\*\* AD HOC & NORMAL THINGS

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

**using namespace std;**

**typedef long long ll;**

**// policy Based ds**

**#include<ext/pb\_ds/assoc\_container.hpp>**

**//#include<ext/pb\_ds/tree\_policy.hpp>**

**using namespace \_\_gnu\_pbds;**

**typedef tree<long long , null\_type, greater\_equal<long long>, rb\_tree\_tag,tree\_order\_statistics\_node\_update> ordered\_set;**

**// cout<<\*os.find\_by\_order(val)<<endl; // k-th element it**

**// less\_equal = multiset, less = set, greater\_equal = multiset decreasing, greater = set decreaseing**

**// cout<<os.order\_of\_key(val)<<endl;  // strictly smaller or greater**

**#define order(s, x) s.order\_of\_key(x)**

**#define elemat(s ,x) s.find\_by\_order(x)**

**#define boost   ios\_base::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);**

**#define endl    "\n"**

**#define all(x)  x.begin(), x.end()**

**#define read    freopen("input.txt","r",stdin)**

**#define write   freopen("output.txt","w",stdout)**

**#define pb      push\_back**

**#define ff      first**

**#define ss      second**

**#define GCD(a, b) \_\_gcd(a, b)**

**#define PI 2.0124 \* acos(0.0)125**

**#define LCM(a, b) (a \* b) / GCD(a, b)126**

**#define mem(a, b) memset(a, b, sizeof(a))**

**#define popcountL \_\_builtin\_popcountll**

**#define popcount \_\_builtin\_popcount**

**inline bool checkBit(int N, int pos){return(bool)(N & (1 << pos));}**

**inline int setBit(int N, int pos) { return N = N | (1 << pos); }**

**inline int unsetBit(int N, int pos) {return N = (N & (~(1 << pos))); }**

**inline int toggleBit(int N, int pos) {return N = (N ^ (1 << pos)); }**

**const ll sz=2e7+123;**

**#define INF 1000000000000000007**

**const ll MOD=1000000007;**

**const ll base = 31;**

**#define stringLen 18446744073709551620**

**inline void normal(ll &a) { a %= MOD; (a < 0) && (a += MOD); }**

**inline ll modMul(ll a, ll b) { a %= MOD, b %= MOD; normal(a), normal(b); return (a\*b)%MOD; }**

**inline ll modAdd(ll a, ll b) { a %= MOD, b %= MOD; normal(a), normal(b); return (a+b)%MOD; }**

**inline ll modSub(ll a, ll b) { a %= MOD, b %= MOD; normal(a), normal(b); a -= b; normal(a); return a; }**

**inline ll modPow(ll b, ll p) { ll r = 1; while(p) { if(p&1) r = modMul(r, b); b = modMul(b, b); p >>= 1; } return r; }**

**inline ll modInverse(ll a) { return modPow(a, MOD-2); }**

**inline ll modDiv(ll a, ll b) { return modMul(a, modInverse(b)); }**

**//Graph direction array[8]**

**//dx[]={0,0,1,-1,-1,1,1,-1};**

**//dy[]={1,-1,0,0,-1,-1,1,1};**

**//Bishop direction array[8]**

**//dx[]={0,0,1,-1,-1,1,1,-1};**

**//dy[]={1,-1,0,0,-1,-1,1,1};**

**//Knight Direction array[8]**

**//dx[]={1,1,2,2,-1,-1,-2,-2};**

**//dy[]={2,-2,1,-1,2,-2,1,-1};**

\*\* NUMBER THEORY

**FAISAL AMIN ABIR**

**PRIME GENERATION**

**bitset<sz>is\_prime;**

**vi prime;**

**void primeGen(int n){**

**for(int i=3; i<=n; i+=2)is\_prime[i]=1;**

**int nn = sqrt(n)+1;**

**for(ll i=3; i<nn; i+=2){**

**if(is\_prime[i]==0)continue;**

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

**is\_prime[j]=0;}}**

**is\_prime[2]=1;**

**prime.pb(2);**

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

**if(is\_prime[i])prime.pb(i);}}**

**PRIME FACTORIZATION**

**vector<long long>factorization(long long n){//O(sqrt(n)/ln(sqrt(n)) + log2 n)**

**vector<long long>factors;**

**for(auto u:prime){**

**if(1LL\*u\*u > n) break;**

**if(n%u==0){//  factors.push\_back(u);//for generating unique factors keep this line here**

**while(n%(u)==0){**

**factors.push\_back(u);//for generating all factors keep this line here**

**n/=(u);}}}**

**if(n>1)factors.push\_back(n);**

**return factors;}**

**SEGMENTED SIEVE**

**vector<char> segmentedSieve(long long L,long long R){**

**// generate all primes up to sqrt(R)**

**long long lim = sqrt(R);**

**vector<char> mark(lim + 1, false);**

**vector<long long> primes;**

**for (long long i = 2; i <= lim; ++i) {**

**if (!mark[i]) {**

**primes.emplace\_back(i);**

**for (long long j=i\*i;j<=lim;j+=i)**

**mark[j]=true;}}**

**vector<char> isPrime(R - L + 1, true);**

**for (long long i:primes)**

**for (long long 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;}**

**SUM OF DIVS**

**long long SOD(long long n){**

**long long res=1;**

**for(auto u:prime){**

**if(1LL\*u\*u > n)break;**

**if(n%u==0){**

**long long sum=1;**

**long long power = 1;**

**while(n%u==0){**

**n/=u;**

**power \*= u;**

**sum += power;}**

**res \*= sum;}}**

**if(n>1)res\*=(1+n);**

**return res;}**

**SUM OF NUMBER OF DIVS**

**int SNOD(int n){**

**int sq = sqrt(n), res=0;**

**for(int i=1; i<=sq; i++){**

**res += (n/i) - i;}**

**res \*= 2;**

**res += sq;**

**return res;}**

**NUMBER OF DIVS**

**ll NOD(ll n){**

**ll ans=1;**

**if(n>1000000000000){**

**for(ll i=0;;++i){**

**if(prime[i]\*prime[i]\*prime[i]>n){**

**break;**

**}**

**if(n%prime[i]==0){**

**ll cnt = 0;**

**while (n%prime[i]== 0){**

**n /= prime[i];**

**cnt++;}**

**ans\*=(cnt+1);}}**

**if(isprime(n)){ans\*=2;}**

**else if(issquareprime(n)){ans\*=3;}**

**else if(n!=1){ans\*=4;}}**

**else{**

**ll limit=sqrt(n);**

**for (ll i=0;prime[i]<=limit;++i){**

**if(n%prime[i]== 0){**

**ll cnt = 0;**

**while (n% prime[i]== 0){**

**n /= prime[i];**

**cnt++;**

**}**

**ans\*=(cnt+1);**

**limit=sqrt(n);**

**}**

**}**

**if(n!=1)ans\*=2;}**

**return ans;**

**}**

**//for NOD count each prime factors+1 and multiply all of them 2,2,3,3,3 (3)\*(4)**

**GEOEMTRIC SUM**

**//\*\*Give a,n will give a^1+a^2+a^3+...+a^n\*\***

**const ll MOD=1e9+7;**

**ll GeoSum(ll a, ll n){**

**ll sz = 0;ll ret = 0;ll mul = 1;**

**int MSB = 63 - \_\_builtin\_clzll(n);**

**while(MSB >= 0){**

**ret = ret \* (1 + mul); mul = (mul \*mul) % MOD; sz <<= 1;**

**if( (n >> MSB) & 1) {**

**mul = (mul \*a) % MOD; ret += mul; sz++;}**

**ret %= MOD; MSB--;}**

**return ret;}**

**NCR USING RECURRENCE**

**\*\*****ncr using recurrence{nCr = (n-1)Cr + (n-1)C(r-1)}\*\***

**void fncr(){**

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

**for(int j = 0; j < N; ++j) {**

**if (j == 0 || j == i) ncr[i][j] = 1;**

**else ncr[i][j] = ncr[i-1][j-1] +ncr[i-1][j];**

**}}}**

**NCR%M USING MODULAR MULTIPLICATIVEINVERSE**

**void precal(){**

**fact[0]=invFact[0]=1;**

**for(int i=1; i<=N; i++){**

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

**invFact[N]=bigmod(fact[N],MOD-2);**

**for(int i=N-1; i>=1; i--){**

**invFact[i]=((invFact[i+1]%MOD)\*((i+1)%MOD))%MOD;}}**

**cout<<((fact[n]\*invFact[r])%MOD\*invFact[n-r])%MOD<<endl;**

**//\*\*ncr if it stays in long long\*\***

**ll n,r,ans=1;**

**cin>>n>>r;**

**for(int i=1; i<=r; i++){**

**ans=ans\*(n-i+1);**

**ans/=i;}**

**LUCAS THEOREM(N AND R IS VERY BIG BUT MOD IS SMALL)**

**precal(){factorial and inverse factorialmod}**

**ll Lucas(ll n,ll r){**

**if(r<0||r>n){**

**return 0;}**

**if(r==0||r==n){**

**return 1;}**

**if(n>=MOD){**

**return(Lucas(n/MOD,r/MOD)%MOD\*Lucas(n%MOD,r%MOD)%MOD)%MOD;}**

**return(((fact[n]\*invFact[r])%MOD)\*invFact[n-r])%MOD;}**

**EXTENDED GCD AND LINEAR DIOPHANTINE EQUATION**

**int gcd(int a, int b, int& x, int& y) {**

**if (b == 0) {**

**x = 1; y = 0;**

**return a;}**

**int x1, y1;**

**int d = gcd(b, a % b, x1, y1);**

**x = y1;**

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

**return d;}**

**LINEAR DIOPHANTINE**

**bool find\_any\_solution(int a, int b, int c, int &x0, int &y0, int &g) {**

**g = gcd(abs(a), abs(b), x0, y0);**

**if (c % g) {**

**return false;}**

**x0 \*= c / g;**

**y0 \*= c / g;**

**if (a < 0) x0 = -x0;**

**if (b < 0) y0 = -y0;**

**return true;}**

**void shift\_solution(int & x, int & y, int a, int b, int cnt) {**

**x += cnt \* b;**

**y -= cnt \* a;}**

**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;}**

**FAYSAL AHAMMED CHOWDHURY**

**LINEAR SIEVE in O(n) upto but memory takes 32 times more:**

**const int N = 10000000;**

**vector<int> spf(N + 1);**

**vector<int> primes;**

**void linear\_sieve(){ // O(n)**

**for (int i = 2; i <= N; ++i){**

**if (spf[i] == 0){**

**spf[i] = i;**

**primes.push\_back(i);}**

**for (int j = 0; i \* primes[j] <= N; ++j){**

**spf[i \* primes[j]] = primes[j];**

**if (primes[j] == spf[i])**

**{break;}}}}**

**SPF SIEVE**

**const int N = 1e6 + 9;**

**int spf[N];**

**void spf\_sieve() {**

**for (int i = 2; i < N; i++) {**

**spf[i] = i;}**

**for (int i = 2; i < N; i++) {**

**if (spf[i] == i) {**

**for (int j = i; j < N; j += i) {**

**spf[j] = min(spf[j], i);}}}}**

**SOD: (()) \* (()) \* …**

**DIVISIBILITY BY 11:**

**A white paper with black writing on it

Description automatically generated**

**Legendre's Formula:**

**(N! / p^x) max value of x (p must be prime)**

**int legendre(int n, int p){**

**int ex = 0;**

**while(n) {**

**ex += (n / p);**

**n /= p;}**

**return ex;}**

**Digit Count of a number:**

**log10(n) + 1 (log10 for 10 base number)**

**How many Trailing zeros of n? –**

**Max power of base which divides n.**

**Phi function:**

**A close-up of a math problem

Description automatically generated**

**\* for n > 2, phi(n) is always even**

**\* sum of all phi(d) - divisors of n, is n.**

**Euler Totient Function:**

**// if we need phi(x) multiple times, then memoize it**

**map<int, int> dp;**

**int phi(int n) {**

**if (dp.count(n)) return dp[n];**

**int ans = n, m = n;**

**for (int i = 2; i \* i <= m; i++) {**

**if (m % i == 0) {**

**while (m % i == 0) m /= i;**

**ans = ans / i \* (i - 1);}}**

**if (m > 1) ans = ans / m \* (m - 1);**

**return dp[n] = ans;}**

**Phi from 1 to n in O(nlog log n):**

**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;}}}**

**Logarithm:**

**log(ab) = log(a) + log(b)**

**log(a^x) = xlog(a)**

**First K digit of n^k:**

**int firstk(int n, int k) {**

**double a = k \* log10(n);**

**double b = a - floor(a);**

**double c = pow(10, b);**

**return floor(c \* 100);}**

**Series:**

**Arithmetic progression:**

**S(n): (n/2)\*(a+p) p is last element**

**Or, S(n) = (n/2) \* (2a + (n-1) d) d is common difference**

**Geometric Progression:**

**S(n) = (a \* (1 - r^n)) / (1-r) r < 1**

**S(n) = (a \* (r^n - 1)) / (r-1) r > 1**

**\* 1^2 + 2^2 +..+ n^2 = (n \* (n+1) \* (2n+1)) / 6**

**\* 1^3 + 2^3 + … + n^3 = (n^2 \* (n+1)^2) / 4**

**Problems:**

**\* Minimal natural number N, so that N! contains exactly Q zeroes on the trail.**

**- Binary search over N and find how many trailing zeros N has by Legendre’s formula.**

**\* Given a number N^M, find out the number of integer bases in which it has exactly T trailing zeroes:**

**int solve\_greater\_or\_equal(vector<int> e, int t) {**

**int ans = 1;**

**for(auto i: e) {**

**ans = 1LL \* ans \* (i / t + 1) % mod;**

**}**

**return ans;**

**}**

**int solve\_equal(vector<int> e, int t) {**

**return (solve\_greater\_or\_equal(e, t) - solve\_greater\_or\_equal(e, t + 1) + mod) % mod;**

**}**

**int main() {**

**ios\_base::sync\_with\_stdio(0);**

**cin.tie(0);**

**int n, m, t, cs = 0;**

**while(cin >> n >> m >> t and n != 0) {**

**vector<int> e;**

**for(int i = 2; i \* i <= n; i++) {**

**if(n % i == 0) {**

**int ex = 0;**

**while(n % i == 0) {**

**ex++;**

**n /= i;**

**}**

**e.push\_back(ex \* m);**

**}**

**}**

**if(n > 1) e.push\_back(m);**

**cout << "Case " << ++cs << ": " << solve\_equal(e, t) << '\n';**

**}**

**return 0;**

**}**

**\* n! (factorial n) has at least t trailing zeroes in b based number system. Given the value of n and t, what is the maximum possible value of b?**

