CodeBook

MU_Resplendence

Md. Emrul Chowdhury
Majharul Islam Rafat
Amanur Rahman

October 2019
Metropolitan University, Sylhet

Table of Contents

Part 1 ⇛ Number Theory

1.1: BigMod

```
ll BigMod(ll a, ll b, ll m) {
    if(!b) {
        return 1 % m;
    }
    ll x = BigMod(a, b / 2, m);
    x = (x * x) % m;
    if(b & 1) {
        x = (x * a) % m;
    } return x;
}
```

1.2: Modular Inverse

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

}

ll modularInverse(ll a, ll n) {
    pair<ll, ll> ret = extendedEuclid(a, n);
    return ((ret.first % n) + n) % n;
}
```

1.3: Sieve

```
bool isPrime[1000101];

void sieve() {
    int MAX = 1000100, sq = sqrt(MAX);
    memset(isPrime, true, sizeof isPrime);
    for(int i = 4; i <= MAX; i += 2) {
        isPrime[i] = 0;
    }
    for(int i = 3; i <= sq; i += 2) {
        if(isPrime[i]) {
            for(int j = i * i; j <= MAX; j += i) {
                isPrime[j] = 0;
            }
        }
    }
    isPrime[1] = 0;
    isPrime[0] = 0;
}
```

1.4: nCr

```
ll f[Max], inv[Max];

ll BigMod(ll a, int e) {
    if(e == -1) {
        e = Mod - 2;
    }
    ll r = 1;
    while(e) {
        if(e & 1) {
            r *= a, r %= Mod;
        }
        a *= a, a %= Mod, e >>= 1;
    }
    return r;
}

ll nCr(int x, int y) {
    if(x < 0 or y < 0 or x < y) {
        return 0;
    }
    return f[x] * (inv[y] * inv[x - y] % Mod) % Mod;
}

void init() {
    f[0] = 1;
    for(int i = 1; i < Max; ++i) {
        f[i] = i * 1LL * f[i - 1] % Mod;
    }
    inv[Max - 1] = BigMod(f[Max - 1], -1);
    for(int i = Max - 1; i; --i) {
        inv[i - 1] = i * 1LL * inv[i] % Mod;
    }
}
```

1.5: nCr with DP

```
ll nCr[Max][Max];

ll rec(ll n, ll r) {
    if(n == r) {
        return 1;
    }
    if(r == 1) {
        return n;
    }
    ll &ret = nCr[n][r];
    if(~ret) {
        return ret;
    }
    ret = rec(n - 1, r) + rec(n - 1, r - 1);
    return ret;
}
```

1.6: Phi Function

```
ll phi[Max];

void phigen(int n) {
    for(int i = 1; i <= n; i++) {
        phi[i] = i;
    }
    for(int p = 2; p <= n; p++) {
        if(phi[p] == p) {
            phi[p] = p - 1;
            for(int i = 2 * p; i <= n; i += p) {
                phi[i] = (phi[i] / p) * (p - 1);
            }
        }
    }
}
```

1.7: Divisor Count

```
ll divcnt[Max];

void DivisorCount(ll n) {
    for(int i = 1; i <= n; i++) {
        for(int j = i; j <= n; j += i) {
            divcnt[j]++;
        }
    }
}
```

1.8: Sum of Divisors in a range

```
ll triangle(ll a, ll b) {
    return (a + b + 1) * (b - a) / 2 ;
}

ll divSum(ll a, ll b) { // Sum of divisors between a to b
    ll n = sqrt(b);
    ll sum = 0;

    for(ll i = 1; i <= n; i++) {
        sum += i * (b / i - a / i) + triangle(max(n, a / i), max(n, b / i));
    }
    return sum;
}

int main() {
    ll a, b;
    cin >> a >> b;
    ll ans = divSum(a - 1, b);
    cout << ans << endl;
    return 0;
}
```

1.9: Divisors of a Factorial

```
vector <ull> vec;

bool isPrime[1000101];

void sieve() {
    int MAX = 1000100, sq = sqrt(MAX);
    memset(isPrime, true, sizeof isPrime);
    for(int i = 4; i <= MAX; i += 2) {
        isPrime[i] = 0;
    }
    for(int i = 3; i <= sq; i += 2) {
        if(isPrime[i]) {
            for(int j = i * i; j <= MAX; j += i) {
                isPrime[j] = 0;
            }
        }
    }
    isPrime[1] = 0;
    isPrime[0] = 0;

    for(int p = 2; p < MAX; p++) {
        if(isPrime[p]) {
            vec.push_back(p);
        }
    }
}

ull factorialDivisors(ull n) {
    ull res = 1;

    for(ull x : vec) {
        ull p = x;

        ull exp = 0;
        while(p <= n) {
            exp = exp + (n / p);
            p = p * x;
        }
        res = res * (exp + 1);
    }

    return res;
}

int main() {
    sieve();
    cout << factorialDivisors(6) << endl;
    return 0;
}
```

1.10: Bitwise Sieve

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

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

```
void Set(int &n, int pos) {
    n = n | (1 << pos);
}

const int Max = 1e8 + 5;
int prime[(Max >> 5) + 2];
vector <int> primes;

void sieve() {
    int lim = sqrt(Max);
    for(int i = 3; i <= lim; i += 2) {
        if(!Check(prime[i >> 5], i & 31)) {
            for(int j = i * i; j <= Max; j += (i << 1)) {
                Set(prime[j >> 5], j & 31);
            }
        }
    }
    primes.push_back(2);
    for(int i = 3; i <= Max; i += 2) {
        if(!Check(prime[i >> 5], i & 31)) {
            primes.push_back(i);
        }
    }
}

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

1.11: Miller Rabin Primality Test

```
inline bool miller(ll p, int iter = 20) {
    if(p == 3 || p == 2 || p == 5) {
        return true;
    }
    if(p % 2 == 0) {
        return false;
    }
    if(p < 3) {
        return false;
    }
    mt19937_64 rnd(chrono::steady_clock::now().time_since_epoch().count());

    for(int i = 0; i < iter; i++) {
        ll a = (rnd()) % (p - 4) + 2;
        ll s = p - 1;
        while(s % 2 == 0) {
            s /= 2;
        }
        ll mod = bigmod(a, s, p);
        if(mod == 1 || mod == p - 1) {
            continue;
        }
        bool flag = 0;
        s *= 2;
        while(s != p - 1) {
            mod = mulmod(mod, mod, p);
            if(mod == p - 1) {
                flag = 1;
                break;
            }
            s *= 2;
        }
        if(flag == 0) {
            return 0;
        }
    }
    return 1;
}
```

Part 2 ⇛ Data Structure

2.1: Merge Sort Tree

```
vector <int> tree[4 * Max];
int ar[Max];
int inp[Max];

void build(int node, int l, int r) {
    if(l == r) {
        tree[node].push_back(ar[l]);
        return;
    }
    int mid = (l + r) >> 1;
    int lf = node * 2;
    int rt = node * 2 + 1;
    build(lf, l, mid);
    build(rt, mid + 1, r);
    merge(tree[lf].begin(), tree[lf].end(), tree[rt].begin(), tree[rt].end(),
back_inserter(tree[node]));
}

int query(int node, int l, int r, int L, int R, int val) {
    if(L > r || l > R) {
        return 0;
    }
    if(L <= l && r <= R) {
        return lower_bound(tree[node].begin(), tree[node].end(), val) -
tree[node].begin();
    }
    int mid = (l + r) >> 1;
    int lf = node * 2;
    int rt = node * 2 + 1;
    int u = query(lf, l, mid, L, R, val);
    int v = query(rt, mid + 1, r, L, R, val);
    return u + v;
}
```

2.2: MO’s Algorithm

```
const int Max = 1e5 + 10;

int BLOCK_SIZE, cnt;
int mp[Max];
int ar[Max];
int ans[Max];
```

```
struct Node {
    int l, r, id;

    Node() {};

    Node(int l_, int r_, int id_) {
        l = l_;
        r = r_;
        id = id_;
    }

    bool operator<(const Node &b) const {
        int block_a = l / BLOCK_SIZE;
        int block_b = b.l / BLOCK_SIZE;
        if(block_a != block_b) {
            return block_a < block_b;
        }
        return r < b.r;
    }
} qr[Max];

void add(int x) {
}

void del(int x) {
}

int main() {
    int n, m, l, r;

    scanf("%d %d", &n, &m);
    BLOCK_SIZE = sqrt(n);
    memset(mp, 0, sizeof mp);
    cnt = 0;

    for(int i = 1; i <= n; i++) {
        scanf("%d", &ar[i]);
    }

    for(int i = 1; i <= m; i++) {
        scanf("%d %d", &l, &r);
        qr[i] = Node(l, r, i);
    }

    sort(qr + 1, qr + m + 1);

    int L = 1, R = 0;
    for(int i = 1; i <= m; i++) {
        int l = qr[i].l;
        int r = qr[i].r;

        while(R < r) {
            R++;
            add(ar[R]);
        }
        while(R > r) {
            del(ar[R]);
            R--;
        }
        while(L < l) {
            del(ar[L]);
            L++;
        }
        while(L > l) {
            L--;
            add(ar[L]);
        }

        ans[qr[i].id] = cnt;
    }

    for(int i = 1; i <= m; i++) {
        printf("%lld\n", ans[i]);
    }
    return 0;
}
```

2.3: Segment Tree

```
const int Max = 1e5 + 10;

int ar[Max];
int seg[Max * 4];

void build(int node, int l, int r) {
    if(l == r) {
        seg[node] = ar[l];
        return;
    }
    int mid = (l + r) >> 1;
    int lf = node * 2;
    int rt = node * 2 + 1;
    build(lf, l, mid);
    build(rt, mid + 1, r);
    seg[node] = seg[lf] + seg[rt];
}

int update(int node, int l, int r, int idx, int val) {
    if(l == r) {
        seg[node] = val;
        return seg[node];
    }
    int mid = (l + r) >> 1, u = seg[node * 2], v = seg[node * 2 + 1];
    if(mid >= idx) {
        u = update(2 * node, l, mid, idx, val);
    }
    else {
        v = update(2 * node + 1, mid + 1, r, idx, val);
    }
    return seg[node] = u + v;
}

int query(int node, int l, int r, int L, int R) {
    if(R < l || L > r) {
        return 0;
    }
    if(L <= l && r <= R) {
        return seg[node];
    }
    int mid = (l + r) >> 1;
    int u = query(node * 2, l, mid, L, R);
    int v = query(node * 2 + 1, mid + 1, r, L, R);
    return u + v;
}
```

2.4: Sqrt Decomposition

```
const int Max = 1e5 + 10;

ll BLOCK_SIZE;
ll BLOCK[1001];
ll ar[100001];

ll getID(ll idx) {
    ll id = idx / BLOCK_SIZE;
    return id;
}

ll calc() {
    for(int i = 1; i < 1000; i++) {
        BLOCK[i] = 0;
    }
}

void del(int idx) {
    int id = getID(idx);
    BLOCK[id] -= ar[idx];
    ar[idx] = 0;
}

void add(int idx, ll val) {
    int id = getID(idx);
    BLOCK[id] += val;
}

ll query(int l, int r) {
    int lf = getID(l);
    int rt = getID(r);

    ll ret = 0;

    if(lf == rt) {
        for(int i = l; i <= r; i++) {
            ret += ar[i];
        }
        return ret;
    }

    for(int i = l; i < (lf + 1) * BLOCK_SIZE; i++) {
        ret += ar[i];
    }

    for(int i = lf + 1; i < rt; i++) {
        ret += BLOCK[i];
    }

    for(int i = rt * BLOCK_SIZE; i <= r; i++) {
        ret += ar[i];
    }
    return ret;
}

int main() {
    int t, n, m;

    scanf("%d", &t);
    int tc = 1;
    while(t--) {
        scanf("%d %d", &n, &m);
        BLOCK_SIZE = sqrt(n);
        calc();
        for(int i = 1; i <= n; i++) {
            scanf("%lld", &ar[i]);
            add(i, ar[i]);
        }
        printf("Case %d:\n", tc++);
        int type, l, r, val;
        while(m--) {
            scanf("%d", &type);
            if(type == 1) {
                scanf("%d", &l);
                l++;
                printf("%lld\n", ar[l]);
                del(l);
            }
            else if(type == 2) {
                scanf("%d %d", &l, &val);
                l++;
                add(l, val);
                ar[l] += val;
            }
            else {
                scanf("%d %d", &l, &r);
                l++;
                r++;
                printf("%lld\n", query(l, r));
            }
        }
    }
    return 0;
}
```

2.5: Disjoint Set

```
int par[Max];
int cnt[Max];
int rnk[Max];

void make_set() {
    for(int i = 0; i < n; i++) {
        par[i] = i;
        cnt[i] = 1;
        rnk[i] = 0;
    }
}

int find_rep(int x) {
    if(x != par[x]) {
        par[x] = find_rep(par[x]);
    }
    return par[x];
}

int union_(int u, int v) {
    if((u = find_rep(u)) != (v = find_rep(v))) {
        if(rnk[u] < rnk[v]) {
            cnt[v] += cnt[u];
            par[u] = par[v];
            return cnt[v];
        }
        else {
            rnk[u] = max(rnk[u], rnk[v] + 1);
            cnt[u] += cnt[v];
            par[v] = par[u];
        }
    }
    return cnt[u];
}
```

