TEAM DEAD TEMPLATES

mhtkrag

December 18, 2023

Contents

1 data-structures

1.1 dsu

```
struct DSU {
  vector<int> parent, siz;
  void init(int n) {
    parent.resize(n);
    siz.resize(n);
    for (int i = 0; i < n; i++) {
      parent[i] = i;
      siz[i] = 1;
    }
  }
  int find(int x) {
    if (x == parent[x]) return x;
    return parent[x] = find(parent[x]);
  }
  void merge(int x, int y) {
    x = find(x);
    y = find(y);
    if (x == y) return;
    if (siz[x] < siz[y]) swap(x, y);
    parent[y] = x;
    siz[x] += siz[y];
  }
  int size(int x) { return siz[find(x)]; }
  bool same(int x, int y) { return find(x) == find(y); }
}
```

1.2 fenwick-tree

```
template <class T>
class BIT {
 private:
  int size;
  vector<T> bit;
  vector<T> arr;

 public:
  BIT(int size) : size(size), bit(size + 1), arr(size)
      {}

  /** Sets the value at index ind to val. */
  void set(int ind, int val) { add(ind, val -
      arr[ind]); }

  /** Adds val to the element at index ind. */
  void add(int ind, int val) {
    arr[ind] += val;
    ind++;
    for (; ind <= size; ind += ind & -ind) {
      bit[ind] += val;
    }
  }

  /** @return The sum of all values in [0, ind]. */
  T pref_sum(int ind) {
    ind++;
    T total = 0;
    for (; ind > 0; ind -= ind & -ind) {
      total += bit[ind];
    }
    return total;
  }
};
```

1.3 lazy-segment-tree

```
template <typename node_type, typename tag_type>
struct lazy_segtree {
  vector<node_type> tree;
  vector<tag_type> lazy;
  int n;
  template <typename Iter>
  void init(Iter first, Iter last, int nn = -1) {
    n = nn;
    if (n == -1) n = distance(first, last);
    tree.resize(4 * n);
    lazy.resize(4 * n);
    build_tree(0, 0, n - 1, first);
  }
  node_type query(int ql, int qr) { return query(0, 0,
      n - 1, ql, qr); }
  void update(int ql, int qr, tag_type const &val) {
    update(0, 0, n - 1, ql, qr, val);
  }

 private:
  template <typename Iter>
  void build_tree(int id, int tl, int tr, Iter first) {
    if (tl == tr) {
      tree[id].init(tl, tr, *(first + tl));
      lazy[id].init(tl, tr);
      return;
    }
    int tm = (tl + tr) / 2;
    build_tree(2 * id + 1, tl, tm, first);
    build_tree(2 * id + 2, tm + 1, tr, first);
    tree[id] = node_type::merge(tree[2 * id + 1],
        tree[2 * id + 2]);
    lazy[id].init(tl, tr);
  }
  void push(int id, int tl, int tr) {
    if (tl != tr) {
      int tm = (tl + tr) / 2;
      tree[2 * id + 1].apply(tl, tm, lazy[id]);
      lazy[2 * id + 1].merge(lazy[id]);
      tree[2 * id + 2].apply(tm + 1, tr, lazy[id]);
      lazy[2 * id + 2].merge(lazy[id]);
    }
    lazy[id].reset();
  }
  node_type query(int id, int tl, int tr, int ql, int
      qr) {
    if (tl > qr || ql > tr) return node_type::phi();
    if (ql <= tl && tr <= qr) return tree[id];
    push(id, tl, tr);
    int tm = (tl + tr) / 2;
    return node_type::merge(query(2 * id + 1, tl, tm,
        ql, qr),
                            query(2 * id + 2, tm + 1, tr,
                                ql, qr));
  }
  void update(int id, int tl, int tr, int ql, int qr,
      tag_type const &val) {
    if (tl > qr || ql > tr) return;
    if (ql <= tl && tr <= qr) {
      tree[id].apply(tl, tr, val);
      lazy[id].merge(val);
```

```
      return;
    }
    push(id, tl, tr);
    int tm = (tl + tr) / 2;
    update(2 * id + 1, tl, tm, ql, qr, val);
    update(2 * id + 2, tm + 1, tr, ql, qr, val);
    tree[id] = node_type::merge(tree[2 * id + 1],
        tree[2 * id + 2]);
  }
};
struct tag {
  ll inc;
  void reset() { inc = 0; }
  void merge(tag const &other) { inc += other.inc; }
  void init(int tl, int tr) {}
};
struct node {
  static node phi() { return {0LL}; }
  static node merge(node const &a, node const &b) {
      return {a.data + b.data}; }
  ll data;
  void apply(ll tl, ll tr, tag const &t) { data += (tr
      - tl + 1) * t.inc; }
  template <typename T>
  void init(ll tl, ll tr, T &ddata) {
    this->data = ddata;
  }
};
```

1.4 segment-tree-beats

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

const int N = 2e5 + 9;

using ll = long long;

struct SGTBeats {
  const ll inf = 1e18;
  int n, n0;
  ll max_v[4 * N], smax_v[4 * N], max_c[4 * N];
  ll min_v[4 * N], smin_v[4 * N], min_c[4 * N];
  ll sum[4 * N];
  ll len[4 * N], ladd[4 * N], lval[4 * N];

  void update_node_max(int k, ll x) {
    sum[k] += (x - max_v[k]) * max_c[k];

    if (max_v[k] == min_v[k]) {
      max_v[k] = min_v[k] = x;
    } else if (max_v[k] == smin_v[k]) {
      max_v[k] = smin_v[k] = x;
    } else {
      max_v[k] = x;
    }

    if (lval[k] != inf && x < lval[k]) {
      lval[k] = x;
    }
  }
  void update_node_min(int k, ll x) {
    sum[k] += (x - min_v[k]) * min_c[k];

    if (max_v[k] == min_v[k]) {
      max_v[k] = min_v[k] = x;
    } else if (smax_v[k] == min_v[k]) {
      min_v[k] = smax_v[k] = x;
    } else {
      min_v[k] = x;
    }