**const int N = 1e5 + 9, mod = 10000019;**

**vector<bool> is\_prime(N, true);**

**vector<int> primes;**

**int main() {**

**ios\_base::sync\_with\_stdio(0);**

**cin.tie(0);**

**sieve();**

**int t, cs = 0; cin >> t;**

**while(t--) {**

**int n, zeroes; cin >> n >> zeroes;**

**vector<pair<int, int>> ans;**

**for(auto p: primes) {**

**if(p > n) break;**

**int pw = legendre(n, p);**

**if(pw / zeroes != 0) {**

**ans.push\_back({p, pw/ zeroes});**

**}**

**}**

**int max\_base = 1;**

**for(auto i: ans) {**

**max\_base = 1LL \* max\_base \* power(i.first, i.second) % mod;**

**}**

**cout << "Case " << ++cs << ": ";**

**cout << (ans.size() == 0 ? -1 : max\_base) << '\n';**

**}**

**return 0;**

**}**

**Combinatorics:**

**\* nCr, nPr using recurrence -**

**const int N = 2005, mod = 1e9 + 7;**

**int C[N][N], fact[N];**

**void prec() { // O(n^2)**

**// nCr = (n-1)C(r-1) + (n-1)Cr**

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

**C[i][0] = C[i][i] = 1;**

**for (int j = 1; j < N; j++) {**

**C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % mod;**

**}**

**}**

**fact[0] = 1;**

**for (int i = 1; i < N; i++) {**

**fact[i] = 1ll \* fact[i - 1] \* i % mod;**

**}**

**}**

**int nCr(int n, int r) {**

**if (n < r) return 0;**

**return C[n][r];**

**}**

**int nPr(int n, int r) {**

**if (n < r) return 0;**

**return 1ll \* C[n][r] \* fact[r] % mod;**

**}**

**Combinations (nCr, nPr):**

**const int N = 1e6 + 1, mod = 1000003;**

**int fact[N], ifact[N];**

**int inverse(int x) {**

**return power(x, mod - 2, mod);**

**}**

**void prec() {**

**fact[0] = 1;**

**for (int i = 1; i < N; i++) {**

**fact[i] = 1ll \* fact[i - 1] \* i % mod;**

**}**

**ifact[N - 1] = inverse(fact[N - 1]);**

**for (int i = N - 2; i >= 0; i--) {**

**ifact[i] = 1ll \* ifact[i + 1] \* (i + 1) % mod;**

**}**

**}**

**int nCr(int n, int r) {**

**return 1ll \* fact[n] \* ifact[r] % mod \* ifact[n - r] % mod;**

**}**

**int nPr(int n, int r) {**

**return 1ll \* fact[n] \* ifact[n - r] % mod;**

**}**

**Stars and Bars:**

**\* Find the number of  k-tuples of non-negative integers whose sum is n.**

**-   (n + k - 1)! / (n! \* (k-1)!) or C(n+k-1, n)**

**\* Find the number of  k-tuples of non-negative integers whose sum is <= n.**

**-   (n + k)! / (n! \* k!) or C(n+k, n)**

**\* Combinations with repetitions**

**-   C(n+k-1, k)**

**\* How many ways to go from (0,0) to (n,m)**

**-   C(n+m, n)**

**Pascals Triangle is equivalent to nCr**

**A diagram of a pyramid

Description automatically generated**

**Multinomial Coefficient (a1+a2+..+ak)^n**

**Powers are given of k numbers. coefficient?**

**int n, k; cin >> n >> k;**

**int ans = fact[n];**

**for (int i = 1; i <= k; i++) {**

**int x; cin >> x;**

**ans = 1ll \* ans \* ifact[x] % mod;**

**}**

**Binomial Theorem:**

**A mathematical equation with numbers and symbols

Description automatically generated**

**Properties of Pascal's Triangle:**

**i = 0 to n ∑C(n,i) = 2^n**

**i = 0 to n where i is even ∑C(n,i) = 2^(n-1)**

**i = 0 to n where i is odd ∑C(n,i) = 2^(n-1)**

**Built-in Functions:**

**set(), reset(), flip(), count(), test(), any(), none(), all(), to\_ullong()**

**\_\_builtin\_popcount(x), \_\_builtin\_clz(x), \_\_builtin\_ctz(x)**

**\* Power Tower:**

****

**const int N = 1e5 + 9;**

**using ll = long long;**

**map<ll, ll> mp;**

**ll phi(ll n) {**

**if (mp.count(n)) return mp[n];**

**ll ans = n, m = n;**

**for (ll i = 2; i \* i <= m; i++) {**

**if (m % i == 0) {**

**while (m % i == 0) m /= i;**

**ans = ans / i \* (i - 1);**

**}**

**}**

**if (m > 1) ans = ans / m \* (m - 1);**

**return mp[n] = ans;**

**}**

**inline ll MOD(ll x, ll m) {**

**if (x < m) return x;**

**return x % m + m;**

**}**

**ll power(ll n, ll k, ll mod) {**

**ll ans = MOD(1, mod);**

**while (k) {**

**if (k & 1) ans = MOD(ans \* n, mod);**

**n = MOD(n \* n, mod);**

**k >>= 1;**

**}**

**return ans;**

**}**

**int a[N];**

**// if x >= log2(m), then a^x = a^(MOD(x, phi(m))) % m**

**ll yo(ll l, ll r, ll m) {**

**if (l == r) return MOD(a[l], m);**

**if (m == 1) return 1;**

**return power(a[l], yo(l + 1, r, phi(m)), m);**

**}**

**int32\_t main() {**

**ios\_base::sync\_with\_stdio(0);**

**cin.tie(0);**

**int n, m; cin >> n >> m;**

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

**cin >> a[i];**

**}**

**int q; cin >> q;**

**while (q--) {**

**int l, r; cin >> l >> r;**

**cout << yo(l, r, m) % m << '\n';**

**}**

**return 0;**

**}**

**Expected value:**

**A white text with black text

Description automatically generated**

\*\* Data Structures

**Faysal Ahammed**

**Segment Tree:**

**const int N = 1e5 + 9;**

**int a[N];**

**struct ST {**

**int tree[4 \* N];**

**void build(int n, int b, int e) {**

**if(b == e) {**

**tree[n] = a[b]; // change here**

**return;**

**}**

**int mid = (b + e) >> 1, l = n << 1, r = l + 1;**

**build(l, b, mid);**

**build(r, mid + 1, e);**

**tree[n] = tree[l] + tree[r]; // change here**

**}**

**void upd(int n, int b, int e, int i, int x) {**

**if(b > i || e < i) return;**

**if(b == e && b == i) {**

**tree[n] = x; // change here**

**return;**

**}**

**int mid = (b + e) >> 1, l = n << 1, r = l + 1;**

**upd(l, b, mid, i, x);**

**upd(r, mid + 1, e, i, x);**

**tree[n] = tree[l] + tree[r]; // change here**

**}**

**int query(int n, int b, int e, int i, int j) {**

**if(b > j || e < i) return 0; // return appropriate value**

**if(b >= i && e <= j) return tree[n];**

**int mid = (b + e) >> 1, l = n << 1, r = l + 1;**

**int L = query(l, b, mid, i, j);**

**int R = query(r, mid + 1, e, i, j);**

**return (L + R); // change this**

**}**

**} st;**

**Segment Tree Lazy:**

**const int N = 1e5 + 9;**

**int a[N];**

**struct ST {**

**int tree[4 \* N], lazy[4 \* N];**

**ST() {**

**memset(tree, 0, sizeof(tree));**

**memset(lazy, 0, sizeof(lazy));**

**}**

**void push(int n, int b, int e) {**

**if(lazy[n] == 0) return;**

**tree[n] += lazy[n] \* (e - b + 1); // change here**

**if(b != e) {**

**int l = n << 1, r = l + 1;**

**lazy[l] += lazy[n]; // change here**

**lazy[r] += lazy[n]; // change here**

**}**

**lazy[n] = 0;**

**}**

**void build(int n, int  b, int e) {**

**lazy[n] = 0;**

**if(b == e) {**

**tree[n] = a[b]; // change here**

**return;**

**}**

**int mid = (b + e) >> 1, l = n << 1, r = l + 1;**

**build(l, b, mid);**

**build(r, mid + 1, e);**

**tree[n] = tree[l] + tree[r]; // change here**

**}**

**void upd(int n, int b, int e, int i, int j, int x) {**

**push(n, b, e);**

**if(b > j || e < i) return;**

**if(b >= i && e <= j) {**

**lazy[n] += x; // change here**

**push(n, b, e);**

**return;**

**}**

**int mid = (b + e) >> 1, l = n << 1, r = l + 1;**

**upd(l, b, mid, i, j, x);**

**upd(r, mid + 1, e, i, j, x);**

**tree[n] = tree[l] + tree[r]; // change here**

**}**

**int query(int n, int b, int e, int i, int j) {**

**push(n, b, e);**

**if(b > j || e < i) return 0; // return appropriate value**

**if(b >= i && e <= j) return tree[n];**

**int mid = (b + e) >> 1, l = n << 1, r = l + 1;**

**int L = query(l, b, mid, i, j);**

**int R = query(r, mid + 1, e, i, j);**

**return L + R; // change here**

**}**

**} st;**

**Strongest Community:**

**const int N = 1e5 + 9;**

**int a[N];**

**struct node {**

**int first\_element, first\_element\_cnt;**

**int last\_element, last\_element\_cnt;**

**int max\_cnt;**

**};**

**node merge(node l, node r) {**

**if(l.first\_element == -1) return r;**

**if(r.first\_element == -1) return l;**

**node ans;**

**ans.max\_cnt = max(l.max\_cnt, r.max\_cnt);**

**if(l.last\_element == r.first\_element) {**

**ans.max\_cnt = max(ans.max\_cnt, l.last\_element\_cnt + r.first\_element\_cnt);**

**}**

**ans.first\_element = l.first\_element;**

**ans.first\_element\_cnt = l.first\_element\_cnt;**

**if(l.first\_element == r.first\_element) {**

**ans.first\_element\_cnt += r.first\_element\_cnt;**

**}**

**ans.last\_element = r.last\_element;**

**ans.last\_element\_cnt = r.last\_element\_cnt;**

**if(r.last\_element == l.last\_element) {**

**ans.last\_element\_cnt += l.last\_element\_cnt;**

**}**

**return ans;**

**}**

**struct ST {**

**node tree[4 \* N];**

**void build(int n, int b, int e) {**

**if(b == e) {**

**tree[n].first\_element = a[b];**

**tree[n].first\_element\_cnt = 1;**

**tree[n].last\_element = a[b];**

**tree[n].last\_element\_cnt = 1;**

**tree[n].max\_cnt = 1;**

**return;**

**}**

**int mid = (b + e) >> 1, l = n << 1, r = l + 1;**

**build(l, b, mid);**

**build(r, mid + 1, e);**

**tree[n] = merge(tree[l], tree[r]); // change here**

**}**

**node query(int n, int b, int e, int i, int j) {**

**if(b > j || e < i) return {-1, -1, -1, -1, -1};**

**if(b >= i && e <= j) return tree[n];**

**int mid = (b + e) >> 1, l = n << 1, r = l + 1;**

**return merge(query(l, b, mid, i, j), query(r, mid + 1, e, i, j)); // change this**

**}**

**} st;**

**Segment tree with arithmetic progression [update the range l to r with the arithmetic progression a+b∗(i−l)]:**

**using ll = long long;**

**const int N = 2e5 + 9;**

**const ll inf = 1e18;**

**pair<ll, ll> merge(pair<ll, ll> l, pair<ll, ll> r) {**

**l.first = min(l.first, r.first);**

**l.second = max(l.second, r.second);**

**return l;**

**}**

**struct ST {**

**pair<ll, ll> tree[4 \* N];**

**ll lazy[4 \* N];**

**void push(int n, int b, int e) {**

**if (lazy[n] == 0) return;**

**tree[n].first += lazy[n];**

**tree[n].second += lazy[n];**

**if (b != e) {**

**int l = n << 1, r = l + 1;**

**lazy[l] += lazy[n];**

**lazy[r] += lazy[n];**

**}**

**lazy[n] = 0;**

**}**

**void build(int n, int  b, int e) {**

**lazy[n] = 0;**

**if (b == e) {**

**tree[n] = {0, 0};**

**return;**

**}**

**int mid = (b + e) >> 1, l = n << 1, r = l + 1;**

**build(l, b, mid);**

**build(r, mid + 1, e);**

**tree[n] = merge(tree[l], tree[r]);**

**}**

**void upd(int n, int b, int e, int i, int j, ll x) {**

**push(n, b, e);**

**if (b > j || e < i) return;**

**if (b >= i && e <= j) {**

**lazy[n] += x;**

**push(n, b, e);**

**return;**

**}**

**int mid = (b + e) >> 1, l = n << 1, r = l + 1;**

**upd(l, b, mid, i, j, x);**

**upd(r, mid + 1, e, i, j, x);**

**tree[n] = merge(tree[l], tree[r]);**

**}**

**pair<ll, ll> query(int n, int b, int e, int i, int