# Team Notebook

HUS.NgoLeHoang

December 7, 2022

## Contents

# 1 DataStructures

## 1.1 BinaryTrie

```
// Test in CSES/Additional Problems/Maximum Xor Subarray.cpp
// Binary Trie
// Based on https://judge.yosupo.jp/submission/72657
// Supports:
// - get min / max / kth element
// - given K, find x: x^K is min / max / kth
//
// Notes:
// - high mem usage. If just need kth_element
//   -> use OrderedSet.h if MAX_VALUE is ~10^6
//   -> use STL/order_statistic.cpp if MAX_VALUE is big /
     custom type
//
// Tested:
// - (insert, remove, min xor) https://judge.yosupo.jp/
     problem/set_xor_min
// - (insert, max xor) https://cses.fi/problemset/task/1655/
template <class Val = long long,    // Value store in trie
          class Count = long long,   // frequency of values
          int B = (sizeof(Val) * 8 - 2) // max number of bits
          >
BinaryTrie {
  struct Node {
    std::array<int, 2> child;
    Count count;
    Node() : child{-1, -1}, count(0) {}
  };
  BinariesTrie() : nodes{Node()} {} // create root node

  // Number of elements in the trie
  Count size() { return nodes[0].count; }

  void insert(Val x, Count cnt = 1) { update(x, cnt); }
  void insert(Val x, Count cnt = -1) { update(x, cnt); }

  // return X: X ^ xor_val is minimum
  pair<Val, Node> min_element(Val xor_val = 0) {
    assert(0 < size());
    return kth_element(0, xor_val);
  }

  pair<Val, Node> max_element(Val xor_val = 0) {
    assert(0 < size());
    return kth_element(size() - 1, xor_val);
  }

  pair<Val, Node> kth_element(Count k, Val xor_val = 0) {
    assert(0 <= k && k < size());
    int u = 0;
    Val x = 0;
    for (int i = B - 1; i >= 0; --i) {
      int b = get_bit(x, i);
      int v0 = get_child(u, b);
      if (nodes[v0].count <= k) {
        k -= nodes[v0].count;
        u = get_child(u, 1 - b);
        x |= 1LL << i;
      } else {
        u = v0;
      }
    }
    return {x, nodes[u]};
  }
  // return frequency of X
  Count count(Val x) {
    int u = 0;
    for (int i = B - 1; i >= 0; --i) {
      int b = get_bit(x, i);
      if (nodes[u].child[b] == -1)
        return 0;
      u = get_child(u, b);
    }
    return nodes[u].count;
  }
  // private:
  vector<Node> nodes;
  int get_child(int p, int b) {
    assert(0 <= p && p < nodes.size());
    assert(0 <= b && b < 2);
    if (nodes[p].child[b] == -1) {
      nodes[p].child[b] = nodes.size();
      nodes.push_back({Node()});
    }
    return nodes[p].child[b];
  }
  void update(Val x, Count cnt) {
    int u = 0;
    for (int i = B - 1; i >= 0; --i) {
      nodes[u].count += cnt;
      assert(nodes[u].count >= 0); // prevent over delete;
      int b = get_bit(x, i);
      int u = get_child(u, b);
    }
    nodes[u].count++;
    assert(nodes[u].count >= 0); // prevent over delete.
  }
  inline int get_bit(Val v, int bit) { return (v >> bit) &
      1; }
};
```

## 1.2 DisjointSet

```
// DisjointSet {{{
#include<iostream>
using namespace std;
struct DSU {
    vector<int> lab;

    DSU(int n) : lab(n+1, -1) {}

    int getRoot(int u) {
        if (lab[u] < 0) return u;
        return lab[u] = getRoot(lab[u]);
    }

    bool merge(int u, int v) {
        u = getRoot(u); v = getRoot(v);
        if (u == v) return false;
        if (lab[u] > lab[v]) swap(u, v);
        lab[u] += lab[v];
        lab[v] = u;
        return true;
    }

    bool same_component(int u, int v) {
        return getRoot(u) == getRoot(v);
    }

    int component_size(int u) {
        return -lab[getRoot(u)];
    }
};
// }}}
int main() {
    int u, v;
    DSU g;
    g.merge(u, v); // Add u to v.
    g.component_size(u); // return size of component which u
        in it.
    g.getRoot(u); // ok
}
```