2.6: Range Minimum Query Using Sparse Table

```
const int Max = 1e5;
int ar[Max];
int tr[Max][18];
int n;
void Read() {
    for(int i = 0; i < n; i++) {
        scanf("%d", &ar[i]);
    }
}
void Build() {
    for(int i = 0; i < n; i++) {
        tr[i][0] = ar[i];
    }
    for(int j = 1; (1 << j) <= n; j++) {
        for(int i = 0; i + (1 << j) - 1 < n; i++) {
            tr[i][j] = max(tr[i][j - 1], tr[ i + (1 << (j - 1)) ][j - 1]);
        }
    }
}
int Query(int l, int r) {
    int k = log2(r - l + 1);
    return max(tr[l][k], tr[r - (1 << k) + 1][k]);
}
```

2.7: Binary Indexed Tree

```
const int Max = 1e5 + 10;

int ar[Max], n;
ll BIT[Max];

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

ll query(int idx) {
    ll ret = 0;
    while(idx > 0) {
        ret += BIT[idx];
        idx -= idx & -idx;
    }
    return ret;
}

ll query(int l, int r) {
    return query(r) - query(l - 1);
}

void build() {
    for(int i = 1; i <= n; i++) {
        update(i, ar[i]);
    }
}

int main() {
    int q, l, r;
    scanf("%d %d", &n, &q);
    for(int i = 1; i <= n; i++) {
        scanf("%d", &ar[i]);
    }
    build();
    while(q--) {
        scanf("%d %d", &l, &r);
        printf("%lld\n", query(l, r));
    }
    return 0;
}
```

2.8: Lazy Propagation

```
const int Max = 1e5 + 10;
ll ar[Max];

struct Node {
    ll prop, sum;
} seg[Max * 4];

void build(ll node, ll l, ll r) {
    if(l == r) {
        seg[node].sum = ar[l];
        return;
    }
    ll lf = node * 2;
    ll rt = node * 2 + 1;
    ll mid = (l + r) / 2;
    build(lf, l, mid);
    build(rt, mid + 1, r);
    seg[node].sum = seg[lf].sum + seg[rt].sum;
    seg[node].prop = 0;
}

ll query(ll node, ll l, ll r, ll L, ll R, ll cary = 0) {
    if(L > r || R < l) {
        return 0;
    }

    if(l >= L and r <= R) {
        return seg[node].sum + cary * (r - l + 1);
    }

    ll lf = node << 1;
    ll rt = (node << 1) + 1;
    ll mid = (l + r) >> 1;

    ll u = query(lf, l, mid, L, R, cary + seg[node].prop);
    ll v = query(rt, mid + 1, r, L, R, cary + seg[node].prop);
    return u + v;
}

void update(ll node, ll l, ll r, ll L, ll R, ll val) {
    if(L > r || R < l) {
        return;
    }
    if(l >= L && r <= R) {
        seg[node].sum += ((r - l + 1) * val);
        seg[node].prop += val;
        return;
    }
    ll lf = node * 2;
    ll rt = (node * 2) + 1;
    ll mid = (l + r) / 2;
    update(lf, l, mid, L, R, val);
    update(rt, mid + 1, r, L, R, val);
    seg[node].sum = seg[lf].sum + seg[rt].sum + (r - l + 1) * seg[node].prop;
}
```

2.9: Kth Element in Segment

```
int findkth(int node, int l, int r, int k) {
    if(l >= r) {
        return l;
    }

    int mid = (l + r) >> 1;

    if(seg[node * 2] >= k) {
        return findkth(node * 2, l, mid, k);
    }
    else {
        return findkth(node * 2 + 1, mid + 1, r, k - seg[node * 2]);
    }
}
```

2.10: Lowest Common Ancestor

```
int L[Max];
int P[Max][22];
int T[Max];
vector <int> G[Max];
```

```
void dfs(int u, int par, int dep) {
    T[u] = par;
    L[u] = dep;

    for(int v : G[u]) {
        if(v == par) {
            continue;
        }
        dfs(v, u, dep + 1);
    }
}

int lca_query(int p, int q) {
    if(L[p] < L[q]) {
        swap(p, q);
    }
    int lg = 1;
    while(1) {
        int nxt = lg + 1;
        if((1 << nxt) > L[p]) {
            break;
        }
        lg++;
    }
    for(int i = lg; i >= 0; i--) {
        if(L[p] - (1 << i) >= L[q]) {
            p = P[p][i];
        }
    }
    if(p == q) {
        return p;
    }
    for(int i = lg; i >= 0; i--) {
        if(P[p][i] != -1 && P[p][i] != P[q][i]) {
            p = P[p][i], q = P[q][i];
        }
    }
    return T[p];
}

void lca_init(int n) {
    memset(P, -1, sizeof(P));
    for(int i = 0; i < n; i++) {
        P[i][0] = T[i];
    }

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

int main() {
    int n, u, v, q;
    scanf("%d", &n);
    for(int i = 1; i < n; i++) {
        scanf("%d %d", &u, &v);
        G[u].push_back(v);
        G[v].push_back(u);
    }
    dfs(0, 0, 0);
    lca_init(n);
    scanf("%d", &q);
    while(q--) {
        scanf("%d %d", &u, &v);
        printf("%d\n", lca_query(u, v));
    }
    return 0;
}

/*
    Be careful about indexing for lca_init()
/*
```

2.10: BIT in Grid

```
int BIT[Max][Max];

void update(int x, int y) {
    for(int i = x; i < Max; i += i & -i) {
        for(int j = y; j < Max; j += j & -j) {
            BIT[i][j]++;
        }
    }
}

int query(int x, int y) {
    int ret = 0;

    for(int i = x; i > 0; i -= i & -i) {
        for(int j = y; j > 0; j -= j & -j) {
            ret += BIT[i][j];
        }
    }

    return ret;
}
```

2.11: Next Greater Element

```
// O(n)
vector<int> nextGreaterElement(vector<int> &arr) {
    int n = arr.size();
    stack<int> s;
    vector<int> ret(n + 1, n);

    for(int i = n - 1; i >= 0; i--) {
        while(!s.empty() && arr[s.top()] <= arr[i]) {
            s.pop();
        }

        if(!s.empty()) {
            ret[i] = s.top();
        }

        s.push(i);
    }
    return ret;
}
```

2.12: Sliding RMQ

```
//O(n)
vector<int> slidingRMQ(vector<int>  &arr, int k) {
    vector<int> ret(arr.size(), 1e9); // ret[i] = minimum of arr[i, i+k-1]
    deque<int> Q;

    for(int i = 0; i < arr.size(); i++) {
        while(!Q.empty() && Q.back() > arr[i]) {
            Q.pop_back();
        }

        Q.push_back(arr[i]);
        if(i >= k && arr[i - k] == Q.front()) {
            Q.pop_front();
        }

        if(i >= k - 1) {
            ret[i - k + 1] = Q.front();
        }
    } return ret;
}
```

2.13: Heavy Light Decomposition

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

typedef long long int ll;
typedef pair <ll, ll> pll;

const int Max = 3e4 + 10;
const int Mod = 1e9 + 7;
const ll Inf = 1LL << 62;

vector <int> G[Max];
int ar[Max];
int inp[Max];
int seg[Max * 6];
int cur, chainNo, pos[Max];
int L[Max];
int P[Max][22];
int T[Max];
int sub[Max];
int chainInd[Max];
int chainHead[Max];

void build(int node, int l, int r) {
    if(l == r) {
        seg[node] = ar[l];
        return;
    }
    int mid = (l + r) >> 1;
    int lf = node * 2;
    int rt = lf + 1;
    build(lf, l, mid);
    build(rt, mid + 1, r);
    seg[node] = seg[lf] + seg[rt];
}
```

```
int update(int node, int l, int r, int idx, int val) {
    if(l == r) {
        seg[node] = val;
        return seg[node];
    }
    int mid = (l + r) >> 1, u = seg[node * 2], v = seg[node * 2 + 1];
    if(mid >= idx) {
        u = update(2 * node, l, mid, idx, val);
    }
    else {
        v = update(2 * node + 1, mid + 1, r, idx, val);
    }
    return seg[node] = u + v;
}

int query(int node, int l, int r, int L, int R) {
    if(R < l || L > r) {
        return 0;
    }
    if(L <= l && r <= R) {
        return seg[node];
    }
    int mid = (l + r) >> 1;
    int u = query(node * 2, l, mid, L, R);
    int v = query(node * 2 + 1, mid + 1, r, L, R);
    return u + v;
}

int query_up(int u, int v) {
    int uchain;
    int vchain = chainInd[v];
    int ans = 0;
    while(true) {
        uchain = chainInd[u];
        if(uchain == vchain) {
            ans += query(1, 0, cur - 1, pos[v], pos[u]);
            break;
        }
        ans += query(1, 0, cur - 1, pos[chainHead[uchain]], pos[u]);
        u = chainHead[uchain];
        u = P[u][0];
    }
    return ans;
}

int lca_query(int p, int q) {
}

void lca_init(int n) {

}

int query(int u, int v) {
    int lca = lca_query(u, v);
    return query_up(u, lca) + query_up(v, lca) - ar[pos[lca]];
}

void update(int idx, int val) {
    ar[pos[idx]] = val;
    update(1, 0, cur - 1, pos[idx], val);
}

void HLD(int u, int prev) {
    if(chainHead[chainNo] == -1) {
        chainHead[chainNo] = u;
    }
    chainInd[u] = chainNo;
    pos[u] = cur;
    ar[cur++] = inp[u];
    int sc = -1, mx = -1;
    for(int i = 0; i < G[u].size(); i++) {
        int v = G[u][i];
        if(prev == v) {
            continue;
        }
        if(sub[v] > mx) {
            mx = sub[v];
            sc = v;
        }
    }
    if(sc != -1) {
        HLD(sc, u);
    }
    for(int i = 0; i < G[u].size(); i++) {
        int v = G[u][i];
        if(v != prev && v != sc) {
            chainNo++;
            HLD(v, u);
        }
    }
}

void dfs(int u, int par, int dep) {
    T[u] = par;
    L[u] = dep;
    sub[u] = 1;
    for(int i = 0; i < G[u].size(); i++) {
        int v = G[u][i];
        if(v == par) {
            continue;
        }
        dfs(v, u, dep + 1);
        sub[u] += sub[v];
    }
}

void Clear() {
    for(int i = 0; i < Max; i++) {
        G[i].clear();
    }
    cur = chainNo = 0;
    memset(chainHead, -1, sizeof chainHead);
}

int main() {
#ifdef Mr_Emrul
    freopen("inputf.in", "r", stdin);
#endif /// Mr_Emrul

    int t, n, q, u, v, ty, idx, val;
    scanf("%d", &t);
    for(int tc = 1; tc <= t; tc++) {
        Clear();
        scanf("%d", &n);
        for(int i = 0; i < n; i++) {
            scanf("%d", &inp[i]);
        }
        for(int i = 1; i < n; i++) {
            scanf("%d %d", &u, &v);
            G[u].push_back(v);
            G[v].push_back(u);
        }
        dfs(0, -1, 0);
        HLD(0, -1);
        lca_init(n);
        build(1, 0, cur - 1);
        printf("Case %d:\n", tc);
        scanf("%d", &q);
        while(q--) {
            scanf("%d", &ty);
            if(ty == 0) {
                scanf("%d %d", &u, &v);
                printf("%d\n", query(u, v));
            }
            else {
                scanf("%d %d", &idx, &val);
                update(idx, val);
            }
        }
    }
    return 0;
}
```

2.14: Trie

```
struct Node {
      bool endmark;
      int cnt;
      Node *nxt[26 + 1];
      Node() {
            endmark = false;
            cnt = 0;
            for(int i = 0; i < 26; i++) {
                  nxt[i] = NULL;
            }
      }
} *root;

void insert(char *str, int len) {
      Node *cur = root;
      for(int i = 0; i < len; i++) {
            int id = str[i] - 'a';
            if(cur->nxt[id] == NULL) {
                  cur->nxt[id] = new Node();
            }
            cur = cur->nxt[id];
            cur->cnt++;
      }
      cur->endmark = true;
}

bool search(char *str, int len) {
      Node *cur = root;
      for(int i = 0; i < len; i++) {
            int id = str[i] - 'a';
            if(cur->nxt[id] == NULL) {
                  return false;
            }
            cur = cur->nxt[id];
      }
      return cur->endmark;
}

int query(char *str, int len) {
      int ans = 0;
      Node *cur = root;
      for(int i = 0; i < len; i++) {
            int id = str[i] - 'a';
            if(cur->nxt[id] == NULL) {
                  return false;
            }
            cur = cur->nxt[id];
            ans = max(ans, (i + 1) * cur->cnt);
      }
      return ans;
}

void del(Node *cur) {
      for(int i = 0; i < 26; i++) {
            if(cur->nxt[i]) {
                  del(cur->nxt[i]);
            }
      }
      delete(cur);
}
```

2.15: 2D Segment Tree

```
const int Max = 2001;
int seg[4 * Max][4 * Max], n, m, ar[Max][Max];

void buildY(int ndx, int lx, int rx, int ndy, int ly, int ry) {
    if(ly == ry) {
        if(lx == rx) {
            seg[ndx][ndy] = ar[lx][ly];
        }
        else {
            seg[ndx][ndy] = seg[ndx * 2][ndy] + seg[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);
    seg[ndx][ndy] = seg[ndx][ndy * 2] + seg[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) {
            seg[ndx][ndy] = val;
        }
        else {
            seg[ndx][ndy] = seg[ndx * 2][ndy] + seg[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);
    }
    seg[ndx][ndy] = seg[ndx][ndy * 2] + seg[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 seg[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);
}
```

2.16: Maximum Histrogram

```
ll histro(vector<ll> &hist) {
    stack <int> st;
    ll res = 0;
    for(int i = 0; i <= hist.size(); i++) {
        ll h = (i == n ? 0 : hist[i]);
        if(st.empty() || h >= hist[st.top()]) {
            st.push(i++);
        }
        else {
            int x = st.top();
            st.pop();
            res = max(res, hist[x] * (st.empty() ? i : i - 1 - st.top()));
        }
    }
    return res;
}
```

2.17: MO’s Algo with Update

```
/*
```

```
Tested Problem:

You are given an array a. You have to answer the following queries:
        1. You are given two integers l and r. Let c[i] be the number of
occurrences of i in a[l: r], where a[l: r] is the subarray of a from l-th
element to r-th inclusive. Find the Mex of {c[0], c[1], ..., c[10^9]}
        2. You are given two integers p to x. Change a[p] to x.
The Mex of a multiset of numbers is the smallest non-negative integer not in
the set.