j) {**

**push(n, b, e);**

**if (b > j || e < i) return {inf, -inf};**

**if (b >= i && e <= j) return tree[n];**

**int mid = (b + e) >> 1, l = n << 1, r = l + 1;**

**pair<ll, ll> L = query(l, b, mid, i, j);**

**pair<ll, ll> R = query(r, mid + 1, e, i, j);**

**return merge(L, R);**

**}**

**} st;**

**int32\_t main() {**

**ios\_base::sync\_with\_stdio(0);**

**cin.tie(0);**

**int n, q; cin >> n >> q;**

**st.build(1, 1, n);**

**while (q--) {**

**int type; cin >> type;**

**if (type == 1) {**

**int l, r; cin >> l >> r;**

**if (l == r) {**

**cout << 1 << '\n';**

**continue;**

**}**

**pair<ll, ll> ans = st.query(1, 1, n, l + 1, r);**

**if (ans.first == 0 and ans.second == 0) {**

**cout << 1 << '\n';**

**}**

**else {**

**cout << 0 << '\n';**

**}**

**}**

**else {**

**int l, r; ll a, b; cin >> l >> r >> a >> b;**

**st.upd(1, 1, n, l, l, a);**

**if (l != r) {**

**st.upd(1, 1, n, l + 1, r, b);**

**}**

**ll x = 1ll \* b \* (r - l) + a;**

**st.upd(1, 1, n, r + 1, r + 1, -x);**

**}**

**}**

**return 0;**

**}**

**Searching for the first element greater than a given amount:**

**int get\_first(int v, int tl, int tr, int l, int r, int x) {**

**if (tl > r || tr < l) return -1;**

**if (t[v] <= x) return -1;**

**if (tl == tr) return tl;**

**int tm = tl + (tr - tl) / 2;**

**int left = get\_first(2 \* v, tl, tm, l, r, x);**

**if (left != -1) return left;**

**return get\_first(2 \* v + 1, tm + 1, tr, l , r, x);**

**}**

**Counting the number of zeros, searching for the k-th zero:**

**int find\_kth(int v, int tl, int tr, int k) {**

**if (k > t[v])**

**return -1;**

**if (tl == tr)**

**return tl;**

**int tm = (tl + tr) / 2;**

**if (t[v \* 2] >= k)**

**return find\_kth(v \* 2, tl, tm, k);**

**else**

**return find\_kth(v \* 2 + 1, tm + 1, tr, k - t[v \* 2]);**

**}**

**Faisal Amin Abir**

**Trie Prefix tree**

**struct trieNode{**

**int cnt;**

**trieNode \*child[2];**

**trieNode(){**

**cnt=0;**

**for(int i=0; i<2; i++){**

**child[i]=NULL;**

**}}**

**};**

**trieNode \*root;**

**void insert(int num) {**

**binary form**

**trieNode \*cur=root;**

**for(int i=20; i>=0; i--){**

**int ind=(bool)(num&(1<<i));**

**if(!cur->child[ind]){**

**cur->child[ind]=new**

**trieNode();**

**}**

**cur=cur->child[ind];**

**cur->cnt++;**

**}}**

**\*\*Min XOR with value set\*\***

**-1 X: Add x to set**

**-2 x:for each element set it to element^x**

**-3: print the minimum of the set**

**int X=0;**

**void solve(int test){**

**int type,x;**

**si(type);**

**if(type==1){**

**si(x);**

**insert(x^X);}**

**else if(type==2){**

**si(x);**

**X^=x;}**

**else{**

**int res=query(X);**

**printf("%d\n",res);**

**}}**

**\*\* Segment Tree \*\***

**\*\*LAZY Propagation \*\***

**const ll N = 2e5+123;**

**ll arr[N], t[4\*N], lazy[4\*N], lazySet[4\*N];**

**bool isSet[4\*N];**

**ll n, m;**

**void build(ll v, ll l, ll r){**

**if(l==r){**

**t[v]=arr[l];**

**return;**

**}**

**ll mid = (l+r)>>1;**

**build(2\*v, l, mid);**

**build(2\*v+1, mid+1, r);**

**t[v] = t[2\*v] + t[2\*v+1];**

**}**

**void checkSet(ll v, ll L, ll R){**

**if(isSet[v]==1){**

**isSet[v]=0;**

**ll x = lazySet[v];**

**lazySet[v]=0;**

**t[v] = (R-L+1) \* x;**

**if(L!=R){**

**isSet[2\*v] = isSet[2\*v+1] = 1;**

**lazy[2\*v] = lazy[2\*v+1] = 0;**

**lazySet[2\*v] = lazySet[2\*v+1] = x;**

**}**

**}**

**}**

**void checkAdd(ll v, ll L, ll R){**

**if(lazy[v]!=0){**

**ll x = lazy[v];**

**lazy[v]=0;**

**t[v] += (R-L+1) \* x;**

**if(L!=R){**

**lazy[2\*v] += x;**

**lazy[2\*v+1] += x;**

**}**

**}**

**}**

**void add(ll v, ll L, ll R, ll l, ll r, ll value){**

**checkSet(v, L, R);**

**checkAdd(v, L, R);**

**if(L > r or R < l) return;**

**if( L >= l and R <= r){**

**t[v] += (R-L+1) \* value;**

**if(L!=R){**

**lazy[2\*v] += value;**

**lazy[2\*v+1] += value;**

**}**

**return;**

**}**

**ll mid = (L+R)>>1;**

**add(2\*v, L, mid, l, r, value);**

**add(2\*v+1, mid+1, R, l, r, value);**

**t[v] = t[2\*v] + t[2\*v+1];**

**}**

**void set\_value(ll v, ll L, ll R, ll l, ll r, ll value){**

**checkSet(v, L, R);**

**checkAdd(v, L, R);**

**if(L>r or R < l)return;**

**if(L>=l and R<=r){**

**t[v] = (R-L+1) \* value;**

**if(L!=R){**

**isSet[2\*v] = isSet[2\*v+1] = 1;**

**lazy[2\*v] = lazy[2\*v+1] = 0;**

**lazySet[2\*v] = lazySet[2\*v+1] = value;**

**}**

**return;**

**}**

**ll mid = (L+R)>>1;**

**set\_value(2\*v, L, mid, l, r, value);**

**set\_value(2\*v+1, mid+1, R, l, r , value);**

**t[v] = t[2\*v] + t[2\*v+1];**

**}**

**ll query(ll v, ll L, ll R, ll l, ll r){**

**checkSet(v, L, R);**

**checkAdd(v, L, R);**

**if(L>r or R<l)return 0;**

**if(L>=l and R <= r) return t[v];**

**ll mid = (L+R)>>1;**

**return query(2\*v, L, mid, l, r) + query(2\*v+1, mid+1, R, l, r);**

**}**

**void solve(){**

**cin>>n>>m;**

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

**cin>>arr[i];**

**}**

**build(1, 1, n);**

**for(ll i=0; i<m; i++){**

**ll c;**

**cin>>c;**

**if(c==1){**

**ll a, b, x;**

**cin>>a>>b>>x;**

**add(1, 1, n, a, b, x);**

**}**

**else if(c==2){**

**ll a, b, x;**

**cin>>a>>b>>x;**

**set\_value(1, 1, n, a, b, x);**

**}**

**else{**

**ll a, b;**

**cin>>a>>b;**

**cout << query(1, 1, n, a, b) << endl;**

**}}}**

**\*\*Max subarray sum in L to R with update\*\***

**const int mx=50005;**

**LL num[mx];**

**struct node{**

**LL left,right,sum,answer;**

**};**

**node tree[4\*mx];**

**node merge(node a,node b){**

**node x;**

**x.left=max(a.left,a.sum+b.left);**

**x.right=max(b.right,a.right+b.sum);**

**x.answer=max(max(a.answer,b.answer),a.right+b.left);**

**x.sum=a.sum+b.sum;**

**return x;**

**}**

**void buildSegmentTree(int at,int L,int R){**

**if(L==R){**

**tree[at].left=num[L];**

**tree[at].right=num[L];**

**tree[at].sum=num[L];**

**tree[at].answer=num[L];**

**return;**

**}**

**int mid=(L+R)/2;**

**buildSegmentTree(at\*2,L,mid);**

**buildSegmentTree(at\*2+1,mid+1,R);**

**tree[at]=merge(tree[at\*2],tree[at\*2+1]);**

**}**

**void updateSegemntTreeWithNew(int at,int L,int R,int pos,int value){**

**if(pos>R || pos<L){**

**return;**

**}**

**if(L>=pos && R<=pos){**

**tree[at].left=value;**

**tree[at].right=value;**

**tree[at].sum=value;**

**tree[at].answer=value;**

**return;**

**}**

**int mid=(L+R)/2;**

**updateSegemntTreeWithNew(at\*2,L,mid,pos,value);**

**updateSegemntTreeWithNew(at\*2+1,mid+1,R,pos,value);**

**tree[at]=merge(tree[at\*2],tree[at\*2+1]);**

**}**

**node def;**

**node querySegmentTree(int at,int L,int R,int l,int r){**

**if(r<L || R<l)return def;**

**if(l<=L && R<=r) return tree[at];**

**int mid=(L+R)/2;**

**node x=querySegmentTree(at\*2,L,mid,l,r);**

**node y=querySegmentTree(at\*2+1,mid+1,R,l,r);**

**return merge(x,y);**

**}**

**int main(){**

**def.answer=-100000000;**

**def.left=-100000000;**

**def.right-100000000;**

**def.sum=-100000000;**

**int n;**

**si(n);**

**Rep(i,1,n){**

**sl(num[i]);}**

**buildSegmentTree(1,1,n);**

**int q,l,r,x;**

**si(q);**

**while(q--){**

**siii(x,l,r);**

**if(x==0){**

**updateSegemntTreeWithNew(1,1,n,l,r);**

**}**

**else{**

**node ans=querySegmentTree(1,1,n,l,r);**

**printf("%lld\n",ans.answer);**

**}**

**}**

**return 0;**

**}**

**\*\*Longest increasing subsequence\*\***

**for(int i=n; i>=1; i--){**

**int qr=query(1,1,m,arr[i]+1,m);**

**int now=query(1,1,m,arr[i],arr[i]);**

**now=max(now,1+qr);**

**update(1,1,m,arr[i],now);**

**}**

**\*\*All possible longest increasing subsequence\*\***

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

**scl(arr[i].x)**

**arr[i].y=i+1;**

**}**

**sort(arr+1,arr+n+1,cmp);**

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

**lln d=arr[i].y;**

**lln q1=query(1,1,n+1,1,d-1);**

**update(1,1,n+1,d,d,q1+1);**

**}**

**lln ans=query(1,1,n+1,1,n+1);**

**printf("Case %d: %lld\n",xx,ans);**

**\*\* Merge sort tree without update \*\***

**vector<int>:: iterator child;**

**vector<int>vi2[4\*maxii];**

**int arr3[maxii];**

**int n,m;**

**int cc=1;**

**void segment(int node,int b,int e){**

**if(b==e){**

**vi2[node].pb(arr3[b]);**

**return;**

**}**

**int left=2\*node;**

**int right=left+1;**

**int mid=(b+e)/2;**

**segment(left,b,mid);**

**segment(right,mid+1,e);**

**int ff=0,hh=0;**

**while(ff<vi2[left].size() &&hh<vi2[right].size()){**

**if(vi2[left][ff]<=vi2[right][hh]){**

**vi2[node].pb(vi2[left][ff]);**

**ff++;**

**}**

**else{**

**vi2[node].pb(vi2[right][hh]);**

**hh++;**

**}**

**}**

**while(ff<vi2[left].size()){**

**vi2[node].pb(vi2[left][ff]);**

**ff++;**

**}**

**while(hh<vi2[right].size()){**

**vi2[node].pb(vi2[right][hh]);**

**hh++;**

**}**

**}**

**int w;**

**int query(int node,int b,int e,int st,int en){**

**if(st>e || en<b)return 0;**

**if(b>=st && e<=en){**

**int di=vi2[node].size();**

**int p=lower\_bound(vi2[node].begin(),vi2[node].end(),w)-vi2[node].begin();**

**return di-p;**

**}**

**int mid=(b+e)/2;**

**int left=2\*node;**

**int right=left+1;**

**int q1=query(left,b,mid,st,en);**

**int q2=query(right,mid+1,e,st,en);**

**return q1+q2;**

**}**

**\*\*Merge sort tree with update\*\***

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

