**#Dijkstra**

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

**const int INF = 1e9;**

**vector<pair<int,int>>ad\_list[N];**

**vi dis(N,INF);**

**bool vis[N];**

**priority\_queue<pi,vpi,greater<pi>> q;**

**void dijkstra(int source)**

**{**

**dis[source] = 0;**

**q.push(mp(0,source));//jehetu amader minimum wt er node ta ke age process korte hobe tai q te**

**always 1st ta wt ,2nd ta node push hobe.tahole wt er upor sorted thakbe.**

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

**{**

**pi top = q.top();**

**int u = top.second;**

**q.pop();**

**if(vis[u])**

**continue;**

**vis[u] = 1;**

**for(auto &child: ad\_list[u])**

**{**

**int v = child.first;//v node ta....u theke jeita te jacche**

**int wt = child.second;//u theke v te jauar cost**

**if(dis[u] + wt < dis[v])**

**{**

**dis[v] = dis[u]+wt;**

**q.push(mp(dis[v],v));**

**}**

**}**

**}**

**}**

**#Custom sort priority queue**

**struct Person {**

**int age;**

**float height;**

**Person(int age, float height): age(age), height(height) {}**

**};**

**struct CompareHeight {**

**bool operator()(Person**

**const & p1, Person**

**const & p2) {**

**return p1.height < p2.height;**

**}**

**};**

**int main() {**

**priority\_queue < Person, vector < Person > , CompareHeight > Q;**

**}**

**#spf**

**#define ll long long int**

**const ll N = 1e7 + 9;**

**int spf[N];**

**void function\_for\_spf()//Time Complexity of this function is O(N\*log(log(N))).**

**{**

**for (ll i = 2; i < N; i++)**

**spf[i] = i;**

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

**{**

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

**{**

**for (ll j = i+i; j < N; j += i)**

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

**spf[j] = i;**

**}**

**}**

**}**

**#Mos Algo**

**/\***

**MO's algo**

**handles offline query in O(q/sqrt(n)) usually**

**maintain proper block size ~ sqrt(n)**

**careful with < in query**

**query indices and array indices are presumed to be 0-indexed**

**\*/**

**const int block\_sz = 550; // ~sqrt(n)**

**const int N = 2e6+5;//max size of the array,max query 1e5**

**int freq[N],mo\_cnt,mo\_sum;**

**int ans[N];**

**int a[N];**

**void add(int idx)**

**{**

**// if one element is added what happens?**

**++freq[a[idx]];**

**if(freq[a[idx]] == 1)**

**mo\_cnt++;//change here**

**//mo\_sum += a[idx] hoto jodi l to r sum chaito**

**}**

**void erase(int idx)**

**{**

**// if one element is removed what happens?**

**--freq[a[idx]];**

**if(freq[a[idx]] == 0)**

**mo\_cnt--;//change here**

**//mo\_sum -= a[idx] hoto jodi l to r sum chaito**

**}**

**struct query{**

**int l,r,idx;**

**query() { }**

**query(int \_l,int \_r,int \_i) : l(\_l),r(\_r),idx(\_i) { }**

**//1)Two queries with L in the same block are sorted as per increasing or dercreading R.**

**//2)Two queries with L in different blocks are sorted as per increasing or decreasing LB (L Block)**

**bool operator < (const query &p) const {**

**if(l/block\_sz != p.l/block\_sz) return l < p.l;**

**return ((l/block\_sz) & 1) ? r > p.r : r < p.r;**

**}**

**};**

**//for shorting the query this function also can be used.**

**bool cmp(const query &a,const query &b) {**

**if(a.l == b.l)//1 number condition**

**return a.r < b.r;**

**return a.l/block\_sz < b.l/block\_sz;//2 number condition**

**//return (a.l/block\_sz < b.l/block\_sz) || (a.l == b.l && a.r < b.r);**

**}**

**void mo(vector<query> &q)**

**{**

**//sort(q.begin(),q.end(),cmp);//using cmp function**

**sort(q.begin(),q.end());**

**memset(ans,-1,sizeof(ans));**

**int c\_l = 0,c\_r = -1;//current l and r**

**fr(q)**

**{**

**//i.l and i.r is the required l and r i.e. given query range**

**while(i.l < c\_l) add(--c\_l);**

**while(i.r > c\_r) add(++c\_r);**

**while(i.l > c\_l) erase(c\_l++);**

**while(i.r < c\_r) erase(c\_r--);**

**ans[i.idx] = max(ans[i.idx] , mo\_cnt);**

**}**

**}**

**void solve()**

**{**

**int n;**

**cin >> n;**

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

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

**int q;**

**cin>> q;**

**vector<query> que;**

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

**{**

**int l,r;**

**cin >> l >> r;**

**que.pb(query(l,r,i));**

**}**

**mo(que);**

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

**cout << ans[i] <<endl;**

**/\***

**//find the number of distinct element between l to r inclusive**

**5**

**1 2 3 4 5**

**3**

**0 2**

**1 4**

**0 4**

**\*/**

**}**

**#Fenwick Tree(1D)**

**class FenwickTree{**

**public:**

**/\* everything is 1 based indexing \*/**

**vector<ll> BIT;**

**FenwickTree(int size){**

**BIT = vector<ll>(size+3, 0);**

**}**

**void update(int idx, ll val, int n){**

**while(idx <= n){**

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

**idx += (idx & - idx);**

**}**

**}**

**ll prefix\_query(int idx){**

**ll res = 0;**

**while(idx > 0){**

**res += BIT[idx];**

**idx -= (idx & -idx);**

**}**

**return res;**

**}**

**ll range\_query(int l, int r){**

**return prefix\_query(r) - prefix\_query(l-1);**

**}**

**};**

**#Graph**

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

**vi ad\_list[N];**

**bool vis[N];**

**ll path\_no = 1;**

**ll path[N];//for path between any two vertex**

**queue<int>q;**

**void dfs(int u,ll path\_no)//TC-->BigO(v+e)**

**{**

**vis[u] = true;**

**fr(ad\_list[u]){**

**path[i] = path\_no;**

**if(!vis[i])**

**dfs(i,path\_no);**

**}**

**}**

**void bfs(int u)**

**{**

**q.push(u);**

**vis[u] = true;**

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