    if (lval[k] != inf && lval[k] < x) {
      lval[k] = x;
    }
  }
  void push(int k) {
    if (n0 - 1 <= k) return;
    if (lval[k] != inf) {
      updateall(2 * k + 1, lval[k]);
      updateall(2 * k + 2, lval[k]);
      lval[k] = inf;
      return;
    }
    if (ladd[k] != 0) {
      addall(2 * k + 1, ladd[k]);
      addall(2 * k + 2, ladd[k]);
      ladd[k] = 0;
    }
    if (max_v[k] < max_v[2 * k + 1]) {
      update_node_max(2 * k + 1, max_v[k]);
    }
    if (min_v[2 * k + 1] < min_v[k]) {
      update_node_min(2 * k + 1, min_v[k]);
    }

    if (max_v[k] < max_v[2 * k + 2]) {
      update_node_max(2 * k + 2, max_v[k]);
    }
    if (min_v[2 * k + 2] < min_v[k]) {
      update_node_min(2 * k + 2, min_v[k]);
    }
  }
  void update(int k) {
    sum[k] = sum[2 * k + 1] + sum[2 * k + 2];

    if (max_v[2 * k + 1] < max_v[2 * k + 2]) {
      max_v[k] = max_v[2 * k + 2];
      max_c[k] = max_c[2 * k + 2];
      smax_v[k] = max(max_v[2 * k + 1], smax_v[2 * k +
          2]);
    } else if (max_v[2 * k + 1] > max_v[2 * k + 2]) {
      max_v[k] = max_v[2 * k + 1];
      max_c[k] = max_c[2 * k + 1];
      smax_v[k] = max(smax_v[2 * k + 1], max_v[2 * k +
          2]);
    } else {
      max_v[k] = max_v[2 * k + 1];
      max_c[k] = max_c[2 * k + 1] + max_c[2 * k + 2];
      smax_v[k] = max(smax_v[2 * k + 1], smax_v[2 * k +
          2]);
    }

    if (min_v[2 * k + 1] < min_v[2 * k + 2]) {
      min_v[k] = min_v[2 * k + 1];
      min_c[k] = min_c[2 * k + 1];
      smin_v[k] = min(smin_v[2 * k + 1], min_v[2 * k +
          2]);
    } else if (min_v[2 * k + 1] > min_v[2 * k + 2]) {
      min_v[k] = min_v[2 * k + 2];
      min_c[k] = min_c[2 * k + 2];
```

```
    smin_v[k] = min(min_v[2 * k + 1], smin_v[2 * k +
        2]);
  } else {
    min_v[k] = min_v[2 * k + 1];
    min_c[k] = min_c[2 * k + 1] + min_c[2 * k + 2];
    smin_v[k] = min(smin_v[2 * k + 1], smin_v[2 * k +
        2]);
  }
}
void _update_min(ll x, int a, int b, int k, int l,
    int r) {
  if (b <= l || r <= a || max_v[k] <= x) {
    return;
  }
  if (a <= l && r <= b && smax_v[k] < x) {
    update_node_max(k, x);
    return;
  }
  push(k);
  _update_min(x, a, b, 2 * k + 1, l, (l + r) / 2);
  _update_min(x, a, b, 2 * k + 2, (l + r) / 2, r);
  update(k);
}
void _update_max(ll x, int a, int b, int k, int l,
    int r) {
  if (b <= l || r <= a || x <= min_v[k]) {
    return;
  }
  if (a <= l && r <= b && x < smin_v[k]) {
    update_node_min(k, x);
    return;
  }
  push(k);
  _update_max(x, a, b, 2 * k + 1, l, (l + r) / 2);
  _update_max(x, a, b, 2 * k + 2, (l + r) / 2, r);
  update(k);
}
void addall(int k, ll x) {
  max_v[k] += x;
  if (smax_v[k] != -inf) smax_v[k] += x;
  min_v[k] += x;
  if (smin_v[k] != inf) smin_v[k] += x;

  sum[k] += len[k] * x;
  if (lval[k] != inf) {
    lval[k] += x;
  } else {
    ladd[k] += x;
  }
}
void updateall(int k, ll x) {
  max_v[k] = x; smax_v[k] = -inf;
  min_v[k] = x; smin_v[k] = inf;
  max_c[k] = min_c[k] = len[k];

  sum[k] = x * len[k];
  lval[k] = x; ladd[k] = 0;
}
void _add_val(ll x, int a, int b, int k, int l, int
    r) {
  if (b <= l || r <= a) {
    return;
  }
  if (a <= l && r <= b) {
    addall(k, x);
    return;
  }
  push(k);
  _add_val(x, a, b, 2 * k + 1, l, (l + r) / 2);
  _add_val(x, a, b, 2 * k + 2, (l + r) / 2, r);
  update(k);
}
void _update_val(ll x, int a, int b, int k, int l,
    int r) {
  if (b <= l || r <= a) {
    return;
  }
  if (a <= l && r <= b) {
    updateall(k, x);
    return;
  }
  push(k);
  _update_val(x, a, b, 2 * k + 1, l, (l + r) / 2);
  _update_val(x, a, b, 2 * k + 2, (l + r) / 2, r);
  update(k);
}

ll _query_max(int a, int b, int k, int l, int r) {
  if (b <= l || r <= a) {
    return -inf;
  }
  if (a <= l && r <= b) {
    return max_v[k];
  }
  push(k);
  ll lv = _query_max(a, b, 2 * k + 1, l, (l + r) / 2);
  ll rv = _query_max(a, b, 2 * k + 2, (l + r) / 2, r);
  return max(lv, rv);
}

ll _query_min(int a, int b, int k, int l, int r) {
  if (b <= l || r <= a) {
    return inf;
  }
  if (a <= l && r <= b) {
    return min_v[k];
  }
  push(k);
  ll lv = _query_min(a, b, 2 * k + 1, l, (l + r) / 2);
  ll rv = _query_min(a, b, 2 * k + 2, (l + r) / 2, r);
  return min(lv, rv);
}

ll _query_sum(int a, int b, int k, int l, int r) {
  if (b <= l || r <= a) {
    return 0;
  }
  if (a <= l && r <= b) {
    return sum[k];
  }
  push(k);
  ll lv = _query_sum(a, b, 2 * k + 1, l, (l + r) / 2);
  ll rv = _query_sum(a, b, 2 * k + 2, (l + r) / 2, r);
  return lv + rv;
}

