1 VIM

1.1 1_vimrc

```
set nu cin ts=4 sw=4 aw hls is
syntax on

colo torte
set nocompatible

inoremap {<CR> {<CR>}<ESC>k$a<CR>
nn <F8> :w <bar> :!vim
nn <F9> :w <bar> :!g++ % -std=c++17 -O2 -Wall -Wextra -g -fsanitize=address -o %<<CR>
nn <F3> :w <bar> :!./%<<CR>
nn <F4> :w <bar> :!./%< <

// command
sp, vsp
<C-w> {n} {< + - >?}

// replace
:%s/target/replacement/gc    // % for global, g for all, c for confirm.
```

1.2 2_code_template

```
#pragma GCC optimize("Ofast")
#include <bits/stdc++.h>
using namespace std;

using ll = long long;
template<typename T> using vec = vector<T>;
template<typename T> using deq = deque<T>;
template<typename T> using p = pair<T, T>;

#define yccc ios_base::sync_with_stdio(false), cin.tie(0)
#define endl '\n'
#define al(a) a.begin(), a.end()
#define eb emplace_back
#define F first
#define S second

int main() {
  yccc;
}
```

1.3 3_tips

- Segment Tree, DP, bitwise DP, 枚舉, 枚舉 + 剪枝, Disjoint Set
- Priority Queue, 單調隊列, Prefix Sum, 偏序
- Graph: SCC, AP, Bridge, LCA, 2-SAT
- Flow, Min-cost Max-flow, Bipartite
- Primal test, PollardRho, KMP, Rabin Fingerprint, FFT
- Convex Hull, 旋轉卡尺, 極角排序

1.4 4_rsync

```
#!/bin/bash

while true; do
  rsync -zavh ~/Desktop/*.cpp /media/redleaf/backup
  sleep 10
done
```

2 data_structure

2.1 disjointset

```
#include <algorithm>
using namespace std;
#define MAX_N 200005
struct  disjointset
{
    int rank[MAX_N];
    int f[MAX_N];
    void init(int N){
        for (int i = 0; i < N;i++){
            f[i] = i;
            rank[i] = 1;
        }

    }
    int find(int v){
        if( f[v] == v)
            return v;
        return f[v] = find(f[v]);
    }
    bool same(int a, int b){
        return find(a) == find(b);
    }
    void Union(int a, int b){
        // f[find(a)] = find(b);
        if(!same(a,b)){
            if(rank[a] < rank[b])
                swap(a, b);
            f[f[b]] = f[a];
            rank[a]++;
        }
    }
};
```

2.2 Fenwick_Tree

```
// l,r means [l, r]
const int maxn = 100000;

struct BIT {
    int data[maxn+1];
    void update(int idx, int val) {
        while (idx <= maxn) {
            data[idx] += val;
            idx += idx & (~idx + 1);
        }
    }
    void update(int l, int r, int val) {
        update(l, val);
        update(r + 1, -val);
    }
    int query(int idx) {
        int res = 0;
        while (idx > 0) {
            res += data[idx];
            idx -= idx & (~idx + 1);
        }
        return res;
    }
    int query(int l, int r) {
        return query(r) - query(l);
    }
};

// Range Modify, Range query prefix sum (all O(logn)).
struct LazyBIT {
    BIT bitAdd, bitSub;
    void update(int l, int r, int val) {
        bitAdd.update(l, r, val);
        bitSub.update(l, r, (l - 1) * val);
        bitSub.update(r + 1, (-r + l - 1) * val);
    }
    int query(int idx) {
        return idx * bitAdd.query(idx) - bitSub.query(idx);
    }
    int query(int l, int r) {
        return query(r) - query(l - 1);
    }
};

// usage: problems that range modify can be turn into polynomial of idx.
/* like range update [l, r]: add 1 to l, 2 to l+1, ... (r-l+1) to r. this problem can be turn into
for idx < l, nothing
for l <= idx <= r, add ((idx - l + 1) + 1)*(idx-l+1) / 2, just (a+b)*h/2.
for idx > r, add (r-l+1 + 1) * (r-l+1) / 2.
Decompose them into separate terms like (idx^2, idx, 2*dix, 2*C, origin val). */
// same thoughts may be use on Segment Tree.
struct Polynomial_Queries{
  vec<BIT<ll>> BITs;
  int n;
  // 0 "idx", 1 constant, 2 doubled "idx^2"
  // 3 doubled "idx", 4 doubled constant, 5 origin array
  Polynomial_Queries(){
    BITs.resize(6);
  }
  void Build(vec<ll> & data){
    n = data.size();
    REP(i, 5)
      BITs[i].Build(n); // implement by yourself.
    BITs[5].Build(data);
  }
  void update(int l, int r, ll val){
    BITs[0].r(l, r, val);
    BITs[1].update(l, r, (1- l) * val);
    BITs[2].update(l, r, 1);
    BITs[3].update(l, r, 1 - 2 * l);
    BITs[4].update(l, r, l * (l * 1LL)- l);
```

```
    ll len = r - l + 1;
    ll r_l = r - l;
    BITs[1].update(r + 1, n, len * val);
    BITs[4].update(r + 1, n, len * r_l);
  }
  ll query(int idx){
    ll ans = 0;
    ans += BITs[0].query(idx) * idx;
    ans += BITs[1].query(idx);
    ll doubled = 0;
    doubled += BITs[2].query(idx) * idx * idx;
    doubled += BITs[3].query(idx) * idx;
    doubled+= BITs[4].query(idx);
    ans += (doubled >> 1);
    ans += BITs[5].query(idx);
    return ans;
  }
  ll query(int l, int r){
    return query(r) - query(l - 1);
  }
};
```

2.3 Li_Chao_Tree

```
// Miminimum Li Chao Tree
typedef long long ftype;
typedef complex<ftype> point;
#define x real
#define y imag

ftype dot(point a, point b) {
    return (conj(a) * b).x();
}

ftype f(point a,  ftype x) {
    return dot(a, {x, 1});
}

const int maxn = 2e5;

point line[4 * maxn];

/*
a line is  y = k * x + b, using point to represent it.
y = (k, b) * (x, 1) (dot operation).
*/
// y = nw.real() * x + nw.imag().
void add_line(point nw, int idx = 1, int l = 0, int r = maxn)
{
    int m = (l + r) / 2;
    bool lef = f(nw, l) < f(line[idx], l);
    bool mid = f(nw, m) < f(line[idx], m);
    if(mid) {
        swap(line[idx], nw);
    }
    if(r - l == 1) {
        return;
    } else if(lef != mid) {
        add_line(nw, 2 * idx, l, m);
    } else {
        add_line(nw, 2 * idx + 1, m, r);
    }
}

// get minimum in some point x;
ftype get(int x, int idx = 1, int l = 0, int r = maxn)
{
    int m = (l + r) / 2;
    if(r - l == 1) {
        return f(line[idx], x);
    } else if(x < m) {
        return min(f(line[idx], x), get(x, 2 * idx, l, m));
    } else {
        return min(f(line[idx], x), get(x, 2 * idx + 1, m, r)
             );
    }
}
```

2.4 pbds

```
#include <bits/stdc++.h>
using namespace std;
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // tree
#include <ext/pb_ds/hash_policy.hpp> // hash
#include <ext/pb_ds/trie_policy.hpp> // trie
#include <ext/pb_ds/priority_queue.hpp> // priority_queue

#include <ext/pb_ds/detail/standard_policies.hpp> // general
using namespace __gnu_pbds;

/*
tree-based container has the following declaration:

template<
typename Key, // Key type
typename Mapped, // Mapped-policy
typename Cmp_Fn = std::less<Key>, // Key comparison functor
typename Tag = rb_tree_tag, // Specifies which underlying
     data structure to use
template<
typename Const_Node_Iterator,
typename Node_Iterator,
typename Cmp_Fn_,
typename Allocator_>
class Node_Update = null_node_update, // A policy for
     updating node invariants
typename Allocator = std::allocator<char> > // An allocator
     type
class tree;

*/

using ordered_set = tree<
int,   // Key type
null_type, // Mapped-policy
less<int>, // Key Compar
rb_tree_tag,
tree_order_statistics_node_update>
;
using order_map= tree<int, int, less<int>, rb_tree_tag,
     tree_order_statistics_node_update>;


void test(){
    int x;
    ordered_set X;
    X.find(x); // find node with value x.
    X.insert(x); // insert node with value x.
    X.erase(it); // erase the node iterator point to.
    X.lower_bound(x); // return the first iterator with value
         >= x.
    X.upper_bound(x); // return the first iterator with value
         > x.
    ordered_set X2;
    X.join(X2); // combine two tree, X2 become empty.
    int r;
    X.split(r, X2);
//  X.split(const Key &r, ordered_set &other);
    // put elements > r into other, if we're using  `greater<
         Key>` then it's putting < r into other.
    ordered_set::point_iterator ptr = X.begin();
    ptr = X.end(); // iterator

    X.clear();
    X.insert(1);
    X.insert(2);
    X.insert(4);
    X.insert(8);
    X.insert(16);

    cout<<*X.find_by_order(1)<<endl; // 2
    cout<<*X.find_by_order(2)<<endl; // 4
    cout<<*X.find_by_order(4)<<endl; // 16
    cout<<(end(X)==X.find_by_order(6))<<endl; // true

    cout<<X.order_of_key(-5)<<endl;  // 0
    cout<<X.order_of_key(1)<<endl;   // 0
    cout<<X.order_of_key(3)<<endl;   // 2
    cout<<X.order_of_key(4)<<endl;   // 2
    cout<<X.order_of_key(400)<<endl; // 5
}
```

2.5 segment_Tree

```
#define LL long long
#define IL(X) ((X << 1) + 1)
#define IR(X) ((X << 1) + 2)
#define MAXN 500005
// add tag
// tag += tag
// val += tag*size

struct segID{
    struct Node{
        LL val;
        LL lazy_tag;
        int size;
    };
    LL dataseq[MAXN];
    Node seq[MAXN * 4 + 5];
    void pull(int index){
        seq[index].val = seq[IL(index)].val + seq[IR(index)].
             val;
    }
    void push(int index){
        seq[IL(index)].lazy_tag += seq[index].lazy_tag;
        seq[IL(index)].val += seq[index].lazy_tag * seq[IL(
             index)].size;
```

```
        seq[IR(index)].lazy_tag += seq[index].lazy_tag;
        seq[IR(index)].val += seq[index].lazy_tag * seq[IR(
            index)].size;
        seq[index].lazy_tag = 0;
    }

    void build(int L, int R, int index){
        if(L == R){
            seq[index].val = dataseq[L];
            seq[index].size = 1;
            seq[index].lazy_tag = 0;
            return;
        }
        int M = (L + R) / 2;
        build(L, M, IL(index));
        build(M + 1, R, IR(index));
        seq[index].size = seq[IL(index)].size + seq[IL(index)
            ].size;
        pull(index);
    }

    void modify(int l, int r, int L, int R, int index, long
         long Add){
        if(l == L && r == R){
            seq[index].lazy_tag += Add;
            seq[index].val += Add * seq[index].size;
            return;
        }
        push(index);
        int M = (L + R) / 2;

        if(r <= M){
            modify(l, r, L, M, IL(index), Add);
        }else if(l > M){
            modify(l, r, M + 1, R, IR(index), Add);
        }else{
            modify(l, M, L, M, IL(index), Add);
            modify(M + 1, r, M + 1, R, IR(index), Add);
        }
        pull(index);
    }

    long long Query(int l, int r, int L, int R, int index){
        if(l == L && r == R){
            return seq[index].val;
        }
        int M = (L + R) / 2;
        push(index);
        if(r <= M){
            return Query(l, r, L, M, IL(index));
        }else if(l > M){
            return Query(l, r, M + 1, R, IR(index));
        }else{
            return Query(l, M, L, M, IL(index)) +
            Query(M + 1, r, M + 1, R, IR(index));
        }

    }
};
```

2.6 Sparse_Table

```
#include <bits/stdc++.h>
using namespace std;
int n;
int v[1000009];
int sparse[22][1000009];
// O(nlogn) preprocess O(1)Query
// sp[x][y] is the answer from (v[x], v[x+2^y-1])
inline void init()
{
    for (int i = 0; i < n; ++i)
        sparse[0][i] = v[i];
    for (int j = 1; (1 << j) <= n; ++j)
        for (int i = 0; i + (1 << j) <= n; ++i)
        sparse[j][i] = min(
        sparse[j - 1][i],
        sparse[j - 1][i + (1 << (j - 1))]
        );
}

// get min of v[l, r].
inline int query(int l, int r)
{
    int k = __lg(r - l + 1);
    return min(sparse[k][l], sparse[k][r - (1 << k) + 1]);
}
```

