      c.push_back(st.back()), st.pop_back();
      for (int v: c) root[v] = c.front();
      BCC.push_back(c);
    }
  }
};
```

Articulation Points

```
struct ArticPoints {
  // count: number of nodes in block cut tree
  int n, tim, count;
  vector<vector<int>> adj, BCC, tree;
  vector<int> st, in, low, isArt, id;

  // one indexed
  void build(int _n, vector<vector<int>>& graph) {
    n = _n;
    adj = graph;

    st.clear(), BCC.clear();
    in.assign(n + 1, 0);
    low.assign(n + 1, 0);
    isArt.assign(n + 1, false);
    for (int v = 1; v <= n; ++v)
      if (not in[v]) tim = 0, dfs(v);

    count = 0;
    tree.clear();
    tree.push_back({});
    id.assign(n + 1, 0);
    for (int v = 1; v <= n; ++v) {
      if (isArt[v]) {
        id[v] = ++count;
        tree.push_back({});
      }
    }

    for (vector<int>&comp : BCC) {
      int node = ++count;
      tree.push_back({});
      for (int u : comp) {
        if (isArt[u]) {
          tree[node].push_back(id[u]);
          tree[id[u]].push_back(node);
        } else id[u] = node;
      }
    }
  }

  void dfs(int v, int p = 0) {
    if(adj[v].empty()) return BCC.push_back({v});

    st.push_back(v);
    in[v] = low[v] = ++tim;
    for (int u : adj[v]) {
      if (u == p) continue;
      if (in[u]) {
        low[v] = min(low[v], in[u]);
        continue;
      }

      dfs(u, v);
      low[v] = min(low[v], low[u]);
      if (low[u] >= in[v]) {
        isArt[v] = (in[v] > 1) or (in[u] > 2);
        vector comp = { v };
        while (comp.back() != u) {
          comp.push_back(st.back());
          st.pop_back();
        }
        BCC.push_back(comp);
      }
    }
  }
};
```

2 - SAT

```
// Don't forget the build
struct Two2Sat {
  int n, timer = 1;
  vector<vector<int>> adj;
  vector<int> in, low, id, st, id_ans;
  vector<bool> stacked;

  // number of objects not the double
  Two2Sat(int count) {
    n = 2 * count;
    adj.assign(n, {});
  }
  int addVar() {
    adj.push_back({});
    adj.push_back({});
    n += 2;
    return n / 2 - 1;
  }

  bool solve() {
    in.assign(n, 0);
    low.assign(n, 0);
    id.assign(n, -1);
    stacked.assign(n, 0);
    id_ans.assign(n, -1);

    for (int u = 0; u < n; ++u)
      if (!in[u]) dfs(u);

    for (int i = 0; i < n - 1; i += 2)
      if(id[i] == id[i + 1]) return 0;
    return 1;
  }
  int neg(int node) { return node^1; }

  // 0-indexed
  void add_edge(int node_u, int node_v) {
    adj[neg(node_u)].push_back(node_v);
    adj[neg(node_v)].push_back(node_u);
  }

  void OR(int u, bool neg_u,int v, bool neg_v) { // {01,10,11}
    u = (u<<1)^neg_u, v = (v<<1)^neg_v;
    add_edge(u, v);
  }
  void XOR(int u, bool neg_u,int v, bool neg_v) { // {01, 10}
    u = (u<<1)^neg_u, v = (v<<1)^neg_v;
    add_edge(u, v);
    add_edge(neg(u), neg(v));
  }
  void XNOR(int u, bool neg_u,int v, bool neg_v) { // {00, 11}
    u = (u<<1)^neg_u, v = (v<<1)^neg_v;
    add_edge(neg(u), v);
    add_edge(u, neg(v));
  }
  void MUST(int u, bool neg_u) { // {1}
    u = (u<<1)^neg_u;
    add_edge(u, u);
  }

  // if u then must v
  void implies(int u, bool neg_u,int v, bool neg_v) {
    u = (u<<1)^neg_u, v = (v<<1)^neg_v;
    add_edge(neg(u), v);
  }

  // (1, 1, 1, 1, 1, 0) || (1, 1, 1, 1, 1, 1)
  void allExpectAtMostOne(vector<pair<int, int>> &v){
    OR(v[0].first, v[0].second, addVar(), 1);
    for(int i = 1; i < (int)v.size(); i++){
      int me = addVar();
      OR(v[i].first, v[i].second, me, 1);
      OR(v[i].first, v[i].second, me - 1, 0);
      OR(me - 1,0,  me, 1);
    }
  }

  void dfs(int u) {
    in[u] = low[u] = timer++;
    stacked[u] = 1;
    st.push_back(u);

    for (int& v : adj[u]) {
      if (!in[v]) dfs(v), low[u] = min(low[u], low[v]);
      else if (stacked[v]) low[u] = min(low[u], in[v]);
    }

    if (low[u] == in[u]) {
      int v = -1;
      int fir = -1;
      while (v != u) {
        v = st.back(), st.pop_back(), stacked[v] = 0;
        fir = (fir == -1 ? v : fir);
        id[v] = fir;
        if(~id[neg(v)]) id_ans[v] = 0;
        else id_ans[v] = 1;
      }
    }
  }

  // 0 indexed
  bool get_val(int idx) { return id_ans[idx * 2]; }
};
```

LCA (Euler)

```
struct LCA {
  int n;
  vector<pair<int, int>> tour;
  vector<int> depth, tin, par;
  SparseTable<pair<int, int>> sp;

  LCA(int sz) {
    n = sz + 1;
    tin.resize(n);
    depth.resize(n);
    par.resize(n);
    tour.reserve(n * 2);
    sp = SparseTable<pair<int, int>>(2 * n);
  }

  void dfs(int u, int p, int d, auto &adj) {
    tin[u] = (int) tour.size();
    depth[u] = d;
    par[u] = p;
    tour.emplace_back(d, u);
    for (int v: adj[u]) {
      if (v == p) continue;
      dfs(v, u, d + 1, adj);
      tour.emplace_back(d, u);
    }
  }

  void build(auto &adj) {
    dfs(1, 0, 0, adj);
    sp.build(tour);
  }

  int get(int u, int v) {
    return sp.query(min(tin[u], tin[v]), max(tin[u],
tin[v])).second;
  }

  int dist(int u, int v) {
    return depth[u] + depth[v] - 2 * depth[get(u, v)];
  }
};
```

HLD

```
vector<vector<int>> adj;
vector<int> par, depth, pos, sz, heavy, head, val;
int cnt = 0;

void dfs(int u, int p, int d = 0) {
    par[u] = p, depth[u] = d, sz[u] = 1;

    int mx = 0;
    for (int &v : adj[u]) {
        if (v == p) continue;
        dfs(v, u, d + 1);
        if (sz[v] > mx) heavy[u] = v, mx = sz[v];
        sz[u] += sz[v];
    }
}

void HLD(int u, int hd) {
    if (!u) return;
    head[u] = hd;

    pos[u] = cnt++;
    HLD(heavy[u], hd);
    for (int &v : adj[u])
        if (v != par[u] && v != heavy[u])
            HLD(v, v);
}

int query(int l, int r) {
    // query on segtree or somthing, [l, r] are included
    return;
}
```

```
// Don't forget to change the operation (res + ...)
int getPath(int u, int v) {
    int a = head[u], b = head[v];

    int res = 0;
    while (a != b) {
        if (depth[a] < depth[b]) swap(a, b), swap(u, v);
        res = res + query(pos[a], pos[u]);
        u = par[a], a = head[u];
    }

    if (pos[u] > pos[v]) swap(u, v);
    res = res + query(pos[u], pos[v]);
    return res;
}

int getSubtree(int u) {
    return query(pos[u], pos[u] + sz[u] - 1);
}

void updateNode(int u, int value) {
    // update pos[u] in the structure
    val[pos[u]] = value;
}

// one indexed
// call init(n), dfs(1, 0), HLD(1, 1); in main
void init(int n) {
    ++n, cnt = 0;
    adj.assign(n, {});
    par.assign(n, {});
    depth.assign(n, {});
    pos.assign(n, {});
    sz.assign(n, {});
    heavy.assign(n, {});
    head.assign(n, {});
    val.assign(n, {});
}
```

Segment Tree

```
struct Node {
    int ign = 0, val;

    Node() : val(ign) {};

    Node(int x) : val(x) {};

    void set(int x) {
        val = x;
    }
};

#define lNode (x * 2 + 1)
#define rNode (x * 2 + 2)
#define md (lx + (rx - lx) / 2)

struct Sagara {
    int n;
    vector<Node> node;

    Sagara(int sz) {
        n = 1;
        while (n < sz) n *= 2;
        node.assign(n * 2, Node());
    }

    Node merge(Node &l, Node &r) {
        Node res = Node();
        res.val = l.val + r.val;
        return res;
    }

    void build(vector<int> &v, int x, int lx, int rx) {
        if (rx - lx == 1) {
            if (lx < v.size()) node[x] = Node(v[lx]);
            return;
        }

        build(v, lNode, lx, md);
        build(v, rNode, md, rx);
        node[x] = merge(node[lNode], node[rNode]);
    }

    void build(vector<int> &v) { build(v, 0, 0, n); }

    void set(int &ind, ll &val, int x, int lx, int rx) {
        if (rx - lx == 1) return node[x].set(val);

        if (ind < md) set(ind, val, lNode, lx, md);
        else set(ind, val, rNode, md, rx);

        node[x] = merge(node[lNode], node[rNode]);
    }

    void set(int ind, ll val) { set(ind, val, 0, 0, n); }

    Node query(int &l, int &r, int x, int lx, int rx) {
        if (lx >= r || rx <= l) return Node();
        if (rx <= r && lx >= l) return node[x];

        Node L = query(l, r, lNode, lx, md);
        Node R = query(l, r, rNode, md, rx);

        return merge(L, R);
    }

    Node query(int l, int r) {
        return query(l, r, 0, 0, n);
    }
};
```

Lazy Segment Tree

```
struct Node {
    int ign = 0, lazy = 0, val = ign;
    bool isLazy = 0;

    Node() {}

    Node(ll x) : val(x) {}

    void add(int x, int lx, int rx) {
        val += x * (rx - lx);
        lazy += x;
        isLazy = 1;
    }
};

#define lNode (x * 2 + 1)
#define rNode (x * 2 + 2)
#define md (lx + (rx - lx) / 2)

struct Sagara {
    int n;
    vector<Node> node;

    Sagara(int sz) {
        n = 1;
        while (n < sz) n *= 2;
        node.assign(n * 2, Node());
    }

    Node merge(Node &l, Node &r) {
        Node res = Node();
        res.val = l.val + r.val;
        return res;
    }

    void propagate(int x, int lx, int rx) {
        if (rx - lx == 1 || !node[x].isLazy) return;

        node[lNode].add(node[x].lazy, lx, md);
        node[rNode].add(node[x].lazy, md, rx);

        node[x].isLazy = node[x].lazy = 0;
    }

    void update(int l, int r, ll val, int x, int lx, int rx) {
        propagate(x, lx, rx);

        if (lx >= r || rx <= l) return;
        if (lx >= l && rx <= r)
            return node[x].add(val, lx, rx);

        update(l, r, val, lNode, lx, md);
        update(l, r, val, rNode, md, rx);

        node[x] = merge(node[lNode], node[rNode]);
    }

    void update(int l, int r, ll val) { update(l, r, val, 0, 0, n); }

    Node query(int l, int r, int x, int lx, int rx) {
        propagate(x, lx, rx);

        if (lx >= l && rx <= r) return node[x];
        if (lx >= r || rx <= l) return Node();

        Node L = query(l, r, lNode, lx, md);
        Node R = query(l, r, rNode, md, rx);

        return merge(L, R);
    }

    Node query(int l, int r) {
        return query(l, r, 0, 0, n);
    }
};
```

Merge Sort Tree

```
struct Node {
    vector<int> v;
    Node() {};
    Node(int x) : v({x}) {};
};

#define lNode (x * 2 + 1)
#define rNode (x * 2 + 2)
#define md (lx + (rx - lx) / 2)

struct MergeSortSagara {
    vector<Node> node;
    int n;

    MergeSortSagara(int _n) {
        n = 1;
        while (n < _n) n <<= 1;
        node.assign(n * 2, Node());
    }