## 1.3 PersistentSegmentTree

```
#include <stdio.h>
#include <iostream>
#include <algorithm>
using namespace std;

#define long long long
#define f1(i,n) for (int i=1; i<=n; i++)
#define f0(i,n) for (int i=0; i<n; i++)

#define N 100005
int m, n, a[N], l[N], Root[N], Peak=0;
int Sum[80*N], Left[80*N], Right[80*N]; // (n*4)+(n log n)

int create(int n){
    if (n==1) { ++Peak; Sum[Peak]=0; return Peak; }
    int u = ++Peak;
    Left[u]=create(n-n/2);
    Right[u]=create(n/2);
    return u;
}

struct node {
    int ll, rr, id;

    node(int L, int R, int X)
    { ll=L, rr=R, id=X; }
    node left()
    { return node(ll, (ll+rr)/2, Left[id]); }
    node right()
    { return node((ll+rr)/2+1, rr, Right[id]); }

    int update(int U, int X){
        if (ll>U || U>rr) return id;
        if (ll==rr) { Sum[++Peak]=X; return Peak; }
        int u = ++Peak;
        Left[u] = left().update(U, X);
        Right[u] = right().update(U, X);
        Sum[u]=Sum[Left[u]]+Sum[Right[u]];
        return u;
    }
    int sum_range(int L, int R){
        if (L>rr || ll>R || L>R) return 0;
        if (L<=ll && rr<=R) return Sum[id];
        int Sum1 = left().sum_range(L, R);
        int Sum2 = right().sum_range(L, R);
        return Sum1 + Sum2;
    }
};

bool as_a(int x, int y)
    { return a[x]<a[y]; }

main(){
    scanf("%d%d", &n, &m);
    f1(i,n) scanf("%d", &a[i]);
//    f1(i,n) printf("%d ", a[i]=rand()%100); printf("\n");

    f1(i,n) l[i]=i;
    sort(l+1, l+n+1, as_a);
    Root[0]=create(n);
    f1(i,n) {
        Root[i]=node(1, n, Root[i-1]).update(l[i], 1);
//        cout << endl << Peak << " " << 80*N << endl;
    }

    f1(i,m) {
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);
        int ll=1, rr=n, mm=(ll+rr)/2;
        while (ll!=rr){
            if (node(1, n, Root[mm]).sum_range(x, y)>=z) rr=
                mm; else ll=mm+1;
            mm=(ll+rr)/2;
        }
        printf("%d\n", a[l[mm]]);
    }
}
```

## 1.4 RMQ

```
// RMQ {{{
//
// Sparse table
// Usage:
// RMQ<int, _min> st(v);
//
// Note:
// - doesn't work for empty range
//
// Tested:
// - https://judge.yosupo.jp/problem/staticrmq

#include <vector>
using namespace std;
template <class T, T *(op)(T, T)> struct RMQ {
  RMQ() = default;
  RMQ(const vector<int> &v) : t{v}, n{(int)v.size()} {
    for (int k = 1; (1 << k) <= n; ++k) {
      t.emplace_back(n - (1 << k) + 1);
      for (int i = 0; i + (1 << k) <= n; ++i) {
        t[k][i] = op(t[k - 1][i], t[k - 1][i + (1 << (k - 1))
            ]);
      }
    }
  }
  // get range [l, r-1]
  // doesn't work for empty range
  T get(int l, int r) const {
    assert(0 <= l && l < r && r <= n);
    int k = __lg(r - l);
    return op(t[k][r - l], t[k][r - (1 << k)]);
  }

private:
  vector<vector<T>> t;
  int n;
};
template <class T> T _min(T a, T b) { return a < b ? a : b;
    }

template <class T> T _max(T a, T b) { return a > b ? a : b;
    }
```

