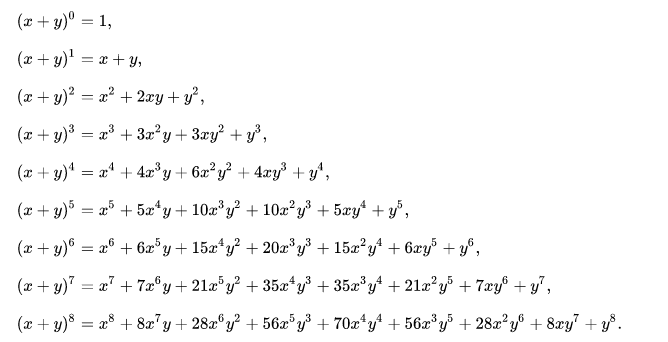
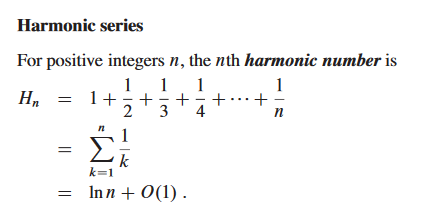
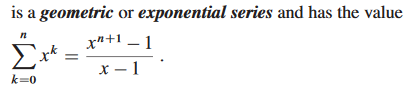
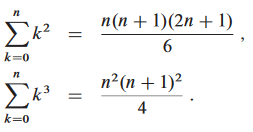
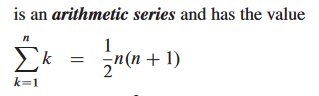
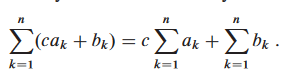
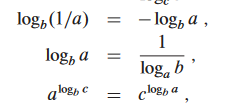
**\*\* SSC \*\***

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

|  |
| --- |
| **\*\* 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;**  **}**  **—----------------------------------------------------------**  **\*\* Binomial Theorem And others\*\*** |

****

****

**Segtree Merge function for most frequent value in range:**

|  |
| --- |
| **node merge(node l, node r) {**  **node ans;**  **ans.max\_val = max(l.max\_val, r.max\_val);**  **if(l.right == r.left) {**  **int cnt = l.right\_cnt + r.left\_cnt;**  **ans.max\_val = max(ans.max\_val, cnt);**  **}**  **ans.left = l.left;**  **ans.left\_cnt = l.left\_cnt;**  **if(l.left == r.left) {**  **ans.left\_cnt += r.left\_cnt;**  **}**  **ans.right = r.right;**  **ans.right\_cnt = r.right\_cnt;**  **if(r.right == l.right) {**  **ans.right\_cnt += l.right\_cnt;**  **}**  **return ans;**  **}**  **SOD: (p1^(e1 + 1) - 2) / (p1 - 1) \* (p2^(e2 + 1) - 2) / (p2 - 1) \* ……..** |

**Legendres 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)**

|  |
| --- |
| **Euler Sieve:**  **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);**  **}**  **}**  **}**  **}** |

**Euler Phi: Phi(n) = n \* ((p1 - 1) / p1) \* ((p2 - 1) / p2) \* …..**

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

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

**Arithmetic progression:**

**Sum of first n elements, 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**

**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);**  **}** |
|  |

**2D prefix sum:**

|  |
| --- |
| **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 static range update:**

|  |
| --- |
| **while (q--) {**  **int x1, y1, x2, y2, x; cin >> x1 >> y1 >> x2 >> y2 >> x;**  **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];**  **}**  **}** |

**Tree Diameter: max cost 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.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*Graph Kruskal using DSU\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**/// Kruskals start**

**#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;**

**}**

**// Kruskals end**