GEMP - UFC Quixadá - ICPC Library

Contents

1 Data Structures

1.1 Distinct Values In Range

```cpp
#include "segment_tree_persistent.h"
namespace DistinctValues{
  const int MAXN = 200010;
  int v0[MAXN], tmp[MAXN];
  vector<int> upd[MAXN];
  void init(vector<int> v){
    int n = v.size();
    map<int, int> last;
    for(int i=0; i<n; i++){
      int x = v[i];
      upd[last[x]].push_back(i);
      last[x] = i+1;
    }
    PerSegTree::build(n, v0);
    for(int i=0; i<n; i++){
      for(int p: upd[i])
        PerSegTree::update(p, 1);
      tmp[i] = PerSegTree::t;
    }
  }
  // How many distinct values are there in a range [a,b]
  // 0-indexed
  int query(int a, int b){
    return PerSegTree::query(a, b, tmp[a]);
  }
};
```

1.2 LiChao Tree

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

```
using namespace std;
const int INF = 0x3f3f3f3f;
class LiChaoTree{
private:
  typedef int t_line;
  struct Line{
    t_line k, b;
    Line() {}
    Line(t_line k, t_line b) : k(k), b(b) {}
  };
  int n_tree, min_x, max_x;
  vector<Line> li_tree;
  t_line f(Line l, int x){
    return l.k * x + l.b;
  }
  void add(Line nw, int v, int l, int r){
    int m = (l + r) / 2;
    bool lef = f(nw, l) > f(li_tree[v], l);
    bool mid = f(nw, m) > f(li_tree[v], m);
    if (mid)
      swap(li_tree[v], nw);
    if (r - l == 1)
      return;
    else if (lef != mid)
      add(nw, 2 * v, l, m);
    else
      add(nw, 2 * v + 1, m, r);
  }
  int get(int x, int v, int l, int r){
    int m = (l + r) / 2;
    if (r - l == 1)
      return f(li_tree[v], x);
    else if (x < m)
      return max(f(li_tree[v], x), get(x, 2 * v, l, m));
    else
      return max(f(li_tree[v], x), get(x, 2 * v + 1, m, r));
  }
public:
  LiChaoTree(int mn_x, int mx_x){
    min_x = mn_x;
    max_x = mx_x;
    n_tree = max_x - min_x + 5;
    li_tree.resize(4 * n_tree, Line(0, -INF));
  }
  void add(t_line k, t_line b){
    add(Line(k, b), 1, min_x, max_x);
  }
  t_line get(int x){
    return get(x, 1, min_x, max_x);
  }
};
```

1.3 Line Container

```
#include <bits/stdc++.h>
#pragma once
using ll = long long;
using namespace std;
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};
struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b);
  }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll getMax(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

1.4 Permutation

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
mt19937_64 rng((int) std::chrono::steady_clock::now().time_since_epoch().count
     ());
namespace Permutation{
  const int MAXN = 500010;
  ll mp[MAXN], sumXor[MAXN], p[MAXN+1], inv[MAXN];
  void init(vector<int> v){
    sumXor[0] = inv[0] = p[0] = 0;
    for(int i=0; i<MAXN; i++){
      mp[i] = rng() + 1;
      p[i+1] = p[i] ^ mp[i];
    }
    for(int i=0; i<v.size(); i++){
      if(v[i] < 0 or v[i] >= MAXN){
        inv[i+1] = 1 + inv[i];
        sumXor[i+1] = sumXor[i];
      }else{
        inv[i+1] = inv[i];
        sumXor[i+1] = sumXor[i] ^ mp[v[i]];
      }
    }
  }
  // Verify if {v[l], v[l+1], ..., v[r]} is {0, 1, ... , r-l+1}
  // 0-indexed;
  bool isPermutation(int l, int r){
    l++, r++;
    if(inv[r] - inv[l-1] > 0)
```

```
      return false;
    return p[r-l+1] == (sumXor[r] ^ sumXor[l-1]);
  }
};
```

1.5 Policy Based Tree

```
#include <bits/stdc++.h>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using namespace std;
template <class T> using ordered_set = tree<T, null_type, less<T>, rb_tree_tag
     , tree_order_statistics_node_update>;
template <class K, class V> using ordered_map = tree<K, V, less<K>,
     rb_tree_tag, tree_order_statistics_node_update>;

//order_of_key (k) : Number of items strictly smaller than k .
//find_by_order(k) : K-th element in a set (counting from zero).
```

1.6 Range Color

```
#include <bits/stdc++.h>
using namespace std;
class RangeColor{
private:
  typedef long long ll;
  struct Node{
    ll l, r;
    int color;
    Node() {}
    Node(ll l1, ll r1, int color1) : l(l1), r(r1), color(color1) {}
    bool operator<(const Node &oth) const{
      return r < oth.r;
    }
  };
  std::set<Node> st;
  vector<ll> ans;
public:
  RangeColor(ll first, ll last, int maxColor){
    ans.resize(maxColor + 1);
    ans[0] = last - first + 1LL;
    st.insert(Node(first, last, 0));
  }
  //get color in position x
  int get(ll x){
    auto p = st.upper_bound(Node(0, x - 1LL, -1));
    return p->color;
  }
  //set newColor in [a, b]
  void set(ll a, ll b, int newColor){
    auto p = st.upper_bound(Node(0, a - 1LL, -1));
    assert(p != st.end());
    ll l = p->l;
    ll r = p->r;
    int oldColor = p->color;
    ans[oldColor] -= (r - l + 1LL);
    p = st.erase(p);
    if (l < a){
      ans[oldColor] += (a - l);
      st.insert(Node(l, a - 1LL, oldColor));
    }
    if (b < r){
      ans[oldColor] += (r - b);
      st.insert(Node(b + 1LL, r, oldColor));
    }
    while ((p != st.end()) and (p->l <= b)){
      l = p->l;
      r = p->r;
      oldColor = p->color;
      ans[oldColor] -= (r - l + 1LL);
      if (b < r){
        ans[oldColor] += (r - b);
        st.erase(p);
        st.insert(Node(b + 1LL, r, oldColor));
        break;
      }else{
        p = st.erase(p);
      }
    }
    ans[newColor] += (b - a + 1LL);
    st.insert(Node(a, b, newColor));
  }
  ll countColor(int x){
    return ans[x];
  }
};
```

1.7 RMQ

```
#include <bits/stdc++.h>
using namespace std;
// Source: https://github.com/brunomaletta/Biblioteca
template<typename T> struct RMQ{
  vector<T> v;
  int n; static const int b = 30;
  vector<int> mask, t;
  int op(int x, int y) { return v[x] < v[y] ? x : y; }
  int msb(int x) { return __builtin_clz(1)-__builtin_clz(x); }
  int small(int r, int sz = b) { return r-msb(mask[r]&((1<<sz)-1)); }
  RMQ(const vector<T>& v_) : v(v_), n(v.size()), mask(n), t(n) {
    for (int i = 0, at = 0; i < n; mask[i++] = at |= 1) {
      at = (at<<1)&((1<<b)-1);
      while (at and op(i, i-msb(at&-at)) == i) at ^= at&-at;
    }
    for (int i = 0; i < n/b; i++) t[i] = small(b*i+b-1);
    for (int j = 1; (1<<j) <= n/b; j++) for (int i = 0; i+(1<<j) <= n/b; i++)
      t[n/b*j+i] = op(t[n/b*(j-1)+i], t[n/b*(j-1)+i+(1<<(j-1))]);
  }
  int getPos(int l, int r){
    if (r-l+1 <= b) return small(r, r-l+1);
    int ans = op(small(l+b-1), small(r));
    int x = l/b+1, y = r/b-1;
    if (x <= y) {
      int j = msb(y-x+1);
      ans = op(ans, op(t[n/b*j+x], t[n/b*j+y-(1<<j)+1]));
    }
    return ans;
  }
  T queryMin(int l, int r) {
    return v[getPos(l, r)];
  }
};
```

1.8 Segment Tree Persistent

```
#include <bits/stdc++.h>
using namespace std;
namespace PerSegTree{
  const int MAX = 2e5 + 10, UPD = 2e5 + 10, LOG = 20;
  const int MAXS = 4 * MAX + UPD * LOG;
  typedef long long pst_t;
  pst_t seg[MAXS];
  int T[UPD], L[MAXS], R[MAXS], cnt, t;
  int n, *v;
  pst_t neutral = 0;
  pst_t join(pst_t a, pst_t b){
    return a + b;
  }
  pst_t build(int p, int l, int r){
    if (l == r)
      return seg[p] = v[l];
    L[p] = cnt++, R[p] = cnt++;
    int m = (l + r) / 2;
    return seg[p] = join(build(L[p], l, m), build(R[p], m + 1, r));
  }
  pst_t query(int a, int b, int p, int l, int r){
    if (b < l or r < a)
      return neutral;
    if (a <= l and r <= b)
      return seg[p];
    int m = (l + r) / 2;
    return join(query(a, b, L[p], l, m), query(a, b, R[p], m + 1, r));
  }
  pst_t update(int a, pst_t x, int lp, int p, int l, int r){
    if (l == r)
      return seg[p] = x;
    int m = (l + r) / 2;
    if (a <= m)
      return seg[p] = join(update(a, x, L[lp], L[p] = cnt++, l, m), seg[R[p] =
           R[lp]]);
    return seg[p] = join(seg[L[p] = L[lp]], update(a, x, R[lp], R[p] = cnt++,
        m + 1, r));
  }
//Public:
  //O(n)
  void build(int n2, int *v2){
    n = n2, v = v2;
    T[0] = cnt++;
    build(0, 0, n - 1);
  }
  //O(log(n))
  pst_t query(int a, int b, int tt){
    return query(a, b, T[tt], 0, n - 1);
  }
  //O(log(n))
  //update: v[idx] = x;
  int update(int idx, pst_t x, int tt = t){
    update(idx, x, T[tt], T[++t] = cnt++, 0, n - 1);
    return t;
  }
}; // namespace perseg
```

1.9 Sparse Table

```
#include <bits/stdc++.h>
using namespace std;
class SparseTable{
private:
  typedef int t_st;
  vector<vector<t_st>> st;
  vector<int> log2;
  t_st neutral = 0x3f3f3f3f;
  int nLog;
  t_st join(t_st a, t_st b){
    return min(a, b);
  }
public:
  template <class MyIterator>
  SparseTable(MyIterator begin, MyIterator end){
    int n = end - begin;
    nLog = 20;
    log2.resize(n + 1);
    log2[1] = 0;
    for (int i = 2; i <= n; i++)
      log2[i] = log2[i / 2] + 1;
    st.resize(n, vector<t_st>(nLog, neutral));
    for (int i = 0; i < n; i++, begin++)
      st[i][0] = (*begin);
    for (int j = 1; j < nLog; j++)
      for (int i = 0; (i + (1 << (j - 1))) < n; i++)
        st[i][j] = join(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
  }
  //0-indexed [a, b]
  t_st query(int a, int b){
    int d = b - a + 1;
    t_st ans = neutral;
    for (int j = nLog - 1; j >= 0; j--){
      if (d & (1 << j)){
        ans = join(ans, st[a][j]);
        a = a + (1 << (j));
      }
    }
    return ans;
  }
  //0-indexed [a, b]
  t_st queryRMQ(int a, int b){
    int j = log2[b - a + 1];
    return join(st[a][j], st[b - (1 << j) + 1][j]);
  }
};
```

1.10 SQRT Decomposition

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

```
struct SqrtDecomposition{
  typedef long long t_sqrt;
  int sqrtLen;
  vector<t_sqrt> block;
  vector<t_sqrt> v;
  template <class MyIterator>
  SqrtDecomposition(MyIterator begin, MyIterator end){
    int n = end - begin;
    sqrtLen = (int)sqrt(n + .0) + 1;
    v.resize(n);
    block.resize(sqrtLen + 5);
    for (int i = 0; i < n; i++, begin++){
      v[i] = (*begin);
      block[i / sqrtLen] += v[i];
    }
  }
  //0-indexed
  void update(int idx, t_sqrt new_value){
    t_sqrt d = new_value - v[idx];
    v[idx] += d;
    block[idx / sqrtLen] += d;
  }
  //0-indexed [l, r]
  t_sqrt query(int l, int r){
    t_sqrt sum = 0;
    int c_l = l / sqrtLen, c_r = r / sqrtLen;
    if (c_l == c_r){
      for (int i = l; i <= r; i++)
        sum += v[i];
    }else{
      for (int i = l, end = (c_l + 1) * sqrtLen - 1; i <= end; i++)
        sum += v[i];
      for (int i = c_l + 1; i <= c_r - 1; i++)
        sum += block[i];
      for (int i = c_r * sqrtLen; i <= r; i++)
        sum += v[i];
    }
    return sum;
  }
};
```

1.11 Union Find With Rollback

```
#include <bits/stdc++.h>
using namespace std;
struct RollbackUF {
  vector<int> e;
  vector<tuple<int, int, int, int>> st;
  RollbackUF(int n) : e(n, -1) {}
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : find(e[x]); }
  int time() { return st.size(); }
  void rollback(int t) {
    while (st.size() > t){
      auto [a1, v1, a2, v2] = st.back();
      e[a1] = v1; e[a2] = v2;
      st.pop_back();
    }
  }
  bool unite(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.push_back({a, e[a], b, e[b]});
    e[a] += e[b]; e[b] = a;
    return true;
  }
};
```

2 Graph Algorithms

2.1 2-SAT

```
#include "strongly_connected_component.h"
using namespace std;
struct SAT{
  typedef pair<int, int> pii;
  vector<pii> edges;
  int n;
  SAT(int size){
    n = 2 * size;
  }
  vector<bool> solve2SAT(){
    vector<bool> vAns(n / 2, false);
    vector<int> comp = SCC::scc(n, edges);
    for (int i = 0; i < n; i += 2){
      if (comp[i] == comp[i + 1])
        return vector<bool>();
      vAns[i / 2] = (comp[i] > comp[i + 1]);
    }
    return vAns;
  }
  int v(int x){
    if (x >= 0)
      return (x << 1);
    x = ~x;
    return (x << 1) ^ 1;
  }
  void add(int a, int b){
    edges.push_back(pii(a, b));
  }
  void addOr(int a, int b){
    add(v(~a), v(b));
    add(v(~b), v(a));
  }
  void addImp(int a, int b){
    addOr(~a, b);
  }
  void addEqual(int a, int b){
    addOr(a, ~b);
    addOr(~a, b);
  }
  void addDiff(int a, int b){
    addEqual(a, ~b);
  }

  // Using maxterms
  void addTruthTable(int a, int b, bool v00, bool v01, bool v10, bool v11){
    if(!v00)
      addOr(a, b);
    if(!v01)
      addOr(a, ~b);
    if(!v10)
      addOr(~a, b);
    if(!v11)
      addOr(~a, ~b);
  }
};
```

2.2 Arborescence

```
#include <bits/stdc++.h>
#include "../data_structures/union_find_with_rollback.h"
using ll = long long;
struct Edge { int a, b; ll w; };
struct Node { /// lazy skew heap node
  Edge key;
  Node *l, *r;
  ll delta;
  void prop() {
    key.w += delta;
    if (l) l->delta += delta;
    if (r) r->delta += delta;
    delta = 0;
  }
  Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
  if (!a || !b) return a ?: b;
  a->prop(), b->prop();
  if (a->key.w > b->key.w) swap(a, b);
  swap(a->l, (a->r = merge(b, a->r)));
  return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }
void free(vector<Node*> &v){
  for(auto &x: v)
    delete x;
}
// O(M * log(N))
// return {sum of weights, vector with parents}
pair<ll, vector<int>> dmst(int n, int r, vector<Edge>& g) {
  RollbackUF uf(n);
  vector<Node*> heap(n);
  vector<Node*> vf;
  for (Edge e : g){
    Node* node = new Node{e};
    vf.push_back(node);
    heap[e.b] = merge(heap[e.b], node);
  }
  ll res = 0;
  vector<int> seen(n, -1), path(n), par(n);
  seen[r] = r;
  vector<Edge> Q(n), in(n, {-1, -1}), comp;
  deque<tuple<int, int, vector<Edge>>> cycs;
  for(int s = 0; s < n; ++s) {
    int u = s, qi = 0, w;
    while (seen[u] < 0) {
      if (!heap[u]){
        free(vf);
        return {-1,{}};
      }
      Edge e = heap[u]->top();
      heap[u]->delta -= e.w, pop(heap[u]);
      Q[qi] = e, path[qi++] = u, seen[u] = s;
      res += e.w, u = uf.find(e.a);
      if (seen[u] == s) { /// found cycle, contract
        Node* cyc = 0;
        int end = qi, time = uf.time();
        do cyc = merge(cyc, heap[w = path[--qi]]);
        while (uf.unite(u, w));
        u = uf.find(u), heap[u] = cyc, seen[u] = -1;
        cycs.push_front({u, time, {&Q[qi], &Q[end]}});
      }
    }
    for(int i = 0; i < qi; ++i) in[uf.find(Q[i].b)] = Q[i];
  }
  for (auto& [u, t, c] : cycs) { // restore sol (optional)
    uf.rollback(t);
    Edge inEdge = in[u];
    for (auto& e : c) in[uf.find(e.b)] = e;
    in[uf.find(inEdge.b)] = inEdge;
  }
  for(int i = 0; i < n; ++i) par[i] = in[i].a;
  free(vf);
  return {res, par};
}
//Careful with overflow
pair<ll, vector<int>> dmstAnyRoot(int n, vector<Edge> v) {
  ll maxEdge = 1000000010;
  ll INF = n*maxEdge;
  for(int i=0; i<n; i++)
    v.push_back(Edge({n, i, INF}));
  auto [ans, dad] = dmst(n+1, n, v);
  if(ans >= 0 and ans < 2*INF){
    for(int i=0; i<n; i++)
      if(dad[i] == n)
        dad[i] = -1;
    dad.pop_back();
    return {ans - INF, dad};
  }else{
    return {-1, {}};
  }
}
```

