**PRECODE**

**##### Techniques #####**

**1. Contribution Technique**

**2. Binary Search on ans**

**3. Binary Search on other thing**

**4. Ternary Search**

**5. Number Theory**

**6. DP**

**7. Segment Tree**

**8. PBDS**

**9. Set/map**

**10. Sieve or Backward Sieve**

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

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

**using namespace std;**

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

**#define TIMER class Timer { private: chrono::time\_point <chrono::steady\_clock> Begin, End; public: Timer () : Begin(), End (){ Begin = chrono::steady\_clock::now(); } ~Timer () { End = chrono::steady\_clock::now();cerr << "\nDuration: " << ((chrono::duration <double>)(End - Begin)).count() << "s\n"; } } T;**

**#define int long long**

**#define ll unsigned long long**

**#define uset tree<int, null\_type, less\_equal<int>, rb\_tree\_tag, tree\_order\_statistics\_node\_update >**

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

**#define Fo(i,k,n) for(int i=k;k<n?i<n:i>n;k<n?i+=1:i-=1)**

**#define vi vector<int>**

**#define vii vector<pair<int,int>>**

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

**#define pb push\_back**

**#define pf push\_front**

**#define F first**

**#define S second**

**#define clr(x,y) memset(x, y, sizeof(x))**

**#define deb(x) cout << #x << "=" << x << endl**

**#define deb2(x, y) cout << #x << "=" << x << "," << #y << "=" << y << endl**

**#define s(x)   x.size()**

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

**#define allg(x) x.begin(),x.end(),greater<int>()**

**#define BOOST ios\_base::sync\_with\_stdio(false);cin.tie(NULL);cout.tie(NULL);**

**#define endl "\n"**

**#define bitOne(x) \_\_builtin\_popcount(x)**

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

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

**const int MOD=1000000007;**

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

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

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

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

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

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

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

**mt19937\_64 rang(chrono::high\_resolution\_clock::now().time\_since\_epoch().count());**

**int rng(int lim) {**

**uniform\_int\_distribution<int> uid(-1000,-1);**

**return uid(rang);**

**}**

***Kth bit on or off***

**bool checkBit(int n, int k){ if (n & (1 << k)) return true; else return false; }**

***GCD***

**int gcd(int a, int b) *// O(logN)***

**{**

**if(!b) return a;**

**return gcd(b,a%b);**

**}**

***Directional Array***

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

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

***precalculate factorial***

**int fact[N];**

**void preFact(){**

**fact[0] = 1;**

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

**fact[i] = (1LL\*fact[i-1]\*i)%mod;**

**if(fact[i] < 0) fact[i] += mod;**

**}}**

***ncr mod***

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

**int denom = (fact[n-r] \* fact[r] \* 1LL)%mod;**

**int res = (1LL \* fact[n] \* inverse(denom))%mod;**

**if(res < 0) res += mod;**

**return res%mod;**

**}**

NUMBER THEORY

**# SIEVE OF ERATOSTHENES**

***// TC: O(n\*log(log(n)))***

**const int MX = 1e7+123;**

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

**vector<int> prime;**

**void primeGen ( int n ){**

**n += 100;**

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

**int sq = sqrt ( n );**

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

**if ( is\_prime[i] == 1 ) {**

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

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

**}}}**

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

**prime.push\_back (2);**

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

**if ( is\_prime[i] == 1 ) prime.push\_back ( i );**

**}}**

**# SMALLEST PRIME FACTOR S**

***// TC: O(n\*log(n))***

**const int N = 1e6;**

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

**void smallestPrimeFactor(int n)**

**{**

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

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

**{**

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

**{**

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

**{**

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