SGTBeats(int n, ll *a) : n(n) {
  n0 = 1;
  while (n0 < n) n0 <<= 1;
  for (int i = 0; i < 2 * n0; ++i) ladd[i] = 0,
      lval[i] = inf;
  len[0] = n0;
  for (int i = 0; i < n0 - 1; ++i) len[2 * i + 1] =
      len[2 * i + 2] = (len[i] >> 1);
```

```
    for (int i = 0; i < n; ++i) {
      max_v[n0 - 1 + i] = min_v[n0 - 1 + i] = sum[n0 -
          1 + i] = (a != nullptr ? a[i] : 0);
      smax_v[n0 - 1 + i] = -inf;
      smin_v[n0 - 1 + i] = inf;
      max_c[n0 - 1 + i] = min_c[n0 - 1 + i] = 1;
    }
    for (int i = n; i < n0; ++i) {
      max_v[n0 - 1 + i] = smax_v[n0 - 1 + i] = -inf;
      min_v[n0 - 1 + i] = smin_v[n0 - 1 + i] = inf;
      max_c[n0 - 1 + i] = min_c[n0 - 1 + i] = 0;
    }
    for (int i = n0 - 2; i >= 0; i--) {
      update(i);
    }
  }

  // all queries are performed on [l, r) segment (right
      exclusive)
  // 0 indexed

  // range minimize query
  void update_min(int a, int b, ll x) {
    _update_min(x, a, b, 0, 0, n0);
  }
  // range maximize query
  void update_max(int a, int b, ll x) {
    _update_max(x, a, b, 0, 0, n0);
  }
  // range add query
  void add_val(int a, int b, ll x) {
    _add_val(x, a, b, 0, 0, n0);
  }
  // range update query
  void update_val(int a, int b, ll x) {
    _update_val(x, a, b, 0, 0, n0);
  }
  // range minimum query
  ll query_max(int a, int b) {
    return _query_max(a, b, 0, 0, n0);
  }
  // range maximum query
  ll query_min(int a, int b) {
    return _query_min(a, b, 0, 0, n0);
  }
  // range sum query
  ll query_sum(int a, int b) {
    return _query_sum(a, b, 0, 0, n0);
  }
};

ll a[N];
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, q; cin >> n >> q;
  for (int i = 0; i < n; i++) {
    cin >> a[i];
  }
  SGTBeats t(n, a);
  while (q--) {
    int ty, l, r; cin >> ty >> l >> r;
    ll x; if (ty < 3) cin >> x;
    if (ty == 0) {
      t.update_min(l, r, x);
    }
    else if (ty == 1) {
      t.update_max(l, r, x);
    }
    else if (ty == 2) {
      t.add_val(l, r, x);
    }
    else {
      cout << t.query_sum(l, r) << '\n';
    }
  }
  return 0;
}
//
    https://judge.yosupo.jp/problem/range_chmin_chmax_add_rang
```

1.5 segment-tree

```
template <typename T, typename CombineT>
struct SegmentTree {
  vector<T> tree;
  CombineT combine;
  T defaultValue;
  ll n;

  void init(ll n, T val) {
    tree.resize(4 * n);
    this->n = n;
    defaultValue = val;
    build_tree(1, 0, n - 1);
  }

  template <typename Itr>
  void init(Itr begin, Itr end) {
    n = distance(begin, end);
    tree.resize(4 * n);

    build_tree(1, 0, n - 1, begin);
  }

  void build_tree(ll id, ll tl, ll tr) {
    if (tl == tr) {
      tree[id] = defaultValue;
      return;
    }
    ll tm = (tl + tr) / 2;
    build_tree(id * 2, tl, tm);
    build_tree(id * 2 + 1, tm + 1, tr);
    tree[id] = combine(tree[id * 2], tree[id * 2 + 1]);
  }

  template <typename Itr>
  void build_tree(ll id, ll tl, ll tr, Itr begin) {
    if (tl == tr) {
      tree[id] = *(begin + tl);
      return;
    }
    ll tm = (tl + tr) / 2;
    build_tree(id * 2, tl, tm, begin);
    build_tree(id * 2 + 1, tm + 1, tr, begin);
    tree[id] = combine(tree[id * 2], tree[id * 2 + 1]);
  }

  T query(ll id, ll tl, ll tr, ll ql, ll qr) {
    if (ql > tr || tl > qr) return defaultValue;
    if (ql <= tl && tr <= qr) return tree[id];
```

```
    ll tm = (tl + tr) / 2;
    return combine(query(id * 2, tl, tm, ql, qr),
                   query(id * 2 + 1, tm + 1, tr, ql, qr));
  }

  T query(ll l, ll r) { return query(1, 0, n - 1, l,
      r); }

  void update(ll id, ll tl, ll tr, ll p, T x) {
    if (tl == tr) {
      tree[id] = x;
      return;
    }
    ll tm = (tl + tr) / 2;
    if (p <= tm)
      update(id * 2, tl, tm, p, x);
    else
      update(id * 2 + 1, tm + 1, tr, p, x);
    tree[id] = combine(tree[id * 2], tree[id * 2 + 1]);
  }

  void update(ll p, T x) { update(1, 0, n - 1, p, x); }
};
```

1.6 sparse-table

```
struct min_op {
  ll operator()(ll a, ll b) { return min(a, b); }
};
struct max_op {
  ll operator()(ll a, ll b) { return max(a, b); }
};
struct gcd_op {
  ll operator()(ll a, ll b) { return __gcd(a, b); }
};
template <typename OperationT>
struct sparse_table {
  vector<vector<ll>> m;
  OperationT op;
  template <typename Itr>
  void init(Itr begin, Itr end) {
    ll sz = end - begin;
    ll lg = 63 - __builtin_clzll(sz);
    m.assign(sz, vector<ll>(lg + 1));
    for (ll j = 0; j <= lg; ++j) {
      ll len = (1 << j);
      for (ll i = 0; i + len - 1 < sz; ++i) {
        if (len == 1) {
          m[i][j] = *(begin + i);
        } else {
          m[i][j] = op(m[i][j - 1], m[i + (1 << (j -
              1))][j - 1]);
        }
      }
    }
  }
  ll query(ll L, ll R) {
    ll j = 63 - __builtin_clzll((R - L + 1));
    return op(m[L][j], m[R + 1 - (1 << j)][j]);
  }
};
```

2 dp

2.1 sos-dp