2.3 Centroid

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 500010;
typedef pair<int, int> pii;
namespace Centroid{
  vector<int> adj[MAXN];
  int sub[MAXN];
  int n;
  void init(int n1){
    n = n1;
    for(int i=0; i<n; i++) adj[i].clear();
  }
  void addEdge(int a, int b){
    adj[a].push_back(b);
    adj[b].push_back(a);
  }
  int dfsS(int u, int p){
    sub[u] = 1;
    for(int to: adj[u]){
      if(to != p)
        sub[u] += dfsS(to, u);
    }
    return sub[u];
  }
  pii dfsC(int u, int p){
    for(int to : adj[u]){
      if(to != p and sub[to] > n/2)
        return dfsC(to, u);
    }
    for(int to : adj[u]){
      if(to != p and (sub[to]*2) == n)
        return pii(u, to);
    }
    return pii(u, u);
  }
  pii findCentroid(){
    dfsS(0, -1);
    return dfsC(0, -1);
  }
}
```

2.4 Centroid Decomposition

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
// O(N*log(N))
// Centroid Decomposition
const int MAXN = 200010;
namespace CD{
  vector<int> adj[MAXN];
  int dad[MAXN], sub[MAXN];
  bool rem[MAXN];
  int centroidRoot, n;
  void init(int n1){
    n = n1;
    for(int i=0; i<n; i++){
      adj[i].clear();
      rem[i] = false;
    }
  }
  int dfs(int u, int p){
    sub[u] = 1;
    for (int to : adj[u]){
      if (!rem[to] and to != p)
        sub[u] += dfs(to, u);
    }
    return sub[u];
  }
  int centroid(int u, int p, int sz){
    for (auto to : adj[u])
      if (!rem[to] and to != p and sub[to] > sz / 2)
        return centroid(to, u, sz);
    return u;
  }
  void getChildren(int u, int p, int d, vector<int> &v){
    v.push_back(d);
    for(int to: adj[u]){
      if(rem[to] or to == p)
        continue;
      getChildren(to, u, d+1, v);
    }
  }
  ll ans = 0;
  int k;
  int decomp(int u, int p){
    int sz = dfs(u, p);
    int c = centroid(u, p, sz);
    if (p == -1)
      p = c;
    dad[c] = p;
    rem[c] = true;
    // Begin
    vector<int> f(sz+1, 0);
    f[0] = 1;
    for (auto to : adj[c]) if (!rem[to]){
      vector<int> v;
      getChildren(to, c, 1, v);
      for(int d: v){ // Query
        if(d <= k and k-d <= sz)
          ans += f[k-d];
      }
      for(int d: v) // Update
        f[d]++;
    }
    // End
    for (auto to : adj[c]){
      if (!rem[to])
        decomp(to, c);
    }
    return c;
  }
  void addEdge(int a, int b){
    adj[a].push_back(b);
    adj[b].push_back(a);
  }
  // Number of k-size paths: O(N * log(N))
  ll solve(int k1){
    assert(n > 0);
    ans = 0, k = k1;
    centroidRoot = decomp(0, -1);
    return ans;
  }
};
```

2.5 Checking Bipartiteness Online

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii;
const int N = 500010;
```

```
pii parent[N];
int rk[N];
int bipartite[N];
void make_set(int v) {
  parent[v] = pii(v, 0);
  rk[v] = 0;
  bipartite[v] = true;
}
pii find_set(int v) {
  if (v != parent[v].first) {
    int parity = parent[v].second;
    parent[v] = find_set(parent[v].first);
    parent[v].second ^= parity;
  }
  return parent[v];
}
void add_edge(int a, int b) {
  int x, y;
  tie(a, x) = find_set(a);
  tie(b, y) = find_set(b);
  if (a == b) {
    if (x == y)
      bipartite[a] = false;
  }else{
    if (rk[a] < rk[b])
      swap (a, b);
    parent[b] = pii(a, x^y^1);
    bipartite[a] &= bipartite[b];
    if (rk[a] == rk[b])
      ++rk[a];
  }
}
bool is_bipartite(int v) {
  return bipartite[find_set(v).first];
}
```

2.6 Edmond's Blossoms

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 510;
// Adaptado de: https://github.com/brunomaletta/Biblioteca/blob/master/Codigo/
     Grafos/blossom.cpp
// Edmond's Blossoms algorithm give a maximum matching in general graphs (non-
     bipartite)
// O(N^3)
namespace EdmondBlossoms{
vector<int> adj[MAXN];
int match[MAXN];
int n, pai[MAXN], base[MAXN], vis[MAXN];
queue<int> q;
void init(int n1){
  n = n1;
  for(int i=0; i<n; i++)
    adj[i].clear();
}
void addEdge(int a, int b){
  adj[a].push_back(b);
  adj[b].push_back(a);
}
void contract(int u, int v, bool first = 1) {
  static vector<bool> bloss;
  static int l;
  if (first) {
    bloss = vector<bool>(n, 0);
    vector<bool> teve(n, 0);
    int k = u; l = v;
    while (1) {
      teve[k = base[k]] = 1;
      if (match[k] == -1) break;
      k = pai[match[k]];
    }
    while (!teve[l = base[l]]) l = pai[match[l]];
  }
  while (base[u] != l) {
    bloss[base[u]] = bloss[base[match[u]]] = 1;
    pai[u] = v;
    v = match[u];
    u = pai[match[u]];
  }
  if (!first) return;
  contract(v, u, 0);
  for (int i = 0; i < n; i++) if (bloss[base[i]]) {
    base[i] = l;
    if (!vis[i]) q.push(i);
    vis[i] = 1;
  }
}
int getpath(int s) {
  for (int i = 0; i < n; i++)
    base[i] = i, pai[i] = -1, vis[i] = 0;
  vis[s] = 1; q = queue<int>(); q.push(s);
  while (q.size()) {
    int u = q.front(); q.pop();
    for (int i : adj[u]) {
      if (base[i] == base[u] or match[u] == i) continue;
      if (i == s or (match[i] != -1 and pai[match[i]] != -1))
        contract(u, i);
      else if (pai[i] == -1) {
        pai[i] = u;
        if (match[i] == -1) return i;
        i = match[i];
        vis[i] = 1; q.push(i);
      }
    }
  }
  return -1;
}
typedef pair<int, int> pii;
vector<pii> maximumMatching(){
  vector<pii> ans;
  memset(match, -1, sizeof(match));
  for (int i = 0; i < n; i++) if (match[i] == -1)
    for (int j : adj[i]) if (match[j] == -1) {
      match[i] = j;
      match[j] = i;
      break;
    }
  for (int i = 0; i < n; i++) if (match[i] == -1) {
    int j = getpath(i);
    if (j == -1) continue;
    while (j != -1) {
      int p = pai[j], pp = match[p];
      match[p] = j;
      match[j] = p;
      j = pp;
    }
  }
  for(int i=0; i < n; i++)
    if(i < match[i])
      ans.emplace_back(i, match[i]);
  return ans;
}
};
```

2.7 Eulerian Path

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii;
template<bool directed=false> struct EulerianPath{
  vector<vector<pii>> adj;
  vector<int> ans, pos;
  vector<bool> used;
  int n, m;
  EulerianPath(int n1){
    n = n1; m = 0;
    adj.assign(n, vector<pii>());
  }
  void addEdge(int a, int b) {
    int at = m++;
    adj[a].push_back({b, at});
    if (!directed) adj[b].push_back({a, at});
  }
  void dfs(int u){
    stack<int> st;
    st.push(u);
    while(!st.empty()){
      u = st.top();
      if(pos[u] < adj[u].size()){
        auto [to, id] = adj[u][pos[u]];
        pos[u]++;
        if(!used[id]){
          used[id] = true;
          st.push(to);
        }
      }else{
        ans.push_back(u);
        st.pop();
      }
    }
  }
  // Remember to call the correct src
  // If you want to check if there is an answer remember to check if all |
       components| > 1 of the graph are connected
  vector<int> getPath(int src){
    pos.assign(n, 0);
    used.assign(m, false);
    ans.clear();
    dfs(src);
    reverse(ans.begin(), ans.end());
    return ans;
  }
};
```

2.8 Find Cycle Negative

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef tuple<int, int, int> Edge;
vector<int> findNegativeCycle(vector<Edge> edges, int n){
  vector<ll> d(n, 0);
  vector<int> p(n, -1);
  int last = -1;
  for(int i = 0; i < n; ++i){
    last = -1;
    for(auto [u, to, w] : edges){
      if(d[u] + w < d[to]){
        d[to] = d[u] + w;
        p[to] = u;
        last = to;
      }
    }
  }
  if(last == -1){
    return {};
  }else{
    for(int i = 0; i < n; i++)
      last = p[last];
    vector<int> cycle;
    for(int v = last; ; v = p[v]){
      cycle.push_back(v);
      if(v == last && cycle.size() > 1)
        break;
    }
    reverse(cycle.begin(), cycle.end());
    return cycle;
  }
}
```

2.9 Flow With Demand

```
#include "dinic.h"
using namespace std;
template <typename flow_t>
struct MaxFlowEdgeDemands{
  Dinic<flow_t> mf;
  vector<flow_t> ind, outd;
  flow_t D;
  int n;
  MaxFlowEdgeDemands(int n) : n(n){
    D = 0;
    mf.init(n + 2);
    ind.assign(n, 0);
    outd.assign(n, 0);
  }
  void addEdge(int a, int b, flow_t cap, flow_t demands){
    mf.addEdge(a, b, cap - demands);
    D += demands;
    ind[b] += demands;
    outd[a] += demands;
  }
  bool solve(int s, int t){
    mf.addEdge(t, s, numeric_limits<flow_t>::max());
    for (int i = 0; i < n; i++){
      if (ind[i]) mf.addEdge(n, i, ind[i]);
      if (outd[i]) mf.addEdge(i, n + 1, outd[i]);
    }
    return mf.maxFlow(n, n + 1) == D;
  }
};
```

2.10 Graph Theorem

```
#include <bits/stdc++.h>
#define all(x) x.begin(),x.end()
using namespace std;
using ll = long long;
using pii = pair<int, int>;
namespace GraphTheorem{
  // return if a sequence of integers d can be represented as the
  // degree sequence of a finite simple graph on n vertices
  bool ErdosGallai(vector<int> d){
    int n = d.size();
    sort(all(d), greater<int>());
    ll sum1 = 0, sum2 = 0;
    int mn = n-1;
    for(int k=1; k<=n; k++){
      sum1 += d[k-1];
      while(k <= mn and k > d[mn])
        sum2 += d[mn--];
      if(mn + 1 < k)
        sum2 -= d[mn++];
      ll a = sum1, b = k*(ll)mn + sum2;
      if(a > b)
        return false;
    }
    return sum1%2 == 0;
  }
  vector<pii> recoverErdosGallai(vector<int> d){
    int n = d.size();
    priority_queue<pii> pq;
    for(int i=0; i<n; i++)
      pq.emplace(d[i], i);
    vector<pii> edges;
    while(!pq.empty()){
      auto [g, u] = pq.top();
      pq.pop();
      vector<pii> aux(g);
      for(int i=0; i<g; i++){
        if(pq.empty())
          return {};
        auto [g2, u2] = pq.top();
        pq.pop();
        if(g2 == 0)
          return {};
        edges.emplace_back(u, u2);
        aux[i] = pii(g2-1, u2);
      }
      for(auto [g2, u2]: aux)
        pq.emplace(g2, u2);
    }
    return edges;
  }
};
```

2.11 Prim

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
const int MAXN = 500010;
namespace Prim{
  vector<pii> adj[MAXN];
  int weight[MAXN];
  bool seen[MAXN];
  int n;
  void init(int n1){
    n = n1;
    for(int i=0; i<n; i++) adj[i].clear();
  }
  void addEdge(int a, int b, int w){
    adj[a].emplace_back(w, b);
    adj[b].emplace_back(w, a);
  }
  ll solve(){
    for(int i=0; i<n; i++){
      weight[i] = 0x3f3f3f3f;
      seen[i] = 0;
    }
    weight[0] = 0;
    priority_queue<pii, vector<pii>, greater<pii> > st;
    st.push(pii(weight[0], 0));
    ll ans = 0;
    while(!st.empty()){
      int u = st.top().second;
      st.pop();
      if(seen[u])
        continue;
      seen[u] = true;
      ans += weight[u];
      for(auto [edge, to]: adj[u]){
        if(!seen[to] and (edge < weight[to])){
          weight[to] = edge;
          st.emplace(weight[to], to);
        }
      }
    }
    return ans;
  }
};
```

2.12 Kuhn

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

```
using namespace std;
mt19937 rng((int)chrono::steady_clock::now().time_since_epoch().count());
namespace Kuhn{
  int na, nb;
  vector<vector<int>> adj;
  vector<int> vis, ma, mb;
  void init(int na1, int nb1){
    na = na1, nb = nb1;
    adj.assign(na, vector<int>());
    vis.assign(na + nb, 0);
    ma.assign(na, -1);
    mb.assign(nb, -1);
  }
  void addEdge(int a, int b) {
    adj[a].push_back(b);
  }
  bool dfs(int u) {
    vis[u] = 1;
    for (int to : adj[u]){
      if(vis[na+to])
        continue;
      vis[na+to] = 1;
      if (mb[to] == -1 or dfs(mb[to])) {
        ma[u] = to, mb[to] = u;
        return true;
      }
    }
    return false;
  }
  int matching() {
    int ans = 0, c = 1;
    for (auto& v: adj)
      shuffle(v.begin(), v.end(), rng);
    while (c) {
      for (int j = 0; j < nb; j++)
        vis[na+j] = 0;
      c = 0;
      for (int i = 0; i < na; i++)
        if (ma[i] == -1 and dfs(i))
          ans++, c = 1;
    }
    return ans;
  }
  pair<vector<int>, vector<int>> minimumVertexCover() {
    matching();
    for (int i = 0; i < na+nb; i++)
      vis[i] = 0;
    for (int i = 0; i < na; i++)
      if (ma[i] == -1)
        dfs(i);
    vector<int> va, vb;
    for (int i = 0; i < na; i++)
      if (!vis[i])
        va.push_back(i);
    for (int i = 0; i < nb; i++)
      if (vis[na+i])
        vb.push_back(i);
    return {va, vb};
  }
  vector<int> maximumAntichain(){
    auto [l, r] = minimumVertexCover();
    set<int> L(l.begin(), l.end());
    set<int> R(r.begin(), r.end());
    vector<int> ans;
    for (int i = 0; i < na; i++)
      if (!L.count(i) and !R.count(i))
        ans.push_back(i);
    return ans;
  }

  vector<vector<int>> minimumNumberChains(){
    matching();
    vector<vector<int>> chains;
    for(int i=0; i<na; i++) if(mb[i] == -1){
      vector<int> path;
      for(int x=i; x != -1; x=ma[x])
        path.push_back(x);
      chains.push_back(path);
    }
    return chains;
  }
};
```

2.13 Link-Cut Tree

