**Techniques:**

1. For counting problems, try counting number of incorrect ways instead of correct ways.

3. Utilize Symmetries

4. Try solving the problem backwards

5.Binary Search the answer

6. Meet in the middle (Solve left half, Solve right half, combine)

7. Greedy

8. DP

9. Analyse complexity carefully

10. **Reduce the problem to some standard problem**

11. Add m when doing modular arithmetic.

12. Carefully analyse reasoning behind adding small details in the Q

13. Use exponential search in case of unbounded search.

14. dfs/bfs/msbfs/cyclic/dsu/bipartite/color/topo/mst/ssp/scc/fcc

/euler/bridge

15. Digit: digit dp, <12:bitmasking, l-r: sweepline, grid: dijkstra/dp

/dfs, graph:add=remove, repeatedsubt: gcd, max: ternary

/2^(10^5):check contribution of numbers/bit, prefix: trie, nge,

n=10^9: sqrt(n)/logn, a+b=a^b+2a&b, x+y=(x|y)+(x&y),

a^-1%m=a^m-2 % m only works when gcd(a,m)=1,

Monotonic: bs, quick jump on tree: binary lift, optimize linear

Rec: expo, subarray: prefixsum/dp ending at i.

A+B = 2\*(A&B) +A^B

**Template c++:**

// JAI SHREE RAM

#include <bits/stdc++.h>

#define ll             long long

#define int            ll

#define uint           unsigned long long

#define dbg(var)       cout<<#var<<"="<<var<<" "

#define all(v)         v.begin(),v.end()

#define srt(v)         sort(v.begin(),v.end())

#define mxe(v)         \*max\_element(v.begin(),v.end())

#define mne(v)         \*min\_element(v.begin(),v.end())

#define ld             long double

#define vi             vector <int>

#define pii            pair <int, int>

#define vpi            vector <pii>

#define vpp            vector <pair<int, pii>>

using namespace std;

const int MOD = 1e9 + 7;

ll gcd(ll a, ll b) {if (b > a) {return gcd(b, a);} if (b == 0) {return a;} return gcd(b, a % b);}

vector<ll> sieve(int n) {int\*arr = new int[n + 1](); vector<ll> vect; for (int i = 2; i <= n; i++)if (arr[i] == 0) {vect.push\_back(i); for (int j = 2 \* i; j <= n; j += i)arr[j] = 1;} return vect;}

void vin( vector<int> &v , int n ){for (int i = 0; i < n; i++){int x ;cin >> x; v.push\_back(x);}}

void vout(vector<int> &v){for (int i = 0; i < v.size(); i++){cout << v[i] << " " ;}cout << endl;}

long long lcm(int a, int b){    return (a / gcd(a, b)) \* b;}

int digits\_count(int n){int d=0;while(n != 0){d++;n /=10;}return d;}

void modadd(int &a , int b) {a=((a%MOD)+(b%MOD))%MOD;}

void modsub(int &a , int b) {a=((a%MOD)-(b%MOD)+MOD)%MOD;}

void modmul(int &a , int b) {a=((a%MOD)\*(b%MOD))%MOD;}

//==================================== compute higher powers with mod ===================================

uint power(int x, int y, int p =  MOD)

{

    unsigned long long res = 1;

    x = x % p;

    while (y > 0)

    {

        if (y & 1)

            res = (res \* x) % p;

        y = y >> 1;

        x = (x \* x) % p;

    }

    return res;

}

// =============================================================================================================

uint modInverse(int n, int p=MOD)       // using fermats little thm. [p needs to be prime which is mostly the case as mod value generally is 1e9+7]

{

    return power(n, p - 2, p);

}

// can also derive this using extended euclidean... however this has a much simpler code....

vector<int> fac(200005,0);

// =========================================Used to calculate nCr of higher values ===================================

uint nCr(int n, int r, int p=MOD)     // faster calculation..

{

    if (n < r)

        return 0;

    if (r == 0)

        return 1;

    return (fac[n] \* modInverse(fac[r], p) % p \* modInverse(fac[n - r], p) % p) % p;

}

void hello()

{

    int n=1,m=0;

    string s;

    cin>>n;

    vi v;

    vin(v,n);    }

int32\_t main()

{

    int t=1;

    cin>>t;

    while(t--)

    {

        hello();

    }

}

**Number Theory:**

1. To calculate sum of factors of a number, we can find the number of prime factors and their exponents. N = ae1 \* be2 \* ce3 …

Then sum = (1 + a + a^2….)(1 + b + b^2 .. )...

Number of factors=(a+1)\*(b+1)...

2.Every **even** integer greater than 2 can be expressed as the sum of 2 primes.

3. For root n prime method, check for 2, 3 then:

for (i=5; i\*i<=n; i=i+6) n%i and n%(i+2)

4. Number of divisors will be prime only if N=p^x where p is prime.

5. Kth prime factor= store smallest factor in seive and repeatedly divide with it to get the answer.

6. fib(n+m)=fib(n)fib(m+1)+fib(n-1)fib(m)

7. A **number is Fibonacci** if and only if one or both of (5\*n2 + 4) or (5\*n2 – 4) is a perfect square

8. every positive Every positive integer can be **written uniquely** as a sum of distinct non-neighbouring Fibonacci numbers.

10. **Root n under mod p exists only if**

n^((p-1)/2) % p = 1

11.divisibility by 4: last 2 digits divisible by 4

12.divisibility by 8: last 3 digits divisible by 8

13. Divisibility by 3,9: sum of digits divisible by 3,9

14. Divisibility by 11: alternate (+ve,-ve) digit sum is divisible by 11

15. Divisibility by 12: divisible by 3 and 4

16. Divisibility by 13: alternating sum in blocks of 3 (L to R) div 13

17. Integral solution of ax+by=c exists if gcd(a,b) divides c

**Probability:**

P(A∩B) = P(A) + P(B) - P(A∪B)

Probability of A if B has happened:

P(A|B) = P(A∩B) / P(B)

expected value is the sum of: [(each of the possible outcomes) × (the probability of the outcome occurring)].