2.7 Treap

```
struct node {
    int key, val; // (key, val)
    int ans; // minans
    int pri, sz; //  priority, size
    node *l, *r;
    int rev, add; // lazy tag

    node () { }
    node (int key) : key(key), val(0), ans(0), pri(rand()),
         sz(1), l(nullptr), r(nullptr), rev(0), add(0){}
    node (int key, int val) : key(key), val(val), ans(val),
         pri(rand()), l(nullptr), r(nullptr), sz(1), rev(0),
         add(0){}
    void push(){
        if(rev){
            swap(l, r);
            if(l) l->rev ^= 1;
            if(r) r->rev ^= 1;
            rev ^= 1;
        }
        if(l){
            l->add += add;
            l->val += val;
            l->ans += add;
        }
        if(l){
            r->add += add;
            r->val += val;
            r->ans += add;
        }
        add = 0;
    }
    void pull(){
        ans = val;
    sz = 1;
    if(l){
      ans = min(ans, l->ans);
      sz += l->sz;
    }
    if(r){
      ans = min(ans, r->ans);
      sz += r->sz;
    }
    }
};
node * root;

int size(node * p){
    return p ? p->sz : 0;
}

void push(node * p){
    if(p){
        p->push();
    }
}

void pull(node * p){
    p->push();
}

node * merge (node * a, node * b) {
    if (!a || !b)  return a ? a : b;
    if (a->pri < b->pri){
        push(a);
        a->r = merge(a->r, b);
        pull(a);
        return a;
    }
    else{
        push(b);
        b->l = merge(a, b->l);
        pull(b);
        return b;
    }
}

// all keys in tree l < key;
void split_by_key(node * rt, node * &a, node * &b, int key)
{
    push(rt);
    if (!rt)
        a = b = nullptr;
    else if (rt->key < key){
        a = rt;
        split_by_key(rt->r, rt->r, b, key);
    }
    else{
        b = rt;
        split_by_key (rt->l, a, rt->l, key);
    }
    pull(rt);
}

// split tree into size(l) = k, size(r) = size(rt) - k.
// all keys in l <= all keys in r.
void split_by_size(node * rt, node * &a, node * &b, int k){
    push(rt);
    if (!rt)
        a = b = nullptr;
    else if(k >= size(rt->l) + 1){
        a = rt;
```

```
        int nk = k - (size(rt->l) + 1);
        split_by_size(rt->r, a->r, b, nk);
    }
    else{
        b = rt;
        split_by_size(rt->l, a, b->l, k);
    }
    pull(rt);
}

// <-- Writing slower, Running Faster -->
// not necessary
void insert1(node * &rt, node * it)
{
    if (!rt)
        rt = it;
    else if (it->pri > rt->pri){
        split_by_key(rt, it->l, it->r, it->key);
        rt = it;
        pull(rt);
    }
    else{
        push(rt);
        insert1(rt->key < it->key ? rt->r : rt->l, it);
        pull(rt);
    }
}

// call this <--- insert item(key, val) --->
void Insert(node * &rt, int key, int val = 0){
    node *newp = new node(key, val);
    node *a, *b;
    split_by_key(rt, a, b, key);
    rt = merge(a, merge(newp, b));
/* <-- Writing slower, Running Faster -->
    insert1(rt, newp);
*/
}

// <-- Writing slower, Running Faster -->
// not necessary
void insert2(node* & rt, node* p, int pos){
  if(!rt){
    rt = p;
  }
  else if(p->pri < rt->pri){
    split_by_size(rt, p->l, p->r, pos);
    rt = p;
    pull(rt);
  }
  else{
    push(rt);
    if (pos <= size(rt->l))
      insert2(rt->l, p, pos);
    else
      insert2(rt->r, p, pos - (size(rt->l) + 1));
    pull(rt);
  }
}

// call this <--- insert item(val) after $pos$ items --->
void Insert_by_pos(node* &rt, int pos, int key, int val = 0){
    node *newp = new node(key);
    node *a, *b;
    split_by_size(rt, a, b, pos);
    rt = merge(a, merge(newp, b));
/* <-- Writing slower, Running Faster -->
    insert2(rt, newp, pos);
*/
}

bool erase (node * & rt, int key) {
    if(!rt)
        return false;
    if (rt->key == key)
    {
        node * del = rt;
        rt = merge(rt->l, rt->r);
        delete del;
        return true;
    }
    if(erase (key < rt->key ? rt->l : rt->r, key)){
        pull(rt);
        return true;
    }
    return false;
}

bool erase_by_pos(node*& rt, int pos){
    node* a, *b, *c;
  split_by_size(rt, a, b, pos-1);
  split_by_size(b, b, c, 1);
  rt = merge(a, c);
  delete b;
}

// return 0-th, 1-th, 2-th, means: greater than x items in
    tree.
int order_of_key(node * root, int key){
    if(!root)
        return 0;
    if(root->key < key)
        return size(root->l) + 1 + order_of_key(root->r, key)
            ;
    else
        return order_of_key(root->l, key);
}

node * find_by_order(node * root, int k){
    if (k <= size(root->l))
        return find_by_order(root->l, k);
    if (k == size(root->l) + 1)
        return root;
    return find_by_order(root->l, k - size(root->l) + 1);
}

/* range query max, range reverse */
// reverse range [l, r] in [1, n]
void reverse(node * &root, int l, int r){
    node *a, *b, *c;
    split_by_size(root, a, b, l - 1);
    split_by_size(b, b, c, r - l + 1);
    b->rev ^= 1;
    root = merge(a, merge(b, c));
}

/* range query max, range reverse */
// revolve by T times in range [l, r] in [1, n]
void Revolve(node* & rt, int l, int r, int T){
  int len = (r-l+1);
  T %= len;
  node* a, *b, *c;
  split_by_size(rt, a, b, l-1);
  split_by_size(b, b, c, len);

  node* b1, *b2;
  split_by_size(b, b1, b2, len - T);
  rt = merge(a, merge( merge(b2, b1), c) );
}

// query range [l, r] in [1, n]
int query(node *& root, int l, int r){
    node *a, *b, *c;
    split_by_size(root, a, b, l - 1);
    split_by_size(b, b, c, r - l + 1);
    int ans = b->ans;
    root = merge(a, merge(b, c));
    return ans;
}

void Modify(node* & rt, int l, int r, int val){
  node* a, *b, *c;
  split_by_size(rt, a, b, l-1);
  split_by_size(b, b, c, (r - l + 1));

  b->add += val;
  b->val += val;
  b->ans += val;
  rt = merge(a, merge(b, c));
}

void heapify(node * t)
{
    if (!t) return;
    node * max = t;
    if (t->l != nullptr && t->l->pri > max->pri)
        max = t->l;
    if (t->r != nullptr && t->r->pri > max->pri)
        max = t->r;
    if (max != t) {
        swap (t->pri, max->pri);
        heapify (max);
    }
}

// Construct a treap on values {a[0], a[1], ..., a[n - 1]} in
    O(n)
node * build (int * a, int n) {
    if (n == 0) return nullptr;
    int mid = n / 2;
    node * t = new node (a[mid]);
    t->l = build (a, mid);
    t->r = build (a + mid + 1, n - mid - 1);
    heapify (t);
    pull(t);
    return t;
}
```

3 geometry

3.1 closest_point

```
template <typename T>
```

```cpp
T ClosestPairSquareDistance(typename vector<Point<T>>::
    iterator l,
                            typename vector<Point<T>>::
                                iterator r)
{
    auto delta = numeric_limits<T>::max();
    if (r - l > 1)
    {
        auto m = l + (r - l >> 1);
        nth_element(l, m, r); // Lexicographical order in
            default
        auto x = m->x;
        delta = min(ClosestPairSquareDistance<T>(l, m),
                    ClosestPairSquareDistance<T>(m, r));
        auto square = [&](T y) { return y * y; };
        auto sgn = [=](T a, T b) {
            return square(a - b) <= delta ? 0 : a < b ? -1 :
                1;
        };
        vector<Point<T>> x_near[2];
        copy_if(l, m, back_inserter(x_near[0]), [=](Point<T>
            a) {
            return sgn(a.x, x) == 0;
        });
        copy_if(m, r, back_inserter(x_near[1]), [=](Point<T>
            a) {
            return sgn(a.x, x) == 0;
        });
        for (int i = 0, j = 0; i < x_near[0].size(); ++i)
        {
            while (j < x_near[1].size() and
                   sgn(x_near[1][j].y, x_near[0][i].y) == -1)
                ++j;
            for (int k = j; k < x_near[1].size() and
                            sgn(x_near[1][k].y, x_near[0][i].
                                y) == 0;
                 ++k)
            {
                delta = min(delta, (x_near[0][i] - x_near[1][
                    k]).norm());
            }
        }
        inplace_merge(l, m, r, [](Point<T> a, Point<T> b) {
            return a.y < b.y;
        });
    }
    return delta;
}
```

3.2 cp__geometry

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

const double eps = 1e-9;
double fcmp(double a, double b = 0, double eps = 1e-9) {
    if (abs(a-b) < eps) return 0;
    return a-b;
}

struct Point
{
    double x, y;
    Point() : x(0), y(0) {}
    Point(double x, double y) : x(x), y(y) {}

    Point operator+(Point b) {
        return Point(x + b.x, y + b.y);
    }

    Point operator-(Point b) {
        return Point(x - b.x, y - b.y);
    }

    Point operator*(double fac) {
        return Point(x * fac, y * fac);
    }

    Point operator/(double fac) {
        return Point(x / fac, y / fac);
    }

    double operator&(Point b) { return x * b.x + y * b.y; }
    // dot operator
    double operator^(Point b) { return x * b.y - y * b.x; }
    // cross operator

    bool operator==(Point b) const {
        return fcmp(x, b.x) == 0 && fcmp(y, b.y) == 0;
    }

    bool operator<(Point b) const {
        if (fcmp(x, b.x) == 0)
            return y < b.y;
        return x < b.x;
    }

    double norm() { return *this & *this; }    // 歐 式 長 度 平 方
    Point prep() { return Point(-y, x); } // 左 旋 直 角 法 向 量
};

// for pointOnSegment
bool collinearity(Point p1, Point p2, Point p3) {
    return fcmp((p1 - p3) ^ (p2 - p3)) == 0;
}

// for pointOnSegment
bool btw(Point p1, Point p2, Point p3) {
    return fcmp((p1 - p3) & (p2 - p3)) <= 0;
}

bool pointOnSegment(Point p1, Point p2, Point p3) {
    return collinearity(p1, p2, p3) && btw(p1, p2, p3);
}

struct Line
{
    Point sp, ep;

    Line() {}
    Line(Point sp, Point ep) : sp(sp), ep(ep) {}
    Line(double x1, double y1, double x2, double y2) : sp(
        Point(x1, y1)), ep(Point(x2, y2)) {}

    Point vec() { return ep - sp; }

    double ori(Point src) {
        return (ep - sp) ^ (src - sp);
    }

    // Regard a line as a function
    Point operator()(double x) { // A + AB * x = the point
        position.
        return sp + vec() * x;
    }

    bool isSegProperIntersection(Line l) {
        return l.ori(sp) * l.ori(ep) < 0 and ori(l.sp) * ori(
            l.ep) < 0;
    }

    bool isSegIntersection(Line l) {
        // hsp = 1, hep = 2, lsp = 3, lep = 4
        double hlsp = ori(l.sp);
        double hlep = ori(l.ep);
        double lhsp = l.ori(sp);
        double lhep = l.ori(ep);
        if(fcmp(hlsp, 0) == 0 and fcmp(hlep,0) == 0)
            return isPointOnSeg(l.sp) || isPointOnSeg(l.ep)
                || l.isPointOnSeg(sp) || l.isPointOnSeg(ep);

        return fcmp(hlsp * hlep) <= 0 and fcmp(lhsp * lhep)
            <= 0;
    }

    bool isPointOnSegProperly(Point p) {
        return fcmp(ori(p)) == 0 and fcmp(((sp - p) & (ep - p
            ))) < 0;
    }
    bool isPointOnSeg(Point p) {
        return fcmp(ori(p)) == 0 and fcmp((sp - p) & (ep - p)
            ) <= 0;
    }

    // notice you should check Segment intersect or not;
    // be careful divided by 0, like l entirely on Line
    Point getIntersection(Line l){
        double hlsp = -ori(l.sp);
        double hlep = ori(l.ep);
        return ((l.sp * hlep) + (l.ep * hlsp)) / (hlsp + hlep
            );
    }

    Point projection(Point p) {
        return operator()(((p - sp) & vec()) / vec().norm());
    }

    double distance(Point p) {
        return Line(projection(p), p).vec().norm();
    }
};

// sort by radian, the left is smaller for parallel lines
auto radCmp = [](Line A, Line B)
{
    Point a = A.vec(), b = B.vec();
    auto sgn = [](Point t) { return (t.y == 0 ? t.x : t.y) <
        0; }; // 0 for in [0, pi), 1 for [pi, 2*pi).
    if (sgn(a) != sgn(b)) // in different side
        return sgn(a) < sgn(b);
    else if (abs(a ^ b) == 0) // same
        return B.ori(A.sp) > 0;
```

```
    return (a ^ b) > 0;
};


// 以原點極角排序逆時針排一圈。最好用整數做，不然應該會有誤差
// 以某點須對點集合做offset 處理
inline bool up (point p) {
  return p.y > 0 or (p.y == 0 and p.x >= 0);
}

sort(v.begin(), v.end(), [] (point a, point b) {
  return up(a) == up(b) ? a.x * b.y > a.y * b.x : up(a) < up(b);
});
```

3.3 Geometry_Theories

3.3.1 Lattice Polygon and Pick's Theorem

A lattice polygon has integer coordinates for all of its vertices

Pick's Theorem : Let i = number of integer points interior the polygon, b = number of integer points on its boundary.

the area of polygon $= A = i + \frac{b}{2} - 1$

3.4 half_plane

```
// Redefine epsilon and infinity as necessary. Be mindful of precision errors.
const long double eps = 1e-9, inf = 1e9;

// Basic point/vector struct.
struct Point {

    long double x, y;
    explicit Point(long double x = 0, long double y = 0) : x(x), y(y) {}

    // Addition, substraction, multiply by constant, dot product, cross product.

    friend Point operator + (const Point& p, const Point& q) {
        return Point(p.x + q.x, p.y + q.y);
    }

    friend Point operator - (const Point& p, const Point& q) {
        return Point(p.x - q.x, p.y - q.y);
    }

    friend Point operator * (const Point& p, const long double& k) {
        return Point(p.x * k, p.y * k);
    }

    friend long double dot(const Point& p, const Point& q) {
        return p.x * q.x + p.y * q.y;
    }

    friend long double cross(const Point& p, const Point& q) {
        return p.x * q.y - p.y * q.x;
    }
};

// Basic half-plane struct.
struct Halfplane {

    // 'p' is a passing point of the line and 'pq' is the direction vector of the line.
    Point p, pq;
    long double angle;

    Halfplane() {}
    Halfplane(const Point& a, const Point& b) : p(a), pq(b - a) {
        angle = atan2l(pq.y, pq.x);
    }

    // Check if point 'r' is outside this half-plane.
    // Every half-plane allows the region to the LEFT of its line.
    bool out(const Point& r) {
        return cross(pq, r - p) < -eps;
    }

    // Comparator for sorting.
    bool operator < (const Halfplane& e) const {
        return angle < e.angle;
    }

    // Intersection point of the lines of two half-planes. It is assumed they're never parallel.
    friend Point inter(const Halfplane& s, const Halfplane& t) {
        long double alpha = cross((t.p - s.p), t.pq) / cross(s.pq, t.pq);
        return s.p + (s.pq * alpha);
    }
};