```
#include <bits/stdc++.h>
using namespace std;
// Link-Cut Tree, directed version.
// All operations are O(log(n)) amortized.
//Source: https://github.com/brunomaletta/Biblioteca/
const int MAXN = 200010;
namespace LCT {
  struct node {
    int p, ch[2];
    node() { p = ch[0] = ch[1] = -1; }
  };
  node t[MAXN];
  bool isRoot(int x) {
    return t[x].p == -1 or (t[t[x].p].ch[0] != x and t[t[x].p].ch[1] != x);
  }
  void rotate(int x) {
    int p = t[x].p, pp = t[p].p;
    if (!isRoot(p)) t[pp].ch[t[pp].ch[1] == p] = x;
    bool d = t[p].ch[0] == x;
    t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
    if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
    t[x].p = pp, t[p].p = x;
  }
  void splay(int x) {
    while (!isRoot(x)) {
      int p = t[x].p, pp = t[p].p;
      if (!isRoot(p))
        rotate((t[pp].ch[0] == p)^(t[p].ch[0] == x) ? x : p);
      rotate(x);
    }
  }
  int access(int v) {
    int last = -1;
    for (int w = v; w+1; last = w, splay(v), w = t[v].p)
      splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
    return last;
  }
// Public:
  void init(int n){
    for(int i=0; i<=n; i++)
      t[i] = node();
  }
  int findRoot(int v) {
    access(v);
    while (t[v].ch[0]+1) v = t[v].ch[0];
    return splay(v), v;
  }
  // V must be root. W will be the dad of V.
  void link(int v, int w) {
    access(v);
    t[v].p = w;
  }
  // Removes edge (v, dad[v])
  void cut(int v) {
    access(v);
    if(t[v].ch[0] == -1)
      return;
    t[v].ch[0] = t[t[v].ch[0]].p = -1;
  }
  int lca(int v, int w) {
    if(findRoot(v) != findRoot(w))
      return -1;
    access(v);
    return access(w);
  }
}
```

2.14 Link-Cut Tree - Edge

```
#include <bits/stdc++.h>
using namespace std;
// Link-Cut Tree - Edge, undirected version.
// All operations are O(log(n)) amortized.
// Source: https://github.com/brunomaletta/Biblioteca/
typedef long long ll;
typedef pair<int, int> pii;
const int MAXN = 100010, MAXQ = 100010;
namespace LCT {
  struct node {
    int p, ch[2];
    ll val, sub;
    bool rev;
    int sz, ar;
    ll lazy;
    node() {}
    node(int v, int ar_) :
    p(-1), val(v), sub(v), rev(0), sz(ar_), ar(ar_), lazy(0) {
      ch[0] = ch[1] = -1;
    }
  };
  node t[MAXN + MAXQ]; // MAXN + MAXQ
  map<pii, int> edges;
  int sz;
  void prop(int x) {
    if (t[x].lazy) {
      if (t[x].ar) t[x].val += t[x].lazy;
      t[x].sub += t[x].lazy*t[x].sz;
      if (t[x].ch[0]+1) t[t[x].ch[0]].lazy += t[x].lazy;
      if (t[x].ch[1]+1) t[t[x].ch[1]].lazy += t[x].lazy;
    }
    if (t[x].rev) {
      swap(t[x].ch[0], t[x].ch[1]);
      if (t[x].ch[0]+1) t[t[x].ch[0]].rev ^= 1;
      if (t[x].ch[1]+1) t[t[x].ch[1]].rev ^= 1;
    }
    t[x].lazy = 0, t[x].rev = 0;
  }
  void update(int x) {
    t[x].sz = t[x].ar, t[x].sub = t[x].val;
    for (int i = 0; i < 2; i++) if (t[x].ch[i]+1) {
      prop(t[x].ch[i]);
      t[x].sz += t[t[x].ch[i]].sz;
      t[x].sub += t[t[x].ch[i]].sub;
    }
  }
  bool is_root(int x) {
    return t[x].p == -1 or (t[t[x].p].ch[0] != x and t[t[x].p].ch[1] != x);
  }
  void rotate(int x) {
    int p = t[x].p, pp = t[p].p;
    if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
    bool d = t[p].ch[0] == x;
    t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
    if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
    t[x].p = pp, t[p].p = x;
    update(p), update(x);
  }
  int splay(int x) {
    while (!is_root(x)) {
      int p = t[x].p, pp = t[p].p;
      if (!is_root(p)) prop(pp);
      prop(p), prop(x);
      if (!is_root(p)) rotate((t[pp].ch[0] == p)^(t[p].ch[0] == x) ? x : p);
      rotate(x);
    }
    return prop(x), x;
  }
  int access(int v) {
    int last = -1;
    for (int w = v; w+1; update(last = w), splay(v), w = t[v].p)
      splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
    return last;
  }
  void rootify(int v);
  void link_(int v, int w) {
    rootify(w);
    t[w].p = v;
  }
  void cut_(int v, int w) {
    rootify(w), access(v);
    t[v].ch[0] = t[t[v].ch[0]].p = -1;
  }
  void makeTree(int v, int w=0, int ar=0) {
    t[v] = node(w, ar);
  }
// Public:
  void init(int n){
    edges.clear();
    sz = 0;
    for(int i=0; i<=n; i++)
      makeTree(i);
  }
  int findRoot(int v) {
    access(v), prop(v);
    while (t[v].ch[0]+1) v = t[v].ch[0], prop(v);
    return splay(v);
  }
  // Checks if v and w are connected
  bool connected(int v, int w) {
    access(v), access(w);
    return v == w ? true : t[v].p != -1;
  }
  // Change v to be root
  void rootify(int v) {
    access(v);
    t[v].rev ^= 1;
  }
  // Sum of the edges in path from v to w
  ll query(int v, int w) {
    rootify(w), access(v);
    return t[v].sub;
  }
  // Sum +x in path from v to w
  void update(int v, int w, int x) {
    rootify(w), access(v);
    t[v].lazy += x;
  }
  // Add edge (v, w) with weight x
  void link(int v, int w, int x) {
    int id = MAXN + sz++;
    edges[pii(v, w)] = id;
    makeTree(id, x, 1);
    link_(v, id), link_(id, w);
  }
  // Remove edge (v, w)
  void cut(int v, int w) {
    int id = edges[pii(v, w)];
    cut_(v, id), cut_(id, w);
  }
  int lca(int v, int w) {
    access(v);
    return access(w);
  }
}
```

2.15 Link-Cut Tree - Vertex

```
#include <bits/stdc++.h>
using namespace std;
// Link-Cut Tree - Vertex, undirected version.
// All operations are O(log(n)) amortized.
// Source: https://github.com/brunomaletta/Biblioteca/
typedef long long ll;
typedef pair<int, int> pii;
const int MAXN = 200010;
namespace lct {
  struct node {
    int p, ch[2];
    ll val, sub;
    bool rev;
    int sz;
    ll lazy;
    node() {}
    node(int v) : p(-1), val(v), sub(v), rev(0), sz(1), lazy(0) {
      ch[0] = ch[1] = -1;
    }
  };
  node t[MAXN];
  void prop(int x) {
    if (t[x].lazy) {
      t[x].val += t[x].lazy, t[x].sub += t[x].lazy*t[x].sz;
      if (t[x].ch[0]+1) t[t[x].ch[0]].lazy += t[x].lazy;
      if (t[x].ch[1]+1) t[t[x].ch[1]].lazy += t[x].lazy;
    }
    if (t[x].rev) {
      swap(t[x].ch[0], t[x].ch[1]);
      if (t[x].ch[0]+1) t[t[x].ch[0]].rev ^= 1;
      if (t[x].ch[1]+1) t[t[x].ch[1]].rev ^= 1;
    }
    t[x].lazy = 0, t[x].rev = 0;
  }
  void update(int x) {
    t[x].sz = 1, t[x].sub = t[x].val;
    for (int i = 0; i < 2; i++) if (t[x].ch[i]+1) {
      prop(t[x].ch[i]);
      t[x].sz += t[t[x].ch[i]].sz;
      t[x].sub += t[t[x].ch[i]].sub;
    }
  }
  bool is_root(int x) {
    return t[x].p == -1 or (t[t[x].p].ch[0] != x and t[t[x].p].ch[1] != x);
  }
  void rotate(int x) {
    int p = t[x].p, pp = t[p].p;
    if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
    bool d = t[p].ch[0] == x;
    t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
    if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
    t[x].p = pp, t[p].p = x;
    update(p), update(x);
  }
  int splay(int x) {
    while (!is_root(x)) {
      int p = t[x].p, pp = t[p].p;
      if (!is_root(p)) prop(pp);
      prop(p), prop(x);
      if (!is_root(p)) rotate((t[pp].ch[0] == p)^(t[p].ch[0] == x) ? x : p);
      rotate(x);
    }
    return prop(x), x;
  }
  int access(int v) {
    int last = -1;
    for (int w = v; w+1; update(last = w), splay(v), w = t[v].p)
      splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
    return last;
  }
  // Public:
```

```
  void makeTree(int v, int w) {
    t[v] = node(w);
  }
  int findRoot(int v) {
    access(v), prop(v);
    while (t[v].ch[0]+1) v = t[v].ch[0], prop(v);
    return splay(v);
  }
  // Checks if v and w are connected
  bool connected(int v, int w) {
    access(v), access(w);
    return v == w ? true : t[v].p != -1;
  }
  // Change v to be root
  void rootify(int v) {
    access(v);
    t[v].rev ^= 1;
  }
  // Sum of the weight in path from v to w
  ll query(int v, int w) {
    rootify(w), access(v);
    return t[v].sub;
  }
  // Sum +x in path from v to w
  void update(int v, int w, int x) {
    rootify(w), access(v);
    t[v].lazy += x;
  }
  // Add edge (v, w)
  void link(int v, int w) {
    rootify(w);
    t[w].p = v;
  }
  // Remove edge (v, w)
  void cut(int v, int w) {
    rootify(w), access(v);
    t[v].ch[0] = t[t[v].ch[0]].p = -1;
  }
  int lca(int v, int w) {
    access(v);
    return access(w);
  }
}
```

2.16 Min-Cut

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
//This algorithm finds the Global Min-Cut in O(|V|^3)
namespace MinCut{
  const int MAXN = 510;
  bool exist[MAXN], in_a[MAXN];
  ll g[MAXN][MAXN], w[MAXN];
  vector<int> v[MAXN];
  int n;
  void init(int n1){
    n = n1;
    memset(g, 0, sizeof(g));
  }
  void addEdge(int a, int b, int w1){
    if(a == b) return;
    g[a][b] += w1;
    g[b][a] += w1;
  }
  pair<ll, vector<int>> mincut() {
    ll best_cost = 0x3f3f3f3f3f3f3f3fLL;
    vector<int> best_cut;
    for (int i=0; i<n; ++i)
      v[i].assign (1, i);
    memset (exist, true, sizeof(exist));
    for(int ph=0; ph<n-1; ++ph) {
      memset (in_a, false, sizeof in_a);
      memset (w, 0, sizeof w);
      for(int it=0, prev=0; it<n-ph; ++it){
        int sel = -1;
        for(int i=0; i<n; ++i)
          if(exist[i] && !in_a[i] && (sel == -1 || w[i] > w[sel]))
            sel = i;
        if(it == n-ph-1){
          if(w[sel] < best_cost)
            best_cost = w[sel],  best_cut = v[sel];
          v[prev].insert (v[prev].end(), v[sel].begin(), v[sel].end());
          for(int i=0; i<n; ++i)
            g[prev][i] = g[i][prev] += g[sel][i];
          exist[sel] = false;
        }else{
          in_a[sel] = true;
          for(int i=0; i<n; ++i)
            w[i] += g[sel][i];
          prev = sel;
        }
      }
    }
    return {best_cost, best_cut};
  }
};
```

2.17 Minimum Cost Maximum Flow

```
#include <bits/stdc++.h>
using namespace std;
//O(MaxFlow * path) or
//O(N * M * Path) = O(N^2*M^2) or O(N*M^2*log(n)) or O(N^3*M)
//                    SPFA          Dijkstra        Dijkstra
template <class T = int>
class MCMF{
private:
  struct Edge{
    int to;
    T cap, cost;
    Edge(int a, T b, T c) : to(a), cap(b), cost(c) {}
  };
  int n;
  vector<vector<int>> edges;
  vector<Edge> list;
  vector<int> from;
  vector<T> dist, pot;
  vector<bool> visit;
  pair<T, T> augment(int src, int sink){
    pair<T, T> flow = {list[from[sink]].cap, 0};
    for (int v = sink; v != src; v = list[from[v] ^ 1].to){
      flow.first = std::min(flow.first, list[from[v]].cap);
      flow.second += list[from[v]].cost;
    }
    for (int v = sink; v != src; v = list[from[v] ^ 1].to){
      list[from[v]].cap -= flow.first;
      list[from[v] ^ 1].cap += flow.first;
    }
    return flow;
  }
  queue<int> q;
  bool SPFA(int src, int sink){
    T INF = numeric_limits<T>::max();
    dist.assign(n, INF);
    from.assign(n, -1);
    q.push(src);
    dist[src] = 0;
    while (!q.empty()){
      int on = q.front();
      q.pop();
      visit[on] = false;
      for (auto e : edges[on]){
        auto ed = list[e];
        if (ed.cap == 0)
          continue;
        T toDist = dist[on] + ed.cost + pot[on] - pot[ed.to];
        if (toDist < dist[ed.to]){
          dist[ed.to] = toDist;
          from[ed.to] = e;
          if (!visit[ed.to]){
            visit[ed.to] = true;
            q.push(ed.to);
          }
        }
      }
    }
    return dist[sink] < INF;
  }
  void fixPot(){
    T INF = numeric_limits<T>::max();
    for (int i = 0; i < n; i++){
      if (dist[i] < INF)
        pot[i] += dist[i];
    }
  }
public:
  MCMF(int size){
    n = size;
    edges.resize(n);
    pot.assign(n, 0);
    dist.resize(n);
    visit.assign(n, false);
  }
  pair<T, T> solve(int src, int sink){
    pair<T, T> ans(0, 0);
    // Remove negative edges: Johnson's Algorithm
    if (!SPFA(src, sink))
      return ans;
    fixPot();
    // Can use dijkstra to speed up depending on the graph
    while (SPFA(src, sink)){
      auto flow = augment(src, sink);
      // When the priority is the minimum cost and not the flow
      // if(flow.second >= 0)
      //   break;
      ans.first += flow.first;
      ans.second += flow.first * flow.second;
      fixPot();
    }
    return ans;
  }
  void addEdge(int u, int to, T cap, T cost){
    edges[u].push_back(list.size());
    list.push_back(Edge(to, cap, cost));
    edges[to].push_back(list.size());
    list.push_back(Edge(u, 0, -cost));
  }
};
```

2.18 Topological Sort

```
#include <bits/stdc++.h>
using namespace std;
namespace TopologicalSort{
  typedef pair<int, int> pii;
  vector<vector<int>> adj;
  vector<bool> visited;
  vector<int> vAns;
  void dfs(int u){
    visited[u] = true;
    for (int to : adj[u]){
      if (!visited[to])
        dfs(to);
    }
    vAns.push_back(u);
  }
  vector<int> order(int n, vector<pii> &edges){
    adj.assign(n, vector<int>());
    for (pii p : edges)
      adj[p.first].push_back(p.second);
    visited.assign(n, false);
    vAns.clear();
    for (int i = 0; i < n; i++){
      if (!visited[i])
        dfs(i);
    }
    reverse(vAns.begin(), vAns.end());
    return vAns;
  }
}; // namespace TopologicalSort
```

2.19 Tree

```
#include "../data_structures/rmq.h"
// build: O(N), queries: O(1)
template<typename T> class Tree{
private:
  typedef pair<int, T> Edge;
  vector<vector<Edge>> adj;
  vector<int> v, level, in;
  vector<T> sum;
  RMQ<T> *rmq = nullptr;
  int n;
  void dfs(int u, int p, int d, T s){
    in[u] = v.size();
    v.push_back(u);
    level.push_back(d);
    sum[u] = s;
    for (auto [to, w] : adj[u]) if(to != p){
      dfs(to, u, d + 1, s + w);
      v.push_back(u);
      level.push_back(d);
    }
  }
public:
  ~Tree(){
    if(rmq != nullptr)
      delete rmq;
  }
  void init(int n1){
    n = n1;
    adj.assign(n, vector<Edge>());
    in.resize(n);
    sum.resize(n);
  }
  void addEdge(int a, int b, T w = 1){
    adj[a].emplace_back(b, w);
    adj[b].emplace_back(a, w);
  }
  void build(int root = 0){
    v.clear(); level.clear();
    dfs(root, -1, 0, 0);
    if(rmq != nullptr)
      delete rmq;
    rmq = new RMQ<int>(level);
  }
  //O(1)
  int lca(int a, int b){
    a = in[a], b = in[b];
    if(a > b)
      swap(a, b);
    return v[rmq->getPos(a, b)];
  }
  //O(1)
  T dist(int a, int b){
    return sum[a] + sum[b] - 2*sum[lca(a, b)];
  }
};
```

