Number Theory

```cpp
#include <bits/stdc++.h >
#define ll long long int
#define MAX 10000109
#define MOD 1000000007

using namespace std;

typedef pair < ll, ll > pii;
const double eps = 1e-9;
bool isPrime[MAX + 5];
ll DivisorSum[MAX + 5];
ll DivisorNo[MAX + 5];
ll modInverse[MAX + 5];
ll SmallestPrime[MAX + 5];
ll PrimeFactor[MAX + 5];
ll Phi[MAX + 5];
ll lp[MAX + 5];
vector <ll> pr, prime;

void seive_N_logN(ll N){
    ///calculate prime upto N in NlogN time
    memset(isPrime, true, sizeof isPrime);
    prime.clear();  isPrime[1] = false; isPrime[0] = false;
    for (ll i = 4; i <= N; i = i + 2)    isPrime[i] = false;

    for (ll i = 3; i * i <= N; i = i + 2)
        if (isPrime[i])
            for (ll j = i * i; j <= N; j += i)
                isPrime[j] = false;

    for (ll i = 1; i < N; i++)
        if (isPrime[i])
            prime.push_back(i);
}

int status[(MAX / 32) + 2];

bool Check(int N, int pos){     return (bool)(N & (1 << pos));  }

int Set(int N, int pos){    return N = N | (1 << pos);  }

void bit_sieve(ll N){
    int sqrtN, i, j;
    sqrtN = (sqrt(N));
    for (i = 3; i <= sqrtN; i += 2)
        if (Check(status[i >> 5], i & 31) == 0)
            for (j = i * i; j <= N; j += (i << 1))
                status[j >> 5] = Set(status[j >> 5], j & 31);
    prime.push_back(2);
    for (i = 3; i <= N; i += 2)
        if (Check(status[i >> 5], i & 31) == 0)
            prime.push_back(i);
}
```

```
ll SegmentedSieve(ll low, ll high){
    ///Segmented Seive Gives Number of Primes in a Range
    ll i, j, start, ans;
    ans = 0;
    i = 0;
    memset(isPrime, true, sizeof isPrime);
    if (low % 2 == 1) ///For Odd number
        i++;
    for (i; i <= high - low; i = i + 2) isPrime[i] = false;

    for (i = 3; i <= sqrt(high) + 1; i = i + 2) {
        start = max((ll) ceil(low / i), (ll) 1);
        if (start == 1) start++;
        start = start * i;
        if (start < low)    start += i;

        for (j = start - low; j <= high - low; j = j + i)
            isPrime[j] = false;
    }
    if (low == 1) {///If low==1 then 1 is not Prime but 2 is Prime
        isPrime[0] = false;
        isPrime[1] = true;
    }
    if (low == 2) /// If low==2 then 2 is Prime
        isPrime[0] = true;

    for (i = 0; i <= high - low; i++) {///Checking Prime
        if (isPrime[i])
            ans++;
    }
    return ans;
}
```

```
///primality test start

ll mulmod(ll a, ll b, ll mo){
    ll q = ((long double) a * (long double) b / (long double) mo);
    ll res = a * b - mo * q;
    return ((res % mo) + mo) % mo;
}

bool miller(ll a, ll d, ll p){
    ll x = bigmod(a, d, p);
    if (x == 1 || x == p - 1)   return true;
    while (d != p - 1)  {
        x = mulmod(x, x, p);
        d *= 2;
        if (x == 1) return false;
        if (x == p - 1) return true;
    }
    return false;
}


bool isPrimes(ll p){
    if (p < 2)  return false;
    if (p == 2) return true;
    if (p != 2 && p % 2 == 0)   return false;
    ll d = p - 1;
    while (d % 2 == 0)  d = d / 2;
    for (ll i = 1; i < 20; i++){
        ll a = abs(rand() % (p - 2)) + 2;
        if (!miller(a, d, p))   return false;
    }
    return true;
}

///primality test end
```

```
ll SumOfDivisor_UpTo_N(ll N){
    ///calculate SumOfDivisor upto N in Sqrt(N) time
    ll i, j, ans;
    ans = 0;
    for (i = 1; i * i <= N; i++){
        j = N / i;
        /// (Summation Upto J) - (Summmation Upto I-1)
        ans += (((j * (j + 1)) / 2) - (((i - 1) * i) / 2));
        ///  Summation of all i that is not used in previous equation
        ans += ((j - i) * i);
    }
    return ans;
}

ll Sum_Of_No_Of_Divisor_upto_N(ll N){ ///up to n
    ll res = 0;
    ll u = sqrt(N);
    for (ll i = 1; i <= u; i++){
        res += (N / i) - i; //Step 1
    }
    res *= 2; //Step 2
    res += u; //Step 3
    return res;
}

void Phi_O_N(ll N){
    ///calculate coprime upto N in N time
    Phi[1] = 1;
    memset(lp, 0, sizeof lp);
    pr.clear();
    for (ll i = 2; i <= N; ++i){
        if (lp[i] == 0){
            lp[i] = i;
            Phi[i] = i - 1;
            pr.push_back(i);
        }
        else{
            ///Calculating phi
            if (lp[i] == lp[i / lp[i]])
                Phi[i] = Phi[i / lp[i]] * lp[i];
            else
                Phi[i] = Phi[i / lp[i]] * (lp[i] - 1);
        }
        for (ll j = 0; j < (ll) pr.size() && pr[j] <= lp[i] && i * pr[j] <=
N; ++j)
            lp[i * pr[j]] = pr[j];
    }
}
```

```
void ModularInverse_O_N(ll N){
    ///Modular_Multiplicative_Inverse upto N in O(N)
    ///     a * (m / a) + m % a = m
    ///    (a * (m / a) + m % a) mod m = m mod m, or
    ///    (a * (m / a) + m % a) mod m = 0, or
    ///    (- (m % a)) mod m = (a * (m / a)) mod m.
    ///     Dividing both sides by (a * (m % a)), we get
    ///     • inverse(a) mod m = ((m/a) * inverse(m % a)) mod m
    ///     inverse(a) mod m = (- (m/a) * inverse(m % a)) mod m

    modInverse[1] = 1; /// this is you know 1 * 1 mod m = 1
    ll m = MOD;
    for (ll i = 2; i <= N; i++)
        modInverse[i] = (-(m / i) * modInverse[m % i]) % m + m;
}

void SmallestPrimeFactor(ll N){
    ///calculate SmallestPrime upto N in NlogN time
    memset(SmallestPrime, 0, sizeof SmallestPrime);
    for (ll i = 2; i < N; i += 2)
        SmallestPrime[i] = 2; ///even numbers have smallest prime factor 2
    for (ll i = 3; i < N; i += 2){
        if (!SmallestPrime[i]){
            SmallestPrime[i] = i;
            for (ll j = i;(j * i) < N; j += 2){ ///j++ produce even number
                if (!SmallestPrime[j * i])  SmallestPrime[j * i] = i;
            }
        }
    }
}

void NumberOfPrimeFactor(ll x){
    ///first call SmallestPrimeFactor
    ll y;
    y = x;
    memset(PrimeFactor, 0, sizeof PrimeFactor);
    while (x != 1){
        ll p = SmallestPrime[x];
        while (x % p == 0)  x /= p;
        PrimeFactor[y]++;
    }
}

ll gcd;

pii ExtendedEuclid(ll a, ll b){ /// returns x, y | ax + by = gcd(a,b)
    if (a == 0){
        gcd = b;
        return pii(1, 0);
    }
    else{
        pii d = ExtendedEuclid(b, a % b);
        return pii(d.second, d.first - d.second * (a / b));
    }
}
```

```
pii __ExtendedEuclid(ll a, ll b){
    ll x0 = 1, y0 = 0;
    ll x1 = 0, y1 = 1;
    while (b != 0){
        ll q = a / b;
        ll m = a - q * b;
        ll x = x0 - q * x1, y = y0 - q * y1;
        a = b;
        b = m;
        x0 = x1, y0 = y1;
        x1 = x, y1 = y;
    }
    return{
        x0,
        y0
    };
}

ll No_Of_Digits_In_N_Fact_In_Base_B(ll N, ll B){

    ///The number of digits in N factorial is :
    ///floor (ln(n!)/ln(B) + 1)

    ll i;
    double ans = 0;
    for (i = 1; i <= N; i++)
        ans += log(i);
    ans = ans / log(B);
    ans = ans + 1;
    return (ll) ans;

    ///*****Another Way
    ///    for(i=1; i<=N; i++)
    ///        ans+= (log10(i)/log10(B));
    ///    ans=ans+1;
    ///    return (ll)ans;
}
```

```
ll No_Of_Trailing_Zeroes_In_N_Fact_In_Base_B(ll N, ll B){
    ///We can break the Base B as a product of primes :
    /// B = a^p1 * b^p2 * c^p3 * ...
    ///Then the number of trailing zeroes in N factorial in Base B is given
by the formulae
    ///min{1/p1(n/a + n/(a*a) + ..), 1/p2(n/b + n/(b*b) + ..), ..}.
    ll i, j, num, ans, total, m;
    ans = pow(10, 10);
    for (i = 0; prime[i] * prime[i] <= B; i++){
        m = prime[i];
        if (B % m == 0){
            num = 0;
            while (B % m == 0){
                B = B / m;
                num++;
            }
            j = 1;
            total = 0;
            while (floor(N / ceil(pow(m, j))) > 0){
                total += N / ceil(pow(m, j)); ///No. of total i in N!
                j++;
            }
            total = total / num;

            ans = min(ans, total);
        }
    }
    if (B > 1){ ///for last prime
        j = 1;
        total = 0;
        while (N / ceil(pow(B, j)) > 0){
            total += N / ceil(pow(B, j));
            j++;
        }
    }
    ans = min(ans, total);
    return ans;
}

ll leadingDigitFact(ll n, ll k){
    /// Find the first K digits of N!
    double fact = 0;
    for (ll i = 1; i <= n; i++){
        fact += log10(i); /// Find log(N!)
    }

    ///Find the floating part of log(N!) of fact
    double q = fact - floor(fact + eps);
    double B = pow(10, q);

    ///Shift decimal point k-1 times
    for (ll i = 0; i < k - 1; i++){
        B *= 10;
    }
    ///Don't forget to floor it
    return floor(B + eps);
}
```

```
ll ncr[1005][1005];

ll NCR(){
    for (ll i = 0; i < 1003; i++)
        ncr[i][0] = 1;
    for (ll i = 1; i < 1003; i++)
        for (ll j = 1; j < 1003; j++)
            ncr[i][j] = ncr[i - 1][j] + ncr[i - 1][j - 1];
            //ncr[i][j] %=mod;
}

//star and bars theorem
//divide n stars into k partitions, where
//each partition is positive
nCr (n - 1, k - 1);
//non negative
nCr (n + k - 1, k - 1);

//Mobius
const int N = 10000000;
vector <int> primes;
int mu[N + 7], sq = sqrt(N);
void mobius (){
    for (int i = 1; i <= N; i++){
        mu[i] = 1;
    }
    for (int i = 0; i < (int) primes.size() and primes[i] <= sq; i++){
        int x = primes[i] * primes[i];
        for (int j = x; j <= N; j += x){
            mu[j] = 0;
        }
    }
    for (int i = 0; i < (int) primes.size(); i++){
        for (int j = primes[i]; j <= N; j += primes[i]){
            mu[j] *= -1;
        }
    }
}


int main(){
    ll t, T, n, b;
    scanf("%lld", & T);
    seive_N_logN(1000 + 4);
    NCR();
    for (t = 1; t <= T; t++){
        scanf("%lld", & n);
        cout << isPrimes(n) << endl;
    }
    return 0;
}
```

```
//Chinese Remainder Theorem

#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii;
#define x first
#define y second
pii extended_euclid(int a, int b) {
      if (b == 0) {
            return pii(1, 0);
      } else {
            pii d = extended_euclid(b, a % b);
            return pii(d.y, d.x - d.y * (a / b));
      }
}
int modular_Inverse(int a, int n) {
      pii ret = extended_euclid(a, n);
      return ((ret.x % n) + n) % n;
}
/// Chinese Remainder Theorem:
/// Returns the smallest
/// number x such that:
///  x % mod[0] = rem[0],
///  x % mod[1] = rem[1],
///  ..................
///  x % mod[k-1] = rem[k-1]
///  Numbers in mod[] are pairwise co prime.

/// x = MOD[1]*Inv[1]*rem[1] + MOD[2]*Inv[2]*rem[2] + . . . +
MOD[k]*Inv[k]*rem[k];
/// MOD[i] = (mod[1]*mod[2]*....*mod[k])/(mod[i]);
/// Find Inv[i] such that, MOD[i]*Inv[i] == 1(% mod[i]) (using modulo
inverse);

int k; /// k is the size.
int mod[105];
int rem[105];
int CRT(){
    int prod = 1;
    for (int i = 0; i < k; i++){
        prod *= mod[i];
    }
    int result = 0;
    for (int i = 0; i < k; i++){
        int pp = prod / mod[i];
        result += rem[i] * modular_Inverse(pp, mod[i]) * pp;
    }
    return (result % prod);
}
int main(void){
    cin >> k;
    for(int i = 0;i<k;i++) cin >> mod[i];
    for(int i = 0;i<k;i++) cin >> rem[i];
    int x = CRT();
    printf("The x is : %d\n",x);
    return 0;
}
```