Var(X) = E(X^2) – m^2

**Extended Euclid’s Algorithm:**

LL gcde(LL a,LL b,LL \*x,LL \*y)

{

if (a == 0)

{

\*x = 0, \*y = 1;

return b;

}

LL x1, y1;

LL gcd = gcde(b%a, a, &x1, &y1);

\*x = y1 - (b/a) \* x1;

\*y = x1;

return gcd;

}

**Modular power**

LL Mpow(LL x, unsigned LL y, LL m)

{

LL res = 1;

x = x % m;

while (y > 0)

{

if (y & 1)

res = (res\*x) % m;

y = y>>1; // y = y/2

x = (x\*x) % m; }

Return res;}

**Euler’s totient:**

Number of integers coprime to n less than n

LL phi(LL n)

{

LL result = n;

for (LL p=2; p\*p<=n; ++p)

{

if (n % p == 0)

{

while (n % p == 0)

n /= p;

result -= result / p;

}

}

if (n > 1)

result -= result / n;

return result;

}

**Largest power of p that divides n!**

// Returns largest power of p that divides n!

int largestPower(int n, int p)

{

// Initialize result

int x = 0;

// Calculate x = n/p + n/(p^2) + n/(p^3) + ....

while (n)

{

n /= p;

x += n;

}

return x;

}

**nCr (with lucas Theorem):**

LL ncrp(LL n, LL r, LL p)

{

LL C[r+1];

memset(C, 0, sizeof(C));

C[0] = 1;

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

{

for ( LL j = min(i, r); j > 0; j--)

C[j] = (C[j] + C[j-1])%p;

}

return C[r];

}

LL ncrpl(LL n,LL r, LL p)

{

if (r==0)

return 1;

int ni = n%p, ri = r%p;

return (ncrpl(n/p, r/p, p) \*

ncrp(ni, ri, p)) % p;

}

**Group Overlapping Ranges**

int groupOverlappingRanges(vector<vector<int>>& ranges) {

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

vector<vector<int>> temp;

temp.push\_back({ranges[0]});

for(int i=1;i<ranges.size();i++){

int last\_fir = temp[temp.size()-1][0];

int last\_val = temp[temp.size()-1][1];

if(ranges[i][0] > last\_val){

temp.push\_back(ranges[i]);

}else{

temp.pop\_back();

int final\_last = max(last\_val, ranges[i][1]);

temp.push\_back({last\_fir, final\_last});

}

} return temp.size();

}

**Number of solutions to a linear eqn:**

LL countSol(LL coeff[], LL start, LL end, LL rhs)

{

// Base case

if (rhs == 0)

return 1;

LL result = 0; // Initialize count of solutions

// One by subtract all smaller or equal coefficiants and recur

for (LL i=start; i<=end; i++)

if (coeff[i] <= rhs)

result += countSol(coeff, i, end, rhs-coeff[i]);

return result;

}

**Ternary Search (max of unimodal function):**

double ts(double start, double end)

{

double l = start, r = end;

for(int i=0; i<200; i++) {

double l1 = (l\*2+r)/3;

double l2 = (l+2\*r)/3;

//cout<<l1<<" "<<l2<<endl;

if(func(l1) > func(l2)) r = l2; else l = l1;

}

return func(r);

}

**Primarily Test**

**Time:sqrt(N);**

bool isPrime(int n)

{

if (n <= 1)

return false;

for (int i = 2; i <= sqrt(n); i++)

if (n % i == 0)

return false;

return true;

}

**Sieve (Time :(NLogLogN) ) (1e6)**

**Prime factorization**

**Time:sqrt(N)**

void primeFactors(int n) {

while (n % 2 == 0) {

cout << 2 << " ";

n = n/2;

}

for (int i = 3; i <= sqrt(n); i = i + 2) {

while (n % i == 0) {

cout << i << " ";

n = n/i;

}

}

if (n > 2)

cout << n << " ";

}

**Time:LogN(Precomputte spf)**

define MAXN 100001

int spf[MAXN];

void sieve()

{

spf[1] = 1;

for (int i=2; i<MAXN; i++)

spf[i] = i;

for (int i=4; i<MAXN; i+=2)

spf[i] = 2;

for (int i=3; i\*i<MAXN; i++) {

if (spf[i] == i) {

for (int j=i\*i; j<MAXN; j+=i)

if (spf[j]==j)

spf[j] = i;

}

}

}

vector<int> getFactorization(int x)

{

vector<int> ret;

while (x != 1) {

ret.push\_back(spf[x]);

x = x / spf[x];

}

return ret;

}

**Binary Exponention**

**Time:LogN**

ll power(ll x, ll y)

{

ll res = 1;

while (y)

{

if (y % 2 == 1)

res = (res \* x) % MOD;

y = y >> 1;

x = (x \* x) % MOD;

}

return res;

}

**Euclid Algorithm(Find GCD)**

**Time:LogN**

int gcd(int a, int b)

{

return b == 0 ? a : gcd(b, a % b);

}

**Note:gcd(a,b)=gcd(a-b,b)=gcd(a,b-a)**

**gcd(a,b,c,d)=gcd(a+k,b+k,c+k,d+k)=**

**gcd(0,b-a,c-a,d-a)**

**gcd(a,b)<=min(a,b)**

**gcd\*lcm=a\*b**

**gcd(n^(a-1),n^(b-1))=n^gcd(a,b)-1**

**Modulo Arithmetics**

(A+B)%P=(A%P+B%P)%P

This works for addition,Sub,Mul But does not work for division so we have to find modulo multiplicative inverse;

(A/B)%P=(A%P \* X\*P)%P where X is Modulo Mul Inverse

**X=A^(-1)=pow(A,P-2)%P; where GCD(A,P)=1**

**# of Divisor from Prime factorization**

N=(y1+1)\*(y2+1)\*(y3+1)+...........