    Node merge(Node &l, Node &r) {
        Node res;

        int i = 0, j = 0;
        while (i < l.v.size() && j < r.v.size()) {
            if (l.v[i] < r.v[j]) res.v.push_back(l.v[i++]);
            else res.v.push_back(r.v[j++]);
        }

        while (i < l.v.size()) res.v.push_back(l.v[i++]);
        while (j < r.v.size()) res.v.push_back(r.v[j++]);
        return res;
    }

    void build(vector<int> &v, int x, int lx, int rx) {
        if (rx - lx == 1) {
            if (lx < v.size()) node[x] = Node(v[lx]);
            return;
        }

        build(v, lNode, lx, md);
        build(v, rNode, md, rx);
        node[x] = merge(node[lNode], node[rNode]);
    }

    void build(vector<int> &v) { build(v, 0, 0, n); }

    int query(int l, int r, int x, int lx, int rx, int val) {
        if (rx <= l || lx >= r) return 0;
        if (lx >= l && rx <= r) return calc(node[x], val);
        return query(l, r, lNode, lx, md, val) + query(l, r, rNode, md, rx, val);

    }

    int query(int l, int r, int val) {
        return query(l, r, 0, 0, n, val);
    }

    // change this function to match your need
    int calc(Node &no, int val) {
        return greater_than(no, val);
    }

    int less_than(Node &no, int val) {
        return lower_bound(no.v.begin(), no.v.end(), val) - no.v.begin();
    }

    int greater_than(Node &no, int val) {
        return no.v.size() - less_than(no, val) - equal(no, val);
    }

    int equal(Node &no, int val) {
        return upper_bound(no.v.begin(), no.v.end(), val) - lower_bound(no.v.begin(), no.v.end(), val);
    }
};
```

2D Segment Tree

```
struct Node {
    ll ign = 0, val;
    Node() : val(ign) {}
    Node(ll x) : val(x) {}
    Node operator+(Node &r) {
        Node res = Node();
        res.val = val + r.val;
        return res;
    }
};

#define lNode (x * 2 + 1)
#define rNode (x * 2 + 2)
#define md (lx + (rx - lx) / 2)

struct Sagara {
    int n;
    vector<Node> node;

    Sagara(int sz) {
        n = 1;
        while (n < sz) n *= 2;
        node.assign(n * 2, Node());
    }

    void set(int &ind, ll &val, int x, int lx, int rx) {
        if (rx - lx == 1) {
            node[x] = Node(val);
            return;
        }

        if (ind < md) set(ind, val, lNode, lx, md);
        else set(ind, val, rNode, md, rx);

        node[x] = node[lNode] + node[rNode];
    }

    void set(int ind, ll val) {set(ind, val, 0, 0, n);}
    Node get(int &i) {return node[n + i - 1];}

    Node query(int &l, int &r, int x, int lx, int rx) {
        if (lx >= r || rx <= l) return Node();
        if (rx <= r && lx >= l) return node[x];
        Node L = query(l, r, lNode, lx, md);
        Node R = query(l, r, rNode, md, rx);
        return L + R;
    }

    Node query(int l, int r) {
        return query(l, r, 0, 0, n);
    }
};

struct Sagara2D {
    vector<Sagara> node;
    int n;

    Sagara2D(int _n, int _m) {
        n = 1;
        while (n < _n) n <<= 1;
        node.assign(n * 2, Sagara(_m));
    }

    void set(int i, int j, int val, int x, int lx, int rx) {
        if (rx - lx == 1)
            return node[x].set(j, val);

        if (i < md) set(i, j, val, lNode, lx, md);
        else set(i, j, val, rNode, md, rx);

        Node L = node[lNode].get(j);
        Node R = node[rNode].get(j);

        node[x].set(j, (L + R).val);
    }

    void set(int i, int j, int val) {
        set(i, j, val, 0, 0, n);
    }

    // r and b are not included
    int query(int t, int b, int l, int r, int x, int lx, int rx) {
        if (rx <= t || lx >= b) return 0;
        if (lx >= t && rx <= b)
            return node[x].query(l, r).val;

        int L = query(t, b, l, r, lNode, lx, md);
        int R = query(t, b, l, r, rNode, md, rx);
        return L + R;
    }
    // top, bottom, right, left
    int query(int t, int b, int r, int l) {
        return query(t, b, r, l, 0, 0, n);
    }
};
```

PST

```
struct PST {
    // root[query_index] returns the root of the version of this query
    // L[x], R[x] are the id's of x left and right child
    vector<int> node, L, R, root;
    int n, curr;
    const int ign = 0; // change this accordigly

    PST(int _n, int q) {
        const int LOG = 30;
        n = _n;
        root.reserve(q + 10);
        L.resize(n * LOG);
        R.resize(n * LOG);
        node.resize(n * LOG);
        curr = 0;
    }

    int merge(int lf, int ri) {
        return lf + ri;
    }

    int init(int lx, int rx) {
        int id = curr++;
        if (rx - lx < 2) {
            node[id] = ign;
            return id;
        }

        int md = (lx + rx) / 2;
        L[id] = init(lx, md);
        R[id] = init(md, rx);

        node[id] = merge(node[L[id]], node[R[id]]);
        return id;
    }

    void init() {
        root.push_back(init(0, n));
    }

    int set(int ind, int val, int x, int lx, int rx) {
        if (ind >= rx || ind < lx) return x;

        int id = curr++;
        if (rx - lx < 2) {
            node[id] = val;
            return id;
        }

        int md = (rx + lx) / 2;
        L[id] = set(ind, val, L[x], lx, md);
        R[id] = set(ind, val, R[x], md, rx);

        node[id] = merge(node[L[id]], node[R[id]]);
        return id;
    }

    void set(int ind, int val, int version = -1) {
        if (~version)
            root.push_back(set(ind, val, root[version], 0, n));
        else
            root.push_back(set(ind, val, root.back(), 0, n)); // work on the latest version
    }

    int query(int l, int r, int x, int lx, int rx) {
        if (rx <= l || lx >= r) return ign;
        if (rx <= r && lx >= l) return node[x];
        int md = (lx + rx) / 2;
        return merge(query(l, r, L[x], lx, md), query(l, r, R[x], md, rx));
    }

    int query(int l, int r, int version = -1) {
        if (~version)
            return query(l, r, root[version], 0, n);
        else
            return query(l, r, root.back(), 0, n);
    }
};
```

Segment Tree Functions

```
// seg tree with hash remember to multiply with inv[l] in the query

// to get prev go to the right before the left

int next_greater(int l, int r, int val, int x, int lx, int rx) {
    if (lx >= r or l >= rx) return -1;
    if (node[x].mx <= val) return -1;
    if (rx - lx == 1) return lx;

    int ans = next_greater(l, r, val, lNode, lx, md);

    if (ans == -1)
        return next_greater(l, r, val, rNode, md, rx);
    return ans;
}

int next_smaller(int l, int r, int val, int x, int lx, int rx) {
    if (lx >= r or l >= rx) return -1;
    if (node[x].mn >= val) return -1;
    if (rx - lx == 1) return lx;

    int ans = next_smaller(l, r, val, lNode, lx, md);

    if (ans == -1)
        return next_smaller(l, r, val, rNode, md, rx);
    return ans;
}

int kth_one(int k, int x, int lx, int rx) {
    if (rx - lx == 1)
        return lx;
    if (k < node[lNode].sum)
        return find_kth_one(k, lNode, lx, md);
    return find_kth_one(k - node[lNode].sum, rNode, md, rx);
}

int kth_one(int k) {
    if(node[0].sum < k) return -1;
    return kth_one(k, 0, 0, n);
}
```

```
// max subarray sum

Node merge(Node &l, Node &r) {
    Node res = Node();
    res.ans = max({l.ans, r.ans, l.suff + r.pre});
    res.pre = max(l.pre, l.sum + r.pre);
    res.suff = max(r.suff, r.sum + l.suff);
    res.sum = r.sum + l.sum;
    return res;
}
```

Iterative Segment Tree

```
const int N = 2e5 + 1;  // limit for array size
int t[2 * N];
int m;  // array size

void build(vector<int> &v) {  // build the tree
    for (int i = 0; i < m; ++i) t[i + m] = v[i];
    for (int i = m - 1; i > 0; --i)
        t[i] = t[i << 1] + t[i << 1 | 1];
}

void modify(int p, int value) {
    for (t[p += m] = value; p > 1; p >>= 1)
        t[p >> 1] = t[p] + t[p ^ 1];
}

int query(int l, int r) {  // sum on interval [l, r)
    int res = 0;
    for (l += m, r += m; l < r; l >>= 1, r >>= 1) {
        if (l % 2) res += t[l++];
        if (r % 2) res += t[--r];
    }
    return res;
}
```

Sparse Table

```
template<class T>
struct SparseTable {
  // change mxLog
  static const int mxLog = 21;
  vector<array<T, mxLog>> table;
  vector<int> lg;
  int n;

  SparseTable(int sz) {
    n = sz;
    table.resize(n + 1);
    lg.resize(n + 1);
    for (int i = 0; i <= n; ++i) lg[i] = __lg(i);
  }

  void build(vector<T> &v) {
    for (int i = 0; i < n; ++i) table[i][0] = v[i];
    for (int j = 1; j < mxLog; ++j)
      for (int i = 0; i + (1 << j) - 1 < n; ++i)
        table[i][j] = merge(table[i][j - 1], table[i + (1 << (j - 1))][j - 1]);
  }

  T merge(T &l, T &r) {
    return min(l, r);
  }

  T query(int l, int r) {
    int j = lg[r - l + 1];
    return merge(table[l][j], table[r - (1 << j) + 1][j]);
  }
};
```

BIT

```
struct BIT {
  int n;
  vector<int> t;

  BIT(int n): n(n), t(n + 1) {}

  void add(int i, int x) {
    for(; i < n; i |= i + 1) t[i] += x;
  }

  int get(int i) {
    int res = 0;
    for(; i >= 0; i = (i & i + 1) - 1) res += t[i];
    return res;
  }

  int get(int l, int r) { return get(r) - get(l - 1); }
};
```

Ordered Set

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;

template<typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;

void myErase(ordered_set<int> &t, int v) {
  int rank = t.order_of_key(v);
  ordered_set<int>::iterator it = t.find_by_order(rank);
  t.erase(it);
}
```

Offline 2D BIT

```
template <typename T> class OfflineBIT2D {
  const int n;
  vector<vector<int>> vals;
  vector<vector<T>> bit;

  /** @return the first index i such that v[i] <= x */
  int ind(const vector<int> &v, int x) {
    return upper_bound(begin(v), end(v), x) - begin(v) - 1;
  }

public:
  OfflineBIT2D(int n, vector<array<int, 2>> &todo) : n(n), vals(n + 1), bit(n + 1) {
    sort(begin(todo), end(todo),
      [](const array<int, 2> &a, const array<int, 2> &b) -> bool {
        return a[1] < b[1];
      });

    for (int i = 1; i <= n; i++) { vals[i].push_back(0); }
    for (auto [r, c] : todo) {
      r++, c++;
      for (; r <= n; r += r & -r) {
        if (vals[r].back() != c) { vals[r].push_back(c); }
      }
    }
    for (int i = 1; i <= n; i++) { bit[i].resize(vals[i].size()); }
  }
```

```
    /** adds val to the point (r, c) */
    void add(int r, int c, T val) {
        r++, c++;
        for (; r <= n; r += r & -r) {
            int i = ind(vals[r], c);
            for (; i < bit[r].size(); i += i & -i) { bit[r][i] += val; }
        }
    }

    /** @returns sum of points with row in [0, r] and column in [0, c] */
    T rect_sum(int r, int c) {
        r++, c++;
        T sum = 0;
        for (; r > 0; r -= r & -r) {
            int i = ind(vals[r], c);
            for (; i > 0; i -= i & -i) { sum += bit[r][i]; }
        }
        return sum;
    }

    /** @returns sum of points with row in [r1, r2] and column in [c1, c2] */
    T rect_sum(int r1, int c1, int r2, int c2) {
        return rect_sum(r2, c2) - rect_sum(r2, c1 - 1) - rect_sum(r1 - 1, c2) + rect_sum(r1 - 1, c1 - 1);
    }
};
```