2.20 Tree ID

```
#include "centroid.h"
#define F first
#define S second
namespace TreeID{
  int id=0;
  map<map<int, int>, int> mpId;
  vector<int> adj[MAXN];
  int treeID(int u, int p){
    map<int, int> mp;
    for(int to: adj[u]){
      if(to != p)
        mp[treeID(to, u)]++;
    }
    if(!mpId.count(mp))
      mpId[mp] = ++id;
    return mpId[mp];
  }
  //Returns a pair of values that represents a tree only. O((N+M)*log(M))
  //0-indexed
  pii getTreeID(vector<pii> &edges, int n){
    for(int i=0; i<n; i++)
      adj[i].clear();
    Centroid::init(n);
    for(pii e: edges){
      adj[e.F].push_back(e.S);
      adj[e.S].push_back(e.F);
      Centroid::addEdge(e.F, e.S);
    }
    pii c = Centroid::findCentroid();
    pii ans(treeID(c.F, -1), treeID(c.S, -1));
    if(ans.F > ans.S)
      swap(ans.F, ans.S);
    return ans;
  }
  bool isomorphic(vector<pii> &tree1, vector<pii> &tree2, int n){
    return getTreeID(tree1, n) == getTreeID(tree2, n);
  }
};
```

2.21 Vertex Cover In Tree

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 200010;
int dp[MAXN][2];
vector<int> adj[MAXN];
// vertexCover(node current, free to choose, dad)
int vertexCover(int u, bool color=true, int p=-1){
  if(dp[u][color] != -1)
    return dp[u][color];
  int case1 = 1, case2 = 0;
  for(int to: adj[u]){
    if(to == p) continue;
    case1 += vertexCover(to, true, u);
    case2 += vertexCover(to, false, u);
  }
  if(color)
    return dp[u][color] = min(case1, case2);
  else
    return dp[u][color] = case1;
}
```

3 Dynamic Programming

3.1 Alien Trick

```cpp
#include <bits/stdc++.h>
#define F first
#define S second
using namespace std;
using ll = long long;
using pll = pair<ll, ll>;
pll solveDP(ll C);
ll solveMax(int k){
  ll lo = 0, hi=1e16, ans=1e16;
  while(lo <= hi){
    ll mid = (lo+hi)>>1;
    if(solveDP(mid).S <= k){
      ans = mid;
      hi = mid - 1;
    }else{
      lo = mid + 1;
    }
  }
  return solveDP(ans).F + k*ans;
}
```

3.2 Divide and Conquer Optimization Implementation

```cpp
#include <bits/stdc++.h>
using namespace std;
int C(int i, int j);
const int MAXN = 100010;
const int MAXK = 110;
const int INF = 0x3f3f3f3f;
int dp[MAXN][MAXK];
void calculateDP(int l, int r, int k, int opt_l, int opt_r){
  if (l > r)
    return;
  int mid = (l + r) >> 1;
  int ans = -INF, opt = mid;
//  int ans = dp[mid][k-1], opt=mid; //If you accept empty subsegment
  for (int i = opt_l; i <= min(opt_r, mid - 1); i++){
    if (ans < dp[i][k - 1] + C(i + 1, mid)){
      opt = i;
      ans = dp[i][k - 1] + C(i + 1, mid);
    }
  }
  dp[mid][k] = ans;
  calculateDP(l, mid - 1, k, opt_l, opt);
  calculateDP(mid + 1, r, k, opt, opt_r);
}
int solve(int n, int k){
  for (int i = 0; i <= n; i++)
    dp[i][0] = -INF;
  for (int j = 0; j <= k; j++)
    dp[0][j] = -INF;
  dp[0][0] = 0;
  for (int j = 1; j <= k; j++)
    calculateDP(1, n, j, 0, n - 1);
  return dp[n][k];
}
```

3.3 Knuth Optimization Implementation

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAXN = 1009;
const ll INFLL = 0x3f3f3f3f3f3f3f3f;
ll C(int a, int b);
ll dp[MAXN][MAXN];
int opt[MAXN][MAXN];
ll knuth(int n){
  for (int i = 0; i < n; i++){
    dp[i][i] = 0;
    opt[i][i] = i;
  }
  for (int s = 1; s < n; s++){
    for (int i = 0, j; (i + s) < n; i++){
      j = i + s;
      dp[i][j] = INFLL;
      for (int k = opt[i][j - 1]; k < min(j, opt[i + 1][j] + 1); k++){
        ll cur = dp[i][k] + dp[k + 1][j] + C(i, j);
        if (dp[i][j] > cur){
          dp[i][j] = cur;
          opt[i][j] = k;
        }
      }
    }
  }
  return dp[0][n - 1];
}
```

4 Math

4.1 Basic Math

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef unsigned long long ull;

ull fastPow(ull base, ull exp, ull mod){
  base %= mod;
  //exp %= phi(mod) if base and mod are relatively prime
  ull ans = 1LL;
  while (exp > 0){
    if (exp & 1LL)
      ans = (ans * (__int128_t)base) % mod;
    base = (base * (__int128_t)base) % mod;
    exp >>= 1;
  }
  return ans;
}
int fastPow(int base, string bigExp, int mod){
  int ans = 1;
  for(char c: bigExp){
    ans = fastPow(ans, 10, mod);
    ans = (ans*1LL*fastPow(base, c-'0', mod))%mod;
  }
  return ans;
}
//\sum_{i = 0}^{n - 1} floor((a * i + b)/m)
// 0 <= n <= 10^9
// 1 <= m <= 10^9
// 0 <= a, b < m
// O(log(a + b + c + d))
ll floor_sum(ll n, ll m, ll a, ll b) {
  ll ans = 0;
  if (a >= m) {
    ans += (n - 1) * n * (a / m) / 2;
    a %= m;
  }
  if (b >= m) {
    ans += n * (b / m);
    b %= m;
  }
  ll y_max = (a * n + b) / m, x_max = (y_max * m - b);
  if (y_max == 0) return ans;
  ans += (n - (x_max + a - 1) / a) * y_max;
  ans += floor_sum(y_max, a, m, (a - x_max % a) % a);
  return ans;
}
ll gcd(ll a, ll b){ return __gcd(a, b); }
ll lcm(ll a, ll b){ return (a / gcd(a, b)) * b; }
void enumeratingAllSubmasks(int mask){
  for (int s = mask; s; s = (s - 1) & mask)
    cout << s << endl;
}
//MOD to Hash
namespace ModHash{
  const uint64_t MOD = (1ll<<61) - 1;
  uint64_t modmul(uint64_t a, uint64_t b){
    uint64_t l1 = (uint32_t)a, h1 = a>>32, l2 = (uint32_t)b, h2 = b>>32;
    uint64_t l = l1*l2, m = l1*h2 + l2*h1, h = h1*h2;
    uint64_t ret = (l&MOD) + (l>>61) + (h << 3) + (m >> 29) + ((m << 35) >> 3)
          + 1;
    ret = (ret & MOD) + (ret>>61);
    ret = (ret & MOD) + (ret>>61);
    return ret-1;
  }
};
```

4.2 Binomial Coefficients

```
#include <bits/stdc++.h>
#include "./modular.h"
using namespace std;
typedef long long ll;
//O(k)
ll C1(int n, int k){
  ll res = 1LL;
  for (int i = 1; i <= k; ++i)
    res = (res * (n - k + i)) / i;
  return res;
}
//O(n^2)
vector<vector<ll>> C2(int maxn, int mod){
  vector<vector<ll>> mat(maxn + 1, vector<ll>(maxn + 1, 0));
  mat[0][0] = 1;
  for (int n = 1; n <= maxn; n++){
    mat[n][0] = mat[n][n] = 1;
    for (int k = 1; k < n; k++)
      mat[n][k] = (mat[n - 1][k - 1] + mat[n - 1][k]) % mod;
  }
  return mat;
}
//O(N)
vector<int> factorial, inv_factorial;
void prevC3(int maxn, int mod){
  factorial.resize(maxn + 1);
  factorial[0] = 1;
  for (int i = 1; i <= maxn; i++)
    factorial[i] = (factorial[i - 1] * 1LL * i) % mod;
  inv_factorial.resize(maxn + 1);
  inv_factorial[maxn] = fastPow(factorial[maxn], mod - 2, mod);
  for (int i = maxn - 1; i >= 0; i--)
    inv_factorial[i] = (inv_factorial[i + 1] * 1LL * (i + 1)) % mod;
}
int C3(int n, int k, int mod){
  if (n < k)
    return 0;
  return (((factorial[n] * 1LL * inv_factorial[k]) % mod) * 1LL *
      inv_factorial[n - k]) % mod;
}
//O(P*log(P))
//C4(n, k, p) = Comb(n, k)%p
vector<int> changeBase(int n, int p){
  vector<int> v;
  while (n > 0){
    v.push_back(n % p);
    n /= p;
  }
  return v;
}
int C4(int n, int k, int p){
  auto vn = changeBase(n, p);
  auto vk = changeBase(k, p);
  int mx = max(vn.size(), vk.size());
  vn.resize(mx, 0);
  vk.resize(mx, 0);
  prevC3(p - 1, p);
  int ans = 1;
  for (int i = 0; i < mx; i++)
    ans = (ans * 1LL * C3(vn[i], vk[i], p)) % p;
  return ans;
}
//O(P^k)
//C5(n, k, p, pk) = Comb(n, k)%(p^k)
int fat_p(ll n, int p, int pk){
  vector<int> fat1(pk, 1);
    int res = 1;
    for(int i=1; i<pk; i++){
    if(i%p == 0)
      fat1[i] = fat1[i-1];
    else
      fat1[i] = (fat1[i-1]*1LL*i)%pk;
  }
  while(n > 1){
    res = (res*1LL*fastPow(fat1[pk-1], n/pk, pk))%pk;
    res = (res*1LL*fat1[n%pk])%pk;
    n /= p;
  }
  return res;
}
ll cnt(ll n, int p){
  ll ans = 0;
  while(n > 1){
    ans += n/p;
    n/=p;
  }
  return ans;
}
int C5(ll n, ll k, int p, int pk){
  ll exp = cnt(n, p) - cnt(n-k, p) - cnt(k, p);
  int d = (fat_p(n-k, p, pk)*1LL*fat_p(k, p, pk))%pk;
  int ans = (fat_p(n, p, pk)*1LL*inv(d, pk))%pk;
  return (ans*1LL*fastPow(p, exp, pk))%pk;
}
```

4.3 Chinese Remainder Theorem

```
#include <bits/stdc++.h>
#include "extended_euclidean.h"
using namespace std;
typedef long long ll;
namespace CRT{
  inline ll normalize(ll x, ll mod){
    x %= mod;
    if (x < 0)
      x += mod;
    return x;
  }
  ll solve(vector<ll> a, vector<ll> m){
    int n = a.size();
    for (int i = 0; i < n; i++)
      normalize(a[i], m[i]);
    ll ans = a[0];
    ll lcm1 = m[0];
    for (int i = 1; i < n; i++){
      ll x, y;
      ll g = extGcd(lcm1, m[i], x, y);
      if ((a[i] - ans) % g != 0)
        return -1;
      ans = normalize(ans + ((((a[i] - ans) / g) * x) % (m[i] / g)) * lcm1, (
          lcm1 / g) * m[i]);
      lcm1 = (lcm1 / g) * m[i]; //lcm(lcm1, m[i]);
    }
    return ans;
  }
} // namespace CRT
```

4.4 Euler’s totient

```
#include <bits/stdc++.h>
using namespace std;
int nthPhi(int n){
  int result = n;
  for (int i = 2; i <= n / i; i++){
    if (n % i == 0){
      while (n % i == 0)
        n /= i;
      result -= result / i;
    }
  }
  if (n > 1)
    result -= result / n;
  return result;
}
vector<int> phiFrom1toN(int n){
  vector<int> vPhi(n + 1);
  vPhi[0] = 0;
  vPhi[1] = 1;
  for (int i = 2; i <= n; i++)
    vPhi[i] = i;
  for (int i = 2; i <= n; i++){
    if (vPhi[i] == i){
      for (int j = i; j <= n; j += i)
        vPhi[j] -= vPhi[j] / i;
    }
  }
  return vPhi;
}
```

4.5 Extended Euclidean

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll extGcd(ll a, ll b, ll &x, ll &y){
  if (b == 0){
    x = 1, y = 0;
    return a;
  }else{
    ll g = extGcd(b, a % b, y, x);
    y -= (a / b) * x;
    return g;
  }
}
//a*x + b*y = g
//a*(x-(b/g)*k) + b*(y+(a/g)*k) = g
bool dioEq(ll a, ll b, ll c, ll &x0, ll &y0, ll &g){
  g = extGcd(abs(a), abs(b), x0, y0);
  if (c % g) return false;
  x0 *= c / g;
  y0 *= c / g;
  if (a < 0) x0 = -x0;
  if (b < 0) y0 = -y0;
  return true;
}
inline void shift(ll &x, ll &y, ll a, ll b, ll cnt){
  x += cnt * b;
  y -= cnt * a;
}
// a1 + m1*x = a2 + m2*y
// Find the first moment that both are equal
```

```
ll findMinimum(ll a1, ll m1, ll a2, ll m2){
  ll a = m1, b = -m2, c = a2 - a1;
  ll x, y, g;
  if (!dioEq(a, b, c, x, y, g))
    return -1;
  a /= g;
  b /= g;
  int sa = a > 0 ? +1 : -1;
  int sb = b > 0 ? +1 : -1;
  shift(x, y, a, b, -x/b);
  if(x < 0)
    shift(x, y, a, b, sb);
  if(y < 0){
    shift(x, y, a, b, y/a);
    if(y < 0)
      shift(x, y, a, b, -sa);
    if(x < 0)
      return -1;
  }
  return a*x*g;
}
ll findAllSolutions(ll a, ll b, ll c, ll minx, ll maxx, ll miny, ll maxy){
  ll x, y, g;
  if(a==0 or b==0){
    if(a==0 and b==0)
      return (c==0)*(maxx-minx+1)*(maxy-miny+1);
    if(a == 0)
      return (c%b == 0)*(maxx-minx+1)*(miny<=c/b and c/b<=maxy);
    return (c%a == 0)*(minx<=c/a and c/a<=maxx)*(maxy-miny+1);
  }
  if (!dioEq(a, b, c, x, y, g))
    return 0;
  a /= g;
  b /= g;
  int sign_a = a > 0 ? +1 : -1;
  int sign_b = b > 0 ? +1 : -1;
  shift(x, y, a, b, (minx - x) / b);
  if (x < minx)
    shift(x, y, a, b, sign_b);
  if (x > maxx)
    return 0;
  ll lx1 = x;
  shift(x, y, a, b, (maxx - x) / b);
  if (x > maxx)
    shift(x, y, a, b, -sign_b);
  ll rx1 = x;
  shift(x, y, a, b, -(miny - y) / a);
  if (y < miny)
    shift(x, y, a, b, -sign_a);
  if (y > maxy)
    return 0;
  ll lx2 = x;
  shift(x, y, a, b, -(maxy - y) / a);
  if (y > maxy)
    shift(x, y, a, b, sign_a);
  ll rx2 = x;
  if (lx2 > rx2)
    swap(lx2, rx2);
  ll lx = max(lx1, lx2);
  ll rx = min(rx1, rx2);
  if (lx > rx)
    return 0;
  return (rx - lx) / abs(b) + 1;
}
```

4.6 Fraction

```
#include <bits/stdc++.h>
using namespace std;
typedef long long f_type;
//Representation of the a/b
struct Fraction {
  f_type a, b;
  Fraction(f_type _a = 0): a(_a), b(1){}
  Fraction(f_type _a, f_type _b) {
    f_type g = __gcd(_a, _b);
    a = _a/g;
    b = _b/g;
    if(b < 0){
      a = -a;
      b = -b;
    }
  }
  Fraction operator+(Fraction oth) {
    return Fraction(a*oth.b + oth.a*b, b*oth.b);
  }
  Fraction operator-(Fraction oth) {
    return Fraction(a*oth.b - oth.a*b, b*oth.b);
  }
  Fraction operator*(Fraction oth) {
    return Fraction(a*oth.a, b*oth.b);
  }
  Fraction operator/(Fraction oth) {
    return Fraction(a*oth.b, b*oth.a);
  }
  bool operator>=(Fraction oth){
    return ((*this) - oth).a >= 0;
  }
  bool operator==(Fraction oth){
    return a == oth.a and b == oth.b;
  }
  operator f_type() {return a/b;}
  operator double() {return double(a)/b;}
};
```

