Tesla, IIIT Hyderabad

December 11, 2017

Contents

1 dp

1.1 1D-1D

```
// http://www.spoj.com/submit/ACQUIRE/id=17751232

#include<bits/stdc++.h>
#define int long long int
using namespace std;
const int MAXN = 1e5 + 5;
int N, dp[MAXN];
pair<int,int> A[MAXN];
vector<pair<int,int> >v, w, arr;

int calculate(int pos, int choose){
       return dp[choose - 1] + (A[pos].second * A[choose].first);
}
```

```
void insert(int pos, int val){

	while(true){

		if(arr.size() == 1){
			int _pos = arr[0].first;
			int _val = arr[0].second;
			_pos = pos - 1;
			arr.pop_back();
			if(_pos) arr.push_back({_pos, _val});
			arr.push_back({MAXN, val});
			break;
		}

		int sz = arr.size();
		int _pos = arr[sz - 1].first;
		int _val = arr[sz - 1].second;

		int __pos = arr[sz - 2].first;
		int __val = arr[sz - 2].second;

		arr.pop_back();
		arr.pop_back();

		if(pos > __pos && pos <= _pos){
			arr.push_back({__pos, __val});
			_pos = pos - 1;
			if(_pos) arr.push_back({_pos, _val});
			arr.push_back({MAXN, val});
			break;
		}

		else arr.push_back({__pos, __val});
	}

	assert(arr[arr.size() - 1].first == MAXN);
}

signed main(){

	ios::sync_with_stdio(false);

	cin >> N;

	for(int i=1; i<=N; ++i){
		int x, y;
		cin >> x >> y;
		v.push_back({x, y});
	}

	sort(v.rbegin(), v.rend());

	w.push_back({v[0].first, v[0].second});
	for(int i=1; i<v.size(); ++i){
		if(v[i].second <= w[w.size() - 1].second) continue;
		w.push_back({v[i].first, v[i].second});
	}

	for(int i=1; i<=w.size(); ++i) A[i].first = w[i-1].first,
	    A[i].second = w[i-1].second;
	N = w.size();

	dp[1] = A[1].first * A[1].second;
	arr.push_back({MAXN, 1});

	// (x, y) min arr means use y till xth index

	for(int i=2; i<=N; ++i){

		int pos = lower_bound(arr.begin(), arr.end(), make_pair(i,
		    0ll)) - arr.begin();
		pos = arr[pos].second;
		dp[i] = dp[pos - 1] + (A[pos].first * A[i].second);
		dp[i] = min(dp[i], dp[i-1] + (A[i].first * A[i].second));

		if(i == N) break;

		// Find max such that we can use i till that index
		int low = i+1, high = N + 1;
		while(low < high){
			int mid = low + high;
			mid >>= 1;

			int pos = lower_bound(arr.begin(), arr.end(),
			    make_pair(mid, 0ll)) - arr.begin();
			pos = arr[pos].second;

			if(calculate(mid, pos) > calculate(mid, i)){
				high = mid;
			}
```

```
                else{
                        low = mid + 1;
                }

        }
        insert(low, i);
    }

    cout << dp[N];

}
```

1.2 Convex-Hull-Trick

```
// https://www.codechef.com/viewplaintext/13174851

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

const int MAX = 1e6 + 6;
int Q;
long long a[MAX], b[MAX];

struct cht{
    struct Line{
        int a;
        long long b;
        long long val;
        double xLeft;
        bool type;
        Line(long long _a = 0 , long long _b = 0){
            a = _a;
            b = _b;
            xLeft = -1e16;
            type = 0;
            val = 0;
        }
        long long valueAt(int x) const{
            return 1LL * a * x + b;
        }
        friend bool areParallel(const Line &l1, const Line &l2){
            return l1.a == l2.a;
        }
        friend double intersectX(const Line &l1 , const Line &l2){
            return areParallel(l1 , l2) ? 1e16 : 1.0 * (l2.b -
                l1.b) / (l1.a - l2.a);
        }
        bool operator < (const Line &l2) const{
            if(!l2.type)
                return a < l2.a;
            return xLeft > l2.val;
        }
};
set < Line > hull;
bool hasPrev(set < Line > :: iterator it){
    return it != hull.begin();
}
bool hasNext(set < Line > :: iterator it){
    return it != hull.end() && next(it) != hull.end();
}
bool irrelevant(const Line &l1 , const Line &l2 , const Line &l3){
    return intersectX(l1,l3) <= intersectX(l1,l2);
}
bool irrelevant(set < Line > :: iterator it){
    return hasPrev(it) && hasNext(it) && (irrelevant(*next(it)
        , *it , *prev(it)));
}
set < Line > :: iterator updateLeftBorder(set < Line > :: iterator
    it){
    if(!hasNext(it)){
        return it;
    }
    double val = intersectX(*it , *next(it));
    Line buf(*it);
    it = hull.erase(it);
    buf.xLeft = val;
    it = hull.insert(it, buf);
    return it;
}
void addLine(int a , long long b){
    Line l3 = Line(a, b);
    auto it = hull.lower_bound(l3);
    if(it != hull.end() && areParallel(*it , l3)){
        if(it -> b > b){
            it = hull.erase(it);
        }
        else{
            return;
```

```
                }
        }
        it = hull.insert(it, l3);
        if(irrelevant(it)){
                hull.erase(it);
                return;
        }
        while(hasPrev(it) && irrelevant(prev(it))){
                hull.erase(prev(it));
        }
        while(hasNext(it) && irrelevant(next(it))){
                hull.erase(next(it));
        }
        it = updateLeftBorder(it);
        if(hasPrev(it)){
                updateLeftBorder(prev(it));
        }
        if(hasNext(it)){
                updateLeftBorder(next(it));
        }
    }
    long long getBest(int x){
        Line q;
        q.val = x;
        q.type = 1;
        auto bestLine = hull.lower_bound(q);
        if(bestLine == hull.end()){
                return 1e16;
        }
        return bestLine -> valueAt(x);
    }
} hull;
signed main(){
    ios::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);

    cin >> Q >> Q;
    while(Q--){
        int t; cin >> t;
        if(t == 1){
                int x, y, z; cin >> x >> y >> z;
                a[y] =  a[y] + x * (b[y] - z);
                b[y] = z;
                hull.addLine(-b[y], -a[y]);
        }
        else{
                int t; cin >> t;
                cout << -hull.getBest(t) << endl;
        }
    }
}
```

1.3 Divide-and-Conquer

```
//
    https://www.hackerrank.com/contests/ioi-2014-practice-contest-2/challenges/g

#include<bits/stdc++.h>
#define int unsigned long long int
using namespace std;

const int MAXN = 8e3 + 3;
const int MAXK = 810;
int A[MAXN], sum[MAXN], dp[MAXK][MAXN];
int N, K;

int get_sum(int x, int y){
   return sum[y] - sum[x-1];
}

void get(int lev, int start, int en, int left, int right){

    if(start > en) return;

    int mid = (start + en) >> 1;

    dp[lev][mid] = dp[lev - 1][max(left, lev - 1)] +
        (get_sum(max(left, lev - 1) + 1,mid) * (mid - max(left, lev -
        1)));
    int op = max(left, lev - 1);

    for(int i=max(left, lev - 1) + 1; i <= min(right, mid - 1); ++i){
        if(dp[lev-1][i] + (get_sum(i+1,mid) * (mid - i)) <
            dp[lev][mid]){
            dp[lev][mid] = dp[lev-1][i] + get_sum(i+1,mid) *
                (mid - i);
            op = i;
        }
```

```
        }

        get(lev, start, mid - 1, left, op);
        get(lev, mid + 1, en, op, right);

}

signed main(){
        cin >> N >> K;

    K = min(K, N);

        for(int i=1; i<=N; ++i) cin >> A[i];

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

        for(int i=1; i<=N; ++i) dp[1][i] = (i * get_sum(1,i));

        for(int i=2; i<=K; ++i){
                get(i, i, N, i-1, N-1);
        }

        cout << dp[K][N] << endl;

}
```

1.4 IOI-Aliens-trick

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

const int MAX = 2e3 + 3;

double X[MAX], Y[MAX], dp[MAX][MAX], mid;
int p[MAX][MAX];
int N, A, B;

double solve(){
        for(int i=1; i<=N; ++i){
                for(int j=0; j<=A; ++j){
                        double &d = dp[i][j];
                        int &pick = p[i][j];
                        d = dp[i - 1][j], pick = 0;

                        if(j && d < dp[i - 1][j - 1] + X[i]){
                                d = dp[i - 1][j - 1] + X[i];
                                pick = 1;
                        }

                        if(d < dp[i - 1][j] + Y[i] - mid){
                                d = dp[i - 1][j] + Y[i] - mid;
                                pick = 2;
                        }

                        if(j && d < dp[i - 1][j - 1] + X[i] + Y[i] - (X[i]
                            * Y[i]) - mid){
                                d = dp[i - 1][j - 1] + X[i] + Y[i] - (X[i] *
                                    Y[i]) - mid;
                                pick = 3;
                        }
                }
        }
}

int main() {
        ios_base::sync_with_stdio(false);

        cin >> N >> A >> B;

        for(int i=1; i<=N; ++i) cin >> X[i];
        for(int i=1; i<=N; ++i) cin >> Y[i];

        double low = 0, high = 1;

        for(int i=0; i<50; ++i){
                mid = (low + high) / 2;
                solve();
                int pos = A, c = 0;
                for(int i=N; i>0; --i){
                        if(p[i][pos] == 1) --pos;
                        else if(p[i][pos] == 2) ++c;
                        else if(p[i][pos] == 3) --pos, ++c;
                }
                if(c > B) low = mid;
                else high = mid;
        }
```