Convex Hull

```
struct Line {
    int m, b;
    mutable function<const Line *()> succ;

    bool operator<(const Line& other) const {
        return m < other.m;
    }

    bool operator<(const int& x) const {
        const Line *s = succ();
        if (not s) return false;
        return b - s->b < (s->m - m) * x;
    }
};

struct HullDynamic: multiset<Line, less<>> {
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return false;
            return y->m == z->m and y->b <= z->b;
        }

        auto x = prev(y);
        if (z == end())
            return y->m == x->m and y->b <= x->b;

        return (long double) (x->b - y->b) * (z->m - y->m) >= (long double) (y->b - z->b) * (y->m - x->m);
    }

    void insert_line(int m, int b) {
        // for minimum
        // m *= -1, b *= -1;
        auto y = insert({ m, b });
        y->succ = [=] {
            return next(y) == end() ? 0 : &*next(y);
        };

        if (bad(y)) return void(erase(y));

        while (next(y) != end() and bad(next(y)))
            erase(next(y));

        while (y != begin() and bad(prev(y)))
            erase(prev(y));
    }

    int eval(int x) {
        auto l = *lower_bound(x);
        // for minimum
        // return -(l.m * x + l.b);
        return l.m * x + l.b;
    }
};
```

Matrix

```
template<class T>
struct Matrix {
  int n, m;
  vector<vector<T>> a;

  Matrix(int _n, int _m) {
    n = _n, m = _m;
    a.assign(n, vector<T>(m));
  }

  // Don't forget mod
  Matrix operator *(const Matrix &b) {
    int r = n, c = b.m, k = m;
    Matrix res(r, c);
    for(int i = 0; i < r; ++i)
      for(int j = 0; j < c; ++j)
        for(int o = 0; o < k; ++o)
          res.a[i][j] += a[i][o] * b.a[o][j];
    return res;
  }


  friend Matrix power(Matrix mat, ll p) {
    Matrix ret(mat.n, mat.n);
    for (int i = 0; i < mat.n; ++i)
      ret.a[i][i] = 1;

    while (p) {
      if (p & 1) ret = ret * mat;
      mat = mat * mat, p >>= 1;
    }
    return ret;
  }
};
```

BIT 2D

```
// this supports update rect and get a single cell
struct BIT2D {
  int n, m;
  vector<vector<int>> bit;

  BIT2D(int _n, int _m) {
    n = _n + 5, m = _m + 5;
    bit.assign(n, vector<int>(m));
  }

  void updateY(int x, int ind, ll val) {
    for (; ind < m; ind |= (ind + 1))
      bit[x][ind] += val;
  }

  void update(int x, int y, int val) {
    for (; x < n; x |= (x + 1))
      updateY(x, y, val);
  }

  // pass it left, right, top, bottom, value
  void updateRect(int l, int r, int t, int b, ll val) {
    update(t, l, val);
    update(t, r + 1, -val);
    update(b + 1, l, -val);
    update(b + 1, r + 1, val);
  }

  int getY(int x, int ind) {
    int v = 0;
    for (; ind >= 0; ind = (ind & (ind + 1)) - 1)
      v += bit[x][ind];
    return v;
  }

  int get(int x, int y) {
    int v = 0;
    for (; x >= 0; x = (x & (x + 1)) - 1)
      v += getY(x, y);
    return v;
  }
};
```

MO Algorithm

```
struct Query { int l, r, ind; };

struct MO {
   int n, sq, l = 0, r = 0;
   ll curr = 0;
   vector<int> v;

   MO(vector<int> &v): v(v) {
      n = v.size();
      sq = sqrt(n) + 1;
   }

   void move(int &lq, int &rq) {
      while (r < rq) add(++r);
      while (l < lq) del(l++);
      while (l > lq) add(--l);
      while (r > rq) del(r--);
   }

   void solve(vector<Query> &qu) {
      sort(qu.begin(), qu.end(),
         [&](auto &lf, auto ri) {
            if (lf.l / sq == ri.l / sq) return lf.r < ri.r;
            return lf.l / sq < ri.l / sq;
         });

      l = qu[0].l, r = qu[0].l;
      add(l);

      vector<ll> res(qu.size());
      for (auto &[lq, rq, iq]: qu) {
         move(lq, rq);
         res[iq] = curr;
      }

      for (ll &i: res) cout << i << endl;
   }
};
```

MO with Updates

```
const int N = 2e5 + 5, SQ = 3500;

struct query {
   int l, r, idx, uIdx;
   bool operator<(const query &other) const {
      if (l / SQ != other.l / SQ)
         return l / SQ < other.l / SQ;
      if (r / SQ != other.r / SQ)
         return r / SQ < other.r / SQ;
      return uIdx < other.uIdx;
   }
};

struct update {
   int idx, val, old;
};

int a[N], ans[N], frq[N], cnt[N];

void add(int idx) {}
void remove(int idx) {}

void upd(update &u, int l, int r) {
   if (u.idx >= l && u.idx <= r)remove(u.idx);
   a[u.idx] = u.val;
   if (u.idx >= l && u.idx <= r)add(u.idx);
}

void cancel(update &u, int l, int r) {
   if (u.idx >= l && u.idx <= r)remove(u.idx);
   a[u.idx] = u.old;
   if (u.idx >= l && u.idx <= r)add(u.idx);
}

void mo(vector<query> &v, vector<update> &u) {
   int l = 0, r = -1;
   int cur = 0;
   for (auto &q: v) {
      while (cur < q.uIdx)upd(u[cur++], l, r);
      while (cur > q.uIdx)cancel(u[--cur], l, r);
      while (r < q.r)add(++r);
      while (l > q.l)add(--l);
      while (r > q.r)remove(r--);
      while (l < q.l)remove(l++);
      while (cnt[ans[q.idx]])ans[q.idx]++;
   }
}
```

Bitmasks

```
int count_numbers_has_ith_bit(int n, int k) {
  ++n;
  int d = ( 1LL << (k + 1) ), p = (1LL << k);
  return n / d * p + max(0LL, n % d - p);
}

int highest_bit(int x) {
  int t = 63 - __builtin_clzll(x);
  return (1LL << t);
}

int lowest_bit(int x) {
  int t = __builtin_ctzll(x);
  return (1LL << t);
}

// loop through submasks

for (int j = mask; j; j = (j - 1) & mask) {

}

int xor_range(int n) { // from 1 to n
  if (n % 4 == 0) return n;
  if (n % 4 == 1) return 1;
  if (n % 4 == 2) return n + 1;
  return 0;
}

int odd_xor(int n) {
  if (n % 2 == 0)
    return ((xor_range(n)) ^ (2LL * xor_range(n / 2LL)));
  else
    return ((xor_range(n)) ^ (2LL * xor_range((n - 1LL) / 2LL)));
}

int odd_xor_range(int l, int r) {
  return odd_xor(l - 1) ^ odd_xor(r);
}

int even_xor_range(int l, int r) {
  int xor_r = 2LL * xor_range(r / 2LL);
  int xor_l = 2LL * xor_range((l - 1LL) / 2LL);
  return (xor_l ^ xor_r);
}
```

XOR of all subarrays

```
int calcSubArrayXORSum(vector<int> &arr) {
  int n = arr.size();
  int sum = 0;
  int multiplier = 1;
  for (int i = 0; i < 30; i++) {
    int oddCount = 0;
    bool isOdd = 0;
    for (int j = 0; j < n; j++) {
      if ((arr[j] & (1 << i)) > 0)
        isOdd = (!isOdd);
      if (isOdd)
        oddCount++;
    }
    for (int j = 0; j < n; j++) {
      sum += (multiplier * oddCount);
      if ((arr[j] & (1 << i)) > 0)
        oddCount = (n - j - oddCount);
    }
    multiplier *= 2;
  }
  return sum;
}
```

SOS DP

```
const int B = 20;
const int M = 1 << B;

// subset contribute to its superset
void forward(vector<int> &dp) {
  for (int i = 0; i < B; ++i)
    for (int m = 0; m < M; ++m)
      if (m & (1 << i))
        dp[m] += dp[m ^ (1 << i)];
}

// superset contribute to its subset
void forwardRev(vector<int> &dp) {
  for (int i = 0; i < B; ++i)
    for (int m = M - 1; ~m; --m)
      if (m & (1 << i))
        dp[m ^ (1 << i)] += dp[m];
}

// remove subset contribution from superset
void backward(vector<int> &dp) {
  for (int i = 0; i < B; ++i)
    for (int m = M - 1; ~m; --m)
      if (m & (1 << i))
        dp[m] -= dp[m ^ (1 << i)];
}

// remove superset contribution from subset
void backwardRev(vector<int> &dp) {
  for (int i = 0; i < B; ++i)
    for (int m = 0; m < M; ++m)
      if (m & (1 << i))
        dp[m ^ (1 << i)] -= dp[m];
}
```

XOR of all subarrays multiplied by length

```
int calcSubArrayXORSum(vector<int> arr) {
  int n = arr.size();
  int sum = 0;
  int multiplier = 1;
  for (int i = 0; i < 30; i++) {
    int oddCount = 0;
    bool isOdd = 0;
    int tot = 0;
    for (int j = 0; j < n; j++) {
      if ((arr[j] & (1 << i)) > 0)
        isOdd = (!isOdd);
      if (isOdd) {
        oddCount++;
        tot += j + 1;
      }
    }
    for (int j = 0; j < n; j++) {
      sum += ((multiplier % mod) * (tot % mod)) %
mod;
      sum %= mod;
      if ((arr[j] & (1 << i)) > 0) {
        oddCount = (n - j - oddCount);
        tot = (n - j) * (n - j + 1) / 2 - tot;
        tot -= oddCount;
      } else
        tot -= oddCount;
    }
    multiplier *= 2;
  }
  return sum;
}
```

Suffix Array

```
struct SuffixArray {
  int n;
  vector<int> suff, lcp, c;

  SuffixArray(int sz) {
    n = sz + 1;
    suff.resize(n);
    lcp.resize(n);
    c.resize(n);
  }

  void countingSort(vector<int> &p) {
    vector<int> cnt(n), pos(n), newP(n);
    for (int i: c) cnt[i]++;


    for (int i = 1; i < n; ++i)
      pos[i] = pos[i - 1] + cnt[i - 1];


    for (int i: p)
      newP[pos[c[i]]++] = i;
    swap(p, newP);
  }

  void build(string &s) {
    s += '$';
    vector<pair<char, int>> a(n);

    for (int i = 0; i < n; ++i)
      a[i] = {s[i], i};

    sort(a.begin(), a.end());

    vector<int> p(n);
    for (int i = 0; i < n; ++i)
      p[i] = a[i].second;

    for (int i = 1; i < n; ++i)
      c[a[i].second] = c[a[i - 1].second] + (a[i].first !=
a[i - 1].first);



    int k = 0;
    while ((1 << k) < n) {
      int bit = 1 << k;

      for (int i = 0; i < n; ++i)
        p[i] = (p[i] - bit + n) % n;

      countingSort(p);

      vector<int> newC(n);
      for (int i = 1; i < n; ++i) {
        int currL = p[i], currR = (p[i] + bit) % n;
        int preL = p[i - 1], preR = (p[i - 1] + bit) % n;
        bool add = (c[currL] != c[preL]) || (c[currR] !=
c[preR]);
        newC[p[i]] = newC[p[i - 1]] + add;
      }

      c = newC;
      ++k;
    }

    suff = p;

    // Build LCP
    k = 0;
    for (int i = 0; i < n - 1; ++i) {
      int pi = c[i];
      int j = p[pi - 1];
      while (s[i + k] == s[j + k])
        ++k;
      lcp[pi] = k;
      k = max(int(0), k - 1);
    }
  }
};
```

Z Algorithm

```
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) {
    z[i] = (i < r) * min(r - i, z[i - l]);
    while (i + z[i] < n && s[i + z[i]] == s[z[i]]) z[i]++;
    if (i + z[i] > r) r = i + z[i], l = i;
  }
  return z;
}
```