*/

const int Max = 1e5 + 10;

inline void read(int &res) {
    char c;
    while(c = getchar(), c <= ' ');
    res = c - '0';
    while(c = getchar(), c >= '0' && c <= '9') {
        res = res * 10 + (c - '0');
    }
}

int n, m, ar[Max], prv[Max], ans[Max];
int block, sz;

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 MEX {
    int cnt[555], freq[Max * 2];
    MEX() {
        memset(cnt, 0, sizeof cnt);
        memset(freq, 0, sizeof freq);
    }
    void add(int x) {
        if(!freq[x]) {
            ++cnt[x / sz];
        }
        ++freq[x];
    }
    void remove(int x) {
        if(freq[x] == 1) {
            --cnt[x / sz];
        }
        --freq[x];
    }
    int get() {
        for(int i = 0; i <= sz; i++) {
            if(cnt[i] < sz) {
                for(int j = i * sz, ed = j + sz; j < ed; j++) {
                    if(!freq[j]) {
                        return j;
                    }
                }
            }
        }
    }
} ds;

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

void add(int x) {
    x = ar[x];
    int &c = cnt[x];
    if(c) {
        ds.remove(c);
    }
    ++c;
    ds.add(c);
}
```

```
void remove(int x) {
    x = ar[x];
    int &c = cnt[x];
    ds.remove(c);
    --c;
    if(c) {
        ds.add(c);
    }
}

void apply(int i, int x) {
    if(l <= i && i <= r) {
        remove(i);
        ar[i] = x;
        add(i);
    }
    else {
        ar[i] = x;
    }
}

int main() {
    vector <int> v;
    read(n);
    read(m);
    for(int i = 0; i < n; ++i) {
        read(ar[i]);
        v.push_back(ar[i]);
    }
    block = pow(n, 0.667);
    int T[m], L[m], R[m];
    for(int i = 0; i < m; ++i) {
        read(T[i]);
        read(L[i]);
        read(R[i]);
        if(T[i] == 2) {
            v.push_back(R[i]);
        }
    }
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());
    map<int, int> mp;
    for(int i = 0; i < v.size(); i++) {
        mp[v[i]] = i + 1;
    }
    for(int i = 0; i < n; i++) {
        ar[i] = prv[i] = mp[ar[i]];
    }
    for(int i = 0; i < m; i++) {
        if(T[i] == 2) {
            R[i] = mp[R[i]];
        }
    }
    sz = sqrt((double)v.size()) + 2;
    u.push_back({-1, -1, -1});
    for(int i = 0; i < m; ++i) {
        int t = T[i], l = --L[i], r = R[i];
        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;
    ds.add(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]);
    }
    return 0;
}
```

2.18: 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], ...)
/*////////////////////////////////////////
```

2.18.1: Version in Segment Tree

```
const int Max = 1e5 + 10;

struct Node {
    int l, r, sum;
} seg[Max * 20];
int root[Max], ar[Max], n, q, idx;

void build(int nd, int l, int r) {
    if(l == r) {
        seg[nd].sum = ar[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(seg[nd].l = ++idx, l, mid);
    build(seg[nd].r = ++idx, mid + 1, r);
    seg[nd].sum = seg[seg[nd].l].sum + seg[seg[nd].r].sum;
}

void update(int &nd, int l, int r, int &i, int &val) {
    seg[++idx] = seg[nd];
    seg[nd = idx].sum += val;
    if(l == r) {
        return;
    }
    int mid = (l + r) >> 1;
    if(i <= mid) {
        update(seg[nd].l, l, mid, i, val);
    }
    else {
        update(seg[nd].r, mid + 1, r, i, val);
    }
}

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 seg[nd].sum;
    }
    int mid = (l + r) >> 1;
    return query(seg[nd].l, l, mid, i, j) + query(seg[nd].r, mid + 1, r, i,
j);
}

int main() {
    scanf("%d", &n);
    for(int i = 1; i <= n; i++) {
        scanf("%d", &ar[i]);
    }
    build(root[0], 1, n);
    scanf("%d", &q);
    int ver = 0;
    while(q--) {
        int ty, idx, l, r, pos, v;
        scanf("%d", &ty);
        if(ty == 1) {
            scanf("%d %d %d", &idx, &pos, &v);
            update(root[++ver] = root[idx], 1, n, pos, v);
        }
        else {
            scanf("%d %d %d", &idx, &l, &r);
            printf("%d\n", query(root[idx], 1, n, l, r));
        }
    }
    return 0;
}
```

2.18.2: Less than or k in a Range

```
const int Max = 1e5 + 10;

struct Node {
    int l, r, sum;
} seg[Max * 20];
int root[Max], ar[Max], n, q, idx, M = 1e5; /* M = Number of unique elements
*/

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

int lesscnt(int a, int b, int l, int r, int k) {
    if(r <= k) {
        return seg[a].sum - seg[b].sum;
    }
    int mid = (l + r) >> 1;
    if(k <= mid) {
        return lesscnt(seg[a].l, seg[b].l, l, mid, k);
    }
    else return lesscnt(seg[a].l, seg[b].l, l, mid, k) +
                 lesscnt(seg[a].r, seg[b].r, mid + 1, r, k);
}

void init() {
    seg[0].l = seg[0].r = seg[0].sum = 0;
    for(int i = 1; i <= n; ++i) {
        update(root[i] = root[i - 1], 0, M, ar[i]);
    }
}
```

2.18.3: Kth Number in a Range

```
/// update(), init() is as like as 2.18.1

int kthnum(int a, int b, int l, int r, int k) {
    if(l == r) {
        return l;
    }
    int cnt = seg[seg[a].l].sum - seg[seg[b].l].sum;
    int mid = (l + r) >> 1;
    if(cnt >= k) {
        return kthnum(seg[a].l, seg[b].l, l, mid, k);
    }
    else {
        return kthnum(seg[a].r, seg[b].r, mid + 1, r, k - cnt);
    }
}
```

Part 3 ⇛ Utilities

3.1: Bit Manipulation

```
ll Set(ll num, ll pos) {
    return num | (1LL << pos);
}

ll Clear(ll num, ll pos) {
    return num & ~(1LL << pos);
}

ll Toggle(ll num, ll pos) {
    return num ^ (1LL << pos);
}

bool Check(ll num, ll pos) {
    return (bool)(num & (1LL << pos));
}
```

3.2: Policy Based Data Structure

```
#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;

template <typename T> using     Set = tree<T, null_type,
                                less<T>, rb_tree_tag,
                                tree_order_statistics_node_update>;

Set <int> st;

int main() {
    st.insert(5);   //Insert
    st.erase(5);   //Delete
    st.insert(1);
    st.insert(2);
    st.insert(9);
    cout << *st.find_by_order(0) << endl;   //Find value by rank
    cout << st.order_of_key(9) << endl;   //Find value's rank

    /* For multiple same element, use pair, store index in second of pair */
    return 0;
}
```

3.3: Direction Array

```
int dx4[] = {0, 0, -1, 1};
int dy4[] = {1, -1, 0, 0};

int dx8[] = {1, 1, 1, 0, 0, -1, -1, -1};
int dy8[] = {1, 0, -1, 1, -1, 1, 0, -1};

int dx_horse[] = {1, 1, -1, -1, 2, 2, -2, -2}
Int dy_horse[] = {2, -2, 2, -2, 1, -1, 1, -1}
```

3.4: Big Integer Template

```
#include <cstdio>
#include <string>
#include <algorithm>
#include <iostream>

using namespace std;

struct Bigint {
  // representations and structures
  string a; // to store the digits
  int sign; // sign = -1 for negative numbers, sign = 1 otherwise

  // constructors
  Bigint() {} // default constructor
  Bigint(string b) {
    (*this) = b;    // constructor for string
  }

  // some helpful methods
  int size() { // returns number of digits
    return a.size();
  }
  Bigint inverseSign() { // changes the sign
    sign *= -1;
    return (*this);
  }
  Bigint normalize(int newSign) {   // removes leading 0, fixes sign
    for(int i = a.size() - 1; i > 0 && a[i] == '0'; i--) {
      a.erase(a.begin() + i);
    }
    sign = (a.size() == 1 && a[0] == '0') ? 1 : newSign;
    return (*this);
  }
  // assignment operator
  void operator = (string b) {   // assigns a string to Bigint
    a = b[0] == '-' ? b.substr(1) : b;
    reverse(a.begin(), a.end());
    this->normalize(b[0] == '-' ? -1 : 1);
  }

  // conditional operators
  bool operator < (const Bigint &b) const {   // less than operator
    if(sign != b.sign) {
      return sign < b.sign;
    }
    if(a.size() != b.a.size()) {
      return sign == 1 ? a.size() < b.a.size() : a.size() > b.a.size();
    }
    for(int i = a.size() - 1; i >= 0; i--) if(a[i] != b.a[i]) {
        return sign == 1 ? a[i] < b.a[i] : a[i] > b.a[i];
      }
    return false;
  }
  bool operator == (const Bigint &b) const {   // operator for equality
    return a == b.a && sign == b.sign;
  }

  // mathematical operators
  Bigint operator + (Bigint b) {   // addition operator overloading
    if(sign != b.sign) {
      return (*this) - b.inverseSign();
    }
    Bigint c;
    for(int i = 0, carry = 0; i < a.size() || i < b.size() || carry; i++) {
      carry += (i < a.size() ? a[i] - 48 : 0) + (i < b.a.size() ? b.a[i] - 48
: 0);
      c.a += (carry % 10 + 48);
      carry /= 10;
    }
    return c.normalize(sign);
  }
  Bigint operator - (Bigint b) {   // subtraction operator overloading
    if(sign != b.sign) {
      return (*this) + b.inverseSign();
    }
    int s = sign;
    sign = b.sign = 1;
    if((*this) < b) {
      return ((b - (*this)).inverseSign()).normalize(-s);
    }
    Bigint c;
    for(int i = 0, borrow = 0; i < a.size(); i++) {
      borrow = a[i] - borrow - (i < b.size() ? b.a[i] : 48);
      c.a += borrow >= 0 ? borrow + 48 : borrow + 58;
      borrow = borrow >= 0 ? 0 : 1;
    }
    return c.normalize(s);
  }
  Bigint operator * (Bigint b) {   // multiplication operator overloading
    Bigint c("0");
    for(int i = 0, k = a[i] - 48; i < a.size(); i++, k = a[i] - 48) {
      while(k--) {
        c = c + b;    // ith digit is k, so, we add k times
      }
      b.a.insert(b.a.begin(), '0'); // multiplied by 10
    }
    return c.normalize(sign * b.sign);
  }
  Bigint operator / (Bigint b) {   // division operator overloading
    if(b.size() == 1 && b.a[0] == '0') {
      b.a[0] /= (b.a[0] - 48);
    }
    Bigint c("0"), d;
    for(int j = 0; j < a.size(); j++) {
      d.a += "0";
    }
    int dSign = sign * b.sign;
    b.sign = 1;
    for(int i = a.size() - 1; i >= 0; i--) {
      c.a.insert(c.a.begin(), '0');
      c = c + a.substr(i, 1);
      while(!(c < b)) {
        c = c - b, d.a[i]++;
      }
    }
    return d.normalize(dSign);
  }
  Bigint operator % (Bigint b) {   // modulo operator overloading
    if(b.size() == 1 && b.a[0] == '0') {
      b.a[0] /= (b.a[0] - 48);
    }
    Bigint c("0");
    b.sign = 1;
    for(int i = a.size() - 1; i >= 0; i--) {
      c.a.insert(c.a.begin(), '0');
      c = c + a.substr(i, 1);
      while(!(c < b)) {
        c = c - b;
      }
    }
    return c.normalize(sign);
  }

  // output method
  void print() {
    if(sign == -1) {
      putchar('-');
    }
    for(int i = a.size() - 1; i >= 0; i--) {
      putchar(a[i]);
    }
  }
};

int main() {
    Bigint a, b, c; // declared some Bigint variables

    /////////////////////////
    // taking Bigint input //
    /////////////////////////
    string input; // string to take input

    cin >> input; // take the Big integer as string
    a = input; // assign the string to Bigint a

    cin >> input; // take the Big integer as string
    b = input; // assign the string to Bigint b

    //////////////////////////////////
    // Using mathematical operators //
    //////////////////////////////////

    c = a + b; // adding a and b
    c.print(); // printing the Bigint
    puts(""); // newline

    c = a - b; // subtracting b from a
    c.print(); // printing the Bigint
    puts(""); // newline

    c = a * b; // multiplying a and b
    c.print(); // printing the Bigint
    puts(""); // newline

    c = a / b; // dividing a by b
    c.print(); // printing the Bigint
    puts(""); // newline

    c = a % b; // a modulo b
    c.print(); // printing the Bigint
    puts(""); // newline

    /////////////////////////////////
    // Using conditional operators //
    /////////////////////////////////

    if(a == b) {
        puts("equal");    // checking equality
    }
    else {
        puts("not equal");
    }

    if(a < b) {
        puts("a is smaller than b");    // checking less than operator
    }

    return 0;
}
```