**{**

**spf[j] = i;**

**}}}}}**

**# NUMBER OF DIVISORS**

***// pre-requisite: primeGen(n)***

***// TC : O(sqrt(n))***

**int NOD(int n){**

**int ans = 1;**

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

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

**int cnt = 0;**

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

**n/=p;**

**cnt++;**

**}**

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

**}**

**if(n > 1) ans \*= 2;**

**return ans;**

**}**

**# SUM OF DIVISORS**

***// \*\* primeGen(n)***

***// TC: sqrt(n)***

**int SOD(int n)**

**{**

**int sum = 1;**

**for(auto p : prime)**

**{**

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

**if(n%p == 0)**

**{**

**int pn = p;**

**while(n%p == 0)**

**{**

**n/=p;**

**pn \*= p;**

**}**

**pn -= 1;**

**pn/=(p-1);**

**sum \*= pn;**

**}}**

**if(n > 1)**

**{**

**int pn = n\*n;**

**pn -= 1;**

**pn /= (n-1);**

**sum \*= pn;**

**}**

**return sum;**

**}**

**# SUM OF DIVISORS FROM 1 TO N**

***// TC: O(n)***

**int SODALL(int n)**

**{**

**int ans = 0;**

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

**{**

**ans += (n/i)\*i;**

**}**

**return ans;**

**}**

**# PRIME FACTORIZATION**

***// \*\* primeGen(n)***

***// TC : O(sqrt(n))***

**vector<int> primeFactorization(int n)**

**{**

**vector<int> pf;**

**for(auto x : prime)**

**{**

**if(x\*x > n) break;**

**while(n%x == 0)**

**{**

**n/=x;**

**pf.push\_back(x);**

**}**

**}**

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

**return pf;**

**}**

**# PHI USING DIV FORMULA**

**const int N = 1e6;**

**vector<int> phi(N);**

***// O(n\*log(n))***

**void phiDiv(int n){**

**phi[0] = phi[1] = 1;**

**for(int i = 2; i <= n; i++) phi[i] = i-1;**

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

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

**phi[j] -= phi[i];**

**}}}**

**# PHI USING SIEVE**

**const int N = 1e6;**

**vector<int> phi(N);**

***// O(n\*loglog(n))***

**void phiSieve(int n){**

**phi[0] = phi[1] = 1;**

**for(int i = 2; i <= n; i++) phi[i] = i;**

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

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

**phi[i]-=1;**

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

**phi[j] = (phi[j] \* (i-1));**

**phi[j] /= i;**

**}}}}**

**# Linear Sieve**

**const int N = 1e7;**

**vector<int> lp(N);**

**vector<int> primes;**

***// TC: (O(n)) // finds primes up to 1e7***

**void sieveLinear(int n){**

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

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

**lp[i] = i;**

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

**}**

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

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

**if(primes[j] == lp[i]) break;**

**}}}**

**# Divs of N**

**vector<int> divs(int n){**

**vector<int> v;**

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

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

**v.push\_back(i);**

**if(n/i != i){**

**v.push\_back(n/i);**

**}}}**

**return v;**

**}**

***/// Extended GCD***

***// O(logN), egcd(a,b).x = (1/a)%b , a & b must be coprime***

**struct triplet**

**{**

**int x;**

**int y;**

**int gcd;**

**};**

**triplet egcd(int a, int b)**

**{**

**if(b == 0)**

**{**

**triplet ans;**

**ans.x = 1; *// must be 1 in base case***

**ans.y = 1; *// y can be anything since y becomes 0 in gcd(x,y)***

**ans.gcd = a;**

**return ans;**

**}**

**triplet ans1 = egcd(b,a%b);**

**triplet ans;**

**ans.x = ans1.y; *// X is the multiplicative inverse of A under B***

**ans.y = ans1.x-(a/b)\*ans1.y;**

**ans.gcd = ans1.gcd;**

**return ans;**

**}**

**int ModInv(int a, int m)**

**{**

**auto ans = egcd(a,m);**

**return (ans.x%m + m)%m; *// to avoid neg value***

***// ans.gcd must be 1***

**}**

**Segmented Sieve**

**vector<char> segmentedSieve(long**

**long L,long long 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+i1)/i\*i);j<=R;j+=i)**

**isPrime[j-L]=false;**

**if (L==1) isPrime[0] = false;**

**return isPrime;}**

**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;}**

**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;}**

**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.**

**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) a is starting element, n is number of elements ,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**