Where y1,y2,y3,.... Are the power of distinct prime number in factotization

**Segmented sieve:**

void simpleSieve(int limit, vector<int> &prime){

vector<bool> mark(limit + 1, true);

for (int p=2; p\*p<limit; p++){

if (mark[p] == true){

for (int i=p\*p; i<limit; i+=p)

mark[i] = false;

}

}

for (int p=2; p<limit; p++){

if (mark[p] == true){

prime.push\_back(p);

cout << p << " ";

}

}

}

void segmentedSieve(int n){

int limit = floor(sqrt(n))+1;

vector<int> prime;

prime.reserve(limit);

simpleSieve(limit, prime);

int low = limit;

int high = 2\*limit;

while (low < n){

if (high >= n)

high = n;

bool mark[limit+1];

memset(mark, true, sizeof(mark));

for (int i = 0; i < prime.size(); i++){

int loLim = floor(low/prime[i]) \* prime[i];

if (loLim < low)

loLim += prime[i];

for (int j=loLim; j<high; j+=prime[i])

mark[j-low] = false;

}

for (int i = low; i<high; i++)

if (mark[i - low] == true)

cout << i << " ";

low = low + limit;

high = high + limit;

}

}

**Nth catalan number:**

unsigned long int catalan(unsigned int n){

if (n <= 1)

return 1;

unsigned long int res = 0;

for (int i = 0; i < n; i++)

res += catalan(i) \* catalan(n - i - 1);

return res;

}

**—-------------------------------------------------------------------------**

**Game Theory:**

1. If nim-sum is non-zero, player starting first wins.

2. Mex: smallest non-negative number not present in a set.

3. Grundy=0 means game lost.

4. Grundy=mex of all possible next states.

5. Sprague-Grundy theorem:

If a game consists of sub games (nim with multiple piles)

Calculate grundy number of each sub game (each pile)

Take xor of all grundy numbers:

If non-zero, player starting first wins.

**Segment tree:**

class SGTree {

vector<int> seg;

public:

SGTree(int n) {

seg.resize(4 \* n + 1);

}

void build(int ind, int low, int high, int arr[]) {

if (low == high) {

seg[ind] = arr[low];

return;

}

int mid = (low + high) / 2;

build(2 \* ind + 1, low, mid, arr);

build(2 \* ind + 2, mid + 1, high, arr);

seg[ind] = min(seg[2 \* ind + 1], seg[2 \* ind + 2]);

}

int query(int ind, int low, int high, int l, int r) {

if (r < low || high < l) return INT\_MAX;

if (low >= l && high <= r) return seg[ind];

int mid = (low + high) >> 1;

int left = query(2 \* ind + 1, low, mid, l, r);

int right = query(2 \* ind + 2, mid + 1, high, l, r);

return min(left, right);

}

void update(int ind, int low, int high, int i, int val) {

if (low == high) {

seg[ind] = val;

return;

}

int mid = (low + high) >> 1;

if (i <= mid) update(2 \* ind + 1, low, mid, i, val);

else update(2 \* ind + 2, mid + 1, high, i, val);

seg[ind] = min(seg[2 \* ind + 1], seg[2 \* ind + 2]);

}

};

**Lazy Propagation:**

class ST{

vector<int> st, lazy;

public:

ST(int n){

st.resize(4 \* n);

lazy.resize(4 \* n);

}

public:

void build(int ind, int low, int high, int arr[]){

if(low == high){

st[ind] = arr[low];

return;

}

int mid = (low + high) >> 1 ;

build(2 \* ind + 1, low, mid, arr);

build(2 \* ind + 2, mid + 1 , high, arr);

st[ind] = st[2\* ind + 1] + st[2\* ind + 2];

}

public:

void update( int ind, int low, int high, int l, int r, int val){

if(lazy[ind] != 0){

st[ind] += (high - low + 1) \* lazy[ind];

if(low != high){

lazy[2 \* ind + 1] = lazy[ind];

lazy[2 \* ind + 2] = lazy[ind];

}

lazy[ind] = 0;

}

if(high < l || r < low){

return;

}

if(low >= l && high <= r) {

st[ind] += (high - low + 1) \* val;

if(low != high){

lazy[2 \* ind + 1] += val;

lazy[2 \* ind + 2] += val;

}

return;

}

int mid = ( low + high) >> 1;

update(2 \* ind + 1, low, mid, l, r, val);

update(2 \* ind + 2, mid + 1, high, l , r, val);

st[ind] = st[2\* ind + 1] + st[2\* ind + 2];

}

public:

int query(int ind, int low, int high, int l, int r){

if(lazy[ind] != 0){

st[ind] += (high - low + 1) \* lazy[ind];

if(low != high){

lazy[2 \* ind + 1] = lazy[ind];

lazy[2 \* ind + 2] = lazy[ind];

}

lazy[ind] = 0;

}

if(high < l || r < low){

return 0;

}

if(low >= l && high <=r){

return st[ind];

}

int mid = ( low + high) >> 1;

int left = query(2 \* ind + 1, low, mid, l, r);

int right = query(2 \* ind + 2, mid + 1, high, l , r);

return left + right;

}

};

//DSU

class DisjointSet

{ vector<int> rank, parent, size, mx, mn;

public:DisjointSet(int n){

rank.resize(n + 1, 0);parent.resize(n + 1);

size.resize(n + 1); mx.resize(n + 1);

mn.resize(n + 1); for (int i = 0; i <= n; i++){

parent[i] = i;size[i] = 1;

mn[i] = i; mx[i] = i;}

} int findUPar(int node){

if (node == parent[node]) return node;

return parent[node] = findUPar(parent[node]);

} int findmin(int node) {

return mn[findUPar(node)];

} int findmax(int node){

return mx[findUPar(node)];

} int findsize(int node) { return size[findUPar(node)];

} void unionByRank(int u, int v)

{

int ulp\_u = findUPar(u);

int ulp\_v = findUPar(v);

if (ulp\_u == ulp\_v)

return;

if (rank[ulp\_u] < rank[ulp\_v]) {

parent[ulp\_u] = ulp\_v;

}

else if (rank[ulp\_v] < rank[ulp\_u]){

parent[ulp\_v] = ulp\_u;

} else { parent[ulp\_v] = ulp\_u;

rank[ulp\_u]++;}

}

void unionBySize(int u, int v) {

int ulp\_u = findUPar(u);

int ulp\_v = findUPar(v);

if (ulp\_u == ulp\_v)

return;

if (size[ulp\_u] < size[ulp\_v]) {

parent[ulp\_u] = ulp\_v;

size[ulp\_v] += size[ulp\_u];

mn[ulp\_v] = min(mn[ulp\_v], mn[ulp\_u]);

mx[ulp\_v] = max(mx[ulp\_v], mx[ulp\_u]);