// Actual algorithm
vector<Point> hp_intersect(vector<Halfplane>& H) {

    Point box[4] = {  // Bounding box in CCW order
        Point(inf, inf),
        Point(-inf, inf),
        Point(-inf, -inf),
        Point(inf, -inf)
    };

    for(int i = 0; i<4; i++) { // Add bounding box half-planes.
        Halfplane aux(box[i], box[(i+1) % 4]);
        H.push_back(aux);
    }

    // Sort by angle and start algorithm
    sort(H.begin(), H.end());
    deque<Halfplane> dq;
    int len = 0;
    for(int i = 0; i < int(H.size()); i++) {

        // Remove from the back of the deque while last half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[len-1], dq[len-2]))) {
            dq.pop_back();
            --len;
        }

        // Remove from the front of the deque while first half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
            dq.pop_front();
            --len;
        }

        // Special case check: Parallel half-planes
        if (len > 0 && fabsl(cross(H[i].pq, dq[len-1].pq)) < eps) {
            // Opposite parallel half-planes that ended up checked against each other.
            if (dot(H[i].pq, dq[len-1].pq) < 0.0)
                return vector<Point>();

            // Same direction half-plane: keep only the leftmost half-plane.
            if (H[i].out(dq[len-1].p)) {
                dq.pop_back();
                --len;
            }
            else continue;
        }

        // Add new half-plane
        dq.push_back(H[i]);
        ++len;
    }

    // Final cleanup: Check half-planes at the front against the back and vice-versa
    while (len > 2 && dq[0].out(inter(dq[len-1], dq[len-2]))) {
        dq.pop_back();
        --len;
    }

    while (len > 2 && dq[len-1].out(inter(dq[0], dq[1]))) {
        dq.pop_front();
        --len;
    }

    // Report empty intersection if necessary
    if (len < 3) return vector<Point>();

    // Reconstruct the convex polygon from the remaining half-planes.
    vector<Point> ret(len);
    for(int i = 0; i+1 < len; i++) {
        ret[i] = inter(dq[i], dq[i+1]);
    }
    ret.back() = inter(dq[len-1], dq[0]);
    return ret;
}
```

3.5 slicing

4 geometry/Convex_Hull

4.1 Andrew's_Monotone_Chain

```cpp
using Polygon = vec<Point>;

Polygon getConvexHull(Polygon poly) {
    sort(poly.begin(), poly.end());

    Polygon hull;
    hull.reserve(poly.size() + 1);
    for (int round = 0; round < 2; round++) {
        int start = hull.size();
        for (Point &pt: poly) {
            while (hull.size() - start >= 2 && Line(hull[hull
                 .size() - 2], hull[hull.size() - 1]).ori(pt)
                  <= 0)
                hull.pop_back();

            hull.emplace_back(pt);
        }

        hull.pop_back();

        reverse(poly.begin(), poly.end());
    }

    if (hull.size() == 2 && hull[0] == hull[1])
        hull.pop_back();

    return hull;
}
```

5 graph

5.1 2SAT

```cpp
// 2-SAT (A or B) and (C or ^B) and (E) = true : O(n) = O(v+e
    )
/* common terms edge building
A : ^A -> A, which means A must be true
not A : A -> ^A , which means A must be false
A or B : ^A -> B, ^B -> A
not A or B : A -> B, ^B -> ^A
not A or not B : A -> ^B, B -> ^A
A xor B : A -> ^B, ^A -> B, B -> ^A, ^B -> A.
*/
struct twoSAT{
    Kosaraju mK;
    int n;
    vector<bool> value;
    void init(int nterm){
        this->n = nterm;
        mK.init(nterm * 2);
    }
    void addEdge(int u, int v){
        mK.addEdge(u, v);
    }
    int rv(int a) {
        if (a >= n) return a - n;
        return a + n;
    }
    void add_clause(int a, int b) { // a or b
        addEdge(rv(a), b), addEdge(rv(b), a);
    }
    void run(){
        mK.run();
    }
    bool satisfy(){ // assume A = i, then ^A = i+nterm
        value.clear();
        value.resize(n);
        for (int i = 0; i < n; i++){
            if(mK.scc[i] == mK.scc[i+n]){
                return false;
            }
            value[i] = mK.scc[i] > mK.scc[i + n];
        }
        return true;
    }
};
```

5.2 Formulas_or_Theorems_GYLin

1. Cayley's Formula: There are n^{n-2} spanning trees of a complete graph with n labelled vertices. (Also, $(n+1)^{n-1}$ labelled rooted forests.) Example: UVa 10843 - Anne's game.
   - The following generalizes Cayley's formula to labelled forests: Let $T_{n,k}$ be the number of labelled forests on n vertices with k connected components, such that vertices $1, 2, ..., k$ all belong to different connected components. Then $T_{n,k} = k \times n^{n-k-1}$.
2. Derangement: A permutation of the elements of a set such that non of the elements appear in their original position. The number of Derangements $der(n)$ can be computed as follow: $der(n) = (n-1) \times (der(n-1) + der(n-2))$, where $der(0) = 1$ and $der(1) = 0$. A basic problem involving derangement is UVa 12024 - Hats (see Section 5.6).
3. Erdos Gallai's Theorem gives a necessary and sufficient condition for a finite sequence of natural numbers to be the *degree sequence* of a simple graph. A sequence of nonnegative integers $d_1 \geq d_2 \geq ... \geq d_n$ can be the degree sequence of a simple graph on n vertices iff $\sum_{i=1}^{n} d_i$ is even and $\sum_{i=1}^{k} d_i \leq k \times (k-1) + \sum_{i=k+1}^{n} min(d_i, k)$ holds for $1 \leq k \leq n$. Example: UVa 10720 - Graph Construction.
   - (題目：已知一個無向圖的所有頂點的度，問能否構造成一個簡單圖)
   - 構成圖判定：所有點的度數和為偶數 (防止溢出可以只判斷奇偶)
   - Havel 定理：將所有邊排序，將度數最大的頂點依次與剩下的頂點連接邊 (從度數大的開始)，去掉度數最大的頂點後構成子問題，如果出現矛盾則失敗，否則成功;
4. Euler's Formula for Planar Graph: $V - E + F = 2$, where F is the number of faces of the Planar Graph. Example: UVa 10178 - Count the Faces.
5. Moser's Circle: Determine the number of pieces into which a circle is divided if n points on its circumference(圓周) are joined by chords with not three internally concurrent(三線交一點). Solution: $g(n) = C_4^n + C_2^n + 1$. Example: UVa 10213 - How Many Pieces of Lands?
6. Pick's Theorem: Let I be the number of integer points in the polygon, A be the area of the polygon, and b be the number of integer points on the boundary, then $A = i + \frac{b}{2} - 1$. Example: UVa 10088 - Trees on My Island.
7. The number of spanning tree of a complete bipartite graph $K_{n,m}$ is $m^{n-1} \times n^{m-1}$. Example: UVa 11719 - Gridlands Airport.

5.3 graph_Theories

5.3.1 Definition

- Vertex Cover: Pick some vertices s.t. each edge covered by a least one vertex
- Matching : Pick some edge s.t. no two edge share same vertex.
- Indepentent vertex Set: Pick some vertices s.t. no two vertices are neighbor.
- edge(vertex) cactus: A graph every edge(vertex) belongs to at most one simple cycle.

5.3.2 Konig's Theorem

In any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover.

5.3.3 Independent Set on Bipartite graph

In any bipartite graph, the complement of mimimum vertex cover is a maximum Independent set.

5.3.4 Minimum Weighted Vertex Cover

二分圖的 minimum weighted vertex cover 可以透過最大流求出，建模方式如下：source 連向所有左邊的點，capacity 是點權，所有右邊的點連向 sink，capacity 是點權，對於二分圖中原本有的邊，從左邊連向右邊，capacity 為 INF。可以透過此圖的 min cut 構造出 vertex cover，而 min cut 可以透過此圖的 max flow 求出。

5.3.5 Biconnected Component

If a Biconnected component is 2-connected

1. it has a least three vertices. (special case: 2 vertex and one edge)
2. 表示任兩點間存在兩條互斥路徑 (環)。且認兩邊可以找到一個環包含兩邊
3. 若該連通分量有一個奇環則雙連通分量的任一點都至少一個奇環覆蓋

5.3.6 DFS Tree

1. The back-edges of the graph all connect a vertex with its descendant in the spanning tree.
2. A back-edge is never a bridge. The edge between u and its parent is a bridge if and only if $dp[u] = 0$ ($dp[u] =$ (# of back-edges going up from u) $-$ (# of back-edges going down from u) $+ \sum_{v \text{ is a child of } u} dp[v]$).
3. In DFS tree of a *cactus*, for any span-edge, at most one back-edge passes over it. Each back-edge forms a simple cycle together with the span-edges it passes over. There's no other simple cycles. (contract cycleId$[v]$! $= v$&&there is no back-edge going down from v)

5.3.7 Euler Tour

1. An undirected graph has an Eulerian cycle iff $deg(u)$ is even for all u, and all vertices with $deg(u)! = 0$ belong to a c.c..
2. An undirected graph can be decomposed into edge-disjoint cycles iff all $deg(u)$ is even . So, a graph has an Eulerian cycle iff it can be decomposed into edge-disjoint cycles and its nonzero-degree vertices belong to a single c.c.
3. An undirected graph has an Eulerian trail iff exactly zero or two vertices have odd degree, and all of its vertices with nonzero degree belong to a single c.c.
4. A directed graph has an Eulerian cycle iff every vertex has equal in-deg and out-deg, and all vertices with nonzero degree belong to a single SCC. Equivalently, a directed graph has an Eulerian cycle iff it can be decomposed into edge-disjoint directed cycles and all with nonzero degree belong to a single SCC.
5. A directed graph has an Eulerian trail iff at most one vertex has (out-deg)−(in-deg) = 1, at most one vertex has (in-deg)−(out-deg) = 1, every other vertex has equal in-degree and out-degree, and all vertices with nonzero degree belong to a single c.c. of the underlying undirected graph.

5.4 Kosaraju_for_SCC

```cpp
// scc[u] will be a topological sort order of each SCC
struct Kosaraju{
    int NodeNum;
    vector<vector<int>> G;
    vector<vector<int>> GT;
    stack<int> st;
    vector<bool> visited;
    vector<int> scc;
    int sccNum;

    void init(int N){
        NodeNum = N;
        G.assign(N, vec<int>());
        GT.assign(N, vec<int>());
        while(!st.empty())
            st.pop();
        visited.assign(N, false);
        scc.assign(N, 0);
        sccNum = 0;
    }
    void addEdge(int u, int v){
        G[u].eb(v);
        GT[v].eb(u);
    }
    void DFS(bool isG, int u, int sccID = -1){
        visited[u] = true;
        vector<vector<int>> &dG = (isG ? G : GT);
        for(int v: dG[u])
        {
            if(!visited[v]){
                DFS(isG, v, sccID);
            }
        }
        if(isG){
            st.push(u);
        }
        else{
            scc[u] = sccID;
        }
    }
    void run(){
        fill(al(visited), false);
        for (int i = 0; i < NodeNum; i++){
            if(!visited[i])
                DFS(true, i);
        }
        fill(al(visited), false);
        while(!st.empty()){
            if(!visited[st.top()])
                DFS(false, st.top(), sccNum++);
            st.pop();
        }
    }

    vector<vector<int>> reduceG(){ //call after run
        vector<vector<int>> reG;
        reG.resize(sccNum);
        for (int i = 0; i < NodeNum; i++){
            for(int w: G[i]){
               if(scc[i] == scc[w])
                   continue;
               reG[scc[i]].emplace_back(scc[w]);
            }
        }
        return reG;
    }
};
```

5.5 Tarjan_for_BiconnectedCC

```cpp
struct edge
{
    int u, v;
    bool is_bridge;
    edge(int u = 0, int v = 0) : u(u), v(v), is_bridge(0) {}
};

vector<int> G[maxn]; // 1-base
vector<edge> E;
vector<int> nG[maxn], bcc[maxn];
int low[maxn], dfn[maxn], Time;
int bcc_id[maxn], bcc_cnt; // 1-base
bool is_cut[maxn];         // whether is av
bool cir[maxn];
int st[maxn], top;

void bcc_init(int n)
{
    Time = bcc_cnt = top = 0;
    for (int i = 1; i <= n; ++i)
        G[i].clear(), dfn[i] = bcc_id[i] = is_cut[i] = 0;
}

inline void add_edge(int u, int v)
{
    G[u].push_back(E.size());
    G[v].push_back(E.size());
    E.push_back(edge(u, v));
}

void dfs(int u, int pa = -1)// call dfs(u) for all unvisited
     node
{
    int child = 0;
    low[u] = dfn[u] = ++Time;
    st[top++] = u;
    for (int eid : G[u])
    {
        int v = E[eid].u ^ E[eid].v ^ u;
        if (!dfn[v])
        {
            dfs(v, u), ++child;
            low[u] = min(low[u], low[v]);
            if (dfn[u] <= low[v])
            {
                is_cut[u] = true; // 結 論 2 對於除了 root 點
                    以外的所有點 v，v 點在 G 上為 AP 的充要
                    條件為其在 T 中至少有一個子節點 w 滿足
                    dfn(v) ≤ low(w)
                // getting bcc
                bcc[++bcc_cnt].clear();
                int t;
                do
                {
                    bcc_id[t = st[--top]] = bcc_cnt;
                    bcc[bcc_cnt].push_back(t);
                } while (t != v);
                bcc_id[u] = bcc_cnt;
                bcc[bcc_cnt].eb(u);
            }
        }
        else if (dfn[v] < dfn[u] && v != pa) // !=pa vary
             important
            low[u] = min(low[u], dfn[v]);
    }
    if (pa == -1 && child < 2)
        is_cut[u] = false;
}

void bcc_solve(int n)
{
    for (int i = 1; i <= n; ++i)
        if (!dfn[i])
            dfs(i);
    // block-cut tree
    for (int i = 1; i <= n; ++i)
        if (is_cut[i])
            bcc_id[i] = ++bcc_cnt, cir[bcc_cnt] = 1;
    for (int i = 1; i <= bcc_cnt && !cir[i]; ++i)
        for (int j : bcc[i])
            if (is_cut[j])
                nG[i].pb(bcc_id[j]), nG[bcc_id[j]].pb(i);
}
```

5.6 Tarjan_for_BridgeCC