3.5: Character Wise I/O Method

```
#define gc getchar_unlocked
#define pc putchar_unlocked

inline void Cin(int &num) {
    num = 0;
    char c = gc();
    int flag = 0;

    while(!((c >= '0' & c <= '9') || c == '-')) {
        c = gc();
    }

    if(c == '-') {
        flag = 1;
        c = gc();
    }

    while(c >= '0' && c <= '9') {
        num = (num << 1) + (num << 3) + c - '0';
        c = gc();
    }

    if(flag == 1) {
        num = 0 - num;
    }
}

inline void Cout(int n) {
    int NN = n, rev, count = 0;
    rev = NN;
    if(NN == 0) {
        pc('0');
        return;
    }
    while((rev % 10) == 0) {
        count++;
        rev /= 10;
    }
    rev = 0;
    while(NN != 0) {
        rev = (rev << 3) + (rev << 1) + NN % 10;
        NN /= 10;
    }
    while(rev != 0) {
        pc(rev % 10 + '0');
        rev /= 10;
    }
    while(count--) {
        pc('0');
    }
}
```

Part 4 ⇛ String

4.1: Hashing

```
const int base = 331;

const int Max = 2e6 + 10;
const int Mod = 1e9 + 7;
const ll Inf = 1LL << 62;

ll pw[Max];
ll Hash[Max];

void pre_power() {
    pw[0] = 1;
    for(int i = 1; i < Max; i++) {
        pw[i] = (pw[i - 1] * base) % Mod;
    }
}

void Hashing(string str, int len) {
    ll hash_val = 0;
    for(int i = 0; i < len; i++) {
        hash_val = (hash_val * base + str[i]) % Mod;
        Hash[i + 1] = hash_val;
    }
}

ll SubstringHash(int l, int r) {
    return (Hash[r] - (Hash[l - 1] * pw[r - l + 1]) % Mod + Mod) % Mod;
}
```

4.2: Z Algorithm

```
//Longest substring from every index which is the prefix of the string

int Z[Max];
char S[Max];

void compute_z_function(int n) {
      int L, R, k;
      L = 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 {
                    k = i - L;
                    if(Z[k] < R - i + 1) {
                        Z[i] = Z[k];
                    }
                    else {
                        L = i;
                        while(R < n && S[R - L] == S[R]) {
                            R++;
                        }
                        Z[i] = R - L;
                        R--;
                    }
                }
        }
}

int main() {
    scanf("%s", S);
    compute_z_function(strlen(S));
    for(int i = 0; i < strlen(S); i++) {
        printf("%d ", Z[i]);
    }
    return 0;
}
```

4.3: KMP

```
int kmp(const string &T, const string &P) {
    if (P.empty()) return 0;

    vector<int> pi(P.size(), 0);
    for (int i = 1, k = 0; i < P.size(); ++i) {
        while (k && P[k] != P[i]) k = pi[k - 1];
        if (P[k] == P[i]) ++k;
        pi[i] = k;
    }

    for (int i = 0, k = 0; i < T.size(); ++i) {
        while (k && P[k] != T[i]) k = pi[k - 1];
        if (P[k] == T[i]) ++k;
        if (k == P.size()) return i - k + 1;
    }

    return -1;
}
```

4.4: Suffix Array

```
const int Max = 1e5 + 10;

char str[Max];

int n, len;
int sa[Max], pos[Max], tmp[Max], lcp[Max];

bool sufCmp(int i, int j) {
    if(pos[i] != pos[j]) {
        return pos[i] < pos[j];
    }
    i += len;
    j += len;
    return (i < n && j < n) ? pos[i] < pos[j] : i > j;
}

void buildSA() {
    memset(sa, 0, sizeof sa);
    memset(pos, 0, sizeof pos);
    memset(lcp, 0, sizeof lcp);
    memset(tmp, 0, sizeof tmp);
    n = strlen(str);
    for(int i = 0; i < n; i++) {
        sa[i] = i, pos[i] = str[i];
    }
    for(len = 1;; len *= 2) {
        sort(sa, sa + n, sufCmp);
        for(int i = 0; i < n - 1; i++) {
            tmp[i + 1] = tmp[i] + sufCmp(sa[i], sa[i + 1]);
        }
        for(int i = 0; i < n; i++) {
            pos[sa[i]] = tmp[i];
        }
        if(tmp[n - 1] == n - 1) {
            break;
        }
    }
}
```

```
void buildLCP() {
    for(int i = 0, k = 0; i < n; i++) {
        if(pos[i] != n - 1) {
            for(int j = sa[pos[i] + 1]; str[i + k] == str[j + k];) {
                k++;
            }
            lcp[pos[i]] = k;
            if(k) {
                k--;
            }
        }
    }
    for(int i = n - 1; i > 0; i--) {
        lcp[i] = lcp[i - 1];
    }
    lcp[0] = 0;
}
```

4.5: Aho Corasick Algorithm

```
/*
    Algo: Given a string inp and some pattern strings s[i], Count of s[i] for
every i as substring in inp
*/

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

const int sz = 1e6 + 10;
const int Max = 505 * 505 + 100;
char inp[sz], s[505][505];
int cnt[505];

struct AhoCorasick {
    vector < int > mark[Max + 7];
    int state, failure[Max + 7];
    int trie[Max + 7][26];

    AhoCorasick() {
        init();
    }

    void init() {
        mark[0].clear();
        fill(trie[0], trie[0] + 26, -1);
        state = 0;
```

```
}

int value(char c) {
    return c - 'a';
}

/*
    Adding s[t] in trie
*/
void add(char *s, int t) {
    int root = 0, id;
    for(int i = 0; s[i]; i++) {
        id = value(s[i]);
        if(trie[root][id] == -1) {
            trie[root][id] = ++state;
            mark[state].clear();
            fill(trie[state], trie[state + 1] + 26, - 1);
        }
        root = trie[root][id];
    }
    mark[root].push_back(t);
}

/*
    Failure function
*/
void computeFailure() {
    queue < int > Q;
    failure[0] = 0;
    for(int i = 0; i < 26; i++) {
        if(trie[0][i] != -1) {
            failure[trie[0][i]] = 0;
            Q.push(trie[0][i]);
        }
        else {
            trie[0][i] = 0;
        }
    }
    while(!Q.empty()) {
        int u = Q.front();
        Q.pop();
        for(int v : mark[failure[u]]) {
            mark[u].push_back(v);
        }
        for(int i = 0; i < 26; i++) {
            if(trie[u][i] != -1) {
                failure[trie[u][i]] = trie[failure[u]][i];
                Q.push(trie[u][i]);
                }
                else {
                    trie[u][i] = trie[failure[u]][i];
                }
            }
        }
    }
} automata;

void countFreq() {
    for(int i = 0, root = 0, id; inp[i]; i++) {
        id = automata.value(inp[i]);
        root = automata.trie[root][id];
        if(root == 0) {
            continue;
        }
        for(int v : automata.mark[root]) {
            cnt[v]++;
        }
    }
}

int main() {
    int t, n, m;
    scanf("%d", &t);
    for(int tc = 1; tc <= t; tc++) {
        scanf("%d", &n);
        scanf("%s", inp);
        automata.init();
        memset(cnt, 0, sizeof cnt);
        for(int i = 0; i < n; i++) {
            scanf("%s", s[i]);
            automata.add(s[i], i);
        }
        automata.computeFailure();
        countFreq();
        printf("Case %d:\n", tc);
        for(int i = 0; i < n; i++) {
            printf("%d\n", cnt[i]);
        }
    }
}
```

4.6: Palindrome Tree

```
/*
    Palindrome tree. Useful structure to deal with palindromes in strings.
O(N)
    This code counts number of palindrome substrings of the string.
*/

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

const int Max = 105000;

struct node {
    int next[26];
    int len;
    int sufflink;
    int num;
};

int len;
char s[Max];
node tree[Max];
int num;            // node 1 - root with len -1, node 2 - root with len 0
int suff;           // max suffix palindrome
long long ans;

bool addLetter(int pos) {
    int cur = suff, curlen = 0;
    int let = s[pos] - 'a';

    while(true) {
        curlen = tree[cur].len;
        if(pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
            break;
        }
        cur = tree[cur].sufflink;
    }
    if(tree[cur].next[let]) {
        suff = tree[cur].next[let];
        return false;
    }

    num++;
    suff = num;
    tree[num].len = tree[cur].len + 2;
    tree[cur].next[let] = num;
    if(tree[num].len == 1) {
        tree[num].sufflink = 2;
        tree[num].num = 1;
        return true;
    }

    while(true) {
        cur = tree[cur].sufflink;
        curlen = tree[cur].len;
        if(pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
            tree[num].sufflink = tree[cur].next[let];
            break;
        }
    }

    tree[num].num = 1 + tree[tree[num].sufflink].num;

    return true;
}

void initTree() {
    num = 2;
    suff = 2;
    tree[1].len = -1;
    tree[1].sufflink = 1;
    tree[2].len = 0;
    tree[2].sufflink = 1;
}

int main() {
    scanf("%s", s);
    len = strlen(s);
    initTree();
    for(int i = 0; i < len; i++) {
        addLetter(i);
        ans += tree[suff].num;
    }
    printf("%lld\n", ans);
    return 0;
}
```

Part 5 ⇛ Math

5.1: Geometry formulas

5.1.1: Perimeter

Perimeter of a square: $s + s + s + s$
s:length of one side

Perimeter of a rectangle: $l + w + l + w$
l: length
w: width

Perimeter of a triangle: $a + b + c$
a, b, and c: lengths of the 3 sides

5.1.2: Area

Area of a square: $s \times s$
s: length of one side

Area of a rectangle: $l \times w$
l: length
w: width

Area of a triangle: $(b \times h)/2$
b: length of base
h: length of height

Area of a trapezoid: $(b_1 + b_2) \times h/2$
b_1 and b_2: parallel sides or the bases
h: length of height

5.1.3: Volume

Volume of a cube: $s \times s \times s$
s: length of one side

Volume of a box: $l \times w \times h$
l: length
w: width
h: height

Volume of a sphere: $(4/3) \times pi \times r^3$

pi: 3.14
r: radius of sphere

Volume of a triangular prism: area of triangle × Height = (1/2 base × height) × Height
base: length of the base of the triangle
height: height of the triangle
Height: height of the triangular prism

Volume of a cylinder: pi × r^2 × Height
pi: 3.14
r: radius of the circle of the base
Height: height of the cylinder

5.2: Geometry Template

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

#define PI acos(-1)
const double INF = 1e4;
const double EPS = 1e-10;

struct Point {
    double x, y;

    Point() {}
    Point(double x, double y): x(x), y(y) {}
    Point(const Point &p): x(p.x), y(p.y) {}

    void input() {
        scanf("%lf%lf", &x, &y);
    }

    Point operator + (const Point &p) const {
        return Point(x + p.x, y + p.y);
    }

    Point operator - (const Point &p) const {
        return Point(x - p.x, y - p.y);
    }

    Point operator * (double c) const {
        return Point(x * c, y * c);
    }
    Point operator / (double c) const {
        return Point(x / c, y / c);
    }
};

vector<Point>polygon;

double getClockwiseAngle(Point p) {
    return -1 * atan2(p.x, -1 * p.y);
}

//compare function to compare clockwise
bool comparePoints(Point p1, Point p2) {
    return getClockwiseAngle(p1) < getClockwiseAngle(p2);
}

// rotate 90 degrees counter clockwise
Point RotateCCW90(Point p) {
    return Point(-p.y, p.x);
}

// rotate 90 degrees clockwise
Point RotateCW90(Point p) {
    return Point(p.y, -p.x);
}

Point RotateCCW(Point p, double t) {
    return Point(p.x * cos(t) - p.y * sin(t), p.x * sin(t) + p.y * cos(t));
}