4.7 FFT

```
#include <bits/stdc++.h>
using namespace std;
struct complex_t {
  double a {0.0}, b {0.0};
  complex_t(){}
  complex_t(double na) : a{na}{}
  complex_t(double na, double nb) : a{na}, b{nb} {}
  const complex_t operator+(const complex_t &c) const {
    return complex_t(a + c.a, b + c.b);
  }
  const complex_t operator-(const complex_t &c) const {
    return complex_t(a - c.a, b - c.b);
  }
  const complex_t operator*(const complex_t &c) const {
    return complex_t(a*c.a - b*c.b, a*c.b + b*c.a);
  }
  const complex_t operator/(const int &c) const {
    return complex_t(a/c, b/c);
  }
};
//using cd = complex<double>;
using cd = complex_t;
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 = w * wlen;
      }
    }
  }
  if (invert){
    for (cd &x : a)
      x = x / n;
  }
}
typedef long long ll;
vector<ll> multiply(vector<int> &a, vector<int> &b) {
  vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
  int n = 1;
  while(n < int(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] = fa[i]*fb[i];
  fft(fa, true);
  vector<ll> result(n);
  for (int i = 0; i < n; i++)
    result[i] = ll(fa[i].a + 0.5);
  return result;
}
vector<ll> scalarProdut(vector<int> t, vector<int> p, bool isCyclic=false) {
  int nt = t.size();
  int np = p.size();
  t.resize(nt+np, 0);
  reverse(p.begin(), p.end());
  if(isCyclic)
    for(int i=nt; i<nt+np; i++)
      t[i] = t[i%nt];
  vector<ll> ans = multiply(t, p);
  for(int i=0; i<nt; i++)
    ans[i] = ans[np-1+i];
  ans.resize(nt);
  return ans;
}
inline int getID(char c){
  return c - 'a';
}
// Find p in text t. Wildcard character *
vector<bool> stringMatchingWithWildcards(string t, string p){
  int nt = t.size();
  int np = p.size();
  vector<cd> fa(nt), fb(np);
  for(int i=0; i<nt; i++){
    double apha = (2*PI*getID(t[i]))/26;
    fa[i] = cd(cos(apha), sin(apha));
  }
  reverse(p.begin(), p.end());
  int k = 0;
  for(int i=0; i<np; i++){
    if(p[i] != '*'){
      double apha = (2*PI*getID(p[i]))/26;
      fb[i] = cd(cos(apha), -sin(apha));
      k++;
    }else{
      fb[i] = cd(0, 0);
    }
  }
  int n = 1;
  while(n < int(nt + np) )
    n <<= 1;
  fa.resize(n);
  fb.resize(n);
  fft(fa, false);
  fft(fb, false);
  for (int i = 0; i < n; i++)
    fa[i] = fa[i]*fb[i];
  fft(fa, true);
  vector<bool> result(nt - np+1);
  for (int i = 0; i < (nt - np+1); i++)
    result[i] = (int(fa[np-1+i].a + 1e-9) == k);
  return result;
}
```

4.8 Gauss

```
#include <bits/stdc++.h>
using namespace std;
const int INF = 0x3f3f3f3f;
typedef long double ld;
const ld EPS = 1e-9;
int gauss(vector<vector<ld>> a, vector<ld> &ans) {
  int n = (int) a.size();
  int m = (int) 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) {
        ld 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++) {
    ld 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;
}
```

4.9 Gauss Xor

```
#include <bits/stdc++.h>
using namespace std;
const int MAXB = 30;
struct GaussXOR {
  int table[MAXB];
  GaussXOR() {
    for(int i = 0; i < MAXB; i++) {
      table[i] = 0;
    }
  }
  int size() {
    int ans = 0;
    for(int i = 0; i < MAXB; i++) {
      if(table[i]) ans++;
    }
    return ans;
  }
  bool isComb(int x) {
    for(int i = MAXB-1; i >= 0; i--) {
      x = std::min(x, x ^ table[i]);
    }
    return x == 0;
  }
  void add(int x) {
    for(int i = MAXB-1; i >= 0; i--) {
      if((table[i] == 0) and ((x>>i) & 1)){
        table[i] = x;
        x = 0;
      } else {
        x = std::min(x, x ^ table[i]);
      }
    }
  }
  int max(){
    int ans = 0;
    for(int i = MAXB-1; i >= 0; i--) {
      ans = std::max(ans, ans ^ table[i]);
    }
    return ans;
  }
};
```

4.10 Montgomery Multiplication

```
#include <bits/stdc++.h>
using namespace std;
using u64 = uint64_t;
using u128 = __uint128_t;
using i128 = __int128_t;
struct u256{
  u128 high, low;
  static u256 mult(u128 x, u128 y){
    u64 a = x >> 64, b = x;
    u64 c = y >> 64, d = y;
    u128 ac = (u128)a * c;
    u128 ad = (u128)a * d;
    u128 bc = (u128)b * c;
    u128 bd = (u128)b * d;
    u128 carry = (u128)(u64)ad + (u128)(u64)bc + (bd >> 64u);
    u128 high = ac + (ad >> 64u) + (bc >> 64u) + (carry >> 64u);
    u128 low = (ad << 64u) + (bc << 64u) + bd;
    return {high, low};
  }
};
//x_m := x*r mod n
struct Montgomery{
  u128 mod, inv, r2;
  //the N will be an odd number
  Montgomery(u128 n) : mod(n), inv(1), r2(-n % n){
    for (int i = 0; i < 7; i++)
      inv *= 2 - n * inv;
    for (int i = 0; i < 4; i++){
      r2 <<= 1;
      if (r2 >= mod)
        r2 -= mod;
    }
    for (int i = 0; i < 5; i++)
      r2 = mult(r2, r2);
  }
  u128 init(u128 x){
    return mult(x, r2);
  }
  u128 reduce(u256 x){
    u128 q = x.low * inv;
    i128 a = x.high - u256::mult(q, mod).high;
    if (a < 0)
      a += mod;
    return a;
  }
  u128 mult(u128 a, u128 b){
    return reduce(u256::mult(a, b));
  }
};
```

4.11 NTT

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MOD = 998244353;
inline int modMul(int a, int b) {
  return (int) ((a*(ll)b) % MOD);
}
namespace ntt {
  int base = 1;
  vector<int> roots = {0, 1};
  vector<int> rev = {0, 1};
  int max_base = -1;
  int root = -1;
  inline int power(int a, long long b) {
    int res = 1;
    while (b > 0) {
      if (b & 1)
        res = modMul(res, a);
      a = modMul(a, a);
      b >>= 1;
    }
    return res;
  }
  inline int inv(int a) {
    a %= MOD;
    if (a < 0) a += MOD;
    int b = MOD, u = 0, v = 1;
    while(a){
      int t = b / a;
      b -= t * a; swap(a, b);
      u -= t * v; swap(u, v);
    }
    assert(b == 1);
    if (u < 0) u += MOD;
    return u;
  }
  void init() {
    int tmp = MOD - 1;
    max_base = 0;
    while (tmp % 2 == 0) {
      tmp /= 2;
      max_base++;
    }
    root = 2;
    while (true) {
      if (power(root, 1 << max_base) == 1) {
        if (power(root, 1 << (max_base - 1)) != 1) {
          break;
        }
      }
      root++;
    }
  }
  void ensure_base(int nbase) {
    if (max_base == -1)
      init();
    if (nbase <= base)
      return;
    assert(nbase <= max_base);
    rev.resize(1 << nbase);
    for (int i = 0; i < (1 << nbase); i++)
      rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
    roots.resize(1 << nbase);
    while (base < nbase) {
      int z = power(root, 1 << (max_base - 1 - base));
      for (int i = 1 << (base - 1); i < (1 << base); i++) {
        roots[i << 1] = roots[i];
        roots[(i << 1) + 1] = modMul(roots[i], z);
      }
      base++;
    }
  }
  void fft(vector<int> &a) {
    int n = (int) a.size();
    assert((n & (n - 1)) == 0);
    int zeros = __builtin_ctz(n);
    ensure_base(zeros);
    int shift = base - zeros;
    for (int i = 0; i < n; i++) {
      if (i < (rev[i] >> shift)) {
        swap(a[i], a[rev[i] >> shift]);
      }
    }
    for (int k = 1; k < n; k <<= 1) {
      for (int i = 0; i < n; i += 2 * k) {
        for (int j = 0; j < k; j++) {
          int x = a[i + j];
          int y = modMul(a[i + j + k], roots[j + k]);
          a[i + j] = x + y - MOD;
          if (a[i + j] < 0) a[i + j] += MOD;
          a[i + j + k] = x - y + MOD;
          if (a[i + j + k] >= MOD) a[i + j + k] -= MOD;
        }
      }
    }
  }
  vector<int> multiply(vector<int> a, vector<int> b, int eq = 0) {
    int need = (int) (a.size() + b.size() - 1);
    int nbase = 0;
    while ((1 << nbase) < need) nbase++;
    ensure_base(nbase);
    int sz = 1 << nbase;
    a.resize(sz);
    b.resize(sz);
    fft(a);
    if (eq)
      b = a;
    else
      fft(b);
    int inv_sz = inv(sz);
    for (int i = 0; i < sz; i++)
      a[i] = modMul(modMul(a[i], b[i]), inv_sz);
    reverse(a.begin() + 1, a.end());
    fft(a);
    a.resize(need);
    return a;
  }
  vector<int> square(vector<int> a) {
    return multiply(a, a, 1);
  }
  vector<int> pow(vector<int> a, ll e){
    int need = (int) ( (a.size()-1)*e + 1);
    int nbase = 0;
    while ((1 << nbase) < need) nbase++;
    ensure_base(nbase);
    int sz = 1 << nbase;
    a.resize(sz);
    fft(a);
    int inv_sz = ntt::inv(sz);
    for (int i = 0; i < sz; i++)
      a[i] = modMul(power(a[i], e), inv_sz);
    reverse(a.begin() + 1, a.end());
    fft(a);
    a.resize(need);
    return a;
  }
  vector<int> pow(vector<int> a, ll exp, int maxSize){
    vector<int> ans(1, 1);
    ans.resize(maxSize, 0);
    a.resize(maxSize, 0);
    while(exp > 0){
      if(exp & 1LL)
        ans = multiply(ans, a);
      a = square(a);
      exp >>= 1;
      ans.resize(maxSize, 0);
      a.resize(maxSize, 0);
    }
    return ans;
  }
};
```

4.12 Prime Number

```
#include <bits/stdc++.h>
#include "basic_math.h"
using namespace std;
typedef unsigned long long ull;
ull modMul(ull a, ull b, ull mod){
  return (a * (__uint128_t)b) % mod;
}
bool checkComposite(ull n, ull a, ull d, int s){
  ull x = fastPow(a, d, n);
  if (x == 1 or x == n - 1)
    return false;
  for (int r = 1; r < s; r++){
    x = modMul(x, x, n);
    if (x == n - 1LL)
      return false;
  }
  return true;
};
bool millerRabin(ull n){
  if (n < 2)
    return false;
  int r = 0;
  ull d = n - 1LL;
  while ((d & 1LL) == 0){
    d >>= 1;
    r++;
  }
  for (ull a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}){
    if (n == a)
      return true;
    if (checkComposite(n, a, d, r))
      return false;
  }
  return true;
}
ull pollard(ull n){
  auto f = [n](ull x) { return modMul(x, x, n) + 1; };
  ull x = 0, y = 0, t = 0, prd = 2, i = 1, q;
  while (t++ % 40 || __gcd(prd, n) == 1){
    if (x == y)
      x = ++i, y = f(x);
    if ((q = modMul(prd, max(x, y) - min(x, y), n)))
      prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}
vector<ull> factor(ull n){
  if (n == 1)
    return {};
  if (millerRabin(n))
    return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), r.begin(), r.end());
  return l;
}
```

4.13 Sieve And Primes

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll ns;
int np;
bitset<10000010> bs;
vector<ll> primes;
void sieve(ll l) {
  ns = l+1;
  bs.set();
  primes.clear();
  bs[0] = bs[1] = 0;
  for (ll i = 2; i < ns; i++) if (bs[i]) {
    for(ll j = i*i; j < ns; j += i)
      bs[j] = 0;
    primes.push_back(i);
  }
  np = primes.size();
}
bool isPrime(ll n) {
  if(n < ns)
    return bs[n];
  for(ll p: primes){
    if(p*p > n) break;
    if(n%p == 0)
      return false;
  }
  return true;
}
vector<ll> primeFactors(ll n) {
  vector<ll> factors;
  for(ll p: primes){
    if(p*p > n) break;
    while(n%p == 0LL) {
      n /= p;
      factors.push_back(p);
    }
  }
  if(n != 1LL) factors.push_back(n);
  return factors;
}
ll numDiv(ll n) {
  ll ans = 1;
  for(ll p: primes){
    if(p*p > n) break;
    ll f = 0;
    while(n%p == 0LL) {
      n /= p;
      f++;
    }
    ans *= (f+1LL);
  }
  return (n != 1LL) ? 2LL*ans : ans;
}
ll sumDiv(ll n) {
  ll ans = 1;
  for(ll p: primes){
    if(p*p > n) break;
    ll power = p;
    while(n%p == 0LL) {
      n /= p;
      power *= p;
    }
    ans *= (power - 1LL)/(p - 1LL);
  }
  if(n != 1LL)
    ans *= (n*n - 1LL)/(n - 1LL);
  return ans;
}
int mobius[1000010];
void sieveMobius(ll l) {
  sieve(l);
  mobius[1] = 1;
  for(int i=2; i<=l; i++)
    mobius[i] = 0;
  for(ll p: primes){
    if(p > l) break;
    for(ll j = p; j <= l; j += p){
      if(mobius[j] != -1){
        mobius[j]++;
        if(j%(p*p) == 0)
          mobius[j] = -1;
      }
    }
  }
  for(int i=2; i<=l; i++){
    if(mobius[i] == -1)
      mobius[i] = 0;
    else if(mobius[i]%2 == 0)
      mobius[i] = 1;
    else
      mobius[i] = -1;
  }
}
```