```cpp
struct edge
{
    int u, v;
    bool is_bridge;
    edge(int u = 0, int v = 0) : u(u), v(v), is_bridge(0) {}
};

vector<int> G[maxn]; // 1-base
```

```cpp
vector<edge> E;
int low[maxn], dfn[maxn], Time;
int bcc_id[maxn], bridge_cnt, bcc_cnt; // 1-base
int st[maxn], top;                     // BCC用

inline void add_edge(int u, int v)
{
    G[u].push_back(E.size());
    G[v].push_back(E.size());
    E.push_back(edge(u, v));
}

void dfs(int u, int pa)
{ // u當前點，re為u連接前一個點的邊
    int v;
    low[u] = dfn[u] = ++Time;
    st[top++] = u;
    for (size_t eid : G[u])
    {
        int v = E[eid].u ^ E[eid].v ^ u;
        if (!dfn[v])
        {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if (dfn[u] < low[v])// 結 論 3 對於包含 r 在內的所有點 v 和 v 在 T 中的子節點 w，邊 e(v, w) 在圖 G 中為bridge 的充要條件為 dfn(v) < low(w)。
            {
                E[eid].is_bridge = 1;
                ++bridge_cnt;
            }
        }
        else if (dfn[v] < dfn[u] && v != pa)
            low[u] = min(low[u], dfn[v]);
    }
    if (dfn[u] == low[u])   //處理BridgeCC
    {
        ++bcc_cnt; // 1-base
        do
            bcc_id[v = st[--top]] = bcc_cnt; //每個點所在的BCC
        while (v != u);
    }
}
inline void bcc_init(int n)
{
    Time = bcc_cnt = bridge_cnt = top = 0;
    E.clear();
    for (int i = 1; i <= n; ++i)
    {
        G[i].clear();
        dfn[i] = 0;
        bcc_id[i] = 0;
    }
}
```

5.7 Tarjan_for_SCC

```cpp
// by atsushi
// sccID[u] will be a REVERSED topological sort order of each SCC
struct tarjan_for_SCC{
    vector<vector<int>> G; // adjacency list
    vector<int> dfn;
    vector<int> low;
    vector<int> sccID;
    stack<int> st; // for SccID
    vector<bool> inSt;
    vector<vector<int>> conG; // contracted graph
    int Time, sccNum;
    void init(int n = 1){ // 0-base
        G.assign(n, vec<int>());
        dfn.assign(n, 0);
        low.assign(n, 0);
        sccID.assign(n, 0);
        inSt.assign(n, false);
        while(!st.empty())
            st.pop();
        conG.clear();
        sccNum = Time = 0;
    }
    void addEdge(int from, int to){
        G[from].eb(to);
    }
    void dfs(int u){ //call DFS(u) for all unvisited vertex
        dfn[u] = low[u] = ++Time; //timestamp > 0
        st.push(u);
        inSt[u] = true;

        for(int v: G[u]){
            if(!dfn[v]){ // dfn[v] = 0 if not visited
                dfs(v);
                low[u] = min(low[u], low[v]);
            }else if(inSt[v])
            { /* v has been visited.
                 if we don't add this, the low[u] will think that u can back to node whose index less to u.
                 inSt[v] is true that u -> v is a cross edge opposite it's a forward edge
            */
                low[u] = min(low[u], dfn[v]);
            }
        }
        if(dfn[u] == low[u]){
            int v;
            do{
                v = st.top(), st.pop();
                sccID[v] = sccNum, inSt[v] = false;
            } while (v != u);
            sccNum++;
        }
    }
    // generate induced graph.
    void generateReG(){
        conG.assign(sccNum, vec<int>());
        for (int u = 0; u < G.size(); u++){
            for(int v: G[u]){
                if(sccID[u] == sccID[v])
                    continue;
                conG[sccID[u]].emplace_back(sccID[v]);
            }
        }
    }
};
```

6 graph/Bipartite

6.1 konig_algorithm

```cpp
const int maxn = 250;
// time complexity: O(EV), V times DFS
// G[i]記錄了左半邊可以配到右邊的那些點
/* bipartite graph be like..
0\ /-0
1-X--1
2/ \/2
3  /\3
4 /  4
5/   5
.    .
.    .
*/
// match[i] 記錄了右半邊配對到左半邊的哪個點
vec<int> G[maxn];
int match[maxn]; // A <=B
bool used[maxn];
bool dfs(int v)
{
    for(int e:G[v])
    {
        if( used[e] ) continue;
        used[e] = true;
        if( match[e] == -1 || dfs( match[e] ) )
        {
            match[e] = v;
            return true;
        }
    }
    return false;
}
int konig(int n) // num of vertices of left side
{
    memset(match,-1,sizeof(match));

    int ans=0;

    for(int i=0;i<n;++i)
    {
        memset(used, 0, sizeof(used));
        if( dfs(i) )
            ans++;
    }

    return ans;
}
void addedge(int u, int v){ // left side, right side
    G[u].eb(v);
}
```

6.2 Kuhn-Munkres

```cpp
// Max weight perfect bipartite matching
// O(V^3)
// by jinkela
#define MAXN 405
```

```cpp
#define INF 0x3f3f3f3f3f3f3f3f
int n;// 1-base，0表示沒有匹配
LL g[MAXN][MAXN]; //input graph
int My[MAXN],Mx[MAXN]; //output match
LL lx[MAXN],ly[MAXN],pa[MAXN],Sy[MAXN];
bool vx[MAXN],vy[MAXN];
void augment(int y){
  for(int x, z; y; y = z){
    x=pa[y],z=Mx[x];
    My[y]=x,Mx[x]=y;
  }
}
void bfs(int st){
  for(int i=1; i<=n; ++i)
    Sy[i] = INF, vx[i]=vy[i]=0;
  queue<int> q; q.push(st);
  for(;;){
    while(q.size()){
      int x=q.front(); q.pop();
      vx[x]=1;
      for(int y=1; y<=n; ++y) if(!vy[y]){
        LL t = lx[x]+ly[y]-g[x][y];
        if(t==0){
          pa[y]=x;
          if(!My[y]){augment(y);return;}
          vy[y]=1,q.push(My[y]);
        }else if(Sy[y]>t) pa[y]=x,Sy[y]=t;
      }
    }
    LL cut = INF;
    for(int y=1; y<=n; ++y)
      if(!vy[y]&&cut>Sy[y]) cut=Sy[y];
    for(int j=1; j<=n; ++j){
      if(vx[j]) lx[j] -= cut;
      if(vy[j]) ly[j] += cut;
      else Sy[j] -= cut;
    }
    for(int y=1; y<=n; ++y){
      if(!vy[y]&&Sy[y]==0){
        if(!My[y]){augment(y);return;}
        vy[y]=1, q.push(My[y]);
      }
    }
  }
}
LL KM(){
  memset(My,0,sizeof(int)*(n+1));
  memset(Mx,0,sizeof(int)*(n+1));
  memset(ly,0,sizeof(LL)*(n+1));
  for(int x=1; x<=n; ++x){
    lx[x] = -INF;
    for(int y=1; y<=n; ++y)
      lx[x] = max(lx[x],g[x][y]);
  }
  for(int x=1; x<=n; ++x) bfs(x);
  LL ans = 0;
  for(int y=1; y<=n; ++y) ans+=g[My[y]][y];
  return ans;
}
```

7 graph/Flow

7.1 Dinic_algorithm

```cpp
// O(V^2E) O(VE) finding argument path
// if unit capacity network then O(min(V^(2/3), E^1/2) E)
// solving bipartite matching O(E sqrt(V)) better than konig and flow(EV)

struct FlowEdge {
    int u, v;
    long long cap, flow = 0;
    FlowEdge(int u, int v, long long cap) : u(u), v(v), cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int u, int v, long long cap) {
        edges.emplace_back(u, v, cap);
        edges.emplace_back(v, u, 0);
        adj[u].push_back(m);
        adj[v].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int id : adj[u]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].v] != -1) continue;
                level[edges[id].v] = level[u] + 1;
                q.push(edges[id].v);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int u, long long pushed) {
        if (pushed == 0) return 0;
        if (u == t) return pushed;

        for (int& cid = ptr[u]; cid < (int)adj[u].size(); cid++) {
            int id = adj[u][cid];
            int v = edges[id].v;
            if (level[u] + 1 != level[v] || edges[id].cap - edges[id].flow < 1)
                continue;

            long long tr = dfs(v, min(pushed, edges[id].cap - edges[id].flow));

            if (tr == 0) continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }

        level[u] = -1;
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs()) break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

7.2 Edmonds-Karp-adjmax

```cpp
// O((V+E)VE) ，簡單寫成 O(VE²)
#include <cstring>
#include <queue>
using namespace std;
#define maxn 100
typedef int Graph[MAXN][MAXN];  // adjacency matrix
Graph C, F, R;  // 容量上限、流量、剩餘容量
bool visit[MAXN]; // BFS經過的點
int path[MAXN];   // BFS tree
int flow[MAXN];   // 源點到各點的流量瓶頸

int BFS(int s, int t)   // 源點與匯點
{
    memset(visit, false, sizeof(visit));

    queue<int> Q;   // BFS queue
    visit[s] = true;
    path[s] = s;
    flow[s] = 1e9;
    Q.push(s);

    while (!Q.empty())
    {
        int i = Q.front(); Q.pop();
        for (int j=0; j<100; ++j)
            // 剩餘網路找擴充路徑
            if (!visit[j] && R[i][j] > 0)
```

```
            {
                visit[j] = true;
                path[j] = i;
                // 一邊找最短路徑，一邊計算流量瓶頸。
                flow[j] = min(flow[i], R[i][j]);
                Q.push(j);

                if (j == t) return flow[t];
            }
    }
    return 0;   // 找不到擴充路徑了，流量為零。
}

int Edmonds_Karp(int s, int t)
{
    memset(F, 0, sizeof(F));
    memcpy(R, C, sizeof(C));

    int f, df;  // 最大流的流量、擴充路徑的流量
    for (f=0; df=BFS(s, t); f+=df)
        // 更新擴充路徑上每一條邊的流量
        for (int i=path[t], j=t; i!=j; i=path[j=i])
        {
            F[i][j] = F[i][j] + df;
            F[j][i] = -F[i][j];
            R[i][j] = C[i][j] - F[i][j];
            R[j][i] = C[j][i] - F[j][i];
        }
    return f;
}
```

7.3 Edmonds_Karp_2

```
#include <bits/stdc++.h>
struct Edge{
    int from, to, cap, flow;
    Edge(int u, int v, int c, int f):from(u), to(v), cap(c), flow(f){}
};
const maxn = 200005;
struct EdmondsKarp{
    int n, m;
    vector<Edge> edges;
    vector<int> G[maxn];
    int a[maxn];
    int p[maxn];
    void init(int n){
        for (int i = 0; i < n; i++)
            G[i].clear();
        edges.clear();
    }
    void AddEdge(int from, int to, int cap){
        edges.push_back(Edge(from, to, cap, 0));
        edges.push_back(Edge(to, from, 0, 0)) // 反向弧
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }
    int Maxflow(int s, int t){
        int flow = 0;
        for (;;){
            memset(a, 0, sizeof(a));
            queue<int> Q;
            Q.push(s);
            a[s] = INF;
            while(!Q.empty()){
                int x = Q.front();
                Q.pop();
                for (int i = 0; i < G[x].size(); i++){
                    Edge &e = edges[G[x][i]];
                    if(!a[e.to] && e.cap > e.flow){
                        p[e.to] = G[x][i];
                        a.[e.to] = min(a[x], e.cap - e.flow);
                        Q.push(e.to);
                    }
                }
                if(a[t])
                    break;
            }
        }
        if(!a[t])
            break;
        for (int u = t; u != s; u = edges[p[u]].from){
            edges[p[u]].flow += a[t];
            edges[p[u] ^ 1].flow -= a[t];
        }
        flow += a[t];
    }
    return flow;
}
```

7.4 Ford_Fulkerson

```
#include <vector>
#include <tuple>
#include <cstring>
using namespace std;

// O((V+E)F)
#define maxn 101
// remember to change used into the maxNode size -- kattis elementary math
bool used[MAXN];
int End;
vector<int> V[MAXN];
vector<tuple<int, int>> E;

// x=>y 可以流 C
// if undirected or 2-direc edge, bakcward Capacity become C;
// Graph build by edge array
// 反向邊的編號只要把自己的編號 xor 1 就能取得
void add_edge(int x, int y,int c)
{
    V[x].emplace_back( E.size() );
    E.emplace_back(y,c);
    V[y].emplace_back( E.size() );
    E.emplace_back(x,0);
}
int dfs(int v, int f)
{
    if( v==End ) return f;
    used[v] = true;
    int e,w;
    for( int eid : V[v] )
    {
        tie(e,w) = E[eid];
        if( used[e] || w==0 ) continue;

        w = dfs(e, min(w,f));
        if( w>0 )
        {
            // 更新流量
            get<1>(E[eid  ]) -= w;
            get<1>(E[eid^1]) += w;
            return w;
        }
    }
    return 0;// Fail!
}
int ffa(int s,int e)
{
    int ans = 0, f;
    End = e;
    while(true)
    {
        memset(used, 0, sizeof(used));
        f = dfs(s, INT_MAX);
        if( f<=0 ) break;
        ans += f;
    }
    return ans;
}
```