Point RotateCW(Point p, double t) {
    return Point(p.x * cos(t) + p.y * sin(t), -p.x * sin(t) + p.y * cos(t));
}

double dot(Point A, Point B) {
    return A.x * B.x + A.y * B.y;
}

double cross(Point A, Point B) {
    return A.x * B.y - A.y * B.x;
}
double dist2(Point A, Point B) {
    return dot(A - B, A - B);
}

// returns distance between two points
double dist(Point A, Point B) {
    return sqrt(dot(A - B, A - B));
```

```
}

// Distance between point A and B
double distBetweenPoint(Point A, Point B) {
    return sqrt(dot(A - B, A - B));
}

// project point c onto line AB (A!=B)
Point ProjectPointLine(Point A, Point B, Point C) {
    return A + (B - A) * dot(C - A, B - A) / dot(B - A, B - A);
}

// Determine if Line AB and CD are parallel or collinear
bool LinesParallel(Point A, Point B, Point C, Point D) {
    return fabs(cross(B - A, C - D)) < EPS;
}

// Determine if Line AB and CD are collinear
bool LinesCollinear(Point A, Point B, Point C, Point D) {
    return LinesParallel(A, B, C, D) && fabs(cross(A - B, A - C)) < EPS &&
fabs(cross(C - D, C - A)) < EPS;
}

// checks if AB intersect with CD
bool SegmentIntersect(Point A, Point B, Point C, Point D) {
    if(LinesCollinear(A, B, C, D)) {
        if(dist2(A, C) < EPS || dist2(A, D) < EPS || dist2(B, C) < EPS ||
dist2(B, D) < EPS) {
            return true;
        }
        if(dot(C - A, C - B) > 0 && dot(D - A, D - B) > 0 && dot(C - B, D - B)
> 0) {
            return false;
        }
        return true;
    }
    if(cross(D - A, B - A) * cross(C - A, B - A) > 0) {
        return false;
    }
    if(cross(A - C, D - C) * cross(B - C, D - C) > 0) {
        return false;
    }
    return true;
}

// Compute the coordinates where AB and CD intersect
Point ComputeLineIntersection(Point A, Point B, Point C, Point D) {
    double a1, b1, c1, a2, b2, c2;
    a1 = A.y - B.y;
    b1 = B.x - A.x;
    c1 = cross(A, B);
    a2 = C.y - D.y;
    b2 = D.x - C.x;
    c2 = cross(C, D);
    double Dist = a1 * b2 - a2 * b1;
    return Point((b1 * c2 - b2 * c1) / Dist, (c1 * a2 - c2 * a1) / Dist);
}

//Project point C onto line segment AB -- return the Point from AB which is
the closest to C --
Point ProjectPointSegment(Point A, Point B, Point C) {
    double r = dot(B - A, B - A);
    if(fabs(r) < EPS) {
        return A;
    }
    r = dot(C - A, B - A) / r;
    if(r < 0) {
        return A;
    }
    if(r > 1) {
        return B;
    }
    return A + (B - A) * r;
}

// return the minimum distance from a point C to a line AB
double DistancePointSegment(Point A, Point B, Point C) {
    return distBetweenPoint(C, ProjectPointSegment(A, B, C));
}

// return distance between P and a point where p is perpendicular on AB. AB er
upore p jei point e lombo shei point theke p er distance
double distToLine(Point p, Point a, Point b) {
    pair<double, double>c;
    double scale = (double)(dot(p - a, b - a)) / (dot(b - a, b - a));
    c.first = a.x + scale * (b.x - a.x);
    c.second = a.y + scale * (b.y - a.y);
    double dx = (double)p.x - c.first, dy = (double)p.y - c.second;
    return sqrt(dx * dx + dy * dy);
}

long long orientation(Point p, Point q, Point r) {
    long long val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);
    if(val > 0) {
        return 1;
    }
    if(val < 0) {
        return 2;
    }
    else {
        return val;
    }

}

// Given three colinear points p, q, r, the function checks if
// point q lies on line segment 'pr'
bool onSegment(Point p, Point q, Point r) {
    if(q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) && q.y <= max(p.y, r.y) &&
q.y >= min(p.y, r.y)) {
        return true;
    }
    return false;
}

//checks if Point P is inside of polygon or not
bool isInside(int n, Point p) {
    if(n < 3) {
        return false;
    }
    Point extreme = Point(INF, p.y); // here INF=1e4
    int count = 0, i = 0;
    do {
        int next = (i + 1) % n;
        if(SegmentIntersect(polygon[i], polygon[next], p, extreme)) {
            if(orientation(polygon[i], p, polygon[next]) == 0) {
                return onSegment(polygon[i], p, polygon[next]);
            }
            count++;
        }
        i = next;
    } while(i != 0);
    return count & 1;
}

// returns the perimeter of a polygon
double polygonPerimeter(int n) {
    double perimeter = 0.0;
    for(int i = 0; i < n - 1; i++) { //polygon vector holds the corner points
of the given polygon
        perimeter += dist(polygon[i], polygon[i + 1]);
    }
    perimeter += dist(polygon[0], polygon[n - 1]);
    return perimeter;
```

```
}

// returns the area of a polygon
double polygonArea(int n) {
    double area = 0.0;
    int j = n - 1;
    for(int i = 0; i < n; i++) {
        area += (polygon[j].x + polygon[i].x) * (polygon[j].y - polygon[i].y);
        j = i;
    }
    return fabs(area) * 0.5;
}

double getTriangleArea(Point a, Point b, Point c) {
    return fabs(cross(b - a, c - a));
}

bool compareConvex(Point X, Point Y) {
    long long ret = orientation(points[0], X, Y);
    if(ret == 0) {
        long long dist11 = dist2(points[0], X);
        long long dist22 = dist2(points[0], Y);
        return dist11 < dist22 ;
    }
    else if(ret == 2) {
        return true ;
    }
    else {
        return false ;
    }
}

Point nextToTop(stack<Point> &S) {
    Point p = S.top();
    S.pop();
    Point res = S.top();
    S.push(p);
    return res;
}

// make a minimum area polygon
stack<Point> convexHull(int N) {
    int ymin = points[0].y, index = 0 ;
    for(int i = 1; i < N; i++) {
        if(points[i].y < ymin || (points[i].y == ymin && points[i].x <
points[index].x)) {
            ymin = points[i].y ;
            index = i ;
        }
    }
    stack<Point>S;
    swap(points[0], points[index]);
    sort(&points[1], &points[N], compareConvex);
    S.push(points[0]);
    for(int i = 1; i < N; i++) {
        while(S.size() > 1 && orientation(nextToTop(S), S.top(), points[i]) !=
2) {
            S.pop();
        }
        S.push(points[i]);
    }
    return S;
}

// Angle between Line AB and AC in degree
double angle(Point B, Point A, Point C) {
    double c = dist(A, B);
    double a = dist(B, C);
    double b = dist(A, C);
    double ans = acos((b * b + c * c - a * a) / (2 * b * c));
    return (ans * 180) / acos(-1);

}

// returns number of vertices on boundary of a polygon
long long picks_theorem_boundary_count() {
    int sz = polygon.size(), i ;
    long long res = __gcd((long long)abs(polygon[0].x - polygon[sz - 1].x),
(long long)abs(polygon[0].y - polygon[sz - 1].y));
    for(i = 0; i < sz - 1 ; i++) {
        res += __gcd((long long)abs(polygon[i].x - polygon[i + 1].x), (long
long)abs(polygon[i].y - polygon[i + 1].y));
    }
    return res;
}

// picks theorem
// Polygon area= inside points + boundary points/2 -1
// return inside points counts
long long lattice_points_inside_polygon() {
    long long ar = polygonArea(n);
    long long b = picks_theorem_boundary_count();
    long long tot = ar + 1 - b / 2;
    return tot;
}
```

5.3: Number Theory Template

```
#define ll long long
#define ull unsigned long long
ll num[50], rem[50];
vector<ll>parts
char s[100005];

// generate pairwise coprime divisors of N.
void computeCoprimeDivisor() {
    for(int i = 2; i < MX; i++) {
        if(prime[i] == 0) {
            // pr vector holds the divisor of i which is pairwise coprime
            pr[i].push_back(i);
            {
                for(int j = i + i; j < MX; j += i) {
                    prime[j] = 1;
                    pr[j].push_back(fnc(j, i));

                }
            }
        }
    }
    ////////////////////////// EXTRA START /////////////////////////////
    primes[0] = 2;
    int k = 1;
    for(int i = 3; i < MX; i += 2) { // storing primes
        if(prime[i] == 0) {
            primes[k++] = i;
        }
    }

    for(int i = 0; i < k; i++) { // computing perfect powers
        int x = primes[i];

        for(ll j = (ll)x * (ll)x ; j <= 100000; j *= x) {
            perf[j] = 1;
        }

    }
    ////////////////////////// EXTRA END  ////////////////////////////
}

// divides the actual string into several integer parts of 9 digits.
// used in efficient numerical string modulo x
void divide(int len) {
    // len= actual numerical string size
    parts.clear();
    ll total = 0;
    int po = 0;
    for(int i = len - 1; i >= 0; i--) {
        ll add = s[i] - '0'; // s is the actual string
        add *= power[po];
        total += add;
        if(po == 9 || i == 0) {

            parts.push_back(total);
            po = 0;
            total = 0;
        }
        else {
            po++;
        }
    }
    reverse(parts.begin(), parts.end());
}

// calculates string modulo x efficiently
ll modulo(ll x) {
    ll mod = 0;
    for(int i = 0; i < parts.size(); i++) {

        mod = mod * power[10];
        mod %= x;
        mod += parts[i];
        mod %= x;

    }
    return mod;
}

// returns S % X where S is a numerical string.
ll stringModx(ll x) {
    divide(strlen(s));
    return modulo(x);
}

// returns inverse(a)%m
ll inv(ll a, ll m) {
    ll m0 = m, t, q;
    ll x0 = 0, x1 = 1;
    if(m == 1) {
        return 0;
    }
    while(a > 1) {
        q = a / m;
        t = m;
        m = a % m, a = t;
        t = x0;
        x0 = x1 - q * x0;
        x1 = t;
    }
    if(x1 < 0) {
        x1 += m0;
    }
    return x1;
}

// Sum of NOD
int SNOD(int n) {
    int res = 0;
    int u = sqrt(n);
    for(int i = 1; i <= u; i++) {
        res += (n / i) - i;
    }
    res *= 2;
    res += u;
    return res;
}

int SOD(int n) {
    int res = 1;
    int sqrtn = sqrt(n);
    for(int i = 0; i < prime.size() && prime[i] <= sqrtn; i++) {
        if(n % prime[i] == 0) {

            // Contains value of (p^0+p^1+...p^a)
            int tempSum = 1;
            int p = 1;
            while(n % prime[i] == 0) {
                n /= prime[i];
                p *= prime[i];
                tempSum += p;
            }
            sqrtn = sqrt(n);
            res *= tempSum;
        }
    }
    if(n != 1) {
        res *= (n + 1);   // Need to multiply (p^0+p^1)
    }
    return res;
}
```

```
// given some numbers num[] and reminders rem[] find
// smallest x such that x%num[i]=rem[i] for all i from 0 to k
ll ChineseRemainder(int k) {
    ll prod = 1;
    for(int i = 0; i < k; i++) {
        prod = prod * 1LL * num[i];
    }
    ll result = 0;
    for(int i = 0; i < k; i++) {
        ll pp = prod / num[i];
        result += rem[i] * inv(pp, num[i]) * pp;
    }
    return result % prod;
}

// Legendre’s formula.. returns largest power of P that divides n!
ll largestPower(ll n, ll p) {
    ll x = 0;
    // Calculate x = n/p + n/(p^2) + n/(p^3) + ....
    while(n) {
        n /= p;
        x += n;
    }
    return x;
}

ll trailing_Zeroes_of_N_Factorial_in_base_b(ll n, ll b) {
    ll ans = (ll)1e18;
    for(int i = 2; i <= sqrt(b); i++) {
        ll cnt = 0;
        while(b % i == 0) {
            b /= i;
            cnt++;
        }
        if(cnt) {
            ans = min(ans, largestPower(n, i) / cnt);
        }
    }
    if(b != 1) {
        ans = min(ans, largestPower(n, b));
    }
    return ans;
}