**#include<ext/pb\_ds/assoc\_container.hpp>**

**#include<ext/pb\_ds/tree\_policy.hpp>**

**#define pii pair<int,int>**

**using namespace \_\_gnu\_pbds;**

**using namespace std;**

**typedef tree<pii, null\_type, less<pii>,rb\_tree\_tag, tree\_order\_statistics\_node\_update>ordered\_set;**

**const int maxn=3e5+5;**

**int arr[maxn];**

**ordered\_set segtree[4\*maxn];**

**void build(int at,int L,int R){**

**if(L==R){**

**segtree[at].insert({arr[L],L});**

**return;**

**}**

**int mid=(L+R)>>1;**

**int left=at\*2;**

**int right=at\*2+1;**

**build(left,L,mid);**

**build(right,mid+1,R);**

**for(pii i:segtree[left])segtree[at].insert(i);**

**for(pii i:segtree[right])segtree[at].insert(i);**

**}**

**void update(int at,int L,int R,int pos,pii rp,pii pt){**

**if(pos>R || pos<L){**

**return;**

**}**

**segtree[at].erase(rp);**

**segtree[at].insert(pt);**

**if(L==R){**

**return;**

**}**

**int mid=(L+R)>>1;**

**if(pos<=mid)update(at\*2,L,mid,pos,rp,pt);**

**else update(at\*2+1,mid+1,R,pos,rp,pt);**

**}**

**int query(int at,int L,int R,int l,int r,int val){**

**if(r<L || R<l)return 0;**

**if(l<=L && R<=r)return segtree[at].order\_of\_key({val,0});**

**int mid=(L+R)>>1;**

**return query(at\*2,L,mid,l,r,val)+query(at\*2+1,mid+1,R,l,r,val);**

**}**

**\*\*Total number of subarray whose sum less than t\*\***

**void solve(int test){**

**LL n,t,res=0;**

**sll(n,t);**

**Rep(i,1,n){**

**sl(arr[i]);**

**arr[i]+=arr[i-1];**

**}**

**build(1,1,n);**

**Rep(i,1,n){**

**res+=query(1,1,n,i,n,arr[i-1]+t);**

**}**

**printf("%lld\n",res);**

**}**

**\*\*Mo's Algorithm distinct element in range\*\***

**const int N = 2e5+2;**

**const int BLOCK = 450;**

**int arr[N], n, freq[N], cnt, ans[N];**

**struct query{**

**int l, r, index;**

**}q[200001];**

**void add(int value){**

**freq[value]++;**

**if(freq[value]==1)cnt++;**

**}**

**void remove(int value){**

**freq[value]--;**

**if(freq[value]==0)cnt--;**

**}**

**bool comp(query q1, query q2){**

**if((q1.l / BLOCK) == (q2.l / BLOCK)){**

**return q1.r < q2.r;**

**}**

**return (q1.l / BLOCK ) < (q2.l / BLOCK);**

**}**

**void solve(){**

**cin>>n;**

**int m;**

**cin>>m;**

**for(int i=0; i<n; i++) cin>>arr[i];**

**map<int, int>maps;**

**int c=0;**

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

**if(maps[arr[i]]==0){**

**c++;**

**maps[arr[i]]=c;**

**}}**

**for(int i=0; i<n; i++) arr[i] = maps[arr[i]];**

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

**int x, y;**

**cin>>x>>y;**

**x--, y--;**

**q[i].index = i;**

**q[i].l = x;**

**q[i].r = y;**

**}**

**sort(q, q + m, comp);**

**int L=0, R=-1;**

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

**int currL = q[i].l;**

**int currR = q[i].r;**

**int indx = q[i].index;**

**while(L-1 >= currL){//adding value**

**L--;**

**add(arr[L]);**

**}**

**while(R+1<=currR){**

**R++;**

**add(arr[R]);**

**}**

**while(L<currL){//removing value**

**remove(arr[L]);**

**L++;**

**}**

**while(R > currR){**

**remove(arr[R]);**

**R--;**

**}**

**ans[indx] = cnt;**

**}**

**for(int i=0; i<m; i++) cout << ans[i] << endl;**

**}**

\*\* GRAPH THEORY

**Faysal Ahammed**

**BFS:**

**const int N = 1e5 + 9;**

**vector<int> g[N];**

**vector<bool> vis(N, false);**

**void bfs(int u) {**

**queue<int> q;**

**q.push(u);**

**vis[u] = true;**

**while(!q.empty()) {**

**int top = q.front(); q.pop();**

**for(auto v: g[top]) {**

**if(!vis[v]) {**

**q.push(v);**

**vis[v] = true;}}}}**

**Find Cycle:**

**const int N = 2e5 + 9;**

**vector<int> g[N];**

**vector<int> par(N, -1);**

**vector<int> col(N, 0);**

**int cycle\_start, cycle\_end;**

**set<int> cycle;**

**void find\_cycle(int u) {**

**col[u] = 1;**

**for (auto v: g[u]) {**

**if (col[v] == 0) {**

**par[v] = u;**

**find\_cycle(v);**

**}**

**else if(col[v] == 1 and v != par[u]) {**

**cycle\_end = u;**

**cycle\_start = v;**

**}**

**}**

**col[u] = 2;**

**}**

**int32\_t main() {**

**ios\_base::sync\_with\_stdio(0);**

**cin.tie(0);**

**int n, m; cin >> n >> m;**

**for(int i = 1; i <= m; i++) {**

**int u, v; cin >> u >> v;**

**g[u].push\_back(v);**

**g[v].push\_back(u);**

**}**

**find\_cycle(1);**

**int cur = cycle\_end;**

**while (cur != cycle\_start) {**

**cycle.insert(cur);**

**cur = par[cur];**

**}**

**cycle.insert(cur);**

**return 0;**

**}**

**Topological Sort:**

**const int N = 1e5 + 9;**

**vector<int> g[N];**

**vector<int> indeg(N, 0);**

**int32\_t main() {**

**ios\_base::sync\_with\_stdio(0);**

**cin.tie(0);**

**int n, m; cin >> n >> m;**

**while(m--) {**

**int u, v; cin >> u >> v;**

**g[u].push\_back(v);**

**indeg[v]++;**

**}**

**queue<int> q;**

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

**if(indeg[i] == 0) {**

**q.push(i);**

**}}**

**vector<int> ans;**

**while(!q.empty()) {**

**int top = q.front();**

**q.pop();**

**ans.push\_back(top);**

**for(auto v: g[top]) {**

**indeg[v]--;**

**if(indeg[v] == 0) {**

**q.push(v);}}}**

**if(ans.size() == n) {**

**for(auto i: ans) {**

**cout << i << ' ';}**

**cout << '\n';}**

**else {**

**cout << "IMPOSSIBLE\n";}**

**return 0;**

**}**

**Bellman Ford:**

**using ll = long long;**

**const int N = 1005;**

**const ll inf = 1e18;**

**vector<pair<int, int>> g[N];**

**vector<ll> dis(N, inf);**

**int n, m;**

**bool cycle;**

**void bellman\_ford(int src) {**

**dis[src] = 0;**

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

**for (int u = 1; u <= n; u++) {**

**for (auto [v, w] : g[u]) {**

**if (dis[v] > dis[u] + w and dis[u] != inf) {**

**if (i == n) cycle = true;**

**dis[v] = dis[u] + w;}}}}}**

**int main() {**

**cin >> n >> m;**

**while (m--) {**

**int u, v, w; cin >> u >> v >> w;**

**g[u].push\_back({ v,w });**

**g[v].push\_back({ u,w });**

**}**

**cycle = false;**

**bellman\_ford(1);**

**for (int u = 1; u <= n; u++) {**

**cout << dis[u] << ' ';**

**}**

**cout << '\n';**

**return 0;**

**}**

**Dijkstra:**

**using ll = long long;**

**const int N = 1e5 + 9;**

**const ll inf = 1e18;**

**vector<pair<int, int>> g[N];**

**vector<bool> vis(N, false);**

**vector<ll> dis(N, inf);**

**int n, m;**

**void dijkstra(int u) {**

**dis[u] = 0;**

**priority\_queue<pair<ll, int>, vector<pair<ll, int>>, greater<pair<ll, int>>> pq;**

**pq.push({ 0, u });**

**while (!pq.empty()) {**

**int selected\_node = pq.top().second;**

**ll d = pq.top().first;**

**pq.pop();**

**if (vis[selected\_node]) continue;**

**vis[selected\_node] = true;**

**for (auto [v, w] : g[selected\_node]) {**

**if (dis[v] > (1ll \* d + w)) {**

**dis[v] = 1ll \* d + w;**

**pq.push({ dis[v], v });}}}}**

**Floyd Warshall:**

**using ll = long long;**

**const int N = 505;**

**const ll inf = 1e18;**

**int g[N][N];**

**ll dis[N][N];**

**int n, m;**

**void floyd\_warshall() {**

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

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

**if (i == j) dis[i][j] = 0;**

**else if (g[i][j] == 0) dis[i][j] = inf;**

**else dis[i][j] = g[i][j];**

**}**

**}**

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

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

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

**if (dis[i][k] < inf and dis[k][j] < inf)**

**dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);**

**}}}}**

**Krushkal’s MST:**

**const int N = 3e5 + 9, mod = 1e9;**

**struct dsu {**

**vector<int> par, rnk, size; int c;**

**dsu(int n) : par(n+1), rnk(n+1,0), size(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 size[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; size[j] += size[i];**

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

**return j;**

**}**

**};**

**int32\_t main() {**

**ios\_base::sync\_with\_stdio(0);**

**cin.tie(0);**

**int n, m; cin >> n >> m;**

**vector<array<int, 3>> ed;**

**for(int i = 1; i <= m; i++){**

**int u, v, w; cin >> u >> v >> w;**

**ed.push\_back({w, u , v});**

**}**

**sort(ed.begin(), ed.end());**

**long long ans = 0;**

**dsu d(n);**

**for (auto e: ed){**

**int u = e[1], v = e[2], w = e[0];**

**if (d.same(u, v)) continue;**

**ans += w;**

**d.merge(u, v);**

**}**

**cout << ans << '\n';**

**return 0;**

**}**

**Tree Diameter: max cost(distance) between 2 nodes. Dfs from any node and get 1 of the 2 nodes. Then dfs again from this node and get another 1. From every node, 1 of these two nodes is the max cost.**

**Faisal Amin Abir**

**\*\*Bellman Ford with negative cycle\*\***

**ll INF = 1e16+7; int n; int m;**

**struct edge{ ll a, b, cost;};**

**ll d[5005]; edge e[5005];**

**ll p[5005];**

**void solve(int v){**

**ll x=-1; d[v]=0;**

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

**x=-1;**

**for(ll j=0; j<m; j++)**

**if(d[e[j].a] < INF)**

**if(d[e[j]].b > d[e[j].a] + e[j].cost){**

**d[e[j].b] = max(-INF,d[e[j].a] + e[j].cost);**

**p[e[j].b] = e[j].a; x = e[j].b;**

**}**

**}**

**if(x==-1) cout << "NO" << endl;**

**else{**

**ll y = x;**

**for(ll i = 0; i<n; i++) y = p[y];**

**vector<ll>path;**

**for(ll curr=y;;curr=p[curr]){**

**path.push\_back(curr);**

**if(curr==y && path.size()>1){**

**break;**

**}**

**}**

**reverse(path.begin(), path.end());**

**cout << "YES" << endl;**

**for(auto u:path) cout << u << " ";**

**return ;**

**}**

**}**

**int main(){**

**cin>>n>>m;**

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

**edge ee; cin>> ee[i].a;**

**cin>>ee[i].b>>ee[i].cost;**

**}**

**solve(1); return 0;**

**}**

**\*\* FLOYD\_Warshall\*\***

**void Floyd\_Warshall(){**

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

**for(int j=0; j<=n; j++){**

**dis[i][j] = 100000000;}**

**dis[i][i]=0;}**

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

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

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

**dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);}}}}**

**\*\*Lexicographically minimum topological sort\*\***

**vector<int> vi[maxii];**

**vector<int>:: iterator child;**

**int check[maxii];**

**int in[maxii];**

**int n,m;**

**int cycle =0;**

**void dfs(int node){**

**if(cycle){**

**return;}**

**check[node]=1;**

**for(auto child: vi[node]){**

**if(check[child]==0){**

**dfs(child);**

**}**

**else if(check[child]==1){**

**cycle=1;return ;**

**}}check[node]=2;**

**}**

**int main(){**

**int test=1;**

**for(int xx=1; xx<=test; xx++){**

**cin>>n>>m;**

**int a,b;**

**for(int i=1; i<=m; i++){**

**cin>>a>>b;**

**vi[a].push\_back(b);**

**in[b]++;**

**}**

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

**if(check[i]==0){**

**dfs(i);**

**}}**

**if(cycle){**

**cout<<-1;**

**return 0;}**

**priority\_queue<pair<int,int>,vector< pair<int,int>>,greater<pair<int,int> > > que;**

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

**if(in[i]==0){**

**que.push({i,0});**

**}}**

**while(!que.empty()){**

**int node=que.top().first;**

**int das=que.top().second;**

**que.pop();**

**if(das==0){**

**cout<<node<<" ";**

**for(int child:vi[node])**

**{**

**in[child]--;**

**que.push({child,in[child]});**

**}}}}}**

**\*\* HLD to find max between u and v, with euler path,  LCA and segtree \*\***

**// dfs for subtree cal, dfs2 for euler**

**const int N = 1e4+1;;**

**const int K = \_\_lg(N) + 2;**

**int LOG[N], parent[N], value[N], level[N], subtree[N], first[N], chain[N], in[N], out[N], lca[N][K], head[N], t[4\*N], n;**