Hashing

```
mt19937
rng(chrono::steady_clock::now().time_since_epoch()
.count());
#define getrand(l, r) uniform_int_distribution<long
long>(l, r)(rng)

struct Hash {
    // everything is zero indexed
    int mod, base, st, N;
    vector<int> pw, inv;

    // st is the start char, _N is the max size of the string
    Hash(int _st = 'a', int _N = 1e6) {
        st = _st - 1;
        N = _N + 1;
        pw.resize(N);
        inv.resize(N);
        gen();
        pre();
    }

    void gen() {
        auto check = [](int x) {
            for (int i = 2; i * i <= x; ++i)
                if (!(x % i)) return false;
            return true;
        };
        mod = getrand(1e8, 2e9);
        base = getrand(30, 120);
        while (!check(mod))--mod;
    }

    void pre() {
        int inv_pw = inverse(base, mod);
        pw[0] = inv[0] = 1;
        for (int i = 1; i < N; ++i) {
            pw[i] = mul(pw[i - 1], base, mod);
            inv[i] = mul(inv[i - 1], inv_pw, mod);
        }
    }
    int push_back(int h, char a, int len) {
        return add(h, mul(a - st, pw[len], mod), mod);
    }

    int push_front(int h, char a) {
        return add(a - st, mul(h, base, mod), mod);
    }
    int concat(int l, int r, int szLeft) {
        return add(l, mul(r, pw[szLeft], mod), mod);
    }
    vector<int> build(string &s) {
        int sz = s.size();
        vector<int> res(sz);
        res[0] = s[0] - st;
        for (int i = 1; i < sz; ++i)
            res[i] = push_back(res[i - 1], s[i], i);
        return res;
    }

    // [l, r] l & r are included
    int getSubstring(int l, int r, vector<int> &hash) {
        int res = hash[r];
        if (l) res = add(res, -hash[l - 1], mod);
        return mul(res, inv[l], mod);
    }

    bool is_palindrome(int l, int r, vector<int> &hsh,
vector<int> &hshRev) {
        int rev_l = hsh.size() - r - 1;
        int rev_r = hsh.size() - l - 1;
        return getSubstring(l, r, hsh) ==
getSubstring(rev_l, rev_r, hshRev);
    }

    // get Hash of path on tree
    void dfs(int u, int p, int l = 0) {
        hsh[u] = h.push_back(hsh[p], c[u], l);
        hshRev[u] = h.push_front(hshRev[p], c[u]);
        lvl[u] = l;
        par[u] = p;

        for (int &v: adj[u])
            if (v != p) dfs(v, u, l + 1);
    }

    int getTreePath(int u, int v, int lc) {
        int u_lc = add(hshRev[u], -mul(hshRev[lc],
pw[lvl[u] - lvl[lc]], mod), mod);

        int lc_v = add(hsh[v], -hsh[par[lc]], mod);
        lc_v = mul(lc_v, inv[lvl[lc]], mod);

        return concat(u_lc, lc_v, lvl[u] - lvl[lc]);
    };
};
```

Aho-Corasick

```
// Patterns must be distinct
struct Aho {
  int N, P, A, st;
  vector<vector<int>> nxt, out;
  vector<int> link, out_link;

  Aho(int A, int st): N(0), P(0), A(A), st(st) {node();}
  int node() {
    nxt.emplace_back(A, 0);
    link.emplace_back(0);
    out_link.emplace_back(0);
    out.emplace_back(0);
    return N++;
  }

  int get (char c) {return c - st;}

  int insert(const string &p) {
    int u = 0;
    for (auto &c: p) {
      if (!nxt[u][get(c)]) nxt[u][get(c)] = node();
      u = nxt[u][get(c)];
    }

    out[u].push_back(P);
    return P++;
  }

  void build() {
    queue <int> q;
    for (q.push(0); !q.empty();) {
      int u = q.front(); q.pop();
      for (int c = 0; c < A; ++c) {
        int v = nxt[u][c];
        if (!v) nxt[u][c] = nxt[link[u]][c];
        else {
          link[v] = u ? nxt[link[u]][c] : 0;
          out_link[v] = out[link[v]].empty() ? out_link[link[v]]
: link[v];
          q.push(v);
        }
      }
    }
  }

  int advance (int u, char c) {
    while (u && !nxt[u][get(c)]) u = link[u];
    u = nxt[u][get(c)];
    return u;
  }
};
```

Trie

```
const int B = 31;
struct Trie {
  struct Node {
    array<int, 2> nxt;
    int pref = 0, end = 0;
    Node() {nxt.fill(-1);}
  };

  vector<Node> t;
  Trie(): t({ Node() }) {}

  void add(int x) {
    int v = 0;
    for(int i = B - 1; ~i; --i) {
      int c = x >> i & 1;
      if(t[v].nxt[c] == -1) {
        t[v].nxt[c] = t.size();
        t.emplace_back();
      }
      v = t[v].nxt[c];
      t[v].pref++;
    }
    t[v].end++;
  }

  int maxXor(int x) {
    int ans = 0, v = 0;
    for(int i = B - 1; ~i; --i) {
      int z = t[v].nxt[0], o = t[v].nxt[1];
      if(x >> i & 1) {
        if(~z && t[z].pref) v = z, ans += 1ll << i;
        else v = o;
      }
      else {
        if(~o && t[o].pref) v = o, ans += 1ll << i;
        else v = z;
      }
    }

    return ans;
  }
};
```

Manacher

```
struct manacher {
  string s;
  vector<int> p;

  // the longest odd palindrome centered at i p[2*i+1]
  // the longest even palindrome centered at i-1, i
p[2*i]
  manacher(string& in) {
    s = manacher_string(in);
    int n = s.size();
    p.assign(n, 0);
    for (int i = 0, l = 0, r = -1, k; i < n; ++i) {
      if (i > r) k = 1;
      else  k = min(p[l + (r - i)], r - i) + 1;
      while (i - k >= 0 && i + k < n && s[i - k] == s[i + k])
        ++k;

      if (i - k == -1 || i + k == n || s[i - k] != s[i + k])
        --k;
      p[i] = k;
      if (i + k > r)
        l = i - k, r = i + k;
    }
  }

  string manacher_string(string& in) {
    int n = in.size();
    string t(2 * n + 1, '#');
    for (int i = 1, j = 0; i < 2 * n + 1; i += 2, ++j)
      t[i] = in[j];
    return t;
  }

  bool is_palindrome(int l, int r) {
    int L = 2 * l + 1;
    int R = 2 * r + 1;
    int mid = (L + R) / 2;
    return p[mid] >= mid - L;
  }
};
```

Min String Rotation

```
int min_cyc(string s) {
  int p = 0;
  s += s;
  vector<int> f(s.size(), -1);
  for (int l = 1, r = 1; r < s.size(); ++r) {
    for (l = f[r -p - 1]; l != -1 && s[p + l + 1] != s[r]; l = f[l])
      if (s[l + p + 1] > s[r])
        p = r - l - 1;

    if (l == -1 && s[p + l + 1] != s[r]) {
      if(s[p + l + 1] > s[r]) p = r;
      f[r - p] = -1;
    }

    else f[r - p] = l + 1;
  }
  return p;
}
```

KMP

```
vector<int> pi_function(string& s) {
  int n = s.size();
  vector<int> pi(n);
  for (int i = 1, j = 0; i < n; ++i, j = pi[i - 1]) {
    while (j && s[i] != s[j]) j = pi[j - 1];
    if (s[i] == s[j]) pi[i] = j + 1;
  }
  return pi;
}
auto compute_automaton(string &s) {
  s += '#';
  int n = s.size();
  vector<int> pi = pi_function(s);
  auto nxt = vector(n, vector<int>(26));

  for (int i = 0; i < n; i++) {
    for (int c = 0; c < 26; c++) {
      if (i > 0 && 'a' + c != s[i])
        nxt[i][c] = nxt[pi[i-1]][c];
      else
        nxt[i][c] = i + ('a' + c == s[i]);
    }
  }

  return nxt;
}
```

Arithmetic Progression

```
pair<vector<int>, vector<int>> build(vector<int> &v) {
    int n = v.size();
    vector<int> suff(n + 2), s(n + 2);
    for (int i = n - 1; i >= 0; --i) {
        suff[i] = suff[i + 1] + v[i];
        s[i] = s[i + 1] + suff[i];
    }

    return {suff, s};
}

int get(int l, int r, vector<int> &suff, vector<int> &s) {
    int res = s[l] - s[r + 1];
    res -= (r - l + 1) * suff[r + 1];
    return res;
}
```

Merge Segments

```
void merge(vector<pair<int, int>> &vc) {
    sort(vc.begin(), vc.end());
    vector<int> start, end;
    int st = vc[0].first, en = vc[0].second;
    for (int i = 1; i < vc.size(); i++) {
        if (vc[i].first <= en) en = max(en, vc[i].second);
        else {
            start.push_back(st);
            end.push_back(en);
            st = vc[i].first;
            en = vc[i].second;
        }
    }
    start.push_back(st);
    end.push_back(en);
}
```

Number of mod in range

```
// count number of y such that y%n = x  (0 -> l)

int f(int l, int n, int x) {
    int cnt = (l / n) + (l % n >= x);
    return cnt;
}
```

Kadane on Matrix

```
int kadane(vector<int> &v) {
    int s = 0, mx = -oo;
    for (int &i: v) {
        s += i;
        mx = max(s, mx);
        s = max(0ll, s);
    }
    return mx;
}

int maxSubmatrixSum(vector<vector<int>> &A) {
    int r = A.size();
    int c = A[0].size();
    int **pre = new int *[r];

    for (int i = 0; i < r; i++) {
        pre[i] = new int;
        for (int j = 0; j < c; j++)
            pre[i][j] = 0;
    }

    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            pre[i][j] = A[i][j];
            if (i)
                pre[i][j] += pre[i][j - 1];
        }
    }

    int maxSum = -oo;
    for (int i = 0; i < c; i++) {
        for (int j = i; j < c; j++) {
            vector<int> v;
            for (int k = 0; k < r; k++) {
                int el = pre[k][j];
                if (i) el -= pre[k][i - 1];
                v.push_back(el);
            }
            maxSum = max(maxSum, kadane(v));
        }
    }

    return maxSum;
}
```

2D Prefix Sum

```
// g is one indexed!!!
vector<vector<int>> g(n+1, vector<int>(m+1));

for(int i = 1; i <= n; i++)
  for(int j = 1; j <= m; j++)
    g[i][j] = g[i][j-1] + mat[i-1][j-1];

for(int j = 1; j <= m; j++)
  for(int i = 1; i <= n; i++)
    g[i][j] += g[i-1][j];

while(q--) {
  int a, b, c, d; cin >> a >> b >> c >> d;
  cout << g[c][d] + g[a - 1][b - 1] - g[a - 1][d] - g[c][b - 1]
<< endl;
}
```

Montonic Stack

```
vector<int> nextGreater(vector<int> &v) {
  int n = v.size();
  vector<int> res(n, n);
  stack<int> st;
  for (int i = 0; i < n; ++i) {
    if (st.empty() || v[i] <= v[st.top()]) st.push(i);
    else {
      res[st.top()] = i;
      st.pop(), --i;
    }
  }
  return res;
}
vector<int> prevGreater(vector<int> &v) {
  int n = v.size();
  vector<int> res(n, -1);
  stack<int> st;
  for (int i = n - 1; i >= 0; --i) {
    // you may need to remove the equal
    if (st.empty() || v[i] <= v[st.top()]) st.push(i);
    else {
      res[st.top()] = i;
      st.pop(), ++i;
    }
  }
  return res;
```

STL Compare

```
struct compare {
  // right is the top -> pq
  // left is the begin -> set, map
  // return false if equal
  bool operator()(# a, # b) const {

  }
};
```

Minimum Swaps

```
int cost(vector<int>& from, vector<int>& to) {
  int n = int(from.size());

  vector<int, int> mp;
  ordered_set<int> st;
  for (int i = 0; i < n; ++i) {
    st.insert(i);
    mp[from[i]] = i;
  }

  int ret = 0;
  for (int i = 0; i < n; ++i) {
    ret += st.order_of_key(mp[to[i]]);
    st.erase(mp[to[i]]);
  }

  return ret;
}
```