```
        mid = high;
        solve();

        printf("%0.5f\n", dp[N][A] + high * B);

        return 0;
}
```

1.5 SOS

```
// http://codeforces.com/contest/165/submission/30179495

#include <bits/stdc++.h>

using namespace std;

const int LOG = 23;
const int MAX = (1 << 23);
const int MAXN = 1e6 + 6;

int N, X[MAX], A[MAXN], dp[MAX];

signed main(){
        ios::sync_with_stdio(false);
        cin.tie(0); cout.tie(0);

        cin >> N;
        for(int i=1; i<=N; ++i){
                int x; cin >> x;
                X[x] = i;
                A[i] = x;
        }

        for(int i=0; i<LOG + 1; ++i){
                for(int j=0; j<MAX; ++j){
                        dp[j] = X[j];
                        if(j & (1LL << i)){
                                if(dp[j] == 0) dp[j] = X[j ^ (1 << i)];
                        }
                }
                for(int j=0; j<MAX; ++j) X[j] = dp[j];
        }
        for(int i=1; i<=N; ++i){
                int need = ((MAX - 1) ^ A[i]);
                int ans = X[need];
                cout << (ans ? A[ans] : -1) << ' ';
        }
}

/*

S(mask, i) = S(mask, i - 1)
                        S(mask, i - 1) union S(mask ^ (1 << i), i - 1) ->
                            ith bit of mask is ON


All subsets of mask.

*/
```

2 ds

2.1 Auxillary-Tree

```
#include <bits/stdc++.h>
#define int long long int
using namespace std;
const int MAX = 1e5 + 5;
const int MOD = 1e9 + 7;
vector<pair<int, int> > v[MAX], aux;
vector<int> tree[MAX], arr;
int start[MAX], en[MAX], sub[MAX], pos[MAX], par[MAX], is[MAX], lev[MAX];
int sum[4 * MAX];
int N, K, Q, timer, root, head[MAX], p[MAX], SIZE[MAX], SUB[MAX];

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

vector<int> group[MAX];

void clear(){
        for(int i=1; i<=N; ++i){
                v[i].clear();
                group[i].clear();
```

```
                start[i] = en[i] = sub[i] = pos[i] = par[i] = is[i] =
                    lev[i] = 0;
                head[i] = p[i] = SIZE[i] = SUB[i] = 0;
        }
        memset(sum, 0, sizeof sum);
        timer = 0, root = 0;
}

int dfs(int cur, int p, int level = 1){
        lev[cur] = level;
        start[cur] = ++timer;
        for(auto it:v[cur]){
                if(it.first == p) continue;
                dfs(it.first, cur, level + 1);
                par[it.first] = cur;
                sub[cur] += sub[it.first];
        }
        en[cur] = ++timer;
        ++sub[cur];
}
void update(int l, int r, int node, int pos, int val){
        if(l > pos || r < pos) return;
        if(l == r){
                sum[node] = val;
                return;
        }
        int mid = (l + r) >> 1;
        update(l, mid, node << 1, pos, val);
        update(mid + 1, r, 1 | node << 1, pos, val);
        sum[node] = (sum[node << 1] + sum[1 | node << 1]) % MOD;
}
void hld(int cur, int p, int h = 1){
        head[cur] = h;
        pos[cur] = ++timer;
        int sc = 0;
        for(auto it:v[cur]){
                if(it.first == p) continue;
                if(sub[it.first] > sub[sc]) sc = it.first;
        }
        if(!sc) return;
        hld(sc, cur, h);
        for(auto it:v[cur]){
                if(it.first == p) continue;
                if(it.first == sc){
                        update(1, N, 1, pos[it.first], it.second);
                        continue;
                }
                hld(it.first, cur, it.first);
                update(1, N, 1, pos[it.first], it.second);
        }
}
int query(int l, int r, int node, int ql, int qr){
        if(l > qr || r < ql) return 0;
        if(l >= ql && r <= qr) return sum[node];
        int mid = (l + r) >> 1;
        return (query(l, mid, node << 1, ql, qr) + query(mid + 1, r, 1 |
            node << 1, ql, qr)) % MOD;
}
int LCA(int x, int y){
        while(true){
                if(head[x] == head[y]){
                        if(lev[x] < lev[y]) return x;
                        return y;
                }
                if(lev[head[x]] < lev[head[y]]) y = par[head[y]];
                else x = par[head[x]];
        }
}
void get_vertices(){
        sort(aux.begin(), aux.end());
        int len = aux.size();
        for(int i=1; i<len; ++i){
                int x = LCA(aux[i].second, aux[i - 1].second);
                aux.push_back({start[x], x});
        }
        for(auto it:aux) is[it.second] = 0;
        vector<pair<int, int> >temp; temp.clear();
        for(auto it:aux){
                if(is[it.second]) continue;
                temp.push_back(it);
                is[it.second] = 1;
        }
        aux.clear();
        for(auto it:temp) aux.push_back(it), is[it.second] = 0;
        sort(aux.begin(), aux.end());
}
int get_distance(int x, int y){
        int ans = 0;
        while(true){
                if(x == y) break;
```

```
            if(head[x] == head[y]){
                ans = ans + ((query(1, N, 1, min(pos[x], pos[y]),
                    max(pos[x], pos[y])) - query(1, N, 1,
                    min(pos[x], pos[y]), min(pos[x], pos[y])) %
                    MOD) + MOD;
                ans %= MOD;
                break;
            }
            if(lev[head[x]] < lev[head[y]]) swap(x, y);
            ans = (ans + query(1, N, 1, pos[head[x]], pos[x])) % MOD;
            x = par[head[x]];
        }
        return ans;
}
void first_dfs(int cur, int parent){
        SUB[cur] = SIZE[cur];
        for(auto it:tree[cur]){
            if(it != parent){
                first_dfs(it, cur);
                SUB[cur] = (SUB[cur] + SUB[it]) % MOD;
                p[it] = cur;
            }
        }
}
int solve(int total){
        edges.clear();
        get_vertices();
        root = aux[0].second;
        for(auto it:aux) tree[it.second].clear();
        stack<int> s; s.push(aux[0].second);
        for(int i=1; i<aux.size(); ++i){
            int cur = aux[i].second;
            while(true){
                if(en[s.top()] < start[cur]) s.pop();
                else break;
            }
            tree[s.top()].push_back(cur);
            edges.push_back({{s.top(), cur}, get_distance(s.top(),
                cur)});
            s.push(cur);
        }
        int ans = 0;
        first_dfs(root, 0);
        for(auto it:edges){
            int x = it.first.first, y = it.first.second, z = it.second;
            if(p[x] == y) swap(x, y);
            int X = total - SUB[y], Y = SUB[y];
            ans = (ans + (((X * 1LL * Y) % MOD) * 1LL * z) % MOD) %
                MOD;
        }
        ans = (ans + ans) % MOD;
        return ans;
}
signed main(){
        ios::sync_with_stdio(false);
        cin.tie(0); cout.tie(0);
        int t; cin >> t;
        while(t--){
            cin >> N >> K;
            for(int i=1; i<=N; ++i){
                int x; cin >> x;
                group[x].push_back(i);
            }
            for(int i=1; i<N; ++i){
                int x, y, z;
                cin >> x >> y >> z;
                v[x].push_back({y, z});
                v[y].push_back({x, z});
            }
            timer = 0;
            dfs(1, 0);
            timer = 0;
            hld(1, 0);
            for(int i=1; i<=K; ++i){
                aux.clear();
                arr.clear();
                // group i has set of vertices for ith query
                for(auto it:group[i]){
                    SUB[it] = 0;
                    SIZE[it] = 1;
                    aux.push_back({start[it], it});
                    arr.push_back(it);
                }
                cout << solve(group[i].size()) << endl;
                for(auto it:group[i]) SIZE[it] = 0;
            }
            clear();
        }
}
```

2.2 Treap-Lazy

```
// https://www.codechef.com/viewplaintext/9981566

#include<bits/stdc++.h>
using namespace std;
const int LOG = 30;
struct node{
	int val , p , arr[LOG] , size , rev , lazy;
	node* l;
	node* r;
	node(int x = 0){
		val = x;
		size = 1;
		rev = lazy = 0;
		p = rand();
		l = r = NULL;
		for(int i=0; i<LOG; ++i){
			arr[i] = (val>>i) & 1;
		}
	}
}; node* start;
int get_size(node* cur){
	if(cur) return cur->size;
	return 0;
}
void print(node* cur){
	if(!cur) return;
	cout << cur->val << endl;
	print(cur->l);
	print(cur->r);
}
int get_lazy(node* cur){
	if(!cur) return 0;
	return cur->lazy;
}
void push(node* cur){
	if(!cur) return;
	if(cur->rev){
		if(cur->l) cur->l->rev ^= 1;
		if(cur->r) cur->r->rev ^= 1;
		swap(cur->l, cur->r);
		cur->rev = 0;
	}
	if(get_lazy(cur)){
		int temp = get_lazy(cur);
		if(cur->l) cur->l->lazy ^= temp;
		if(cur->r) cur->r->lazy ^= temp;
		cur->val ^= temp;
		for(int i=0; i<LOG; ++i){
			if(temp & 1LL<<i){
				cur->arr[i] = get_size(cur) - cur->arr[i];
			}
		}
		cur->lazy = 0;