```
const ll MLOG = 20;
const ll MAXN = 1 << MLOG;
ll dp[MAXN];

void forward1() { // adding element to all its super set
  for (ll bit = 0; bit < MLOG; ++bit) {
    for (ll i = 0; i < MAXN; ++i) {
      if (i & (1 << bit)) {
        dp[i] += dp[i ^ (1 << bit)];
      }
    }
  }
}

void backward1() { // add a[i] to a[j] if j&i = i
  for (ll bit = 0; bit < MLOG; ++bit) {
    for (ll i = MAXN - 1; i >= 0; --i) {
      if (i & (1 << bit)) {
        dp[i] -= dp[i ^ (1 << bit)];
      }
    }
  }
}
void forward2() { // add elements to its subsets
  for (ll bit = 0; bit < MLOG; ++bit) {
    for (ll i = MAXN - 1; i >= 0; --i) {
      if (i & (1 << bit)) {
        dp[i ^ (1 << bit)] += dp[i];
      }
    }
  }
}
void backward2() {
  for (ll bit = 0; bit < MLOG; ++bit) {
    for (ll i = 0; i < MAXN; ++i) {
      if (i & (1 << bit)) {
        dp[i ^ (1 << bit)] -= dp[i];
      }
    }
  }
}
```

3 graph

3.1 bellaman-ford

```
const int N = 3e5 + 9;
struct st {
  int a, b, cost;
} e[N];
const int INF = 2e9;
int32_t main() {
  int n, m;
  cin >> n >> m;
  for(int i = 0; i < m; i++) cin >> e[i].a >> e[i].b >>
      e[i].cost;
  int s;
  cin >> s;//is there any negative cycle which is
      reachable from s?
```

```
  vector<int> d (n, INF);//for finding any cycle(not
      necessarily from s) set d[i] = 0 for all i
  d[s] = 0;
  vector<int> p (n, -1);
  int x;
  for (int i=0; i<n; ++i) {
    x = -1;
    for (int j=0; j<m; ++j) {
      if (d[e[j].a] < INF) {
        if (d[e[j].b] > d[e[j].a] + e[j].cost) {
          d[e[j].b] = max (-INF, d[e[j].a] +
              e[j].cost);//for overflow
          p[e[j].b] = e[j].a;
          x = e[j].b;
        }
      }
    }
  }
  if (x == -1) cout << "No negative cycle from "<<s;
  else {
    int y = x; //x can be on any cycle or reachable
        from some cycle
    for (int i=0; i<n; ++i) y = p[y];

    vector<int> path;
    for (int cur=y; ; cur=p[cur]) {
      path.push_back (cur);
      if (cur == y && path.size() > 1) break;
    }
    reverse (path.begin(), path.end());

    cout << "Negative cycle: ";
    for (int i=0; i<path.size(); ++i) cout << path[i]
        << ' ';
  }
  return 0;
}
```

3.2 floyd-warshall

```
// d is the adjacency matrix
int d[N][N];

  for (int k = 1; k <= n; ++k) {
    for (int i = 1; i <= n; ++i) {
      for (int j = 1; j <= n; ++j) {
        d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
      }
    }
  }
```

3.3 scc

```
vector<vector<int>> adj, adj_rev;
vector<bool> used;
vector<int> order, component;

void dfs1(int v) {
  used[v] = true;

  for (auto u : adj[v])
    if (!used[u]) dfs1(u);
  order.push_back(v);
}

void dfs2(int v) {
  used[v] = true;
  component.push_back(v);

  for (auto u : adj_rev[v])
    if (!used[u]) dfs2(u);
}

int main() {
  int n;
  // ... read n ...

  for (;;) {
    int a, b;
    // ... read next directed edge (a,b) ...
    adj[a].push_back(b);
    adj_rev[b].push_back(a);
  }

  used.assign(n, false);

  for (int i = 0; i < n; i++)
    if (!used[i]) dfs1(i);

  used.assign(n, false);
  reverse(order.begin(), order.end());

  for (auto v : order)
    if (!used[v]) {
      dfs2(v);

      // ... processing next component ...

      component.clear();
    }

  // Condensed Graph
  vector<int> roots(n, 0);
  vector<int> root_nodes;
  vector<vector<int>> adj_scc(n);

  for (auto v : order)
    if (!used[v]) {
      dfs2(v);

      int root = component.front();
      for (auto u : component) roots[u] = root;
      root_nodes.push_back(root);

      component.clear();
    }

  for (int v = 0; v < n; v++)
    for (auto u : adj[v]) {
      int root_v = roots[v], root_u = roots[u];

      if (root_u != root_v)
          adj_scc[root_v].push_back(root_u);
    }
}
```

4 misc

4.1 bitwise-tricks

```
// iterating over subsets
for (int x = mask;; x = (x - 1) & mask) {
  // Code here...
  if (x == 0) break;
}
```

4.2 cppt

```
#include <bits/stdc++.h>
using namespace std;
#define all(x) begin(x), end(x)
#define OUT(T) cout << "Case #" << T << ": "
#ifndef _DEBUG
#define endl '\n'
#endif
#ifdef _DEBUG
void dbg_out() { cerr << endl; }
template <typename Head, typename... Tail>
void dbg_out(Head H, Tail... T) {
  cerr << ' ' << H;
  dbg_out(T...);
}
#define dbg(...) cerr << "(" << #__VA_ARGS__ << "):",
    dbg_out(__VA_ARGS__)
#else
#define dbg(...)
#endif
#define ckmin(x, y) x = min((x), (y))
#define ckmax(x, y) x = max((x), (y))