Mod Operations & Comb

```
// make sure a & b < mod
int add(int a, int b, int mod) {
  ll res = (ll) a + b;
  if (res >= mod) res -= mod;
  if (res < 0) res += mod;
  return res;
}

int mul(int a, int b, int mod) {
  return (1LL * a * b) % mod;
}

int power(int a, int b, int mod) {
  int ret = 1;
  while (b) {
    if (b & 1) ret = 1ll * ret * a % mod;
    a = 1ll * a * a % mod, b >>= 1;
  }
  return ret;
}

int inverse(int b, int mod) {
  return power(b, mod - 2, mod);
}

struct Comb {
  vector<int> fact; // pre process fact inv too :)
  Comb(int n) {
    fact.assign(n + 5, 1);
    for (int i = 1; i <= n; ++i) fact[i] = mul(i, fact[i - 1]);
  }

  int nPr(int n, int r) {
    return n < r ? 0 : mul(fact[n], inverse(fact[n - r]));
  }

  int nCr(int n, int r) {
    return mul(nPr(n, r), inverse(fact[r]));
  }
};
```

Linear Sieve

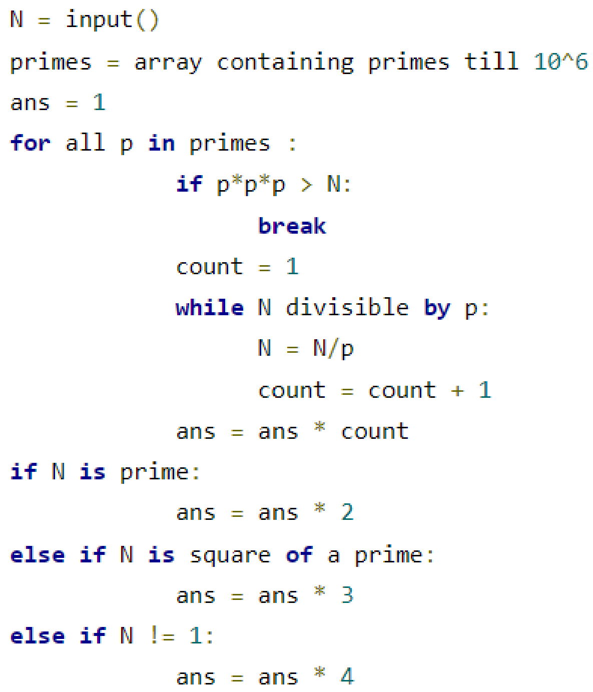
```
// pr is all the primes, low[x] is the lowest prime of x
vector<int> pr, low;
void Sieve(int n) {
  low.assign(n + 1, 0);
  for (int i = 2; i <= n; ++i) {
    if (!low[i]) {
      low[i] = i;
      pr.push_back(i);
    }

    for (int &j: pr) {
      if (j > low[i] || i * j > n) break;
      low[j * i] = j;
    }
  }
}
```

Segmented Sieve

```
vector<bool> segmentedSieve(int L, int R) {
  // generate all primes up to sqrt(R)
  int lim = sqrtl(R);
  vector<bool> mark(lim + 1, false);
  vector<int> primes;
  for (int i = 2; i <= lim; ++i) {
    if (!mark[i]) {
      primes.emplace_back(i);
      for (int j = i * i; j <= lim; j += i)
        mark[j] = true;
    }
  }
  vector<bool> isPrime(R - L + 1, true);
  for (int i: primes)
    for (int j = max(i * i, (L + i - 1) / i * i); j <= R; j += i)
      isPrime[j - L] = false;
  if (L == 1)
    isPrime[0] = false;
  return isPrime;
}
```

Count Divisors up to 1e18



```
N = input()
primes = array containing primes till 10^6
ans = 1
for all p in primes :
        if p*p*p > N:
            break
        count = 1
        while N divisible by p:
            N = N/p
            count = count + 1
        ans = ans * count
if N is prime:
        ans = ans * 2
else if N is square of a prime:
        ans = ans * 3
else if N != 1:
        ans = ans * 4
```

Int128

```
__int128 read() {
  __int128 x = 0, f = 1;
  char ch = getchar();
  while (ch < '0' || ch > '9') {
    if (ch == '-') f = -1;
    ch = getchar();
  }
  while (ch >= '0' && ch <= '9') {
    x = x * 10 + ch - '0';
    ch = getchar();
  }
  return x * f;
}

void print(__int128 x) {
  if (x < 0) {
    putchar('-');
    x = -x;
  }
  if (x > 9) print(x / 10);
  putchar(x % 10 + '0');
}
```

Prime Tester

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

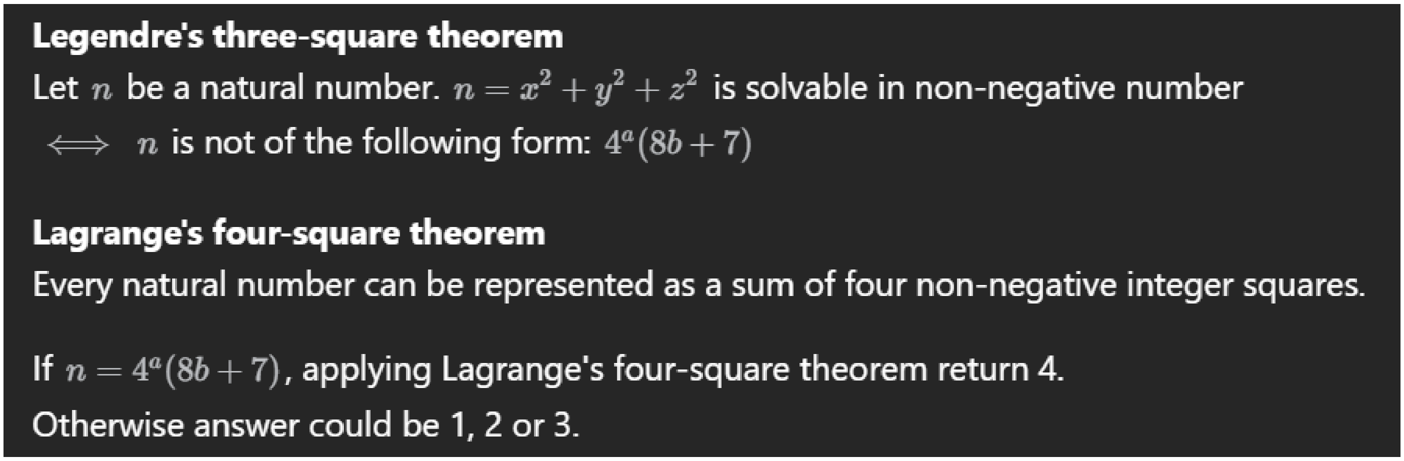
Sieve up to 1e9

```
vector<int> sieve(const int N, const int Q = 17, const int L = 1 << 15) {
  static const int rs[] = {1, 7, 11, 13, 17, 19, 23, 29};
  struct P {
    P(int p) : p(p) {}
    int p;
    int pos[8];
  };
  auto approx_prime_count = [](const int N) -> int {
    return N > 60184 ? N / (log(N) - 1.1)
            : max(1., N / (log(N) - 1.11)) + 1;
  };
  const int v = sqrt(N), vv = sqrt(v);
  vector<bool> isp(v + 1, true);
  for (int i = 2; i <= vv; ++i)
    if (isp[i]) {
      for (int j = i * i; j <= v; j += i) isp[j] = false;
    }
  const int rsize = approx_prime_count(N + 30);
  vector<int> primes = {2, 3, 5};
  int psize = 3;
  primes.resize(rsize);
  vector<P> sprimes;
  size_t pbeg = 0;
  int prod = 1;
  for (int p = 7; p <= v; ++p) {
    if (!isp[p]) continue;
    if (p <= Q) prod *= p, ++pbeg, primes[psize++] = p;
    auto pp = P(p);
    for (int t = 0; t < 8; ++t) {
      int j = (p <= Q) ? p : p * p;
      while (j % 30 != rs[t]) j += p << 1;
      pp.pos[t] = j / 30;
    }
    sprimes.push_back(pp);
  }
  vector<unsigned char> pre(prod, 0xFF);
  for (size_t pi = 0; pi < pbeg; ++pi) {
    auto pp = sprimes[pi];
    const int p = pp.p;
    for (int t = 0; t < 8; ++t) {
      const unsigned char m = ~(1 << t);
      for (int i = pp.pos[t]; i < prod; i += p) pre[i] &= m;
    }
  }
  const int block_size = (L + prod - 1) / prod * prod;
  vector<unsigned char> block(block_size);
  unsigned char *pblock = block.data();
  const int M = (N + 29) / 30;
  for (int beg = 0; beg < M; beg += block_size, pblock -= block_size) {
    int end = min(M, beg + block_size);
    for (int i = beg; i < end; i += prod)
      copy(pre.begin(), pre.end(), pblock + i);
    if (beg == 0) pblock[0] &= 0xFE;
    for (size_t pi = pbeg; pi < sprimes.size(); ++pi) {
      auto &pp = sprimes[pi];
      const int p = pp.p;
      for (int t = 0; t < 8; ++t) {
        int i = pp.pos[t];
        const unsigned char m = ~(1 << t);
        for (; i < end; i += p) pblock[i] &= m;
        pp.pos[t] = i;
      }
    }
    for (int i = beg; i < end; ++i)
      for (int m = pblock[i]; m > 0; m &= m - 1)
        primes[psize++] = i * 30 + rs[__builtin_ctz(m)];
  }
  while (psize > 0 && primes[psize - 1] > N) --psize;
  primes.resize(psize);
  return primes;
}
```

Theorem (Fermat)

Every prime of the form 4k +1 is the sum of two squares. A positive integer n is the sum of two squares if and only if all prime factors of the form 4k - 1 have an even exponent in the prime- factorization of n.

```
//Get mth digit after decimal in a/b
int get(int a,int b, int m){
    mod = b;
    return (a * power(10, m - 1) * 10 / b) % 10;
}
```



Legendre's three-square theorem

Let n be a natural number. $n = x^2 + y^2 + z^2$ is solvable in non-negative number $\iff$ n is not of the following form: $4^a(8b+7)$

Lagrange's four-square theorem

Every natural number can be represented as a sum of four non-negative integer squares.

If $n = 4^a(8b+7)$, applying Lagrange's four-square theorem return 4.

Otherwise answer could be 1, 2 or 3.

FFT

```
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> & a, bool invert) {
  int n = a.size();

  for (int i = 1, j = 0; i < n; i++) {
    int bit = n >> 1;
    for (; j & bit; bit >>= 1)
      j ^= bit;
    j ^= bit;

    if (i < j)
      swap(a[i], a[j]);
  }

  for (int len = 2; len <= n; len <<= 1) {
    double ang = 2 * PI / len * (invert ? -1 : 1);
    cd wlen(cos(ang), sin(ang));
    for (int i = 0; i < n; i += len) {
      cd w(1);
      for (int j = 0; j < len / 2; j++) {
        cd u = a[i+j], v = a[i+j+len/2] * w;
        a[i+j] = u + v;
        a[i+j+len/2] = u - v;
        w *= wlen;
      }
    }
  }

  if (invert) {
    for (cd & x : a)
      x /= n;
  }
}

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;
}
```

FFTMOD

```
#define rep(aa, bb, cc) for(int aa = bb; aa < cc;aa++)
#define sz(a) (int)a.size()
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vector<complex<long double>> R(2, 1);
  static vector<C> rt(2, 1);  // (^ 10% faster if double)
  for (static int k = 2; k < n; k *= 2) {
    R.resize(n); rt.resize(n);
    auto x = polar(1.0L, acos(-1.0L) / k);
    rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
  }
  vi rev(n);
  rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
  rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
        // C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-
rolled)  /// include-line
        auto x = (double *)&rt[j+k], y = (double
*)&a[i+j+k];       /// exclude-line
        C z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);
/// exclude-line
        a[i + j + k] = a[i + j] - z;
        a[i + j] += z;
      }
}

template<int M> vi convMod(const vi &a, const vi &b)
{
  if (a.empty() || b.empty()) return {};
  vi res(sz(a) + sz(b) - 1);
  int B=32-__builtin_clz(sz(res)), n=1<<B,
cut=int(sqrt(M));
```