Graph Theory

```
//2-SAT

#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

const int maxn = 21000;

struct SCC {
    vector <int> g[maxn];
    int disc[maxn], low[maxn], st[maxn], cycle[maxn], id[maxn];
    int visited[maxn];
    int tail, t, cnt;

    SCC (){
        t = tail = cnt = 0;
        memset (visited, 0, sizeof visited);
    }

    void tarjan (int u){
        disc[u] = low[u] = t++;
        st[tail++] = u;
        visited[u] = 1;
        for (int i = 0; i < (int) g[u].size(); i++){
            int v = g[u][i];
            if (visited[v] == 0){
                tarjan (v);
                low[u] = min (low[u], low[v]);
            }
            else if (visited[v] == 1){
                low[u] = min (low[u], low[v]);
            }
        }
        if (low[u] == disc[u]){
            cnt++;
            while (1){
                int v = st[tail - 1];
                tail--;
                visited[v] = 2;
                cycle[v] = u;
                id[v] = cnt;
                if (u == v) break;
            }
            id[u] = cnt;
        }
    }

    void findSCC (int n){
        for (int i = 1; i <= n; i++){
            if (visited[i] == 0)
```

```
                tarjan (i);
            }
        }
};
```

```
struct TwoSat{
    int n;
    SCC scc;
    vector <int> res;
    inline int Not (int a){
        if (a > n) return a - n;
        return n + a;
    }

    void mustTrue (int a){
        scc.g[Not (a)].push_back (a);
    }

    void xorClause (int a, int b){
        scc.g[a].push_back (Not (b));
        scc.g[Not (a)].push_back (b);
        scc.g[b].push_back (Not (a));
        scc.g[Not (b)].push_back (a);
    }

    void orClause (int a, int b){
        scc.g[Not (a)].push_back (b);
        scc.g[Not (b)].push_back (a);
    }

    void andClause (int a, int b){
        mustTrue (a);
        mustTrue (b);
    }

    bool possible (){
        scc.findSCC (n + n);
        for (int i = 1; i <= n; i++){
            if (scc.cycle[i] == scc.cycle[Not (i)])
                return false;
        }
        return true;
    }

    void findSAT (){
        for (int i = 1; i <= n; i++){
            if (scc.id[i] < scc.id[Not (i)]){
                res.push_back (i);
            }
        }
    }
};
```

```
//Articulation Point Bridge

const int maxn = 100005;
const int V = 100000;
int dfsRoot, rootChildren;
vector <int> dfs_num,dfs_low,dfs_parent,articulation_vertex;
int dfsNumberCounter;
vector <pair <int, int> > AdjList[maxn];

void articulationPointAndBridge(int u)
{
    dfs_low[u] = dfs_num[u] = dfsNumberCounter++;
    for (int j = 0; j < (int)AdjList[u].size(); j++)
    {
        pair <int, int> v = AdjList[u][j];
        if (dfs_num[v.first] == false)
        {
            dfs_parent[v.first] = u;
            if (u == dfsRoot)
                rootChildren++;
            articulationPointAndBridge(v.first);
            if (dfs_low[v.first] >= dfs_num[u])
                articulation_vertex[u] = true;
            if (dfs_low[v.first] > dfs_num[u])
                printf(" Edge (%d, %d) is a bridge\n", u, v.first);
            dfs_low[u] = min(dfs_low[u], dfs_low[v.first]);
        }
        else if (v.first != dfs_parent[u])
            dfs_low[u] = min(dfs_low[u], dfs_num[v.first]);
    }
}
```

```
//BPM

ll n,m;
vector<ll> g[55];
ll lt[55];
ll rt[55];
bool visited[55];

bool dfs(ll u){
    if(visited[u]) return false;
    visited[u]=true;
    for(ll i=0;i<g[u].size();i++){
        ll v=g[u][i];
        if(rt[v]==-1){
            rt[v]=u;
            lt[u]=v;
            return true;
        }
    }
    for(ll i=0;i<g[u].size();i++){
        ll v=g[u][i];
        if(dfs(rt[v])){
            rt[v]=u;
            lt[u]=v;
            return true;
        }
    }
    return false;
}

ll match(){
    memset(lt,-1,sizeof lt);
    memset(rt,-1,sizeof rt);
    bool done=false;
    while(!done){
        done=true;
        memset(visited,false,sizeof visited);
        for(ll i=1;i<=n;i++){
            if(lt[i]==-1 and dfs(i)){
                done=false;
            }
        }
    }

    ll ret=0;
    for(ll i=1;i<=n;i++) ret+=(int)lt[i]!=-1;
    return ret;
}
```

```
//Centroid Decomposition

#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mx=100005;

ll n;
vector<ll> adj[mx];
char ans[mx];
bool brk[mx];
ll subsize[mx];

void calculatesize(ll u,ll par)
{
	subsize[u]=1;
	for(ll i=0;i<(ll)adj[u].size();i++){
		ll v=adj[u][i];
		if(v==par or brk[v]==true)continue;
		calculatesize(v,u);
		subsize[u]+=subsize[v];
	}
}

ll getcentroid(ll u,ll par,ll n){
	ll ret=u;
	for(ll i=0;i<(ll)adj[u].size();i++){
		ll v=adj[u][i];

		if(v==par or brk[v]==true)continue;
		if(subsize[v]>(n/2)){
			ret=getcentroid(v,u,n);
			break;
		}
	}
	return ret;
}

void decompose(ll u,char rank){


	calculatesize(u,-1);

	ll c=getcentroid(u,-1,subsize[u]);
	brk[c]=true;
	ans[c]=rank;
	for(ll i=0;i<(ll)adj[c].size();i++){
		ll v=adj[c][i];
		if(brk[v]==true)continue;
		decompose(v,rank+1);
	}
```

```
}



int main(){
	scanf("%lld",&n);
	for(ll i=0;i<n-1;i++){
		ll a,b;
		scanf("%lld %lld",&a,&b);
		adj[a].push_back(b);
		adj[b].push_back(a);
	}
	decompose(1,'A');

	for(ll i=1;i<=n;i++){
		printf("%c ",ans[i]);
	}
}
```

```
//Dinic's Maxflow


#include <bits/stdc++.h>
using namespace std;

const int maxn = 1003;
const int inf = 100000000;

class edge{
public:
    int a, b, cap, flow;
    edge (int _a, int _b, int _cap, int _flow){
        a = _a; b = _b, cap = _cap; flow = _flow;
    }
};

vector <edge> e;
vector <int> g[maxn];
int s, t, d[maxn], q[maxn], ptr[maxn];

void add_edge(int a, int b, int cap){
    edge e1 = edge (a, b, cap, 0);
    edge e2 = edge (b, a, 0, 0);
    g[a].push_back ((int)e.size());
    e.push_back (e1);
    g[b].push_back ((int)e.size());
    e.push_back (e2);
}

bool bfs (){
    int qh = 0, qt = 0;
    q[qt++] = s;
    memset (d, -1, sizeof d);
    d[s] = 0;
    while (qh < qt and d[t] == -1){
        int v = q[qh++];
        for (int i = 0; i < (int)g[v].size(); i++){
            int id = g[v][i];
            int to = e[id].b;
            if (d[to] == -1 and e[id].flow < e[id].cap){
                q[qt++] = to;
                d[to] = d[v] + 1;
            }
        }
    }
    return d[t] != -1;
}
```

```
int dfs (int v, int flow){
    if (!flow) return 0;
    if (v == t) return flow;
    for (;ptr[v] < (int) g[v].size(); ptr[v]++){
        int id = g[v][ptr[v]], to = e[id].b;
        if (d[to] != d[v] + 1) continue;
        int pushed = dfs (to, min (flow, e[id].cap - e[id].flow));
        if (pushed){
            e[id].flow += pushed;
            e[id ^ 1].flow -= pushed;
            return pushed;
        }
    }
    return 0;
}

int dinic (){
    int flow = 0;
    while (1){
        if (!bfs()) break;
        memset (ptr, 0, sizeof ptr);
        while (int pushed = dfs(s, inf)){
            flow += pushed;
            if (pushed == 0) break;
        }
    }
    return flow;
}
```