} else {

parent[ulp\_v] = ulp\_u; size[ulp\_u] += size[ulp\_v];

mn[ulp\_u] = min(mn[ulp\_v], mn[ulp\_u]);

mx[ulp\_u] = max(mx[ulp\_v], mx[ulp\_u]);

}

}

};

**—-------------------------------------------------------------------------**

**Shortest Path**

**Dijkstra's Algorithm**

//Time:ElogE (vertices start from 0) (Input 2 vector)

struct PQ {

public:

int key,value; };

auto comp = [](PQ a, PQ b) {

if (a.value < b.value) return false;

else if (a.value > b.value) return true;

else { // when value are same

if (a.key < b.key) return false;

else return true;

} };

class Solution

{

public:

vector <int> dijkstra(int V, vector<vector<int>> adj[], int S) {

priority\_queue<PQ, vector<PQ>, decltype(comp)> pq(comp); // (value,weight)

vector<bool>vis(V+1,false);

pq.push({S,0});

vector<ll>ans(V,MOD); // Check by putting infinity

ans[S]=0;

while(!pq.empty())

{

auto tp=pq.top();

if(vis[tp.key])

{ pq.pop(); continue; }

vis[tp.key]=true;

pq.pop();

for(auto x:adj[tp.key])

{

if(!vis[x[0]])

{

if(ans[x[0]]>x[1]+ans[tp.key])

{

pq.push({x[0],x[1]+ans[tp.key]}); // Concentrate here we pushed edge wait + shortest distance of it's parent from source;

ans[x[0]]=x[1]+ans[tp.key];

} } } }

return ans; } };

**Bellman-Ford Algorithm(directed graph)**

**//**Time:V\*E (-ve weights )(0 to n-1 edge)(Input 1 vector edge(x,y,z))

class Solution {

public:

int isNegativeWeightCycle(int n, vector<vector<int>>edges)

{

vector<ll>count(n,INT\_MAX);

count[0]=0;

for(ll i=0;i<n-1;i++)

{

for(auto x:edges)

{

if(count[x[0]]!=INT\_MAX && count[x[1]]>count[x[0]]+x[2])

count[x[1]]=count[x[0]]+x[2];

}

}

for(auto x:edges)

{

if(count[x[0]]!=INT\_MAX && count[x[1]]>count[x[0]]+x[2])

return 1;

}

return 0;

}

};

**Floyd Warshall Algorithm(+ve number)**

Time:(N^3) (Shortest path between all the vertex)

while (t--)

{

ll V, E, Q;

cin >> V >> E >> Q;

vector<vector<ll>> adj[V];

ll dis[V][V];

for (ll i = 0; i < V; i++){

for (ll j = 0; j < V; j++) {

if (i == j)

dis[i][j] = 0;

else

dis[i][j] =INF;

}

}

ll i = 0;

while (i++ < E)

{

ll u, v, w;

cin >> u >> v >> w;

u--;

v--;

ll j=dis[u][v];

dis[u][v] = min(w,j);

j=dis[v][u];

dis[v][u] = min(w,j);

vector<ll> t1, t2;

t1.push\_back(v);

t1.push\_back(w);

adj[u].push\_back(t1);

t2.push\_back(u);

t2.push\_back(w);

adj[v].push\_back(t2);

}

for (ll k = 0; k < V; k++){

for (ll i = 0; i < V; i++){

for (ll j = 0; j < V; j++) {

if (dis[i][j] > (dis[i][k] + dis[k][j])

&& (dis[k][j] != INF && dis[i][k] != INF))

dis[i][j] = dis[i][k] + dis[k][j];

}

}

}

**Minimum Spanning Tree**

**Prim's Algorithm** (Input 2 vector)(0-indexed)

Time:(ElogE) (Insert the edges and takeout the minimum edge everytime till the priority queue is not empty)

struct PQ{

public:

int key;int value;

};

auto comp = [](PQ a, PQ b)

{

if (a.value < b.value) return false;

else if (a.value > b.value) return true;

else {

if (a.key < b.key) return false;

else return true; }

};

class Solution

{

public:

//Function to find sum of weights of edges of the Minimum Spanning Tree.

int spanningTree(int V, vector<vector<int>> adj[])

{

priority\_queue<PQ, vector<PQ>, decltype(comp)> pq(comp); // (value,weight)

//vector<ll>parent(V,-1); If we want to store parent or resultant Spanning tree

vector<bool>vis(V+1,false);

pq.push({0,0});

//parent[0]=-1;

ll ans=0;

while(!pq.empty())

{

auto tp=pq.top();

if(vis[tp.key])

{

pq.pop(); continue;

}

if(!vis[tp.key])

ans+=tp.value;

vis[tp.key]=true;

pq.pop();

for(auto x:adj[tp.key])

{

if(!vis[x[0]])

{

pq.push({x[0],x[1]});

//parent[x[0]]=tp.key;

} } }

return ans;

}

};

**Kruskal's Algorithm and DSU (1-indexed)**

Time(:ELogE) Space :(E+V) ;(Input 2 vector)

struct DSU {

vector<int> par, rnk, sz;

int c;

DSU(int n) : par(n + 1), rnk(n + 1, 0), sz(n + 1, 1), c(n) {

for (int i = 1; i <= n; ++i) par[i] = i;

}

int find(int i) {

return (par[i] == i ? i : (par[i] = find(par[i])));

}

bool same(int i, int j) {

return find(i) == find(j);

}

int get\_size(int i) {

return sz[find(i)];

}

int count() {

return c; //connected components

}

int merge(int i, int j) {

if ((i = find(i)) == (j = find(j))) return -1;

else --c;

if (rnk[i] > rnk[j]) swap(i, j);

par[i] = j;

sz[j] += sz[i];

if (rnk[i] == rnk[j]) rnk[j]++;

return j;

}

};

class Solution

{

public:

//Function to find sum of weights of edges of the Minimum Spanning Tree.

int spanningTree(int V, vector<vector<int>> adj[])

{

DSU d(V);

map<ll,vector<pair<ll,ll>>>mp;

for(ll i=1;i<=V;i++)