**{**

**u = q.front();**

**q.pop();**

**fr(ad\_list[u])**

**{**

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

**q.push(i);**

**vis[i] = true;**

**}**

**}**

**}**

**}**

**void is\_conected\_graph(int v)//TC-->BigO(v+e)**

**{**

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

**{**

**if(!vis[i])**

**{**

**cout << "DISCONNECTED GRAPH" << endl;**

**return;**

**}**

**}**

**cout << "CONNECTED GRAPH"<< endl;**

**/\***

**test case for connected and disconnected graph**

**4 5**

**1 2**

**1 3**

**2 3**

**2 4**

**3 4**

**CONNECTED GRAPH**

**4 3**

**2 3**

**2 4**

**3 4**

**DISCONNECTED GRAPH**

**\*/**

**}**

**void connected\_component(int v)//TC-->BigO(v+e).**

**{**

**ll connected\_com = 0;**

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

**{**

**if(!vis[i])**

**{**

**connected\_com++;**

**dfs(i,path\_no);**

**path\_no++;**

**}**

**}**

**// cout << "Total Connected Component is " << connected\_com << endl;**

**//with this func i can count total connected component**

**//can detect path for any two two vetex i.e. is there any path between u and v**

**/\***

**test case for is the there any path between u and v**

**6 4**

**1 2**

**1 3**

**3 4**

**5 6**

**3(query)**

**1 5**

**5 6**

**3 6**

**TC-->bigO((v+e)+q)-->(v+e)**

**\*/**

**}**

**void solve()**

**{**

**int v,e;**

**cin>> v >> e;**

**while(e--)**

**{**

**int u,v;**

**cin >> u >> v;**

**ad\_list[u].pb(v);**

**ad\_list[v].pb(u);**

**}**

**// dfs(1);**

**// is\_conected\_graph(v);**

**connected\_component(v);**

**int q;**

**cin >> q;**

**while(q--)**

**{**

**int u,v;**

**cin >> u >> v;**

**if(path[u] == path[v])**

**cout << "Has a path between them" << endl;**

**else**

**cout << "No path between them" << endl;**

**}**

**}**

**#Bipartite Graph**

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

**vi ad\_list[N];**

**bool vis[N];**

**int col[N];//for bipartite graph**

**bool bipartite;**

**void dfs(int u)//TC-->BigO(v+e).DFS is a recursive travers algorithm for greaph.**

**{**

**vis[u] = true;**

**fr(ad\_list[u])**

**{**

**if(!vis[i])**

**{**

**col[i] = col[u] ^ 1;//changing the color of adjacent node of u.u 0 hole or adjacent node hobe 1**

**dfs(i);**

**}**

**else**

**{**

**if(col[u] == col[i])**

**bipartite = false;**

**}**

**}**

**}**

**void solve()**

**{**

**int v,e;**

**cin >> v >> e;**

**while(e--)**

**{**

**int u,v;**

**cin>> u >>v;**

**ad\_list[u].pb(v);**

**ad\_list[v].pb(u);**

**}**

**bipartite = true;**

**//connected graph hoile 1 theke call korlei hobe.**

**// dfs(1);**

**//for disconnected graph**

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

**{**

**if(!vis[i])**

**dfs(i);**

**}**

**if(bipartite)**

**cout << "The graph is a bipartite graph" << endl;**

**else**

**cout << "The graph is not a bipartite graph" << endl;**

**/\***

**Test Case:**

**4 4**

**1 2**

**1 3**

**2 4**

**3 4**

**connected bipartite graph.**

**3 3**

**1 2**

**1 3**

**2 3**

**connected not bipartite graph.**

**5 3**

**1 5**

**2 3**

**2 4**

**disconnected bipartite graph.**

**4 3**

**2 3**

**2 4**

**3 4**

**disconnected not bipartite graph.**

**\*/**

**}**

**#Sieve**

**const ll N = 1e8+9;//1e8+9 pojjonto run kore sudu codeblocks a.**

**bool mark[N];**

**vl vc;**

**void s()**

**{**

**mark[0] = mark[1] = 1;//mark.set(0),mark.set(1);ei vabeo likha jaito**

**for(ll i = 2;i < N;i++)**

**{**

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

**{**

**vc.pb(i);**

**for(ll j = i+i;j < N;j += i)**

**mark[j] = 1;//mark.set(j);**

**}**

**}**

**// for(int i = 0;i < vc.size();i++)**

**// cout << vc[i] << endl;**

**// cout << vc.size();**

**}**

**#Segment Tree (Range + Lazy)**

**const int N = 5e5 + 9;**

**int a[N];**

**struct ST {**

**#define lc (n << 1)**

**#define rc ((n << 1) + 1)**

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

**ST(){**

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

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

**}**

**//parent er main value and child gulor lazy value update korar function**

**inline void push(int n,int b,int e)//change this**

**{**

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

**return;**

**t[n] = t[n] + lazy[n]\*(e-b+1);//parent node er value update**

**if(b != e)**

**{**

**lazy[lc] = lazy[lc] + lazy[n];//left child er lazy value update**

**lazy[rc] = lazy[rc] + lazy[n];//right child er lazy value update**

**}**

**lazy[n] = 0;//jehetu parent node er child er lazy value update kore pelsi so parent er lazy 0 kore dite hobe**

**}**

**//value merge kore return korbe query er jonno**

**inline int combine(int a,int b)//change this**

**{**

**return a+b;**

**}**

**//parent node er value save korar function**

**inline void pull(int n)//change this**

**{**

**t[n] = t[lc] + t[rc];**

**}**

**//building the tree**

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

**{**

**lazy[n] = 0;//change this**

**if(b == e)**

**{**

**t[n] = a[b];**

**return;**

**}**

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

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

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

**pull(n);**

**}**

**//updating the value in a segment**

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

**{**

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

**if(e < i or b > j)**

**return;**

**if(b >= i and e <= j)**

**{**

**lazy[n] = x;//set lazy**

**push(n,b,e);//ei line na korleo hobe**

**return;**

**}**

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

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