//Directed MST

```
// Directed minimum spanning tree
//
// Given a directed weighted graph and root node, computes the minimum
spanning
// directed tree (arborescence) on it.
//
// Complexity: O(N * M), where N is the number of nodes, and M the number of
edges

struct Edge { int x, y, w; };

int dmst(int N, vector<Edge> E, int root) {
  const int oo = 1e9;

  vector<int> cost(N), back(N), label(N), bio(N);
  int ret = 0;

  for (;;) {
    REP(i, N) cost[i] = oo;
    for (auto e : E) {
      if (e.x == e.y) continue;
      if (e.w < cost[e.y]) cost[e.y] = e.w, back[e.y] = e.x;
    }
    cost[root] = 0;
    REP(i, N) if (cost[i] == oo) return -1;
    REP(i, N) ret += cost[i];

    int K = 0;
    REP(i, N) label[i] = -1;
    REP(i, N) bio[i] = -1;
    REP(i, N) {
      int x = i;
      for (; x != root && bio[x] == -1; x = back[x]) bio[x] = i;

      if (x != root && bio[x] == i) {
        for (; label[x] == -1; x = back[x]) label[x] = K;
        ++K;
      }
    }
    if (K == 0) break;
    REP(i, N) if (label[i] == -1) label[i] = K++;
    for (auto &e : E) {
      int xx = label[e.x];
      int yy = label[e.y];
      if (xx != yy) e.w -= cost[e.y];
      e.x = xx;
      e.y = yy;
```

```
    }

    root = label[root];
    N = K;
  }

  return ret;
}
//HLD


///node count: 0 to n - 1
#include<bits/stdc++.h>
using namespace std;

const int maxn = 500050;

int n, ptr, chainno;
vector <int> adj[maxn];
int level[maxn];
int sparse[maxn][20];
int subsize[maxn];
int chainid[maxn];
int chainhead[maxn];
int base[maxn];
int posbase[maxn];
int tree[maxn*6];
int lazy[maxn*6];

void clean(){
    for(int i = 0; i < maxn; i++){
        adj[i].clear();
        chainid[i] = -1;
        chainhead[i] = -1;
        base[i] = 0;
        level[i] = -1;
        posbase[i] = -1;
        subsize[i] = 0;
    }
    memset(sparse, -1, sizeof sparse);
    memset(tree, 0, sizeof tree);
    ptr = 1;
    chainno = 1;
}

void dfs(int u, int p, int depth){
    sparse[u][0] = p;
    level[u] = depth;
    subsize[u] = 1;
    for(int i = 0; i < adj[u].size(); i++){
        int v = adj[u][i];
        if(v == p) continue;
        dfs(v, u, depth + 1);
        subsize[u] += subsize[v];
    }
}
```

```
void lca_init(){
    for(int j = 1; (1<<j) < n; j++){
        for(int i = 0; i < n; i++){
            if(sparse[i][j - 1] != -1) sparse[i][j] = sparse[sparse[i][j-
1]][j-1];
        }
    }
}

int query_lca(int p, int q){
    if(level[p] < level[q]) swap(p,q);
    int log = 0;
    while((1<<log) <= level[p]) log++;
    log--;
    for(int i = log; i >= 0; i--){
        if(level[p] - (1<<i) >= level[q]){
            p = sparse[p][i];
        }
    }
    if(p == q) return p;
    for(int i = log; i >= 0; i--){
        if(sparse[p][i] != -1 and sparse[p][i] != sparse[q][i]){
            p = sparse[p][i];
            q = sparse[q][i];
        }
    }
    return sparse[p][0];
}


void hld(int cur, int p, int cst){
    if(chainhead[chainno] == -1)
        chainhead[chainno] = cur;
    chainid[cur] = chainno;
    posbase[cur] = ptr;
    base[ptr++] = cst;

    int sc = -1;
    int mx = -10000000000;

    for(int i = 0; i < adj[cur].size(); i++){
        int v = adj[cur][i];
        if(v == p) continue;
        if(sc == -1 or subsize[sc] < subsize[v]){
            sc = v;
            mx = 0;/// here goes cost,for this problem it is 0. i.e.
cost[cur][i];
        }
    }

    if(sc != -1) hld(sc, cur, mx);

    for(int i = 0; i < adj[cur].size(); i++){
        int v = adj[cur][i];
        if(v == p or v == sc) continue;
        chainno++;
        int cst = 0; /// same
```

```
        hld(v, cur, cst);
    }
}


void build(int nd, int b, int e){
    if(b == e){
        tree[nd] = base[b];
        lazy[nd] = 0;
        return;
    }
    int lnd = nd * 2;
    int rnd = lnd + 1;
    int mid = (b + e) / 2;
    build(lnd, b, mid);
    build(rnd, mid + 1, e);
    tree[nd] = 0;
    lazy[nd] = 0;
}

inline void push_down(int nd, int b, int e){
    tree[nd] += (e - b + 1) * lazy[nd];
    if(b != e){
        lazy[nd*2] += lazy[nd];
        lazy[nd*2 + 1] += lazy[nd];
    }
    lazy[nd] = 0;
}

int query_tree(int nd, int b, int e, int i, int j){
    if(lazy[nd] != 0) push_down(nd, b, e);
    if(i > e or j < b) return 0;
    if(i <= b and e <= j) return tree[nd];
    int lnd = nd * 2;
    int rnd = lnd + 1;
    int mid = (b + e) / 2;
    int ret1 = query_tree(lnd, b, mid, i, j);
    int ret2 = query_tree(rnd, mid + 1, e, i, j);
    return ret1 + ret2;
}

int query_up(int u, int v){
    int vchain = chainid[v];
    int uchain;
    int ret = 0;
    while(1){
        uchain = chainid[u];
        if(uchain == vchain){
            ///if (v == u) break;
            ret += query_tree(1, 1, ptr, posbase[v], posbase[u]);
            break;
        }
        ret += query_tree(1, 1, ptr, posbase[chainhead[uchain]], posbase[u]);
        u = chainhead[uchain];
        u = sparse[u][0];
    }
    return ret;
```

```
}

int query(int u,int v){
    int lca = query_lca(u,v);
    int ret1 = query_up(u,lca);
    int ret2 = query_up(v,lca);
    int ret3 = query_up(lca,lca);
    return ret1 + ret2 - ret3;
}

void update_tree(int nd, int b, int e, int i, int j, int c){
    if(lazy[nd] != 0) push_down(nd, b, e);
    if(i > e or j < b) return;
    if(i <= b and e <= j){
        lazy[nd] +=c;
        push_down(nd, b, e);
        return ;
    }
    int lnd = nd * 2;
    int rnd = lnd + 1;
    int mid = (b + e) / 2;
    update_tree(lnd, b, mid, i, j, c);
    update_tree(rnd, mid + 1, e, i, j, c);
}

void update_up(int u, int v, int c){
    int vchain = chainid[v];
    int uchain;
    while(1){
        uchain = chainid[u];
        if(uchain == vchain){
            ///if (u == v) break;
            update_tree(1, 1, ptr, posbase[v], posbase[u], c);
            break;
        }
        update_tree(1, 1, ptr, posbase[chainhead[uchain]], posbase[u], c);
        u = chainhead[uchain];
        u = sparse[u][0];
    }
}
void update(int u, int v, int c){
    int lca = query_lca(u, v);
    update_up(v, lca, c);
    update_up(u, lca, c);
    update_up(lca, lca, -c);
}

int main(){
//    freopen("input.txt","r",stdin);

    int t, u, v, c;
    scanf("%d", &t);
    for(int ts = 1; ts <= t; ts++){
        clean();
        scanf("%d", &n);
        for(int i = 0; i < n-1; i++){
            scanf("%d %d", &u, &v);
```

```
            adj[u].push_back(v);
            adj[v].push_back(u);
        }

        dfs(0,-1,1);
        lca_init();
        hld(0,-1,0);
        build(1, 1, ptr);
        int q;
        scanf("%d", &q);
        while(q--){
            scanf("%d %d %d", &u, &v, &c);
            update(u, v, c);
        }
        printf("Case #%d:\n",ts);
        for(int i = 0; i < n; i++){
            printf("%d\n",query(i,i));
        }
    }
}
```

```
//Kth Best Shortest Path


int m, n, deg[MM], source, sink, K, val[MM][12];
struct edge {
	int v, w;
}adj[MM][500];
struct info {
	int v, w, k;
	bool operator < ( const info &b ) const {
		return w > b.w;
	}
};
priority_queue < info, vector <info> > Q;
void kthBestShortestPath() {
	int i, j;
	info u, v;
	for( i = 0; i < n; i++ ) for( j = 0; j < K; j++ ) val[i][j] = inf;
	u.v = source; u.k = 0; u.w = 0;
	Q.push(u);
	while( !Q.empty() ) {
		u = Q.top();
		Q.pop();
		for( i = 0; i < deg[u.v]; i++ ) {
			v.v = adj[u.v][i].v;
			int cost = adj[u.v][i].w + u.w;
			for( v.k = u.k; v.k < K; v.k++ ) {
				if( cost == inf ) break;
				if( val[v.v][v.k] > cost ) {
					swap( cost, val[v.v][v.k] );
					v.w = val[v.v][v.k];
					Q.push(v);
					break;
				}
			}
			for( v.k++; v.k < K; v.k++ ) {
				if( cost == inf ) break;
				if( val[v.v][v.k] > cost ) swap( cost, val[v.v][v.k]
);
			}
		}
	}
}
```

//LCA on MST

```
///LCA on MST

const ll maxn = 100005;
const ll inf = 100000000000;
ll n, m, q;
vector <pair <ll, pair <ll, ll> > > edgelist;
vector <pair <ll, ll> > g[maxn];
ll parent[maxn];
ll sparse[maxn][18];
ll minsparse[maxn][18];
ll parentcost[maxn];
ll level[maxn];

ll findparent (ll a){
    if (parent[a] == a) return a;
    return parent[a] = findparent(parent[a]);
}

void makeset (ll a, ll b){
    ll pa = findparent(a);
    ll pb = findparent(b);
    parent[pa] = pb;
}

void dfs (ll u, ll par,ll d){
    level[u] = d;
    for (ll i = 0; i < g[u].size(); i++){
        ll k = g[u][i].first;
        if (k != par){

            parent[k] = u;
            parentcost[k] = g[u][i].second;
            dfs (k, u,d+1);
        }
    }
}

void build(){
    for(int i = 1; i <= n; i++){
        sparse[i][0] = parent[i];
        minsparse[i][0] = parentcost[i];
    }

    for(int j = 1; (1 << j) <=n; j++){
        for(int i = 1; i <=n; i++){
            if(sparse[i][j-1] != -1){
```

```
                minsparse[i][j] = min(minsparse[i][j-
1],minsparse[sparse[i][j-1]][j-1]);
                sparse[i][j] = sparse[sparse[i][j-1]][j-1];
            }
        }
    }
}


ll  query(ll p, ll q){
    if(level[p] < level[q]){
        swap(p,q);
    }

    ll log = 1;
    ll ans = inf;
    while( (1 << log) <= level[p]) log++;log--;
    for(ll i = log; i >=0 ; i--){
        if(level[p] - (1LL << i) >= level[q]){
            ans = min(minsparse[p][i],ans);
            p = sparse[p][i];
        }
    }

    if(p == q) return ans;

    for(ll i = log; i >= 0; i--){
        if(sparse[p][i] != -1 && (sparse[p][i] != sparse[q][i])){
            ans = min(ans,minsparse[p][i]);
            ans = min(ans,minsparse[q][i]);
            p = sparse[p][i];
            q = sparse[q][i];
        }
    }

    return min(minsparse[p][0],min(minsparse[q][0],ans));

}
int main (){
//    filein;
    for(int i = 0; i < maxn; i++){
        for(int j = 0; j < 18; j++){
            minsparse[i][j] = inf;
            sparse[i][j] = -1;
        }
    }

    scanf("%lld %lld %lld", &n, &m, &q);
    for (ll i = 0; i < m; i++){
        ll u, v, c;
        scanf("%lld %lld %lld", &u, &v, &c);
        edgelist.push_back (make_pair (c, make_pair (u, v)));
    }
    sort (edgelist.rbegin(), edgelist.rend());
    for (ll i = 1; i <= maxn; i++)
        parent[i] = i;
    for (ll i = 0; i < edgelist.size(); i++){
```

```
        ll c = edgelist[i].first;
        ll u = edgelist[i].second.first;
        ll v = edgelist[i].second.second;
        ll pu = findparent(u);
        ll pv = findparent(v);
        if (pu != pv){
            g[u].push_back (make_pair (v, c));
            g[v].push_back (make_pair (u, c));
           //cout << u << " " << v  << " " << c << endl;
            makeset (pu, pv);
        }
    }

    parent[1] = -1;
    parentcost[1] = 0;
    dfs (1, 0,1);
    build();

    while(q--){
       ll a,b;
       scanf("%lld %lld",&a,&b);

       ll ans = query(a,b);

       printf("%lld\n",ans);
    }


    return 0;
}
```

```
//DSU on Tree


int sz[maxn];
void getsz(int v, int p){
    sz[v] = 1;  // every vertex has itself in its subtree
    for(auto u : g[v])
        if(u != p){
            getsz(u, v);
            sz[v] += sz[u]; // add size of child u to its parent(v)
        }
}


//SACK O(nlog^2n)

map<int, int> *cnt[maxn];
void dfs(int v, int p){
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p){
           dfs(u, v);
           if(sz[u] > mx)
               mx = sz[u], bigChild = u;
       }
    if(bigChild != -1)
        cnt[v] = cnt[bigChild];
    else
        cnt[v] = new map<int, int> ();
    (*cnt[v])[ col[v] ] ++;
    for(auto u : g[v])
       if(u != p && u != bigChild){
           for(auto x : *cnt[u])
               (*cnt[v])[x.first] += x.second;
       }
    //now (*cnt[v])[c] is the number of vertices in subtree of vertex v that
has color c. You can answer the queries easily.

}

//SACK - O(nlogn)

vector<int> *vec[maxn];
int cnt[maxn];
void dfs(int v, int p, bool keep){
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p && sz[u] > mx)
```

```
            mx = sz[u], bigChild = u;
    for(auto u : g[v])
       if(u != p && u != bigChild)
            dfs(u, v, 0);
    if(bigChild != -1)
        dfs(bigChild, v, 1), vec[v] = vec[bigChild];
    else
        vec[v] = new vector<int> ();
    vec[v]->push_back(v);
    cnt[ col[v] ]++;
    for(auto u : g[v])
       if(u != p && u != bigChild)
            for(auto x : *vec[u]){
                cnt[ col[x] ]++;
                vec[v] -> push_back(x);
            }
    //now (*cnt[v])[c] is the number of vertices in subtree of vertex v that
has color c. You can answer the queries easily.
    // note that in this step *vec[v] contains all of the subtree of vertex
v.
    if(keep == 0)
        for(auto u : *vec[v])
            cnt[ col[u] ]--;
}
```

```
//SCC Kosaraju


#include <bits/stdc++.h>
using namespace std;

typedef long long int lli;

#define FILE_IN freopen("in.txt","r",stdin);
#define FILE_OUT freopen("out.txt","w",stdout);
#define FILE_IO FILE_IN;FILE_OUT;
#define FAST_IO ios::sync_with_stdio(false);cin.tie(NULL);

vector <lli> adj_list[100000 + 10];
vector <lli> adj_list_reversed[100000 + 10];
lli cost[100000 + 10];
bool visited1[100000 + 10];
bool visited2[100000 + 10];
lli begin_time[100000 + 10];
lli finish_time[100000 + 10];
vector <lli> scc[100000 + 10];

#define MOD 1000000007
vector <pair <lli,lli> > v;

lli t;


void dfs1(lli src){
   if(visited1[src])
      return;

   begin_time[src] = ++t;
   visited1[src] = true;
   for(lli i = 0; i < adj_list[src].size(); i++)
      dfs1(adj_list[src][i]);

   finish_time[src] = ++t;
}

void dfs2(lli src,lli p){
   if(visited2[src])
      return;

   visited2[src] = true;
   scc[p].push_back(src);
```

```
    for(lli i = 0; i < adj_list_reversed[src].size(); i++){
        dfs2(adj_list_reversed[src][i],p);
    }
}

int main(){
    lli n,a,b,m;
    scanf("%lld",&n);

    for(lli i_n = 1; i_n <= n; i_n++){
        scanf("%lld",&cost[i_n]);
    }

    scanf("%lld",&m);

    for(lli i_m = 0; i_m < m; i_m++){
        scanf("%lld %lld",&a,&b);

        adj_list[a].push_back(b);
        adj_list_reversed[b].push_back(a);

    }

    for(lli i_n = 1; i_n <= n; i_n++){
        if(visited1[i_n] == false)
            dfs1(i_n);
    }

    for(lli i_n = 1; i_n <= n; i_n++){
        v.push_back(make_pair(finish_time[i_n],i_n));
    }

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

    lli number = 1;

    for(lli i_n = v.size() - 1; i_n >= 0; i_n--){
        if(visited2[v[i_n].second] == false){
            dfs2(v[i_n].second,number++);
        }
    }
}
```