	}
}
int get(node* cur, int pos){
	if(!cur) return 0;
	return cur->arr[pos];
}
void update(node* cur){
	if(!cur) return;
	cur->size = get_size(cur->l) + 1 + get_size(cur->r);
	push(cur->l);
	push(cur->r);
	for(int i=0; i<LOG; ++i){
		cur->arr[i] = get(cur->l, i) + get(cur->r, i) + ((cur->val
		    >> i) & 1);
	}
}
void merge(node* &cur, node* x, node* y){
	push(x);
	push(y);
	if(!x) cur = y;
	else if(!y) cur = x;
	else if(x->p > y->p){
		merge(x->r, x->r, y);
		cur = x;
	}
	else{
		merge(y->l, x, y->l);
		cur = y;
	}
	update(cur);
}
void split(node* cur, node* &L, node* &R, int key, int add = 1){
	if(!cur){
		L = R = NULL; return;
```

```
        }
        push(cur);
        int pos = add + get_size(cur->l);
        if(pos<= key){
                split(cur->r, cur->r, R, key, pos + 1);
                L = cur;
        }
        else{
                split(cur->l, L, cur->l, key, add);
                R = cur;
        }
        push(cur);
        update(cur);
}
void query(int x, int y){
        node* A = NULL;
        node* B = NULL;
        node* C = NULL;
        split(start, A, B, x-1);
        split(B, B, C, y - x + 1);
        int ans = 0;
        assert(B);
        for(int i=0; i<LOG; ++i) ans += B->arr[i];
        printf("%d\n", ans);
        merge(B, B, C);
        merge(start, A, B);
}
void reverse(int x, int y){
        node* A = NULL;
        node* B = NULL;
        node* C = NULL;
        split(start, A, B, x-1);
        split(B, B, C, y - x + 1);
        B->rev ^= 1;
        merge(B, B, C);
        merge(start, A, B);
}
void lazy(int x, int y, int z){
        node* A = NULL;
        node* B = NULL;
        node* C = NULL;
        split(start, A, B, x-1);
        split(B, B, C, y - x + 1);
        assert(B);
        B->lazy ^= z;
        merge(B, B, C);
        merge(start, A, B);
}
int main(){
        int N;
        scanf("%d", &N);
        for(int i=1; i<=N; ++i){
                int x;
                scanf("%d", &x);
                node* temp = new node(x);
                if(i == 1) start = temp;
                else merge(start, start, temp);
        }
        int Q;
        cin >> Q;
        for(int i=1; i<=Q; ++i){
                int t, x , y;
                scanf("%d%d%d", &t, &x, &y);
                if(t == 1) query(x,y);
                else if(t == 2) reverse(x,y);
                else{
                        int z;
                        scanf("%d", &z);
                        lazy(x,y,z);
                }
        }
}
```

2.3 TreapSubsetSumDP

```
// https://www.codechef.com/viewplaintext/13723886

#include <bits/stdc++.h>
using namespace std;
struct node{
        node* l;
        node* r;
        int frq[15], size, rev, p, val;
        node(int v){
                l = r = NULL;
                size = 1;
                p = rand();
                rev = 0;
```

```
                for(int i=0; i<15; ++i) frq[i] = 0;
                frq[v] = 1;
                val = v;
        }
}; node* start;

const int MAX = 1e5 + 5;
int N, Q, cnt = 0, orr[15], c[MAX];
map<int, int> MAP;
vector<int> v;
bitset<MAX> dp;

int get_size(node* cur){
        if(!cur) return 0;
        return cur->size;
}
int get(node* cur, int idx){
        if(!cur) return 0;
        return cur->frq[idx];
}
void push(node* cur){
        if(!cur) return;
        if(cur->rev){
                if(cur->l) cur->l->rev ^= 1;
                if(cur->r) cur->r->rev ^= 1;
                swap(cur->l, cur->r);
                cur->rev = 0;
        }
}
void update(node* cur){
        if(!cur) return;
        cur->size = get_size(cur->l) + 1 + get_size(cur->r);
        push(cur->l);
        push(cur->r);
        for(int i=0; i<15; ++i){
                cur->frq[i] = get(cur->l, i) + get(cur->r, i);
        }
        ++cur->frq[cur->val];
}
void merge(node* &cur, node* x, node* y){
        push(x); push(y);
        if(!x) cur = y;
        else if(!y) cur = x;
        else if(x->p > y->p){
                merge(x->r, x->r, y);
                cur = x;
        }
        else{
                merge(y->l, x, y->l);
                cur = y;
        }
        update(cur);
}
void split(node* cur, node* &L, node* &R, int key, int add = 1){
        if(!cur){
                L = R = NULL;
                return;
        }
        push(cur);
        int pos = add + get_size(cur->l);
        if(pos<= key){
                split(cur->r, cur->r, R, key, pos + 1);
                L = cur;
        }
        else{
                split(cur->l, L, cur->l, key, add);
                R = cur;
        }
        push(cur);
        update(cur);
}

int solve(int W){
        v.clear();
        dp.reset();
        for(int i=1; i<=W; ++i){
                if(c[i] == 0) continue;
                if(c[i] & 1){
                        --c[i];
                        v.push_back(i);
                }
                else{
                        --c[i]; --c[i];
                        v.push_back(i); v.push_back(i);
                }
                int temp = (i << 1);
                int inc = (c[i] >> 1);
                if(temp <= W) c[temp] += inc;
        }
        dp.set(0);
```

```
	for(auto it:v) dp |= (dp << it);
	return dp.test(W);
}

int main(){
	ios::sync_with_stdio(false);
	cin.tie(0); cout.tie(0);
	cin >> N >> Q;
	for(int i=1; i<=N; ++i){
		int w, ww; cin >> w; ww = w;
		if(MAP.find(w) == MAP.end()) MAP[w] = ++cnt;
		w = MAP[w];
		orr[w] = ww;
		node* cur = new node(w);
		if(i == 1) start = cur;
		else merge(start, start, cur);
	}
	for(int i=1; i<=Q; ++i){
		int ty; cin >> ty;
		if(ty == 1){
			int pos, w, ww; cin >> pos >> w; ww = w;
			if(MAP.find(w) == MAP.end()) MAP[w] = ++cnt;
			w = MAP[w];
			orr[w] = ww;

			node* A = NULL;
			node* B = NULL;
			node* C = NULL;

			split(start, A, B, pos - 1);
			split(B, B, C, 1);

			assert(get_size(B) == 1);
			B->frq[B->val] = 0;

			B->val = w;
			B->frq[B->val] = 1;

			merge(start, A, B);
			merge(start, start, C);
		}
		else if(ty == 2){
			int l, r; cin >> l >> r;
			node* A = NULL;
			node* B = NULL;
			node* C = NULL;
			split(start, A, B, l - 1);
			split(B, B, C, r - l + 1);
			B->rev ^= 1;
			merge(B, B, C);
			merge(start, A, B);
		}
		else{
			int l, r, w;
			cin >> l >> r >> w;
			node* A = NULL;
			node* B = NULL;
			node* C = NULL;

			split(start, A, B, l - 1);
			split(B, B, C, r - l + 1);

			assert(B);

			memset(c, 0, sizeof c);
			for(int i=0; i<15; ++i){
				if(B->frq[i]){
					int val = orr[i];
					if(val) c[val] += B->frq[i];
				}
			}

			bool yes = solve(w);
			if(yes) cout << "Yes\n";
			else cout << "No\n";

			merge(start, A, B);
			merge(start, start, C);
		}
	}
}
```

2.4 Wavelet-Trees

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

const int MAX = 1e5 + 5;
```

```
int N, M, A[MAX], back[MAX];
vector<int> s[MAX * 4], v[MAX * 4], arr;
map<int, int> MAP;

void build(int low, int high, int node){
	int parent = node >> 1;
	for(auto it:s[parent]){
		if(it >= low and it <= high) s[node].push_back(it);
	}

	if(low == high) return;

	int mid = (low + high) >> 1;

	if(s[node].size()){
		if(s[node][0] > mid) v[node].push_back(0);
		else v[node].push_back(1);
		for(int i=1; i<s[node].size(); ++i){
			if(s[node][i] > mid)
			    v[node].push_back(v[node].back());
			else v[node].push_back(v[node].back() + 1);
		}
	}
	else return;

	build(low, mid, node << 1);
	build(mid + 1, high, 1 | node << 1);
}

int query(int low, int high, int node, int l, int r, int rank){

	if(low == high) return low;

	int mid = (low + high) >> 1, new_l = 0, new_r = 0, to_l = 0, to_r
	    = 0;

	to_l = v[node][r];
	if(l) to_l = to_l - v[node][l - 1];

	to_r = r - l + 1 - to_l;

	if(to_l >= rank){
		if(l) new_l = v[node][l - 1];
		new_r = new_l + to_l - 1;
		return query(low, mid, node << 1, new_l, new_r, rank);
	}
	else{
		if(l) new_l = l - v[node][l - 1];
		new_r = new_l + to_r - 1;
		return query(mid + 1, high, 1 | node << 1, new_l, new_r,
		    rank - to_l);
	}
}

void compress(){
	for(int i=1; i<=N; ++i) arr.push_back(A[i]);
	sort(arr.begin(), arr.end());
	int c = 0;
	for(auto it:arr){
		if(MAP.find(it) == MAP.end()) MAP[it] = ++c;
	}
	for(int i=1; i<=N; ++i){
		int v = MAP[A[i]];
		back[v] = A[i];
		A[i] = v;
	}
}

int main() {
	ios_base::sync_with_stdio(false);

	cin >> N >> M;
	for(int i=1; i<=N; ++i) cin >> A[i];

	compress();

	int mid = (1 + MAX) >> 1;

	for(int i=1; i<=N; ++i) s[1].push_back(A[i]);

	if(s[1][0] > mid) v[1].push_back(0);
	else v[1].push_back(1);

	for(int i=1; i<N; ++i){
		if(s[1][i] > mid) v[1].push_back(v[1].back());
		else v[1].push_back(v[1].back() + 1);
	}

	build(1, mid, 2);
```

```
	build(mid + 1, MAX, 3);

	while(M--){
		int L, R, K; cin >> L >> R >> K;
		--L; --R;
		cout << back[query(1, MAX, 1, L, R, K)] << endl;
	}

	return 0;
}
```