{

for(auto x:adj[i])

mp[x[1]].push\_back({i,x[0]});

}

ll c=0;

ll ans=0;

for(auto x:mp){

for(auto y:x.second){

if(d.find(y.first)!=d.find(y.second)){

d.merge(y.first,y.second);

ans+=x.first;

c++;

if(c==V-1)

return ans;

} } } } };

**Bipartite Graph**

Time:(V+E) ( Used to divide the set into two sets using coloring)

Input(Just like dfs )

vector<ll>vis(1e5+1,false);

vector<ll>col(1e5+1,-1);

vector<int> adj[100001];

ll f=0;

void dfs(ll vertex,ll color)

{

vis[vertex]=true;

col[vertex]=color;

for(ll child:adj[vertex])

{

if(!vis[child])

{

dfs(child,1-color);

}

else

{

if(col[child]==color) // ODD CYCLE

f=1;

}

}

}

**Topological sort and cycle in directed graph**

Time(V+E) Space(V) Input(Simple DFS directed)

vector<ll>vis(1e5+1,false);

vector<ll>parent(1e5+1,-1);

vector<vector<ll>>adj;

ll f=0;

// ll pos=-1; Used In Printing Cycle

// ll rep=-1;

ll V;

stack<ll>st;

class Solution {

public:

//map<ll,vector<ll>>v;

map<ll,bool>vis; // can use array also

bool isCyclicUtil(int vertex, bool visited[],bool \*recStack) {

if(visited[vertex] == false) {

visited[vertex] = true;

recStack[vertex] = true;

list<int>::iterator i;

for(auto child:adj[vertex]) {

if ( !visited[child] && isCyclicUtil(child, visited, recStack) )

return true;

else if (recStack[child])

return true;

} }

recStack[vertex] = false;

return false;

}

bool isCyclic()

{

bool \*visited = new bool[V];

bool \*recStack = new bool[V];

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

visited[i] = false;

recStack[i] = false;

}

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

if ( !visited[i] && isCyclicUtil(i, visited, recStack))

return true;

return false;

}

// TOPOSORT DFS

void dfs(ll vertex)

{

vis[vertex]=true;

for(ll child:adj[vertex]){

if(!vis[child]){

dfs(child);

st.push(child);

} }

}

void findOrder(ll n, vector<vector<ll>>& pre) {

V=n;

vector<int>ans;

if(isCyclic()) // For checking Directed Acyclic graph;

{ cout<<"IMPOSSIBLE";

}

else{

for(ll i=1;i<=n;i++) {

if(!vis[i]) {

dfs(i); st.push(i); // Pushing into stack

}

} } };

**Strongly Connected Component:**

**Kosaraju’s algo**

**Time:(Simple DFS)**

**->Used to find number of strongly Connected Component**

**->Can also be used to find members in each Component**

->(1 indexed)

stack<ll>st;

// This dfs traversal is to push into the stack the element just like topological sort

void dfs(ll vertex,vector<ll>adj[],vector<bool>&vis1) {

vis1[vertex]=true;

for(ll child:adj[vertex]) {

if(!vis1[child]){

dfs(child,adj,vis1);

st.push(child);

}}}

// This dfs is for traversal of transpose graph(SCC remains same even after transpose of graph)

void dfs1(ll vertex,vector<ll>trans[],vector<bool>&vis2) {

vis2[vertex]=true;

for(ll child:trans[vertex]){

if(!vis2[child]){

dfs1(child,trans,vis2);

}}}

void solve()

{

ll V,E;

cin>>V>>E;

vector<ll>adj[V+1],trans[V+1];

vector<bool>vis2(V+1,false),vis1(V+1,false);

while(E--){

ll x,y;

cin>>x>>y;

adj[x].pb(y);

trans[y].pb(x); }

dfs(1,adj,vis1);

st.push(1);

ll scc=0;

while(!st.empty()){

ll x=st.top();

st.pop();

if(!vis2[x]){

dfs1(x,trans,vis2);

scc++;

}}

cout<<scc<<"\n";

}

**Articulation Point(cut-vertex)**

#include<bits/stdc++.h>

using namespace std;

#define V 5

#define pb push\_back

unordered\_map<int,vector<int>> adj;

void DFS(int u,vector<int>& disc,vector<int>& low,vector<int>& parent,vector<bool>& articulation\_Point)

{

static int time = 0;

disc[u] = low[u] = time;

time+=1;

int children = 0;

for(int v: adj[u])

{

if(disc[v]==-1) //If v is not visited

{

children += 1;

parent[v] = u;

DFS(v,disc,low,parent,articulation\_Point);

low[u] = min(low[u],low[v]);

if(parent[u]==-1 and children>1) //Case-1: U is root

articulation\_Point[u] = true;

if(parent[u]!=-1 and low[v]>=disc[u]) //Case-2: Atleast 1 component will get separated

articulation\_Point[u] = true;

}

else if(v!=parent[u]) //Ignore child to parent edge

low[u] = min(low[u],disc[v]);

}

}

void findAPs\_Tarjan()

{

vector<int> disc(V,-1),low(V,-1),parent(V,-1);

vector<bool> articulation\_Point(V,false); //Avoids cross-edge

for(int i=0;i<V;++i)

if(disc[i]==-1)

DFS(i,disc,low,parent,articulation\_Point);

cout<<"Articulation Points are: ";

for(int i=0;i<V;++i)

if(articulation\_Point[i]==true)

cout<<i<<" ";

}

**Bridges Point(cut edge)**

**Time:(V+E)**

#include<bits/stdc++.h>

using namespace std;

#define V 5

#define pb push\_back

unordered\_map<int,vector<int>> adj;

void DFS(int u,vector<int>& disc,vector<int>& low,vector<int>& parent,vector<pair<int,int>>& bridge)

{

static int time = 0;

disc[u] = low[u] = time;

time+=1;

for(int v: adj[u])

{

if(disc[v]==-1) //If v is not visited

{

parent[v] = u;

DFS(v,disc,low,parent,bridge);

low[u] = min(low[u],low[v]);

if(low[v] > disc[u])

bridge.pb({u,v});

}

else if(v!=parent[u]) //Ignore child to parent edge

low[u] = min(low[u],disc[v]);