7.5 MinCostMaxFlow-cp

```
struct CostFlow {
    static const int MAXN = 350;
    const ll INF = 1ll<<60;

    struct Edge {
        int to, r;
        ll rest, c;

        Edge() {}
        Edge(int to, int r, ll rest, ll c) : to(to), r(r), rest(rest), c(c) {}
    };

    int n, pre[MAXN], preL[MAXN];
    bool inq[MAXN];
    ll dis[MAXN], fl, cost;
    vec<Edge> G[MAXN];

    void init() {
        for (int i = 0; i < MAXN; i++) G[i].clear();
    }

    void add_edge(int u, int v, ll rest, ll c) {
        G[u].eb(v, G[v].size(), rest, c);
        G[v].eb(u, G[u].size()-1, 0, -c);
    }

    p<ll> flow(int s, int t) {
        fl = cost = 0;

        while (true) {
            fill(dis, dis+MAXN, INF);
```

```
            fill(inq, inq+MAXN, 0);
            dis[s] = 0;

            queue<int> que;
            que.emplace(s);
            while (!que.empty()) {
                int u = que.front();
                que.pop();

                inq[u] = 0;
                for (int i = 0; i < G[u].size(); i++) {
                    int v = G[u][i].to;
                    ll w = G[u][i].c;

                    if (G[u][i].rest > 0 and dis[v] > dis[u]
                         + w) {
                        pre[v] = u;
                        preL[v] = i;
                        dis[v] = dis[u] + w;
                        if (!inq[v]) {
                            inq[v] = 1;
                            que.emplace(v);
                        }
                    }
                }
            }

            if (dis[t] == INF) break;
            ll tf = INF;
            for (int v = t, u, l; v != s; v = u) {
                u = pre[v]; l = preL[v];
                tf = min(tf, G[u][l].rest);
            }

            for (int v = t, u, l; v != s; v = u) {
                u = pre[v]; l = preL[v];
                G[u][l].rest -= tf;
                G[v][G[u][l].r].rest += tf;
            }

            cost += tf * dis[t];
            fl += tf;
        }

        return {fl, cost};
    }
};
```

7.6 MinCostMaxFlow

```
// by jinkela
template <typename TP>
struct MCMF
{
  static const int MAXN = 440;
  static const TP INF = 999999999;
  struct edge
  {
    int v, pre;
    TP r, cost;
    edge(int v, int pre, TP r, TP cost) : v(v), pre(pre), r(r
         ), cost(cost) {}
  };
  int n, S, T;
  TP dis[MAXN], PIS, ans;
  bool vis[MAXN];
  vector<edge> e;
  int g[MAXN];
  void init(int _n)
  {
    memset(g, -1, sizeof(int) * ((n = _n) + 1));
    e.clear();
  }
  void add_edge(int u, int v, TP r, TP cost, bool directed =
       false)
  {
    e.push_back(edge(v, g[u], r, cost));
    g[u] = e.size() - 1;
    e.(
      edge(u, g[v], directed ? 0 : r, -cost));
    g[v] = e.size() - 1;
  }
  TP augment(int u, TP CF)
  {
    if (u == T || !CF)
      return ans += PIS * CF, CF;
    vis[u] = 1;
    TP r = CF, d;
    for (int i = g[u]; ~i; i = e[i].pre)
    {
      if (e[i].r && !e[i].cost && !vis[e[i].v])
      {
        d = augment(e[i].v, min(r, e[i].r));
        e[i].r -= d;
        e[i ^ 1].r += d;
        if (!(r -= d))
          break;
      }
    }
    return CF - r;
  }
  bool modlabel()
  {
    for (int u = 0; u <= n; ++u)
      dis[u] = INF;
    static deque<int> q;
    dis[T] = 0, q.push_back(T);
    while (q.size())
    {
      int u = q.front();
      q.pop_front();
      TP dt;
      for (int i = g[u]; ~i; i = e[i].pre)
      {
        if (e[i ^ 1].r && (dt = dis[u] - e[i].cost) < dis[e[i
             ].v])
        {
          if ((dis[e[i].v] = dt) <= dis[q.size() ? q.front()
               : S])
          {
            q.push_front(e[i].v);
          }
          else
            q.push_back(e[i].v);
        }
      }
    }
    for (int u = 0; u <= n; ++u)
      for (int i = g[u]; ~i; i = e[i].pre)
        e[i].cost += dis[e[i].v] - dis[u];
    return PIS += dis[S], dis[S] < INF;
  }
  TP mincost(int s, int t)
  {
    S = s, T = t;
    PIS = ans = 0;
    while (modlabel())
    {
      do
        memset(vis, 0, sizeof(bool) * (n + 1));
      while (augment(S, INF));
    }
    return ans;
  }
};
```

8 graph/Matching

8.1 blossom_matching

```
// by jinkela
// 最大圖匹配
//  O(V²(V+E))
#define MAXN 505
int n; //1-base
vector<int> g[MAXN];
int MH[MAXN]; //output MH
int pa[MAXN],st[MAXN],S[MAXN],v[MAXN],t;
int lca(int x,int y){
  for(++t;;swap(x,y)){
    if(!x) continue;
    if(v[x]==t) return x;
    v[x] = t;
    x = st[pa[MH[x]]];
  }
}
#define qpush(x) q.push(x),S[x]=0
void flower(int x,int y,int l,queue<int>&q){
  while(st[x]!=l){
    pa[x]=y;
    if(S[y=MH[x]]==1)qpush(y);
    st[x]=st[y]=l, x=pa[y];
  }
}
bool bfs(int x){
  iota(st+1, st+n+1, 1);
  memset(S+1,-1,sizeof(int)*n);
  queue<int>q; qpush(x);
  while(q.size()){
    x=q.front(),q.pop();
    for(int y:g[x]){
      if(S[y]==-1){
        pa[y]=x,S[y]=1;
        if(!MH[y]){
          for(int lst;x;y=lst,x=pa[y])
            lst=MH[x],MH[x]=y,MH[y]=x;
          return 1;
        }
        qpush(MH[y]);
      }else if(!S[y]&&st[y]!=st[x]){
```

```
        int l=lca(y,x);
        flower(y,x,l,q),flower(x,y,l,q);
      }
    }
  }
  return 0;
}
int blossom(){
  memset(MH+1,0,sizeof(int)*n);
  int ans=0;
  for(int i=1; i<=n; ++i)
    if(!MH[i]&&bfs(i)) ++ans;
  return ans;
}
```

9 graph/Minimum_Spanning_Tree

9.1 Kruskal

```
#include <tuple>
#include <vector>
#include <algorithm>
#include <numeric> // for iota(first, last, val) setting iterator value
using namespace std;

struct DSU // disjoint set no rank-comp-merge
{
    vector<int> fa;
    DSU(int n) : fa(n) { iota(fa.begin(), fa.end(), 0); } // auto fill fa from 0 to n-1
    int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }
    void merge(int x, int y) { fa[find(x)] = find(y); }
};
int kruskal(int V, vector<tuple<int, int, int>> E) // save all edges into E, instead of saving graph via adjacency list
{
    sort(E.begin(), E.end());
    DSU dsu(V);
    int mcnt = 0;
    int ans = 0;
    for (auto e : E)
    {
        int w, u, v; // w for start, u for des, v for val
        tie(w, u, v) = e;
        if (dsu.find(u) == dsu.find(v))
            continue;
        dsu.merge(u, v);
        ans += w;
        if (++mcnt == V - 1)
            break;
    }
    return ans;
}
```

9.2 prim

```
#include <vector>
#include <queue>
#include <utility>
using namespace std;
#define enp pair<int, int> // pair<edge_val, node>
int prim_pq(vector<vector<enp>> E){
    vector<bool> vis;
    vis.resize(E.size(), false);
    vis[0] = true;
    priority_queue<enp> pq;
    for(auto e: E[0]){
        pq.emplace(-e.first, e.second);
    }
    int ans = 0; // min value for MST
    while(pq.size()){
        int w, v; // edge-weight, vertex index
        tie(w, v) = pq.top();
        pq.pop();
        if(vis[v])
            continue;
        w = -w;
        vis[v] = true;
        ans += w;
        for(auto e: E[v]){
            pq.emplace(-e.first, e.second);
        }
    }
    return ans;
}
```

10 graph/Shortest_Path

10.1 bellman-ford

```
vector<tuple<int, int, int>> edges;
vector<int> dis;
const int inf = 0x3f3f3f3f;
// return true if contain cycles
bool Bellman_Ford(int src)
{
    int V; // # of vertices
    int E = edges.size();
    dis.resize(V, inf);
    dis[src] = 0;
    for (int i = 0; i < V - 1; i++)
    {
        for (int j = 0; j < E; j++){
            int u, v, w;
            tie(u, v, w) = edges[j];
            if(dis[u] != inf && dis[u] + w < dis[v]){
                dis[v] = dis[u] + w;
            }
        }
    }
    for (int j = 0; j < E; j++){
        int u, v, w;
        tie(u, v, w) = edges[j];
        if(dis[u] != inf && dis[u] + w < dis[v]){
            return true;
        }
    }
    return false;
}
```

10.2 dijkstra

```
vec<vec<p<int>>> Graph; // (w, v)

vec<int> dis; // distance result
void dijkstra(int u) {
    priority_queue<p<int>, vec<p<int>>, greater<p<int>>> pq;

    dis[u] = 0;
    pq.emplace(0, u);

    while(pq.size()){
        auto cur = pq.top();
        pq.pop();

        if(cur.first != dis[cur.second])
            continue;

        for (auto it: Graph[cur.second]){
            if (cur.first + it.first < dis[it.second]){
                dis[it.second] = cur.first + it.first;
                pq.emplace(dis[it.second], it.second);
            }
        }
    }
}
```

10.3 SPFA

```
vector<vector<pii>> G; // (w, v)
vector<int> dis;
const int inf = 0x3f3f3f3f;

void SPFA(int src){
    int V = G.size();
    dis.resize(V, inf);
    vector<bool> inq(V, false);
    vector<int> Q;
    dis[src] = 0, inq[src] = true, Q.push(src);
    while(q.size()){
        int u = Q.front();
        inq[u] = false, Q.pop();
        for(pii& e: G[u]){
            int w, v;
            tie(w, v) = e;
            if(dis[u] + w < dis[v]){
                dis[v] = dis[u] + w;
                if(inq[v] == false)
                    Q.push(v);
                inq[v] = true;
            }
        }
    }
}
```

11 graph/Tree

11.1 backpack_onTree

```
// 樹上依賴背包問題
// 上下界優化 Time complexity = O(NM)
// 另有Postorder 的順序做DP也能做到O(NM)
void dfs(int u)
{
    siz[u]=1;
    f[u][1]=a[u];
    int i,j,k,v;
    for (i=head[u];i;i=nxt[i])
    {
        v=to[i];
        dfs(v);
        for (j=min(m+1,siz[u]+siz[v]);j>=1;--j)
        {
            for (k=max(1,j-siz[u]);k<=siz[v]&&k<j;++k)
            {
                f[u][j]=max(f[u][j],f[u][j-k]+f[v][k]);
            }
        }
        siz[u]+=siz[v];
    }
}
```

11.2 Heavy_Light_Decompocision

```
//subT[v] : 以v為根的子樹節點數, 要先預處理
//linkR[v] : v所在的長鏈中最靠近root的節點
//linkP[v] : v與linkR[v]的距離
void HLD(int root) {
  linkR[root] = root;
  linkP[root] = 0;
  queue<int> Q;
  Q.emplace(root);

  while(!Q.empty()) {
    int v = Q.front(); Q.pop();
    pii res(-1, -1);
    for(auto &e : Tree[v]) {
      if(linkP[e] != -1) continue;
      if(subT[e] > res.first) res = pii(subT[e], e);
      Q.emplace(e);
    }


    if(res.second == -1) continue;
    linkR[res.second] = linkR[v];
    linkP[res.second] = linkP[v] + 1;

    for(auto &e : Tree[v]) {
      if(linkP[e] != -1) continue;
      linkR[e] = e;
      linkP[e] = 0;
    }
  }
}
```

11.3 Lowest_Common_Ancestor

```
#define MAXN 200005
int N = MAXN;
int pa[31][MAXN]; // pa(i, u), vertex u's 2^i ancestor.
void ComputeP()
{
    for (int i = 1; i < lgN; ++i)  // i = 0 is pre-built
    {
        for (int x = 0; x < N; ++x)
        {
            pa[i][x] = (pa[i - 1][x] == -1 ? -1 : pa[i - 1][
                pa[i - 1][x]]);
        }
    }
}
/* Binary Search Version */
int D[MAXN], L[MAXN];
vec<vec<int>> G;
int tstamp = 0;
// call this first
void DFS(int u, int pa){
    D[u] = tstamp++;
    for(int v: G[u]){
        if( v == pa ) continue;
        DFS(v, u);
    }
    L[u] = tstamp++;
}
bool isPa(int u, int v){
    return D[u] <= D[v] && L[u] >= D[v];
}

int LCA(int u, int v){
    if(isPa(u,v))
        return u;
    if(isPa(v,u))
        return v;
    for (int i = 30; i >= 0; i--){
        if(pa[i][u] != -1 && !isPa(pa[i][u], v))
            u = pa[i][u];
    }
    return pa[0][u];
}


/* jump up version */
int D[MAXN]; // depth
int LCA(int u, int v)
{
    if (D[u] > D[v])
        swap(u, v);
    int s = D[v] - D[u];
    for (int i = 0; i < 31; ++i) // adjust to same depth
        if (s & (1 << i))
            v = pa[i][v];

    if (u == v)
        return v;

    // because they are at same depth
    // jump up if they are different
    // think about that if P[u][i] == P[v][i]
    // then that point must be the ancestor of LCA or LCA
        itself
    // by this, we will stop at LCA's child
    for (int i = 31 - 1; i >= 0; --i)
        //
        if (pa[i][u] != pa[i][v])
        {
            u = pa[i][u];
            v = pa[i][v];
        }
    return pa[0][u];
}
```

11.4 minimax

```
const inf = 0x3f3f3f3f;
int alpha_beta(int u, int alph = -inf, int beta = inf, bool
     is_max) { //
  if (!son_num[u]) return val[u];
  if (is_max) {
    for (int i = 0; i < son_num[u]; ++i) {
      int d = son[u][i];
      alph = max(alph, alpha_beta(d, alph, beta, is_max ^ 1))
          ;
      if (alph >= beta) break;
    }
    return alph;
  } else {
    for (int i = 0; i < son_num[u]; ++i) {
      int d = son[u][i];
      beta = min(beta, alpha_beta(d, alph, beta, is_max ^ 1))
          ;
      if (alph >= beta) break;
    }
    return beta;
  }
}
```

11.5 Tree_Centroid

```
// Tree_Centroid
vector<int> G[20000];
int N;
int centroid;
int centroid_subtree_sz;
int tree_centroid(int u, int pa)
{
    int sz = 1; // tree size of u.
    int maxsub = 0; // max subtree size of u

    for(int v:G[u])
    {
        if (v==pa)continue;
        int sub = tree_centroid(v, u);
        maxsub = max(maxsub, sub);
        sz += sub;
    }
    maxsub = max(maxsub, N-sz);

    if (maxsub <= N/2)
    {
        centroid = u;
```

```
        centroid_subtree_sz = maxsub;
    }
    return sz;
}
```