// clang-format off
template <typename T> ostream &operator<<(ostream &out,
    const vector<T> &v) { for (const auto &x : v) out
    << x << ' '; return out; }
template <typename T> istream &operator>>(istream &in,
    vector<T> &v) { for (auto &x : v) in >> x; return
    in; }
// clang-format on

using ll = long long;
using lld = long double;
using pll = pair<ll, ll>;
using pii = pair<int, int>;

void solve(ll _t) {}

int main() {
  ios_base::sync_with_stdio(false), cin.tie(NULL);

  ll T = 1;
  cin >> T;
  for (ll t = 1; t <= T; ++t) solve(t);
}
```

4.3 ordered-set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T, typename ComparatorFn = less<T>>
using ordered_set = tree<T, null_type, ComparatorFn,
    rb_tree_tag,
                         tree_order_statistics_node_update>;
```

5 number-theory

5.1 crt

```
using T = __int128;
// ax + by = __gcd(a, b)
// returns __gcd(a, b)
T extended_euclid(T a, T b, T &x, T &y) {
  T xx = y = 0;
  T yy = x = 1;
  while (b) {
    T q = a / b;
    T t = b; b = a % b; a = t;
    t = xx; xx = x - q * xx; x = t;
    t = yy; yy = y - q * yy; y = t;
  }
  return a;
}
// finds x such that x % m1 = a1, x % m2 = a2. m1 and
    m2 may not be coprime
// here, x is unique modulo m = lcm(m1, m2). returns
    (x, m). on failure, m = -1.
pair<T, T> CRT(T a1, T m1, T a2, T m2) {
  T p, q;
  T g = extended_euclid(m1, m2, p, q);
  if (a1 % g != a2 % g) return make_pair(0, -1);
  T m = m1 / g * m2;
  p = (p % m + m) % m;
  q = (q % m + m) % m;
  return make_pair((p * a2 % m * (m1 / g) % m + q * a1
      % m * (m2 / g) % m) % m, m);
}
```

5.2 euler-totient-function

```
// phi(n) = # of integers in [1, n] that are relatively
    prime to n
// phi(p) = p - 1, where p is prime
// phi(p^k) = p^k - p^(k - 1) = p^k * (1 - 1 / p) for
    prime p;
void phi_1_to_n(int n) {
  vector<int> phi(n + 1);
  for (int i = 0; i <= n; i++) phi[i] = i;

  for (int i = 2; i <= n; i++) {
    if (phi[i] == i) {
      for (int j = i; j <= n; j += i) phi[j] -= phi[j]
          / i;
    }
  }
}
```

5.3 extended-euclid

```
using ll = long long;
ll extended_euclid(ll a, ll b, ll &x, ll &y) {
```

```
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
  }
  ll x1, y1;
  ll d = extended_euclid(b, a % b, x1, y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return d;
}
ll inverse(ll a, ll m) {
  ll x, y;
  ll g = extended_euclid(a, m, x, y);
  if (g != 1) return -1;
  return (x % m + m) % m;
}
```

5.4 modular-int

```
const ll MOD = 1e9 + 7;
ll binexp(ll a, ll b, ll p = MOD) {
  if (b < 0) return 0;
  ll res = 1;
  while (b > 0) {
    if (b & 1) b--, res = (res * a) % p;
    a = (a * a) % p;
    b >>= 1;
  }
  return res;
}

inline ll modinv(ll x, ll p = MOD) { return binexp(x, p
    - 2, p); }
template <ll mod>
struct mi_ {
  ll value;
  mi_() = default;
  mi_(ll x) : value(x % mod) {}
  mi_(const mi_ &m) : value(m.value % mod) {}
  mi_ &operator=(const mi_ &m) {
    value = m.value;
    return *this;
  }
  ll inverse_value() const { return modinv(value, mod);
      }
  mi_ &operator+=(const mi_ &m) {
    value = (value + m.value) % mod;
    return *this;
  }
  mi_ &operator-=(const mi_ &m) {
    value = (mod + value - m.value) % mod;
    return *this;
  }
  mi_ &operator*=(const mi_ &m) {
    value = (value * m.value) % mod;
    return *this;
  }
  mi_ &operator/=(const mi_ &m) {
    value = (value * m.inverse_value()) % mod;
    return *this;
  }
  mi_ &operator++() {
    value++;
    value %= mod;
    return *this;
  }
  mi_ &operator--() {
    value--;
    value %= mod;
    return *this;
  }
  mi_ operator*(const mi_ &b) { return mi_(value *
      b.value); }
  mi_ operator*(ll b) { return mi_(value * b); }
  mi_ operator-(const mi_ &b) { return mi_(mod + value
      - b.value); }
  mi_ operator-(ll b) { return mi_(mod + value - b); }
  mi_ operator+(const mi_ &b) { return mi_(value +
      b.value); }
  mi_ operator+(ll b) { return mi_(value + b); }
  mi_ operator/(const mi_ &b) { return mi_(value *
      modinv(b.value, mod)); }
  mi_ operator/(ll b) { return mi_(value * modinv(b,
      mod)); }
};
template <ll mod>
ostream &operator<<(ostream &out, const mi_<mod> &m) {
  out << m.value % mod;
  return out;
}
template <ll mod>
istream &operator>>(istream &in, mi_<mod> &m) {
  ll x;
  in >> x;
  m.value = (x % mod);
  return in;
}
using mi = mi_<MOD>;
vector<mi> factorial;
void init_factorial() {
  factorial.resize(1000005);
  factorial[0] = factorial[1] = 1;
  for (ll i = 2; i < 1000005; ++i) {
    factorial[i] = (factorial[i - 1] * i);
  }
}
inline mi choose(const mi &a, const mi &b) {
  if (a.value < b.value) return 0;
  return factorial[a.value] /
         (factorial[b.value] * factorial[(a.value -
             b.value)]);
}
```