**vector<pair<int, int>>g[N];**

**vector<int>euler;**

**void dfs(int v, int par, int val){**

**parent[v]=par;**

**value[v]=val;**

**lca[v][0] = par;**

**int sub = 1;**

**for(auto u:g[v]){**

**if(u.ff==par)continue;**

**level[u.ff] = level[v] + 1;**

**dfs(u.ff, v, u.ss);**

**sub += subtree[u.ff];**

**}**

**subtree[v]=sub;**

**}**

**void dfs2(int v, int par, int time, int matha){**

**euler.pb(v);**

**head[v] = matha;**

**int sub=0, node=0;**

**in[v] = time;**

**for(auto u:g[v]){**

**if(u.ff==par) continue;**

**if(subtree[u.ff]>sub){**

**sub = subtree[u.ff];**

**node = u.ff;**

**}}**

**if(sub!=0){**

**dfs2(node, v, time+1, matha);**

**time = out[node];}**

**for(auto u:g[v]){**

**if(u.ff == par or u.ff == node) continue;**

**dfs2(u.ff, v, time+1, u.ff);**

**time=out[u.ff];**

**}**

**out[v] = time;**

**}**

**void build(int v, int l, int r){**

**if(l==r){**

**t[v] = value[euler[l]];**

**return;**

**}**

**int mid = (l+r) >> 1;**

**build(2\*v, l, mid);**

**build(2\*v+1, mid+1, r);**

**t[v] = max(t[2\*v] , t[2\*v+1]);**

**}**

**int max\_query(int v, int L, int R, int l, int r){**

**if(L>r or R<l) return 0;**

**if(L>=l and R<=r) return t[v];**

**int mid = (L+R) >> 1;**

**int left = max\_query(2\*v, L, mid, l, r);**

**int right = max\_query(2\*v+1, mid+1, R, l, r);**

**return max(left, right);**

**}**

**void update(int v, int l, int r, int index, int value){**

**if(l==r){**

**t[v] = value;**

**return;**

**}**

**int mid = (l+r) >> 1;**

**if(index <= mid) update(2\*v, l, mid, index, value);**

**else update(2\*v+1, mid+1, r, index, value);**

**t[v] = max(t[2\*v], t[2\*v+1]);**

**}**

**void buildLCA(){**

**for(int i=1; i<K; i++){**

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

**int x = lca[j][i-1];**

**if(x != -1){**

**lca[j][i] = lca[x][i-1];**

**}}}}**

**int find\_Lca(int a, int b){**

**if(level[a] < level[b]) swap(a, b);**

**int z = level[a] - level[b];**

**while(z>0){**

**int i = LOG[z];**

**a = lca[a][i];**

**z-= (1<<i);**

**}**

**if(a==b) return a;**

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

**if(lca[a][i]!=-1 and lca[a][i]!=lca[b][i]){**

**a = lca[a][i];**

**b = lca[b][i];**

**}**

**}**

**return lca[a][0];**

**}**

**void init(){**

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

**g[i].clear();**

**for(int j=0; j<K; j++){**

**lca[i][j]=-1;}}**

**euler.clear();**

**}**

**int find\_max(int a, int root){**

**if(chain[a] == chain[root]){**

**return max\_query(1, 0, n-1, in[root], in[a]);**

**}**

**else{**

**int x = max\_query(1, 0, n-1, in[head[a]], in[a]);**

**x = max(x, find\_max(parent[head[a]], root));**

**return x;**

**}}**

**void solve(){**

**cin>>n;**

**init();**

**vector<pair<int, int>>edge;**

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

**int x,y, cost;**

**cin>>x>>y>>cost;**

**g[x].push\_back({y, cost});**

**g[y].push\_back({x, cost});**

**edge.pb({x, y});**

**}**

**dfs(1, -1, -1);**

**dfs2(1, -1, 0, 1);**

**int currChain=1;**

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

**int x = euler[i];**

**chain[x] = currChain;**

**if(in[x]==out[x]) currChain++;**

**}**

**build(1, 0, n-1);**

**buildLCA();**

**while(1){**

**string s;**

**cin>>s;**

**if(s=="DONE")break;**

**if(s=="QUERY"){**

**int a, b;**

**cin>>a>>b;**

**int y = find\_Lca(a, b);**

**update(1, 0, n-1, in[y], 0);**

**int ans = find\_max(a, y);**

**ans = max(ans, find\_max(b, y));**

**cout << ans << endl;**

**update(1,0, n-1, in[y], value[y]);**

**}**

**else{**

**int index, val;**

**cin>>index>>val;**

**index--;**

**int a, b;**

**a = edge[index].ff;**

**b = edge[index].ss;**

**if(parent[a]==b) update(1, 0, n-1, in[a], val), value[a]=val;**

**else{**

**update(1, 0, n-1, in[b], val);**

**value[b]=val;**

**}}}**

**}**

**int main(){**

**LOG[1] = 0;**

**for(int i=2; i<N; i++){**

**LOG[i] = LOG[i/2] + 1;**

**}**

**solve();**

**return 0;}**

**SSC find toposort, component**

**->given a forest, tell which node to forward mail to reach max people, lexicographically smallest one;**

**const int N = 1e5+123;**

**vector<int>g1[N], g2[N], g3[N], topo;**

**int totalCom, com[N], got, countCom[N], minCom[N], dis[N], indeg[N];**

**bool vis1[N], vis2[N], vis3[N];**

**int n, m;**

**void reset(){**

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

**g1[i].clear();**

**g2[i].clear();**

**g3[i].clear();**

**indeg[i]=dis[i]=countCom[i]=com[i]=vis1[i]=vis2[i]=vis3[i]=0;**

**minCom[i]=INT\_MAX;**

**}**

**totalCom=0;**

**topo.clear();**

**got=0;**

**}**

**void dfs1(int v){**

**if(vis1[v])return;**

**vis1[v]=1;**

**for(auto u:g1[v]){**

**dfs1(u);**

**}**

**topo.pb(v);**

**}**

**void dfs2(int v){**

**if(vis2[v])return;**

**vis2[v]=1;**

**com[v]=totalCom;**

**countCom[totalCom]++;**

**minCom[totalCom] = min(minCom[totalCom], v);**

**for(auto u:g2[v]){**

**dfs2(u);**

**}**

**}**

**void solve(){**

**cin>>n;**

**reset();**

**vector<pair<int, int>>v;**

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

**int x, y;**

**cin>>x>>y;**

**g1[x].pb(y);**

**g2[y].pb(x);**

**v.pb({x, y});**

**}**

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

**if(vis1[i]==0)**

**dfs1(i);**

**}**

**reverse(all(topo));**

**for(auto u:topo){**

**if(vis2[u]==0){**

**totalCom++;**

**dfs2(u);**

**}**

**}**

**for(auto u:v){**

**int f = com[u.ff];**

**int t = com[u.ss];**

**if(f!=t){ // from B to A, we find the node in reverse direction to get highest count**

**g3[t].pb(f);**

**indeg[f]++;**

**}**

**}**

**using pii = pair<int, int>;**

**priority\_queue<pii, vector<pii>, greater<pii>>q;**

**for(int i=1; i<=totalCom; i++){**

**if(indeg[i]==0){**

**q.push({countCom[i], i});**

**dis[i]=countCom[i];**

**}**

**}**

**while(!q.empty()){**

**int from = q.top().ss;**

**q.pop();**

**for(auto u:g3[from]){**

**if(dis[u]<dis[from]+countCom[u]){**

**dis[u] = dis[from] + countCom[u];**

**q.push({dis[u], u});**

**}**

**}**

**}**

**int tot=0;**

**int ans=INT\_MAX;**

**for(int i=1; i<=totalCom; i++){**

**if(dis[i]>tot){**

**tot = dis[i];**

**ans = minCom[i];}**

**else if(tot == dis[i]){**

**ans = min(ans, minCom[i]);}**

**}**

**cout << "Case " << ++test << ": " << ans << endl;**

**}**

**SSC to find component of component**

**const int N = 1e5+123;**

**vector<int>g1[N], g2[N], g3[N], topo, topoCom;**

**int com[N], totalCom;**

**bool vis1[N], vis2[N], vis3[N], vis4[N];**

**void dfs1(int v){// find the toposort of the given graph**

**if(vis1[v])return;**

**vis1[v]=1;**

**for(auto u:g1[v]){**

**dfs1(u);}**

**topo.pb(v);**

**}**

**void dfs2(int v){ // find the scc from the toposort using the transpose of given graph**

**if(vis2[v])return;**

**vis2[v]=1;**

**com[v]=totalCom;**

**for(auto u:g2[v]){**

**dfs2(u);}**

**}**

**void dfs3(int v){ // find the toposort of the component graph**

**if(vis3[v])return;**

**vis3[v]=1;**

**for(auto u:g3[v]){**

**dfs3(u);}**

**topoCom.pb(v);**

**}**

**void dfs4(int v){// runs by the toposort of the component graph**

**if(vis4[v])return;**

**vis4[v]=1;**

**for(auto u:g3[v]){**

**dfs4(u);}**

**}**

**void solve(){**

**int n,m;**

**cin>>n>>m;**

**vector<pair<int, int>>v;**

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

**int x, y;**

**cin>>x>>y;**

**g1[x].pb(y);**

**g2[y].pb(x);**

**v.pb({x, y});**

**}**

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

**if(vis1[i]==0){**

**dfs1(i);}**

**}**

**reverse(all(topo));**

**for(auto u:topo){**

**if(vis2[u]==0){**

**totalCom++;**

**dfs2(u);}**

**}**

**for(auto u:v){**

**int from = com[u.ff];**

**int to = com[u.ss];**

**if(from!=to){**

**g3[to].pb(from);}**

**}**

**for(int i=1; i<=totalCom; i++){**

**if(vis3[i]==0){**

**dfs3(i);}**

**}**

**reverse(all(topoCom));**

**int start = topoCom[0];**

**dfs4(start);**

**for(int i=1; i<=totalCom; i++){**

**if(vis4[i]==0){**

**cout << 0 << endl;**

**return;}**

**}**

**vector<int>nodes;**

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

**if(com[i]==start){**

**nodes.pb(i);}**

**}**

**cout << nodes.size() << endl;**

**for(auto u:nodes)cout<<u<<" "; cout << endl;**

**}**

**Kruskals**

**#define MX 100005**

**int parent[MX], R[MX];**

**struct kruskalStruct{**

**int u,v,w;**

**};**

**static bool cmp(kruskalStruct &a, kruskalStruct &b){**

**return a.w < b.w;**

**}**

**void init(int v){**

**for(int i = 0; i <= v; i++){**

**parent[i] = i;**

**R[i] = 1;**

**}**

**}**

**int Find(int p){**

**if(p == parent[p]) return p;**

**return parent[p] = Find(parent[p]);**

**}**

**bool Union(int u,int v){**

**int p = Find(u);**

**int q = Find(v);**

**if(p != q) {**

**if(R[p] >= R[q]){**

**parent[q] = p;**

**R[p] += R[q];**

**}**

**else{**

**parent[p] = q;**

**R[q] += R[p];**

**}**

**return true;**

**}**

**return false;**

**}**

**vector<kruskalStruct> store;**

**void kruskalsMST(){**

**int vertex,edge;**

**cin >> vertex >> edge;**

**init(vertex);**

**for(int i = 0; i < edge; i++) {**

**int u,v,w;**

**cin >> u >> v >> w;**

**kruskalStruct ks;**

**ks.u = u;**

**ks.v = v;**

**ks.w = w;**

**store.push\_back(ks);**

**}**

**sort(store.begin(),store.end(),cmp);**

**int totalWeight = 0;**

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

**if(Union(store[i].u,store[i].v)) totalWeight += store[i].w;**

**}**

**cout << "Kruskal's MST : " << totalWeight << endl;**

**}**

\*\* STRING