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

**pull(n);**

**}**

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

**{**

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

**if(e < i or b > j)**

**return 0;**

**if(b >= i and e <= j)**

**return t[n];**

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

**return combine(query(lc,b,mid,i,j),query(rc,mid+1,e,i,j));**

**}**

**}t;**

**void solve()**

**{**

**int n;**

**cin >> n;**

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

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

**t.build(1,1,n);//building the tree**

**t.upd(1,1,n,2,3,2);//udating the value of segment 2 to 3 with adding 1**

**cout << t.query(1,1,n,1,n);//printig the sum of semment 1 to n**

**}**

**#Segment Tree (Single Ind Update)**

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

**int a[N];**

**int t[4\*N];//saving the sum of segment in the tree**

**void build(int node,int b,int e)//b theke e node er range**

**{**

**if(b == e)**

**{**

**t[node] = a[b];**

**return;**

**}**

**int l = node\*2;**

**int r = node\*2 + 1;**

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

**build(l,b,mid);//l node er segment building**

**build(r,mid+1,e);// r node er segment building**

**t[node] = t[l]+t[r];//saving the sum of this node.Change this one according to question**

**}**

**int query(int node,int b,int e,int i,int j)//query hocche i theke j(TC-->Log(N))**

**{**

**if(e < i or b > j)//current node er range ta query range er bahire**

**return 0;//return appropriate value according to question**

**if(b >= i and e <= j)//current node er range ta query range er vitore**

**return t[node];**

**//current node er range ta query range ke intersect korse**

**int l = node\*2;**

**int r = node\*2 + 1;**

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

**return query(l,b,mid,i,j) + query(r,mid+1,e,i,j);//Change this one according to question**

**}**

**void upd(int node,int b,int e,int i,int x)//i index er value ke update kore x banaute hobe(TC-->log(n))**

**{**

**if(i > e or i < b)**

**return;**

**if(b == e and b == i)**

**{**

**t[node] = x;//single update**

**return;**

**}**

**int l = node\*2;**

**int r = node\*2 + 1;**

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

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

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

**t[node] = t[l]+t[r];//Change this one according to question**

**}**

**void solve()**

**{**

**int n;**

**cin >> n;**

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

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

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

**upd(1,1,n,5,0);**

**cout << query(1,1,n,1,5) << endl;**

**}**

**#Topshort -> TC(V+E)**

**//topshort fast -->TC(v+e)**

**const int N = 100;**

**int indeg[N];**

**vi ad\_list[N];**

**bool vis[N];**

**void solve()**

**{**

**int v,e;**

**cin >> v >> e;**

**while(e--)**

**{**

**int u,v;**

**cin>> u >> v;//age u er kaj ,then v er kaj hobe.u --> v**

**indeg[v]++;**

**ad\_list[u].pb(v);**

**}**

**vi ans,in\_deg\_zero;**

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

**{**

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

**{**

**in\_deg\_zero.pb(i);**

**// vis[i] = true;**

**}**

**}**

**while(ans.size() < v)//ei khane v bar choltese**

**{**

**if(in\_deg\_zero.empty())**

**{**

**cout << "IMPOSSIBLE" << endl;**

**return;**

**}**

**int cur;**

**cur = in\_deg\_zero.back();**

**in\_deg\_zero.pop\_back();**

**ans.pb(cur);**

**vis[cur] = true;//eita 47 number line a kore dileo hoito**

**fr(ad\_list[cur]){**

**indeg[i]--;**

**if(!vis[i] and indeg[i] == 0)**

**{**

**in\_deg\_zero.pb(i);**

**vis[i] = true;**

**}**

**}**

**}**

**fr(ans)**

**cout << i << " ";**

**/\***

**TC**

**3 3**

**1 2**

**2 3**

**3 1**

**IMPOSSIBLE CZ THE GRAPH IS CYCLIC..TOPSHORT ONLY POSSIBLE IN**

**DIRECTED UNCYCLIC GRAPH**

**3 2**

**2 1**

**2 3**

**ANS :2 --> 1 --> 3**

**4 5**

**1 3**

**1 4**

**4 2**

**4 3**

**2 3**

**ANS :1 --> 4 --> 2 --> 3**

**\*/**

**}**

**#LCA (lowest common ancestor)**

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

**vi ad\_list[N];**

**int level[N];**

**int up[N][20];**

**void dfs(int u,int par,ll lvl)//TC-->BigO(v+e)**

**{**

**level[u] = lvl;**

**fr(ad\_list[u]){**

**if(i != par)//this line can be deleted if we delete the 47th line also**

**dfs(i,u,lvl+1);**

**}**

**}**

**//This function TC-->O(Nlog(N))**

**void binary\_lifting(int src,int par)**

**{**

**up[src][0] = par;**

**for(int i = 1;i < 20;i++)**

**{**

**if(up[src][i-1] == -1)**

**up[src][i] = -1;**

**else**

**up[src][i] =up[up[src][i-1]][i-1];**

**}**

**fr(ad\_list[src])**

**{**

**if(i != par)**

**binary\_lifting(i,src);**

**}**

**}**

**int lift\_node(int u,int jump)//jump hobe 2 er power akare**

**{**

**if(u == -1 or jump == 0)**

**return u;**

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

**{**

**if(jump >= (1 << i))**

**return lift\_node(up[u][i],jump-(1<<i));**

**}**

**}**

**//TC-->O(log(n))**

**int lca(int u,int v)**

**{**

**if(level[u] < level[v])//root theke jar lvl beshi tare u te rakhtesi**

**swap(u,v);**

**u = lift\_node(u,level[u]-level[v]);//Now u and v are at the same level**

**if(u == v)//dont be confused.we made the level same not u and v**

**return u;**

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

**{**

**if(up[u][i] != up[v][i])//jump from u to n1 and from v to n2 untill n1 != n2.when n1 == n2 then we reach from u and v to 1 step lower node of our desired lca node.**