12 hashing

12.1 hashingVec

```
const ll Prime = 0xdefaced;

struct VectorHash {
    size_t operator()(const std::vector<int>& v) const {
        std::hash<int> hasher;
        size_t seed = 0;
        for (const int& i : v) {
            seed ^= hasher(i) + 0x9e3779b9 + (seed<<6) + (
                seed>>2);
        }
        return seed;
    }
};

std::unordered_set<std::vector<int>, VectorHash> H;
```

13 number_theory

13.1 Biginteger

```
struct BigInteger {
    static const int BASE = 100000000;
    static const int WIDTH = 8;
    vec<int> s;

    BigInteger(long long num = 0) { *this = num; }
    BigInteger operator = (long long num)   {
        s.clear();
        do{
            s.push_back(num % BASE);
            num /= BASE;
        } while (num > 0);
        return *this;
    }

    BigInteger operator = (const string& str){
        s.clear();
        int x, len = (str.length() - 1) / WIDTH + 1;
        for (int i = 0; i < len;i++){
            int end = str.length() - i * WIDTH;
            int start = max(0, end - WIDTH);
            sscanf(str.substr(start, end - start).c_str(), "%
                d", &x);
            s.push_back(x);
        }
        return *this;
    }

    BigInteger operator+ (const BigInteger b) const{
        BigInteger c;
        c.s.clear();
        for(int i=0,g=0;;i++){
            if(g== 0 && i >=s.size() && i >=b.s.size())
                break;
            int x = g;
            if(i<s.size()) x+=s[i];
            if(i<b.s.size()) x+=b.s[i];
            c.s.push_back(x % BASE);
            g = x/ BASE;
        }
        return c;
    }

    BigInteger operator+=(const BigInteger& b){
        *this = *this + b;
        return *this;
    }

    BigInteger operator* (const BigInteger b)const{
        BigInteger c;
        c.s.clear();
        long long mul;
        for (int i = 0;i < s.size(); i++)
        {
            long long carry = 0;
            for (int g = 0; g < b.s.size();g++){
                mul = (long long)(s[i]) * (long long)(b.s[g])
                    ;
                mul += carry;
                if(i + g < c.s.size()){
                    c.s[i+g] += mul % BASE;
                }else{
                    c.s.push_back(mul % BASE);
                }
                carry = mul / BASE;
            }
        }
        for (int i = 0; i < c.s.size(); i++){
            if(c.s[i] >= BASE){
                if(i + 1 < c.s.size()){
                    c.s.push_back(c.s[i] / BASE);
                }else{
                    c.s[i + 1] += c.s[i] / BASE;
                }
                c.s[i] %= BASE;
            }
        }
        return c;
    }

    bool operator< (const BigInteger& b) const{
        if(s.size() != b.s.size()) return s.size() < b.s.size
            ();
        for(int i=s.size() -1 ; i>=0;i--)
            if(s[i] != b.s[i]) return s[i] < b.s[i];
        return false; // Equal
    }

    bool operator> (const BigInteger& b) const{return b < *
        this;}
    bool operator<= (const BigInteger& b) const {return !(b<*
        this);}
    bool operator>=(const BigInteger& b) const {return !(*
        this < b);}
    bool operator!=(const BigInteger& b) const {return b< *
        this || *this < b;}
    bool operator==(const BigInteger& b)const {return !(b<*
        this) && !(*this<b);}
};

ostream& operator<< (ostream &out, const BigInteger& x){
    out << x.s.back();
    for (int i = x.s.size() - 2;i >= 0;i--){
        char buf[20];
        sprintf(buf,"%08d",x.s[i]);
        for(int j = 0;j<strlen(buf);j++) out << buf[j];
    }
    return out;
}

istream& operator>> (istream &in, BigInteger & x){
    string s;
    if(!(in >> s)) return in;
    x= s;
    return in;
}
```

13.2 BigInteger2

```
class BigInt{
 private:
  using lld = int_fast64_t;
  #define PRINTF_ARG PRIdFAST64
  #define LOG_BASE_STR "9"
  static constexpr lld BASE = 1000000000;
  static constexpr int LOG_BASE = 9;
  vector<lld> dig; bool neg;
  inline int len() const { return (int) dig.size(); }
  inline int cmp_minus(const BigInt& a) const {
    if(len() == 0 && a.len() == 0) return 0;
    if(neg ^ a.neg)return a.neg ^ 1;
    if(len()!=a.len())
      return neg?a.len()-len():len()-a.len();
    for(int i=len()-1;i>=0;i--) if(dig[i]!=a.dig[i])
      return neg?a.dig[i]-dig[i]:dig[i]-a.dig[i];
    return 0;
  }
  inline void trim(){
    while(!dig.empty()&&!dig.back())dig.pop_back();
    if(dig.empty()) neg = false;
  }
 public:
  BigInt(): dig(vector<lld>()), neg(false){}
  BigInt(lld a): dig(vector<lld>()){
    neg = a<0; dig.push_back(abs(a));
    trim();
  }
  BigInt(const string& a): dig(vector<lld>()){
    assert(!a.empty()); neg = (a[0]=='-');
    for(int i=((int)a.size())-1;i>=neg;i-=LOG_BASE){
      lld cur = 0;
      for(int j=min(LOG_BASE-1,i-neg);j>=0;j--)
        cur = cur*10+a[i-j]-'0';
      dig.push_back(cur);
    } trim();
```

```
  }
  inline bool operator<(const BigInt& a)const
    {return cmp_minus(a)<0;}
  inline bool operator<=(const BigInt& a)const
    {return cmp_minus(a)<=0;}
  inline bool operator==(const BigInt& a)const
    {return cmp_minus(a)==0;}
  inline bool operator!=(const BigInt& a)const
    {return cmp_minus(a)!=0;}
  inline bool operator>(const BigInt& a)const
    {return cmp_minus(a)>0;}
  inline bool operator>=(const BigInt& a)const
    {return cmp_minus(a)>=0;}
  BigInt operator-() const {
    BigInt ret = *this;
    ret.neg ^= 1; return ret;
  }
  BigInt operator+(const BigInt& a) const {
    if(neg) return -(-(*this)+(-a));
    if(a.neg) return (*this)-(-a);
    int n = max(a.len(), len());
    BigInt ret; ret.dig.resize(n);
    lld pro = 0;
    for(int i=0;i<n;i++) {
      ret.dig[i] = pro;
      if(i < a.len()) ret.dig[i] += a.dig[i];
      if(i < len()) ret.dig[i] += dig[i];
      pro = 0;
      if(ret.dig[i] >= BASE) pro = ret.dig[i]/BASE;
      ret.dig[i] -= BASE*pro;
    }
    if(pro != 0) ret.dig.push_back(pro);
    return ret;
  }
  BigInt operator-(const BigInt& a) const {
    if(neg) return -(-(*this) - (-a));
    if(a.neg) return (*this) + (-a);
    int diff = cmp_minus(a);
    if(diff < 0) return -(a - (*this));
    if(diff == 0) return 0;
    BigInt ret; ret.dig.resize(len(), 0);
    for(int i=0;i<len();i++) {
      ret.dig[i] += dig[i];
      if(i < a.len()) ret.dig[i] -= a.dig[i];
      if(ret.dig[i] < 0){
        ret.dig[i] += BASE;
        ret.dig[i+1]--;
      }
    }
    ret.trim(); return ret;
  }
  BigInt operator*(const BigInt& a) const {
    if(!len()||!a.len()) return 0;
    BigInt ret; ret.dig.resize(len()+a.len()+1);
    ret.neg = neg ^ a.neg;
    for(int i=0;i<len();i++)
      for(int j=0;j<a.len();j++){
        ret.dig[i+j] += dig[i] * a.dig[j];
        if(ret.dig[i+j] >= BASE) {
          lld x = ret.dig[i+j] / BASE;
          ret.dig[i+j+1] += x;
          ret.dig[i+j] -= x * BASE;
        }
      }
    ret.trim(); return ret;
  }
  BigInt operator/(const BigInt& a) const {
    assert(a.len());
    if(len() < a.len()) return 0;
    BigInt ret; ret.dig.resize(len()-a.len()+1);
    ret.neg = a.neg;
    for(int i=len()-a.len();i>=0;i--){
      lld l = 0, r = BASE;
      while(r-l > 1){
        lld mid = (l+r)>>1;
        ret.dig[i] = mid;
        if(ret*a<=(neg?-(*this):(*this))) l = mid;
        else r = mid;
      }
      ret.dig[i] = l;
    }
    ret.neg ^= neg; ret.trim();
    return ret;
  }
  BigInt operator%(const BigInt& a) const {
    return (*this) - (*this) / a * a;
  }
  friend BigInt abs(BigInt a) { a.neg = 0; return a; }
  friend void swap(BigInt& a, BigInt& b){
    swap(a.dig, b.dig); swap(a.neg, b.neg);
  }
  friend istream& operator>>(istream& ss, BigInt& a){
    string s; ss >> s; a = s; return ss;
  }
  friend ostream&operator<<(ostream&o, const BigInt&a){
    if(a.len() == 0) return o << '0';
    if(a.neg) o << '-';
    o << a.dig.back();
    for(int i=a.len()-2;i>=0;i--)
      o<<setw(LOG_BASE)<<setfill('0')<<a.dig[i];
    return o;
  }
  inline void print() const {
    if(len() == 0){putchar('0');return;}
    if(neg) putchar('-');
    printf("%" PRINTF_ARG, dig.back());
    for(int i=len()-2;i>=0;i--)
      printf("%0" LOG_BASE_STR PRINTF_ARG, dig[i]);
  }
  #undef PRINTF_ARG
  #undef LOG_BASE_STR
};
```

13.3 Binpower

```
long long binpow(long long a, long long b, long long m) {
    a %= m;
    long long res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a % m;
        a = a * a % m;
        b >>= 1;
    }
    return res;
}

vector<int> applyPermutation(vector<int> sequence, vector<int
    > permutation) {
    vector<int> newSequence(sequence.size());
    for(int i = 0; i < sequence.size(); i++) {
        newSequence[permutation[i]] = sequence[i];
    }
    return newSequence;
}

// O(nlogk) to apply permutation b times on sequence
vector<int> permute(vector<int> sequence, vector<int>
     permutation, long long b) {
    while (b > 0) {
        if (b & 1) {
            sequence = applyPermutation(sequence, permutation
                );
        }
        permutation = applyPermutation(permutation,
             permutation);
        b >>= 1;
    }
    return sequence;
}
```

13.4 Catalan_Number

```
const int MOD = 1000000009;
const int MAX = 1000000009;
int catalan[MAX];

void init(int n) {
    catalan[0] = catalan[1] = 1;
    for (int i=2; i<=n; i++) {
        catalan[i] = 0;
        for (int j=0; j < i; j++) {
            catalan[i] += (catalan[j] * catalan[i-j-1]) % MOD
                ;
            if (catalan[i] >= MOD) {
                catalan[i] -= MOD;
            }
        }
    }
}

//pass CSES Bracket Sequences I
//O(nlogMOD)
void init(ll n) {
    //use long long
  catalan[0] = 1;
  for(ll i=1;i<=n;i++)
    catalan[i] = catalan[i-1] * 2 * (2*i+1) % MOD * binpow(i
        +2, MOD-2, MOD) % MOD;
}

/*
C[0] = C[1] = 1.
C[n] = C[k]*C[n-1-k], k from 0 to n-1.  if n >= 2.

C[n] is 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 binary search trees that will be formed with N keys.
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). * full binary tree: nodes with either 2 or no children.
The number of monotonic lattice paths from point (0,0) to point (n,n) in a square lattice of size n*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...n using n rectangles (The ladder consists of n columns, where i-th column has a height ).
*/
```

13.5 Chinese_Remainder_Theorem

```cpp
//need ext_gcd
//pass 2022-NCPC-Pre-D
//find smallest n s.t n%m_i = x_i
ll chineseReminder(vector<ll> &m, vector<ll> &x) {
  ll total = 1, ans = 0;
  ll s = 0, t = 0;
  vector<ll> e;
  for(auto &i : m) total*=i;
  for(int i=0;i<(int)m.size();i++) {
    ext_gcd(m[i], total/m[i], s, t);
    e.emplace_back(t * (total / m[i]));
  }
  for(int i=0;i<(int)m.size();i++) (ans+= (e[i] * x[i] % total)) %= total;
  return (ans+total)%total;
}
```

13.6 FFT

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

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

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

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

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

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

// if doing on real number polynomial, just change int to double. And check real() >= eps ? real() : 0 at line 62 (generating result)
vec<ll> multiply(vec<ll> const& a, vec<ll> const& b) {
    vec<ll> fa(a.begin(), a.end()), fb(b.begin(), b.end());

    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] = fa[i] * fb[i]; // NTT: fa[i] * fb[i] % mod;
    fft(fa, true);

    vec<ll> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real()); // NTT: result[i] = fa[i];

    //* if multiplying two number
    /*
    int carry = 0;
    for (int i = 0; i < n; i++) {
        result[i] += carry;
        carry = result[i] / 10;
        result[i] %= 10;
    }
    */

    return result;
}

int main() {
    string sa,sb;
    cin >> sa >> sb;

    int na = sa.size(), nb = sb.size();
    vec<int> a(na, 0), b(nb, 0);  //* vector from LSB to MSB.

    for(int i = 0; i < na; i++) a[i] = sa[na - 1 - i] - '0';
    for(int i = 0; i < nb; i++) b[i] = sb[nb - 1 - i] - '0';

    vec<int> res = multiply(a, b);
    for(int i = res[na + nb - 1] ? na + nb - 1: na + nb - 2; i >= 0; i--)
       cout << res[i];
}