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

NTT

```
const ll mod = (119 << 23) + 1, root = 62; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.


ll modpow(ll b, ll e, int m = mod) {
    ll ans = 1;
    for (; e; b = b * b % m, e /= 2)
        if (e & 1) ans = ans * b % m;
    return ans;
}

// Primitive Root of the mod of form 2^a * b + 1
int generator () {
    vector<int> fact;
    int phi = mod-1, n = phi;
    for (int i=2; i*i<=n; ++i)
        if (n % i == 0) {
            fact.push_back(i);
            while (n % i == 0) n /= i;
        }

    if (n > 1) fact.push_back (n);
    for (int res=2; res<=mod; ++res) {
        bool ok = true;
        for (size_t i=0; i<fact.size() && ok; ++i)
            ok &= modpow (res, phi / fact[i]) != 1;
        if (ok)  return res;
    }
    return -1;
}
```

```
void ntt(vector<int> &a) {
    int n = (int)a.size(), L = 31 - __builtin_clz(n);
    vector<int> rt(2, 1);
    for (int k = 2, s = 2; k < n; k *= 2, s++) {
        rt.resize(n);
        int z[] = {1, modpow(root, mod >> s, mod)};
        for (int i = k; i < 2*k; ++i) rt[i] = (ll)rt[i / 2] * z[i & 1] %
mod;
    }

    vector<int> rev(n);
    for (int i = 0; i < n; ++i) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    for (int i = 0; i < n; ++i) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; ++j) {
                int z = (ll)rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
                a[i + j + k] = ai - z + (z > ai ? mod : 0);
                ai += (ai + z >= mod ? z - mod : z);
            }
        }
    }
}
vector<int> conv(const vector<int> &a, const vector<int>
&b) {
    if (a.empty() || b.empty()) return {};
    int s = (int)a.size() + (int)b.size() - 1, B = 32 -
__builtin_clz(s), n = 1 << B;
    int inv = modpow(n, mod - 2, mod);
    vector<int> L(a), R(b), out(n);
    L.resize(n), R.resize(n);
    ntt(L), ntt(R);
    for (int i = 0; i < n; ++i) out[-i & (n - 1)] = (ll)L[i] * R[i] % mod
* inv % mod;
    ntt(out);
    return {out.begin(), out.begin() + s};
}

ll CRT(ll a, ll m1, ll b, ll m2) {
    __int128 m = m1*m2;
    ll ans = a*m2%m*modpow(m2, m1-2, m1)%m +
m1*b%m*modpow(m1, m2-2, m2)%m;
    return ans % m;
}


// CRT
int mod, root, desired_mod = 1000000007;
const int mod1 = 167772161;
const int mod2 = 469762049;
const int mod3 = 754974721;
const int root1 = 3;
const int root2 = 3;
const int root3 = 11;



int CRT(int a, int b, int c, int m1, int m2, int m3) {
    __int128 M = (__int128)m1*m2*m3;
    ll M1 = (ll)m2*m3;
    ll M2 = (ll)m1*m3;
    ll M3 = (ll)m2*m1;

    int M_1 = modpow(M1%m1, m1 - 2, m1);
    int M_2 = modpow(M2%m2, m2 - 2, m2);
    int M_3 = modpow(M3%m3, m3 - 2, m3);

    __int128 ans = (__int128)a*M1*M_1;
    ans += (__int128)b*M2*M_2;
    ans += (__int128)c*M3*M_3;

    return (ans % M) % desired_mod;
}
```

String Matching

```
void solve(int tc) {
    string s, patt; cin >> s >> patt;
    int n = s.length(), m = patt.length();

    vector<int> poly1(n), poly2(m);

    vector<int> ans_match(n);

    for (int i = 0; i < 26; ++i) {
        for (int j = 0; j < n; ++j)
            poly1[j] = (s[j] - 'a') == i;

        for (int j = 0; j < m; ++j)
            poly2[j] = (patt[m-j-1] - 'a') == i;

        vector<int> ans = multiply(poly1, poly2);
        for (int j = 0; j < n; ++j)
            ans_match[j] += ans[m-1+j];
    }


    int tot = 0;
    vector<int> pos;
    int wild_cnt = count(patt.begin(), patt.end(), '*');
    for (int i = 0; i < n; ++i) {
        if(ans_match[i] == m - wild_cnt) {
            ++tot;
            pos.push_back(i);
        }
    }

    cout << tot << "\n";
    for(auto & p : pos) cout << p << " ";
    cout << "\n";
}
```

String Matching (Wildcard)

```
void solve(int tc) {
    string s, patt; cin >> s >> patt;
    int n = (int)s.length(), m = (int)patt.length();

    vector<cd> poly1(n), poly2(m);

    for (int i = 0; i < n; ++i) {
        double angle = 2*PI*(s[i]-'a')/26;
        poly1[i] = cd(cos(angle), sin(angle));
    }
    for (int i = 0; i < m; ++i) {
        if(patt[m-i-1] == '*') poly2[i] = cd(0,0); // Wild Card
        else {
            double angle = 2*PI*(patt[m-i-1]-'a')/26;
            poly2[i] = cd(cos(angle), -sin(angle));
        }
    }

    vector<cd> ans = multiply(poly1, poly2);
    int wild_cnt = (int)count(patt.begin(), patt.end(), '*');

    int tot = 0;
    vector<int> pos;


    for (int i = 0; i < n; ++i) {
        if(fabs(ans[m-1+i].real() - (m - wild_cnt)) < eps &&
fabs(ans[m-1+i].imag()) < eps) {
            ++tot;
            pos.push_back(i);
        }
    }

    cout << tot << "\n";
    for(auto & p : pos) cout << p << " ";
    cout << "\n";
}
```

Multiply BigInt

```
string mul_two_big_int(const string &s1, const string &s2) {
    int n = s1.size(), m = s2.size();

    vector<int> poly1(n), poly2(m);
    for (int i = 0; i < n; ++i) poly1[n-i-1] = s1[i] - '0';
    for (int i = 0; i < m; ++i) poly2[m-i-1] = s2[i] - '0';

    vector<int> ans = multiply(poly1, poly2);
    int k = ans.size();

    for (int i = 0; i < k - 1; ++i) {
        ans[i + 1] += ans[i] / 10;
        ans[i] = ans[i] % 10;
    }

    string final = to_string(ans[k - 1]);
    for (int i = k - 2; i >= 0; --i)
        final += (char)(ans[i] + '0');

    for (int i = 0; i < k; ++i)
        if(final[i] != '0') return final.substr(i);

    return "0";
}
```

FWHT

```
int add(int a, int b) { return (a + b) % mod; }
int sub(int a, int b) { return (a - b + mod) % mod; }

void fwht(vector<int> &a, int inv, int f) {
    int sz = a.size();
    for (int len = 1; 2 * len <= sz; len <<= 1) {
        for (int i = 0; i < sz; i += 2 * len) {
            for (int j = 0; j < len; j++) {
                int x = a[i + j];
                int y = a[i + j + len];

                if (f == 0) {
                    if (!inv)  a[i + j] = y, a[i + j + len] = add(x,  y);
                    else  a[i + j] = sub(y, x), a[i + j + len] = x;
                }
                else if (f == 1) {
                    if (!inv)  a[i + j + len] = add(x, y);
                    else  a[i + j + len] = sub(y, x);
                }
                else {
                    a[i + j] = add(x, y);
                    a[i + j + len] = sub(x, y);
                }
            }
        }
    }
}

// 0:AND, 1:OR, 2:XOR
vector<int> mul(vector<int> a, vector<int> b, int f) {
    int sz = a.size();
    fwht(a, 0, f);  fwht(b, 0, f);
    vector<int> c(sz);
    for (int i = 0; i < sz; ++i) {
        c[i] = 1ll * a[i] * b[i] % mod;
    }
    fwht(c, 1, f);
    if (f) {
        int sz_inv = power(sz, mod - 2);
        for (int i = 0; i < sz; ++i) {
            c[i] = 1ll * c[i] * sz_inv % mod;
        }
    }
    return c;
}
```

Get Min Cut Edges

// get any set of edges to achieve the min cut (max flow)

```
vector<array<int, 2>> getEdges() {
  vector<int> srcSide(n + 1);
  queue<int> q;
  q.push(s), srcSide[s] = 1;

  while(!q.empty()) {
    int u = q.front();
    q.pop();
    for(auto &i: adj[u]) {
      auto &[_, v, f, c] = edges[i];
      if(!srcSide[v] && f != c) q.push(v), srcSide[v] = 1;
    }
  }

  vector<array<int, 2>> res;
  for(int i = 0; i < edges.size(); i += 2) {
    auto &[u, v, f, c] = edges[i];
    if(srcSide[u] != srcSide[v] && c != oo)
res.push_back({u, v});
  }

  return res;
}
```

Mobius

```
mobius[1] = -1;
for (int i = 1; i < VALMAX; i++) {
  if (mobius[i]) {
    mobius[i] = -mobius[i];
    for (int j = 2 * i; j < VALMAX; j += i) {
      mobius[j] += mobius[i];
    }
  }
}
```

Pascal

```
struct pascal_triangel {
  vector<vector<int>> nCr;

  pascal_triangel(int n) {
    nCr = vector<vector<int>>(n + 1, vector<int>(n +
1));

    nCr[0][0] = 1;
    for (int i = 1; i <= n; i++) {
      nCr[i][0] = 1;
    }
    for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= i; j++) {
        // mod overflow
        nCr[i][j] = nCr[i - 1][j] + nCr[i - 1][j - 1];
      }
    }
  }
};


// Parity of nCr
int nCr(int n, int r){
  if(n < r) return 0;
  return (n & r) == r;
}
```

phi function

```
// phi[p] = p - 1
// phi[p^k] = p^k - p^(k - 1)
// phi[a * b] = phi[a] * phi[b]
// sum:[d|n] phi[d] = n
// a^(phi[m]) = 1 % m
// a^(n) = a^(n % phi[m]) % m

long long phi(long long n) {
    long long ans = n;
    for (int p = 2; 1LL * p * p <= n; p++) {
        if (n % p == 0) {
            while (n % p == 0) {
                n /= p;
            }
            ans -= ans / p;
        }
    }
    if (n > 1) {
        ans -= ans / n;
    }
    return ans;
}


const int N = #;
int phi[N];
void calc_phi() {
    for (int i = 1; i < N; i++) {
        phi[i] = i;
    }
    for (int i = 2; i < N; i++) {
        if (phi[i] == i) {
            for (int j = i; j < N; j += i) {
                phi[j] -= phi[j] / i;
            }
        }
    }
}
```

extended eculidian

```
// x * a + y * b = g
// x = x0 + b / g * t
// y = y0 - a / g * t

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

// extended (a, m, x, y) -> inverse of a
if (g != 1) {
cout << "No solution!";
} else {
x = (x % m + m) % m;
cout << x << endl;
}

vector<int> invs(vector<int> a, int mod) {
    int n = int(a.size());
    vector<int> ret(n);
    int v = 1;
    for (int i = 0; i != n; ++i) {
        ret[i] = v;
        v = 1LL * v * a[i] % mod;
    }
    auto [x, y] = extended_gcd(v, mod);
    x = (x % mod + mod) % mod;
    for (int i = n - 1; i >= 0; --i) {
        ret[i] = 1LL * x * ret[i] % mod;
        x = 1LL * x * a[i] % mod;
    }
    return ret;
}
```

Counting

Combinatorics

▼ nPr

$$\frac{!n}{!(n-r)}$$

▼ nCr

$$\frac{nPr}{!r} = \frac{!n}{!r * !(n-r)}$$

Recurrance relation

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

▼ Stars and Bars

the number of ways to distribute n balls in k boxes

suppose you have k - 1 bars that and you want to place them between the balls

$$\binom{n+k-1}{k-1}$$

▼ Lucas Theorem

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

- `mi` is the coefficient of the number `m` after converting it to base p (the digits it self)
- `ni` is the coefficient of the number `n` after converting it to base p (the digits it self)