**{**

**u = up[u][i];**

**v = up[v][i];**

**}**

**}**

**return lift\_node(u,1);//Now u is one step lower node of lca.so we just need 1 jump to reach in LCA.**

**}**

**void solve()**

**{**

**int v,q;**

**cin>> v >> q;**

**for(int i= 2;i <= v;i++)**

**{**

**int x;**

**cin>>x;**

**ad\_list[x].pb(i);**

**ad\_list[i].pb(x);**

**}**

**dfs(1,-1,0);//root 1 er kono par nai so or par -1,ar or lvl 0**

**binary\_lifting(1,-1);**

**while(q--)**

**{**

**int u,v;**

**cin>> u >> v;**

**cout << lca(u,v) << endl;**

**}**

**}**

**#LCA+MST+Problem**

**struct node**

**{**

**int wt,u,v,id;**

**node(int weight,int l,int r,int i)**

**{**

**wt = weight;**

**u = l;**

**v = r;**

**id = i;**

**}**

**};**

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

**int siz[N],parent[N];**

**vpi ad\_list\_for\_mst[N];**

**int level[N];**

**int max\_edge[N][31];**

**int up[N][31];**

**void dfs(int u,int par,int lvl)**

**{**

**level[u] = lvl;**

**fr(ad\_list\_for\_mst[u])**

**{**

**if(i.first != par)**

**dfs(i.first,u,lvl+1);**

**}**

**}**

**//overall TC of DSU is constant.O(alpha(N)) --> reversed ackerman function..It increase very slowly.That is almost N.**

**void make(int n)//add new node to our current sets**

**{**

**parent[n] = n;**

**siz[n] = 1;**

**}**

**int find(int n)//return the root of that set where the node n belonged to.**

**{**

**if(parent[n] == n)**

**return n;**

**return parent[n] = find(parent[n]);//path compression**

**}**

**void uni(int a,int b)//set node a and b to the same group**

**{**

**a = find(a);//a node er root ta finding**

**b = find(b);//b node er root ta finding**

**if(a != b)//when both a and b are not belonged to the same set**

**{**

**//union by size**

**//We will add small tree to the big tree so that the depth of the tree will be smaller**

**if(siz[a] < siz[b])**

**swap(a,b);**

**parent[b] = a;**

**siz[a] += siz[b];**

**}**

**}**

**//This function TC-->O(Nlog(N))**

**void binary\_lifting(int src,int par)**

**{**

**up[src][0] = par;**

**for(int i = 1;i <= 20;i++)**

**{**

**if(up[src][i-1] == -1){**

**up[src][i] = -1;**

**max\_edge[src][i] = -1;**

**}**

**else{**

**up[src][i] =up[up[src][i-1]][i-1];**

**max\_edge[src][i] = max(max\_edge[src][i-1],max\_edge[up[src][i-1]][i-1]);**

**}**

**}**

**fr(ad\_list\_for\_mst[src])**

**{**

**if(i.first != par){**

**max\_edge[i.first][0] = i.second;**

**binary\_lifting(i.first,src);**

**}**

**}**

**}**

**int lift\_node(int u,int jump)//jump hobe 2 er power akare**

**{**

**if(u == -1 or jump == 0)**

**return u;**

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

**{**

**if(jump >= (1 << i))**

**return lift\_node(up[u][i],jump-(1<<i));**

**}**

**}**

**//TC-->O(log(n))**

**int lca(int u,int v)**

**{**

**if(level[u] < level[v])//root theke jar lvl beshi tare u te rakhtesi**

**swap(u,v);**

**u = lift\_node(u,level[u]-level[v]);//Now u and v are at the same level**

**if(u == v)//dont be confused.we made the level same not u and v**

**return u;**

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

**{**

**if(up[u][i] != up[v][i])//jump from u to n1 and from v to n2 untill n1 != n2.when n1 == n2 then we reach from u and v to 1 step lower node of our desired lca node.**