5 Geometry

5.1 Basic Geometry

```
#include <bits/stdc++.h>
using namespace std;
#define POINT_DOUBLE
#ifdef POINT_DOUBLE
  // Se necessario, apelar para __float128
  typedef double ftype;
  typedef long double ftLong;
  const double EPS = 1e-9;
  #define eq(a, b) (abs((a) - (b)) < EPS)
  #define lt(a, b) (((a) + EPS) < (b))
  #define gt(a, b) ((a) > ((b) + EPS))
  #define le(a, b) ((a) < ((b) + EPS))
  #define ge(a, b) (((a) + EPS) > (b))
#else
  typedef int32_t ftype;
  typedef int64_t ftLong;
  #define eq(a, b) ((a) == (b))
  #define lt(a, b) ((a) < (b))
  #define gt(a, b) ((a) > (b))
  #define le(a, b) ((a) <= (b))
  #define ge(a, b) ((a) >= (b))
#endif
//Begin Point 2D
struct Point2d{
  ftype x, y;
  Point2d() {}
  Point2d(ftype x1, ftype y1) : x(x1), y(y1) {}
  Point2d operator+(const Point2d &t){
    return Point2d(x + t.x, y + t.y);
  }
  Point2d operator-(const Point2d &t){
    return Point2d(x - t.x, y - t.y);
  }
  Point2d operator*(ftype t){
    return Point2d(x * t, y * t);
  }
  Point2d operator/(ftype t){
    return Point2d(x / t, y / t);
  }
  bool operator<(const Point2d &o) const{
    return lt(x, o.x) or (eq(x, o.x) and lt(y, o.y));
  }
  bool operator==(const Point2d &o) const{
    return eq(x, o.x) and eq(y, o.y);
  }
  friend std::istream& operator >> (std::istream &is, Point2d &p) {
    return is >> p.x >> p.y;
  }
  friend std::ostream& operator << (std::ostream &os, const Point2d &p) {
    return os << p.x << ’ ’ << p.y;
  }
};
ftLong pw2(ftype a){
  return a * (ftLong)a;
}
//Scalar product
ftLong dot(Point2d a, Point2d b){
  return a.x*(ftLong)b.x + a.y*(ftLong)b.y;
}
ftLong norm(Point2d a){
  return dot(a, a);
}
double len(Point2d a){
  return sqrtl(dot(a, a));
}
double dist(Point2d a, Point2d b){
  return len(a - b);
}
//Vector product
ftLong cross(Point2d a, Point2d b){
  return a.x * (ftLong)b.y - a.y * (ftLong)b.x;
}
//Projection size from A to B
double proj(Point2d a, Point2d b){
  return dot(a, b) / len(b);
}
//The angle between A and B
double angle(Point2d a, Point2d b){
  return acos(dot(a, b) / len(a) / len(b));
}
//Left rotation. Angle in radian
Point2d rotateL(Point2d p, double ang){
  return Point2d(p.x * cos(ang) - p.y * sin(ang), p.x * sin(ang) + p.y * cos(
      ang));
}
//90 degree left rotation
Point2d perpL(Point2d a){
  return Point2d(-a.y, a.x);
}
//0-> 1o,2o quadrant, 1-> 3o,4o
int half(Point2d &p){
  if (gt(p.y, 0) or (eq(p.y, 0) and ge(p.x, 0)))
    return 0;
  else
    return 1;
}
//angle(a) < angle(b)
bool cmpByAngle(Point2d a, Point2d b){
  int ha = half(a), hb = half(b);
  if (ha != hb){
    return ha < hb;
  }else{
    ftLong c = cross(a, b);
    if(eq(c, 0))
      return lt(norm(a), norm(b));
    else
      return gt(c, 0);
  }
}
inline int sgn(ftLong x){
  return ge(x, 0) ? (eq(x, 0) ? 0 : 1) : -1;
}
//-1: angle(a, b) < angle(b, c)
// 0: angle(a, b) = angle(b, c)
//+1: angle(a, b) > angle(b, c)
int cmpAngleBetweenVectors(Point2d a, Point2d b, Point2d c){
  ftLong dotAB = dot(a, b), dotBC = dot(b, c);
  int sgnAB = sgn(dotAB), sgnBC = sgn(dotBC);
  if(sgnAB == sgnBC){
    //Careful with overflow
    ftLong l = pw2(dotAB)*dot(c, c), r = pw2(dotBC)*dot(a, a);
    if(l == r)
      return 0;
    if(sgnAB == 1)
      return gt(l, r)? -1 : +1;
    return lt(l, r)? -1 : +1;
  }else{
    return (sgnAB > sgnBC)? -1 : +1;
  }
}
//Line parameterized: r1 = a1 + d1*t
//This function can be generalized to 3D
Point2d intersect(Point2d a1, Point2d d1, Point2d a2, Point2d d2){
  return a1 + d1 * (cross(a2 - a1, d2) / cross(d1, d2));
}
//Distance between the point(a) and segment(ps1, ps2)
//This function can be generalized to 3D
ftLong distance_point_to_segment(Point2d a, Point2d ps1, Point2d ps2) {
  if(ps1 == ps2)
    return dist(ps1, a);
  Point2d d = ps2 - ps1;
  ftLong t = max(ftLong(0), min(ftLong(1), ftLong(dot(a-ps1, d)/len(d))));
  Point2d proj = ps1 + Point2d(d.x*t, d.y*t);
  return dist(a, proj);
}
//Distance between the point(a) and line(pl1, pl2)
//This function can be generalized to 3D
double dist(Point2d a, Point2d pl1, Point2d pl2){
  //crs = parallelogram area
  double crs = cross(Point2d(a - pl1), Point2d(pl2 - pl1));
  //h = area/base
  return abs(crs / dist(pl1, pl2));
}
long double area(vector<Point2d> p){
  long double ret = 0;
  for (int i = 2; i < (int)p.size(); i++)
    ret += cross(p[i] - p[0], p[i - 1] - p[0]) / 2.0;
  return abs(ret);
}
long long latticePointsInSeg(Point2d a, Point2d b){
  long long dx = abs(a.x - b.x);
  long long dy = abs(a.y - b.y);
  return gcd(dx, dy) + 1;
}
ftLong signed_area_parallelogram(Point2d p1, Point2d p2, Point2d p3){
  return cross(p2 - p1, p3 - p2);
}
long double triangle_area(Point2d p1, Point2d p2, Point2d p3){
  return abs(signed_area_parallelogram(p1, p2, p3)) / 2.0;
}
bool pointInTriangle(Point2d a, Point2d b, Point2d c, Point2d p){
  ftLong s1 = abs(cross(b - a, c - a));
  ftLong s2 = abs(cross(a - p, b - p)) + abs(cross(b - p, c - p)) + abs(cross(
       c - p, a - p));
  return eq(s1, s2);
}
bool clockwise(Point2d p1, Point2d p2, Point2d p3){
  return lt(signed_area_parallelogram(p1, p2, p3), 0);
}
bool counter_clockwise(Point2d p1, Point2d p2, Point2d p3){
  return gt(signed_area_parallelogram(p1, p2, p3), 0);
}
//End Point 2D

//Begin Line
ftLong det(ftype a, ftype b, ftype c, ftype d){
  return a * (ftLong)d - b * (ftLong)c;
}
struct Line{
  ftype a, b, c;
  Line() {}
  Line(ftype a1, ftype b1, ftype c1) : a(a1), b(b1), c(c1){
    normalize();
  }
  Line(Point2d p1, Point2d p2){
    a = p1.y - p2.y;
    b = p2.x - p1.x;
    c = -a * p1.x - b * p1.y;
    normalize();
  }
  void normalize(){
#ifdef POINT_DOUBLE
    ftype z = sqrt(pw2(a) + pw2(b));
#else
    ftype z = __gcd(abs(a), __gcd(abs(b), abs(c)));
#endif
    if(eq(z, 0)) return;
    a /= z;
    b /= z;
    c /= z;
    if (lt(a, 0) or (eq(a, 0) and lt(b, 0))){
      a = -a;
      b = -b;
      c = -c;
    }
  }
```

```
};
bool intersect(Line m, Line n, Point2d &res){
  ftype zn = det(m.a, m.b, n.a, n.b);
  if (eq(zn, 0))
    return false;
  res.x = -det(m.c, m.b, n.c, n.b) / zn;
  res.y = -det(m.a, m.c, n.a, n.c) / zn;
  return true;
}
bool parallel(Line m, Line n){
  return eq(det(m.a, m.b, n.a, n.b), 0);
}
bool equivalent(Line m, Line n){
  return eq(det(m.a, m.b, n.a, n.b), 0) &&
         eq(det(m.a, m.c, n.a, n.c), 0) &&
         eq(det(m.b, m.c, n.b, n.c), 0);
}
//Distance from a point(x, y) to a line m
double dist(Line m, ftype x, ftype y){
  return abs(m.a * (ftLong)x + m.b * (ftLong)y + m.c) /
         sqrt(m.a * (ftLong)m.a + m.b * (ftLong)m.b);
}
//End Line

//Begin Segment
struct Segment{
  Point2d a, b;
  Segment() {}
  Segment(Point2d a1, Point2d b1) : a(a1), b(b1) {}
};
bool inter1d(ftype a, ftype b, ftype c, ftype d){
  if (gt(a, b)) swap(a, b);
  if (gt(c, d)) swap(c, d);
  return le(max(a, c), min(b, d));
}
bool check_intersection(Segment s1, Segment s2){
  Point2d a = s1.a, b = s1.b, c = s2.a, d = s2.b;
  if (eq(cross(a - c, d - c), 0) && eq(cross(b - c, d - c), 0))
    return inter1d(a.x, b.x, c.x, d.x) && inter1d(a.y, b.y, c.y, d.y);
  return sgn(cross(b - a, c - a)) != sgn(cross(b - a, d - a)) &&
         sgn(cross(d - c, a - c)) != sgn(cross(d - c, b - c));
}
inline bool betw(ftype l, ftype r, ftype x){
  return le(min(l, r), x) and le(x, max(l, r));
}
bool intersect(Segment s1, Segment s2, Segment &ans){
  Point2d a = s1.a, b = s1.b, c = s2.a, d = s2.b;
  if (!inter1d(a.x, b.x, c.x, d.x) || !inter1d(a.y, b.y, c.y, d.y))
    return false;
  Line m(a, b);
  Line n(c, d);
  if (parallel(m, n)){
    if (!equivalent(m, n))
      return false;
    if (b < a)
      swap(a, b);
    if (d < c)
      swap(c, d);
    ans = Segment(max(a, c), min(b, d));
    return true;
  }else{
```

```cpp
    Point2d p(0, 0);
    intersect(m, n, p);
    ans = Segment(p, p);
    return betw(a.x, b.x, p.x) && betw(a.y, b.y, p.y) &&
           betw(c.x, d.x, p.x) && betw(c.y, d.y, p.y);
  }
}
//End Segment

//Begin Circle
struct Circle{
  ftype x, y, r;
  Circle() {}
  Circle(ftype x1, ftype y1, ftype r1) : x(x1), y(y1), r(r1){};
};
bool pointInCircle(Circle c, Point2d p){
  return ge(c.r, dist(Point2d(c.x, c.y), p));
}
//CircumCircle of a triangle is a circle that passes through all the vertices
Circle circumCircle(Point2d a, Point2d b, Point2d c){
  Point2d u((b - a).y, -((b - a).x));
  Point2d v((c - a).y, -((c - a).x));
  Point2d n = (c - b) * 0.5;
  double t = cross(u, n) / cross(v, u);
  Point2d ct = (((a + c) * 0.5) + (v * t));
  double r = dist(ct, a);
  return Circle(ct.x, ct.y, r);
}
//InCircle is the largest circle contained in the triangle
Circle inCircle(Point2d a, Point2d b, Point2d c){
  double m1 = dist(a, b);
  double m2 = dist(a, c);
  double m3 = dist(b, c);
  Point2d ct = ((c * m1) + (b * m2) + a * (m3)) / (m1 + m2 + m3);
  double sp = 0.5 * (m1 + m2 + m3);
  double r = sqrt(sp * (sp - m1) * (sp - m2) * (sp - m3)) / sp;
  return Circle(ct.x, ct.y, r);
}
//Minimum enclosing circle, O(n)
Circle minimumCircle(vector<Point2d> p){
  random_shuffle(p.begin(), p.end());
  Circle c = Circle(p[0].x, p[0].y, 0.0);
  for (int i = 0; i < (int)p.size(); i++){
    if (pointInCircle(c, p[i]))
      continue;
    c = Circle(p[i].x, p[i].y, 0.0);
    for (int j = 0; j < i; j++){
      if (pointInCircle(c, p[j]))
        continue;
      c = Circle((p[j].x + p[i].x) * 0.5, (p[j].y + p[i].y) * 0.5, 0.5 * dist(
          p[j], p[i]));
      for (int k = 0; k < j; k++){
        if (pointInCircle(c, p[k]))
          continue;
        c = circumCircle(p[j], p[i], p[k]);
      }
    }
  }
  return c;
}
//Return the number of the intersection
```

```cpp
int circle_line_intersection(Circle circ, Line line, Point2d &p1, Point2d &p2)
    {
  ftLong r = circ.r;
  ftLong a = line.a, b = line.b, c = line.c + line.a * circ.x + line.b * circ.
      y; //take a circle to the (0, 0)
  ftLong x0 = -a * c / (pw2(a) + pw2(b)), y0 = -b * c / (pw2(a) + pw2(b));
            //(x0, y0) is the shortest distance point of the line for (0, 0)
  if (gt(pw2(c), pw2(r) * (pw2(a) + pw2(b)))){
    return 0;
  }
  else if (eq(pw2(c), pw2(r) * (pw2(a) + pw2(b)))){
    p1.x = p2.x = x0 + circ.x;
    p1.y = p2.y = y0 + circ.y;
    return 1;
  }else{
    ftLong d_2 = pw2(r) - pw2(c) / (pw2(a) + pw2(b));
    ftLong mult = sqrt(d_2 / (pw2(a) + pw2(b)));
    p1.x = x0 + b * mult + circ.x;
    p2.x = x0 - b * mult + circ.x;
    p1.y = y0 - a * mult + circ.y;
    p2.y = y0 + a * mult + circ.y;
    return 2;
  }
}
//Return the number of the intersection
int circle_intersection(Circle c1, Circle c2, Point2d &p1, Point2d &p2){
  if (eq(c1.x, c2.x) and eq(c1.y, c2.y)){
    if (eq(c1.r, c2.r))
      return -1; //INF
    else
      return 0;
  }else{
    Circle circ(0, 0, c1.r);
    Line line;
    line.a = -2 * (c2.x - c1.x);
    line.b = -2 * (c2.y - c1.y);
    line.c = pw2(c2.x - c1.x) + pw2(c2.y - c1.y) + pw2(c1.r) - pw2(c2.r);
    int sz = circle_line_intersection(circ, line, p1, p2);
    p1.x += c1.x;
    p2.x += c1.x;
    p1.y += c1.y;
    p2.y += c1.y;
    return sz;
  }
}

bool checkIfTheSegmentIsCompletelyCoveredByCircles(vector<Circle> &vc, Segment
     s){
  vector<Point2d> v = {s.a, s.b};
  Line l(s.a, s.b);
  for (Circle c : vc){
    Point2d p1, p2;
    int inter = circle_line_intersection(c, l, p1, p2);
    if (inter >= 1 and betw(s.a.x, s.b.x, p1.x) and betw(s.a.y, s.b.y, p1.y))
      v.push_back(p1);
    if (inter == 2 and betw(s.a.x, s.b.x, p2.x) and betw(s.a.y, s.b.y, p2.y))
      v.push_back(p2);
  }
  sort(v.begin(), v.end());
  bool ans = true;
  for (int i = 1; i < (int)v.size(); i++){
    bool has = false;
    for (Circle c : vc){
      if (pointInCircle(c, v[i - 1]) and pointInCircle(c, v[i])){
        has = true;
        break;
      }
    }
    ans = ans && has;
  }
  return ans;
}

void tangents(Point2d c, double r1, double r2, vector<Line> &ans){
  double r = r2 - r1;
  double z = pw2(c.x) + pw2(c.y);
  double d = z - pw2(r);
  if (lt(d, 0))
    return;
  d = sqrt(abs(d));
  Line l;
  l.a = (c.x * r + c.y * d) / z;
  l.b = (c.y * r - c.x * d) / z;
  l.c = r1;
  ans.push_back(l);
}
vector<Line> tangents(Circle a, Circle b){
  vector<Line> ans;
  for (int i = -1; i <= 1; i += 2)
    for (int j = -1; j <= 1; j += 2)
      tangents(Point2d(b.x - a.x, b.y - a.y), a.r * i, b.r * j, ans);
  for (size_t i = 0; i < ans.size(); ++i){
    ans[i].c -= ans[i].a * a.x + ans[i].b * a.y;
    ans[i].normalize();
  }
  return ans;
}
//End Circle
```

5.2 Circle Area Union

```
#include "basic_geometry.h"
using namespace std;

const double PI = acos(-1);
pair<double, double> isCC(Circle circ1, Circle circ2){
  Point2d c1(circ1.x, circ1.y), c2(circ2.x, circ2.y);
  double r1 = circ1.r, r2 = circ2.r;
  double d = dist(c1, c2);
  double x1 = c1.x, x2 = c2.x, y1 = c1.y, y2 = c2.y;
  double mid = atan2(y2 - y1, x2 - x1);
  double a = r1, c = r2;
  double t = acos((a * a + d * d - c * c) / (2 * a * d));
  return make_pair(mid - t, mid + t);
}
int testCC(Circle circ1, Circle circ2){
  Point2d c1(circ1.x, circ1.y), c2(circ2.x, circ2.y);
  double r1 = circ1.r, r2 = circ2.r;
  double d = dist(c1, c2);
  if (le(r1 + r2, d))
    return 1; // not intersected or tged
  if (le(r1 + d, r2))
    return 2; // C1 inside C2
  if (le(r2 + d, r1))
    return 3; // C2 inside C1
  return 0;   // intersected
}
struct event_t{
  double theta;
  int delta;
  event_t(double t, int d) : theta(t), delta(d) {}
  bool operator<(const event_t &r) const{
    if (fabs(theta - r.theta) < EPS)
      return delta > r.delta;
    return theta < r.theta;
  }
};
vector<event_t> e;
void add(double begin, double end){
  if (begin <= -PI)
    begin += 2 * PI, end += 2 * PI;
  if (end > PI){
    e.push_back(event_t(begin, 1));
    e.push_back(event_t(PI, -1));
    e.push_back(event_t(-PI, 1));
    e.push_back(event_t(end - 2 * PI, -1));
  }else{
    e.push_back(event_t(begin, 1));
    e.push_back(event_t(end, -1));
  }
}
double calc(Point2d c, double r, double a1, double a2){
  double da = a2 - a1;
  double aa = r * r * (da - sin(da)) / 2;
  Point2d p1 = Point2d(cos(a1), sin(a1)) * r + c;
  Point2d p2 = Point2d(cos(a2), sin(a2)) * r + c;
  return cross(p1, p2) / 2 + aa;
}
/* O(n^2logn), please remove coincided circles first. */
double circle_union(vector<Circle> &vc){
  int n = vc.size();
  for (int i = n - 1; i >= 0; i--){
    if (eq(vc[i].r, 0)){
      swap(vc[i], vc[n - 1]);
      n--;
      continue;
    }
    for (int j = 0; j < i; j++){
      if (eq(vc[i].x, vc[j].x) and eq(vc[i].y, vc[j].y) and eq(vc[i].r, vc[j].r)){
        swap(vc[i], vc[n - 1]);
        n--;
      }
    }
  }
  if (n == 0)
    return 0;
  vc.resize(n);
  vector<double> cntarea(2 * n, 0);
  for (int c = 0; c < n; c++){
    int cvrcnt = 0;
    e.clear();
    for (int i = 0; i < n; i++){
      if (i != c){
        int r = testCC(vc[c], vc[i]);
        if (r == 2){
          cvrcnt++;
        } else if (r == 0){
          auto paa = isCC(vc[c], vc[i]);
          add(paa.first, paa.second);
        }
      }
    }
    if (e.size() == 0){
      double a = PI * vc[c].r * vc[c].r;
      cntarea[cvrcnt] -= a;
      cntarea[cvrcnt + 1] += a;
    } else {
      e.push_back(event_t(-PI, 1));
      e.push_back(event_t(PI, -2));
      sort(e.begin(), e.end());
      for (int i = 0; i < int(e.size()) - 1; i++){
        cvrcnt += e[i].delta;
        double a = calc(Point2d(vc[c].x, vc[c].y), vc[c].r, e[i].theta, e[i +
            1].theta);
        cntarea[cvrcnt - 1] -= a;
        cntarea[cvrcnt] += a;
      }
    }
  }
  double ans = 0;
  for(int i=1; i<=n; i++)
    ans += cntarea[i];
  return ans;
}
```