- Hockey Stick Identity

$$\sum_{k=r}^{n} \binom{k}{r} = \binom{n+1}{r+1}$$

- Derangements

the number of ways to shuffle a set consisting of `n` elements

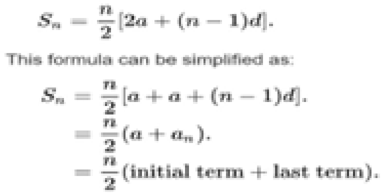
no element stays in his initial position

$$D(n) = (n-1) * (D(n-1) + D(n-2))$$

Counting Tricks

1. Dearrangements :
   1st : NC0 * N! - NC1 * (N-1)! + NC2 * (N-2)! - NC3 * (N-3)! + ... +NCN * 0!
   2st : Dn = (n - 1)(Dn-1 + Dn-2)
   to set k fixed points --> (N)C(K) * D(N-K)

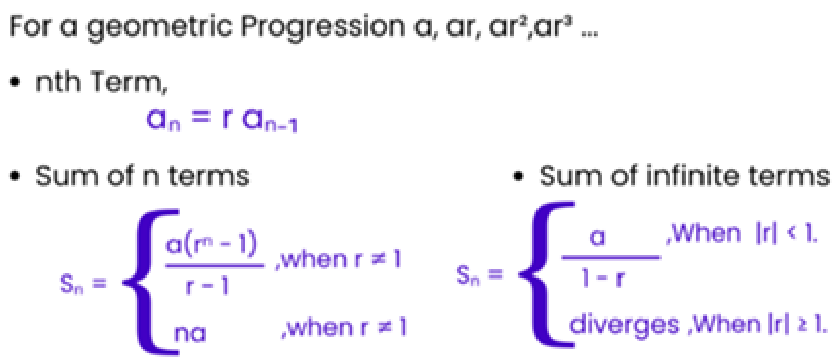
2. NcR = N-1cR + N-1cR-1
3. stars and bars:
   N balls, K baskets : N+K-1 c K-1
   if no basket can have 0 balls : N-1 c K-1
   we can multiply this by N! to get all permutations of it
4. The last digit in the a^b is repeated every 4 times
5. To choose things in increasing order -> use combinations
6. Lattice grid -> number of ways to go from (x1, y1) to (x2, y2)
   a. (x2+y2) – (x1+y1) C (x2 – x1)
7. If we can repeat choosing the element we must use stars and bars not combination (bars represent the switch between element and the other).
8. Number of multisets **[with size(k)]** that have numbers from **0** to **n (n+1 numbers)**
   a. (n + k) C n
9. $\sum_{i=1}^{n} \frac{i}{2^i} = 2^{-n}(-n + 2^{n+1} - 2)$
10. Arithmetic progression:



$$S_n = \frac{n}{2}[2a + (n-1)d].$$

This formula can be simplified as:

$$\begin{aligned} S_n &= \frac{n}{2}[a + a + (n-1)d]. \\ &= \frac{n}{2}(a + a_n). \\ &= \frac{n}{2}(\text{initial term} + \text{last term}). \end{aligned}$$



For a geometric Progression a, ar, ar^2,ar^3 ...

- nth Term,

$$a_n = r\, a_{n-1}$$

- Sum of n terms

$$S_n = \begin{cases} \frac{a(r^n - 1)}{r - 1} & \text{,when } r \neq 1 \\ na & \text{,when } r \neq 1 \end{cases}$$

- Sum of infinite terms

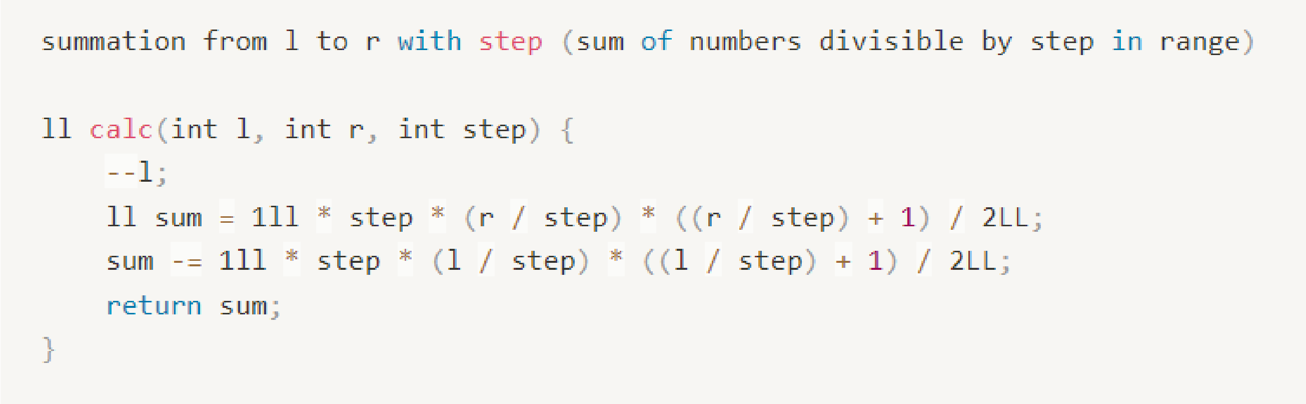
$$S_n = \begin{cases} \frac{a}{1 - r} & \text{,When } |r| < 1. \\ \text{diverges} & \text{,When } |r| \geq 1. \end{cases}$$

Math Formulas & Theoroms

9. $\sum_{i=1}^{n} \frac{i}{2^i} = 2^{-n}(-n + 2^{n+1} - 2)$

Sum

$$\sum_{i=1}^{n} i\, b^i = \frac{b\,(n\, b^{n+1} - (n+1)\, b^n + 1)}{(b-1)^2}$$



```
summation from l to r with step (sum of numbers divisible by step in range)

ll calc(int l, int r, int step) {
    --l;
    ll sum = 1ll * step * (r / step) * ((r / step) + 1) / 2LL;
    sum -= 1ll * step * (l / step) * ((l / step) + 1) / 2LL;
    return sum;
}
```

For two coprime positive integers m and n, that largest number that cannot be written as am + bn (cannot be made by adding up m and n) will be m*n — m — n.

Cayley's formula is the number of spanning trees of N nodes which is equal to N^{N-2}

Number of spanning trees in complete bipartite graph $G_{X,Y}$ is : $X^{Y-1} \times Y^{X-1}$

Such that

X : is the number of nodes in the first set.

Y : is the number of nodes in the second set.

int SumOfOddCubes(int n) {return n * n * (2 * n * n - 1);}

int SumOfOddSquares(int n) {return n * (2 * n + 1) * (2 * n - 1) / 3;}

int SumOfEvenCubes(int n) {return 2 * n * n * (n + 1) * (n + 1);}

int SumOfEvenSquares(int n) {return 2 * n * (n + 1) * (2 * n + 1) / 3;}

- The number of factors of n is : $\tau(n) = \prod_{i=1}^{k} (\alpha_i + 1),$

- Sum of factors of n : $\sigma(n) = \prod_{i=1}^{k} (1 + p_i + \ldots + p_i^{\alpha_i}) = \prod_{i=1}^{k} \frac{p_i^{a_i+1} - 1}{p_i - 1},$

- The product of factors : $\mu(n) = n^{\tau(n)/2}$

$$\sum_{k=1}^{n} k = \tfrac{1}{2}\left(n^2 + n\right)$$

$$\sum_{k=1}^{n} k^2 = \tfrac{1}{6}\left(2\,n^3 + 3\,n^2 + n\right)$$

$$\sum_{k=1}^{n} k^3 = \tfrac{1}{4}\left(n^4 + 2\,n^3 + n^2\right)$$

$$\sum_{k=1}^{n} k^4 = \tfrac{1}{30}\left(6\,n^5 + 15\,n^4 + 10\,n^3 - n\right)$$

$$\sum_{k=1}^{n} k^5 = \tfrac{1}{12}\left(2\,n^6 + 6\,n^5 + 5\,n^4 - n^2\right)$$

$$\sum_{k=1}^{n} k^6 = \tfrac{1}{42}\left(6\,n^7 + 21\,n^6 + 21\,n^5 - 7\,n^3 + n\right)$$

$$\sum_{k=1}^{n} k^7 = \tfrac{1}{24}\left(3\,n^8 + 12\,n^7 + 14\,n^6 - 7\,n^4 + 2\,n^2\right)$$

$$\sum_{k=1}^{n} k^8 = \tfrac{1}{90}\left(10\,n^9 + 45\,n^8 + 60\,n^7 - 42\,n^5 + 20\,n^3 - 3\,n\right)$$

$$\sum_{k=1}^{n} k^9 = \tfrac{1}{20}\left(2\,n^{10} + 10\,n^9 + 15\,n^8 - 14\,n^6 + 10\,n^4 - 3\,n^2\right)$$

$$\sum_{k=1}^{n} k^{10} = \tfrac{1}{66}\left(6\,n^{11} + 33\,n^{10} + 55\,n^9 - 66\,n^7 + 66\,n^5 - 33\,n^3 + 5\,n\right).$$

Gauss

```
const double EPS = 1e-9;
// it doesn't actually have to be infinity or a big number
const int INF = 2;

int gauss (vector < vector<double> > a, vector<double> & ans) {
  int n = a.size(), a[0].size() - 1;

  vector<int> where (m, -1);
  for (int col=0, row=0; col<m && row<n; ++col) {
    int sel = row;
    for (int i=row; i<n; ++i)
      if (abs (a[i][col]) > abs (a[sel][col]))
        sel = i;
    if (abs(a[sel][col]) < EPS) continue;
    for (int i=col; i<=m; ++i)
      swap (a[sel][i], a[row][i]);
    where[col] = row;

    for (int i=0; i<n; ++i)
      if (i != row) {
        double c = a[i][col] / a[row][col];
        for (int j=col; j<=m; ++j)
          a[i][j] -= a[row][j] * c;
      }
    ++row;
  }

  ans.assign (m, 0);
  for (int i=0; i<m; ++i)
    if (where[i] != -1)
      ans[i] = a[where[i]][m] / a[where[i]][i];
  for (int i=0; i<n; ++i) {
    double sum = 0;
    for (int j=0; j<m; ++j)
      sum += ans[j] * a[i][j];
    if (abs (sum - a[i][m]) > EPS)
      return 0;
  }

  for (int i=0; i<m; ++i)
    if (where[i] == -1)
      return INF;
  return 1;
}
```

Guass Mod

```
vector<int> gauss(vector<vector<int>> matrix) {
  int n = matrix.size();
  if (n == 0) return {};
  int vars = matrix[0].size() - 1;

  int row = 0;
  for (int col = 0; col < vars; col++) {
    if (row >= n) break;

    int pivot = -1;
    for (int i = row; i < n; i++) {
      if (matrix[i][col] != 0) {
        pivot = i;
        break;
      }
    }
    if (pivot == -1) continue;

    swap(matrix[row], matrix[pivot]);
    int inv = power(matrix[row][col], mod - 2);
    for (int j = col; j <= vars; j++) {
      matrix[row][j] = 1ll * matrix[row][j] * inv % mod;
    }

    for (int i = 0; i < n; i++) {
      if (i != row && matrix[i][col] != 0) {
        int factor = matrix[i][col];
        for (int j = col; j <= vars; j++) {
          matrix[i][j] -= 1ll * factor * matrix[row][j] % mod;
          if(matrix[i][j] < 0) matrix[i][j] += mod;
        }
      }
    }
    row++;
  }

  for (int i = 0; i < n; i++) {
    bool all_zero = true;
    for (int j = 0; j < vars; j++) {
      if (matrix[i][j] != 0) {
        all_zero = false;
        break;
      }
    }
    if (all_zero && matrix[i][vars] != 0)
      return {};
  }
  // Check for multiple solutions
  if (row < vars) return {};
  vector<int> solution(vars);
  for (int i = 0; i < vars; i++) {
    if (i < matrix.size()) solution[i] = matrix[i][vars];
    else solution[i] = 0;
  }

  return solution;
```