5.5 polard-rho

```
using ll = long long;
namespace PollardRho {
  mt19937
      rnd(chrono::steady_clock::now().time_since_epoch().count
  const int P = 1e6 + 9;
  ll seq[P];
  int primes[P], spf[P];
  inline ll add_mod(ll x, ll y, ll m) {
    return (x += y) < m ? x : x - m;
  }
  inline ll mul_mod(ll x, ll y, ll m) {
    ll res = __int128(x) * y % m;
    return res;
```

```
  // ll res = x * y - (ll)((long double)x * y / m +
      0.5) * m;
  // return res < 0 ? res + m : res;
}
inline ll pow_mod(ll x, ll n, ll m) {
  ll res = 1 % m;
  for (; n; n >>= 1) {
    if (n & 1) res = mul_mod(res, x, m);
    x = mul_mod(x, x, m);
  }
  return res;
}
// O(it * (logn)^3), it = number of rounds performed
inline bool miller_rabin(ll n) {
  if (n <= 2 || (n & 1 ^ 1)) return (n == 2);
  if (n < P) return spf[n] == n;
  ll c, d, s = 0, r = n - 1;
  for (; !(r & 1); r >>= 1, s++) {}
  // each iteration is a round
  for (int i = 0; primes[i] < n && primes[i] < 32;
      i++) {
    c = pow_mod(primes[i], r, n);
    for (int j = 0; j < s; j++) {
      d = mul_mod(c, c, n);
      if (d == 1 && c != 1 && c != (n - 1)) return
          false;
      c = d;
    }
    if (c != 1) return false;
  }
  return true;
}
void init() {
  int cnt = 0;
  for (int i = 2; i < P; i++) {
    if (!spf[i]) primes[cnt++] = spf[i] = i;
    for (int j = 0, k; (k = i * primes[j]) < P; j++) {
      spf[k] = primes[j];
      if (spf[i] == spf[k]) break;
    }
  }
}
// returns O(n^(1/4))
ll pollard_rho(ll n) {
  while (1) {
    ll x = rnd() % n, y = x, c = rnd() % n, u = 1, v,
        t = 0;
    ll *px = seq, *py = seq;
    while (1) {
      *py++ = y = add_mod(mul_mod(y, y, n), c, n);
      *py++ = y = add_mod(mul_mod(y, y, n), c, n);
      if ((x = *px++) == y) break;
      v = u;
      u = mul_mod(u, abs(y - x), n);
      if (!u) return __gcd(v, n);
      if (++t == 32) {
        t = 0;
        if ((u = __gcd(u, n)) > 1 && u < n) return u;
      }
    }
    if (t && (u = __gcd(u, n)) > 1 && u < n) return u;
  }
}
vector<ll> factorize(ll n) {
  if (n == 1) return vector <ll>();
  if (miller_rabin(n)) return vector<ll> {n};
  vector <ll> v, w;
  while (n > 1 && n < P) {
    v.push_back(spf[n]);
    n /= spf[n];
  }
  if (n >= P) {
    ll x = pollard_rho(n);
    v = factorize(x);
    w = factorize(n / x);
    v.insert(v.end(), w.begin(), w.end());
  }
  return v;
 }
}
```

5.6 sieve

```
struct sieve_t {
  sieve_t(int n, bool gen_primes = false, bool
      gen_sieve = false) {
    is_prime.assign(n + 1, true);
    is_prime[0] = is_prime[1] = false;
    for (int i = 2; i * i <= n; ++i) {
      for (int j = i * i; j <= n; j += i)
        is_prime[j] = false;
    }
    if (gen_primes) {
      for (int i = 2; i <= n; ++i) {
        if (is_prime[i])
          primes.push_back(i);
      }
    }
    if (gen_sieve) {
      sieve.assign(n + 1, -1);
      for (int i = 2; i <= n; ++i) {
        if (is_prime[i]) {
          sieve[i] = i;
          if ((ll)i * i <= n) {
            for (int j = i * i; j <= n; j += i) {
              if (sieve[j] == -1)
                sieve[j] = i;
            }
          }
        }
      }
    }
  }
  // requires gen_fact; works only upto sz;
  vector<int> fast_factorize(int k) {
    vector<int> res;
    while (k > 1) {
      ll p = sieve[k];
      res.push_back(p);
      k /= p;
    }
    return res;
  }
  // requies gen_primes; works upto sz*sz;
  vector<int> factorize(int k) {
    vector<int> res;
    for (int p : primes) {
      if (p * p > k)
        break;
      while (k % p == 0) {
        k /= p;
        res.push_back(p);
```

```
      }
    }
    if (k > 1)
      res.push_back(k);
    return res;
  }
  vector<bool> is_prime;
  vector<int> primes;
  vector<int> sieve;
};
```

6 strings

6.1 kmp

```
#include <bits/stdc++.h>
using namespace std;
// pi[i] = longest proper prefix of s[0..i] which is
    alos a suffix;
// online algorithm;
vector<int> prefix_function(string const& s) {
  int n = s.length();
  vector<int> pi(n);
  for (int i = 1; i < n; ++i) {
    int j = pi[i - 1];
    while (j > 0 && s[i] != s[j]) {
      j = pi[j - 1];
    }
    if (s[i] == s[j]) j++;
    pi[i] = j;
  }
}
// Applications:
// finding occurences: Concat 's # t' and check in
    where pi[i] = |S|
// counting prefixes;
vector<int> prefix_occurences(vector<int> const& pi,
    int n) {
  vector<int> ans(n + 1);
  for (int i = 0; i < n; i++) ans[pi[i]]++;
  for (int i = n - 1; i > 0; i--) ans[pi[i - 1]] +=
      ans[i];
  for (int i = 0; i <= n; i++) ans[i]++;
  return ans;
}
// compression, if k = n - pi[n-1], divides n then 'k'
    is smallest
// unit to compress the string 's';
// aut[i][j] = automaton going from state 'i' with
    character 'j';
void compute_automaton(string s, vector<vector<int>>&
    aut) {
  s += '#';
  int n = s.size();
  vector<int> pi = prefix_function(s);
  aut.assign(n, vector<int>(26));
  for (int i = 0; i < n; i++) {
    for (int c = 0; c < 26; c++) {
      if (i > 0 && 'a' + c != s[i])
        aut[i][c] = aut[pi[i - 1]][c];
      else
        aut[i][c] = i + ('a' + c == s[i]);
    }
  }
}
```