**{**

**u = up[u][i];**

**v = up[v][i];**

**}**

**}**

**return lift\_node(u,1);//Now u is one step lower node of lca.so we just need 1 jump to reach in LCA.**

**}**

**int max\_edge\_in\_the\_cycle(int u,int v){**

**if(level[u] < level[v])//root theke jar lvl beshi tare u te rakhtesi**

**swap(u,v);**

**int jump = level[u] - level[v];**

**int res = 0;**

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

**{**

**if(jump & (1 << i))**

**{**

**res = max(res,max\_edge[u][i]);**

**u = up[u][i];**

**}**

**}**

**return res;**

**}**

**bool cmp(const node &A,const node &B)**

**{**

**return A.wt < B.wt;**

**}**

**void solve()**

**{**

**int n,q;**

**cin >> n >> q;**

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

**make(i);**

**vector<node>edges;**

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

**{**

**int u,v,wt;**

**cin >> u >> v >> wt;**

**edges.pb(node(wt,u,v,i));**

**}**

**sort(all(edges),cmp);**

**// fr(edges)**

**// cout << i.u << " "<< i.v << " " << i.wt << " " << i.id <<endl;**

**vi ans(q+5,-1);**

**vi edge\_taken\_by\_mst(q+5,0);**

**int total\_wt\_of\_mst = 0;**

**fr(edges)**

**{**

**int wt = i.wt;**

**int u = i.u;**

**int v = i.v;**

**int id = i.id;**

**if(find(u) != find(v))**

**{**

**total\_wt\_of\_mst += wt;**

**uni(u,v);**

**ad\_list\_for\_mst[u].pb(mp(v,wt));**

**ad\_list\_for\_mst[v].pb(mp(u,wt));**

**edge\_taken\_by\_mst[id] = 1;**

**}**

**}**

**dfs(1,-1,0);**

**binary\_lifting(1,-1);**

**fr(edges)**

**{**

**if(!edge\_taken\_by\_mst[i.id])**

**{**

**int max\_edge\_val = 0;**

**int u = i.u;**

**int v = i.v;**

**int lca\_of\_uv = lca(u,v);**

**max\_edge\_val = max(max\_edge\_val,max\_edge\_in\_the\_cycle(lca\_of\_uv,u));**

**max\_edge\_val = max(max\_edge\_val,max\_edge\_in\_the\_cycle(lca\_of\_uv,v));**

**ans[i.id] = total\_wt\_of\_mst-max\_edge\_val+i.wt;**

**}**

**else**

**ans[i.id] = total\_wt\_of\_mst;**

**}**

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

**cout << ans[i] << endl;**

**}**

**#Binary Lifting**

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

**vi tree[N];**

**int up[N][20];//These information allow us to jump from any node to any ancestor above it in  O(log(N)) time**

**void binary\_lifting(int src,int par)**

**{**

**up[src][0] = par;//2^0 = 1 node uprer ancestor src er parent nijei**

**for(int i = 1;i < 20;i++)//2^20 == 1e6..**

**{**

**if(up[src][i-1] == -1)**

**up[src][i] = -1;**

**else**

**up[src][i] = up[up[src][i-1]][i-1];//u node theke 2^x node uporer node = up[src][2^x-1][2^x-1]..2^x = 2^x-1 + 2^x-1**

**}**

**fr(tree[src])**

**{**

**if(i != par)**

**binary\_lifting(i,src);**

**}**

**}**

**//per query TC-->O(log(n))**

**int query(int v,int k){**

**if(v == -1 or k == 0)**

**return v;**

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

**{**

**if(k >= (1<<i))**

**return query(up[v][i],k-(1<<i));**

**}**

**}**

**void solve()**

**{**

**int n,q;**

**cin>> n >> q;**

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

**{**

**int x;**

**cin>>x;**

**tree[x].pb(i);**

**tree[i].pb(x);**

**}**

**binary\_lifting(1,-1);//root 1 er upore ar kono lvl nai.tai er jonno parent -1**

**while(q--)**

**{**

**int v,k;**

**cin >>v >> k;**

**cout << query(v,k) << endl;**

**}**

**}**

**#DSU**

**int const N = 3e5+9;**

**int siz[N],parent[N];**

**//overall TC of DSU is constant.O(alpha(N)) --> reversed ackerman function..It increase very slowly.That is almost N.**