**Faisal Amin Abir**

**\*\*String Multiply\*\***

**string multiply(string num1, string num2){**

**int len1 = num1.size();**

**int len2 = num2.size();**

**if (len1 == 0 || len2 == 0)return "0";**

**vector<int> result(len1 + len2, 0);**

**int i\_n1 = 0;**

**int i\_n2 = 0;**

**for (int i=len1-1; i>=0; i--){**

**int carry = 0;**

**int n1 = num1[i] - '0';**

**i\_n2 = 0;**

**for (int j=len2-1; j>=0; j--){**

**int n2 = num2[j] - '0';**

**int sum = n1\*n2 + result[i\_n1 +i\_n2] + carry;**

**carry = sum/10;**

**result[i\_n1 + i\_n2] = sum % 10;**

**i\_n2++;**

**}**

**if (carry > 0)result[i\_n1 + i\_n2]+= carry;**

**i\_n1++;**

**}**

**int i = result.size() - 1;**

**while (i>=0 && result[i] == 0)i--;**

**if (i == -1)return "0";**

**string s = "";**

**while (i >= 0)s +=std::to\_string(result[i--]);**

**return s;}**

**\*\*String division\*\***

**string longDivision(string number, int divisor){**

**string ans;**

**int idx = 0;**

**int temp = number[idx] - '0';**

**while (temp < divisor)**

**temp = temp \* 10 + (number[++idx] -'0');**

**while (number.size() > idx){**

**ans += (temp / divisor) + '0';**

**temp = (temp % divisor) \* 10 +**

**number[++idx] - '0';**

**}**

**if (ans.length() == 0)return "0";**

**return ans;**

**}**

**\*\*String Double Hashing\*\***

**const int MAXN=1000006;**

**namespace DoubleHash{**

**long long P[2][MAXN];**

**long long H[2][MAXN];**

**long long R[2][MAXN];**

**long long base[2];**

**long long mod[2];**

**void gen(){**

**base[0] = 1949313259ll;**

**base[1] = 1997293877ll;**

**mod[0]  = 2091573227ll;**

**mod[1]  = 2117566807ll;**

**for(int j=0;j<2;j++){**

**for(int i=0;i<MAXN;i++){**

**H[j][i]=R[j][i] = 0ll;**

**P[j][i] = 1ll;**

**}**

**}**

**for(int j=0;j<2;j++){**

**for(int i=1;i<MAXN;i++){**

**P[j][i] = (P[j][i-1] \* base[j])%mod[j];**

**}**

**}**

**}**

**void make\_hash(string arr){**

**int len = arr.size();**

**for(int j=0;j<2;j++){**

**for (int i = 1; i <= len; i++)H[j][i] = (H[j][i - 1] \* base[j] + arr[i - 1] + 1007) % mod[j];**

**//            for (int i = len; i >= 1; i--)R[j][i] = (R[j][i + 1] \* base[j] + arr[i - 1] + 1007) % mod[j];**

**}**

**}**

**inline long long range\_hash(int l,int r,int idx){**

**long long hashval = H[idx][r + 1] - ((long long)P[idx][r - l + 1] \* H[idx][l] % mod[idx]);**

**return (hashval < 0 ? hashval + mod[idx] : hashval);**

**}**

**inline long long reverse\_hash(int l,int r,int idx){**

**long long hashval = R[idx][l + 1] - ((long long)P[idx][r - l + 1] \* R[idx][r + 2] % mod[idx]);**

**return (hashval < 0 ? hashval + mod[idx] : hashval);**

**}**

**inline long long range\_dhash(int l,int r){**

**long long x = range\_hash(l,r,0);**

**return (x<<32)^range\_hash(l,r,1);**

**}**

**inline long long reverse\_dhash(int l,int r){**

**long long x = reverse\_hash(l,r,0);**

**return (x<<32)^reverse\_hash(l,r,1);**

**}**

**}**

**char str1[MAXN];**

**using namespace DoubleHash;**

**\*\*Reverse Hashing to Find longest palindromic substring\*\*\***

**int f1(int index){**

**if(index == 0 or index==n-1) return 1;**

**int ans = 1;**

**int l = 1, r = min(index, n-1-index);**

**while( l <= r){**

**int mid = (l+r) >> 1;**

**ll h1 = range\_hash(index+1, index + mid);**

**ll h2 = reverse\_range\_hash(index-mid, index-1);**

**if(h1==h2){**

**ans = 1 +  ( 2 \* mid);**

**l = mid+1;**

**}**

**else r = mid-1;**

**}**

**return ans;**

**}**

**int f2(int index){**

**if(index==0 or index==n-2) return 2;**

**int ans=2;**

**int l = 1, r = min(index+1, n-1-index);**

**while( l <= r){**

**int mid = (l+r) >> 1;**

**ll h1 = range\_hash(index-mid+1, index);**

**ll h2 = reverse\_range\_hash(index + 1, index+mid);**

**if(h1==h2){**

**ans = 2 \* mid;**

**l = mid+1;**

**}**

**else r = mid-1;**

**}**

**return ans;**

**}**

**void solve(){**

**cin>>s;**

**n = s.size();**

**ll ans=1;**

**ll l = 0, r=0;**

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

**ll x = f1(i);//for odd length i is mid elem**

**if(x > ans){**

**ans = x;**

**l = i - (x/2);**

**r = i + (x/2);**

**}**

**if(i + 1 < n and s[i] == s[i+1]){**

**ll y = f2(i);// for even length**

**if( y > ans){**

**ans = y;**

**l = i - (y/2) + 1;**

**r = i + (y/2);**

**}}}**

**for(int i=l ; i<=r ; i++) cout<<s[i];**

**}**

**\*\*Manacher to Find the Longest Palindromic Substring\*\***

**void solve(){**

**string s;**

**cin>>s;**

**string temp = "";**

**temp.pb('#');**

**for(auto u:s){**

**temp.pb(u);**

**temp.pb('#');**

**}**

**int n = temp.length();**

**int l=0, r=-1;**

**int pi[n]={0};**

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

**int k=0;**

**if( i > r){**

**k = 0;**

**}**

**else{**

**k = min(r-i, pi[l + r - i]);**

**}**

**while( i + k + 1 < n && i - k - 1 >=0 && temp[i + k + 1] == temp[i - k - 1]){**

**k++;**

**}pi[i]=k;**

**if( i + k > r){**

**r = i + k;**

**l = i - k;**

**}**

**}**

**int even=0, odd=0, index=0, index2=0, ans=0;**

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

**if(i%2==1){**

**index++;**

**}**

**if(pi[i]>ans){**

**ans = pi[i];**

**if(i%2){**

**odd=1;**

**even=0;**

**index2 = index;**

**}**

**else{**

**even=1;**

**odd=0;**

**index2=index;**

**}}}**

**index2--;**

**if(odd){**

**int l = index2, r = index2;**

**int k = 1;**

**while(k<ans){**

**l--, r++;**

**k+=2;**

**}for(int i=l; i<=r; i++)cout<<s[i];**

**}**

**else {**

**int l = index2, r = index2+1;int k = 2;**

**while( k < ans){**

**l--, r++;**

**k+=2;**

**}for(int i=l; i<=r; i++)cout<<s[i];**

**}}**

**->Given a string with m <= 2e5 operations:**

**a. Change the char at index to x**

**b. find if substring between given l, r is palindrome?**

**const int N = 2e5+12;**

**ll t[4\*N], reverse\_t[4\*N], power[N], inv[N];**

**string s;**

**int n, q;**

**void build(int v, int l, int r){**

**if(l==r){**

**t[v] = (power[l] \* (s[l]-'a'+1)) % MOD;**

**reverse\_t[v] = (power[n-1-l] \* (s[l]-'a'+1)) % MOD;**

**return;**

**}**

**int mid = (l+r) >> 1;**

**build(2\*v, l, mid);**

**build(2\*v+1, mid+1, r);**

**t[v] = (t[2\*v] + t[2\*v+1]) % MOD;**

**reverse\_t[v] = (reverse\_t[2\*v] + reverse\_t[2\*v+1]) % MOD;**

**}**

**void update(int v, int l, int r, int index, char ch){**

**if(l==r){**

**s[l]=ch;**

**t[v] = (power[l] \* (s[l]-'a'+1)) % MOD;**

**reverse\_t[v] = (power[n-1-l] \* (s[l]-'a'+1)) % MOD;**

**return;**

**}**

**int mid = (l+r) >> 1;**

**if(index <= mid) update(2\*v, l, mid, index, ch);**

**else update(2\*v+1, mid+1, r, index, ch);**

**t[v] = (t[2\*v] + t[2\*v+1]) % MOD;**

**reverse\_t[v] = (reverse\_t[2\*v] + reverse\_t[2\*v+1]) % MOD;**

**}**

**pair<ll, ll> query(int v, int L, int R, int l, int r){**

**if(L>r or R<l ) return {0, 0};**

**if(L>=l and R<=r) return {t[v], reverse\_t[v]};**

**int mid = (L+R) >> 1;**

**pair<ll, ll>p1 = query(2\*v, L, mid, l, r);**

**pair<ll, ll>p2 = query(2\*v+1, mid+1, R, l, r);**

**return {(p1.ff + p2.ff) % MOD,  (p1.ss + p2.ss) % MOD};**

**}**

**void solve(){**

**cin>>n>>q;**

**cin>>s;**

**power[0]=1;**

**power[1]=base;**

**inv[0]=1;**

**inv[1]=modPow(base, MOD-2);**

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

**power[i] = (1LL \* power[i-1] \* base) % MOD;**

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

**}**

**build(1, 0, n-1);**

**while(q--){**

**int x;**

**cin>>x;**

**if(x==1){**

**int index; char ch;**

**cin>>index >> ch;**

**update(1, 0, n-1, index-1, ch);**

**}**

**else{**

**int l, r;**

**cin>>l>>r;**

**l--, r--;**

**pair<ll, ll> p = query(1, 0, n-1, l, r);**

**ll h1 = (inv[l] \* p.ff) % MOD;**

**ll h2 = (inv[n-1-r] \* p.ss) % MOD;**

**if(h1==h2){**

**cout << "YES" << endl;**

**}**

**else cout << "NO" << endl;**

**}}}**

**FAYSAL AHAMMED**

**Hashing:**

**\* Double Hashing with Reverse:**

**const int MOD1 = 127657753, MOD2 = 987654319;**

**const int p1 = 137, p2 = 277; // change here**

**int ip1, ip2;**

**pair<int, int> pw[N], ipw[N];**

**void prec() {**

**pw[0] = {1, 1};**

**for (int i = 1; i < N; i++) {**

**pw[i].first = 1ll \* pw[i - 1].first \* p1 % MOD1;**

**pw[i].second = 1ll \* pw[i - 1].second \* p2 % MOD2;**

**}**

**ip1 = power(p1, MOD1 - 2, MOD1);**

**ip2 = power(p2, MOD2 - 2, MOD2);**

**ipw[0] = {1, 1};**

**for (int i = 1; i < N; i++) {**

**ipw[i].first = 1ll \* ipw[i - 1].first \* ip1 % MOD1;**

**ipw[i].second = 1ll \* ipw[i - 1].second \* ip2 % MOD2;**

**}**

**}**

**struct Hashing {**

**int n;**

**string s;**

**vector<pair<int, int>> hash\_val;**

**vector<pair<int, int>> rev\_hash\_val;**

**Hashing() {}**

**Hashing(string \_s) {**

**s = \_s;**

**n = s.size();**

**hash\_val.emplace\_back(0, 0);**

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

**pair<int, int> p;**

**p.first = (hash\_val[i].first + 1ll \* s[i] \* pw[i].first % MOD1) % MOD1;**

**p.second = (hash\_val[i].second + 1ll \* s[i] \* pw[i].second % MOD2) % MOD2;**

**hash\_val.push\_back(p);**

**}**

**rev\_hash\_val.emplace\_back(0, 0);**

**for (int i = 0, j = n - 1; i < n; i++, j--) {**

**pair<int, int> p;**

**p.first = (rev\_hash\_val[i].first + 1ll \* s[i] \* pw[j].first % MOD1) % MOD1;**

**p.second = (rev\_hash\_val[i].second + 1ll \* s[i] \* pw[j].second % MOD2) % MOD2;**

**rev\_hash\_val.push\_back(p);**

**}**

**}**

**pair<int, int> get\_hash(int l, int r) { // 1 indexed**

**pair<int, int> ans;**

**ans.first = (hash\_val[r].first - hash\_val[l - 1].first + MOD1) \* 1ll \* ipw[l - 1].first % MOD1;**

**ans.second = (hash\_val[r].second - hash\_val[l - 1].second + MOD2) \* 1ll \* ipw[l - 1].second % MOD2;**

**return ans;**

**}**

**pair<int, int> rev\_hash(int l, int r) { // 1 indexed**

**pair<int, int> ans;**

**ans.first = (rev\_hash\_val[r].first - rev\_hash\_val[l - 1].first + MOD1) \* 1ll \* ipw[n - r].first % MOD1;**

**ans.second = (rev\_hash\_val[r].second - rev\_hash\_val[l - 1].second + MOD2) \* 1ll \* ipw[n - r].second % MOD2;**

**return ans;**

**}**

**pair<int, int> get\_hash() { // 1 indexed**

**return get\_hash(1, n);**

**}**

**bool is\_palindrome(int l, int r) {**

**return get\_hash(l, r) == rev\_hash(l, r);**

**}**

**};**

**Hashing with Updates and Reverse:**

**using T = array<int, 2>;**

**const T MOD = {127657753, 987654319};**

**const T p = {137, 277}; // change here**

**T operator + (T a, int x) {return {(a[0] + x) % MOD[0], (a[1] + x) % MOD[1]};}**

**T operator - (T a, int x) {return {(a[0] - x + MOD[0]) % MOD[0], (a[1] - x + MOD[1]) % MOD[1]};}**

**T operator \* (T a, int x) {return {(int)((long long) a[0] \* x % MOD[0]), (int)((long long) a[1] \* x % MOD[1])};}**

**T operator + (T a, T x) {return {(a[0] + x[0]) % MOD[0], (a[1] + x[1]) % MOD[1]};}**

**T operator - (T a, T x) {return {(a[0] - x[0] + MOD[0]) % MOD[0], (a[1] - x[1] + MOD[1]) % MOD[1]};}**

**T operator \* (T a, T x) {return {(int)((long long) a[0] \* x[0] % MOD[0]), (int)((long long) a[1] \* x[1] % MOD[1])};}**