}

}

void findBridges\_Tarjan()

{

vector<int> disc(V,-1),low(V,-1),parent(V,-1);

vector<pair<int,int>> bridge;

for(int i=0;i<V;++i)

if(disc[i]==-1)

DFS(i,disc,low,parent,bridge);

cout<<"Bridges are: \n";

for(auto it: bridge)

cout<<it.first<<"-->"<<it.second<<"\n";

}

**—-------------------------------------------------------------------------**

**// Z ALGO -> O(string length)**

long[] Z;

//Code for Z algo

void Zalgo(String s){

int n = s.length();

Z = new long[n];

int l = 0, r = 0;

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

if (i > r){

l = r = i;

while (r<n && s.charAt(r-l) == s.charAt(r)){

r++;

}

Z[i] = r-l;

r--;

}

else{

int k = i-l;

if (Z[k] < r-i+1){

Z[i] = Z[k];

}

else{

l = i;

while (r < n && s.charAt(r-l) == s.charAt(r)){

r++;

}

Z[i] = r-l;

r--;

}}}}

**// MANACHER ALGO O(string length)**

// d1[i] - how many palindromes of odd length with center at i

public static int[] oddPalindromes(String s) {

int n = s.length();

int[] d1 = new int[n];

int l = 0, r = -1;

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

int k = (i > r ? 0 : Math.min(d1[l + r - i], r - i)) + 1;

while (i + k < n && i - k >= 0 && s.charAt(i + k) == s.charAt(i - k)) ++k;

d1[i] = k--;

if (i + k > r) {

l = i - k;

r = i + k;

}

}

return d1;

}

// d2[i] - how many palindromes of even length with center at i

public static int[] evenPalindromes(String s) {

int n = s.length();

int[] d2 = new int[n];

int l = 0, r = -1;

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

int k = (i > r ? 0 : Math.min(d2[l + r - i + 1], r - i + 1)) + 1;

while (i + k - 1 < n && i - k >= 0 && s.charAt(i + k - 1) == s.charAt(i - k)) ++k;

d2[i] = --k;

if (i + k - 1 > r) {

l = i - k;

r = i + k - 1;

}

}

return d2;

}

**// KMP O(string length)**

public static int[] prefixFunction(String s) {

int[] p = new int[s.length()];

int k = 0;

for (int i = 1; i < s.length(); i++) {

while (k > 0 && s.charAt(k) != s.charAt(i)) k = p[k - 1];

if (s.charAt(k) == s.charAt(i))

++k;

p[i] = k;

}

return p;

}

public static int findSubstring(String haystack, String needle) {

int m = needle.length();

if (m == 0)

return 0;

int[] p = prefixFunction(needle);

for (int i = 0, k = 0; i < haystack.length(); i++) {

while (k > 0 && needle.charAt(k) != haystack.charAt(i)) k = p[k - 1];

if (needle.charAt(k) == haystack.charAt(i))

++k;

if (k == m)

return i + 1 - m;

}

return -1;

}

public static int minPeriod(String s) {

int n = s.length();

int[] p = prefixFunction(s);

int maxBorder = p[n - 1];

int minPeriod = n - maxBorder;

// check periodicity

// if (minPeriod < n && n % minPeriod != 0) return -1;

return minPeriod;

}

**/\*\*\* Uses rolling hash to quickly determine substring equality.\*/**

**Some additional pairs to consider:**

**- P = 2122331213, K = 104717**

**- P = 2124749677, K = 104711**

public static class SubHashMulti {

private static final int[] P = { 2131131137, 2147483647 };

private static final int[] K = { 104723, 104729 };

private static final int HASHES = P.length;

private static final int MAX\_LEN = 2\_000\_002;

private static int UPTO = 1;

private static long[][] POW = new long[HASHES][MAX\_LEN];

private static long[][] INV = new long[HASHES][MAX\_LEN];

static {

for (int j = 0; j < HASHES; ++j) {

POW[j][0] = 1;

POW[j][1] = K[j];

INV[j][0] = 1;

INV[j][1] = modInverse(K[j], P[j]);

}

}

private static void loadPows(int upper) {

for (int j = 0; j < HASHES; ++j) {

for (int i = UPTO + 1; i <= upper; ++i) {

POW[j][i] = POW[j][i - 1] \* POW[j][1] % P[j];

INV[j][i] = INV[j][i - 1] \* INV[j][1] % P[j];

}

}

UPTO = Math.max(UPTO, upper);

}

private final long[] S;

private final long[][] H;

public SubHashMulti(long[] x) {

loadPows(x.length);

S = x;

H = new long[HASHES][S.length + 1];

for (int j = 0; j < HASHES; ++j) {

for (int i = 0; i < S.length; ++i) {

H[j][i + 1] = (H[j][i] + S[i] \* POW[j][i]) % P[j];

}

}

}

public SubHashMulti(int[] x) {

this(toLongArray(x));

}

public SubHashMulti(char[] x) {

this(toLongArray(x));

}

public SubHashMulti(String x) {

this(x.toCharArray());

}

public long[] sub(int loInclusive, int hiExclusive) {

long[] hash = new long[HASHES];

for (int j = 0; j < HASHES; ++j) {

hash[j] = (H[j][hiExclusive] + P[j] - H[j][loInclusive]) \* INV[j][loInclusive] % P[j];

}

return hash;

}

public int length() {

return S.length;

}

private static long[] toLongArray(char[] x) {

long[] arr = new long[x.length];

for (int i = 0; i < x.length; ++i) {

arr[i] = x[i];

}

return arr;

}

private static long[] toLongArray(int[] x) {

long[] arr = new long[x.length];

for (int i = 0; i < x.length; ++i) {

arr[i] = x[i];

}

return arr;

}

/\*\*

\* Computes the value of (b ^ e) % mod.