// returns nTh catalan number
ll catalan_number(ll n) {
    ll c = nCr(2 * n, n);
    return c / (n + 1);
    /* Application of catalan numbers
    1. Number of possible Binary Search Trees with n keys.
    2. Number of expressions containing n pairs of parentheses which are
correctly matched. For n = 3, possible expressions are ((())), ()(()), ()()(),
(())(), (()()).
    3. Number of ways a convex polygon of n+2 sides can split into triangles
by connecting vertices.
    4. Number of full binary trees (A rooted binary tree is full if every
vertex has either two children or no children) with n+1 leaves.
    5. Number of different Unlabeled Binary Trees can be there with n nodes.
    6. The number of paths with 2n steps on a rectangular grid from bottom
left, i.e., (n-1, 0) to top right (0, n-1) that do not cross above the main
diagonal.
    7. Number of ways to insert n pairs of parentheses in a word of n+1
letters, e.g., for n=2 there are 2 ways: ((ab)c) or (a(bc)). For n=3 there are
5 ways, ((ab)(cd)), (((ab)c)d), ((a(bc))d), (a((bc)d)), (a(b(cd))).
    8. Number of Dyck words of length 2n. A Dyck word is a string consisting
of n X’s and n Y’s such that no initial segment of the string has more Y’s
than X’s.  For example, the following are the Dyck words of length 6: XXXYYY
XYXXYY     XYXYXY     XXYYXY     XXYXYY.
    9. Number of ways to form a “mountain ranges” with n upstrokes and n
down-strokes that all stay above the original line.The mountain range
interpretation is that the mountains will never go below the horizon.
    10. Number of stack-sortable permutations of {1, …, n}. A permutation w
is called stack-sortable if S(w) = (1, …, n), where S(w) is defined
recursively as follows: write w = unv where n is the largest element in w and
u and v are shorter sequences, and set S(w) = S(u)S(v)n, with S being the
identity for one-element sequences.
    11. Number of permutations of {1, …, n} that avoid the pattern 123 (or
any of the other patterns of length 3); that is, the number of permutations
with no three-term increasing subsequence. For n = 3, these permutations are
132, 213, 231, 312 and 321. For n = 4, they are 1432, 2143, 2413, 2431, 3142,
3214, 3241, 3412, 3421, 4132, 4213, 4231, 4312 and 4321
    */
}

ull mul(ull a, ull b) {
    ull res = 0;
    while(b) {
        if(b & 1LL) {
            res = (res + a);
        }
        if(res >= n) {
            return 0;
        }
        a = (a << 1LL);
        b >>= 1LL;
    }
    return res;
}

int p, primes[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 51,
53, 59, 61, 67, 71};
void backtrack(int i, int lim, ull val, ull r) {
    if(r > res) {
        res = r;
    }
    if(i == p) {
        return;
    }
    int d;
    ull x = val;
    for(d = 1; d <= lim; d++) {
        x = mul(x, primes[i]);
        if(x == 0) {
            return;
        }
        backtrack(i + 1, d, x, r * (d + 1));
    }
}

ull maximum_NOD_of_any_number_less_than_N(ull n) {
    p = sizeof(primes) / sizeof(int);
    res = 0;
    backtrack(0, 100, 1, 1);
    return res;
}
```

Part 6 ⇛ Graph

6.1: Kruskal

```
const int Max = 15e3 + 10;

struct Node {
    int u, v, w;
} g[Max];

bool less(Node a, Node b) {
    return a.w < b.w;
}

bool more(Node a, Node b) {
    return a.w > b.w;
}

int node, edge, parent[Max];
int Find_parent(int n) {
    //cout << ": " << n << " " << parent[n] << endl;
    if(parent[n] == n) {
        return n;
    }

    return Find_parent(parent[n]);
}

void graph() {
    for(int i = 1; ; i++) {
        cin >> g[i].u >> g[i].v >> g[i].w;

        edge++;
    }
}

int kruskal() {
    int sum = 0;

    for(int i = 0; i <= node; i++) {
        parent[i] = i;
    }

    for(int i = 1; i <= edge; i++) {
        int u = Find_parent(g[i].u), v = Find_parent(g[i].v);
        //cout << u << " " << v << endl;
        if(u != v) {
            //printf("The cost from %d to %d is : %d\n", s1[i], s2[i], w[i]);
            parent[u] = v;
            sum += g[i].w;
        }
    }

    return sum;
}

int main() {
    int t; cin >> t;

    for(int tc = 1; tc <= t; tc++) {
        cin >> node;
        edge = 0;
        graph();

        // Minimus
        sort(g + 1, g + edge + 1, less);
        cout << kruskal() << endl;

        // Maximum
        sort(g + 1, g + edge + 1, more);
        cout << kruskal() << endl;
    }

    return 0;
}
```

6.2: Dijkstra

```
struct Node {
    int at, cost;
    Node(int _at, int _cost) {
        at = _at;
        cost = _cost;
    }
};

bool operator<(Node a, Node b) {
    return a.cost > b.cost;
}

struct Edge {
    int v, w;
    Edge(int _v, int _w) {
        v = _v;
        w = _w;
    }
};

vector <Edge> G[10001];
priority_queue <Node> pq;
int dist[10001];

int n, m, s;

void dijsktra(int src) {
    for(int i = 1; i <= n; i++) {
        dist[i] = 1e9;
    }
    dist[src] = 0;
    pq.push(Node(src, 0));
    while(!pq.empty()) {
        Node u = pq.top();
        pq.pop();

        if(u.cost != dist[u.at]) {
            continue;
        }
        for(int i = 0; i < G[u.at].size(); i++) {
            Edge e = G[u.at][i];
            if(dist[e.v] > u.cost + e.w) {
                dist[e.v] = u.cost + e.w;
                pq.push(Node(e.v, dist[e.v]));
            }
        }
    }
}
```

6.3: Dijkstra with Set

```
#include<bits/stdc++.h>
#define MX 100005
#define INF 2000100100105
#define LL long long
#define pb push_back
#define mp make_pair
using namespace std;
vector<pair<LL, LL> > v[MX];
vector<pair<LL, LL> > ::iterator vt;
int edge, node, path[MX];
LL dist[MX];
```

```
void graph() {
    LL a, b, w;
    for(int i = 1; i <= node; i++) {
        dist[i] = INF;
    }
    for(int i = 1; i <= edge; i++) {
        cin >> a >> b >> w;
        v[a].pb(mp(b, w));
        v[b].pb(mp(a, w));
    }
}

void dijkstra(int src) {
    set<pair<LL, LL> > s;
    set<pair<LL, LL> > ::iterator it;
    dist[src] = 0;
    s.insert(mp(0, src));
    while(s.size()) {
        it = s.begin();
        int u = it->second;
        s.erase(it);
        for(vt = v[u].begin(); vt != v[u].end(); vt++) {
            if(dist[u] + vt->second < dist[vt->first]) {
                if(dist[vt->first] != INF) {
                    s.erase(s.find(make_pair(dist[vt->first], vt->first)));
                }
                dist[vt->first] = dist[u] + vt->second;
                s.insert(mp(dist[vt->first], vt->first));
                path[vt->first] = u;
            }
        }

    }
}

void print_path(int src) {
    if(src == 0) {
        return;
    }
    print_path(path[src]);
    cout << src << " ";
}

int main() {
    ios_base::sync_with_stdio(false);
    cin >> node >> edge;
    graph();
    dijkstra(1);
    //cout << dist[node] << endl;
    if(dist[node] != INF) {
        print_path(node);
        cout << endl;
    }
    else {
        cout << -1 << endl;
    }
    return 0;
}
```

6.4: Dinic Maxflow

```
///V^2*E Complexity
///number of augment path * (V+E)
///Base doesn't matter

const int INF = 2000000000;
const int MAXN = 100;///total nodes
const int MAXM = 10000;///total edges

int N, edges;
int last[MAXN], Prev[MAXM], head[MAXM];
int Cap[MAXM], Flow[MAXM];
int dist[MAXN];
int nextEdge[MAXN];///used for keeping track of next edge of ith node

queue<int> Q;

void init(int N) {
    edges = 0;
    memset(last, -1, sizeof(int)*N);
}

//cap=capacity of edges , flow = initial flow
inline void addEdge(int u, int v, int cap, int flow) {
    head[edges] = v;
    Prev[edges] = last[u];
    Cap[edges] = cap;
    Flow[edges] = flow;
    last[u] = edges++;

    head[edges] = u;
    Prev[edges] = last[v];
    Cap[edges] = 0;
    Flow[edges] = 0;
    last[v] = edges++;
```

```
}

inline bool dinicBfs(int S, int E, int N) {
    int from = S, to, cap, flow;
    memset(dist, 0, sizeof(int)*N);
    dist[from] = 1;

    while(!Q.empty()) {
        Q.pop();
    }

    Q.push(from);

    while(!Q.empty()) {
        from = Q.front(); Q.pop();

        for(int e = last[from]; e >= 0; e = Prev[e]) {
            to = head[e];
            cap = Cap[e];
            flow = Flow[e];

            if(!dist[to] && cap > flow) {
                dist[to] = dist[from] + 1;
                Q.push(to);
            }
        }
    }

    return (dist[E] != 0);
}

inline int dfs(int from, int minEdge, int E) {
    if(!minEdge) {
        return 0;
    }

    if(from == E) {
        return minEdge;
    }

    int to, e, cap, flow, ret;

    for(; nextEdge[from] >= 0; nextEdge[from] = Prev[e]) {
        e = nextEdge[from];
        to = head[e];
        cap = Cap[e];
        flow = Flow[e];
        if(dist[to] != dist[from] + 1) {
            continue;
        }

        ret = dfs(to, min(minEdge, cap - flow), E);

        if(ret) {
            Flow[e] += ret;
            Flow[e ^ 1] -= ret;
            return ret;
        }
    }

    return 0;
}

int dinicUpdate(int S, int E) {
    int flow = 0;

    while(int minEdge = dfs(S, INF, E)) {
        if(minEdge == 0) {
            break;
        }

        flow += minEdge;
    }

    return flow;
}

int maxFlow(int S, int E, int N) {
    int totFlow = 0;

    while(dinicBfs(S, E, N)) {
        for(int i = 0; i <= N; i++) {
            nextEdge[i] = last[i];    /// update last edge of ith node
        }

        totFlow += dinicUpdate(S, E);
    }

    return totFlow;
}
```

6.5: HopcroftKarp

```
//Esqrt(V) Complexity
//0 Based
//Edge from set a to set b
const int MAXN1 = 50010; //nodes in set a
const int MAXN2 = 50010; //nodes in set b
const int MAXM = 150010; //number of edges

int n1, n2, edges, last[MAXN1], prev[MAXM], head[MAXM];
int matching[MAXN2], dist[MAXN1], Q[MAXN1];
bool used[MAXN1], vis[MAXN1]; //vis is cleared in each dfs

// n1 = number of nodes in set a, n2 = number of nodes in set b
void init(int _n1, int _n2) {
    n1 = _n1;
    n2 = _n2;
    edges = 0;
    fill(last, last + n1, -1);
}

void addEdge(int u, int v) {
    head[edges] = v;
    prev[edges] = last[u];
    last[u] = edges++;
}

void bfs() {
    fill(dist, dist + n1, -1);
    int sizeQ = 0;

    for(int u = 0; u < n1; ++u) {
        if(!used[u]) {
            Q[sizeQ++] = u;
            dist[u] = 0;
        }
    }

    for(int i = 0; i < sizeQ; i++) {
        int u1 = Q[i];

        for(int e = last[u1]; e >= 0; e = prev[e]) {
            int u2 = matching[head[e]];

            if(u2 >= 0 && dist[u2] < 0) {
                dist[u2] = dist[u1] + 1;
                Q[sizeQ++] = u2;
            }
        }
    }
}

bool dfs(int u1) {
    vis[u1] = true;

    for(int e = last[u1]; e >= 0; e = prev[e]) {
        int v = head[e];
        int u2 = matching[v];

        if(u2 < 0 || (!vis[u2] && dist[u2] == dist[u1] + 1 && dfs(u2))) {
            matching[v] = u1;
            used[u1] = true;
            return true;
        }
    }

    return false;
}

int augmentPath() {
    bfs();
    fill(vis, vis + n1, false);
    int f = 0;

    for(int u = 0; u < n1; ++u)
        if(!used[u] && dfs(u)) {
            ++f;
        }

    return f;
}
```

6.6: Articulation Bridge

```
void ArticulationBridge(int u) {
      low[u] = dist[u] = ++cur;
      vist[u] = 1;
      for(int i = 0; i < G[u].size(); i++) {
            int v = G[u][i];
            if(vist[v]) {
                  if(v != par[u]) {
                        low[u] = min(low[u], dist[v]);
                  }
            }
            else {
                  par[v] = u;
                  ArticulationBridge(v);
                  low[u] = min(low[u], low[v]);
                  if(low[v] > dist[u]) {
                        ans += 1;
                  }
            }
      }
}
```

6.7: Articulation Point

```
void ArticulationPoint(int u) {
    low[u] = dist[u] = ++cur;
    vist[u] = 1;
    int tot = 0;
    for(int i = 0; i < G[u].size(); i++) {
        int v = G[u][i];
        tot++;
        if(vist[v]) {
            if(v != par[u]) {
                low[u] = min(low[u], dist[v]);
            }
        }
        else {
            par[v] = u;
            ArticulationPoint(v);
            low[u] = min(low[u], low[v]);
            if(tot > 1 and par[u] == -1) {
                st.insert(u);
            }
            if(par[u] != -1 and low[v] >= dist[u]) {
                st.insert(u);
            }
        }
    }
}
```

6.8: Bellman Ford