3 flow$_n$etworks

3.1 MCMF

```
// Min-Cost Max Flow

// Set value for NODES and INF(also check in spfa function), call
    addEdge(u, v, capacity, cost)
// and for the answer use the loop in the end of main.

#include <bits/stdc++.h>
#define fr(x) scanf("%d", &x)
using namespace std;

const int NODES = 256, INF = 600000000;
int dis[NODES], prev2[NODES], inQ[NODES];

struct edge {
	int to, cap, cost;
};

vector<edge> e;
vector<int> g[NODES];

void addEdge(int u, int v, int cap, int cost) {
	g[u].push_back(e.size());
	e.push_back({v, cap, cost});
	g[v].push_back(e.size());
	e.push_back({u, 0, -cost});
}

int spfa(int start, int sink) {
	queue<int> q;
	memset(dis, 64, sizeof(dis));
	dis[start] = 0;
	prev2[start] = -1;
	inQ[start] = 1;
	q.push(start);
	while(!q.empty()) {
		int u = q.front();
		q.pop();
		inQ[u] = 0;
		for(auto&v:g[u]) {
			if(e[v].cap && dis[e[v].to] > dis[u] + e[v].cost) {
				prev2[e[v].to] = v;
				dis[e[v].to] = dis[u] + e[v].cost;
				if(!inQ[e[v].to]) q.push(e[v].to);
				inQ[e[v].to] = 1;
			}
		}
	}
	if(dis[sink] > INF) {
		return INF;
	}
	int temp = prev2[sink], aug;
	aug = e[temp].cap;
	while(temp!=-1) {
		aug = min(aug, e[temp].cap);
		temp = prev2[e[1^temp].to];
	}
	temp = prev2[sink];
	while(temp!=-1) {
		e[temp].cap -= aug;
		e[1^temp].cap += aug;
		temp = prev2[e[1^temp].to];
	}
	return (dis[sink]*aug);
}

signed main() {
	int n, m;

	// Problem Specific Code
	fr(n);
	fr(m);
	int s = n + 1;
```

```
int t = n + 2;
vector<int> balance(110);
addEdge(n, 1, 1000000000, 0);

int ans = 0;
for(int i=1; i<=m; ++i) {
	int a, b, c, f;
	fr(a);
	fr(b);
	fr(c);
	fr(f);
	balance[a] += f;
	balance[b] -= f;
	if(f <= c) {
		addEdge(a, b, c - f, 1);
		addEdge(a, b, 1000000000, 2);
		addEdge(b, a, f, 1);
	}
	else {
		ans += f - c;
		addEdge(a, b, 1000000000, 2);
		addEdge(b, a, f - c, 0);
		addEdge(b, a, c, 1);
	}
}

int sumB = 0;
for(int i=1; i<=n; ++i) {
	if(balance[i] > 0) {
		addEdge(i, t, balance[i], 0);
		sumB += balance[i];
	}
	else {
		addEdge(s, i, -balance[i], 0);
	}
}

// Max Flow Loop, Currently min cost is returned
while(1) {
	int temp = spfa(s, t);
	if(temp == INF) break;
	ans += temp;
}
printf("%d\n", ans);
return 0;
}
```

3.2 dinics

```
class Dinics {
public:
	typedef int flowType; // can use float/double
	static const flowType INF = 1e9; // maximum capacity
	static const flowType EPS = 0; // minimum capacity/flow change

private:
	int nodes, src, dest;
	vector<int> dist, q, work;
	struct Edge {
	  int to, rev;
	  flowType f, cap;
	};
	vector< vector<Edge> > g;

	bool dinic_bfs() {
	  fill(dist.begin(), dist.end(), -1);
	  dist[src] = 0;
	  int qt = 0;
	  q[qt++] = src;
	  for (int qh = 0; qh < qt; qh++) {
	    int u = q[qh];
	    for (int j = 0; j < (int) g[u].size(); j++) {
	      Edge &e = g[u][j];
	      int v = e.to;
	      if (dist[v] < 0 && e.f < e.cap) {
	        dist[v] = dist[u] + 1;
	        q[qt++] = v;
	      }
	    }
	  }
	  return dist[dest] >= 0;
	}
	int dinic_dfs(int u, int f) {
	  if (u == dest)
	    return f;
	  for (int &i = work[u]; i < (int) g[u].size(); i++) {
	    Edge &e = g[u][i];
	    if (e.cap <= e.f) continue;
```

```
      int v = e.to;
      if (dist[v] == dist[u] + 1) {
        flowType df = dinic_dfs(v, min(f, e.cap - e.f));
        if (df > 0) {
          e.f += df;
          g[v][e.rev].f -= df;
          return df;
        }
      }
    }
    return 0;
  }

public:
  Dinics(int n): dist(n, 0), q(n, 0),
      work(n, 0), g(n), nodes(n) {}

  // s->t (cap); t->s (rcap)
  void addEdge(int s, int t, flowType cap, flowType rcap = 0) {
    g[s].push_back({t, (int) g[t].size(), 0, cap});
    g[t].push_back({s, (int) g[s].size() - 1, 0, rcap});
  }
  flowType maxFlow(int _src, int _dest) {
    src = _src;
    dest = _dest;
    flowType result = 0;
    while (dinic_bfs()) {
      fill(work.begin(), work.end(), 0);
          flowType delta;
      while ((delta = dinic_dfs(src, INF)) > EPS)
        result += delta;
    }
    return result;
  }
};

class Bipartite: Dinics {
  int m, n;
  int s, t;
public:
  Bipartite(int m, int n, int dcap = 1): Dinics(m + n + 2) {
    this->m = m;
    this->n = n;
    s = 0, t = m + n + 1;
    for (int i = 1; i <= m; i++) Dinics::addEdge(s, i, dcap);
    for (int j = m + 1; j <= m + n; j++) Dinics::addEdge(j, t,
        dcap);
  }
  void addEdge(int i, int j, flowType cap = 1) {
    Dinics::addEdge(i + 1, j + m + 1, cap);
  }
  flowType maxMatch() {
    return Dinics::maxFlow(s, t);
  }
};

/* Usage:
  Dinics flow(<number of nodes>);
  flow.addEdge(<from>, <to>, <capacity>[, <reverse capacity>]);
  ...
  result = flow.maxFlow(<source>, <sink>);

  Bipartite b(<lhs>, <rhs>[, <default capacity>]);
  b.addEdge(<from>, <to>[, <capacity>]);
  ...
  result = b.maxMatch();
*/
```

3.3 hopcroft-karp

```
class HopcroftKarp {
public:
  static const int INF = 1e9;

private:
  int U, V, nil;
  vector<int> pairU, pairV, dist;
  vector< vector<int> > adj;

  bool bfs() {
    queue<int> q;
    for (int u = 0; u < U; u++) {
      if (pairU[u] == nil) {
        dist[u] = 0;
        q.push(u);
      } else {
        dist[u] = INF;
      }
```

```
    }
    dist[nil] = INF;
    while (not q.empty()) {
      int u = q.front(); q.pop();
      if (dist[u] >= dist[nil]) continue;
      for (int v: adj[u]) {
        if (dist[pairV[v]] == INF) {
          dist[pairV[v]] = dist[u] + 1;
          q.push(pairV[v]);
        }
      }
    }
    return dist[nil] != INF;
  }
  bool dfs(int u) {
    if (u == nil) return true;
    for (int v: adj[u]) {
      if (dist[pairV[v]] == dist[u] + 1) {
        if (dfs(pairV[v])) {
          pairV[v] = u;
          pairU[u] = v;
          return true;
        }
      }
    }
    dist[u] = INF;
    return false;
  }
public:
  HopcroftKarp(int U_, int V_) {
    U = U_, V = V_;
    nil = U;
    adj.resize(U + 1);
    pairU.resize(U + 1);
    pairV.resize(V);
    dist.resize(U + 1);
  }
  void addEdge(int u, int v) {
    adj[u].push_back(v);
  }
  int maxMatch() {
    fill(pairU.begin(), pairU.end(), nil);
    fill(pairV.begin(), pairV.end(), nil);
    int res = 0;
    while (bfs()) {
      for (int u = 0; u < U; u++) {
        if (pairU[u] == nil && dfs(u)) res++;
      }
    }
    return res;
  }
};
```