\*/

private static long modPow(long b, long e, long mod) {

long p = b;

long ans = 1;

while (e > 0) {

if ((e & 1) == 1) {

ans = ans \* p % mod;

}

p = p \* p % mod;

e >>= 1;

}

return ans;

}

public static long modInverse(long a, long mod) {

return modPow(a, mod - 2, mod);

}

}

\* USAGE

\* 1) SubHashMulti pattern = new SubHashMulti("AABA");

\* 2) SubHashMulti text = new SubHashMulti("AABAACAADAABAABA");

\* 3) long[] checkwith = pattern.sub(0, pattern.length() - 1);

\* 4) for (int i = 0; i <= text.length() - pattern.length(); i++) {

\* 5) long[] sub = text.sub(i, i + pattern.length() - 1);

\* 6) if (Arrays.equals(checkwith, sub)) {

\* w.println(i);

**Binary Trie**

class Node{

Node[] child;

Node(){

child = new Node[2];

}

}

class Trie{

Node root;

Trie(){

root= new Node();

}

}

Trie tr;

Node mroot;

long find\_best(Node cur,long xor){

long res= 0;

for(int i=31;i>=0;i--){

if(((xor>>>i)&1)==1){

if(cur.child[0]!=null){

res+=(1<<i);

cur = cur.child[0];

}else{

cur = cur.child[1];

}

}else{

if(cur.child[1]!=null){ res+=(1<<i);

cur = cur.child[1];

}else{

cur = cur.child[0];

}

}

}

return res;

}

void insert(Node cur,long addxor){

for(int i=31;i>=0;i--){

if(((addxor>>>i)&1)==1){

if(cur.child[1]==null) cur.child[1] = new Node();

cur = cur.child[1];

}else{

if(cur.child[0]==null) cur.child[0] = new Node();

cur = cur.child[0];

}

}

}

tr = new Trie();

mroot = tr.root;

**NOTE FOR TERNARY SEARCH+FLOOR BINARY SEARCH**

Ternary Search

-WE USE IT TO FIND MIN/MAX IN UNIMODAL FUNCTION

it is used for unimodal functions(which has only one maxima or minima) /\ V

Time complexity of ternary search is O(2\*log3n) which is slightly more than binary search

int m1 = l+(r-l/2),m2=r-(r-l/3)

if(ar[m1]==elem||ar[m2]==elem) break;

if(elem<ar[m1]) l=l, r= m1-1;

else if(elem>ar[m2]) r=r, l= m2+1;

else l=m1+1, r=m2-1;

say we have to find the min of a quadratic equation f(x) having a>0

if(f(m1)>f(m2)) l = m1;

else if(f(m2)>f(m1)) r = m2;

else if(f(m1)==f(m2)) l = m1; r= m2;

lekin range ko chota kab tak krenge .. more the precision more will be the accuracy

while(r-l>0.000001) 1e-6 or 1e-7 allowed (the range r and l will be given in question)(notice when

we have such constraint in while loop we dont do l=m1+1 instead just l=m1 etc..

but for l<=r we do l = m1+1 and not l = m1

**LCA USING BINARY LIFTING**

// **Pre-processing(nlogn)** ArrayList<Integer>[] adj;

int[][] par;

int[] dep;

void dfs(int src,int parent,int depth){

par[src][0] = parent;

dep[src] = depth;

// dp[src][0] = val[src];

for(int i=1;i<20;i++){ // calculating all possible jumps for a node

par[src][i] = par[par[src][i-1]][i-1];

// dp[src][i] = gcd(dp[src][i-1],dp[par[src][i-1]][i-1]);

}

for(int child:adj[src]){

if(child!=parent){

dfs(child, src, depth+1);

}

}

}

int lca(int u,int v){

if(dep[v]>dep[u]){

// u must lie below

int temp = v;

v = u;

u = temp;

}

// ans = 0;

int diff = dep[u]-dep[v];

for(int i=19;i>=0;i--){

if((diff&(1<<i))!=0){

// ans = gcd(ans,dp[u][i]);

u = par[u][i];

}

}

if(u==v) return u; // ans = gcd(ans,val[u]);

for(int i=19;i>=0;i--){

if(par[u][i]!=par[v][i]){

// ans = gcd(ans,dp[u][i])

u = par[u][i];

// ans = gcd(ans,dp[v][i])

v = par[v][i];

}

}

return par[u][0]; // ans = gcd(ans,val[u],val[v],val[par[u][0]]);

}

// gcd path aggregrate problem precomputation TC: O(nlognlogn)

// query TC: O(lognlogn)

par = new int[n+1][20]; // 20 coz 2^20>10^6 which is

// the tree size normally given in problems

dfs(1,0,0); // precomputation in nlogn where

// n is number of vertices

**Techniques**

-draw something/write stuff down/simulate a process

-don't implement something unless if ur fairly confident its correct

Random stuff to check when **WA**:

-if code is way too long/cancer then reassess

-switched N/M -int overflow -switched variables

-wrong MOD -hardcoded edge case incorrectly

-read the problem statement again

Random stuff to check when **TLE**:

-continue instead of break -condition in for/while loop bad

Random stuff to check when **RTE**:

-switched N/M -long to int/int overflow

-division by 0 -edge case for empty list/data structure/N=1

**DIGIT DP**

long dp(int idx,int lo,int hi,int rem){

if(idx==len){

if(rem==0) return 1;

else return 0;

}

if(memo[idx][lo][hi][rem]!=-1) return memo[idx][lo][hi][rem];

long ans = 0;

int loLim = 0;

int hiLim = 9;

if(hi==1){

hiLim = R.charAt(idx)-'0';

}

if(lo==1){

loLim = L.charAt(idx)-'0';

}

for(int dig=loLim;dig<=hiLim;dig++){

// if hi is zero(not bound) nhi will stay 0

// if hi is one(bound tight) nhi will remain 1 only if digit==hiLim

int nhi = hi;

if(dig!=hiLim) nhi = 0;

int nlo = lo;

if(dig!=loLim) nlo = 0;

int nrem = (rem+dig)%D;

ans = (ans + dp(idx+1,nlo,nhi,nrem))%MOD;

}

return memo[idx][lo][hi][rem] = ans;

}

String L,R;

int D;

long[][][][] memo;

long len;

void solve(){

L = scn.next();

R = scn.next();

len = R.length();

// L must have the same length as R

// coz trie works on strings of similar length only

long extra = R.length()-L.length();

StringBuilder sb = new StringBuilder(); while(extra-->0) sb.append('0');

L = sb.toString()+ L;