**A math equations on a graph paper

Description automatically generated**

**DATA STRUCTURE**

***Segment Tree***

**const int MX = 1e5+10;**

**int arr[MX];**

**int Tree[MX\*4];**

**void init(int node, int b, int e) *// O(n) --> 2n nodes***

**{**

**if(b==e)**

**{**

**Tree[node] = arr[b];**

**return;**

**}**

**int Left = node\*2;**

**int Right = (node\*2)+1;**

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

**init(Left,b,mid);**

**init(Right,mid+1,e);**

**Tree[node] = Tree[Left] + Tree[Right];**

**}**

**int query(int node, int b, int e, int l, int r) *// O(4\*Log(n))***

**{**

**if(l > e || r < b) return 0;**

**if(l<=b && e<=r)**

**{**

**return Tree[node];**

**}**

**int Left = node\*2;**

**int Right = (node\*2)+1;**

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

**int leftTreeVal = query(Left,b,mid,l,r);**

**int rightTreeVal = query(Right,mid+1,e,l,r);**

**return leftTreeVal+rightTreeVal;**

**}**

**void update(int node, int b, int e, int i, int val) *// O(LogN)***

**{**

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

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

**{**

**Tree[node] = val;**

**return;**

**}**

**int Left = node\*2;**

**int Right = (node\*2)+1;**

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

**update(Left,b,mid,i,val);**

**update(Right,mid+1,e,i,val);**

**Tree[node] = Tree[Left] + Tree[Right];**

**}**

***Segment Tree with Lazy Propagation***

**const int mx = 1e5+10;**

**int arr[mx];**

**struct**

**{**

**int sum,prop;**

**}Tree[mx\*4];**

**void init(int node, int b, int e) *// O(NlogN)*{**

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

**Tree[node].sum = arr[b];**

**return;**

**}**

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

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

**int right = (node\*2)+1;**

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

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

**Tree[node].sum = Tree[left].sum+Tree[right].sum;**

**}**

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

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

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

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

**int right = left+1;**

**Tree[left].sum += (mid-b+1)\*Tree[node].prop;**

**Tree[right].sum += (e-mid)\*Tree[node].prop;**

**Tree[left].prop += Tree[node].prop;**

**Tree[right].prop += Tree[node].prop;**

**}**

**Tree[node].prop = 0;**

**}**

**void update(int node,int b, int e, int l, int r, int val) *// O(4\*logN)*{**

**if(Tree[node].prop != 0){**

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

**}**

**if(l > e || r < b) return;**

**if(l <= b && r >= e)**

**{**

**Tree[node].sum += (val\*(e-b+1));**

**Tree[node].prop += val;**

**if(Tree[node].prop != 0)**

**{**

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

**}**

**return;**

**}**

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

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

**int right = (node\*2)+1;**

**update(left,b,mid,l,r,val);**

**update(right,mid+1,e,l,r,val);**

**Tree[node].sum = Tree[left].sum+Tree[right].sum;**

**}**

**int query(int node,int b,int e,int l,int r) *// O(4\*logN)***

**{**

**if(Tree[node].prop != 0)**

**{**

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

**}**

**if(l > e || r < b) return 0;**

**if(l <= b && r >= e)**

**{**

**return Tree[node].sum;**

**}**

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

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

**int right = (node\*2)+1;**

**int val1 = query(left,b,mid,l,r);**

**int val2 = query(right,mid+1,e,l,r);**

**return val1 + val2;**

**}**

***// Fenwick Tree***

**vector<int> Bit1; *// assign O(n) space for n elements***

**vector<int> Bit2; *// assign O(n) space for n elements***

**void Update(vector<int>& Bit, int idx, int val) *// O(logN) -> single time update korte logN time lage...taile N ta items er jonne O(NlogN) time lagbe***