6.2 z-algorithm

```
vector<int> z_function(string s) {
  int n = s.size();
  vector<int> z(n);
  int l = 0, r = 0;
  for (int i = 1; i < n; i++) {
    if (i < r) {
      z[i] = min(r - i, z[i - l]);
    }
    while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
      z[i]++;
    }
    if (i + z[i] > r) {
      l = i;
      r = i + z[i];
    }
  }
  return z;
}
```

7 tree

7.1 binary-lifting

```
struct binary_lift {
  vector<vector<ll>> children;
  vector<ll> depth;
  const ll LOG = 18;
  void init(vector<vector<ll>> &adj) {
    ll n = adj.size();
    depth.resize(n);
    children.assign(n, vector<ll>(LOG + 1));
    function<void(ll, ll, ll)> dfs = [&](ll u, ll p, ll
        d) {
      depth[u] = d;
      children[u][0] = p;
      for (ll i = 1; i <= LOG; ++i) {
        children[u][i] = children[children[u][i - 1]][i
            - 1];
      }
      for (ll v : adj[u]) {
        if (v != p)
          dfs(v, u, d + 1);
      }
    };
    dfs(0, 0, 0);
  }
  ll lift_node(ll n, ll d) {
    for (ll i = LOG; i >= 0; --i) {
      if (d & (1 << i))
        n = children[n][i];
    }
    return n;
  }
  ll lca(ll u, ll v) {
    if (depth[u] < depth[v])
      swap(u, v);
    u = lift_node(u, depth[u] - depth[v]);
    if (u == v)
      return u;
```

```
    for (ll i = LOG; i >= 0; --i) {
      if (children[u][i] != children[v][i]) {
        u = children[u][i];
        v = children[v][i];
      }
    }
    return children[u][0];
  }
  ll dist(ll u, ll v) { return depth[u] + depth[v] - 2
      * depth[lca(u, v)]; }
};
```

7.2 euler-tour

```
struct euler_tour {
  vector<ll> in, out;
  ll timer = 0;
  void init(vector<vector<ll>> &adj) {
    ll n = adj.size();
    in.resize(n);
    out.resize(n);
    function<void(ll, ll)> dfs = [&](ll u, ll p) {
      in[u] = timer++;
      for (ll v : adj[u]) {
        if (v != p)
          dfs(v, u);
      }
      out[u] = timer++;
    };
    dfs(0, 0);
  }
  bool is_ancestor(ll u, ll v) { return in[u] <= in[v]
      && out[u] >= out[v]; }
};
```

7.3 hld

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

const int N = 2e5 + 5;
const int D = 19;
const int S = (1 << D);

int n, q, v[N];
vector<int> adj[N];

int sz[N], p[N], dep[N];
int st[S], id[N], tp[N];

void update(int idx, int val) {
        st[idx += n] = val;
        for (idx /= 2; idx; idx /= 2) st[idx] =
            max(st[2 * idx], st[2 * idx + 1]);
}

int query(int lo, int hi) {
        int ra = 0, rb = 0;
        for (lo += n, hi += n + 1; lo < hi; lo /= 2, hi
            /= 2) {
                if (lo & 1) ra = max(ra, st[lo++]);
                if (hi & 1) rb = max(rb, st[--hi]);
        }
        return max(ra, rb);
}

int dfs_sz(int cur, int par) {
        sz[cur] = 1;
        p[cur] = par;
        for (int chi : adj[cur]) {
                if (chi == par) continue;
                dep[chi] = dep[cur] + 1;
                p[chi] = cur;
                sz[cur] += dfs_sz(chi, cur);
        }
        return sz[cur];
}

int ct = 1;

void dfs_hld(int cur, int par, int top) {
        id[cur] = ct++;
        tp[cur] = top;
        update(id[cur], v[cur]);
        int h_chi = -1, h_sz = -1;
        for (int chi : adj[cur]) {
                if (chi == par) continue;
                if (sz[chi] > h_sz) {
                        h_sz = sz[chi];
                        h_chi = chi;
                }
        }
        if (h_chi == -1) return;
        dfs_hld(h_chi, cur, top);
        for (int chi : adj[cur]) {
                if (chi == par || chi == h_chi) continue;
                dfs_hld(chi, cur, chi);
        }
}

int path(int x, int y) {
        int ret = 0;
        while (tp[x] != tp[y]) {
                if (dep[tp[x]] < dep[tp[y]]) swap(x, y);
                ret = max(ret, query(id[tp[x]], id[x]));
                x = p[tp[x]];
        }
        if (dep[x] > dep[y]) swap(x, y);
        ret = max(ret, query(id[x], id[y]));
        return ret;
}

int main() {
        scanf("%d%d", &n, &q);
        for (int i = 1; i <= n; i++) scanf("%d", &v[i]);
        for (int i = 2; i <= n; i++) {
                int a, b;
                scanf("%d%d", &a, &b);
                adj[a].push_back(b);
                adj[b].push_back(a);
        }
        dfs_sz(1, 1);
        dfs_hld(1, 1, 1);
        while (q--) {
                int t;
                scanf("%d", &t);
                if (t == 1) {
                        int s, x;
                        scanf("%d%d", &s, &x);
                        v[s] = x;
```

```
                update(id[s], v[s]);
            } else {
                int a, b;
                scanf("%d%d", &a, &b);
                int res = path(a, b);
                printf("%d ", res);
            }
        }
}
```

7.4 tree-isomorphism