4 geometry

4.1 closest pair

```
const Segment InfSegment = {{-INF, -INF}, {INF, INF}};

Cord adis2(Cord a, Cord b) { return (a - b) * (a - b); }

Segment closest_pair_rec(PointList Px, PointList Py) {
  int n = Px.size();
  if (n < 2) return InfSegment;
  if (n == 2) return make_pair(Px[0], Px[1]);

  int m = n / 2;
  PointList Pxl(Px.begin(), Px.begin() + m), Pxr(Px.begin() + m,
      Px.end());
  PointList Pyl, Pyr;
  Pyl.reserve(Pxl.size()); // prevent allocation/resize overhead
  Pyr.reserve(Pxr.size()); // --do--

  Point mid = Px[m];
  for (int i = 0; i < n; i++) {
    (compareX(Py[i], mid) ? Pyl : Pyr).push_back(Py[i]);
  }

  auto resl = closest_pair_rec(Pxl, Pyl);
  auto resr = closest_pair_rec(Pxr, Pyr);
  auto res = (dist2(resl) < dist2(resr)) ? resl : resr;
  Cord mindis = dist2(res);

  Pxl.clear(); Pyl.clear();
  Pxr.clear(); Pyr.clear();

  PointList Pys;
```

```
  Pys.reserve(n);
  for (auto& p: Py) {
    if (adis2(p.x, mid.x) <= mindis) Pys.push_back(p);
  }
  for (int i = 0; i < Pys.size(); i++) {
    for (int j = i + 1; j < Pys.size(); j++) {
      if (adis2(Pys[i].y, Pys[j].y) > mindis) break;
      if (dist2(res) > dist2(Pys[i], Pys[j])) {
        res = {Pys[i], Pys[j]};
      }
    }
  }
  return res;
}

Segment closest_pair(PointList P)
{
  vector<Point> Px(P), Py(P);
  sort(Px.begin(), Px.end(), compareX);
  for (int i = 0; i < Px.size() - 1; i++) {
    if (dist2(Px[i], Px[i + 1]) == 0) return {Px[i], Px[i + 1]};
  }
  sort(Py.begin(), Py.end(), compareY);
  return closest_pair_rec(Px, Py);
}
```

4.2 complex

```
struct Complex {
  Cord x, y;
  Complex(Cord xx, Cord yy, bool isPolar = false) {
    if (isPolar) {
      x = xx * cos(yy);
      y = xx * sin(yy);
    } else {
      x = xx, y = yy;
    }
  }
  Complex(Point a, Point b) {
    x = b.x - a.x;
    y = b.y - a.y;
  }
  Cord mod() { return hypot(x, y); }
  Cord angle() {
    if (isZero(x)) return (y > 0 ? PI / 2 : -PI / 2);
    return atan(y / x);
  }
  void add(const Complex& b) { x += b.x; y += b.y; }
  void mult(const Complex& b) {
    Cord tx = x * b.x - y * b.y;
    Cord ty = x * b.y + y * b.x;
    x = tx, y = ty;
  }
  void rotate(Cord ang) { mult(Complex(1.0, ang, true)); }
  void scale(Cord len) { x *= len; y *= len; }
  void unit() { scale(1.0 / mod()); }
};
typedef Complex vect;

Cord dot(vect a, vect b) { return a.x * b.x + a.y * b.y; }
Cord cross(vect a, vect b) { return a.x * b.y - b.x * a.y; }
Cord angle(Point a, Point o, Point b) {
  auto oa = vect(o, a), ob = vect(o, b);
  oa.unit(); ob.unit();
  return acos(dot(oa, ob));
}
```

4.3 convex-hull

```
vector<Point> convexHull(vector<Point> P) {
  vector<Point> up, dn;
  sort(P.begin(), P.end(), compareX);

  for (auto& p: P) {
    while (up.size() > 1 && !cw(*(up.end() - 2), *(up.end() - 1), p))
      up.pop_back();
    while (dn.size() > 1 && !ccw(*(dn.end() - 2), *(dn.end() - 1), p))
      dn.pop_back();
    up.push_back(p);
    dn.push_back(p);
  }

  for (int i = dn.size() - 2; i > 0; i--)
    up.push_back(dn[i]);
  return up;
```

```
}
```

4.4 geometry

```
typedef long double Cord;
const Cord INF = 1e18, EPS = 1e-9;
const Cord PI = acos(-1);
bool isZero(Cord val) { return (-EPS < val && val < EPS); }
Cord ABS(Cord val) { return val >= 0 ? val : -val; }

struct Point { Cord x, y; };
typedef pair<Point, Point> Segment;
typedef vector<Point> PointList;

bool compareX(const Point& a, const Point& b) {
  if (a.x == b.x) return a.y <= b.y;
  return a.x < b.x;
}
bool compareY(const Point& a, const Point& b) {
  if (a.y == b.y) return a.x <= b.x;
  return a.y < b.y;
}

// Det:- +ve = cw, -ve = ccw, 0 = collinear
Cord Determ(Point A, Point B, Point C) {
  return (A.y - B.y) * (C.x - B.x) - (C.y - B.y) * (A.x - B.x);
}

#define cw(...) (Determ(__VA_ARGS__) > 0)
#define ccw(...) (Determ(__VA_ARGS__) < 0)
#define cwl(...) (Determ(__VA_ARGS__) >= 0)
#define ccwl(...) (Determ(__VA_ARGS__) <= 0)

Cord dist2(Point a, Point b) {
  Cord dx = a.x - b.x;
  Cord dy = a.y - b.y;
  return (dx * dx + dy * dy);
}
Cord dist2(Segment s) {
  return dist2(s.first, s.second);
}
Cord dist(Point a, Point b) {
  return sqrt(dist(a, b));
}
}
Cord dist(Segment s) {
  return dist(s.first, s.second);
}
```

5 graph

5.1 2SAT

```
// 2-SAT

// Set value for NODES, then call add_impl etc. depending on use case.
// Then, call dfs1, dfs2 and in the end, iterate over topo_sort order
    using the loop at the end of main.

#include <bits/stdc++.h>
#define fr(x) scanf("%d", &x)
using namespace std;

const int NODES = 2*100010;

vector<int> revadj[NODES], adj[NODES], top_sort, working;
int marked[NODES];
stack<int> finishing;
int comp[NODES], c = 0;

void add_impl(int v1, int v2) {
       adj[v1].push_back(v2); revadj[v2].push_back(v1);
}
void add_equiv(int v1, int v2) {
       add_impl(v1, v2);add_impl(v2, v1);
}
void add_or(int v1, int v2) {
       add_impl(1^v1, v2);add_impl(1^v2, v1);
}
void add_xor(int v1, int v2) {
       add_or(v1, v2);add_or(1^v1, 1^v2);
}
void add_true(int v1) {
       add_impl(1^v1, v1);
}
void add_and(int v1, int v2) {
```

```
	add_true(v1);add_true(v2);
}

void dfs1(int v) {
	marked[v] = true;
	for (auto i: adj[v]){
		if (marked[i] == 0){
			dfs1(i);
		}
	}
	top_sort.push_back(v);
	finishing.push(v);
}

void dfs2(int v) {
	marked[v] = true;
	for (auto i: revadj[v]){
		if (marked[i] == 0){
			dfs2(i);
		}
	}
	comp[v] = c;
}

vector<int> v[100010];
int main() {
	// Problem Specific Code
	int n, m;
	fr(n);
	fr(m);
	for(int i=1; i<=n; ++i){
		int l;
		fr(l);
		while(l--) {
			int temp;
			fr(temp);
			v[i].push_back(temp);
		}
	}
	for(int i=2; i<=n; ++i) {
		int same = 1;
		for(int j=0; j<min(v[i].size(), v[i-1].size()); ++j) {
			if(v[i][j] != v[i-1][j]) {
				same = 0;
				if(v[i][j] > v[i-1][j]) {
					add_or(((2*v[i-1][j])),
					    (1^(2*v[i][j])));
				}
				else {
					add_true(1^(2*v[i][j]));
					add_true((2*v[i-1][j]));
				}
				break;
			}
		}
		if(same) {
			if(v[i-1].size() > v[i].size()) {
				printf("No\n");
				return 0;
			}
		}
	}
	// From here, the 2 SAT specific code starts
	for(int i=2; i<=(2*m+1); ++i){
		if(!marked[i]){
			dfs1(i);
		}
	}
	memset(marked, 0, sizeof marked);
	while(finishing.size()) {
		if(!marked[finishing.top()]){
			c++;
			dfs2(finishing.top());
		}
		finishing.pop();
	}
	for(int i=2; i<=(2*m+1); ++i) {
		if(comp[i] == comp[i^1]) {
			printf("No\n");
			return 0;
		}
	}
	memset(marked, 0, sizeof marked);
	for(auto &z: top_sort){
		if(marked[z>>1] == 0) {
			marked[z>>1] = 1;
			if((z & 1) == 0) working.push_back(z>>1);
		}
	}
	printf("Yes\n%d\n", working.size());
```

```
	for(auto &z: working) {
		printf("%d ", z);
	}
	return 0;
}
```