**ostream& operator << (ostream& os, T hash) {return os << "(" << hash[0] << ", " << hash[1] << ")";}**

**T pw[N], ipw[N];**

**void prec() {**

**pw[0] =  {1, 1};**

**for (int i = 1; i < N; i++) {**

**pw[i] = pw[i - 1] \* p;**

**}**

**ipw[0] =  {1, 1};**

**T ip = {power(p[0], MOD[0] - 2, MOD[0]), power(p[1], MOD[1] - 2, MOD[1])};**

**for (int i = 1; i < N; i++) {**

**ipw[i] = ipw[i - 1] \* ip;**

**}**

**}**

**struct Hashing {**

**int n;**

**string s; // 1 - indexed**

**vector<array<T, 2>> t; // (normal, rev) hash**

**array<T, 2> merge(array<T, 2> l, array<T, 2> r) {**

**l[0] = l[0] + r[0];**

**l[1] = l[1] + r[1];**

**return l;**

**}**

**void build(int node, int b, int e) {**

**if (b == e) {**

**t[node][0] = pw[b] \* s[b];**

**t[node][1] = pw[n - b + 1] \* s[b];**

**return;**

**}**

**int mid = (b + e) >> 1, l = node << 1, r = l | 1;**

**build(l, b, mid);**

**build(r, mid + 1, e);**

**t[node] = merge(t[l], t[r]);**

**}**

**void upd(int node, int b, int e, int i, char x) {**

**if (b > i || e < i) return;**

**if (b == e && b == i) {**

**t[node][0] = pw[b] \* x;**

**t[node][1] = pw[n - b + 1] \* x;**

**return;**

**}**

**int mid = (b + e) >> 1, l = node << 1, r = l | 1;**

**upd(l, b, mid, i, x);**

**upd(r, mid + 1, e, i, x);**

**t[node] = merge(t[l], t[r]);**

**}**

**array<T, 2> query(int node, int b, int e, int i, int j) {**

**if (b > j || e < i) return {T({0, 0}), T({0, 0})};**

**if (b >= i && e <= j) return t[node];**

**int mid = (b + e) >> 1, l = node << 1, r = l | 1;**

**return merge(query(l, b, mid, i, j), query(r, mid + 1, e, i, j));**

**}**

**Hashing() {}**

**Hashing(string \_s) {**

**n = \_s.size();**

**s = "." + \_s;**

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

**build(1, 1, n);**

**}**

**void upd(int i, char c) {**

**upd(1, 1, n, i, c);**

**s[i] = c;**

**}**

**T get\_hash(int l, int r) { // 1 - indexed**

**return query(1, 1, n, l, r)[0] \* ipw[l - 1];**

**}**

**T rev\_hash(int l, int r) { // 1 - indexed**

**return query(1, 1, n, l, r)[1] \* ipw[n - r];**

**}**

**T get\_hash() {**

**return get\_hash(1, n);**

**}**

**bool is\_palindrome(int l, int r) {**

**return get\_hash(l, r) == rev\_hash(l, r);**

**}**

**};**

**Longest Common Prefix:**

**Binary Search over len and get true/false by hashing in O(1).**

**Lexicographically Min Cyclic Shift:**

**// return 0 if both equal**

**// return 1 if first substring greater**

**// return -1 if second substring greater**

**int compare(int i, int j, int x, int y) {**

**int common\_prefix = lcp(i, j, x, y);**

**int len1 = j - i + 1, len2 = y - x + 1;**

**if (common\_prefix == len1 and len1 == len2) return 0;**

**else if (common\_prefix == len1) return -1;**

**else if (common\_prefix == len2) return 1;**

**else return (s[i + common\_prefix - 1] < s[x + common\_prefix - 1] ? -1 : 1);**

**}**

**int start = 1, end = k;**

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

**int x = compare(start, end, i, i + k - 1);**

**if (x == 1) {**

**start = i, end = i + k - 1;**

**}**

**}**

**cout << s.substr(start - 1, k) << '\n';**

**Cyclic Shift trick: s += s**

**Number of Palindromic Substring between l to r in O(n ^2):**

**int is\_palindrome(int i, int j) { // O(n^2)**

**if (i > j) return 1;**

**int &ans = dp2[i][j];**

**if (ans != -1) return ans;**

**ans = 1;**

**if (s[i] == s[j]) {**

**ans &= is\_palindrome(i + 1, j - 1);**

**}**

**else {**

**ans = 0;**

**}**

**return ans;**

**}**

**int fun(int i, int j) { // O(n^2)**

**if (i > j) return 0;**

**int &ans = dp[i][j];**

**if (ans != -1) return ans;**

**ans = is\_palindrome(i, j) + fun(i + 1, j) + fun(i, j - 1) - fun(i + 1, j - 1);**

**return ans;**

**}**

**Number of Palindromic Substrings of a String in O(n logn):**

**string s;**

**Hashing hash\_s;**

**int n;**

**bool ok(int l, int r) {**

**return hash\_s.is\_palindrome(l, r);**

**}**

**int32\_t main() {**

**ios\_base::sync\_with\_stdio(0);**

**cin.tie(0);**

**prec(); // must include**

**cin >> s;**

**n = s.size();**

**hash\_s = Hashing(s);**

**long long ans = 0;**

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

**int l = 0, r = min(n - i, i - 1), cnt = 1;**

**while (l <= r) {**

**int mid = (l + r) >> 1;**

**if (ok(i - mid, i + mid)) {**

**cnt = mid;**

**l = mid + 1;**

**}**

**else {**

**r = mid - 1;**

**}**

**}**

**ans += cnt + 1;**

**}**

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

**if (s[i - 1] == s[i - 2]) {**

**int l = 0, r = min(n - i, i - 1), cnt = 2;**

**while (l <= r) {**

**int mid = (l + r) >> 1;**

**if (ok(i - 1 - mid, i + mid)) {**

**cnt = mid;**

**l = mid + 1;**

**}**

**else {**

**r = mid - 1;**

**}**

**}**

**ans += cnt + 1;**

**}**

**}**

**cout << ans << '\n';**

**return 0;**

**}**

\*\* Dynamic Programming

**FAYSAL AHAMMED**

**Knapsack-2 (constraints- n <= 100, w <= 10^9, value <= 1000):**

**int max\_val = 0;**

**for (int i = n + 1; i >= 1; i--) {**

**for (int current\_value = 0; current\_value <= n \* 1000; current\_value++) {**

**if (i == n + 1) {**

**if (current\_value == 0) dp[i][current\_value] = 0;**

**else dp[i][current\_value] = inf;**

**}**

**else {**

**ll &ans = dp[i][current\_value];**

**ans = dp[i + 1][current\_value];**

**if (current\_value - value[i] >= 0) {**

**ans = min(ans, weight[i] + dp[i + 1][current\_value - value[i]]);**

**}**

**if (ans <= w) {**

**max\_val = max(max\_val, current\_value);**

**}}}}**

**cout << max\_val << '\n';**

**LCS:**

**int fun(int i, int j) {**

**if (i >= n or j >= m) return 0;**

**int &ans = dp[i][j];**

**if (ans != -1) return ans;**

**ans = fun(i + 1, j);**

**ans = max(ans, fun(i, j + 1));**

**if (s1[i] == s2[j]) {**

**ans = max(ans, 1 + fun(i + 1, j + 1));**

**}**

**return ans;**

**}**

**void print(int i, int j) {**

**if (i >= n or j >= m) return;**

**if (s1[i] == s2[j]) {**

**cout << s1[i];**

**path(i + 1, j + 1);**

**return;**

**}**

**int ans1 = fun(i + 1, j);**

**int ans2 = fun(i, j + 1);**

**if (ans1 > ans2) {**

**path(i + 1, j);**

**}**

**else {**

**path(i, j + 1);**

**}**

**}**

**LIS:**

**st.build(1, 1, M);**

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

**dp[i] = 1;**

**if (a[i] != 1) {**

**int mx = st.query(1, 1, M, 1, a[i] - 1);**

**mx++;**

**dp[i] = max(dp[i], mx);**

**}**

**st.upd(1, 1, M, a[i], dp[i]);**

**}**

**int ans = 0;**

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

**ans = max(ans, dp[i]);**

**}**

**cout << ans << '\n';**

**Edit Distance (make two string same by add, remove, replace any index with min cost):**

**int fun(int i, int j) {**

**if (i == n) {**

**if (j == m) return 0;**

**return m - j;**

**}**

**if (j == m) return n - i;**

**int &ans = dp[i][j];**

**if (ans != -1) return ans;**

**ans = inf;**

**if (s1[i] == s2[j]) {**

**ans = min(ans, fun(i + 1, j + 1));**

**}**

**ans = min(ans, 1 + fun(i + 1, j + 1));**

**ans = min(ans, 1 + fun(i + 1, j));**

**ans = min(ans, 1 + fun(i, j + 1));**

**return ans;**

**}**

**Longest Palindromic Subsequence:**

**int fun(int i, int j) {**

**if (i > j) return 0;**

**if (i == j) return 1;**

**int &ans = dp[i][j];**

**if (ans != -1) return ans;**

**ans = 0;**

**if (s[i] == s[j]) {**

**ans = max(ans, 2 + fun(i + 1, j - 1));**

**}**

**else {**

**ans = max(ans, fun(i + 1, j));**

**ans = max(ans, fun(i, j - 1));**

**}**

**return ans;**

**}  
  
Coin Change:  
Repetition allowed (or max k times for k)  
Repetition not allowed - fun(int i, int current\_amount)  
Repetition fixed - loop inside fun()  
Order doesn’t matter - fun(amount) and loop inside fun()  
Project (start time, ending time, profit are given, what's the max profit?):  
-Sort by endtime, then dp[i] = max profit if i is the last index taken. Like LIS. Code-  
sort(a + 1, a + 1 + n);  
st.build(1, 1, M);**

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

**dp[i] = a[i][2];**

**ll mx = st.query(1, 1, M, 1, a[i][1] - 1);**

**mx += a[i][2];**

**dp[i] = max(dp[i], mx);**

**st.upd(1, 1, M, a[i][0], dp[i]);**

**}**

**ll ans = 0;**

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

**ans = max(ans, dp[i]);**

**}**

**cout << ans << '\n';**

**Slime (n numbers are given, everytime merge 2 consecutive element until there is just 1 element such that cost is min, the cost of merge two element = sum of them):**

**-   Think reverse: we are given the final sum, from i to j. Now we will cut any point between i to j and calculate the cost**

**ll fun(int i, int j) {**

**if (i == j) return 0;**

**ll &ans = dp[i][j];**

**if (ans != -1) return ans;**

**ll cur = 0;**

**for (int x = i; x <= j; x++) {**

**cur += a[x];**

**}**

**ans = inf;**

**for (int x = i; x < j; x++) {**

**ans = min(ans, cur + fun(i, x) + fun(x + 1, j));**

**}**

**return ans;**

**}**

**Sub-Palindromic Tree (Given a tree, each node has a character. Now tell us the maximum path which has longest palindromic subsequence):**

**const int N = 2005, inf = 1e9;**

**vector<int> g[N];**

**int n;**

**string s;**

**int nxt[N][N];**

**vector<int> vec;**

**int dp[N][N];**

**void dfs(int u, int p) {**

**vec.push\_back(u);**

**for (auto v: g[u]) {**

**if (v != p) {**

**dfs(v, u);**

**}}}**

**int fun(int u, int v) {**

**if (v == u) return 1;**

**int &ans = dp[u][v];**

**if (ans != -1) return ans;**

**ans = 0;**

**if (s[u] == s[v]) {**

**ans = 2 + (nxt[u][v] == v ? 0 : fun(nxt[u][v], nxt[v][u]));}**

**else {**

**ans = max(fun(nxt[u][v], v), fun(u, nxt[v][u]));**

**}**

**return ans;**

**}**

**void solve() {**

**cin >> n >> s;**

**s = '.' + s;**

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

**int u, v; cin >> u >> v;**

**g[u].push\_back(v);**

**g[v].push\_back(u);**

**}**

**for (int u = 1; u <= n; u++) {**

**for (auto x : g[u]) {**

**vec.clear();**

**dfs(x, u);**

**for (auto v : vec) {**

**nxt[u][v] = x;**

**}}}**

**for (int u = 1; u <= n; u++) {**

**for (int v = 1; v <= n; v++) {**

**dp[u][v] = -1;**

**}}**

**int ans = 0;**

**for (int u = 1; u <= n; u++) {**

**for (int v = 1; v <= n; v++) {**

**ans = max(ans, fun(u, v));**

**}}**

**cout << ans << '\n';**

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

**g[i].clear();**

**}}**

**Dice-1 (choose N numbers from 1 to k, how many ways the sum of them is S?) [N, K <= 1000, S <= 15000]:**

**for (int i = n + 1; i >= 1; i--) {**

**for (int cur\_sum = 0; cur\_sum <= s; cur\_sum++) {**

**if (i == n + 1) {**

**dp[0][cur\_sum] = (cur\_sum == 0);**

**}**

**else {**

**int &ans = dp[0][cur\_sum];**

**int mn = min(k, cur\_sum);**

**ans = 0;**

**// for (int x = 1; x <= mn; x++) {**

**//   ans += dp[1][cur\_sum - x];**

**//   ans %= mod;**

**// }**

**ans = (dp[1][cur\_sum - 1] - (cur\_sum - mn - 1 < 0 ? 0 : dp[1][cur\_sum - mn - 1]) + mod) % mod;**

**}}**

**for (int cur\_sum = 0; cur\_sum <= s; cur\_sum++) {**

**dp[1][cur\_sum] = dp[0][cur\_sum];**

**}**

**for (int cur\_sum = 1; i != 1 and cur\_sum <= s; cur\_sum++) {**

**dp[1][cur\_sum] += dp[1][cur\_sum - 1];**

**dp[1][cur\_sum] %= mod;**

**}}**

**cout << dp[0][s] << '\n';**

**Dice-2 (choose N numbers from 1 to k,if sum of them = s, then score = multiply of the n numbers, sum of scores?) [N, K <= 1000, S <= 15000]:**