**{**

**int N = Bit.size();**

**for(idx; idx<N; idx+=(idx&-idx))**

**{**

**Bit[idx]+=val;**

**}**

**}**

**int Sum(vector<int>& Bit, int idx) *// O(logN)***

**{**

**int sum = 0;**

**for(idx; idx>0; idx-=(idx&-idx))**

**{**

**sum += Bit[idx];**

**}**

**return sum;**

**}**

**void RangeUpdate(int l, int r, int val) *// O(4\*logN)***

**{**

**Update(Bit1,l,val);**

**Update(Bit1,r+1,-val);**

**Update(Bit2,l,val\*(l-1));**

**Update(Bit2,r+1,-val\*r);**

**}**

**int PrefixSum(int idx)**

**{**

**return Sum(Bit1,idx)\*idx - Sum(Bit2,idx);**

**}**

**int RangeSum(int l,int r)**

**{**

**return PrefixSum(r)-PrefixSum(l-1);**

**}**

**GRAPH**

***//DFS cycle detection***

**bool dfsCycle(int vertex,int parent){**

**bool a = false;**

**vis[vertex] = true;**

**for(auto child : adj[vertex]){**

**if(child != parent && vis[child]){**

**return true;**

**}else if(vis[child] == false){**

**a = dfsCycle(child,vertex);**

**}}**

**return a;**

**}**

**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;**

**}**

***//Kruskals Algirithm***

**#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;**

**}**

**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.**

***// Dijkstra***

**const int N = 1e5+100;**

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

**int wt[N];**

**void dijkstra(int source, int nodes) *// TC : O(E + Vlog2(V))***

**{**

**for(int i = 0; i < nodes; i++) wt[i] = 1e18;**

**wt[source] = 0;**

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

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

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

**{**

**int curV = pq.top().S;**

**int curVW = pq.top().F;**

**pq.pop();**

**if(curVW > wt[curV]) continue;**

**for(auto child : adj[curV])**

**{**

**int childV = child.F;**

**int childVW = child.S;**

**if(curVW + childVW < wt[childV])**

**{**

**wt[childV] = curVW + childVW;**

**pq.push({wt[childV],childV});**

**}}}}**

***// BFS***

**const int N = 1e3;**

**bool vis[N];**

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

**void bfs(int source)**

**{**

**vis[source] = true;**

**queue<int> q;**

**q.push(source);**

**while(q.empty() == false)**

**{**

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

**q.pop();**

**for(auto child : adj[curVertex])**

**{**

**if(vis[child]) continue;**

**q.push(child);**

**vis[child] = true;**

**}}}**

**STRING**

**\*\*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;**

**}**

***// Hashing***

**const int MAXN=1000006;**

**namespace DoubleHash{**

**int P[2][MAXN];**

**int H[2][MAXN];**

**int R[2][MAXN];**

**int base[2];**

**int 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 int range\_hash(int l,int r,int idx){**

**int 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 int reverse\_hash(int l,int r,int idx){**

**int 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 int range\_dhash(int l,int r){**

**int x = range\_hash(l,r,0);**

**return (x<<32)^range\_hash(l,r,1);**

**}**

**inline int reverse\_dhash(int l,int r){**

**int x = reverse\_hash(l,r,0);**

**return (x<<32)^reverse\_hash(l,r,1);**

**}**

**}**

**char str1[MAXN];**

**using namespace DoubleHash;**

**TECHNIQUE**

***// Next Greater Prev Greater***

**vector<pair<int,int>> ng(n,{-1,-1});**

**stack<int> stk;**

***// leff***

**for(int i = n-1; i >= 0; i--)**

**{**

**if(stk.empty())**

**{**

**stk.push(i);**

**}else**

**{**

**while(stk.size() && v[i] > v[stk.top()])**

**{**

**ng[stk.top()].first = i;**

**stk.pop();**

**}**

**stk.push(i);**

**}**

**}**

**while(stk.empty() == false) stk.pop();**

***// right***

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

**{**

**if(stk.empty())**

**{**

**stk.push(i);**

**}else**

**{**

**while(stk.size() && v[i] > v[stk.top()])**

**{**

**ng[stk.top()].second = i;**

**stk.pop();**

**}**

**stk.push(i);**

**}**

**}**

***// Ternary Search***

**double ternary\_search(double l, double r) {**

**double eps = 1e-9;**

**while (r - l > eps) {**

**double m1 = l + (r - l) / 3;**

**double m2 = r - (r - l) / 3;**

**double f1 = f(m1);**

**double f2 = f(m2);**

**if (f1 < f2)**

**l = m1;**

**else**

**r = m2;**

**}**

**return f(l);**

**}**

**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';**

**}**

**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.**

**MICELLANEOUS**

**A math equations on a graph paper

Description automatically generated**

**A person writing on a whiteboard

Description automatically generated**

**A graph paper with a diagram and writing

Description automatically generated**

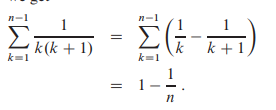
**Formulas:**

**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

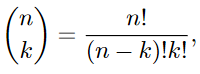
Description automatically generated**

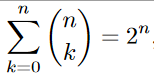
**A mathematical equation with numbers and symbols