5.3 Circles to Tree

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii;
struct Circle{
  int x, y, r, id;
  Circle(){}
  Circle(int x1, int y1, int r1, int id1): x(x1), y(y1), r(r1), id(id1){}
};
// a^2 + b^2 == c^2
double findB(double a, double c){
  return sqrt(c*c - a*a);
}
//- There is no intersection between the circles
//- The parent of circle i will be the smallest circle that includes i
namespace CirclesToTree{
  int X = 0;
  int n;
  vector<Circle> vc;
  vector<int> p;
  struct SetElement{
    int id;
    int side; //Up:1, Down:-1
    SetElement(int id1, int side1): id(id1), side(side1){};
    double getY(int x = X) const{
      return vc[id].y + side*findB(vc[id].x - x, vc[id].r);
    }
    bool operator <(const SetElement &o) const{
      auto l = getY(), r = o.getY();
      if(abs(l-r)<1e-9)
        return vc[id].r*side < vc[o.id].r*o.side;
      else
        return l < r;
    }
  };
  long long pw2(int a){
    return a*1LL*a;
  }
  bool contains(int big, int small){
    if(big == -1 or small == -1) return false;
    Circle &s = vc[small], &b = vc[big];
    if(s.r > b.r) return false;
    return pw2(s.x-b.x) + pw2(s.y-b.y) <= pw2(b.r-s.r);
  }
  void updateParent(int id, int par){
    if(par != -1 and p[id] == -1) p[id] = par;
  }
//Public
  vector<vector<int>> solve(vector<Circle> circles){
    vc = circles; n = vc.size();
    p.assign(n, -1);
    vector<vector<int>> adj(n, vector<int>());
    vector<pii> events;
    for(auto c: vc){
      events.emplace_back(c.x-c.r, ~c.id);
      events.emplace_back(c.x+c.r, c.id);
    }
    sort(events.begin(), events.end());
    set<SetElement> st;
    for(auto e: events){
      X = e.first;
      int id = e.second;
      if(id < 0){
        id = ~id;
        auto it = st.lower_bound(SetElement(id, -2));
        if(it != st.end()){
          int id2 = it->id;
          if(contains(id2, id)) updateParent(id, id2);
          if(contains(p[id2], id)) updateParent(id, p[id2]);
        }
        if(it != st.begin()){
          it--;
          int id2 = it->id;
          if(contains(id2, id)) updateParent(id, id2);
          if(contains(p[id2], id)) updateParent(id, p[id2]);
        }
        st.emplace(id, 1);
        st.emplace(id, -1);
        if(p[id] != -1){
          adj[p[id]].push_back(id);
        }
      }else{
        st.erase(SetElement(id, 1));
        st.erase(SetElement(id, -1));
      }
    }
    return adj;
  }
};
```

5.4 Count Lattices

```
#include "../../code/math/fraction.h"
Fraction f_1 = 1;
//Calculates number of integer points (x,y) such for 0<=x<n and 0<y<=floor(k*x
    +b)
//O(log(N)*log(MAXV))
f_type count_lattices(Fraction k, Fraction b, f_type n) {
  auto fk = (f_type)k;
  auto fb = (f_type)b;
  auto cnt = 0LL;

  if (k >= f_1 || b >= f_1) {
    cnt += (fk * (n - 1) + 2 * fb) * n / 2;
    k = k - Fraction(fk, 1);
    b = b - Fraction(fb, 1);
  }
  auto t = k * Fraction(n, 1) + b;
  auto ft = (f_type)t;
  if (ft >= 1) {
    cnt += count_lattices(f_1 / k, (t - Fraction((f_type)t, 1)) / k, (f_type)t
        );
  }
  return cnt;
}
```

5.5 Convex Hull

```
#include "basic_geometry.h"
using namespace std;
//If accept collinear points then change for <=
bool cw(Point2d a, Point2d b, Point2d c) {
  return lt(cross(b - a, c - b), 0);
}
//If accept collinear points then change for >=
bool ccw(Point2d a, Point2d b, Point2d c) {
  return gt(cross(b - a, c - b), 0);
}
// Returns the points clockwise
vector<Point2d> convex_hull(vector<Point2d> a){
  if (a.size() == 1)
    return a;
  sort(a.begin(), a.end());
  a.erase(unique(a.begin(), a.end()), a.end());
  vector<Point2d> up, down;
  Point2d p1 = a[0], p2 = a.back();
  up.push_back(p1);
  down.push_back(p1);
  for (int i = 1; i < (int)a.size(); i++){
    if ((i == int(a.size() - 1)) || cw(p1, a[i], p2)){
      while (up.size() >= 2 && !cw(up[up.size() - 2], up[up.size() - 1], a[i])
          )
        up.pop_back();
      up.push_back(a[i]);
    }
    if ((i == int(a.size() - 1)) || ccw(p1, a[i], p2)){
      while (down.size() >= 2 && !ccw(down[down.size() - 2], down[down.size()
          - 1], a[i]))
        down.pop_back();
      down.push_back(a[i]);
    }
  }
  a.clear();
  for (int i = 0; i < (int)up.size(); i++)
    a.push_back(up[i]);
  for (int i = down.size() - 2; i > 0; i--)
    a.push_back(down[i]);
  return a;
}
```

5.6 Convex Polygon

```
#include "convex_hull.h"
using namespace std;
//Checks if the point P belongs to the segment AB
bool pointInSegment(Point2d &a, Point2d &b, Point2d &p) {
  if(!eq(cross(a-p, b-p), 0))
    return false;
  return betw(a.x, b.x, p.x) && betw(a.y, b.y, p.y);
}
struct ConvexPolygon{
  vector<Point2d> vp;
  ConvexPolygon(vector<Point2d> aux){
    //The points have to be clockwise
    vp = convex_hull(aux);
  }
  //O(log(N))
  //Accepts points on the edge
  bool pointInPolygon(Point2d point){
    if(vp.size() < 3)
      return pointInSegment(vp[0], vp[1], point);
    if(!eq(cross(vp[1]-vp[0], point-vp[0]), 0) and sgn(cross(vp[1]-vp[0],
        point-vp[0])) != sgn(cross(vp[1]-vp[0], vp.back()-vp[0])) )
      return false;
    if(!eq(cross(vp.back()-vp[0], point-vp[0]), 0) and sgn(cross(vp.back()-vp
        [0], point-vp[0])) != sgn(cross(vp.back() - vp[0], vp[1]-vp[0])) )
      return false;
    if(eq(cross(vp[1]-vp[0], point-vp[0]), 0))
      return ge(norm(vp[1]-vp[0]), norm(point-vp[0]));
    int pos = 1, l = 1, r = vp.size() - 2;
    while(l <= r){
      int mid = (l + r)/2;
      if(le(cross(vp[mid] - vp[0], point - vp[0]), 0)){
        pos = mid;
        l = mid+1;
      }else{
        r = mid-1;
      }
    }
    return pointInTriangle(vp[0], vp[pos], vp[pos+1], point);
  }
};
```

5.7 General Polygon

```
#include "basic_geometry.h"
const int INSIDE=-1, BOUNDARY=0, OUTSIDE=1;
struct GeneralPolygon{
  vector<Point2d> vp;
  GeneralPolygon(vector<Point2d> aux){
    vp = aux;
  }
  // -1 inside, 0 boundary, 1 outside
  int pointInPolygon(Point2d pt) {
    int n = vp.size(), w = 0;
    for(int i=0; i<n; i++){
      if(pt == vp[i])
        return 0;
      int j = (i+1==n?0:i+1);
      if(vp[i].y == pt.y and vp[j].y == pt.y) {
        if (min(vp[i].x, vp[j].x) <= pt.x and pt.x <= max(vp[i].x, vp[j].x))
          return 0;
      }else{
        bool below = vp[i].y < pt.y;
        if (below != (vp[j].y < pt.y)) {
          auto orientation = cross(pt-vp[i], vp[j]-vp[i]);
          if (orientation == 0) return 0;
          if (below == (orientation > 0))
            w += below ? 1 : -1;
        }
      }
    }
    return (w==0?1:-1);
  }
};
```

5.8 Nearest Pair Of Points

```
#include <bits/stdc++.h>
using namespace std;
struct pt {
  long long x, y, id;
  pt(){}
  pt(int _x, int _y, int _id=-1):x(_x), y(_y), id(_id){}
};
namespace NearestPairOfPoints{
  struct cmp_x {
    bool operator()(const pt & a, const pt & b) const {
      return a.x < b.x || (a.x == b.x && a.y < b.y);
    }
  };
  struct cmp_y {
    bool operator()(const pt & a, const pt & b) const {
      return a.y < b.y;
    }
  };
  int n;
  vector<pt> v;
  vector<pt> t;
  double mindist;
  pair<int, int> best_pair;
  void upd_ans(const pt & a, const pt & b) {
    double dist = sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y));
    if (dist < mindist) {
      mindist = dist;
      best_pair = {a.id, b.id};
    }
  }
  void rec(int l, int r) {
    if (r - l <= 3) {
      for (int i = l; i < r; ++i) {
        for (int j = i + 1; j < r; ++j) {
          upd_ans(v[i], v[j]);
        }
      }
      sort(v.begin() + l, v.begin() + r, cmp_y());
      return;
    }
    int m = (l + r) >> 1;
    int midx = v[m].x;
    rec(l, m);
    rec(m, r);
    merge(v.begin() + l, v.begin() + m, v.begin() + m, v.begin() + r, t.begin
         (), cmp_y());
    copy(t.begin(), t.begin() + r - l, v.begin() + l);
    int tsz = 0;
    for (int i = l; i < r; ++i) {
      if (abs(v[i].x - midx) < mindist) {
        for (int j = tsz - 1; j >= 0 && v[i].y - t[j].y < mindist; --j)
          upd_ans(v[i], t[j]);
        t[tsz++] = v[i];
      }
    }
  }
  pair<int, int> solve(vector<pt> _v){
    v = _v;
    n = v.size();
    t.resize(n);
    sort(v.begin(), v.end(), cmp_x());
    mindist = 1E20;
    rec(0, n);
    return best_pair;
  }
};
```

5.9 Point 3D

```
#include <bits/stdc++.h>
using namespace std;
//#define POINT_DOUBLE
#ifdef POINT_DOUBLE
  typedef double ftype;
  typedef long double ftLong;
  const double EPS = 1e-9;
  #define eq(a, b) (abs(a-b)<EPS)
  #define lt(a, b) ((a+EPS)<b)
  #define gt(a, b) (a>(b+EPS))
  #define le(a, b) (a<(b+EPS))
  #define ge(a, b) ((a+EPS)>b)
#else
  typedef int32_t ftype;
  typedef int64_t ftLong;
  #define eq(a, b) (a==b)
  #define lt(a, b) (a<b)
  #define gt(a, b) (a>b)
  #define le(a, b) (a<=b)
  #define ge(a, b) (a>=b)
#endif
//Point3D
struct Point3d{
  ftype x, y, z;
  Point3d() {}
  Point3d(ftype x, ftype y, ftype z) : x(x), y(y), z(z) {}
  Point3d operator+(Point3d t){
    return Point3d(x + t.x, y + t.y, z + t.z);
  }
  Point3d operator-(Point3d t){
    return Point3d(x - t.x, y - t.y, z - t.z);
  }
  Point3d operator*(ftype t){
    return Point3d(x * t, y * t, z * t);
  }
  Point3d operator/(ftype t){
    return Point3d(x / t, y / t, z / t);
  }
};
ftLong dot(Point3d a, Point3d b){
  return a.x * (ftLong)b.x + a.y * (ftLong)b.y + a.z * (ftLong)b.z;
}
double len(Point3d a){
  return sqrt(dot(a, a));
}
double dist(Point3d a, Point3d b){
  return len(a-b);
}
double proj(Point3d a, Point3d b){
  return dot(a, b) / len(b);
}
//theta -> XY; phi -> ZY;
Point3d toVetor(double theta, double phi, double r){
  return Point3d(r*cos(theta)*sin(phi), r*sin(theta)*sin(phi), r*cos(phi));
}
double getAngleTheta(Point3d p){
  return atan2(p.y, p.x);
}
double getAnglePhi(Point3d p){
  return acos(p.z/len(p));
}
Point3d rotateX(Point3d p, double ang){
  return Point3d(p.x, p.y*cos(ang)-p.z*sin(ang), p.y*sin(ang)+p.z*cos(ang));
}
Point3d rotateY(Point3d p, double ang){
  return Point3d(p.x*cos(ang)+p.z*sin(ang), p.y, -p.x*sin(ang)+p.z*cos(ang));
}
Point3d rotateZ(Point3d p, double ang){
  return Point3d(p.x*cos(ang)-p.y*sin(ang), p.x*sin(ang)+p.y*cos(ang), p.z);
}
//Rotation in relation to the normal axis
Point3d rotateNormal(Point3d v, Point3d n, double ang){
  double theta = getAngleTheta(n);
  double phi = getAnglePhi(n);
  v = rotateZ(v, -theta);
  v = rotateY(v, -phi);
  v = rotateZ(v, ang);
  v = rotateY(v, phi);
  v = rotateZ(v, theta);
  return v;
}
Point3d cross(Point3d a, Point3d b){
  return Point3d(a.y * b.z - a.z * b.y,
                 a.z * b.x - a.x * b.z,
                 a.x * b.y - a.y * b.x);
}
ftLong triple(Point3d a, Point3d b, Point3d c){
  return dot(a, cross(b, c));
}
Point3d planeIntersect(Point3d a1, Point3d n1, Point3d a2, Point3d n2, Point3d
      a3, Point3d n3){
  Point3d x(n1.x, n2.x, n3.x);
  Point3d y(n1.y, n2.y, n3.y);
  Point3d z(n1.z, n2.z, n3.z);
  Point3d d(dot(a1, n1), dot(a2, n2), dot(a3, n3));
  return Point3d(triple(d, y, z),
                 triple(x, d, z),
                 triple(x, y, d)) / triple(n1, n2, n3);
}
struct Sphere{
  ftype x, y, z, r;
  Sphere(){}
  Sphere(ftype x, ftype y, ftype z,  ftype r):x(x), y(y), z(z), r(r){}
};
//Minimum enclosing Sphere, O(n*70000)
//It is also possible to do with ternary search in the 3 dimensions
Sphere minimumSphere(vector<Point3d> vp){
  Point3d ans(0, 0, 0);
  int n = vp.size();
  for(Point3d p: vp)
    ans = ans + p;
  ans = ans/n;
  double P = 0.1;
  double d = 0, e = 0;
  for(int i = 0; i < 70000; i++){
    int f = 0;
    d = dist(ans, vp[0]);
    for (int j = 1; j < n; j++) {
      e = dist(ans, vp[j]);
      if (d < e) {
        d = e;
        f = j;
      }
    }
    ans = ans + (vp[f]-ans)*P;
    P *= 0.998;
  }
  return Sphere(ans.x, ans.y, ans.z, d);
}
```

5.10 Triangle

```
#include <bits/stdc++.h>
using namespace std;
typedef long double ld;
const ld PI = acosl(-1);
```

```
struct Triangle{
  ld a, b, c;
  Triangle(){}
  Triangle(ld a1, ld b1, ld c1):a(a1), b(b1), c(c1){
    fix();
  }
  ld area(){
    ld s = (a + b + c)/2;
    return sqrtl(s*(s-a)*(s-b)*(s-c));
  }
  void fix(){
    if(a > b) swap(a, b);
    if(a > c) swap(a, c);
    if(b > c) swap(b, c);
  }
  tuple<ld, ld, ld> angle(){
    fix();
    ld h = (2*area())/c;
    ld aa = asin(h/b);
    ld bb = asin(h/a);
    return {aa, bb, PI - aa - bb};
  }
};
```

6 String Algorithms

6.1 Min Cyclic String