D = scn.nextInt();

memo = new long[10001][2][2][101];

for(int i=0;i<10001;i++){

Arrays.fill(memo[i][0][0], -1);

Arrays.fill(memo[i][0][1], -1);

Arrays.fill(memo[i][1][0], -1);

Arrays.fill(memo[i][1][1], -1);

}

long ans = dp(0,1,1,0); // lo and hi are tight bound initially

w.println(ans);

}

**2 POINTER:** increasing L( doing L++), R must reamin the same or increase.

**LONGEST INCREASING SUBSEQ**

Returns a longest increasing subsequence in the given array.

public int[] longestIncreasingSubsequence(int[] X) {

final int N = X.length;

int[] P = new int[N];

int[] M = new int[N + 1];

int L = 0;

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

int lo = 1, hi = L;

while (lo <= hi) {

int mid = (lo + hi + 1) / 2;

if (X[M[mid]] < X[i]) {

lo = mid + 1;

} else {

hi = mid - 1;

}

}

int newL = lo;

P[i] = M[newL - 1];

M[newL] = i;

if (newL > L) {

L = newL;

}

}

int[] S = new int[L];

int k = M[L];

for (int i = L - 1; i >= 0; i--) {

S[i] = X[k];

k = P[k];

}

return S;

}

**DP With Bitmasking**

- Time complexity of n\*n\*2^n works well for n = 20,21,22

**- Subset generation using bitmasks is O(2^n\*n) while using recursion its only O(2^n)**

**- n&(n-1) drops lsb(rightmost 1), n&(-n): lowest set bit number**

**- Transition optimization in dp is possible using prefix sums array.**

**- State optimization in dp is possible only if x+y+z+d = n, no need to store n, also n can be made global and d can be written as n-x-y-z , so we reduced from a 5D dp to a 3D dp.**

**- when n<=20, you can generate all subsets possible (2^n) and do some operations on them**

**- BIPARTITE GRAPH WILL NEVER HAVE ODD CYCLE**

**- SMALLEST number divisible by all of the numbers is LCM**

- **Bit Manipulation : O(32n) – split problem into bits and solve for each of the bits**

**- gcd(a,b) will lie in the factors of a+b!!**

**- Observation: adhoc, primefactor,gcd, lcm, factor,square/sqrt**

**SPARSE TABLE**

class SparseTable {

public int[] log;

public int[][] table;

public int N;

public int K;

public SparseTable(int N) {

this.N = N;

log = new int[N + 2];

K = Integer.numberOfTrailingZeros(Integer.highestOneBit(N));

table = new int[N][K + 1];

sparsywarsy();

}

private void sparsywarsy() {

log[1] = 0;

for (int i = 2; i <= N + 1; i++)

log[i] = log[i / 2] + 1;

}

public void lift(int[] arr) {

int n = arr.length;

for (int i = 0; i < n; i++)

table[i][0] = arr[i];

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

for (int i = 0; i + (1 << j) <= n; i++)

table[i][j] = Math.min(table[i][j - 1], table[i + (1 << (j - 1))][j - 1]);

}

public int query(int L, int R) {

// inclusive, 1 indexed

R--;

L--;

int mexico = log[R - L + 1];

return Math.min(table[L][mexico], table[R - (1 << mexico) + 1][mexico]);

}

}

/\*

\* USAGE - DS FOR RANGE QUERIES + NO UPDATES

\* 1) int[] arr = {7, 2, 3, 0, 5, 10, 3, 12, 18};

\* 2) SparseTable st = new SparseTable(arr.length);

\* 3) st.lift(arr); // precomputation O(NlogN)

\* 4) int ans = st.query(1, 4); // 1 indexed i.e returns min of {7,2,3,0}

\* w.println(ans);

\* O(1) per query for overlapping problems like min,max,gcd

\* O(logN) per query for range sum

\*/

**computes divisors of each number uptill n in O(nlogn) time**

**\* works well for n<=10^7**

publicstaticArrayList<Integer>[] findsDivisorsOfEachNumberUptil(int n){

ArrayList<Integer>[] ans = new ArrayList[n+1];

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

ans[i] = new ArrayList<>();

}

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

for(int j=i;j<=n;j+=i){

ans[j].add(i);

}

}

return ans;

}

**MATRIX EXPO**

// multiplies two matrices

public static long[][] multiply(long[][] left, long[][] right) {

long MOD = 1000000007L;

int N = left.length;

int M = right[0].length;

long[][] res = new long[N][M];

for (int a = 0; a < N; a++)

for (int b = 0; b < M; b++)

for (int c = 0; c < left[0].length; c++) {

res[a][b] += (left[a][c] \* right[c][b]) % MOD;

if (res[a][b] >= MOD)

res[a][b] -= MOD;

}

return res;

}

// matrix exponentiation

public static long[][] power(long[][] grid, long pow) {

long[][] res = new long[grid.length][grid[0].length];

for (int i = 0; i < res.length; i++)

res[i][i] = 1L;

long[][] curr = grid.clone();

while (pow > 0) {

if ((pow & 1L) == 1L)

res = multiply(curr, res);

pow >>= 1;

curr = multiply(curr, curr);

}

return res;

}

(ADJ MATRIX)^n gives the number of paths from i..j having length n

**COMBINATIONS**

\* Computes the modulo result of (n choose k) IN LINEAR TIME.\*/

private static final long[] F = new long[400001];

private static final long[] INV = new long[F.length];

private static final long[] FI = new long[F.length];

static {

INV[1] = 1;

for (int i = 2; i < INV.length; ++i) {

INV[i] = MOD - (MOD / i) \* INV[MOD % i] % MOD;

}

F[0] = FI[0] = 1;

for (int i = 1; i < F.length; ++i) {

F[i] = (i \* F[i - 1]) % MOD;

FI[i] = (INV[i] \* FI[i - 1]) % MOD;

}

}

public static long C(int n, int k) {

return F[n] \* FI[k] % MOD \* FI[n - k] % MOD;

}

**Recurrence Relation:(Find Nth Term )**

Time:LogN

[1 2 3 K]\*[2\*2]^n=[fn fn+1]

K:No of dependent Term

Example:

[F1 F2]\*[Magic matrix]^n=[Fn Fn+1]

(Use of matrix Expo)