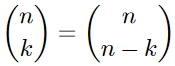
Description automatically generated**

**A black and white image of a mathematical equation

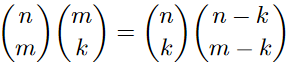
Description automatically generated  
--------------------------------------**

****

****

****

****

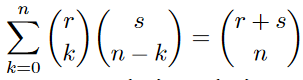
****

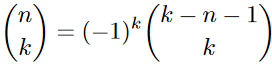
**A close-up of a number

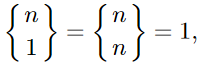
Description automatically generated**

**A close-up of a number

Description automatically generated**

****

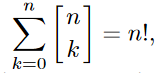
****

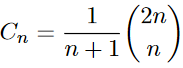
****

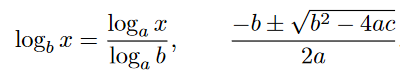
**A math equation with numbers and symbols

Description automatically generated**

****

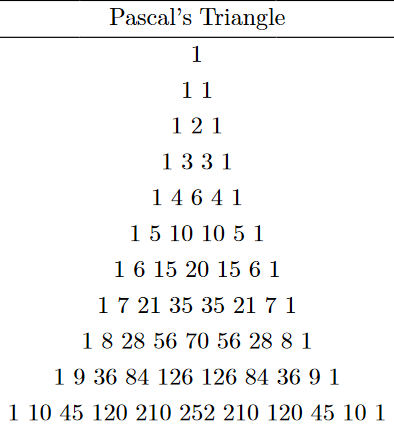
****

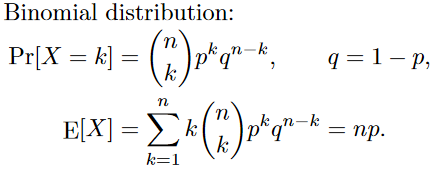
****

****

**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

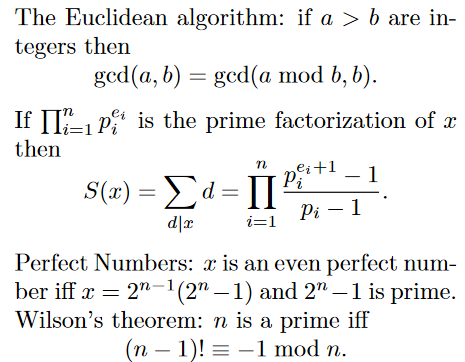
Description automatically generated A math equations and formulas

Description automatically generated  
A math equations and formulas

Description automatically generated**

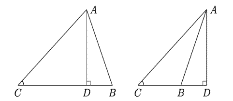
**A white sheet with black text and numbers

Description automatically generated**

****

**A triangle with a square in the middle

Description automatically generated  
**

**  
**

**A triangle with letters and numbers

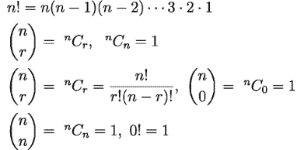
Description automatically generated  
  
A math problem with numbers and a plus

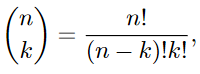
Description automatically generated  
  
**

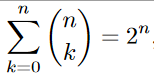
****

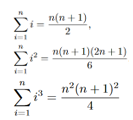
****

****

****

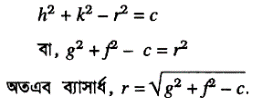
****

****

****

**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 = **

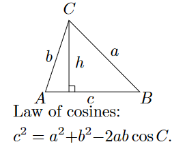
****

**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**