```
#include <bits/stdc++.h>
using namespace std;
string min_cyclic_string(string s){
  s += s;
  int n = s.size();
  int i = 0, ans = 0;
  while (i < n / 2){
    ans = i;
    int j = i + 1, k = i;
    while (j < n && s[k] <= s[j]){
      if (s[k] < s[j])
        k = i;
      else
        k++;
      j++;
    }
    while (i <= k)
      i += j - k;
  }
  return s.substr(ans, n / 2);
}
```

6.2 Suffix Automaton

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
struct SuffixAutomaton{
  struct state{
    int len, link, first_pos;
    bool is_clone = false;
    map<char, int> next;
  };
  vector<state> st;
  int sz, last;
  SuffixAutomaton(string s){
    st.resize(2 * s.size() + 10);
    st[0].len = 0;
    st[0].link = -1;
    st[0].is_clone = false;
    sz = 1;
    last = 0;
    for (char c : s)
      insert(c);
    preCompute();
  }
  void insert(char c){
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    st[cur].first_pos = st[cur].len - 1;
    st[cur].is_clone = false;
    int p = last;
    while (p != -1 && !st[p].next.count(c)){
      st[p].next[c] = cur;
      p = st[p].link;
    }
    if (p == -1){
      st[cur].link = 0;
    }else{
      int q = st[p].next[c];
      if (st[p].len + 1 == st[q].len){
        st[cur].link = q;
      }else{
        int clone = sz++;
        st[clone].len = st[p].len + 1;
        st[clone].next = st[q].next;
        st[clone].link = st[q].link;
        st[clone].first_pos = st[q].first_pos;
        st[clone].is_clone = true;
        while (p != -1 && st[p].next[c] == q){
          st[p].next[c] = clone;
          p = st[p].link;
        }
        st[q].link = st[cur].link = clone;
      }
    }
    last = cur;
  }
  // Dado o estado v e o tamanho l do match atual, retorna o proximo estado
  // e o tamanho do match apos ler o caractere c
  void nxt(int &v, int &l, char c){
    while (v and !st[v].next.count(c)){
      v = st[v].link;
      l = st[v].len;
    }
    if (st[v].next.count(c)){
      v = st[v].next[c];
      l++;
    }
  }
  string lcs(string s){
    int v = 0, l = 0, best = 0, bestpos = 0;
    for (int i = 0; i < (int)s.size(); i++){
      while (v and !st[v].next.count(s[i])){
        v = st[v].link;
        l = st[v].len;
      }
      if (st[v].next.count(s[i])){
        v = st[v].next[s[i]];
        l++;
      }
      if (l > best){
        best = l;
        bestpos = i;
      }
    }
    return s.substr(bestpos - best + 1, best);
  }
  vector<ll> dp;
  vector<int> cnt;
  ll dfsPre(int s){
    if (dp[s] != -1)
      return dp[s];
    dp[s] = cnt[s]; //Accepts repeated substrings
    //dp[s] = 1; //Does not accept repeated substrings
    for (auto p : st[s].next)
      dp[s] += dfsPre(p.second);
    return dp[s];
  }
  void preCompute(){
    cnt.assign(sz, 0);
    vector<pair<int, int>> v(sz);
    for (int i = 0; i < sz; i++){
      cnt[i] = !st[i].is_clone;
      v[i] = make_pair(st[i].len, i);
    }
    sort(v.begin(), v.end(), greater<pair<int, int>>());
    for (int i = 0; i < sz - 1; i++)
      cnt[st[v[i].second].link] += cnt[v[i].second];
    dp.assign(sz, -1);
    dfsPre(0);
  }
};
```

6.3 Suffix Tree

```
#include <bits/stdc++.h>
typedef long long ll;
using namespace std;
namespace SuffixTree {
const int NS = 60; //Number of strings
const int MAXN = 100010; //Number of letters
int cn, cd, ns, en = 1, lst;
string S[NS]; int lastS = -1;
/* sufn[si][i] no do sufixo S[si][i...] */
vector<int> sufn[NS];
struct Node {
  int l, r, si=0;
  int p, suf=0;
  map<char, int> adj;
  Node() : l(0), r(-1){ suf = p = 0; }
  Node(int l1, int r1, int s1, int p1) : l(l1), r(r1), si(s1), p(p1) {}
  inline int len() { return r - l + 1; }
  inline int operator[](int i) { return S[si][l + i]; }
  inline int& operator()(char c) { return adj[c]; }
};
Node t[2*MAXN];
inline int new_node(int l, int r, int s, int p) {
  t[en] = Node(l, r, s, p);
  return en++;
}
void init(){
  t[0] = Node();
  cn=0, cd=0, ns=0, en = 1, lst=0;
  lastS = -1;
}
//The strings are inserted independently
void add_string(string s, char id=’$’) {
  assert(id < ’A’);
  s += id;
  S[++lastS] = s;
  sufn[lastS].resize(s.size() + 1);
  cn = cd = 0;
  int i = 0; const int n = s.size();
  for(int j = 0; j < n; j++){
    for(; i <= j; i++) {
      if(cd == t[cn].len() && t[cn](s[j]))
        cn = t[cn](s[j]), cd = 0;
      if(cd < t[cn].len() && t[cn][cd] == s[j]) {
        cd++;
        if(j < (int)s.size() - 1) break;
        else {
          if(i) t[lst].suf = cn;
          for(; i <= j; i++) {
            sufn[lastS][i] = cn;
            cn = t[cn].suf;
          }
        }
      } else if(cd == t[cn].len()) {
        sufn[lastS][i] = en;
        if(i) t[lst].suf = en;
        lst = en;
        t[cn](s[j]) = new_node(j, n - 1, lastS, cn);
        cn = t[cn].suf;
        cd = t[cn].len();
      } else {
        int mid = new_node(t[cn].l, t[cn].l + cd - 1, t[cn].si, t[cn].p);
        t[t[cn].p](t[cn][0]) = mid;
        if(ns) t[ns].suf = mid;
        if(i) t[lst].suf = en;
        lst = en;
        sufn[lastS][i] = en;
        t[mid](s[j]) = new_node(j, n - 1, lastS, mid);
        t[mid](t[cn][cd]) = cn;
        t[cn].p = mid; t[cn].l += cd;
        cn = t[mid].p;
        int g = cn? j - cd : i + 1;
        cn = t[cn].suf;
        while(g < j && g + t[t[cn](S[lastS][g])].len() <= j)
          cn = t[cn](S[lastS][g]), g += t[cn].len();
        if(g == j)
```

```
          ns = 0, t[mid].suf = cn, cd = t[cn].len();
        else
          ns = mid, cn = t[cn](S[lastS][g]), cd = j - g;
      }
    }
  }
}
bool match(string &s, int i=0, int no=0, int iEdge=0){
  if(i == (int)s.size())
    return true;
  if(iEdge == t[no].len()){ //I arrived at the Node
    if(t[no].adj.count(s[i]))
      return match(s, i+1, t[no].adj[s[i]], 1);
    else
      return false;
  }
  if(t[no][iEdge] == s[i])
    return match(s, i+1, no, iEdge+1);
  return false;
}
typedef tuple<int, int, int> tp;
// O(n), substring <i, l, r> = s[i..l], s[i..l+1], ..., s[i..r]
void getDistinctSubstrings(vector<tp> &v, int no=0, int d=0){
  d += t[no].len() - t[no].adj.empty();
  int l = t[no].l, r = t[no].r - t[no].adj.empty();
  if(l <= r){
    v.emplace_back(r - d + 1, l, r);
  }
  for(auto [x, to]: t[no].adj)
    getDistinctSubstrings(v, to, d);
}
};
```

7 Miscellaneous

7.1 Automaton

```
#include <bits/stdc++.h>
using namespace std;
const int K = 26;
struct Automaton{
  int n;
  vector<array<int, K>> to;
  vector<bool> accept;
  Automaton(int sz, bool acceptAll=true){
    to.assign(sz, {0});
    accept.assign(sz, acceptAll);
    n = sz;
  }
};
const int INTERSECT=0, UNION=1;
Automaton join(Automaton a, Automaton b, int op=INTERSECT){
  Automaton ret(a.n ∗ b.n);
  for(int i=0; i<a.n; i++){
    for(int j=0; j<b.n; j++){
      int st = i ∗ b.n + j;
      if(op == INTERSECT)
        ret.accept[st] = a.accept[i] and b.accept[j];
      else
        ret.accept[st] = a.accept[i] or b.accept[j];
      for(int k=0; k<K; k++)
        ret.to[st][k] = a.to[i][k] ∗ b.n + b.to[j][k];
    }
  }
  return ret;
}
```

7.2 Fast IO

```
#include <bits/stdc++.h>
int readInt () {
  bool minus = false;
  int result = 0;
  char ch;
  ch = getchar();
  while (true) {
    if (ch == ’-’) break;
    if (ch >= ’0’ && ch <= ’9’) break;
    ch = getchar();
  }
  if (ch == ’-’) minus = true; else result = ch-’0’;
  while (true) {
    ch = getchar();
    if (ch < ’0’ || ch > ’9’) break;
    result = result∗10 + (ch - ’0’);
  }
  if (minus)
    return -result;
  else
    return result;
}
```

7.3 Mo Algorithm

```
#include <bits/stdc++.h>
using namespace std;
const int BLOCK_SIZE = 700;
void remove(int idx);
void add(int idx);
void clearAnswer();
int getAnswer();
struct Query{
  int l, r, idx;
  bool operator<(Query other) const{
    if (l / BLOCK_SIZE != other.l / BLOCK_SIZE)
      return l < other.l;
    return (l / BLOCK_SIZE & 1) ? (r < other.r) : (r > other.r);
  }
};
vector<int> mo_s_algorithm(vector<Query> queries){
  vector<int> answers(queries.size());
  sort(queries.begin(), queries.end());
  clearAnswer();
  int L = 0, R = 0;
  add(0);
  for(Query q : queries){
    while(q.l < L) add(--L);
    while(R < q.r) add(++R);
    while(L < q.l) remove(L++);
    while(q.r < R) remove(R--);
    answers[q.idx] = getAnswer();
  }
  return answers;
}
```

7.4 Parallel Binary Search

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 100010;
int ans[MAXN];
bool test(int x);
void add(int k);
void remove(int k);
void solve(int i, int j, vector<int> &v){
  if(v.empty())
    return;
  if(i == j){
    for(int x: v)
      ans[x] = i;
    return;
  }
  int mid = (i+j)/2;
  for(int k=i; k<=mid; k++)
    add(k);
  vector<int> left, right;
  for(int x: v){
    if(test(x))
      left.push_back(x);
    else
      right.push_back(x);
  }
  solve(mid+1, j, right);
  for(int k=mid; k>=i; k--)
    remove(k); // Or roolback();
  solve(i, mid, left);
}
```

7.5 Pragma

```
#pragma GCC optimize("O3", "unroll-loops")
#pragma GCC target("avx2")
#pragma GCC target("popcnt")
```

7.6 Random Function

```
#include <bits/stdc++.h>
using namespace std;
mt19937 rng((int) std::chrono::steady_clock::now().time_since_epoch().count())
    ;
inline int rand(int l, int r){
  return uniform_int_distribution<int>(l, r)(rng);
}
inline double rand(double l, double r){
  return uniform_real_distribution<double>(l, r)(rng);
}
mt19937_64 rng_64((int) std::chrono::steady_clock::now().time_since_epoch().
    count());
inline int64_t rand(int64_t l, int64_t r){
  return uniform_int_distribution<int64_t>(l, r)(rng_64);
}
void randomShuffle(vector<int> &v){
  shuffle(v.begin(), v.end(), rng);
}
```

7.7 Sprague Grundy

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1010;
int version;
int used[MAXN];
int mex() {
  for(int i=0; ; ++i)
    if(used[i] != version)
      return i;
}
int g[MAXN];
// Can remove 1, 2 and 3
void grundy(){
  //Base case depends on the problem
  g[0] = 0;
  g[1] = 1;
  g[2] = 2;
  //Inductive case
  for(int i=3; i<MAXN; i++){
    version++;
    used[g[i-1]] = version;
    used[g[i-2]] = version;
    used[g[i-3]] = version;
    g[i] = mex();
  }
}
string solve(vector<int> v){
  grundy();
  int ans = 0;
  for(int x: v)
    ans ^= g[x];
  return ((ans != 0) ? "First" : "Second");
}
```

8 Theorems and Formulas

8.1 Binomial Coefficients

$(a+b)^n = \binom{n}{0}a^n + \binom{n}{1}a^{n-1}b + \binom{n}{2}a^{n-2}b^2 + \cdots + \binom{n}{k}a^{n-k}b^k + \cdots + \binom{n}{n}b^n$

Pascal's Triangle: $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$

Symmetry rule: $\binom{n}{k} = \binom{n}{n-k}$

Factoring in: $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$

Sum over k: $\sum_{k=0}^{n} \binom{n}{k} = 2^n$

Sum over n: $\sum_{m=0}^{n} \binom{m}{k} = \binom{n+1}{k+1}$

Sum over n and k: $\sum_{k=0}^{m} \binom{n+k}{k} = \binom{n+m+1}{m}$

Sum of the squares: $\binom{n}{0}^2 + \binom{n}{1}^2 + \cdots + \binom{n}{n}^2 = \binom{2n}{n}$

Weighted sum: $1\binom{n}{1} + 2\binom{n}{2} + \cdots + n\binom{n}{n} = n2^{n-1}$

Connection with the Fibonacci numbers: $\binom{n}{0} + \binom{n-1}{1} + \cdots + \binom{n-k}{k} + \cdots + \binom{0}{n} = F_{n+1}$

More formulas: $\sum_{k=0}^{m}(-1)^k \cdot \binom{n}{k} = (-1)^m \cdot \binom{n-1}{m}$

8.2 Catalan Number

Recursive formula: $C_0 = C_1 = 1$

$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}, n \geq 2$

Analytical formula: $C_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{1}{n+1}\binom{2n}{n}, n \geq 0$

The first few numbers Catalan numbers, C_n (starting from zero): $1, 1, 2, 5, 14, 42, 132, 429, 1430, \ldots$

The Catalan number C_n is the solution for:

- Number of correct bracket sequence consisting of n opening and n closing brackets.
- The number of rooted full binary trees with $n + 1$ leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.
- The number of ways to completely parenthesize $n + 1$ factors.
- The number of triangulations of a convex polygon with $n + 2$ sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
- The number of ways to connect the $2n$ points on a circle to form n disjoint chords.
- The number of non-isomorphic full binary trees with n internal nodes (i.e. nodes having at least one son).
- The number of monotonic lattice paths from point $(0, 0)$ to point (n, n) in a square lattice of size $n \times n$, which do not pass above the main diagonal (i.e. connecting $(0, 0)$ to (n, n)).
- Number of permutations of length n that can be stack sorted (i.e. it can be shown that the rearrangement is stack sorted if and only if there is no such index $i < j < k$, such that $a_k < a_i < a_j$).
- The number of non-crossing partitions of a set of n elements.
- The number of ways to cover the ladder $1 \ldots n$ using n rectangles (The ladder consists of n columns, where i^{th} column has a height i).

8.3 Euler's Totient

If p is a prime number: $\phi(p) = p - 1$ and $\phi(p^k) = p^k - p^{k-1}$

If a and b are relatively prime, then: $\phi(ab) = \phi(a) \cdot \phi(b)$

In general: $\phi(ab) = \phi(a) \cdot \phi(b) \cdot \dfrac{gcd(a, b)}{\phi(gcd(a, b))}$

This interesting property was established by Gauss: $\sum_{d|n} \phi(d) = n$, Here the sum is over all positive divisors d of n.

Euler's theorem: $a^{\phi(m)} \equiv 1 \pmod{m}$, if a and m are relatively prime.

Generalization: $a^n \equiv a^{\phi(m)+[n \bmod \phi(m)]} \mod m$, for arbitrary a, m and n $\geq log_2(m)$.

8.4 Formulas

Count the number of ways to partition a set of n labelled objects into k nonempty labelled subsets.

$$f(n, k) = \sum_{i=0}^{k}(-1)^i \binom{k}{i}(k - i)^n$$

Stirling Number 2nd: Partitions of an n element set into k not-empty set. Or count the number of ways to partition a set of n labelled objects into k nonempty unlabelled subsets.

$$S_{2nd}(n, k) = \left\{ {n \atop k} \right\} = \frac{1}{k!}\sum_{i=0}^{k}(-1)^i \binom{k}{i}(k - i)^n$$

Euler's formula: $f = e - v + 2$

Euler's formula to n Lines or Segment if there is no three lines/segments that contains the same point: $R = intersects + component - n$

Number of regions in a planar graph: $R = E - V + C + 1$ where C is the number of connected components

Given a and b co-prime, $n = a \cdot x + b \cdot y$ where $x \geq 0$ and $y \geq 0$. You are required to find the least value of n, such that all currency values greater than or equal to n can be made using any number of coins of denomination a and b: $n = (a - 1) * (b - 1)$

generalization of the above problem, n is multiple of $gcd(a, b)$: $n = lcm(a, b) - a - b + gcd(a, b)$

8.5 Primes

If $n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$, then:

Number of divisors is $d(n) = (e_1 + 1) \cdot (e_2 + 1) \cdots (e_k + 1)$.

Sum of divisors is $\sigma(n) = \frac{p_1^{e_1+1}-1}{p_1-1} \cdot \frac{p_2^{e_2+1}-1}{p_2-1} \cdots \frac{p_k^{e_k+1}-1}{p_k-1}$