5.2 Centroid

```
// http://codeforces.com/gym/100570/problem/F

#include<bits/stdc++.h>
#define int long long int
using namespace std;
const int MAXN = 1e5 + 5;
int size[MAXN], answer[MAXN], h[MAXN], done[MAXN];
int tot = 0;
vector<int> parent, nodes, distanc;
vector<pair<int,int> > qr[MAXN], v[MAXN];
int N, Q;
void dfs_size(int cur, int par){
	size[cur] = 1;
	++tot;
	for(auto it:v[cur]){
		if(it.first != par && done[it.first] == 0){
			dfs_size(it.first, cur);
			size[cur] += size[it.first];
		}
	}
}
int find(int cur, int par){
	for(auto it:v[cur]){
		if(done[it.first] || it.first == par) continue;
		if(size[it.first] > (tot >> 1)) return find(it.first, cur);
	}
	return cur;
}
void dfs_parent(int cur, int par, int dis){
	h[cur] = dis;
	parent.push_back(dis);
	for(auto it:v[cur]){
		if(done[it.first] || it.first == par) continue;
		dfs_parent(it.first, cur, dis + it.second);
	}
}
void dfs_child(int cur, int par, int dis){
	distanc.push_back(dis);
	nodes.push_back(cur);
	for(auto it:v[cur]){
		if(done[it.first] || it.first == par) continue;
		dfs_child(it.first, cur, dis + it.second);
	}
}
void decompose(int cur){
	tot = 0;
	dfs_size(cur, 0);
	cur = find(cur, 0);
	parent.clear();
	dfs_parent(cur, 0, 0);
	sort(parent.begin(), parent.end());
	for(auto q:qr[cur]){
		answer[q.second] += upper_bound(parent.begin(),
		    parent.end(), q.first) - parent.begin();
	}
	done[cur] = 1;
	for(auto it:v[cur]){
		if(done[it.first]) continue;
		nodes.clear();
		distanc.clear();
		dfs_child(it.first, cur, it.second);
		sort(distanc.begin(), distanc.end());
		for(auto node:nodes){
			for(auto q:qr[node]){
				answer[q.second] +=
				    upper_bound(parent.begin(),
				    parent.end(), q.first - h[node]) -
				    parent.begin();
				answer[q.second] -=
				    upper_bound(distanc.begin(),
				    distanc.end(), q.first - h[node]) -
				    distanc.begin();
			}
		}
	}
	for(auto it:v[cur]){
		if(done[it.first]) continue;
		decompose(it.first);
	}
}
```

```
main(){
	cin >> N >> Q;
	for(int i=1; i<N; ++i){
		int x, y, z;
		cin >> x >> y >> z;
		v[x].push_back({y, z});
		v[y].push_back({x, z});
	}
	for(int i=1; i<=Q; ++i){
		int node, val;
		cin >> node >> val;
		qr[node].push_back({val, i});
	}
	decompose(1);
	for(int i=1; i<=Q; ++i) cout << answer[i] << endl;
}
```

5.3 HLD

```
// https://www.codechef.com/viewplaintext/16084415
// Values are associated with edges -> Values are associated with child
    node
// Query from x to y is exclusive of LCA

#include <bits/stdc++.h>
#define int long long int

using namespace std;

const int MAX = 1e5 + 5;

int N, Q, timer = 0;
int lev[MAX], p[MAX], sub[MAX], pos[MAX], head[MAX], s[4 * MAX],
    back[MAX], val[MAX];
vector<pair<int, int> > v[MAX];
vector<pair<pair<int, int>, int> > e;

void clear(){
	timer = 0;
	for(int i=1; i<=N; ++i){
		v[i].clear();
		lev[i] = p[i] = sub[i] = pos[i] = head[i] = back[i] =
		    val[i] = 0;
	}
	memset(s, 0, sizeof s);
	e.clear();
}

int func(int x) {
	int cnt = __builtin_popcount(x);
	int x1 = ((1LL<<cnt) - 1);
	int y1 = (x1<<(50-cnt));
	return (x1 ^ y1);
}


void dfs(int cur, int par){
	lev[cur] = lev[par] + 1;
	p[cur] = par;
	sub[cur] = 1;
	for(auto it:v[cur]){
		if(it.first != par){
			dfs(it.first, cur);
			sub[cur] += sub[it.first];
		}
	}
}

void hld(int cur, int par, int h = 1){
	pos[cur] = ++timer;
	back[timer] = cur;
	head[cur] = h;
	int sc = 0;
	for(auto it:v[cur]){
		if(it.first != par){
			if(sub[it.first] > sub[sc]) sc = it.first;
		}
	}
	if(!sc) return;
	hld(sc, cur, h);
	for(auto it:v[cur]){
		if(it.first == par || it.first == sc) continue;
		hld(it.first, cur, it.first);
	}
}

void build(int l, int r, int node){
	if(l == r){
```

```
                s[node] = func(val[back[l]]);
                return;
        }
        int mid = (l + r) >> 1;
        build(l, mid, node << 1);
        build(mid + 1, r, 1 | node << 1);
        s[node] = max(s[node << 1], s[1 | node << 1]);
}

int query(int l, int r, int node, int ql, int qr){
        if(l > qr || r < ql) return 0;
        if(l >= ql && r <= qr) return s[node];
        int mid = (l + r) >> 1;
        return max(query(l, mid, node << 1, ql, qr), query(mid + 1, r, 1 |
            node << 1, ql, qr));
}

void update(int l, int r, int node, int pos, int val){
        if(l > pos || r < pos) return;
        if(l == r){
                s[node] = func(val);
                return;
        }
        int mid = (l + r) >> 1;
        update(l, mid, node << 1, pos, val);
        update(mid + 1, r, 1 | node << 1, pos, val);
        s[node] = max(s[node << 1], s[1 | node << 1]);
}

int get_distance(int x, int y){
        int ans = 0;
        while(head[x] != head[y]){
                if(lev[head[x]] < lev[head[y]]) swap(x, y);
                int l = pos[x];
                int r = pos[head[x]];
                if(l > r) swap(l, r);
                ans = max(ans, query(1, N, 1, l, r));
                x = p[head[x]];
        }
        if(x == y) return ans;
        if(lev[x] < lev[y]) swap(x, y);
        int l = pos[x];
        int r = pos[y];
        ans = max(ans, query(1, N, 1, min(l, r) + 1, max(l, r)));
        return ans;
}

signed main() {
        ios_base::sync_with_stdio(false);

        int t; cin >> t;
        while(t--){
                cin >> N;
                for(int i=1; i<N; ++i){
                        int x, y, z; cin >> x >> y >> z;
                        e.push_back({{x, y}, z});
                        v[x].push_back({y, z});
                        v[y].push_back({x, z});
                }

                dfs(1, 0);
                hld(1, 0);

                for(auto &it:e){
                        if(p[it.first.first] == it.first.second)
                            swap(it.first.first, it.first.second);
                        val[it.first.second] = it.second;
                }

                build(1, N, 1);

                cin >> Q;
                while(Q--){
                        int ty; cin >> ty;
                        if(ty == 1){
                                int P, Q; cin >> P >> Q;
                                --P;
                                int x = e[P].first.first, y =
                                    e[P].first.second;
                                update(1, N, 1, pos[y], Q);

                        }
                        else{
                                int x, y; cin >> x >> y;
                                cout << get_distance(x, y) << endl;
                        }
                }

                clear();
```

```
    }

    return 0;
}
```

5.4 tarjan

```
namespace Tarjan {
  vector<int> pre, low;
  vector<vector<int>> comps;
  vector<bool> onstack;
  stack<int> st;
  vector<int> *adj;
  int dfsno;

  void push(int u) { st.push(u); onstack[u] = 1; }
  int pop() { int v = st.top(); st.pop(); onstack[v] = 0; return v; }

  void dfs(int u) {
    pre[u] = low[u] = dfsno++;
    push(u);

    for (int v: adj[u]) {
      if (pre[v] == -1) {
        dfs(v);
        low[u] = min(low[u], low[v]);
      } else if (onstack[v]) {
        low[u] = min(low[u], pre[v]);
      }
    }
    if (low[u] == pre[u]) {
      comps.emplace_back();
      auto& curr = comps.back();
      int v;
      do curr.push_back(v = pop()); while (v != u);
    }
  }

  vector<vector<int>> computeSCC(int n, vector<int>* iadj) {
    pre = vector<int>(n, -1);
    low = vector<int>(n, -1);
    onstack = vector<bool>(n, false);
    st = stack<int>();
    comps.clear();

    dfsno = 0;
    adj = iadj;
    for (int u = 0; u < n; u++) if (pre[u] == -1) dfs(u);
    return comps;
  }
};
```

6 math

6.1 extended-euclid

```
// ax + by = gcd(a, b)
// @args: {a, b}
// @returns: {x, y}
template<class T>
pair<T, T> extended_gcd(T a, T b) {
  T x0 = 1, x1 = 0;
  T y0 = 0, y1 = 1;
  T r0 = a, r1 = b;
  while (r1 != 0) {
    T q = r0 / r1, temp;
    temp = r1, r1 = r0 - q * r1, r0 = temp;
    temp = x1, x1 = x0 - q * x1, x0 = temp;
    temp = y1, y1 = y0 - q * y1, y0 = temp;
  }
  // gcd(a, b) = r0
  return {x0, y0};
}
```