//! compute the coefficients modulo some prime number p.
/*
|   p = a * 2^b + 1   | a   | b   | prime factor |
|:-------------------:| --- | --- |:------------:|
|           3         | 1   | 1   |       2      |
|           5         | 1   | 2   |       2      |
|          17         | 1   | 4   |       3      |
|          97         | 3   | 5   |       5      |
|         193         | 3   | 6   |       5      |
|         257         | 1   | 8   |       3      |
|        7681         | 15  | 9   |      17      |
|       12289         | 3   | 12  |      11      |
|       40961         | 5   | 13  |       3      |
|       65537         | 1   | 16  |       3      |
|      786433         | 3   | 18  |      10      |
|     5767169         | 11  | 19  |       3      |
|     7340033         | 7   | 20  |       3      |
|    23068673         | 11  | 21  |       3      |
|   104857601         | 25  | 22  |       3      |
|   167772161         | 5   | 25  |       3      |
|   469762049         | 7   | 26  |       3      |
|   998244353         | 119 | 23  |       3      |
|  1004535809         | 479 | 21  |       3      |
|  2013265921         | 15  | 27  |       3      |
|  2281701377         | 17  | 27  |       3      |
|  3221225473         | 3   | 30  |       5      |
|  75161927681        | 35  | 31  |       3      |
|  77309411329        | 9   | 33  |       7      |
| 206158430209        | 3   | 36  |      22      |
| 2061584302081       | 15  | 37  |       7      |
| 2748779069441       | 5   | 39  |       3      |
| 6597069766657       | 3   | 41  |       5      |
| 39582418599937      | 9   | 42  |       5      |
| 79164837199873      | 9   | 43  |       5      |
| 263882790666241     | 15  | 44  |       7      |
| 1231453023109121    | 35  | 45  |       3      |
| 1337006139375617    | 19  | 46  |       3      |
| 3799912185593857    | 27  | 47  |       5      |
| 4222124650659841    | 15  | 48  |      19      |
| 7881299347898369    | 7   | 50  |       6      |
| 31525197391593473   | 7   | 52  |       3      |
| 180143985094819841  | 5   | 55  |       6      |
| 1945555039024054273 | 27  | 56  |       5      |
| 4179340454199820289 | 29  | 57  |       3      |

*/

ll inverse(ll a, ll m){ // returns a^-1 mod m, 0 if not found
    ll x, y;
    ll g = ext_gcd(a, m, x, y);

    if (g != 1) {
        return 0;
    }
    else {
        x = (x % m + m) % m;
        return x;
    }
```

```
}
// const_a, root_pw, prime_factor need to adjust by prime number
const ll mod = 7340033;
const ll const_a = 7;
const ll root_pw = 1 << 20;
const ll prime_factor = 3; // or call generator(mod).
const ll root = binpow(prime_factor, const_a);
const ll root_1 = inverse(root, mod);

void fft(vector<ll> & 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) {
        ll wlen = invert ? root_1 : root;
        for (int i = len; i < root_pw; i <<= 1)
            wlen = (1LL * wlen * wlen % mod);

        for (int i = 0; i < n; i += len) {
            ll w = 1;
            for (int j = 0; j < len / 2; j++) {
                ll u = a[i+j], v = (1LL * a[i+j+len/2] * w % mod);
                a[i+j] = u + v < mod ? u + v : u + v - mod;
                a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
                w = (1LL * w * wlen % mod);
            }
        }
    }

    if (invert) {
        ll n_1 = inverse(n, mod);
        for (ll & x : a)
            x = (1LL * x * n_1 % mod);
    }
}
```

13.7 Fib

```
// Cassini's identity : F_{n-1} F_{n+1} - F_n^2= (-1)^n
// The "addition" rule : F_{n+k} = F_k * F_{n+1} + F_{k-1} * F_n
// k = n, F_{2n} = F_n * (F_{n+1} + F_{n-1})
// F_{2k} = F_k * (2F_{k+1} - F_k)
// F_{2k+1} =  F_k^2 + F_{k+1}^2

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

13.8 formula

13.8.1 圖論

1. 對於平面圖，$F = E - V + C + 1$，C 是連通分量數
2. 對於平面圖，$E \leq 3V - 6$
3. 對於連通圖 G，最大獨立點集的大小設為 I(G)，最大匹配大小設為 M(G)，最小點覆蓋設為 Cv(G)，最小邊覆蓋設為 Ce(G)。對於任意連通圖：
   (a) $I(G) + Cv(G) = |V|$
   (b) $M(G) + Ce(G) = |V|$
4. 對於連通二分圖：
   (a) $I(G) = Cv(G)$
   (b) $M(G) = Ce(G)$

13.8.2 dinic 特殊圖複雜度

1. 單位流：$O\left(min\left(V^{3/2}, E^{1/2}\right)E\right)$
2. 二分圖：$O\left(V^{1/2}E\right)$

13.8.3 學長公式

1. $\sum_{d|n}\phi(n) = n$
2. Harmonic series $H_n = \ln(n) + \gamma + 1/(2n) - 1/(12n^2) + 1/(120n^4)$
3. $\gamma = 0.57721566490153286060651209008240243104215$
4. 格雷碼 $= n \oplus (n >> 1)$
5. $SG(A + B) = SG(A) \oplus SG(B)$
6. 旋轉矩陣 $M(\theta) = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix}$

13.8.4 基本數論

1. $\sum_{i=1}^{n}\sum_{j=1}^{n} lcm(i,j) = n\sum_{d|n} d \times \phi(d)$

13.8.5 排組公式

1. k 卡特蘭 $\frac{C_n^{kn}}{n(k-1)+1} \cdot C_m^n = \frac{n!}{m!(n-m)!}$
2. $H(n,m) \cong x_1 + x_2 \ldots + x_n = k, num = C_k^{n+k-1}$
3. Stirling number of 2^{nd},n 人分 k 組方法數目
   (a) $S(0,0) = S(n,n) = 1$
   (b) $S(n,0) = 0$
   (c) $S(n,k) = kS(n-1,k) + S(n-1,k-1)$
4. Bell number,n 人分任意多組方法數目
   (a) $B_0 = 1$
   (b) $B_n = \sum_{i=0}^{n} S(n,i)$
   (c) $B_{n+1} = \sum_{k=0}^{n} C_k^n B_k$
   (d) $B_{p+n} \equiv B_n + B_{n+1} mod p$, p is prime
   (e) $B_{p^m+n} \equiv mB_n + B_{n+1} mod p$, p is prime
   (f) From B_0 : 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975
5. Derangement, 錯排, 沒有人在自己位置上
   (a) $D_n = n!(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} \ldots + (-1)^n \frac{1}{n!})$
   (b) $D_n = (n-1)(D_{n-1} + D_{n-2}), D_0 = 1, D_1 = 0$
   (c) From D_0 : 1, 0, 1, 2, 9, 44, 265, 1854, 14833, 133496
6. Binomial Equality
   (a) $\sum_k \binom{r}{m+k}\binom{s}{n-k} = \binom{r+s}{m+n}$
   (b) $\sum_k \binom{l}{m+k}\binom{s}{n+k} = \binom{l+s}{l-m+n}$
   (c) $\sum_k \binom{l}{m+k}\binom{s+k}{n}(-1)^k = (-1)^{l+m}\binom{s-m}{n-l}$
   (d) $\sum_{k\leq l}\binom{l-k}{m}\binom{s}{k-n}(-1)^k = (-1)^{l+m}\binom{s-m-1}{l-n-m}$
   (e) $\sum_{0\leq k\leq l}\binom{l-k}{m}\binom{q+k}{n} = \binom{l+q+1}{m+n+1}$
   (f) $\binom{r}{k} = (-1)^k\binom{k-r-1}{k}$
   (g) $\binom{r}{m}\binom{m}{k} = \binom{r}{k}\binom{r-k}{m-k}$
   (h) $\sum_{k\leq n}\binom{r+k}{k} = \binom{r+n+1}{n}$
   (i) $\sum_{0\leq k\leq n}\binom{k}{m} = \binom{n+1}{m+1}$
   (j) $\sum_{k\leq m}\binom{m+r}{k}x^k y^k = \sum_{k\leq m}\binom{-r}{k}(-x)^k(x+y)^{m-k}$

13.8.6 冪次, 冪次和

1. $a^b \% P = a^{b\%\varphi(p)+\varphi(p)}, b \geq \varphi(p)$
2. $1^3 + 2^3 + 3^3 + \ldots + n^3 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}$
3. $1^4 + 2^4 + 3^4 + \ldots + n^4 = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$
4. $1^5 + 2^5 + 3^5 + \ldots + n^5 = \frac{n^6}{6} + \frac{n^5}{2} + \frac{5n^4}{12} - \frac{n^2}{12}$
5. $0^k + 1^k + 2^k + \ldots + n^k = P(k), P(k) = \frac{(n+1)^{k+1} - \sum_{i=0}^{k-1} C_i^{k+1} P(i)}{k+1}, P(0) = n+1$
6. $\sum_{k=0}^{m-1} k^n = \frac{1}{n+1}\sum_{k=0}^{n} C_k^{n+1} B_k m^{n+1-k}$
7. $\sum_{j=0}^{m} C_j^{m+1} B_j = 0, B_0 = 1$
8. 除了 $B_1 = -1/2$，剩下的奇數項都是 0
9. $B_2 = 1/6, B_4 = -1/30, B_6 = 1/42, B_8 = -1/30, B_{10} = 5/66, B_{12} = -691/2730, B_{14} = 7/6, B_{16} = -3617/510, B_{18} = 43867/798, B_{20} = -174611/330,$

13.8.7 Burnside's lemma

1. $|X/G| = \frac{1}{|G|}\sum_{g\in G}|X^g|$
2. $X^g = t^{c(g)}$
3. G 表示有幾種轉法，X^g 表示在那種轉法下，有幾種是會保持對稱的，t 是顏色數，$c(g)$ 是循環節不動的面數。
4. 正立方體塗三顏色，轉 0 有 3^6 個元素不變，轉 90 有 6 種，每種有 3^3 不變，180 有 3×3^4，120(角) 有 8×3^2，180(邊) 有 6×3^3，全部 $\frac{1}{24}\left(3^6 + 6 \times 3^3 + 3 \times 3^4 + 8 \times 3^2 + 6 \times 3^3\right) = 57$

13.8.8 Count on a tree

1. Spanning Tree
   (a) 完全圖 $n^n - 2$
   (b) 一般圖 (Kirchhoff's theorem)$M[i][i] = degree(V_i)$,$M[i][j] = -1$,if have $E(i,j)$,0 if no edge. delete any one row and col in A, $ans = det(A)$

13.9 gcds

```cpp
// O(log(min(a, b)))
// returns gcd and one solution to a*x+b*y=gcd(a,b)
int ext_gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = ext_gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

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

// find one solution (x0, y0) s.t. a*x0+b*y0=c
// first by finding a sol for a*x+b*y=g.
// since c % g = 0, a*x*(c/g)+b*x*(c/g)=c.
bool find_any_solution(int a, int b, int c, int &x0, int &y0,
     int &g) {
    g = ext_gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false; // proof: a linear combination of a, b
            is divisible by gcd
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

void shift_solution(int & x, int & y, int a, int b, int cnt)
     {
    x += cnt * b;
    y -= cnt * a;
}

// # of sols(x,y) s.t. a*x+b*y = c between x,y range
int find_all_solutions(int a, int b, int c, int minx, int
     maxx, int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(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_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
/*
iterate all solutions
x = lx + k (b/g) for all k >= 0, until x = rx.

Theorems
1. The set of solution a*x+b*y=c is
    x = x0 + k*(b/g), y = x0 - k*(a/g).
2. smallest possible val
 x' + y' = x + y + k(b-a)g, minimize k*(b-a)
if a<b pick smallest k, a>b otherwise.
*/
```

13.10 Integer_factorization

```cpp
// need to build prime vector first.
vec<ll> primes;

vec<ll> trial_division4(ll n) {
    vec<ll> fac;

    for (ll d : primes) {
        if (d * d > n)
            break;
        while (n % d == 0) {
            fac.eb(d);
            n /= d;
        }
    }
    if (n > 1)
        fac.eb(n);

    return fac;
}

// MOON
vector<int> primes;
vector<int> LPFs(n + 1, 1);
for (int i = 2; i < n; ++i) {
  if (LPFs[i] == 1) {
    LPFs[i] = i;
    primes.emplace_back(i);
  }
  for (auto p : primes) {
    if (1LL * i * p > n) break;
    LPFs[i * p] = i;
    if (i % p == 0) break;
  }
}

void print_prime_factor(int x) {
  while (x != 1) {
    int factor = SPFs[x], cnt = 0;
    for (; x % factor == 0; ++cnt)
      x /= factor;
    cout << factor << ": " << cnt << endl;
  }
}
```

13.11 low_bit

```cpp
int lowbit(int x) { return x & (~x + 1); }
// O(3^n) // bitmask dp
for (int i = 0; i < n; i++) {
    for (int res = i; res; res = (res - 1) & i) { // all
         subset
    }
    // or this way
    int b = 0;
    do{

    }while((b = (b - i) & i));
}

int x = 5328; // 00000000000000000001010011010000
cout << __builtin_clz(x) << "\n"; // # of 0s at the beginning
     of the number = 19
cout << __builtin_ctz(x) << "\n"; // # of 0s at the end of
     the number = 4
cout << __builtin_popcount(x) << "\n"; // # of 1s in the
     number = 5
cout << __builtin_parity(x) << "\n"; // the parity(even or
     odd) of # of 1s = 1
```

13.12 nCr

```cpp
const int MAX = 3000005;
const ll mod = 998244353;

ll fact[MAX], tcaf[MAX]; // tcaf[a] = fact[a]^-1 mod n
```

```

ll binpow(ll x, ll d, ll m = mod) {
    if (d < 0) d += mod - 1;

    ll y = 1;
    do{
        if (d & 1) (y *= x) %= mod;
        (x *= x) %= mod;
    } while (d >>= 1);

    return y;
}

// Call this first.
void init(int n) {
    fact[0] = 1;
    for (int i = 1; i <= n; i++)
        fact[i] = i * fact[i - 1] % mod;
    for (int i = n; i >= 0; --i)
        tcaf[i] = binpow(fact[i], -1);
}

// Invoke nCr via this.
ll nCr(int n, int r) {
    if (r < 0 || r > n) return 0;
    return fact[n] * tcaf[r] % mod * tcaf[n - r] % mod;
}
```