//Stable Marriage

```
/* A person has an integer preference for each of the persons of the opposite
 * sex, produces a matching of each man to some woman. The matching will
follow:
 *          - Each man is assigned to a different woman (n must be at least
m)
 *          - No two couples M1W1 and M2W2 will be unstable.
 * Two couples are unstable if (M1 prefers W2 over W1 and W1 prefers M2 over
M1)
 * INPUT: m - number of man, n - number of woman (must be at least as large
as m)
 *          - L[i][]: the list of women in order of decreasing preference of
man i
 *          - R[j][i]: the attractiveness of i to j.
 * OUTPUTS: - L2R[]: the mate of man i (always between 0 and n-1)
 *          - R2L[]: the mate of woman j (or -1 if single)        */
int m, n, L[MAXM][MAXW], R[MAXW][MAXM], L2R[MAXM], R2L[MAXW], p[MAXM];
void stableMarriage() {
      memset( R2L, -1, sizeof( R2L ) );
      memset( p, 0, sizeof( p ) );
      for( int i = 0; i < m; i++ ) { // Each man proposes...
            int man = i;
            while( man >= 0 ) {
                  int wom;
                  while( 1 ) {
                        wom = L[man][p[man]++];
                        if( R2L[wom] < 0 || R[wom][man] > R[wom][R2L[wom]] )
break;
                  }
                  int hubby = R2L[wom];
                  R2L[L2R[man] = wom] = man;
                  man = hubby;
            }
      }
}
```

Data Structures

```
//2D Segment Tree
//Needs O(16nm) memory!!!
const int maxn = 2001;
int tree[4*maxn][4*maxn], n, m, arr[maxn][maxn];
void buildY(int ndx, int lx, int rx, int ndy, int ly, int ry) {
	if(ly == ry) {
		if(lx == rx) tree[ndx][ndy] = arr[lx][ly];
		else tree[ndx][ndy] = tree[ndx*2][ndy] + tree[ndx*2+1][ndy];
		return;
	} int mid = ly + ry >> 1;
	buildY(ndx, lx, rx, ndy*2, ly, mid);
	buildY(ndx, lx, rx, ndy*2+1, mid+1, ry);
	tree[ndx][ndy] = tree[ndx][ndy*2] + tree[ndx][ndy*2+1];
}
void buildX(int ndx, int lx, int rx) {
	if(lx != rx) {
		int mid = lx + rx >> 1;
		buildX(ndx*2, lx, mid);
		buildX(ndx*2+1, mid+1, rx);
	} buildY(ndx, lx, rx, 1, 0, m-1);
}
void updateY(int ndx, int lx, int rx, int ndy, int ly, int ry, int y, int
val) {
	if(ly == ry) {
		if(lx == rx) tree[ndx][ndy] = val;
		else tree[ndx][ndy] = tree[ndx*2][ndy] + tree[ndx*2+1][ndy];
		return;
	} int mid = ly + ry >> 1;
	if(y <= mid) updateY(ndx, lx, rx, ndy*2, ly, mid, y, val);
	else updateY(ndx, lx, rx, ndy*2+1, mid+1, ry, y, val);
	tree[ndx][ndy] = tree[ndx][ndy*2] + tree[ndx][ndy*2+1];
}
void updateX(int ndx, int lx, int rx, int x, int y, int val) {
	if(lx != rx) {
		int mid = lx + rx >> 1;
		if(x <= mid) updateX(ndx*2, lx, mid, x, y, val);
		else updateX(ndx*2+1, mid+1, rx, x,y, val);
	} updateY(ndx, lx, rx, 1, 0, m-1, y,val);
}
int queryY(int ndx, int ndy, int ly, int ry, int y1, int y2) {
	if(ry < y1 || ly > y2) return 0;
	if(y1 <= ly && ry <= y2)
		return tree[ndx][ndy];
	int mid = ly + ry >> 1;
	return queryY(ndx, ndy*2, ly, mid, y1, y2) +
		queryY(ndx, ndy*2+1, mid+1, ry, y1, y2);
}
```

```
int queryX(int ndx, int lx, int rx, int x1, int y1, int x2, int y2) {
	if(rx < x1 || lx > x2) return 0;
	if(x1 <= lx && rx <= x2) {
		return queryY(ndx, 1, 0, m-1, y1, y2);
	} int mid = lx + rx >> 1;
	return queryX(ndx*2, lx, mid, x1,y1,x2,y2) +
		queryX(ndx*2+1, mid+1, rx, x1,y1,x2,y2);
}


//BIT

#include <bits/stdc++.h>
using namespace std;

const int maxn = 100005;
int tree[maxn];

void update (int idx, int val){
	while (idx <= n){
		tree[idx] += val;
		idx += idx & (-idx);
	}
}

int query (int idx){
	int sum = 0;
	while (idx > 0){
		sum += tree[idx];
		idx -= idx & (-idx);
	}
	return sum;
}
```

```
//Histogram

#include<bits/stdc++.h>
using namespace std;
typedef long long ll ;

ll ara[30005];
ll n;

ll histogram(){
  ll area=0;
  stack<int> st;
  ll i;
  for(i=0;i<n;i++){
    if(st.empty() || ara[st.top()]<=ara[i]) st.push(i);
    else{
    while(!st.empty() && ara[st.top()]>ara[i]){
      ll t=st.top();
      st.pop();
      ll tmarea=ara[t]*(st.empty() ? i: i-st.top()-1);
      area=max(area,tmarea);
    }
    st.push(i);
  }

  }
  while(!st.empty()){
    ll t=st.top();
    st.pop();
    ll tmarea=ara[t]*(st.empty()? i: i-st.top()-1);
    area=max(area,tmarea);
  }
  return area;
}

int main()
{
  //freopen("input.txt","r",stdin);
  ll t;
  scanf("%lld",&t);
  for(ll ts=1;ts<=t;ts++){

    scanf("%lld",&n);
    for(int i=0;i<n;i++) scanf("%lld",&ara[i]);
    printf("Case %lld: %lld\n",ts,histogram());
  }
```

```
}
```

```
//Merge Sort Tree

#include <bits/stdtr1c++.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

int n, ar[MAX];
vector<int> tree[MAX << 2];

void merge_sort_tree(int idx, int a, int b, int* ar){ /// hash = 974987
    int p = idx << 1, q = p | 1, c = (a + b) >> 1, d = c + 1;
    int i = 0, j = 0, k = 0, u = c - a + 1, v = b - d + 1, len = b - a + 1;

    tree[idx].resize(len, 0);
    if (a == b){
        tree[idx][0] = ar[a];
        return;
    }

    merge_sort_tree(p, a, c, ar);
    merge_sort_tree(q, d, b, ar);
    while (len--){
        if (i == u) tree[idx][k++] = tree[q][j++];
        else if (j == v) tree[idx][k++] = tree[p][i++];
        else if (tree[p][i] < tree[q][j]) tree[idx][k++] = tree[p][i++];
        else tree[idx][k++] = tree[q][j++];
    }
}

/// Count of numbers <= k in the segment l-r
inline int query(int idx, int a, int b, int l, int r, int k){ /// hash =
476541
    int p = idx << 1, q = p | 1;
    int c = (a + b) >> 1, d = c + 1;

    if (a == l && b == r){
        if (tree[idx][0] > k) return 0;
        else return upper_bound(tree[idx].begin(), tree[idx].end(), k) -
tree[idx].begin();
```

```
    }
    if (r <= c) return query(p, a, c, l, r, k);
    else if (l >= d) return query(q, d, b, l, r, k);
    else return query(p, a, c, l, c, k) + query(q, d, b, d, r, k);
}

int main(){

}
```