6.2 fft

```
// FFT

// Make 2 vector<base> a and b with values: coefficient of x^0, x^1, x^2
    ...
// Resize them to double the size as powers double after multiplication.
// Then call fft(a, 0) and fft(b, 0) to convert them to point form.
// Then multiply using ans[i] = a[i]*b[i].
```

```
// Then call fft(ans, 1) to get back to coefficient form.

#include <bits/stdc++.h>
#define fr(x) scanf("%d",&x)
using namespace std;
#define base complex<double>
const double PI=acos(-1);
vector<base> a, ans;

inline void fft(vector<base> &a, bool invert) {
	int logn=0, n=a.size();
	while((1<<logn)<n) ++logn;
	for(int i=1, j=0; i<n; ++i) {
		int bit = (n>>1);
		for(; j>=bit; bit>>=1)
			j -= bit;
		j += bit;
		if(i < j)
			swap (a[i], a[j]);
	}
	for(int len=2;len<=n;(len<<=1)) {
		double ang = 2*PI/len;
		if(invert) ang = -ang;
		base wlen(cos(ang), sin(ang));
		for(int i=0; i<n; i+=len) {
			base w(1);
			for(int j=0; j<(len/2); ++j) {
				base u = a[i+j], v = w*a[i+j+len/2];
				a[i+j] = u+v;
				a[i+j+len/2] = u-v;
				w *= wlen;
			}
		}
	}
	if(invert) {
		for(int i=0; i<n; ++i) {
			a[i] /= n;
		}
	}
}

// This function exponentiates a polynomial to degree k.
inline void exp(int k) {
	ans.push_back(1);
	for(int i=0; k; ++i) {
		if(k&(1<<i)) {
			ans.resize(max(ans.size(),a.size()));
			a.resize(max(ans.size(),a.size()));
			a.resize(a.size()<<1);
			ans.resize(ans.size()<<1);
			fft(a, 0);
			fft(ans, 0);
			for(int i=0; i<ans.size(); ++i) {
				ans[i]*=a[i];
			}
			fft(ans, 1);
			for(int i=0; i<ans.size(); ++i) {
				if(real(ans[i])>0.5) ans[i]=1;
				else ans[i]=0;
			}
			fft(a, 1);
			k=(k^(1<<i));
		}
		else {
			a.resize(a.size()<<1);
		}
		fft(a, 0);
		for(int i=0;i<a.size();++i){
			a[i] = a[i]*a[i];
		}
		fft(a, 1);
		for(int i=0;i<a.size();++i){
			if(real(a[i])>0.5) a[i]=1;
			else a[i]=0;
		}
	}
}

int main(){
	int n, k, temp;
	fr(n);
	fr(k);
	a.resize(1024);
	for(int i=1; i<=n; ++i) {
		fr(temp);
		a[temp]=1;
	}
	exp(k);
	for(int i=0; i<ans.size(); ++i) {
		if(real(ans[i])>0.5) printf("%d ",i);
```

```
    }
    return 0;
}
```

6.3 miller-rabin

```
// For n < 2^64; p = 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37.

namespace MillerRabin {
  typedef long long LL;
  long long modpow(LL a, LL n, LL mod) {
    LL res = 1;
    for (; n > 0; n >>= 1) {
      if (n & 1) res = (res * a) % mod;
      a = (a * a) % mod;
    }
    return res;
  }

  // false => composite; true => maybe prime
  bool witness(LL N, int a, LL d) {
    LL x = modpow(a, d, N);
    if (x == 1 || x == N - 1) return true;
    for (; d != N - 1; d <<= 1) {
      x = (x * x) % N;
      if (x == 1) return false;
      if (x == N - 1) return true;
    }
    return false;
  }

  int wit[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
  bool is_prime(LL N) {
    if (N <= 1) return false;
    LL d;
    for (d = N - 1; d % 2 == 0; d >>= 1);
    for (int p: wit) {
      if (p > N - 2) break;
      // is_prime = is_prime && witness(...)
      if (!witness(N, p, d)) return false;
    }
    return true;
  }
}
```

```
}
```

7 misc

7.1 mo-updates

```
// https://www.codechef.com/viewplaintext/12824787

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

const int MAX = 2e5 + 5;
const int LOG = 25;

int p[MAX][LOG + 1], in[MAX], out[MAX], lev[MAX], getNode[MAX], c[MAX],
    frq[MAX], fans[MAX], vis[MAX];
int timer = 0, BLOCK, N, Q, ans = 0;

map<int,int> M;

vector<int> v[MAX];
vector<int> w;
vector<pair<int, pair<int,int> > >qr;

struct query{
       int l, r, qtime, utime, lca;
};

query que[MAX];

int id[MAX], cc[MAX];
//
void dfs(int cur, int par){
       p[cur][0] = par;
       lev[cur] = lev[par] + 1;
       in[cur] = ++timer;
       getNode[timer] = cur;
       for(auto it:v[cur]){
              if(it == par) continue;
              dfs(it, cur);
       }
       out[cur] = ++timer;
```

```
        getNode[timer] = cur;
}

int getLCA(int x, int y){
        if(lev[x] < lev[y]) swap(x, y);
        int d = lev[x] - lev[y];
        for(int i=0; i<LOG; ++i){
                if(p[x][i]){
                        if((d & (1 << i))){
                                x = p[x][i];
                        }
                }
        }
        if(x == y) return x;
        for(int i=LOG-1; i>=0; --i){
                if((p[x][i] && p[y][i]) && (p[x][i] != p[y][i])){
                        x = p[x][i];
                        y = p[y][i];
                }
        }
        // assert(p[x][0] == p[y][0]);
        return p[x][0];
}

bool compare(query &x, query &y){
        if(x.l / BLOCK == y.l / BLOCK){
                if(x.r / BLOCK == y.r / BLOCK){
                        return x.utime < y.utime;
                }
                return x.r / BLOCK < y.r / BLOCK;
        }
        return x.l / BLOCK < y.l / BLOCK;
}

int add(int node){
        int val = c[node];
        if(frq[val] == 0) ++ans;
        ++frq[val];
}

int remove(int node){
        int val = c[node];
        if(frq[val] == 1) --ans;
        --frq[val];
}
```

```
void move(int idx){
        if(idx == 0) return;
        int node = getNode[idx];
        if(vis[node]) remove(node);
        else add(node);
        vis[node] ^= 1;
}

void refresh(int time){
        int node = id[time];
        if(vis[node]) remove(node);
        swap(c[node], cc[time]);
        if(vis[node]) add(node);
}

int main(){

        ios::sync_with_stdio(false);
        cin.tie(0); cout.tie(0);

        cin >> N >> Q;
        // assert(N*Q <= 20000000);
        for(int i=1; i<=N; ++i){
                cin >> c[i];
                w.push_back(c[i]);
        }
        for(int i=1; i<N; ++i){
                int x, y; cin >> x >> y;
                v[x].push_back(y);
                v[y].push_back(x);
        }

        if(N <= 100000) BLOCK = 1750;
        if(N <= 90000) BLOCK = 1630;
        if(N <= 80000) BLOCK = 1510;
        if(N <= 10000) BLOCK = 390;
        dfs(1, 0);

        for(int i=1; i<LOG; ++i){
                for(int j=1; j<=N; ++j){
                        if(p[j][i-1] && p[p[j][i-1]][i-1]){
                                p[j][i] = p[p[j][i-1]][i-1];
                        }
                }
```

```
}

for(int i=1; i<=Q; ++i){
	int x, y, z; cin >> x >> y >> z;
	qr.push_back({x, {y, z}});
	if(x == 2) w.push_back(z);
}

sort(w.begin(), w.end());
int xx = 0;
for(auto it:w){
	if(M[it] == 0){
		M[it] = ++xx;
	}
}
for(int i=1; i<=N; ++i) c[i] = M[c[i]];
for(auto &it:qr){
	if(it.first == 2){
		it.second.second = M[it.second.second];
	}
}

int X = 1, Y = 1;
for(int i=1; i<=Q; ++i){
	int t, x, y;
	t = qr[i-1].first;
	x = qr[i-1].second.first;
	y = qr[i-1].second.second;
	if(t == 1){
		if(in[x] > in[y]) swap(x, y);
		int z = getLCA(x, y);
		if(z == x){
			que[X].l = in[x];
			que[X].r = in[y];
			que[X].qtime = X;
			que[X].utime = Y;
			que[X].lca = 0;
		}
		else{
			que[X].l = out[x];
			que[X].r = in[y];
			que[X].qtime = X;
			que[X].utime = Y;
			que[X].lca = z;
		}
		assert(que[X].l <= que[X].r);
		++X;
	}
	else{
		id[Y] = x;
		cc[Y] = y;
		++Y;
	}
}


sort(que+1, que+X, compare);

int l = 0, r = -1, T = 1;

for(int i=1; i<X; ++i){
	query &q = que[i];
	// if(q.l > q.r) swap(q.l, q.r);
	// assert(q.l <= q.r);
	while(q.l < l) move(--l);
	while(r < q.r) move(++r);
	while(l < q.l) move(l++);
	while(q.r < r) move(r--);
	while(T < q.utime) refresh(T++);
	while(q.utime < T) refresh(--T);
	if(q.lca) add(q.lca);
	fans[q.qtime] = ans;
	if(q.lca) remove(q.lca);
}


for(int i=1; i<X; ++i) cout << fans[i] << '\n';

// for(int i=1; i<X; ++i) cout << getNode[que[i].l] << ' ' <<
    getNode[que[i].r] << ' ' << que[i].qtime << ' ' <<
    que[i].utime << ' ' << que[i].lca << endl;

}
```