**void make(int n)//add new node to our current sets**

**{**

**parent[n] = n;**

**siz[n] = 1;**

**}**

**int find(int n)//return the root of that set where the node n belonged to.**

**{**

**if(parent[n] == n)**

**return n;**

**return parent[n] = find(parent[n]);//path compression**

**}**

**void uni(int a,int b)//set node a and b to the same group**

**{**

**a = find(a);//a node er root ta finding**

**b = find(b);//b node er root ta finding**

**if(a != b)//when both a and b are not belonged to the same set**

**{**

**//union by size**

**//We will add small tree to the big tree so that the depth of the tree will be smaller**

**if(siz[a] < siz[b])**

**swap(a,b);**

**parent[b] = a;**

**siz[a] += siz[b];**

**}**

**}**

**void solve()**

**{**

**//There are n node.q query.Find the total distinct connected component**

**int n,q;**

**cin >> n >> q;**

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

**make(i);**

**while(q--)**

**{**

**int u,v;**

**cin>> u >> v;**

**uni(u,v);**

**}**

**int connected\_com = 0;**

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

**{**

**if(find(i) == i)**

**connected\_com++;**

**}**

**cout << connected\_com << endl;**

**/\***

**4 2**

**1 2**

**2 3**

**ans --> 2**

**\*/**

**}**

**#MST**

**//Fist add dsu functions**

**//MST ALGORITHM with kruskal.TC-->O(E)**

**void solve()**

**{**

**int n,e;**

**cin >> n >> e;**

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

**make(i);**

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

**while(e--)**

**{**

**int u,v,wt;**

**cin>> u >> v>>wt;**

**//we will sort the edges depends on weight**

**edges.pb(mp(wt,mp(u,v)));**

**}**

**sort(all(edges));**

**int total\_c = 0;**

**fr(edges)**

**{**

**int wt = i.first;**

**int u = i.second.first;**

**int v = i.second.second;**

**//if the root of u and v is not same then connecting u and v will not create any cycle**

**if(find(u) != find(v))**

**{**

**uni(u,v);**

**total\_c += wt;**

**cout << u << " " << v << endl;**

**}**

**}**

**cout << total\_c << endl;**

**/\***

**Test case:**

**6 9**

**5 4 9**

**1 4 1**

**5 1 4**

**4 3 5**

**4 2 3**

**1 2 2**

**3 2 3**

**3 6 8**

**2 6 7**

**\*/**

**}**

**#BELLMAN FORD ALGO**

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

**int const INF = 1e18+9;**

**vi dis(N,INF);**

**struct node**

**{**

**int u,v,wt;**

**node(int l,int r,int weight)**

**{**

**u = l;**

**v = r;**

**wt = weight;**

**}**

**};**

**//Bellman ford algo for single source shortes path**

**//It detect negative weighted cycle in graph**

**//TC --> O(n-1) \* O(e).Slower than dijkstra**

**void solve()**

**{**

**int n,e;**

**cin >> n >> e;**

**vector<node>edges;**

**while(e--)**

**{**

**int u,v,wt;**

**cin >> u >> v >> wt;**

**edges.pb(node(u,v,wt));**

**}**

**int s;**

**cin >>s;**

**dis[s] = 0;**

**for(int i = 1;i <=n-1;i++)//realxing every edge for n-1 times**

**{**

**fr(edges)**

**{**

**if(dis[i.u] + i.wt < dis[i.v])**

**dis[i.v] = dis[i.u] + i.wt;**

**}**

**}**

**//one more relaxation to check wheather the graph contains any negative weighted cylce or not**

**fr(edges)**

**{**

**if(dis[i.u] + i.wt < dis[i.v])**

**{**

**cout << "The graph cotains negative wighted cylce!So shortest path can not be possible!";**

**return;**

**}**

**}**

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

**cout << s << " to " << i << " shortest distance is " << dis[i] << endl;**

**/\***

**4 4**

**1 2 1**

**2 3 1**

**3 4 1**

**1 3 1**

**1**

**//negative weighted cycle graph**

**3 3**

**1 2 3**

**2 3 4**

**3 4 -8**

**1**

**\*/**

**}**

**#FLoyd Warshall ALgo**

**int const N = 510;**

**int const INF = 1e18;**

**int dis[N][N];**

**//FLoyd Warshall ALgo**

**//It can handle negative weighted graph but can not handle negative weighted cyclic graph**

**//TC->O(n^3)**

**void solve()**

**{**

**int n,e;**

**cin>> n >> e;**

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

**{**

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

**{**

**if(i == j)**

**dis[i][j] = 0;**

**else**

**dis[i][j] = INF;**

**}**

**}**

**while(e--)**

**{**

**int u,v,wt;**

**cin >> u >> v >> wt;**

**dis[u][v] = wt;**

**//dist[v][u] = wt; undirected graph hole eitau hobe**

**}**

**for(int k = 1;k <= n;k++)//k is the level or nodes.every time we allow a path i to j to reach through k nodes.**

**{**

**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]);**

**}**

**}**

**}**

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

**{**

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

**{**

**if(dis[i][j] == INF)**

**cout << "I ";**

**else**

**cout << dis[i][j] << " ";**

**}**

**cout << endl;**

**}**

**}**

**#Shortest Path with bfs**

**void bfs(int u)**

**{**

**q.push(u);**

**vis[u] = true;**

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

**{**

**u = q.front();**

**q.pop();**

**fr(ad\_list[u])**

**{**

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

**q.push(i);**

**path[i] = u;// i tomo node a ashci u node theke.tai i te u hobe**

**// dis[i] = dis[u] + 1;//source node theke jekono node a jauar shortest path,jodi tader moddhe kono path thake**

**vis[i] = true;**

**}**

**}**

**}**

**}**

**#Diameter of a Tree**

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

**vi ad\_list[N];**

**int depth[N];**

**int mx\_dia,mx\_node;**

**void dfs(int u,int p)//TC-->BigO(v+e)**

**{**

**depth[u] = depth[p]+1;**

**if(depth[u] > mx\_dia)**

**{**

**mx\_dia = depth[u];**

**mx\_node = u;**

**}**

**fr(ad\_list[u]){**

**if(i != p)**

**dfs(i,u);**

**}**

**}**

**void solve()**

**{**

**int v,e;**

**cin>> v;**

**e = --v;**

**while(e--)**

**{**

**int u,v;**

**cin >> u >> v;**

**ad\_list[u].pb(v);**

**ad\_list[v].pb(u);**

**}**

**dfs(1,0);**

**dfs(mx\_node,0);**

**cout << mx\_dia-1 << endl;**

**}**

**#Depth of a Tree**

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

**vi ad\_list[N];**

**int depth[N];**

**//root node theke baki node gular distance**

**void dfs(int u,ll p)//TC-->BigO(v+e)**

**{**

**depth[u] = depth[p]+1;**

**fr(ad\_list[u]){**

**if(i != p)//this line can be deleted if we delete the 47th line also**

**dfs(i,u);**

**}**

**}**

**void solve()**

**{**

**int v,e;**

**cin>> v;**

**e = v-1;**

**while(e--)**

**{**

**int u,v;**

**cin >> u >> v;**

**ad\_list[u].pb(v);**

**ad\_list[v].pb(u);//this line can be deleted if we delete the 32th line also**

**}**

**dfs(1,0);**

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

**cout << depth[i] << " ";**

**}**

**#CYCLE DETECTION AND CYCLE PATH(Undirected)**

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

**using namespace std;**

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

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

**int col[N], par[N];**

**bool cycle;**

**deque<int>path;**

**int f = 1,l = 0;**

**void dfs(int u) {**

**if(f)**

**path.push\_back(u);**

**col[u] = 1;**

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

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

**par[v] = u;**

**dfs(v);**

**}**

**else if (col[v] == 1) {**

**cycle = true;**

**// you can track the cycle using par array**

**f = 0;**

**if(l == 0)**

**l = v;**

**}**

**}**

**col[u] = 2;//je shokol node theke cycle create kora possible hobe na tader color 2 hoye jabe.That is kono node er jonno or adjacent list visit kora ses but kono cycle pai nai,oi khetre oi node er color 2 hoiye jabe**