//MO's Algorithm with Update

```
const int maxn = 1e5 + 10;
int n, m, a[maxn], prv[maxn], ans[maxn], block;

struct query {
      int l, r, id, t, blcl, blcr;
      query(int _a, int _b, int _c, int _d) {
            l = _a, r = _b;
            id = _c, t = _d;
            blcl = l / block;
            blcr = r / block;
      }
      bool operator < (const query &p) const {
            if(blcl != p.blcl) return l < p.l;
            if(blcr != p.blcr) return r < p.r;
            return t < p.t;
      }
}; vector<query> q;

struct update {
      int pos, pre, now;
}; vector<update> u;


struct lol {
      void add(int x) {}
      void remove(int x) {}
      int get() {}
} ds;

int l, r, t;
int cnt[maxn * 2];

void add(int x) { // Add a[x] to ds
}
void remove(int x) { // Remove a[x] from ds
}
void apply(int i, int x) { // Change a[i] to x
      if(l <= i && i <= r) {
            remove(i);
            a[i] = x;
            add(i);
      } else a[i] = x;
}
```

```
int main(int argc, char const *argv[]) {
	read(n); read(m);
	block = pow(n, 0.6667);

	for(int i = 0; i < n; ++i)
		read(a[i]), last[i] = a[i];

	u.push_back({-1, -1, -1});

	for(int i = 0; i < m; ++i) {
		int t, l, r;
		read(t); read(l); read(r);
		if(t == 1) { --r;
			q.push_back(query(l, r, q.size(), u.size() - 1));
		} else {
			u.push_back({l, prv[l], r});
			prv[l] = r;
		}
	}

	sort(q.begin(), q.end());
	l = 0, r = -1, t = 0;

	for(int i = 0; i < q.size(); i++) {
		while(t < q[i].t) ++t, apply(u[t].pos, u[t].now);
		while(t > q[i].t) apply(u[t].pos, u[t].pre), --t;

		while(r < q[i].r) add(++r);
		while(l > q[i].l) add(--l);
		while(r > q[i].r) remove(r--);
		while(l < q[i].l) remove(l++);

		ans[q[i].id] = ds.get();
	}

	for(int i = 0; i < q.size(); i++)
		printf("%d\n", ans[i]);
}
```

```
//MO's Algorithm

const int maxn = 1e5+10;
int n, Q, block, arr[maxn], cnt[maxn], ans[maxn];
struct query{
	int l, r, id;
	bool operator < (const query &q) const {
		int a = l/block, b = q.l/block;
		return a == b ? r < q.r : a < b;
	}
} q[maxn];
int curr = 0;

void add(int i) {
	int x = arr[i];
	cnt[x]++;
	if(cnt[x] == 1) curr++;
}
void remove(int i) {
	int x = arr[i];
	cnt[x]--;
	if(cnt[x] == 0) curr--;
}
void MO() {
	block = sqrt(n) + 1;
	sort(q, q + m);
	int l = 0, r = -1; curr = 0;
	for(int i = 0; i < m; i++) {
		while(l > q[i].l) add(--l);
		while(r < q[i].r) add(++r);
		while(l < q[i].l) remove(l++);
		while(r > q[i].r) remove(r--);
		ans[q[i].id] = curr;
	}
}
```

```cpp
//Policy Based Data Structure

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

#include <ext/pb_ds/detail/standard_policies.hpp>
#include<bits/stdc++.h>


using namespace __gnu_pbds;
using namespace std;

typedef
tree<int,null_type,std::less<int>,rb_tree_tag,tree_order_statistics_node_upda
te> ordered_set;
int main(){

    ordered_set X;
    X.insert(1);
    X.insert(2);
    X.insert(4);
    X.insert(8);
    X.insert(16);

    cout<<*X.find_by_order(1)<<endl; // 2
    cout<<*X.find_by_order(2)<<endl; // 4
    cout<<*X.find_by_order(4)<<endl; // 16
    cout<<(end(X)==X.find_by_order(6))<<endl; // true
}
```

```
//Persistent Segment Tree

/*////////////////////////////////////////
Persistent Segment Tree
To update in a version first copy the root,
then make update in the version. Something like this -
root[y] = root[x];
update(root[y], ...)
/*////////////////////////////////////////

const int maxn = 1e5+10;
struct node {
	int l, r, sum;
} t[maxn * 20];
int root[maxn], a[maxn], n, m, k, idx, M;

void update(int &nd, int l, int r, int &i) {
	t[++idx] = t[nd];
	++t[nd = idx].sum; //  += v;
	if(l == r) return;
	int m = l + r >> 1;
	if(i <= m) update(t[nd].l, l, m, i);
	else update(t[nd].r, m + 1, r, i);
}

int query(int nd, int l, int r, int &i, int &j) {
	if(r < i || l > j) return 0;
	if(i <= l && r <= j) return t[nd].sum;
	int m = l + r >> 1;
	return query(t[nd].l, l, m, i, j) + query(t[nd].r, m + 1, r, i, j);
}
// a = root[r], b = root[l - 1]
int lesscnt(int a, int b, int l, int r, int k) {
	if(r <= k) return t[a].sum - t[b].sum;
	int m = l + r >> 1;
	if(k <= m) return lesscnt(t[a].l, t[b].l, l, m, k);
	else return lesscnt(t[a].l, t[b].l, l, m, k) +
			lesscnt(t[a].r, t[b].r, m + 1, r, k);
}
int kthnum(int a, int b, int l, int r, int k) {
	if(l == r) return l;
	int cnt = t[t[a].l].sum - t[t[b].l].sum;
	int m = l + r >> 1;
	if(cnt >= k) return kthnum(t[a].l, t[b].l, l, m, k);
	else return kthnum(t[a].r, t[b].r, m + 1, r, k - cnt);
}
void init(int v = 1) {
	t[0].l = t[0].r = t[0].sum = 0;
```

```
	for(int i = 1; i <= n; ++i)
		update(root[i] = root[i - 1], 0, M, a[i]);
}
```

```
//Segment Tree on Euler Path

#include <bits/stdc++.h>
#define mx 30010
using namespace std;

typedef long long int ll;

ll genieCnt[mx];
vector <ll> adjList[mx];
ll sparseTable[mx][16];
ll parent[mx];
ll disc[mx],finish[mx];
ll levelNode[mx];
pair <ll,ll> start_finish[mx];
ll arr[2*mx];
ll tree[4*2*mx];
ll n;
ll _time;

void dfs(ll src,ll _p,ll ht){
	disc[src] = ++_time;
	arr[_time] = genieCnt[src];
	levelNode[src] = ht;

	for(ll i = 0; i < adjList[src].size(); i++){
		ll v = adjList[src][i];
		if(v != _p){
			parent[v] = src;
			dfs(v,src,ht+1);
		}
	}

	finish[src] = ++_time;
	arr[_time] = -genieCnt[src];
	start_finish[src] = make_pair(disc[src],finish[src]);
}

void build(ll node, ll left, ll right){
	if(left == right){
		tree[node] = arr[left];
		return;
	}

	ll mid = (left + right)/2;
	build(2*node,left,mid);
	build(2*node+1,mid+1,right);
```

```
	tree[node] = tree[2*node] + tree[2*node+1];
}

ll query(ll node, ll left, ll right, ll qleft, ll qright){
	if(qleft > right || qright < left)  return 0;

	if(left >= qleft && right <= qright)	return tree[node];

	ll mid = (left + right) / 2;

	ll a = query(2*node,left,mid,qleft,qright);
	ll b = query(2*node+1,mid+1,right,qleft,qright);

	return a + b;
}

ll printQuery(ll nodeA, ll nodeB){
	ll lca;
	ll node1 = nodeA,node2 = nodeB;

	if(levelNode[nodeB] < levelNode[nodeA])
		swap(nodeA,nodeB);


	for(ll i = 15; i >= 0; i--){
		if(levelNode[nodeB] - (1 << i) >= levelNode[nodeA]){
			nodeB = sparseTable[nodeB][i];
		}
	}

	if(nodeA == nodeB)	lca = nodeA;
	else{
	for(ll i = 15; i >=0; i--){
		if(sparseTable[nodeA][i] != sparseTable[nodeB][i]){
			nodeA = sparseTable[nodeA][i];
			nodeB = sparseTable[nodeB][i];
		}
	}

	lca = sparseTable[nodeA][0];
	}
	//cout << "lca: " << lca << endl;
	ll ans =
query(1,1,2*n,start_finish[lca].first,start_finish[node1].first);
	ll ans2 =
query(1,1,2*n,start_finish[lca].first,start_finish[node2].first);

	return ans + ans2 -
query(1,1,2*n,start_finish[lca].first,start_finish[lca].first);
}

void update(ll node, ll left, ll right, ll idx, ll val){
	if(left == right){
		tree[node] = val;
		return;
	}
```

```
	ll mid = (left + right)/2;

	if(idx >= left && idx <= mid)
		update(2*node,left,mid,idx,val);
	else
		update(2*node+1,mid+1,right,idx,val);

	tree[node] = tree[2*node] + tree[2*node+1];
}


void reset(){
	for(ll i = 0; i < mx; i++){
		parent[i] = 0;
		genieCnt[i] = 0;
		adjList[i].clear();

		disc[i] = 0;
		finish[i] = 0;
		levelNode[i] = 0;
		start_finish[i] = make_pair(0,0);
		arr[i] = 0;

		for(ll j = 0; j < 16; j++)
			sparseTable[i][j] = 0;
	}
	n = 0;
	_time = 0;
	memset(tree,0,sizeof tree);
}

int main(){
	//freopen("in.txt","r",stdin);
	//freopen("out.txt","w",stdout);
	ll test;

	scanf("%lld",&test);

	for(ll i_t = 0; i_t < test; i_t++){
		reset();

		scanf("%lld",&n);

		for(ll i = 0; i < n; i++){
			scanf("%lld",genieCnt+i);
		}

		for(ll i = 0; i < n-1; i++){
			ll a,b;
			scanf("%lld %lld",&a,&b);
			adjList[a].push_back(b);
			adjList[b].push_back(a);
		}

		parent[0] = -1;
		dfs(0,-1,1);
```

```
            for(ll i = 0; i < n; i++)
                  sparseTable[i][0] = parent[i];


            for(ll i = 1; i < 16; i++){
                  for(ll j = 0; j < n; j++){
                        ll par = sparseTable[j][i-1];
                        if(par == -1){
                              sparseTable[j][i] = -1;
                        }
                        else{
                              sparseTable[j][i] = sparseTable[par][i-1];
                        }
                  }
            }

            build(1,1,2*n);

            ll q;

            scanf("%lld",&q);
            printf("Case %lld:\n",i_t+1);
            for(ll i = 0; i < q; i++){
                  ll cmd,l,r;

                  scanf("%lld %lld %lld",&cmd,&l,&r);
                  if(cmd == 0){
                        ll val = printQuery(l,r);
                        printf("%lld\n",val);
                  }
                  else{

      //printf("(%lld,%lld)\n",start_finish[l].first,start_finish[l].second);
                        update(1,1,2*n,start_finish[l].first,r);
                        update(1,1,2*n,start_finish[l].second,-r);
                  }
            }
      }
}
```