8 strings

8.1 aho-corasick

```
// Aho-corasick

// Implemetation 1

const int MAXN = 404, MOD = 1e9 + 7, sigma = 26;

int cnt[MAXN], term[MAXN], len[MAXN], to[MAXN][sigma], link[MAXN], sz = 1;
void add_str(string s)
{
	int cur = 0;
	for(auto c: s)
	{
		if(!to[cur][c - 'a'])
		{
			to[cur][c - 'a'] = sz++;
			len[to[cur][c - 'a']] = len[cur] + 1;
		}
		cur = to[cur][c - 'a'];
	}
	term[cur] = cur;
}

void push_links()
{
	int que[sz];
	int st = 0, fi = 1;
	que[0] = 0;
	while(st < fi)
	{
		int V = que[st++];
		int U = link[V];
		if(!term[V]) term[V] = term[U];
		for(int c = 0; c < sigma; c++)
			if(to[V][c])
			{
				link[to[V][c]] = V ? to[U][c] : 0;
				que[fi++] = to[V][c];
			}
			else
			{
				to[V][c] = to[U][c];
			}
	}
}

void go(string &s){
	int x = 0;
	s+=  $  ;
	for(auto &c: s) {
		while(1){
			if(to[x].count(c)) {
				x=to[x][c];
				break;
			}
			if(!x) {
				break;
			}
			x = link[x];
		}
		cnt[x]++;
	}
}

// Implementation 2

const int MAXN = 1e6 + 42;

map<char, int> to[MAXN];
int cnt[MAXN], link[MAXN], que[MAXN], sz = 1;

void add_str(string s, int k)
{
	int v = 0;
	for(auto c: s)
	{
		if(!to[v][c]) to[v][c] = sz++;
		v = to[v][c];
	}
}

void push_links()
{
	link[0] = -1;
	int st = 0, fi = 1;
	que[0] = 0;
	while(st < fi)
	{
		int v = que[st++];
		for(auto it: to[v])
```

```
                {
                        int u = it.second;
                        int c = it.first;
                        int j = link[v];
                        while(j != -1 && !to[j][c]) j = link[j];
                        if(j != -1) link[u] = to[j][c];
                        que[fi++] = u;
                }
        }
}

void go(string &s){
        int x = 0;
        s+=  $  ;
        for(auto &c: s) {
                while(1){
                        if(to[x].count(c)) {
                                x=to[x][c];
                                break;
                        }
                        if(!x) {
                                break;
                        }
                        x = link[x];
                }
                cnt[x]++;
        }
}
```

8.2 suffix-array

```
// Suffix Array

// Call computeSA, it will calculate suffix array in sa[], position of
    suffix i...n in sa[] in pos[]
// and sparse table in tree[][].
// Query lcp for suffix i...n and j...n using calcLCP(i, j)

#include<bits/stdc++.h>
#define fr(x) scanf("%d",&x)
using namespace std;

const int LEN = 100020, LOGLEN = 20;

char s[LEN];
int pos[LEN], sa[LEN], ra[2*LEN], temp[LEN], tree[LOGLEN][LEN], val[LEN],
    msb[LEN];

void computeSA() {
        int len, k;
        len=strlen(s);
        for(int mx=0, i=0;i<len;i++) {
                if(i>=(1<<(mx+1)))
                        ++mx;
                msb[i]=mx;
        }
        for(int i=0; i<(len<<1); ++i) {
                ra[i] = -1;
        }
        for(int i=0; i<len; ++i) {
                sa[i] = i;
                ra[i] = s[i];
        }
        k = 0;
        sort(sa, sa+len, [](int a,int b) {
                return ra[a]<ra[b];
        });
        k=1;
        while(k<len) {
                sort(sa, sa+len, [k](int a,int b) {
                        return
                            (ra[a]==ra[b])?(ra[a+k]<ra[b+k]):(ra[a]<ra[b]);
                });
                temp[sa[0]] = 1;
                for(int i=1; i<len; ++i){
                        if(ra[sa[i]]==ra[sa[i-1]]&&ra[sa[i]+k]==ra[sa[i-1]+k])
                            temp[sa[i]]=temp[sa[i-1]];
                        else temp[sa[i]]=temp[sa[i-1]]+1;
                }
                for(int i=0; i<len; ++i) ra[i]=temp[i];
                k<<=1;
        }
        for(int i=0; i<len; ++i) {
                pos[sa[i]] = i;
        }
        for(int i=0, l=0; i<len; ++i) {
                if(pos[i]==0) {
                        l=0;
```

```
                continue;
            }
            while(s[i+l] == s[sa[pos[i]-1]+l]) l++;
            tree[0][pos[i]] = pos[i];
            val[pos[i]] = l;
            l = max(0, l-1);
        }
        for(int j = 1; j < LOGLEN; j++) {
            for(int i = 0; i + (1 << j) - 1 < len; i++) {
                if(val[tree[j - 1][i]] < val[tree[j - 1][i + (1 <<
                    (j - 1))]]) {
                    tree[j][i] = tree[j-1][i];
                }
                else {
                    tree[j][i] = tree[j-1][i + (1 << (j - 1))];
                }
            }
        }
}

int query(int a, int b) {
        int k=msb[b-a+1];
        if(val[tree[k][a]] < val[tree[k][b-(1<<k)+1]]) {
            return tree[k][a];
        }
        else return tree[k][b-(1<<k)+1];
}

int calcLCP(int l, int r) {
        return val[query(min(pos[l], pos[r])+1, max(pos[l], pos[r]))];
}

int main(){
        int t, len;
        long long ans;
        fr(t);
        while(t--){
            scanf(" %s",&s);
            len=strlen(s);
            computeSA();
            ans = len;
            for(int i=1; i<len; ++i) {
                ans+=calcLCP(0, i);
            }
            printf("%lld\n",ans);
        }
        return 0;
}
```

8.3 suffix-automaton

```
struct SuffixAutomaton {
        vector<map<char,int>> edges; // edges[i] : the labeled edges from
            node i
        vector<int> link;          // link[i]  : the parent of i
        vector<int> length;        // length[i] : the length of the
            longest string in the ith class
        int last;                  // the index of the equivalence class
            of the whole string

        SuffixAutomaton(string s) {
            // add the initial node
            edges.push_back(map<char,int>());
            link.push_back(-1);
            length.push_back(0);
            last = 0;

            for(int i=0;i<s.size();i++) {
                // construct r
                edges.push_back(map<char,int>());
                length.push_back(i+1);
                link.push_back(0);
                int r = edges.size() - 1;

                // add edges to r and find p with link to q
                int p = last;
                while(p >= 0 && edges[p].find(s[i]) ==
                    edges[p].end()) {
                    edges[p][s[i]] = r;
                    p = link[p];
                }
                if(p != -1) {
                    int q = edges[p][s[i]];
                    if(length[p] + 1 == length[q]) {
                        // we do not have to split q, just
                            set the correct suffix link
                        link[r] = q;
                    } else {
```

```
                        // we have to split, add q'
                        edges.push_back(edges[q]); // copy
                            edges of q
                        length.push_back(length[p] + 1);
                        link.push_back(link[q]); // copy
                            parent of q
                        int qq = edges.size()-1;
                        // add qq as the new parent of q and r
                        link[q] = qq;
                        link[r] = qq;
                        // move short classes pointing to q
                            to point to q'
                        while(p >= 0 && edges[p][s[i]] == q) {
                                edges[p][s[i]] = qq;
                                p = link[p];
                        }
                    }
                }
                last = r;
            }

            vector<int> terminals;
            int p = last;
            while(p > 0) {
                terminals.push_back(p);
                p = link[p];
            }
        }
};
```

8.4 z-algo

```
// Z Algo

// computeZ calculates the Z function for the string s and stores it in
    the array z[].

#include<bits/stdc++.h>
#define fr(x) scanf("%d",&x)
using namespace std;

const int LEN = 2000010;
string s;
int z[LEN];

void computeZ() {
        int len, l, r;
        len = s.length();
        z[0] = len;
        l = r = -1;
        for(int i=1; i<len; ++i) {
                z[i]=0;
                if(r>i) {
                        z[i]=min(z[i-l],r-i+1);
                }
                while(s[i+z[i]]==s[z[i]]) ++z[i];
                if(i+z[i]-1 > r) {
                        r=i+z[i]-1;
                        l=i;
                }
        }
}

char temps[1000010];
int main(){
        int t;
        fr(t);
        string a, b;
        vector<int> ans;
        while(t--) {
                ans.clear();
                scanf(" %s", temps);
                b = string(temps);
                scanf(" %s", temps);
                a = string(temps);
                s = a + '$' + b;
                computeZ();
                for(int i=a.length(); i<s.length(); ++i) {
                        if(z[i] == a.length()) {
                                ans.push_back(i - a.length());
                        }
                }
                if(ans.size()) {
                        printf("%d\n", ans.size());
                        for(auto&z2: ans) {
                                printf("%d ", z2);
                        }
```

```
                puts("");
            }
            else {
                printf("Not Found\n");
            }
        }
}
```