SQRT

```
vector<int> arr;
vector<vector<int>> blk;
int sq;

void build() {
  int n = arr.size(), sq = ceil(sqrt(n));
  blk.assign(sq, {});
  for (int i = 0; i < n; ++i)
    blk[i / sq].push_back(arr[i]);

  for(auto & SD : blk) sort(SD.begin(), SD.end());
}

// 0-indexed
int query(int l, int r, int v) {
  int ans = 0;
  while (l % sq && l <= r) {
    /* check every arr[l] alone*/
    l++;
  }
  while (l + sq - 1 <= r) {
    /* check the block i and get answer from it */
    l += sq;
  }
  while (l <= r) {
    /* check every arr[l] alone*/
    l++;
  }

  return ans;
}

// 0-indexed
void update(int idx, long long val) {
  vector<int> &SD = blk[idx / sq];
  SD[lower_bound(SD.begin(), SD.end(), arr[idx]) -
SD.begin()] = val;
  arr[idx] = val;

  for (int i = 0; i < SD.size() - 1; ++i) {
    if(SD[i] > SD[i + 1]) swap(SD[i], SD[i + 1]);
  }
  for (int i = SD.size() - 1; i > 0; --i) {
    if(SD[i] < SD[i - 1]) swap(SD[i], SD[i - 1]);
  }
}
```

Basis

```
struct Basis {
  int B, sz = 0;
  vector<int> b;

  Basis(int _B): B(_B), b(vector<int>(B)) {}

  void insert(int x) {
    for(int i = B - 1; ~i; --i) {
      if(!(x >> i & 1)) continue;
      if(!b[i]) {
        b[i] = x, ++sz;
        return;
      }
      x ^= b[i];
    }
  }

  bool can(int x) {
    for(int i = B - 1; ~i; --i) {
      if(!(x >> i & 1)) continue;
      if(!b[i]) return 0;
      x ^= b[i];
    }
    return !x;
  }

  int mx() {
    int x = 0;
    for(int i = B - 1; ~i; --i) x = max(x, x ^ b[i]);
    return x;
  }

  int kth(int k) {
    int x = 0, cnt = 1ll << sz;
    for(int i = B - 1; ~i; --i) {
      if(!b[i]) continue;
      cnt /= 2;
      if(x >> i & 1) {
        if(cnt >= k) x ^= b[i];
        else k -= cnt;
      }
      else if(cnt < k) x ^= b[i], k -= cnt;
    }
    return x;
  }

  void make_and(int x) {
    sz = 0;
    vector<int> newly;
    for (int i = B - 1; i >= 0; --i) {
      b[i] &= x;
      if(b[i]) newly.push_back(b[i]);
      b[i] = 0;
    }
    for(auto &i:newly) insert(i);
  }

  void make_or(int x) {
    sz = 0;
    vector<int> newly;
    for (int i = B - 1; i >= 0; --i) {
      b[i] |= x;
      if(b[i]) newly.push_back(b[i]);
      b[i] = 0;
    }
    for(auto &i:newly) insert(i);
  }
};
```

Basis Prefix

```
const int LOG = 21;
struct Basis {
  int basis[LOG];
  int lst_idx[LOG];
  int sz;

  Basis() {
    sz = 0;
    for (int i = LOG - 1; i >= 0; --i) {
      basis[i] = 0;
      lst_idx[i] = -1;
    }
  }
  void insert(int x, int idx) {
    for (int i = LOG - 1; i >= 0; --i) {
      if ((x & (1ll << i)) == 0) continue;
      if(lst_idx[i] < idx)
      {
        if(lst_idx[i] == -1) sz++;
        swap(x, basis[i]);
        swap(lst_idx[i], idx);
      }
      x ^= basis[i];
    }
  }
  int get_max(int l) {
    int ans = 0;
    for (int i = LOG - 1; i >= 0; --i) {
      if(basis[i] && !(ans & (1ll<<i)) && lst_idx[i] >= l)
        ans ^= basis[i];
    }
    return ans;
  }
};
```

Li Chao

```
// To get an instance : node *root = EMPTY;
typedef pair<long long, long long> line;
int start_x, end_x;

ll sub(const line &l, ll x) {
  return l.first * x + l.second;
}

double inter(const line &l1, const line &l2) {
  return (l2.second - l1.second) / (l1.first - l2.first -
.0);
}

extern struct node *const EMPTY;
struct node {
  line l;
  node *lf, *rt;
  node() : l({0, 0}), lf(this), rt(this) {}
  node(line l) : l(l), lf(EMPTY), rt(EMPTY) {}
};
node *const EMPTY = new node;

void insert(line l, node *&cur, ll ns = start_x, ll ne =
end_x) {
  if (cur == EMPTY) {
    cur = new node(l);
    return;
  }
  if (l.first == cur->l.first) {
    cur->l = max(cur->l, l);
    return;
  }
```

```
    auto x = inter(l, cur->l);
    if (x < ns || x > ne) {
        if (sub(l, ns) > sub(cur->l, ns))cur->l = l;
        return;
    }
    ll mid = ns + (ne - ns) / 2;
    if (x <= mid) {
        if (sub(l, ne) > sub(cur->l, ne)) swap(l, cur->l);
        insert(l, cur->lf, ns, mid);
    } else {
        if (sub(l, ns) > sub(cur->l, ns)) swap(l, cur->l);
        insert(l, cur->rt, mid + 1, ne);
    }
}

ll query(ll x, node *cur, ll ns = start_x, ll ne = end_x) {
    auto ret = sub(cur->l, x);
    if(x < ns || x > ne)
        return LLONG_MIN;
    if(ns == ne)
        return ret;
    ll mid = ns + (ne - ns) / 2;
    if (x <= mid)
        return max(ret, query(x, cur->lf, ns, mid));
    return max(ret, query(x, cur->rt, mid + 1, ne));
}
```

Tree Sack

```
// Problem: How many nodes in the subtree of node
// "u" with color equals to "x".
vector<vector<int>> adj;
vector<int> sz, big, col;

vector<vector<pair<int, int>>> Q; // Offline Queries
vector<int> ans; // Answer for each query

int freq[(int)1e5 + 1]; // DS

// Calculate the size for each subtree
// determine the "big" child for each node.
void pre(int u, int p) {
    ++sz[u];
    for(auto &v : adj[u]) {
        if(v == p) continue;
        pre(v, u);
        sz[u] += sz[v];
        if(!big[u] || sz[v] > sz[big[u]]) big[u] = v;
    }
}


// Make the desired operation (Add, Remove).
void operation(int u, int p, int d) {
    freq[col[u]] += d;
    for(auto &v : adj[u]) {
        if(v == p) continue;
        operation(v, u, d);
    }
}

// The flag keep will tell us
// either to keep the node "u" in the DS or not.
void dfs(int u, int p, bool keep) {
    for(auto &v : adj[u]) {
        if(v == p || v == big[u]) continue;
        dfs(v, u, 0);
    }
    if(big[u]) dfs(big[u], u, 1);

    // Add to the DS
    ++freq[col[u]];
    for(auto &v : adj[u]) {
        if(v == p || v == big[u]) continue;
        operation(v, u, +1);
    }

    // Answer the Queries
    for(auto &[c, idx] : Q[u]) {
        ans[idx] = freq[c];
    }

    // Remove form the DS
    if(!keep) operation(u, p, -1);
}
```

Two Stack Queue

```
const int oo = 2e18;
struct TwoStackQ {
  struct Node {
    int mx = -oo, mn = oo, val;
    Node(): val(0) {}
    Node(int x): mx(x), mn(x), val(x) {}
  };

  stack<Node> a, b;

  int size() { return a.size() + b.size(); }

  void mrg(Node &a, Node &b) {
    a.mn = min(a.mn, b.mn);
    a.mx = max(a.mx, b.mx);
  }

  void push(int val) {
    auto nd = Node(val);
    if(!a.empty()) mrg(nd, a.top());
    a.push(nd);
  }

  void move() {
    while(!a.empty()) {
      auto nd = Node(a.top().val);
      if(!b.empty()) mrg(nd, b.top());
      b.push(nd), a.pop();
    }
  }

  Node get() {
    Node res;
    if(!b.empty()) mrg(res, b.top());
    if(!a.empty()) mrg(res, a.top());
    return res;
  }

  void pop() {
    if(b.empty()) move();
    if(!b.empty()) b.pop();
  }
};
```

D&C DP

```
const int N = 3001, oo = 2e18;
int dp[N][2], it = 1, L = 1, R = 0, sum = 0;
int a[N];

void add(int i) {}
void rem(int i) {}

void move(int l, int r) {
  while(R < r) add(++R);
  while(L > l) add(--L);
  while(R > r) rem(R--);
  while(L < l) rem(L++);
}

void go(int l, int r, int lx, int rx) {
  if(l > r) return;

  int m = (l + r) / 2, opt = 1;
  for(int i = lx; i <= min(rx, m); ++i) {
    move(i, m);
    int curr = dp[i - 1][it ^ 1] + sum;
    if(curr > dp[m][it])
      dp[m][it] = curr, opt = i;
  }

  go(l, m - 1, lx, opt);
  go(m + 1, r, opt, rx);
}

void magic() {
  int n, k; cin >> n >> k;
  for(int i = 1; i <= n; ++i) cin >> a[i];

  // base case
  memset(dp, 0, sizeof dp);
  it = 1;
  for(int i = 1; i <= k; ++i, it ^= 1)
    go(1, n, 1, n);

  cout << dp[n][k & 1];
}
```

Treap

```
mt19937
rng(chrono::steady_clock::now().time_since_epoch()
.count());
struct treap {
    treap * left;
    treap * right;
    long long p;
    int val, sz;
    int lazy;
    bool is_lazy;
    treap(int v) {
        left = right = NULL;
        p = rng();
        val = v;
        sz = 1;
        lazy = is_lazy = 0;
    }
};

int size(treap *tp) {
    return tp == NULL ? 0 : tp->sz;
}

treap * recalc(treap * tp) {
    tp->sz = size(tp->left) + size(tp->right) + 1;
    return tp;
}

array<treap *, 2> split(treap * tp, int value) { // 1-indexed
    if(tp == NULL) return {NULL, NULL};

    if (tp->val <= value) {
        auto [l, r] = split (tp->right, value);
        tp->right = l;
        return {recalc(tp), r};
    }

    auto [l, r] = split (tp->left, value);
    tp->left = r;
    return {l, recalc(tp)};
}
```

```
treap * merge(treap * l, treap * r) {
    if(l == NULL) return r;
    if(r == NULL) return l;

    if(l->p < r->p) {
        l->right = merge(l->right, r);
        return recalc(l);
    }

    r->left = merge(l, r->left);
    return recalc(r);
}

void put_into(treap * tp1, treap *& tp2) {
    if(tp1 == NULL) return;
    auto [a, b] = split(tp2, tp1->val);
    tp2 = merge(merge(a, new treap(tp1->val)), b);
    put_into(tp1->left, tp2);
    put_into(tp1->right, tp2);
    delete tp1;
    tp1 = NULL;
}
```

Implicit Treap

```
mt19937
rng(chrono::steady_clock::now().time_since_epoch()
.count());
#define getrand(l, r) uniform_int_distribution<long
long>(l, r)(rng)

struct TreapNode {
    int sz = 1, rev = 0;
    char val;
    ll p = getrand(1, 2e18);
    TreapNode *l = NULL, *r = NULL;
    TreapNode(char a): val(a) {}
};

using Treap = TreapNode*;

int size(Treap t) {
    return t? t->sz : 0;
}

Treap recalc(Treap t) {
    t->sz = 1 + size(t->l) + size(t->r);
    return t;
}

void prop(Treap t) {
    if(!t->rev) return;
    swap(t->l, t->r);
    if(t->l) t->l->rev ^= 1;
    if(t->r) t->r->rev ^= 1;
    t->rev = 0;
}

Treap merge(Treap l, Treap r) {
    if(!l || !r) return r ? r : l;
    prop(l), prop(r);

    if(l->p < r->p) {
        l->r = merge(l->r, r);
        return recalc(l);
    }

    r->l = merge(l, r->l);
    return recalc(r);
}

array<Treap, 2> split(Treap t, int sz) {
    if(!t) return {NULL, NULL};
    prop(t);
    if(size(t->l) >= sz) {
        auto [left, right] = split(t->l, sz);
        t->l = right;
        return {left, recalc(t)};
    }

    auto [left, right] = split(t->r, sz - size(t->l) - 1);
    t->r = left;
    return {recalc(t), right};
}

Treap apply(Treap t, int l, int r) {
    auto [a, b] = split(t, r);
    auto [c, d] = split(a, l - 1);
    d->rev = 1;
    return merge(merge(c, d), b);
}

void print(Treap t) {
    if(!t) return;
    prop(t);
    print(t->l);
    cout << t->val;
    print(t->r);
}
```