```
struct edge {
    int u, v, w;
    edge(int _u, int _v, int _w) {
        u = _u;
        v = _v;
        w = _w;
    }
};

const int MAX = 1e5 + 7, INF = 1e7 + 7;
vector < edge > adj;
long long dist[MAX];
int par[MAX];
int V, E;

void bellman_ford(int src) {
    for(int i = 1; i <= V; i++) {
        dist[i] = INF;
    }
    dist[src] = 0;
    par[src] = -1;
    for(int i = 1; i < V; i++) {
        int flag = 0;
        for(auto j : adj) {
            if(dist[j.v] > dist[j.u] + j.w) {
                dist[j.v] = dist[j.u] + j.w;
                par[j.v] = j.u;
                flag = 1;
            }
        }
        if(!flag) {
            break;
        }
    }
}

void print_path(int src, int node) {
    vector < int > path;
    int i = node;
    while(i != -1) {
        path.push_back(i);
        i = par[i];
    }
    for(i = path.size() - 1; i >= 0; i--) {
        cout << path[i] << " ";
    }
    cout << endl;
}

bool negetive_cycle(int src) {
    bellman_ford(src);
    for(auto i : adj) {
        if(dist[i.v] > dist[i.u] + i.w) {
            return true;
        }
    }
    return false;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int uu, vv, ww;
    cin >> V >> E;
    for(int i = 0; i < E; i++) {
        cin >> uu >> vv >> ww;
        adj.push_back(edge(uu, vv, ww));
    }
    if(negetive_cycle(1)) {
        cout << "Negetive Cycle is found!!\n";
    }
    else if(dist[V] < INF) {
        print_path(1, V);
    }
    else {
        cout << -1 << endl;
    }
}
```

6.9: Strongly Connected Component

```
const int MAX = 1e3 + 10;
vector <int> graph[MAX], reverseGraph[MAX], components[MAX];

bool vis[MAX];
int scc[MAX], compCount;
stack<int>nodes;

void DFS(int src) {
    vis[src] = 1;
    for(auto i : graph[src]) {
        if(!vis[i]) {
            DFS(i);
        }
    }
    nodes.push(src);
}

void DFS2(int src) {
    vis[src] = 1;
    for(auto i : reverseGraph[src]) {
        if(!vis[i]) {
            DFS2(i);
        }
    }
    components[compCount].push_back(src);
    scc[src] = compCount;
}

void init() {
    compCount = 1;
    for(int i = 1; i < MAX; i++) {
        graph[i].clear(), reverseGraph[i].clear(), components[i].clear();
    }
}

void addEdge(int u, int v) {
    graph[u].push_back(v);
    reverseGraph[v].push_back(u);
}

void kosaraju_SCC(int n) {
    memset(vis, 0, sizeof vis);
    for(int i = 1; i <= n; i++) {
        if(!vis[i]) {
            DFS(i);
        }
    }
    memset(vis, 0, sizeof vis);
    while(nodes.size()) {
        int top = nodes.top();
        nodes.pop();
        if(!vis[top]) {
            DFS2(top);
            compCount++;
        }
    }
}

void print_SCCs() {
    for(int i = 1; i < compCount; i++) {
        cout << "Component " << i << ":\n";
        for(auto j : components[i]) {
            cout << j << " -> ";
        }
        cout << endl;
    }
}

int main() {
    int n, m, u, v;
    init();
    cin >> n >> m;
    for(int i = 0; i < m; i++) {
        cin >> u >> v;
        addEdge(u, v);
    }
    kosaraju_SCC(n);
    print_SCCs();
}
```

6.10: Ford Fulkerson for Max Flow

```
const int MAXN = 1050;
const int INF = (int) 1e9;
struct edge {
    int from, to, f, cap;
};
int n, m;
vector <edge> e;
vector <int> g[MAXN];
bool used[MAXN];
int s, t;
long long ans;

void init() {
    for(int i=0;i<MAXN;i++)
        g[i].clear();
    e.clear(); memset(used,0,sizeof(used)); ans = 0;
}

void addEdge(int from, int to, int cap) {
    edge ed;

    ed.from = from; ed.to = to; ed.f = 0; ed.cap = cap;
    e.push_back(ed);
    g[from].push_back((int) e.size() - 1);
    ed.from = to; ed.to = from; ed.f = cap; ed.cap = cap;
    e.push_back(ed);
    g[to].push_back((int) e.size() - 1);
}

long long pushFlow(int v, long long flow = INF) {
    used[v] = true;
    if (v == t)
        return flow;
    for (int i = 0; i < (int) g[v].size(); i++) {
        int ind = g[v][i];
        int to = e[ind].to;
        int f = e[ind].f;
        int cap = e[ind].cap;
        if (used[to] || cap - f == 0)
            continue;
        long long pushed = pushFlow(to, min(flow, (long long)cap - f));
        if (pushed > 0) {
            e[ind].f += pushed;
            e[ind ^ 1].f -= pushed;
            return pushed;
        }
    }
    return 0;
}

long long maxFlow() {
    long long ans = 0;
    while(true) {
        memset(used, 0, sizeof(used));
        long long add = pushFlow(s);
        if(add == 0) {
            break;
        }
        ans += add;
    }
    return ans;
}
```

6.11: Directed MST (Edmond's Algorithm)

```
/* Algo
      1 . For each node find the minimum incoming edge except root
      2 . Look for cycle if there is no cycle its MST
      3 . If cycle then consider one cycle as a one single vertex , give
every vertex a new indx  and again recalculate the every edge
      4 . goto step 2
*/
// Directed MST (Edmond's Algo) Template Credit https://github.com/shakilaust
// Tested On LightOJ 1380

#include<bits/stdc++.h>
using namespace std;
#define INF 1e9
const int Maxn = 1005 ; // Highest Vertex

struct edge {
    int u, v, w ;
    edge() {}
    edge(int _u, int _v, int _w) {
        u = _u, v = _v, w = _w ;
    }
};

int nodes, edges, root ;   // nV = number of vertex , nE = number of edge ,
root is root
int vis[Maxn], parnt[Maxn] ;  // vis[] will store from which cycle it belogs ,
pre[] store its parnt
int Idx[Maxn] ; // will store new indxing Id
int dis[Maxn] ; // store the lowest incoming edge of a root
vector < edge > vec ;

int DMST() {
    int ans  = 0, i, u,  v, w  ;
    while(true) {
        int i  ;
        for(i = 0; i < nodes; i++) {
            dis[i] = INF ;
            vis[i] = -1 ;
            Idx[i] = -1 ;
        }
        for(i = 0; i < vec.size(); i++) {
            u = vec[i].u ;
            v = vec[i].v ;
            w = vec[i].w ;
            if(u != v && dis[v] > w) { // lowest Incoming Edge
                parnt[v] = u ;
                dis [v] = w ;
            }
        }
        parnt[root] = root ;
        dis[root] = 0 ;
        for(i = 0; i < nodes; i++) {
            if(dis[i] == INF) {
                return -1; // its not possible to reach
            }
            ans += dis[i];
        }
        int idx = 0 ;
        // cycle detection
        for(i = 0; i < nodes; i++) {
            if(vis[i] == -1) { // not yet visited
                int cur = i ;
                while(vis[cur] == -1) {
                    vis[cur] = i ;
                    cur = parnt[cur] ;
                }
                if(cur == root || vis[cur] != i) {
                    continue ;    // not cycle
                }
                Idx[cur] = idx ; // new indexing
                for(u = parnt[cur] ; cur != u ; u = parnt[u]) {
                    Idx[u] = idx ;
                }
                idx++;
            }
        }
        if(idx == 0) {
            break ;    // no cycle
        }
        for(i = 0; i < nodes; i++) {
            if(Idx[i] == -1) { // yet not find any grp
                Idx[i] = idx++;
            }
        }
        for(i = 0; i < vec.size(); i++) {
            vec[i].w -= dis[vec[i].v];
            vec[i].u = Idx[vec[i].u] ;
            vec[i].v = Idx[vec[i].v];
        }
        nodes = idx++;
        root = Idx[root];
    }
    return ans;
}
```

6.12: Prim’s Algorithm for finding MST

```
const int Max = 1e5 + 10;

bool vist[Max];
vector <pll> G[Max];

ll prim(int src) {
    priority_queue <pll, vector<pll>, greater<pll>> q;
    ll mn = 0;
    q.push(make_pair(0, src));
    while(!q.empty()) {
        pll p = q.top();
        q.pop();
        int u = p.second;
        if(vist[u] == true) {
            continue;
        }
        mn += p.first;
        vist[u] = true;
        for(pll v : G[u]) {
            if(vist[v.second] == false) {
                q.push(v);
            }
        }
    }
    return mn;
}

int main() {
    int n, m, u, v;
    ll w, mn;
    cin >> n >> m;
    for(int i = 1; i <= m; i++) {
        cin >> u >> v >> w;
        G[u].push_back(make_pair(w, v));
        G[v].push_back(make_pair(w, u));
    }
    mn = prim(1);
    cout << mn << endl;
    return 0;
}
```

6.13: Centroid Decomposition

```
/*
Tested Problem:

  Each node has a color, white or black. All the nodes are black initially.

  We will ask you to perfrom some instructions of the following form:

    0 i : change the color of i-th node(from black to white, or from white to
black).
    1 v : ask for the minimum dist(u, v), node u must be white(u can be equal
to v). Obviously, as long as node v is white, the result will always be 0.
*/

const int Max = 2e5 + 10;

int L[Max];
int P[Max][22];
int T[Max];
int ans[Max];
vector <int> G[Max];

void dfs(int u, int par, int dep) {
    T[u] = par;
    L[u] = dep;
    for(int v : G[u]) {
        if(v == par) {
            continue;
        }
        dfs(v, u, dep + 1);
    }
}

int lca_query(int p, int q) {

}

void lca_init(int n) {

}

struct CentriodDecomposition {
    bool vist[Max];
    int sub[Max];
    int path[Max];

    CentriodDecomposition() {
        memset(vist, 0, sizeof vist);
        memset(sub, 0, sizeof sub);
        memset(path, 0, sizeof path);
    }

    int centroid(int u, int par, int sz) {
        for(int v : G[u]) {
            if(v == par) {
                continue;
            }
            if(!vist[v]) {
                if(sub[v] > sz) {
                    return centroid(v, u, sz);
                }
            }
        }
        return u;
    }

    void calc(int u, int par) {
        sub[u] = 1;
        for(int v : G[u]) {
            if(v == par) {
                continue;
            }
            if(!vist[v]) {
                calc(v, u);
                sub[u] += sub[v];
            }
        }
    }

    void decompose(int u, int par) {
        calc(u, par);
        int c = centroid(u, -1, sub[u] / 2);
        vist[c] = 1;
        path[c] = par;
        for(int v : G[c]) {
            if(!vist[v]) {
                decompose(v, c);
            }
        }
    }
} tree;

int dist(int u, int v) {
    int lca = lca_query(u, v);
    return L[u] + L[v] - 2 * L[lca];
```

```
}

int cnt[Max];
multiset <int> st[Max];

struct UpdateQuery {
    UpdateQuery() {
        for(int i = 0; i < Max; i++) {
            ans[i] = 1e9;
        }
    }

    int query(int u) {
        int ret = 1e9;
        for(int v = u; v + 1 > 0; v = tree.path[v]) {
            if(st[v].size()) {
                ret = min(ret, *st[v].begin() + dist(u, v));
            }
        }
        if(ret == 1e9) {
            ret = -1;
        }
        return ret;
    }

    void update(int u) {
        for(int v = u; v + 1 > 0; v = tree.path[v]) {
            if(cnt[u] & 1) {
                st[v].insert(dist(u, v));
            }
            else {
                st[v].erase(st[v].find(dist(u, v)));
            }
        }
    }
} ds;

int main() {
    int n, q, ty, u, v;
    scanf("%d", &n);
    for(int i = 1; i < n; i++) {
        scanf("%d %d", &u, &v);
        G[u].push_back(v);
        G[v].push_back(u);
    }
    dfs(1, -1, 0);
    lca_init(n);
    tree.decompose(1, -1);
    scanf("%d", &q);
    while(q--) {
        scanf("%d %d", &ty, &u);
        if(ty == 0) {
            cnt[u]++;
            ds.update(u);
        }
        else {
            printf("%d\n", ds.query(u));
        }
    }
    return 0;
}
```

6.14: Stable Marriage Problem

```
const int Max = 1e3 + 10;

int n, pref[2 * Max][Max];

bool freeMan[Max];
int wPartner[Max];

bool wPreferM_M1(int w, int m, int m1) {
    for(int i = 1; i <= n; i++) {
        if(pref[w][i] == m) {
            return true;
        }
        if(pref[w][i] == m1) {
            return false;
        }
    }
}

void stableMarriage() {
    memset(freeMan, 0, sizeof freeMan);
    memset(wPartner, -1, sizeof wPartner);
    int rem = n;
    while(rem) {
        int m;
        for(int i = 1; i <= n; i++) {
            if(freeMan[i] == 0) {
                m = i;
                break;
            }
        }
        for(int i = 1; i <= n && freeMan[m] == 0; i++) {
            int w = pref[m][i];
            if(wPartner[w - n] == -1) {
                freeMan[m] = 1;
                wPartner[w - n] = m;
                rem--;
            }
            else {
                int m1 = wPartner[w - n];
                if(wPreferM_M1(w, m, m1)) {
                    freeMan[m1] = 0;
                    freeMan[m] = 1;
                    wPartner[w - n] = m;
                }
            }
        }
    }
    for(int i = 1; i <= n; i++) {
        printf(" (%d %d)", wPartner[i], i + n);
    }
    printf("\n");
}
```

6.15: Floyd Warshall