String

```cpp
//KMP

#include <bits/stdc++.h>
using namespace std;

const int maxn = 1000006;
char P[maxn], T[maxn];
int b[maxn], n, m;

void kmpPreprocess (){
    int i = 0, j = -1;
    b[0] = -1;
    while (i < m){
        while (j >= 0 and P[i] != P[j])
            j = b[j];
        i++; j++;
        b[i] = j;
    }
}

void kmpSearch (){
    int i = 0, j = 0;
    while (i < n){
        while (j >= 0 and T[i] != P[j])
            j = b[j];
        i++; j++;
        if (j == m){
            // pattern found at index i - j
        }
    }
}
```

```
//Suffix Array

#include <bits/stdc++.h>
using namespace std;

const int maxn = 100005;
char T[maxn];
int n;
int ra[maxn], tra[maxn];
int sa[maxn], tsa[maxn];
int c[maxn], phi[maxn];
int lcp[maxn], plcp[maxn];

void reset (){
    for (int i = 0; i < maxn; i++){
        ra[i] = tra[i] = sa[i] = tsa[i] = c[i] = phi[i] = lcp[i] = plcp[i] =
0;
    }
}

void countingSort (int k){
    int i, sum, maxi = max (300, n);
    memset (c, 0, sizeof c);
    for (i = 0; i < n; i++)
        c[i + k < n ? ra[i + k] : 0]++;
    for (i = sum = 0; i < maxi; i++){
        int t = c[i]; c[i] = sum; sum += t;
    }
    for (i = 0; i < n; i++)
        tsa[c[sa[i] + k < n ? ra[sa[i] + k] : 0]++] = sa[i];
    for (i = 0; i < n; i++)
        sa[i] = tsa[i];
}

void buildSA (){
    int i, k, r;
    for (i = 0; i < n; i++) ra[i] = T[i];
    for (i = 0; i < n; i++) sa[i] = i;
    for (k = 1; k < n; k <<= 1){
        countingSort(k);
        countingSort(0);
        tra[sa[0]] = r = 0;
        for (i = 1; i < n; i++){
            tra[sa[i]] = (ra[sa[i]] == ra[sa[i - 1]] and ra[sa[i] + k] ==
ra[sa[i - 1] + k]) ? r : ++r;
```

```
        }
        for (i = 0; i < n; i++)
            ra[i] = tra[i];
        if (ra[sa[n - 1]] == n - 1) break;
    }
}

void buildLCP (){
    int i, L;
    phi[sa[0]] = -1;
    for (i = 1; i < n; i++)
        phi[sa[i]] = sa[i - 1];
    for (i = L = 0; i < n; i++){
        if (phi[i] == -1){
            plcp[i] = 0;
            continue;
        }
        while (T[i + L] == T[phi[i] + L])
            L++;
        plcp[i] = L;
        L = max (L - 1, 0);
    }
    for (i = 0; i < n; i++)
        lcp[i] = plcp[sa[i]];
}
```

```
//Trie

#include <bits/stdc++.h>
using namespace std;

const int maxn = 100005;

struct Trie{
    int next[27][mx];
    int endmark[mx];
    bool created[mx];
    int sz;

    void insertTrie (string& s){
        int v = 0;
        for (int i = 0; i < (int)s.size(); i++){
            int c = s[i] - 'a';
            if (!created[next[c][v]]){
                next[c][v] = ++sz;
                created[sz] = true;
            }
            v = next[c][v];
        }
        endmark[v]++;
    }

    bool searchTrie (string& s){
        int v = 0;
        for (int i = 0; i < (int)s.size(); i++){
            int c = s[i] - 'a';
            if (!created[next[c][v]])
                return false;
            v = next[c][v];
        }
        return (endmark[v] > 0);
    }
};
```

```
//Z-Algo

void compute_z_function(const char *S, int N) {
	int L = 0, R = 0;
	for (int i = 1; i < N; ++i) {
		if (i > R) {
			L = R = i;
			while (R < N && S[R - L] == S[R]) ++R;
			Z[i] = R - L; --R;
		} else {
			int k = i - L;
			if (Z[k] < R - i + 1) Z[i] = Z[k];
			else {
				L = i;
				while (R < N && S[R - k] == S[R]) ++R;
				Z[i] = R - L; --R;
			}
		}
	}
}
```

```cpp
//Aho Corasick

#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mx=100005;

//const int Node=250005;
const int Node=5000;

vector<ll> levelnd;

struct Trie
{
    int root,curNodeId;
    struct node
    {
        int val;
        int endcnt;
        int child[27];
        bool endmark;
        vector<int> endlist;
        void clear(){
            memset(child,0,sizeof child);
            val=endcnt=0;
            endlist.clear();
            endmark=false;
        }
    }trie[Node];

    void clear(){
        trie[root].clear();
        root=curNodeId=0;
    }

    void addTrie(string str,int id){
        int len=str.length();
        int cur=root;
        for(ll i=0;i<len;i++){
            int c = str[i]-'a';
            if(trie[cur].child[c]==0){
                curNodeId++;
                trie[curNodeId].clear();
                trie[curNodeId].val=c;
                trie[cur].child[c]=curNodeId;
            }
```

```
            cur=trie[cur].child[c];
        }
        trie[cur].endlist.push_back(id);
        trie[cur].endmark=true;
    }
    bool searchTrie (string str){
        int len=str.length();
        int cur=root;
        for (int i = 0; i < len; i++){
            int c = str[i] - 'a';
            if(trie[cur].child[c] == 0)return false;
            cur=trie[cur].child[c];
        }
        return trie[cur].endmark;
    }

};

struct  AhoCorasick
{
    Trie tr;
    int fail[Node];
    int par[Node];

    void clear(){
        tr.clear();
        memset(fail,0,sizeof fail);
        memset(par,0,sizeof par);
    }

    void Calculate_failure(){
        queue<int> q;
        q.push(tr.root);
        while(!q.empty()){
            ll u=q.front();q.pop();
            levelnd.push_back(u);
            for(ll i=0;i<26;i++){
                ll v=tr.trie[u].child[i];
                if(v!=0){
                    q.push(v);
                    par[v]=u;
                }
            }

            if(u==tr.root){
                fail[u]=0;
                par[u]=0;
                continue;
            }

            int p=par[u];
            int val=tr.trie[u].val;
            int f=fail[p];
            while(f!=0 and tr.trie[f].child[val]==0){
                f=fail[f];
            }
            fail[u]=tr.trie[f].child[val];
```

```
            if(u==fail[u])fail[u]=0;

        }
    }

    int GoTo(int nd,int c){
        if(tr.trie[nd].child[c]!=0){
            return tr.trie[nd].child[c];
        }

        int f=fail[nd];
        while(f!=0 and tr.trie[f].child[c]==0) f=fail[f];
        return tr.trie[f].child[c];
    }

    void findMatching(string str){
        int cur=tr.root;
        int len=str.length();
        for(ll i=0;i<len;i++){
            int c=str[i]-'a';
            cur=GoTo(cur,c);
            tr.trie[cur].endcnt++;
        }
    }


};
int n;
string T;
int ans[Node];

int main(){
    //freopen("input.txt","r",stdin);
    int t;
    scanf("%d",&t);
    AhoCorasick ahocorasick;
    for(int ts=1;ts<=t;ts++){
        printf("Case %d:\n",ts);
        scanf("%d",&n);
        ahocorasick.clear();
        levelnd.clear();
        memset(ans,0,sizeof ans);
        cin>>T;
        string temp;
        for(int i=1;i<=n;i++){
            cin>>temp;
            ahocorasick.tr.addTrie(temp,i);
        }
        ahocorasick.Calculate_failure();
        ahocorasick.findMatching(T);
        for(int i=(int)levelnd.size()-1;i>=0;i--){
            int u=levelnd[i];

            int f=ahocorasick.fail[u];
            ahocorasick.tr.trie[f].endcnt+=ahocorasick.tr.trie[u].endcnt;
            for(int j=0;j<(int)ahocorasick.tr.trie[u].endlist.size();j++){
                int qid=ahocorasick.tr.trie[u].endlist[j];
```

```
                ans[qid]+=ahocorasick.tr.trie[u].endcnt;
            }
        }
        for(int i=1;i<=n;i++){
            printf("%d\n",ans[i]);
        }
    }
    return 0;
}
```

Miscellaneous

```
//2D Lis O(nlogn)

typedef pair<int,int> pii;
pii p[100005];
set<pii> s[100005];
set<pii>::iterator it,it1;
int main(){
	int n,i,lo,hi,mid,lb,k,t,cs = 1;
	scanf("%d",&n);
	for(i = 0;i<n;i++) scanf("%d %d",&p[i].first,&p[i].second);
	s[0].insert(p[0]);
	k = 0;
	for(i = 1;i<n;i++){
		lo = 0; hi = k,lb = -1;
		while(lo<=hi){
			mid = (lo + hi) / 2;
			it = s[mid].lower_bound(p[i]);
			if(it!=s[mid].begin() ){
				it1 = it,it1--;
				if((*it1).first==p[i].first) it --;
			}
			if(it!=s[mid].begin() && (*(--it)).second<p[i].second)
lo = mid + 1,lb = max(lb,mid);
			else hi = mid - 1;
		}
		lb++;
		k = max(k,lb);
		it = s[lb].lower_bound(pii(p[i].first,-inf));
		if(it==s[lb].end() || ((*it).first>p[i].first ||
(*it).second>p[i].second))
			s[lb].insert(p[i]);
		it = s[lb].upper_bound(p[i]);
		while(it!=s[lb].end()){
			if((*it).first>=p[i].first && (*it).second >= p[i].second){
				it1 = it, it1++;
				s[lb].erase(it);
				it = it1;
			}
			else break;
		}
	}
	printf("%d\n",k+1);
	return 0;
```

```
}
```

```
//Big Integer
#include <bits/stdc++.h>

#define MAXL 1010
#define MAXS 1000010
#define MAXP 90000
#define INFIN 1001001001

#define pq priority_queue
#define btc(x) __builtin_popcount(x)
#define mp(x, y) make_pair(x, y)
#define paii pair< int, int >
#define pasi pair< string, int >
#define pais pair< int, string >
#define mem(a,b) memset(a, b, sizeof(a))
#define pb(a) push_back(a)
#define pi (2*acos(0))
#define oo 1<<20
#define dd double
#define ll long long int
#define llu long long unsigned
#define ERR 1e-5
#define fst first
#define sec second
#define SZ(a) (int)a.size()
#define All(a) a.begin(),a.end()

#define SIZE 1000
#define REP(i,n) for(i=0; i<n; i++)
#define REV(i,n) for(i=n; i>=0; i--)
#define FOR(i,p,k) for(i=p; i<k; i++)
#define Sort(x) sort(x.begin() , x.end())
#define Reverse(x) reverse(x.begin() , x.end())


using namespace std;

string Addition(string a,string b);
string Multiplication(string a,string b);
string Multiplication(string a,int k);
string Subtraction(string a,string b);
string Division(string a,string b);
string Division(string a,int k);
string Div_mod(string a,string b);
int Div_mod(string a,int k);
string cut_leading_zero(string a);
```

```
int compare(string a,string b);

int main()
{
    string a, b, c;
    cin >> a >> b;
    c = Addition( a , b );
    cout<<"Addition : "<<c<<endl;
    c = Subtraction( a , b );
    cout<<"Subtract : "<<c<<endl;
    c = Multiplication( a , b );
    cout<<"Multipli : "<<c<<endl;
    c = Division( a , b );
    cout<<"Division : "<<c<<endl;
    c = Div_mod( a , b );
    cout<<"Div_modd : "<<c<<endl;
    return 0;
}

string Multiplication(string a,string b)
{
    int i, j, multi, carry;
    string ans, temp;

    ans = "0";
    REV(j,SZ(b)-1)
    {
        temp = "";
        carry = 0;
        REV(i,SZ(a)-1)
        {
            multi = (a[i]-'0')*(b[j]-'0')+carry;
            temp += (multi%10+'0');
            carry = multi/10;
        }
        if( carry ) temp += (carry+'0');
        Reverse(temp);
        temp += string(SZ(b)-j-1,'0');
        ans = Addition(ans,temp);
    }
    ans = cut_leading_zero(ans);
    return ans;
}

string Multiplication(string a,int k)
{
    string ans;
    int i, sum, carry=0;

    REV(i,SZ(a)-1)
    {
        sum = (a[i]-'0')*k+carry;
        carry = sum/10;
        ans+=(sum%10)+'0';
    }
    while(carry)
    {
```

```
        ans += (carry%10)+'0';
        carry/=10;
    }
    Reverse(ans);
    ans = cut_leading_zero(ans);
    return ans;
}

string Addition(string a,string b)
{
    int carry=0, i;
    string ans;

    if( SZ(a)>SZ(b) ) b = string(SZ(a)-SZ(b),'0')+b;
    if( SZ(b)>SZ(a) ) a = string(SZ(b)-SZ(a),'0')+a;

    ans.resize(SZ(a));

    REV(i,SZ(a)-1)
    {
        int sum = carry+a[i]+b[i]-96;
        ans[i] = (char)(sum%10+'0');
        carry = sum/10;
    }
    if( carry ) ans.insert(0,string(1,carry+'0'));
    ans = cut_leading_zero(ans);
    return ans;
}

string Subtraction(string a,string b)
{
    int borrow = 0, i, sub;
    string ans;
    if( SZ(b)<SZ(a) ) b = string(SZ(a)-SZ(b),'0')+b;
    REV(i,SZ(a)-1)
    {
        sub = a[i]-b[i]-borrow;
        if( sub<0 )
        {
            sub += 10;
            borrow = 1;
        }
        else borrow = 0;
        ans += sub+'0';
    }
    Reverse( ans );
    ans = cut_leading_zero(ans);
    return ans;
}

string Division(string a,string b)
{
    string mod, temp, ans="0";
    int i, j;

    REP(i,SZ(a))
    {
```

```
        mod += a[i];
        mod = cut_leading_zero(mod);
        FOR(j,0,10)
        {
            temp = Multiplication(b,j);
            if( compare(temp,mod)==1 )
                break;
        }
        temp = Multiplication(b,j-1);
        mod = Subtraction(mod,temp);
        ans += (j-1)+'0';
    }
    mod = cut_leading_zero(mod);
    ans = cut_leading_zero(ans);
    return ans;
}

string Division(string a,int k)
{
    int i, sum=0;
    string ans = "0";

    REP(i,SZ(a))
    {
        sum = (sum*10+(a[i]-'0'));
        ans += (sum/k)+'0';
        sum = sum%k;
    }
    ans = cut_leading_zero(ans);
    return ans;
}

string Div_mod(string a,string b)
{
    string mod, temp, ans="0";
    int i, j;

    REP(i,SZ(a))
    {
        mod += a[i];
        mod = cut_leading_zero(mod);

        FOR(j,1,10)
        {
            temp = Multiplication(b,j);
            if( compare(temp,mod)>0 )
                break;
        }
        temp = Multiplication(b,j-1);
        mod = Subtraction(mod,temp);
        ans += (j-1)+'0';
    }
    mod = cut_leading_zero(mod);
    ans = cut_leading_zero(ans);
    return mod;
}
```

```
int Div_mod(string a,int k)
{
    int i, sum=0;
    REP(i,SZ(a))
        sum = (sum*10+(a[i]-'0'))%k;
    return sum;
}

int compare(string a,string b)
{
    int i;
    a = cut_leading_zero(a);
    b = cut_leading_zero(b);

    if( SZ(a)>SZ(b) ) return 1;
    if( SZ(a)<SZ(b) ) return -1;
    REP(i,SZ(a))
    {
        if( a[i]>b[i] ) return 1;
        else if( a[i]<b[i] ) return -1;
    }
    return 0;
}

string cut_leading_zero(string a)
{
    string s;
    int i;
    if( a[0]!='0' ) return a;
    REP(i,SZ(a)-1) if( a[i]!='0' ) break;
    FOR(i,i,SZ(a)) s+=a[i];
    return s;
}
```

```
//Bit Hacks

#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

unsigned int reverse_bits(unsigned int v){
    v = ((v >> 1) & 0x55555555) | ((v & 0x55555555) << 1);
    v = ((v >> 2) & 0x33333333) | ((v & 0x33333333) << 2);
    v = ((v >> 4) & 0x0F0F0F0F) | ((v & 0x0F0F0F0F) << 4);
    v = ((v >> 8) & 0x00FF00FF) | ((v & 0x00FF00FF) << 8);
    return ((v >> 16) | (v << 16));
}

/// Returns i if x = 2^i and 0 otherwise
int bitscan(unsigned int x){
    __asm__ volatile("bsf %0, %0" : "=r" (x) : "0" (x));
    return x;
}

/// Returns next number with same number of 1 bits
unsigned int next_combination(unsigned int x){
    unsigned int y = x & -x;
    x += y;
    unsigned int z = x & -x;
    z -= y;
    z = z >> bitscan(z & -z);
    return x | (z >> 1);
}

int main(){
}
```

```
//Double Hashing

#include <bits/stdtr1c++.h>
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
using namespace std;
struct simplehash{
    int len;
    long long base, mod;
    vector <int> P, H, R;
    simplehash(){}
    simplehash(const char* str, long long b, long long m){
        base = b, mod = m, len = strlen(str);
        P.resize(len + 3, 1), H.resize(len + 3, 0), R.resize(len + 3, 0);

        for (int i = 1; i <= len; i++) P[i] = (P[i - 1] * base) % mod;
        for (int i = 1; i <= len; i++) H[i] = (H[i - 1] * base + str[i - 1] +
1007) % mod;
        for (int i = len; i >= 1; i--) R[i] = (R[i + 1] * base + str[i - 1] +
1007) % mod;
    }
    inline int range_hash(int l, int r){
        int hashval = H[r + 1] - ((long long)P[r - l + 1] * H[l] % mod);
        return (hashval < 0 ? hashval + mod : hashval);
    }

    inline int reverse_hash(int l, int r){;
        int hashval = R[l + 1] - ((long long)P[r - l + 1] * R[r + 2] % mod);
        return (hashval < 0 ? hashval + mod : hashval);
    }
};
struct stringhash{
    simplehash sh1, sh2;
    stringhash(){}
    stringhash(const char* str){
        sh1 = simplehash(str, 1949313259, 2091573227);
        sh2 = simplehash(str, 1997293877, 2117566807);
    }

    inline long long range_hash(int l, int r){
        return ((long long)sh1.range_hash(l, r) << 32) ^ sh2.range_hash(l,
r);
    }

    inline long long reverse_hash(int l, int r){
```

```
        return ((long long)sh1.reverse_hash(l, r) << 32) ^
sh2.reverse_hash(l, r);
    }
};

int main(){

}
```