13.13 phi

```
// phi(n):= # of number in [1,n] s.t. co-prime to n.
/*
Theorems:
1. phi(p) = p-1 if p is a prime
2. phi(p^k) = p^k-p^{k-1} if p is a prime
3. phi(a*b) = phi(a)*phi(b) if gcd(a,b)=1.
*/
// O(sqrt(n))
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}

// by phi(n) = n*(1-1/p1)*(1-1/p2)*..
// O(nloglogn)
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}

// Gauss phi's property: sum{phi(d) , for all d|n} = n.
// O(nlogn)
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    phi[0] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i++)
        phi[i] = i - 1;

    for (int i = 2; i <= n; i++)
        for (int j = 2 * i; j <= n; j += i)
            phi[j] -= phi[i];
}
```

13.14 PollardRho

```
//find a factor in O(n^1/4)
//n need to be Composite number
//sometimes it may fail, just keep running.
//Floyd version, wait for Brent version
ll mul(ull a, ull b, ull m) {//need unsigned long long to avoid overflow
  ll ans = 0;
  while(b) {
    if(b&1) {
      ans+=a;
      if(ans>=m) ans-=m;
    }
    a<<=1, b>>=1;
    if(a>=m) a-=m;
  }
  return ans;
}

mt19937 mt(time(nullptr));
ll f(ll x, ll& c, ll& pmod) {
  return (mul(x,x,pmod)+c%pmod)%pmod;
}

ll pollard(ll x) {
  if(x == 4) return 2;
  ll c = mt()%x;
  ll a=2, b=2;

  while(1) {
    a = f(a, c, x);
    b = f(f(b, c, x), c, x);
    ll d = __gcd(x, abs(a-b));
    if(a==b) return -1;//in cycle
    if(d!=1) return d;//find
  }
}
```

13.15 Primal_tests

```
// O(sqrt(n))
bool isPrime(int x) {
    for (int d = 2; d * d <= x; d++) {
        if (x % d == 0)
            return false;
    }
    return true;
}

// rely on Fermat's little theorem
// : a^(p-1) = 1(mod p) if p is a prime and gcd(a,p) = 1.
/*
Carmichael Number : if a^(n-1)=1(mod n) for every a prime to n.
There exist only 646 Carmichael Number <= 10^9.
*/
bool probablyPrimeFermat(int n, int iter=5) {
    if (n < 4)
        return n == 2 || n == 3;

    for (int i = 0; i < iter; i++) {
        int a = 2 + rand() % (n - 3);
        if (binpower(a, n - 1, n) != 1)
            return false;
    }
    return true;
}
//Miller Rabin for long long range
ull mul(ull a, ull b, ull m) {
  ull ans = 0;
  while(b>0) {
    if(b&1) {
      ans+=a;
      if(ans>=m) ans-=m;
    }
    a<<=1, b>>=1;
    if(a>=m) a-=m;
  }
  return ans;
}

ull fpow(ull a, ull n, ull m) {
  if(n == 0) return 1;
  if(n%2 == 0) return fpow(mul(a, a, m), n/2, m);
  return mul(a, fpow(mul(a, a, m), n/2, m), m);
}

bool MillerRabin(ll n) {
  if(n == 2) return true;
  if(n<2 || n%2 == 0) return false;

  ll u = n-1, t = 0;
  while(u%2 == 0) u>>=1, t++;

  for(ll a : {2,3,5,7,11,13,17,19,23,29,31,37}) {
    if(n == a) return true;
    ll x = fpow(a, u, n);
    if(x == 1 || x == n-1) continue;

    for(int i=0;i<t;i++) {
      x = mul(x, x, n);
      if(x == 1) return false;
      if(x == n-1) break;
    }
    if(x == n-1) continue;
    return false;
```

```
  }
  return true;
}

//* --------- copy from cp-algorithm.

using u64 = uint64_t;
using u128 = __uint128_t;

u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128)result * base % mod;
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_composite(u64 n, u64 a, u64 d, int s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
};

bool MillerRabin(u64 n) { // returns true if n is prime, else
      returns false.
    if (n < 2)
        return false;

    int r = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }

    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
         37}) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```

13.16 primitive_root

```
ll generator (ll p) {
    vector<ll> fact;
    ll phi = p-1,  n = phi;
    for (ll i=2; i*i<=n; ++i)
        if (n % i == 0) {
            fact.push_back (i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
        fact.push_back (n);

    for (ll res=2; res<=p; ++res) {
        bool ok = true;
        for (size_t i=0; i<fact.size() && ok; ++i)
            ok &= binpow (res, phi / fact[i], p) != 1;
        if (ok)  return res;
    }
    return -1;
}
```

13.17 Sieve_of_Eratosthenes

```
void SieveErato(){
    int n; // becase of ll
    vec<bool> is_prime(n+1, true);
    is_prime[0] = is_prime[1] = false;
    for (int i = 2; i * i <= n; i++) {
        if (is_prime[i]) {
            for (int j = i * i; j <= n; j += i)
                is_prime[j] = false;
        }
    }
}

// O((R-L+1)loglog(R) + sqrt(R)loglog(sqrt(R)))
vec<char> segmentedSieve(ll L, ll R) {
    // generate all primes up to sqrt(R)
    ll lim = sqrt(R);
    vec<char> mark(lim + 1, false);
    vec<ll> primes;
    for (ll i = 2; i <= lim; ++i) {
        if (!mark[i]) {
            primes.eb(i);
            for (ll j = i * i; j <= lim; j += i)
                mark[j] = true;
        }
    }

    vec<char> isPrime(R - L + 1, true);
    for (ll i : primes)
        for (ll j = max(i * i, (L + i - 1) / i * i); j <= R;
             j += i)
            isPrime[j - L] = false;
    if (L == 1)
        isPrime[0] = false;
    return isPrime;
}

// O((R-L+1)log(R) + sqrt(R))
vec<char> segmentedSieveNoPreGen(ll L, ll R) {
    vec<char> isPrime(R - L + 1, true);
    ll lim = sqrt(R);
    for (ll i = 2; i <= lim; ++i)
        for (ll j = max(i * i, (L + i - 1) / i * i); j <= R;
             j += i)
            isPrime[j - L] = false;
    if (L == 1)
        isPrime[0] = false;
    return isPrime;
}
```

14 python

14.1 python

```
#!/usr/bin/env python3

# import
import math
from math import *
import math as M
from math import sqrt

# input
n = int(input())
a = [ int(x) for x in input().split() ]

# EOF
while True:
    try:
        solve();
    except:
        break;

# output
print(x, sep=' ');
print(''.join(str(x)+' ' for x in a))
print('{:5d}'.format(x))

# sort
a.sort()
sorted(a)

# list
a = [ x for x in range(n) ]
a.append(x);

# basic operators
a, b = 10, 20

a / b # 0.5
a // b # 0
a % b # 10
a ** b # 10^20

# if, else if, else
if a == 0:
    print('zero')
elif a > 0:
    print("positive")
else:
    print("negative")

# loop
while a == b and b == c:
for i in LIST:

# stack
stack = [3, 4, 5]
```

```python
stack.append(6) # push
stack.pop() # pop
stack[-1] # top
len(stack) # size

# queue
from collections import deque
queue = deque([3, 4, 5])
queue.append(6) # push
queue.popleft() # pop
queue[0] # top
len(queue) # size

# random
from random import *
randrange(L, R, step) # [L,R) L+k*step
randint(L, R) # int from [L, R]
choice(list) # pick 1 item from list
choices(list, k) # pick k item
shuffle(list) # shuffle
uniform(L, R) # float from [L, R]

# decimal
from fractions import Fraction
from decimal import Decimal, getcontext
getcontext().prec = 250 # set precision

itwo = Decimal(0.5)
two = Decimal(2)

N = 200
def angle(cosT):
    """given cos(theta) in decimal return theta"""
    for i in range(N):
        cosT = ((cosT + 1) / two) ** itwo
    sinT = (1 - cosT * cosT) ** itwo
    return sinT * (2 ** N)

pi = angle(Decimal(-1))

Decimal('1.115').quantize(Decimal('.00'), ROUND_HALF_UP) # input should be str() -> '1.115'
Decimal('1.5').quantize(Decimal('0'), ROUND_HALF_UP)

# file IO
r = open("filename.in")
a = r.read() # read whole content into one string

w = open("filename.out", "w")
w.write('123\n') # write

# IO redirection
import sys
sys.stdin = open('filename.in')
sys.stdout = open('filename.out', 'w')
```

15 string

15.1 KMP

```cpp
// T for Text, P for Pattern
vec<int> KMP(const string& P) {
    vec<int> f(P.size(), -1);
    int len = f[0] = -1;
    for (int i = 1; i < P.size(); ++i) {
        while (len != -1 && P[len + 1] != P[i])
            len = f[len];
        if (P[len + 1] == P[i])
            ++len;
        f[i] = len;
    }
    return f;
}

// find S in T
vec<int> KMP_match(vec<int> fail, const string& S, const string& T) {
    vec<int> res; // start from these points
    int n = S.size();

    int i = -1;
    for (int j = 0; j < T.size(); ++j) {
        while (i != -1 && T[j] != S[i + 1])
            i = fail[i];

        if (T[j] == S[i + 1]) i++;
        if (i == n - 1)
            res.eb(j - n + 1), i = fail[i];
    }

    return res;
}

// Counting the number of occurrences of each prefix
void prefix_occur(){
    vector<int> ans(n + 1);
    for (int i = 0; i < n; i++)
        ans[pi[i]]++;
    for (int i = n-1; i > 0; i--)
        ans[pi[i-1]] += ans[i];
    for (int i = 0; i <= n; i++)
        ans[i]++;
}

// we set pi[0] = 0, and if (i+1) % ((i+1) - prefix[i]) == 0,
// the minimum circular string length will be (i+1) - prefix[i],
// otherwise it will be (i+1) (no circular).
// ex. abcabcabcabcabc = abc*5.
```

15.2 Minimal_Rotation

```cpp
string Minimal_Rotation(string &s) {
  int n = (int)s.size();
  int i=0, j=1, k=0;
  while(k<n && i<n && j<n) {
    if(s[(i+k)%n] == s[(j+k)%n]) k++;
    else {
      s[(i+k)%n] > s[(j+k)%n] ? i = i+k+1 : j = j+k+1;
      if(i == j) i++;
      k = 0;
    }
  }
  i = min(i,j);
  return s.substr(i) + s.substr(0,i);
}
```

15.3 Rabin_Fingerprint

```cpp
#define MAX 1000000
#define prime_mod 1073676287
ll h[MAX]; // 1-index, stores hashing of str[1...i]
ll h_base[MAX];
char str[MAX];
void hash_init(int len, ll prime = 0xdefaced)
{
    h_base[0] = 1, h[0] = 0;
    for (int i = 1; i <= len; i++){
        h[i] = (h[i - 1] * prime + str[i - 1]) % prime_mod;
        h_base[i] = (h_base[i - 1] * prime) % prime_mod;
    }
}

ll get_hash(int l, int r){
    return ((h[r+1] - h[l] * h_base[r - l + 1] % prime_mod) + prime_mod) % prime_mod;
}
```

15.4 Suffix_Array

```cpp
//O(nlgn) to build suffix array
//wait for O(n) version
vector<int> SA(string s) {
  s = s + "$";
  int n = (int)s.size();
  const int alphabet = 256;
  vector<int> c(n), p(n), cnt(max(n, alphabet));
  fill(cnt.begin(), cnt.end(), 0);

  for(int i=0;i<n;i++) cnt[s[i]]++;
  for(int i=1;i<alphabet;i++) cnt[i]+=cnt[i-1];
  for(int i=n-1;i>=0;i--) p[--cnt[s[i]]] = i;

  c[p[0]] = 0;
  int classes = 1;
  for(int i=1;i<n;i++) {
    if(s[p[i]] != s[p[i-1]]) classes++;
    c[p[i]] = classes-1;
  }

  vector<int> pn(n), cn(n);
  for(int h=1;h<n;h<<=1) {
    for(int i=0;i<n;i++) {
      pn[i] = p[i] - h;
      if(pn[i] < 0) pn[i]+=n;
    }
    fill(cnt.begin(), cnt.begin() + classes, 0);

    for(int i=0;i<n;i++) cnt[c[pn[i]]]++;
    for(int i=1;i<classes;i++) cnt[i]+=cnt[i-1];
    for(int i=n-1;i>=0;i--) p[--cnt[c[pn[i]]]] = pn[i];

    cn[p[0]] = 0;
    classes = 1;
```

```
    for(int i=1;i<n;i++) {
      pii cur = pii(c[p[i]], c[(p[i] + h) % n]);
      pii prev = pii(c[p[i-1]], c[(p[i-1] + h) %n]);
      if(cur != prev) classes++;
      cn[p[i]] = classes - 1;
    }
    c.swap(cn);
  }
  return p;
}

//O(n) to build lcp array
vector<int> LCP(vector<int> &sa, string &s) {
  int n = (int)sa.size();

  vector<int> order(n);
  for(int i=0;i<n;i++)
    order[sa[i]] = i;
  vector<int> lcp(n - 1);
  int k = 0;
  for(int i=0;i<n;i++) {
    if(order[i] == n-1){
      k = 0;
      continue;
    }

    int j = sa[order[i] + 1];
    while(i+k<n-1 && j+k<n-1 && s[i+k] == s[j+k])
      k++;
    lcp[order[i]] = k;
    if(k) k--;
  }
  return lcp;
}
```

15.5 Trie

```
//fat data structure
struct node {
  int hit = 0;
  node* next[26] = {};
};

using pnode = node* ;

void insert(const char *s, pnode* root) {
  if(!*root)
    *root = new node;
  if(s[0] == '\0') {
    (*root)->hit++;
  } else {
    insert(s+1, &(*root)->next[*s-'a']);
  }
}
```

15.6 Z

```
vector<int> Z(string &s) {
  vector<int> z((int)s.size());
  fill(z.begin(), z.end(), 0);
  int L, R;
  L = R = 0;
  for(int i=1;i<(int)s.size();i++) {
    if(i <= R && i + z[i-L] - 1 < R) {
      z[i] = z[i-L];
    }  else {
      z[i] = max(0, R-i+1);
      while(s[z[i]] == s[i + z[i]])
        z[i]++;
      L = i;
      R = i + z[i] - 1;
    }
  }
  return z;
}
```

ACM ICPC Team

Reference - Duracell!

Contents