```
const int Max = 205;
const int inf = 1e9 + 10;

int n, m;
int adj[Max][Max], path[Max][Max];
int dist[Max][Max], minimax[Max][Max], maximin[Max][Max], prob[Max][Max];
bool hasPath[Max][Max];

/// Calculates shortest path for all pairs in dist[][] variable
void floydWarsh() {
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) {
            dist[i][j] = adj[i][j];
        }
    }

    for(int via = 1; via <= n; via++) {
        for(int from = 1; from <= n; from++) {
            for(int to = 1; to <= n; to++) {
                if(dist[from][via] + dist[via][to] < dist[from][to]) {
                    dist[from][to] = dist[from][via] + dist[via][to];
                    path[from][to] = path[via][to];
                }
            }
        }
    }
}

/// finding a path between two nodes that minimizes the maximum edge cost in
the path
void mini_max() {
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) {
            minimax[i][j] = adj[i][j];
        }
    }

    for(int via = 1; via <= n; via++) {
        for(int from = 1; from <= n; from++) {
            for(int to = 1; to <= n; to++) {
                minimax[from][to] = min(minimax[from][to],
max(minimax[from][via], minimax[via][to]));
            }
        }
    }
}

/// finding a path between two nodes that maximizes the minimum edge cost in
the path
void maxi_min() {
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) {
            maximin[i][j] = (adj[i][j] == inf) ? 0 : adj[i][j];
        }
    }

    for(int via = 1; via <= n; via++) {
        for(int from = 1; from <= n; from++) {
            for(int to = 1; to <= n; to++) {
                maximin[from][to] = max(maximin[from][to],
min(maximin[from][via], maximin[via][to]));
            }
        }
    }
}

/// Calculates if a path exits between two nodes for all pairs in directed
graph
void transitiveClosure() {
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) {
            hasPath[i][j] = (adj[i][j] == inf) ? 0 : adj[i][j];
        }
    }

    for(int via = 1; via <= n; via++) {
        for(int from = 1; from <= n; from++) {
            for(int to = 1; to <= n; to++) {
                hasPath[from][to] |= (hasPath[from][via] & hasPath[via][to]);
            }
        }
    }
}

/// Safest path variant of Floyd-Warshall example
/// input: p is an probability matrix (probability of survival) for n nodes
///     e.g. p[i][j] is the probability of survival moving directly from i to
j.
///     the probability from a node to itself e.g. p[i][i] should be
initialized to 1
/// output: p[i][j] will contain the probability of survival using the safest
path from i to j.
void safest_path() {
    for(int via = 1; via <= n; via++) {
        for(int from = 1; from <= n; from++) {
            for(int to = 1; to <= n; to++) {
                prob[from][to] = max(prob[from][to], prob[from][via] *
prob[via][to]);
            }
        }
    }
}

/// Check if the graph contains negative cycle in connected graph
/// Distance from any node to itself should always be 0
/// For disconnected graph, check whether the distance is 0 for all node
bool negativeCycle() {
    floydWarsh();
    return dist[0][0] < 0;
}

/// Finds the path from i to j
void findPath(int i, int j) {
    cout << i << endl;
    if(i != j) {
        findPath(path[i][j], j);
    }
}
```

```
/// initializes adjacency matrix and path
void init() {
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) {
            adj[i][j] = inf;
        }
    }
    for(int i = 1; i <= n; i++) {
        adj[i][i] = 0, path[i][i] = i;
    }
}
```

6.16: Johnson's Algorithm

```
// Problem : JHNSN - Johnsons Algorithm
// Link : https://w...content-available-to-author-only...j.com/problems/JHNSN/

const int Max = 1e3 + 10;
struct Info {
    int v, w;
    Info() {}
    Info(int _v, int _w) {
        v = _v ;
        w = _w;
    }
};
struct Edge {
    int u, v, w;
    Edge() {}
    Edge(int _u, int _v, int _w) {
        u = _u;
        v = _v;
        w = _w;
    }
};

vector<Info>adj[Max];
vector<Edge>edges;
int n, m;
int h[Max], dist[Max][Max];

void init() {
    for(int i = 0; i < Max; i++) {
        adj[i].clear();
    }
}
```

```
bool bellman_ford(int src) {
    /// runs n times instead of n-1 for source node
    /// returns 1 if negative cycle is found
    /// Stores distance from source in h[]
}
void dijkstra(int src) {
}
void modifyGraph(int sign) {
    for(int i = 1; i <= n; i++) {
        for(auto &j : adj[i]) {
            j.w += sign * (h[i] - h[j.v]);
        }
    }
}
bool johnson_algorithm() {
    /// Adding source node to calculate h[]
    int src = 0;
    edges.clear();
    for(int i = 1; i <= n; i++) {
        for(auto j : adj[i]) {
            edges.push_back(Edge(i, j.v, j.w));
        }
        adj[src].push_back(Info(i, 0));
        edges.push_back(Edge(src, i, 0));
    }
    /// Modifying edges to avoid negative weight edges
    if(bellman_ford(src)) {
        return 0;
    }
    modifyGraph(1);
    /// Running Dijkstra for each node
    for(int i = 1; i <= n; i++) {
        dijkstra(i);
    }
    modifyGraph(-1);
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) {
            if(dist[i][j] < 1e8) {
                dist[i][j] -= (h[i] - h[j]);
            }
        }
    }
    adj[src].clear();
    return 1;
}
```

Part 7 ⇛ Dynamic Programming

7.1: LCS

```
const int Max = 1000 + 5;

int dp[Max][Max];
bool vist[Max][Max];

string s, t;
int n, m;

int lcs(int i, int j) {
    if(i >= n or j >= m) {
        return 0;
    }
    int &ret = dp[i][j];
    bool &vis = vist[i][j];
    if(vis) {
        return ret;
    }
    vis = 1;
    int res = 0;
    if(s[i] == t[j]) {
        res = 1 + lcs(i + 1, j + 1);
    }
    else {
        res = max(lcs(i + 1, j), lcs(i, j + 1));
    }
    return ret = res;
}

string ans;

void solution(int i, int j) {
    if(i >= n or j >= m) {
        return;
    }
    if(s[i] == t[j]) {
        ans += s[i];
        solution(i + 1, j + 1);
    }
    else {
        if(lcs(i + 1, j) > lcs(i, j + 1)) {
            solution(i + 1, j);
        }
        else {
            solution(i, j + 1);
        }
    }
}

int main() {
    int T;
    cin >> T;
    for(int tc = 1; tc <= T; tc++) {
        cin >> s >> t;
        n = s.size();
        m = t.size();
        memset(vist, 0, sizeof vist);
        int maxlen = lcs(0, 0);
        ans = "";
        solution(0, 0);
        cout << ans << '\n';
    }
}
```

7.2: Longest Increasing Subsequence (nlogn)

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

int main() {
    int n, t;
    vector<int> v;
    vector<int> LIS;
    scanf("%d", &n);
    for(int i = 1; i <= n; i++) {
        scanf("%d", &t);
        v.push_back(t);
        vector<int> ::iterator it;
        it = lower_bound(LIS.begin(), LIS.end(), t);
        if(it != LIS.end()) {
            *it = t;
        }
        else {
            LIS.push_back(t);
        }
    }
    cout << LIS.size() << endl;
    return 0;
}
```

7.3: Coin Change

*In a strange shop there are **n** types of coins of value **A_1, A_2 ... A_n**. You have to find the number of ways you can make **K** using the coins. You can use any coin at most **K** times.*

*For example, suppose there are three coins 1, 2, 5. Then if **K = 5** the possible ways are:*

11111
1112
122
5

So, 5 can be made in 4 ways.

Solution:

```
ll ar[Max];

int main() {
    ll t, n, k;

    while(scanf("%lld", &t) == 1) {
        int tc = 1;
        while(t--) {
            scanf("%lld %lld", &n, &k);
            ll dp[k + 1];
            memset(dp, 0, sizeof dp);
            dp[0] = 1;

            for(int i = 1; i <= n; i++) {
                scanf("%lld", &ar[i]);
            }

            for(int i = 1; i <= n; i++) {
                for(int j = 1; j <= k; j++) {
                    if(ar[i] <= j) {
                        dp[j] = dp[j] % Mod + dp[j - ar[i]] % Mod;
                        dp[j] %= Mod;
                    }
                }
            }

            printf("Case %d: %lld\n", tc, dp[k]);
            tc++;
        }
    }

    return 0;
}
```

7.4: SOS DP

```
/*
Problem: Given a fixed array A of 2^N integers, we need to calculate ∀ x
function F(x) = Sum of all A[i] such that x&i = i, i.e., i is a subset of x.
*/

for(int i = 0; i < (1 << N); ++i) {
    F[i] = A[i];
}

for(int i = 0; i < N; ++i) {
    for(int mask = 0; mask < (1 << N); ++mask) {
        if(mask & (1 << i)) {
            F[mask] += F[mask ^ (1 << i)];
        }
    }
}
```

Part 8 ⇛ Matrix

8.1: Matrix Exponentiation

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

typedef long long int ll;
typedef pair <ll, ll> pll;

const int Max = 2e6 + 10;
const int Mod = 1e4 + 7;
const ll Inf = 1LL << 62;

struct Matrix {
    int n, m;
    vector <vector <int>> mat;

    Matrix() {}

    Matrix(int _n, int _m) {
        n = _n, m = _m;
        mat = vector <vector <int>> (n, vector <int> (m));
    }

    void print() {
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < m; j++) {
                printf("%d", mat[i][j]);
                if(j == m - 1) {
                    printf("\n");
                }
                else {
                    printf(" ");
                }
            }
        }
    }
};

Matrix Multiply(Matrix a, Matrix b, int Mod) {
    Matrix c = Matrix(a.n, b.m);
    for(int i = 0; i < a.n; i++) {
        for(int j = 0; j < b.m; j++) {
            c.mat[i][j] = 0;
            for(int k = 0; k < a.m; k++) {
                c.mat[i][j] += (1LL * a.mat[i][k] * b.mat[k][j]) % Mod;
                c.mat[i][j] %= Mod;
            }
        }
    }
    return c;
}

Matrix Add(Matrix a, Matrix b, int Mod) {
    Matrix c = Matrix(a.n, a.n);
    for(int i = 0; i < a.n; i++) {
        for(int j = 0; j < a.n; j++) {
            c.mat[i][j] = a.mat[i][j] + b.mat[i][j];
            c.mat[i][j] %= Mod;
        }
    }
    return c;
}

Matrix Pow(Matrix a, ll p, int Mod) {
    if(p == 1) {
        return a;
    }
    Matrix x = Pow(a, p / 2, Mod);
    x = Multiply(x, x, Mod);
    if(p & 1) {
        x = Multiply(x, a, Mod);
    }
    return x;
}

int main() {
#ifdef Mr_Emrul
    freopen("inputf.in", "r", stdin);
#endif /// Mr_Emrul

    int t, n, a, b, c;
    scanf("%d", &t);
    for(int tc = 1; tc <= t; tc++) {
        scanf("%d %d %d %d", &n, &a, &b, &c);
        if(n <= 2) {
            printf("Case %d: 0\n", tc);
            continue;
        }
        Matrix M = Matrix(4, 4);
        M.mat[0][0] = a;
        M.mat[0][2] = b;
        M.mat[0][3] = c;
        M.mat[1][0] = 1;
        M.mat[2][1] = 1;
        M.mat[3][3] = 1;
        Matrix A = Matrix(4, 1);
        A.mat[3][0] = 1;
        M = Pow(M, n - 2, Mod);
        A = Multiply(M, A, Mod);
        //A.print();
        printf("Case %d: %d\n", tc, A.mat[0][0]);
    }
    return 0;
}
```

8.2: Matrix Power Sum

```
Matrix solveBinary(const Matrix &a, int n) {
    if(n == 1) {
        return a;
    }

    Matrix answer = new_matrix;
    Matrix tmp = new_matrix;
    answer = solveBinary(a, n / 2);
    if(n & 1) {
        tmp = mod_pow(a, n / 2 + 1);
        answer = Add(answer, multiply(tmp, answer));
        answer = Add(tmp, answer);
    }
    else {
        tmp = mod_pow(a, n / 2);
        answer = Add(answer, multiply(tmp, answer));
    }
    return answer;
}

/*
      We know that the matrix A^x can be solved quickly by the matrix fast
      power. We must consider how to reduce the computation is S = A + A ^ 2
      + A ^ 3 + ... A ^ k + during the addition of this time-consuming
      operation. Below we consider matrix A as a normal number A.
      For S(10) we have:
      S(10) = A + A^2 + A^3 +A^4 + A^5 + A^5 * (A +  A^2 +A^3 +A^4 + A^5 )
      For S(5) we have:
      S(5) =  A + A^2 + A^3 + A^3 * ( A + A^2 )
      For S(2) we have:
      S(2) = A + A * (A)
      The value of A has been given in the question, then we can get the
      value of S(2) by recursive backtracking, and find the value of A^3,
      then we can get the value of S(5) by the above formula, and then get S
```

```
        The value of (10). Then for each S(k) we can do this two-way recursive
        solution. Note that S(n) here requires an additional value of A^(n/2+1)
        when n is odd (for example, S(5) above). Finally, add the matrix's fast
        power and addition operations.
*/
```