**}**

**int32\_t main() {**

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

**cin.tie(0);**

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

**for (int i = 1; i <= n; i++) {//i theke u te jabe**

**int u;**

**cin >> u;**

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

**}**

**cycle = false;**

**vector<int> ans;**

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

**if (col[i] == 0)dfs(i);**

**if(cycle)**

**{**

**while(path.front() != l and !path.empty())**

**path.pop\_front();**

**cout << path.size() << endl;**

**for(auto &i:path)**

**cout << i << " ";//printing cycle path**

**break;**

**}**

**}**

**return 0;**

**}**

**#Cycle Detection(Directed)**

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

**bool vis[N];**

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

**int f = 0;**

**bool cycle;**

**void dfs(int u,int p){**

**vis[u] = 1;**

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

**if(vis[v] and v == p)**

**continue;**

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

**if(!f)**

**f = v;//u theke v er moddhe fist cycle ta belong kore.only first cycle ta detect korar jonno ei logic**

**cycle = true;**

**}**

**else**

**dfs(v,u);**

**}**

**}**

**void solve()**

**{**

**int n,e;**

**cin >>n >> e;**

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

**int u,v;**

**cin >> u >> v;**

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

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

**}**

**cycle = false;**

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

**{**

**if(!vis[i])**

**dfs(i,0);**

**}**

**if(cycle)**

**cout << "Cycle ache" << endl;**

**else**

**cout << "Cycle Nai" << endl;**

**}**

**#PBDS**

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

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

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

**using namespace std;**

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

**//ordered\_set**

**template <typename DT>**

**using oset = tree<DT, null\_type, less<DT>, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;**

**//ordered\_set for func**

**template <typename DT, typename FUNC>**

**using o\_set = tree<DT, null\_type, FUNC, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;**

**//ordered\_map**

**template <typename DT1, typename DT2>**

**using omap = tree<DT1, DT2, less<DT1>, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;**

**//ordered\_map for func**

**template <typename DT1, typename DT2, typename FUNC>**

**using o\_map = tree<DT1, DT2, FUNC, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;**

**/**

**\* - less<data\_type>--> Increasingly sorted set**

**\* - less\_equal<data\_type> --> Increasingly sorted multiset**

**\* - greater<data\_type> --> Decreasingly sorted set**

**\* - greated\_equal<data\_type> --> Decreasingly sorted multiset**

**\* - Problem link ---> https://www.spoj.com/problems/ORDERSET/en/**

**/**

**int main()**

**{**

**oset<int> s;**

**s.insert(2);**

**s.insert(1);**

**s.insert(5);**

**s.insert(10);**

**cout << s.order\_of\_key(10) << endl;**

**cout << \*s.find\_by\_order(2) << endl;**

**}**

**#Big POW**

**ll powss(ll a, ll p)//for big pow value**

**{**

**if(p==0)**

**return 1;**

**if(p==1)**

**return a;**

**else if(p&1)**

**{**

**return a\*powss(a\*a,p/2);**

**}**

**else**

**{**

**return powss(a\*a,p/2);**

**}**

**}**

**#Power Of TWO**

**bool powerOftwo(ll x)**

**{**

**if(x == 0)**

**return false;**

**else**

**return !(x&(x-1));**

**}**

**#All possible Permutation**

**void permutation(vector<ll>v)//O(N\*N!)....O(N) time to find the next permutation and there are N! number of permutations for an array of size N**