//Hashing

```
struct Hash {
      struct base {
            string s; int b, mod;
            vector<int> hash, p;
            void init(string &_s, int _b, int _mod) { // b > 26, prime.
                  s = _s; b = _b, mod = _mod;
                  hash.resize(s.size());
                  p.resize(s.size());
                  hash[0] = s[0] - 'A' + 1; p[0] = 1;
                  for(int i = 1; i < s.size(); ++i) {
                        hash[i] = (ll) hash[i - 1] * b % mod;
                        hash[i] += s[i] - 'A' + 1;
                        if(hash[i] >= mod) hash[i] -= mod;
                        p[i] = (ll) p[i - 1] * b % mod;
                  } cout << endl;
            }
            int get(int l, int r) {
                  int ret = hash[r];
                  if(l) ret -= (ll) hash[l - 1] * p[r - l + 1] % mod;
                  if(ret < 0) ret += mod;
                  return ret;
            }
      } h[2];
      void init(string &s) {
            h[0].init(s, 29, 1e9+7);
            h[1].init(s, 31, 1e9+9);
      }
      pair<int, int> get(int l, int r) {
            return { h[0].get(l, r), h[1].get(l, r) };
      }
} H;
```

```
//Notes for hashing


/// Important Notes for Hashing

/**
    1. Single Hashing:
    Mod = large prime number(Ex: 1e9+7)
    B = base (the smallest prime greater than total distinct characters)(Ex:
29)
    For string "abcba"
    Hash = (a*(B^0) + b*(B^1) + c*(B^2) + b*(B^3) + a*(B^4))%Mod

    2. Double Hashing:
    Maintain two hash value for each string using above approach.
    For the second hash, use different Mod and B.
    2nd B should be twin prime with first B(Ex: 31)
    2nd Mod should be twin prime with first Mod(Ex: 1e9+9)

    3. Hashing If Position Doesn't Matter (Only the number of occurrence
matters):
    Mod and B same as above
    For string "abcba" (Which is same as "aabbc")
    Hash = ((B^a) + (B^b) + (B^c) + (B^b) + (B^a))%Mod


    4. If string s is same as it's rotation (Circular)
    Hash = Sum over the hash of all the adjacent pair of characters
**/

/// Barnestaine String Hashing
/// Another useful technique to get hash of a string

LLU barnestaine(const char* s) {
    LLU hash = 0, c;
    while((c = *s++)){
        hash = ((hash << 5) + hash) ^ c;
    }
    return hash;
}
```

```
//LIS O(nlogn) with full path

int num[MX], mem[MX], prev[MX], array[MX], res[MX],maxlen;
void LIS(int SZ,int num[]){
	CLR(mem), CLR(prev), CLR(array), CLR(res) ;
	int i , k;
	maxlen = 1;
	array[0] = -inf;
	RFOR(i,1,SZ+1)	array[i] = inf;
	prev[0] = -1, mem[0] = num[0];
	FOR(i,SZ){
		k = lower_bound( array , array + maxlen + 1, num[i] ) - array;
		if( k == 1) array[k] = num[i], mem[k] = i, prev[i] = -1;
		else  array[k] = num[i], mem[k] = i,prev[i] = mem[k -1];
		if(k > maxlen)	maxlen = k;
	}
	k = 0;
	for(i = mem[maxlen];i != -1;i = prev[i])  res[k++] = num[i];
}
```

```
//LIS


#include <bits/stdc++.h>
using namespace std;

#define Size 100005
#define Mod 1000000007LL

int N;
int A[100005];
int LIS[100005];
int LDS[100005];
vector<int> List;

void calculate_LIS(){
    List.clear();
    for(int i = 0;i<N;i++){
        vector<int>::iterator low =
lower_bound(List.begin(),List.end(),A[i]);
        LIS[i] = (low - List.begin());
        if(low == List.end()) List.pb(A[i]);
        else *low = A[i];
    }
}

void calculate_LDS(){
    List.clear();
    reverse(A,A+N);
    for(int i = 0;i<N;i++){
        vector<int>::iterator low =
lower_bound(List.begin(),List.end(),A[i]);
        LDS[i] = (low - List.begin());
        if(low == List.end()) List.pb(A[i]);
        else *low = A[i];
    }
    reverse(A,A+N);
    reverse(LDS,LDS+N);
}

void solve(){
    calculate_LIS();
    calculate_LDS();
    /// LIS[i] = length of LIS found till position i;
    /// LDS[i] = length of LDS found till position i;
```

```
    int resLIS = 0,resLDS = 0;
    for(int i = 0;i<N;i++){
        resLIS = max(resLIS,LIS[i]);
        resLDS = max(resLDS,LDS[i]);
    }
    pf("LIS: %d , LDS: %d\n",resLIS,resLDS);
}

int main(){
    int nCase;
    sf("%d",&nCase);
    for(int cs = 1;cs<=nCase;cs++){
        sf("%d",&N);
        for(int i = 0;i<N;i++){
            sf("%d",&A[i]);
        }
        solve();
    }
}
```

```
//Matrix Exponentiation

#define MAX_N 2
struct Matrix
{
    int mat[MAX_N][MAX_N];
};
Matrix matMul(Matrix a, Matrix b)
{
    Matrix ans;
    int i, j, k;
    for (i = 0; i < MAX_N; i++)
        for (j = 0; j < MAX_N; j++)
            for (ans.mat[i][j] = k = 0; k < MAX_N; k++)
                ans.mat[i][j] += a.mat[i][k] * b.mat[k][j];
    return ans;
}
Matrix matPow(Matrix base, int p)
{
    Matrix ans;
    int i, j;
    for (i = 0; i < MAX_N; i++)
        for (j = 0; j < MAX_N; j++)
            ans.mat[i][j] = (i == j);
    while (p)
    {
        if (p & 1)
            ans = matMul(ans, base);
        base = matMul(base, base);
        p >>= 1;
    }
    return ans;
}
```

```cpp
//Maximum XOR Subset

#include <bits/stdtr1c++.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define bitlen(x) ((x) == 0 ? (0) : (64 - __builtin_clzll(x)))
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

long long ar[MAX];

long long solve(int n, long long* ar){ /// hash = 220650
    vector <long long> v[64];
    for (int i = 0; i < n; i++) v[bitlen(ar[i])].push_back(ar[i]);

    long long m, x, res = 0;
    for (int i = 63; i > 0; i--){
        int l = v[i].size();
        if (l){
            m = v[i][0];
            res = max(res, res ^ m);

            for (int j = 1; j < l; j++){
                x = m ^ v[i][j];
                if (x) v[bitlen(x)].push_back(x);
            }
            v[i].clear();
        }
    }
    return res;
}

int main(){
    return 0;
}
```

```
//Overflow

#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MOD 1007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

long long mul(long long a, long long b){
    long long res = 0;

    while (b){
        if (b & 1LL) res = (res + a) % MOD;
        a = (a << 1LL) % MOD;
        b >>= 1LL;
    }

    return res;
}

long long mul(long long a, long long b){
   if ((MOD < 3037000500LL)) return ((a * b) % MOD);
   long double res = a;
   res *= b;
   long long c = (long long)(res / MOD);
   a *= b;
   a -= c * MOD;
   if (a >= MOD) a -= MOD;
   if (a < 0) a += MOD;
   return a;
}

long long binary_div(long long a, long long b){ /* (a + b) / 2 without
overflow */
    long long x = (a >> 1LL), y = (b >> 1LL);
    long long res = x + y;
    if ((a & 1) && (b & 1)) res++;
    return res;
}



const long long LIM = LLONG_MAX;
```

```cpp
uint64_t mul(uint64_t a, uint64_t b){
    a %= MOD, b %= MOD;
    if (a > b) swap(a, b);
    if (!a) return 0;
    if (a < (LIM / b)) return ((a * b) % MOD);

    uint64_t res = 0;
    int x, k = min(30, __builtin_clzll(MOD) - 1);
    int bitmask = (1 << k) - 1;

    while (a > 0){
        x = a & bitmask;
        res = (res + (b * x)) % MOD;
        a >>= k;
        b = (b << k) % MOD;
    }
    return res;
}

/// Not sure, morris vesion
long long mul(long long a, long long b, long long MOD) {
    long long x = (long long)((double)a * b / MOD + 0.5);
    long long res = ((a * b) - (x * MOD)) % MOD;
    if (res < 0) res += MOD;
    return res;
}

int main(){
}
```

```c
//Rolling Hash

#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

const unsigned long long base = 1968647011ULL;

int n, ar[MAX];
unsigned long long P[MAX];

void RollingHash(int len, bool gen){
    int i, j;
    unsigned long long h = 0, x;

    if (gen){
        P[0] = 1ULL;
        for (i = 1; i < MAX; i++) P[i] = (P[i - 1] * base);
    }

    for (i = 0; i < len; i++) h = (h * base) + ar[i];
    for (i = 0; (i + len) <= n; i++){
        /* h contains required hash value now */

        x = (h - (P[len - 1] * ar[i]));
        h = (x * base) + ar[i + len];
    }
    return 0;
}

int main(){
}
```

```
//Ternary Search

/// Include My Code Template
#include <bits/stdc++.h>
using namespace std;

LL calcCost(LL mid) {
	/// Calculate cost for mid and return:
}

LL ternarySearch() {
	LL L = 0, R = Max, cnt = 70;
	LL res = Max;

	while (cnt--) {
		/// Both of this approach works.

		//double mid_L = (L + (R - L) / 3);
		//double mid_R = (R - (R - L) / 3);

		double mid_L = (2*L + R) / 3;
		double mid_R = (L + 2*R) / 3;

		if (calcCost(mid_R) < calcCost(mid_L)){
			L = mid_L;
			res = min(res,calcCost(mid_R));
		}else{
			R = mid_R;
		}
	}

	res = min(res,calcCost(L));
	res = min(res,calcCost(R));
	return res;
}
```

```
//To Roman

string toRoman( int n ) {
    if( n < 4 ) return fill( 'i', n );
    if( n < 6 ) return fill( 'i', 5 - n ) + "v";
    if( n < 9 ) return string( "v" ) + fill( 'i', n - 5 );
    if( n < 11 ) return fill( 'i', 10 - n ) + "x";
    if( n < 40 ) return fill( 'x', n / 10 ) + toRoman( n % 10 );
    if( n < 60 ) return fill( 'x', 5 - n / 10 ) + 'l' + toRoman( n % 10 );
    if( n < 90 ) return string( "l" )+fill('x',n/10-5) + toRoman( n % 10 );
    if( n < 110 ) return fill( 'x', 10 - n / 10 ) + "c" + toRoman( n % 10 );
    if( n < 400 ) return fill( 'c', n / 100 ) + toRoman( n % 100 );
    if( n < 600 ) return fill( 'c', 5 - n / 100 ) + 'd' + toRoman( n % 100 );
    if( n < 900 ) return string("d")+fill('c',n/100- 5 ) + toRoman( n % 100
);
    if( n < 1100 ) return fill( 'c', 10 - n / 100 ) + "m" + toRoman( n % 100
);
    if( n < 4000 ) return fill( 'm', n / 1000 ) + toRoman( n % 1000 );
    return "?";
}
```