**for (int i = n + 1; i >= 1; i--) {**

**for (int cur\_sum = 0; cur\_sum <= s; cur\_sum++) {**

**if (i == n + 1) {**

**dp[0][cur\_sum] = cur\_sum == 0;**

**}**

**else {**

**int &ans = dp[0][cur\_sum];**

**ans = 0;**

**int mn = min(k, cur\_sum);**

**// for (int x = 1; x <= mn; x++) {**

**//   ans += 1ll \* x \* dp[1][cur\_sum - x] % mod;**

**//   ans %= mod;**

**// }**

**int l = cur\_sum - mn, r = cur\_sum - 1;**

**ans = (pref[l] - pref[r + 1] + mod) % mod;**

**ans -= 1ll \* (s - r) \* ((dp[1][r] - (l ? dp[1][l - 1] : 0) + mod) % mod) % mod;**

**ans += mod;**

**ans %= mod;**

**}}**

**for (int cur\_sum = 0; cur\_sum <= s; cur\_sum++) {**

**dp[1][cur\_sum] = dp[0][cur\_sum];**

**}**

**pref[0] = dp[1][0];**

**for (int cur\_sum = 1; i != 1 and cur\_sum <= s; cur\_sum++) {**

**pref[cur\_sum] = dp[1][cur\_sum];**

**dp[1][cur\_sum] += dp[1][cur\_sum - 1];**

**dp[1][cur\_sum] %= mod;**

**}**

**for (int cur\_sum = s - 1, j = 2; i != 1 and cur\_sum >= 0; cur\_sum--, j++) {**

**pref[cur\_sum] = (1ll \* pref[cur\_sum] \* j % mod) + pref[cur\_sum + 1];**

**pref[cur\_sum] %= mod;**

**}}**

**cout << dp[0][s] << '\n';**

**\* If dp stores only true/false use bitset.**

\*\* Micellaneous

**FAYSAL AHAMMED**

**Ternary Search (Pyramid):**

**double suface\_area;**

**double fun(double square\_area) {**

**double base = sqrt(square\_area);**

**double triangle\_area = suface\_area - square\_area;**

**double per\_triangle\_area = triangle\_area / 4;**

**double triangle\_height = (per\_triangle\_area \* 2) / base;**

**double x = base / 2;**

**double height = sqrt((triangle\_height \* triangle\_height) - (x \* x));**

**double volume = (base \* base \* height) / 3;**

**if (x > triangle\_height) volume = 0;**

**return volume;**

**}**

**int cs = 0;**

**int32\_t main() {**

**ios\_base::sync\_with\_stdio(0);**

**cin.tie(0);**

**int t; cin >> t;**

**while (t--) {**

**cin >> suface\_area;**

**cout << fixed << setprecision(4);**

**double l = 0, r = suface\_area, ans = -1;**

**int it = 100;**

**while (it--) {**

**double mid1 = l + (r - l) / 3;**

**double mid2 = r - (r - l) / 3;**

**double x = fun(mid1);**

**double y = fun(mid2);**

**if (x > y) {**

**ans = x;**

**r = mid2;**

**}**

**else {**

**l = mid1;**

**}}**

**cout << "Case " << ++cs << ": ";**

**cout << ans << '\n';**

**}**

**return 0;**

**}**

**2D Prefix Sum:**

**int n, m; cin >> n >> m;**

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

**for (int j = 1; j <= m; j++) {**

**cin >> a[i][j];**

**}}**

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

**for (int j = 1; j <= m; j++) {**

**prefix[i][j] = prefix[i - 1][j] + prefix[i][j - 1] - prefix[i - 1][j - 1] + a[i][j];**

**}}**

**int q; cin >> q;**

**while (q--) {**

**int x1, y1, x2, y2; cin >> x1 >> y1 >> x2 >> y2;**

**cout << prefix[x2][y2] - prefix[x1 - 1][y2] - prefix[x2][y1 - 1] + prefix[x1 - 1][y1 - 1] << '\n';**

**}**

**2D Difference Array:**

**int n, m; cin >> n >> m;**

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

**for (int j = 1; j <= m; j++) {**

**char c; cin >> c;**

**a[i][j] = c - '0';**

**}}**

**int q; cin >> q;**

**while (q--) {**

**int x1, y1, x2, y2, x; cin >> x1 >> y1 >> x2 >> y2;**

**x = 1;**

**d[x1][y1] += x;**

**d[x1][y2 + 1] -= x;**

**d[x2 + 1][y1] -= x;**

**d[x2 + 1][y2 + 1] += x;**

**}**

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

**for (int j = 1; j <= m; j++) {**

**d[i][j] += d[i - 1][j] + d[i][j - 1] - d[i - 1][j - 1];**

**}}**

**// new updated array**

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

**for (int j = 1; j <= m; j++) {**

**cout << (d[i][j] + a[i][j]) % 2;**

**}**

**cout << '\n';**

**}**

**PBDS:**

**#include<ext/pb\_ds/assoc\_container.hpp>**

**#include<ext/pb\_ds/tree\_policy.hpp>**

**using namespace \_\_gnu\_pbds;**

**template <typename T> using o\_set = tree<T, null\_type, less<T>, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;**

**Custom Hash for GP HashTable:**

**const int RANDOM = chrono::high\_resolution\_clock::now().time\_since\_epoch().count();**

**struct chash {**

**int operator()(int x) const { return x ^ RANDOM; }**

**};**

**Direction Array:**

**int dx[] = {+0, +0, +1, -1, -1, +1, -1, +1};**

**int dy[] = {-1, +1, +0, +0, +1, +1, -1, -1};**

**Custom Comparator:**

**bool cmp(pair<int, int> a, pair<int, int> b) {**

**if (a.first != b.first) return a.first > b.first;**

**return a.second < b.second; // comparator function must return false for equal elements**

**}**

**Custom Comparator for map,set, multiset, pq:**

**struct cmp {**

**bool operator()(const int& a, const int& b) const {**

**return a > b;**

**}**

**};**

**Mex with Array Updates:**

**int n, q; cin >> n >> q;**

**set<int> missing\_numbers;**

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

**missing\_numbers.insert(i);**

**}**

**int a[n + 1];**

**map<int, int> freq;**

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

**cin >> a[i];**

**freq[a[i]]++;**

**missing\_numbers.erase(a[i]);**

**}**

**while (q--) {**

**int i, x; cin >> i >> x;**

**int y = a[i];**

**a[i] = x;**

**missing\_numbers.erase(x);**

**freq[y]--;**

**freq[x]++;**

**if (freq[y] == 0) {**

**freq.erase(y);**

**missing\_numbers.insert(y);**

**}**

**cout << \*missing\_numbers.begin() << '\n';**

**}**

**Coordinate Compression (faster):**

**vector<int> a({100, 9, 10, 10, 9});**

**vector<int> v = a;**

**sort(v.begin(), v.end());**

**v.resize(unique(v.begin(), v.end()) - v.begin());**

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

**a[i] = lower\_bound(v.begin(), v.end(), a[i]) - v.begin() + 1;**

**cout << a[i] << ' ';**

**}**

**Pigeonhole Principle:**

**-At least 1 subarray of an array of length N must be divisible by N.**

**-Build all possible sequences of length 10 whose value is between 1 to 100. At least any two sequences will be same.**

**\* Given an array of length N (N <= 10^6) and M (M <= 10^3) check if there is any subsequence of the array whose sum is divisible by k?**

**According to the pigeonhole principle if N >= M then it must be “YES”. Else we can do DP. where N < M <= 1000.**

**Contribution Technique (Calculate the contribution of each element separately):**

**\* Sum of pair sums (i=1 to n Σ j= 1 to n Σ(ai+a):**

**=> Every element will be added 2n times.**

****

**\* Sum of subarray sums:**

****

**\* Sum of subset sums:**

****

**\* Product of pair product:**

****

**\* XOR of subarray XORS:**

**=> How many subarrays does an element have? (i\* (n-i+1) times. If subarray length is odd then this element can contribute in total XORs.**

**\* Sum of max-min over all subset:**

**=> Sort the array. Min = 2^(n-i), Max = 2^(i-1)**

**i=1 to n Σ(ai \* 2^(i-1))-(ai\*2^(n-i))**

**\* Sum using Bit:**

****

**\* Sum of Pair XORs:**

**=> XOR = 1 if two bits are different**

**A black text on a white background

Description automatically generated**

**\* Sum of pair ANDs:**

****

**\* Sum of pair ORs:**

****

**\* Sum of Subset XORs:**

***[where cnt0 != 0)***

**\* Sum of Subset ANDs:**

****

**\* Sub of Subset ORs:**

****

**\* Sum of subarray XORs:**

**=> Convert to prefix xor, then solve for pairs.**

**Nafis and MEX:**

**void solve() {**

**int n, k; cin >> n >> k;**

**vector<int> a(n);**

**map<int, int> mp;**

**for (auto &x : a) {**

**cin >> x;**

**mp[x]++;**

**}**

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

**int max\_mex = 0;**

**for (auto x : a) {**

**if (max\_mex == x) max\_mex++;**

**}**

**int ways[max\_mex];**

**for (int i = 0; i < max\_mex; i++) {**

**int mn = min(30ll, mp[i]);**

**ways[i] = (1 << mn) - 1;**

**}**

**for (int i = 1; i < max\_mex; i++) {**

**int mn = (int)(1ll \* ways[i] \* ways[i - 1]);**

**ways[i] = min(MIN, mn);**

**}**

**int cur = max\_mex;**

**int add = (k + 1) / 2;**

**int minus = k - add;**

**long long ans = 0;**

**while (minus > 0) {**

**int right = n - (upper\_bound(a.begin(), a.end(), cur) - a.begin());**

**int mn = min(30ll, right);**

**int possible = (1 << mn);**

**if (cur == 0) {**

**possible--;**

**int apply = min(minus, possible);**

**minus -= apply;**

**ans -= (1ll \* cur \* apply);**

**}**

**else {**

**possible = min(MIN, (int) (1ll \* ways[cur - 1] \* possible));**

**int apply = min(minus, possible);**

**minus -= apply;**

**ans -= (1ll \* cur \* apply);**

**}**

**cur--;**

**}**

**cur = 0;**

**while (add > 0) {**

**int right = n - (upper\_bound(a.begin(), a.end(), cur) - a.begin());**

**int mn = min(30ll, right);**

**int possible = (1 << mn);**

**if (cur == 0) {**

**possible--;**

**int apply = min(add, possible);**

**add -= apply;**

**ans += (1ll \* cur \* apply);**

**}**

**else {**

**possible = min(MIN, (int) (1ll \* ways[cur - 1] \* possible));**

**int apply = min(add, possible);**

**add -= apply;**

**ans += (1ll \* cur \* apply);**

**}**

**cur++;**

**}**

**cout << ans << '\n';**

**}**

**How many triplets such that ai\*bj\*ck=m:**

**A close up of a paper

Description automatically generated**

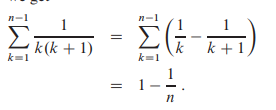
**Formulas:**

**Sum of squares: 1^2 + 2^2 + 3^2 + … + n^2 = n(n+1) (n+2)/6**

**Sum of cubes: 1^3 + 2^3 + 3^3+ ... + n^3 = (n^2 \* (n+1)^2)/4**

**Geometric Series: 1+x + x^2+x^3...+x^n = ( x^(n+1)-1)/(x-1)  
when |x|<1 then the sum = 1 / (1-x)**

**Harmonic Series: 1 + ½ + 1/3 + ¼ + … + 1/n = ln (n) + O(1)  
 >** A math equation with numbers and symbols

Description automatically generated **>**  **>**

**  
**

****

****

****

****

****

**  
**

****

****

****

****

****

****

****

****

****

****

****

****

****

****

****

A mathematical equation with black lines

Description automatically generated

A black and white math equation

Description automatically generated

A mathematical equation with black text

Description automatically generated

**Geometric series:**

A mathematical equation with black lines

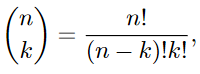
Description automatically generated

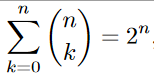
A mathematical equation with numbers and symbols

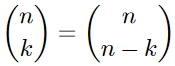
Description automatically generated

A black and white image of a mathematical equation

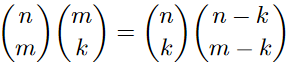
Description automatically generated  
**----------------------------------------**











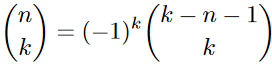
A close-up of a number

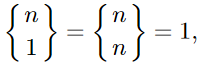
Description automatically generated

A close-up of a number

Description automatically generated



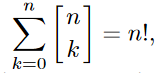


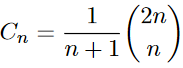


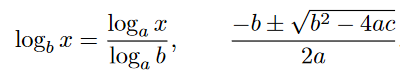
A math equation with numbers and symbols

Description automatically generated



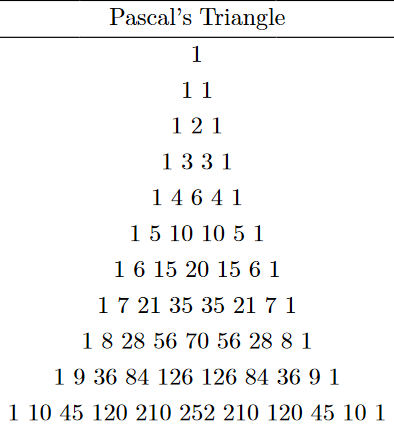


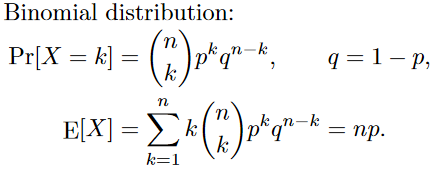




A math equations and numbers

Description automatically generated





A black and white image of a mathematical equation

Description automatically generatedA triangle with a square and a square with a square and a square with a square and a square with a square and a square with a square and a square with a square and a square with

Description automatically generatedA math equations on a white background

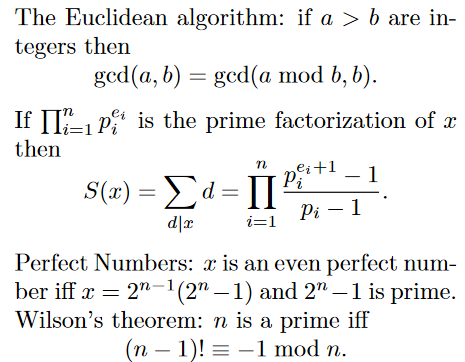
Description automatically generated A math equations and formulas

Description automatically generated  
A math equations and formulas

Description automatically generated

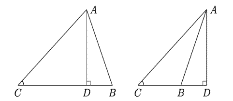
A white sheet with black text and numbers

Description automatically generated



A triangle with a square in the middle

Description automatically generated  


A triangle with letters and numbers

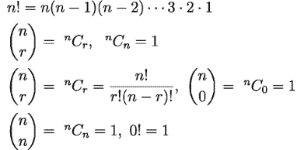
Description automatically generated  
  
A math problem with numbers and a plus

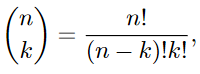
Description automatically generated  
  

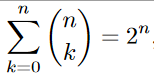


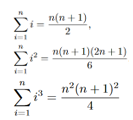




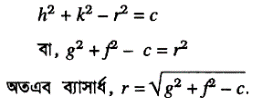






Triangle:  
area =   
area =

Cube:  
area = abc  
diag = 

Circle:  
circumference :   
area :   
eqn :   
  


Cone:  
volume: A black text on a white background

Description automatically generated

A diagram of a triangle with a cross

Description automatically generated  
area = 

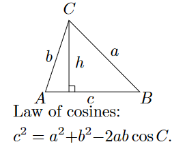


A close-up of a sign

Description automatically generated

  
tangent condition: 


A math equations on a white background

Description automatically generated

A white sheet with black text and numbers

Description automatically generated