**{**

**sort(all(v));**

**cout << "All possible permutations with the elements:\n";**

**do**

**{**

**fr(v)**

**cout << i << " ";**

**cout << endl;**

**}**

**while (next\_permutation(all(v)));**

**}**

**string decToBinary(int n)**

**{**

**string s;**

**int i = 0;**

**while (n > 0)**

**{**

**s = to\_string(n % 2) + s;**

**n = n / 2;**

**i++;**

**}**

**return s;**

**}**

**ll binaryToDecimal(string n)**

**{**

**string num = n;**

**ll dec\_value = 0;**

**int base = 1;**

**int len = num.length();**

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

**{**

**if (num[i] == '1')dec\_value += base;**

**base = base \* 2;**

**}**

**return dec\_value;**

**}**

**ll gcd(ll a, ll b)**

**{**

**while (b)**

**{**

**a %= b;**

**swap(a, b);**

**}**

**return a;**

**}**

**ll lcm(ll a, ll b)**

**{**

**return (a / gcd(a, b) \* b);**

**}**

**ll ncr(ll a, ll b)**

**{**

**ll x = max(a - b, b), ans = 1;**

**for (ll K = a, L = 1; K >= x + 1; K--, L++)**

**{**

**ans = ans \* K;**

**ans /= L;**

**}**

**return ans;**

**}**

**ll egcd(ll a, ll b, ll &x, ll &y)**

**{**

**if (a == 0)**

**{**

**x = 0;**

**y = 1;**

**return b;**

**}**

**ll x1, y1;**

**ll d = egcd(b % a, a, x1, y1);**

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

**y = x1;**

**return d;**

**}**

**db pytha(db x1, db x2, db y1, db y2)**

**{**

**return sqrt((pow((x2 - x1), 2)) + (pow((y2 - y1), 2)));**

**} /// for two ending points pythagoras law**

**double make\_radian(double x)**

**{**

**return (x \* pi) / 180;**

**}**

**double make\_degree(double x)**

**{**

**return (x \* 180) / pi;**

**}**

**ll binmul(ll a, ll b, ll p)**

**{**

**ll res = 0ll;**

**while (b)**

**{**

**if (b & 1)**

**res = (res + a) % p, b--;**

**else**

**a = (a + a) % p, b /= 2;**

**}**

**return res;**

**} //(a\*b)%p**

**ll binpow(ll a, ll b, ll p)**

**{**

**ll res = 1ll;**

**while (b)**

**{**

**if (b & 1)**

**res = binmul(res, a, p), b--;**

**else**

**a = binmul(a, a, p), b /= 2;**

**}**

**return res;**

**} //(a^b)%p**

**ll inverse(ll a, ll p)**

**{**

**return binpow(a, p - 2, p);**

**} //(a^-1)%p == (a^p-2)%p**

**string big\_add(string a,string b)**

**{**

**string ans;**

**if(a.size() > b.size())**

**{**

**while(b.size() != a.size())b = '0'+b;**

**}**

**else**

**{**

**while(a.size() != b.size())**

**a = '0'+a;**

**}**

**int carry = 0;**

**for(int i = a.size()-1;i >= 0;i--)**

**{**

**int x = a[i]-'0';**

**int y = b[i]-'0';**

**int sum = x+y+carry;**

**if(sum >= 10)**

**{**

**carry = 1;**

**sum %= 10;**

**}**

**else**

**carry = 0;**

**ans += sum+'0';**

**}**

**if(carry)**

**ans += '1';**

**reverse(all(ans));**

**return ans;**

**}**

**string big\_sub(string a,string b)**

**{**

**string ans;**

**if(a.size() > b.size())**

**{**

**while(b.size() != a.size())**

**b = '0'+b;**

**}**

**else**

**{**

**while(a.size() != b.size())**

**a = '0'+a;**

**}**

**int carry = 0;**

**for(int i = a.size()-1;i >= 0;i--)**

**{**

**int x = a[i]-'0';**

**int y = b[i]-'0';**

**if(carry)y++;**

**int f = 0;**

**if(x < y)**

**{**

**x += 10;**

**f = 1;**

**}**

**int sub = x-y;//x theke y biyog(a string theke b string biyog)**

**ans += sub+'0';**

**if(f)**

**carry = 1;**

**else**

**carry = 0;**

**}**

**while(ans.back() == '0')**

**ans.pop\_back();**

**reverse(all(ans));**

**return ans;**

**}**

**int charToint(char ch)**

**{**

**return ch-'0';**

**}**

**char intTochar(int n)**

**{**

**return 48+n;**

**//return '0'+n;**

**}**

**#Notes**

**Number Theory**

**1. 3 ways to do modulo inverse. 1/a % m**

**which is equivalent to ax = 1 mod m**

**a. m is prime - a^(m-2) = 1/a mod m**

**b. m is not prime - ax+my = 1 find x by**

**egcd**

**c. m is not prime - a^phi(m)-1 = 1/a**

**mod m**

**2. NOD(n) = (a1+1) \* (a2+1) \* (a3+1) \* … \***

**(ak+1)**

**3. SOD(n) = (pk^(ak+1) - 1 ) / ( pk - 1 )**

**4. Egcd solve the equation ax + by =**

**gcd(a, b)**

**a. X += k \* b/gcd(a,b) ; Y -= k \***

**a/gcd(a,b)**

**5. Diophantine equation : ax + by = c**

**solution exist iff c | gcd(a, b)**

**6. gcd(a, b) = gcd(-a, y) = gcd(x, -y) =**

**gcd(-x, -y)**

**7. gcd(a, b) = gcd(a, b+a) = gcd(a, b-a),**

**here b > a**

**8. gcd(a[0], a[1], a[2],..,a[n]) = gcd(a[0],**

**a[1]-a[0], a[2]-a[1],..,a[n]-a[n-1])**

**a. k is added to all value then just add**

**it to a[0]**

**9. phi(n) = n \* (1-1/p1) \* (1-1/p2) .. (1-1/pk)**

**10. Number of x <= n and gcd(x, n) = 1 is**

**equal to phi(n/d)**

**11. Sum of above x is equal to phi(n) \* n / 2**

**12. Sum of phi(d) = n for all d.**

**13. Arithmetic progression : a + (d+a) +**

**(2d+a) ..**

**a. Nth term : a + (n-1)d**

**b. Sum of n term : n/2[2a + (n-1)\*d]**

**c. Sum of AP : n/2[1st term + nth term]**

**14. Geometric progression: a + ar + ar^2 +**

**ar^(n-1)**

**a. Nth term : ar^(n-1)**

**b. Sum of n term : a[(r^n - 1)/(r-1)]**

**c. Sum of n term : (nth\*r -**

**1st)/(r-1)**

**15. Harmonic sum 1+1/2+1/3+..+1/n = logn**

**+1**

**Bitwise**

**16. - Check kth bit of a number by : x & ( 1**

**<< k ) if greater then 0 then kth bit is 1**

**otherwise 0.**

**17. - Set kth bit of a number by : x = x | ( 1**

**<< k ) .**

**18. - Reset kth bit of a number by : x = x &**

**~( 1<<k )**

**19. - Toggle kth bit of a number by : x = x ^**

**( 1<<k ).**

**20. - Check x is a power of 2 by : x & (x-1)**

**== 0 then it is power of 2 otherwise not.**

**21. - All bit of x are on if x & (x + 1) == 0 .**

**22. - XOR of 1 to N in constant time :**

**(n&3==0)→n,(n&3==1)→1,**

**(n&3==2)→n+1,otherwise 0.**

**23. - Find k bit number whose all bit are on**

**: number = ( 1 << k ) - 1**

**24. - 2's complement of a number means if**

**we traverse a binary number from LSB**

**to MSB and if we find a set bit, after**

**that first set bit if we flip remaining bits**

**it will for 2's complement . We can get**

**2's complement of x by : ( -x ).**

**25. - 2's biggest power which is divisor of x**

**: 1 << ( \_\_builtin\_ctz( x ) ).**

**26. - 2's smallest power which is not**

**smaller then x : 1 << ( 32 - \_\_builtin\_clz(**

**x - 1 ) ). But it has Undefined Behavior**

**for x ≤ 1 so we need an extra check for**

**this.**

**27. a+b = (a^b) + 2\*(a&b)**

**28. a|b = (a^b) + (a&b)**

**Graph**

**29. Mincut = Max flow**

**30. Maximum independent set = V -**

**matching**

**31. Minimum vertex cover = matching**

**32. Minimum edge cover (general) = V -**

**matching**

**Geometry and Trigonometry**

**33. sin(th)^2 + cos(th)^2 = 1**

**34. sec(th)^2 - tan(th)^2 = 1**

**35. cosec(th)^2 - cot(th)^2 = 1**

**36. a/sinA = b/sinB = c/sinC = 2R**

**37. a^2 = b^2 + c^2 - 2bccosA**

**38. A = bcosC + ccosB**

**39. T.area = 1/2 bcsinA =**

**sqrt(s\*(s-a)\*(s-b)\*(s-c))**

**40. In circle radius,r = (2\*T.area)/(a+b+c)**

**41. Circumcicle radius,R = abc/4T.area**

**42. Rr = abc/4s**

**43. T.area = 1/2bh**

**44. T.area =**

**1/2(x1\*(y1-y3)+x2\*(y3-y1)+x3\*(y1-y2))**

**45. C.area = pi \* r^2**

**46. Circumference = 2\*pi\*r**

**47. Length of arc = pi\*r\*th/180**

**48. Sector area = (th/360) \* pi \* r^2**

**49. y-y1 = m ( x-x1)**

**50. y = mx + c**

**51. Line - Ax + By = C or Ax+By+C = 0**

**52. Parametric - p(t) = p0 + t(p1-p0)**

**53.Div Sum Formula:(P1^(e1+1) - 1 / p1 - 1)\*(P2^(e2+1) - 1 / p2 - 1)...**

**(Pk^(ek+1) - 1 / pk - 1)**

54.**Max Prime Gap = (log(n))^2**

**55.ceil(a,b) = (a+b-1)/b**

**56.1^2 + 2^2 + 3^2 + ……+n^2 = (n\*(n+1)\*(2n+1))/6**

**57.1^3 + 2^3 + 3^3 +.........+n^3 =( (n\*(n+1))/2)^2**

**58.1^3 + 2^3 + 3^3 +.........+n^3 = (1+2+3+......+n)^2**

**59.2+4+6+8+.....+n = n(n+1)**

**60.1+3+5+7+......+n = n^2**

**61.2^2 + 4^2 + 6^2 + …..+ n^2=(2n\*(n+1)\*(2n+1))/3**

**62.1^2 + 3^2 + 5^2 + ….+ n^2 = (n\*(2n+1)\*(2n-1))/3**

**63.1^odd = odd-1**

**64.1^even = even+1**

**65.make the kth bit to set of x = (x|(1<<k))**

**66.make the kth bit to unset of x = (x&~(1<<k))**

**67.There is no intersection between a pair of segment if**

**max(L) > min(R)**