```
const int N = 3e5 + 9, mod = 1e9 + 97;
template <const int32_t MOD>
struct modint {
  int32_t value;
  modint() = default;
  modint(int32_t value_) : value(value_) {}
  inline modint<MOD> operator + (modint<MOD> other)
      const { int32_t c = this->value + other.value;
      return modint<MOD>(c >= MOD ? c - MOD : c); }
  inline modint<MOD> operator - (modint<MOD> other)
      const { int32_t c = this->value - other.value;
      return modint<MOD>(c < 0 ? c + MOD : c); }
  inline modint<MOD> operator * (modint<MOD> other)
      const { int32_t c = (int64_t)this->value *
      other.value % MOD; return modint<MOD>(c < 0 ? c +
      MOD : c); }
  inline modint<MOD> & operator += (modint<MOD> other)
      { this->value += other.value; if (this->value >=
      MOD) this->value -= MOD; return *this; }
  inline modint<MOD> & operator -= (modint<MOD> other)
      { this->value -= other.value; if (this->value <
      0) this->value += MOD; return *this; }
  inline modint<MOD> & operator *= (modint<MOD> other)
      { this->value = (int64_t)this->value *
      other.value % MOD; if (this->value < 0)
      this->value += MOD; return *this; }
  inline modint<MOD> operator - () const { return
      modint<MOD>(this->value ? MOD - this->value : 0);
      }
  modint<MOD> pow(uint64_t k) const { modint<MOD> x =
      *this, y = 1; for (; k; k >>= 1) { if (k & 1) y
      *= x; x *= x; } return y; }
  modint<MOD> inv() const { return pow(MOD - 2); } //
      MOD must be a prime
  inline modint<MOD> operator / (modint<MOD> other)
      const { return *this * other.inv(); }
  inline modint<MOD> operator /= (modint<MOD> other) {
      return *this *= other.inv(); }
  inline bool operator == (modint<MOD> other) const {
      return value == other.value; }
  inline bool operator != (modint<MOD> other) const {
      return value != other.value; }
  inline bool operator < (modint<MOD> other) const {
      return value < other.value; }
  inline bool operator > (modint<MOD> other) const {
      return value > other.value; }
};
template <int32_t MOD> modint<MOD> operator * (int64_t
    value, modint<MOD> n) { return modint<MOD>(value)
    * n; }
template <int32_t MOD> modint<MOD> operator * (int32_t
    value, modint<MOD> n) { return modint<MOD>(value %
    MOD) * n; }
template <int32_t MOD> istream & operator >> (istream &
    in, modint<MOD> &n) { return in >> n.value; }
template <int32_t MOD> ostream & operator << (ostream &
    out, modint<MOD> n) { return out << n.value; }

using mint = modint<mod>;

mint pw[N];
const mint P = 998244353, Q = 1e9 + 33, R = 99999989;
const int base = 10;
struct Tree {
  int n;
  vector<vector<int>> g;
  Tree() {}
  Tree(int _n) : n(_n) {
    g.resize(n + 1);
  }
  void add_edge(int u, int v) {
    g[u].push_back(v);
    g[v].push_back(u);
  }
  vector<int> bfs(int s) {
    queue<int> q;
    vector<int> d(n + 1, n * 2);
    d[0] = -1;
    q.push(s);
    d[s] = 0;
    while(!q.empty()) {
      int u = q.front();
      q.pop();
      for(auto v : g[u]) if(d[u] + 1 < d[v]) {
          d[v] = d[u] + 1;
          q.push(v);
        }
    }
    return d;
  }
  vector<int> get_centers() {
    auto du = bfs(1);
    int v = max_element(du.begin(), du.end()) -
        du.begin();
    auto dv = bfs(v);
    int u = max_element(dv.begin(), dv.end()) -
        dv.begin();
    du = bfs(u);
    vector<int> ans;
    for(int i = 1; i <= n; i++) if(du[i] + dv[i] ==
        du[v] && du[i] >= du[v] / 2 && dv[i] >= du[v] /
        2) {
        ans.push_back(i);
      }
    return ans;
  }
  mint yo(int u, int pre = 0) {
    vector<mint> nw;
    for(auto v : g[u]) if(v != pre) nw.push_back(yo(v,
        u));
    mint ans = 0;
    for(auto x : nw) {
      ans = ans + P.pow(x.value);
    }
    ans = ans * Q + R;
    return ans;
  }
  bool iso(Tree &t) {
    auto a = get_centers();
    auto b = t.get_centers();
```

```
    for(auto x : a) for(auto y : b) if(yo(x) ==
        t.yo(y)) return 1;
    return 0;
  }
};
```

7.5 tree-lifing

```
/*
 * Does all the binary lifting tasks in the same
 * time complexity but uses only O(n) memory;
 */
struct tree_lifting {
  vector<int> dep, jmp, fa;
  int n;
  void add_leaf(int cur, int par) {
    fa[cur] = par;
    dep[cur] = 1 + dep[par];
    if (dep[par] - dep[jmp[par]] == dep[jmp[par]] -
        dep[jmp[jmp[par]]]) {
      jmp[cur] = jmp[jmp[par]];
    } else {
      jmp[cur] = par;
    }
  }
  void dfs(int cur, int par, vector<vector<int>> &adj) {
    add_leaf(cur, par);
    for (int it : adj[cur]) {
      if (it == par)
        continue;
      dfs(it, cur, adj);
    }
  }
  void init(int nn) {
    dep.resize(nn), jmp.resize(nn), fa.resize(nn);
    this->n = nn;
  }
  int lift(int cur, int k) {
    int new_depth = max(dep[cur] - k, 0);
    while (dep[cur] > new_depth) {
      if (dep[jmp[cur]] >= new_depth)
        cur = jmp[cur];
      else
        cur = fa[cur];
    }
    return cur;
  }
  int lca(int u, int v) {
    if (dep[u] > dep[v])
      swap(u, v);
    v = lift(v, dep[v] - dep[u]);
    while (u != v) {
      if (jmp[v] != jmp[u]) {
        u = jmp[u];
        v = jmp[v];
      } else {
        u = fa[u];
        v = fa[v];
      }
    }
    return u;
  }
  int dist(int u, int v) { return dep[u] + dep[v] - 2 *
      dep[lca(u, v)]; }
};
```