Dynamic Programming

```cpp
//Open Close Interval DP


#include<bits/stdc++.h>

using namespace std;

typedef long long ll;
const int mx=100005;
const int mod=1000000007;

int n,k;

int ara[205];
int dp[205][205][1005];

int func(int idx,int g,int tk){
    if(g<0) return 0;
    if(tk<0)return 0;
    if(idx==n+1){
        if(tk>=0 and g==0)return 1;
        else return 0;
    }
    if(dp[idx][g][tk]!=-1)return dp[idx][g][tk];
    ll ret=0;
    ///open a group
    ret+=func(idx+1,g+1,tk-g*(ara[idx]-ara[idx-1]));
    ret%=mod;
    ///close a group
    ret+=(((ll)g*(ll)func(idx+1,g-1,tk-g*(ara[idx]-ara[idx-1])))%mod);
    ret%=mod;
    ///enter a current group
    ret+=(((ll)g*(ll)func(idx+1,g,tk-g*(ara[idx]-ara[idx-1])))%mod);
    ret%=mod;
    ///open a group and close
    ret+=func(idx+1,g,tk-g*(ara[idx]-ara[idx-1]));
    ret%=mod;

    return dp[idx][g][tk]=(int)ret;
}
```

```
int main(){
    //freopen("input.txt","r",stdin);
    memset(dp,-1,sizeof dp);
    scanf("%d %d",&n,&k);
    for(int i=1;i<=n;i++){
        scanf("%d",&ara[i]);
    }
    sort(ara+1,ara+n+1);
    ara[0]=ara[1];
    int ans=func(1,0,k);
    printf("%d\n",ans);
    return 0;
}
```

//Sibling DP

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

using namespace std;

typedef long long ll;
const ll mx=105;
const ll inf=1000000000000000000;
const ll mod=10000000007;

ll n,k;
ll dp[mx][mx];
vector<pair<ll,ll> > adj2[mx];
vector<pair<ll,ll> > adj[mx];

ll dfs(ll par,ll idx,ll remk){
    if(remk<0)return inf;
    if(adj[par].size()<idx+1)return 0;
    ll u=adj[par][idx].first;
    if(dp[u][remk]!=-1)return dp[u][remk];

    ll ret=inf;
    ll under=0,sibling=0;

    ///creating new group
    if(par!=0){
        under=1+dfs(u,0,k);
        sibling=dfs(par,idx+1,remk);
        ret=min(ret,under+sibling);
    }


    ///divide the current group
    ll temp=remk-adj[par][idx].second;
    for(ll childk=temp;childk>=0;childk--){
        ll siblingk=temp-childk;
        under=0;
        sibling=0;
        under=dfs(u,0,childk);
```

```
        sibling=dfs(par,idx+1,siblingk);
        ret=min(ret,under+sibling);
    }

    return dp[u][remk] = ret;

}


void make(ll u,ll par){
    for(ll i=0;i<(ll)adj2[u].size();i++){
        ll v=adj2[u][i].first;
        if(v==par)continue;
        adj[u].push_back(make_pair(adj2[u][i].first,adj2[u][i].second));
        make(v,u);
    }
}

inline void clean(){
    for(ll i=0;i<mx;i++){
        adj[i].clear();
        adj2[i].clear();
    }
    memset(dp,-1,sizeof dp);
}

int main(){
    //freopen("input.txt","r",stdin);

    ll t;
    scanf("%lld",&t);
    for(ll ts=1;ts<=t;ts++){
        clean();
        scanf("%lld %lld",&n,&k);
        ll u,v,c;
        for(ll i=0;i<n-1;i++){
            scanf("%lld %lld %lld",&u,&v,&c);
            adj2[u].push_back(make_pair(v,c));
            adj2[v].push_back(make_pair(u,c));
        }
        adj[0].push_back(make_pair(1,0));
        make(1,0);
        ll ans=1+dfs(0,0,k);

        printf("Case %lld: %lld\n",ts,ans);
    }
    return 0;
}
```

```cpp
//Tree DP 766E

#include<bits/stdc++.h>

using namespace std;

#define fRead(x)        freopen(x,"r",stdin)
#define fWrite(x)       freopen (x,"w",stdout)

#define LL              long long
#define ULL             unsigned long long
#define ff              first
#define ss              second
#define pb              push_back
#define INF             5e13
#define PI              acos(-1.0)
#define mk              make_pair
#define pii             pair<int,int>
#define pll             pair<LL,LL>
#define all(a)          a.begin(),a.end()
#define Unique(a)       sort(all(a)),a.erase(unique(all(a)),a.end())

#define min3(a,b,c)     min(a,min(b,c))
#define max3(a,b,c)     max(a,max(b,c))
#define min4(a,b,c,d)   min(a,min(b,min(c,d)))
#define max4(a,b,c,d)   max(a,max(b,max(c,d)))
#define SQR(a)          ((a)*(a))
#define FOR(i,a,b)      for(int i=a;i<=b;i++)
#define ROF(i,a,b)      for(int i=a;i>=b;i--)
#define REP(i,b)        for(int i=0;i<b;i++)
#define MEM(a,x)        memset(a,x,sizeof(a))
#define ABS(x)          ((x)<0?-(x):(x))

#define SORT(v)         sort(v.begin(),v.end())
#define REV(v)          reverse(v.begin(),v.end())

#define FastRead        ios_base::sync_with_stdio(0);cin.tie(nullptr);
bool Check(int N,int pos) { return (bool)(N&(1<<pos)); }
```

```
int Set(int N,int pos) { return (N|(1<<pos)); }

vector<int>G[100005];
int dp[100005][24][2];
int n;
int ara[100005];
LL mul[24];
int nob = 23;
void dfs(int u,int pre)
{
   for(int v : G[u]){
      if(v==pre)continue;
      dfs(v,u);

      for(int i = 0;i < nob;i++){
         int bit = Check(ara[u],i);

         mul[i] += dp[u][i][0] * dp[v][i][1];
         mul[i] += dp[u][i][1] * dp[v][i][0];
//          cout << "from to " <<  u << " " << v << endl;
//          cout << dp[u][i][0] << " " << dp[v][i][1] << " ";
//          cout << dp[u][i][1] << " " << dp[v][i][0] << "\n";
         if(bit == 0){
            dp[u][i][0] += dp[v][i][0];
            dp[u][i][1] += dp[v][i][1];
         }else{
            dp[u][i][0] += dp[v][i][1];
            dp[u][i][1] += dp[v][i][0];
         }
      }
   }
   for(int i = 0;i < nob;i++){
         if(Check(ara[u],i))dp[u][i][1] += 1;
         else dp[u][i][0] += 1;
   }
   for(int i = 0;i < nob;i++){
      mul[i] += dp[u][i][1];
   }

}
int main(){
   scanf("%d",&n);
   for(int i = 1;i<=n;i++)scanf("%d",&ara[i]);
   FOR(i,1,n-1)
   {
      int a,b;
      scanf("%d %d",&a,&b);
      G[a].pb(b);
      G[b].pb(a);
   }
   dfs(1,0);
   LL ans = 0;
   for(int i = 0;i<nob;i++){
      ans = ans + (1LL << i) * mul[i];
   }
   printf("%lld\n",ans);
}
```

```
//Template

#include <bits/stdc++.h>

#define INF 1000000
#define MOD 1000000007
#define pause system("pause")
#define clock 1.0 * clock() / CLOCKS_PER_SEC
#define filein freopen ("in.txt", "r", stdin)
#define fileout freopen ("out.txt", "w", stdout)

void setBit(int& N, int p){N|=(1<<p);}
void resetBit(int& N, int p){N&=~(1<<p);}
bool checkBit(int& N, int p){return N&(1<<p);}

template<class T>
inline void input(T &x) {
    register char c = getchar();x = 0;
    int neg = 0;
    for(; ((c<48 || c>57) && c != '-'); c = getchar());
    if(c=='-'){neg = 1;c = getchar();}
    for(; c>47 && c<58 ; c = getchar()){x = (x<<1) + (x<<3) + c - 48;}
    if(neg) x = -x;
}

//more faster
template <class T>
inline bool input (T &ret){
    char c; int sgn;
    if (c = getchar(), c == EOF) return 0;
    while (c != '-' && (c < '0' || c > '9')) c = getchar();
    sgn = (c == '-') ? -1 : 1, ret = (c == '-') ? 0 : (c - '0');
    while (c = getchar(), c >= '0' && c <= '9') ret = ret * 10 + (c - '0');
    ret *= sgn;
    return 1;
}

const int dr[]={0,-1,0,1,-1,-1,1,1};
const int dc[]={1,0,-1,0,1,-1,-1,1};
```