## 1.5 SegmentTreeBeats

```
// Segment tree beats
// Tutorial:
// - https://codeforces.com/blog/entry/57319
// - https://www.youtube.com/watch?v=wFqKgrW1IMQ
//
// AC: https://www.acmicpc.net/problem/17474

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

#define int long long
#define FOR(i, a, b) for (int i = (a), _##i = (b); i <= _##i
    ; ++i)
#define REP(i, a) for (int i = 0, _##i = (a); i < _##i; ++i)

struct Node {
    int max1; // max value
    int max2; // 2nd max value (must be different from max1)
    int cnt_max; // how many indices have value == max1
    int sum;
    int lazy;
```

```
    Node() {}
    Node(int val) { // initialize with a single number.
        max1 = val;
        max2 = -1; // Note that values are in [0, 2^31), so
             -1 works here.
        cnt_max = 1;
        sum = val;
        lazy = -1; // Note that values are in [0, 2^31), so
             -1 works here.
    }

    void setMin(int val) { // for each i, set a[i] = min(a[i
         ], val)
        assert(val > max2);

        if (max1 <= val) return;

        // Sample: 1 3 5 8 8 --> 1 3 5 6 6
        sum -= (max1 - val) * cnt_max;
        lazy = val;
        max1 = val;
    }
} it[8000111];

Node operator + (const Node& a, const Node& b) {
    Node res;
    res.max1 = max(a.max1, b.max1);

    res.max2 = max(a.max2, b.max2);
    if (a.max1 != res.max1) res.max2 = max(res.max2, a.max1);
    if (b.max1 != res.max1) res.max2 = max(res.max2, b.max1);

    res.cnt_max = 0;
    if (a.max1 == res.max1) res.cnt_max += a.cnt_max;
    if (b.max1 == res.max1) res.cnt_max += b.cnt_max;

    res.sum = a.sum + b.sum;
    res.lazy = -1;
    return res;
}

void down(int i) {
    if (it[i].lazy < 0) return;

    it[i*2].setMin(it[i].lazy);
    it[i*2+1].setMin(it[i].lazy);

    it[i].lazy = -1;
}
```

```
int a[1000111];
void build(int i, int l, int r) {
    if (l == r) {
        it[i] = Node(a[l]);
        return;
    }
    int mid = (l + r) / 2;
    build(i*2, l, mid);
    build(i*2 + 1, mid + 1, r);

    it[i] = it[i*2] + it[i*2 + 1];
}

void setMin(int i, int l, int r, int u, int v, int x) {
    if (v < l || r < u) return;
    if (it[i].max1 <= x) return;
    // now max1 > x

    if (u <= l && r <= v && it[i].max2 < x) {
        it[i].setMin(x);
        return;
    }

    down(i);
    int mid = (l + r) / 2;
    setMin(i*2, l, mid, u, v, x);
    setMin(i*2 + 1, mid+1, r, u, v, x);
    it[i] = it[i*2] + it[i*2 + 1];
}

int getMax(int i, int l, int r, int u, int v) {
    if (v < l || r < u) return -1;
    if (u <= l && r <= v) return it[i].max1;

    down(i);
    int mid = (l + r) / 2;
    return max(getMax(i*2, l, mid, u, v),
           getMax(i*2+1, mid+1, r, u, v));
}

int getSum(int i, int l, int r, int u, int v) {
    if (v < l || r < u) return 0;
    if (u <= l && r <= v) return it[i].sum;

    down(i);
    int mid = (l + r) / 2;
    return getSum(i*2, l, mid, u, v) + getSum(i*2+1, mid+1, r
         , u, v);
}
```

```
int32_t main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    // read initial array
    int n; cin >> n;
    FOR(i,1,n) cin >> a[i];

    // initialize segment tree beats
    build(1, 1, n);

    // queries
    int q; cin >> q;
    while (q--) {
        int typ; cin >> typ;
        if (typ == 1) { // for each i in [l, r] set a[i] =
             min(a[i], x)
            int l, r, x; cin >> l >> r >> x;
            setMin(1, 1, n, l, r, x);
        } else if (typ == 2) { // find max(a[i]) for i in [l,
              r]
            int l, r; cin >> l >> r;
            cout << getMax(1, 1, n, l, r) << '\n';
        } else { // find sum(a[i]) for i in [l, r]
            int l, r; cin >> l >> r;
            cout << getSum(1, 1, n, l, r) << '\n';
        }
    }
    return 0;
}
```

## 1.6 WalevetMatrix

```
// WaveletMatrix {{{
// Copied from https://github.com/dacin21/dacin21_codebook/
     blob/master/trees/wavelet_matrix.cpp
//
// Notes:
// - Index from 0
// - k (for k-th query) from 0
// - Need to remove #define int long long
//
// Tested:
// - (kth query) https://judge.yosupo.jp/problem/
     range_kth_smallest
// - (range_count) https://judge.yosupo.jp/problem/
     static_range_frequency

// Bit Presum {{{
```

```
class Bit_Presum {
public:
    static constexpr uint32_t omega = CHAR_BIT * sizeof(
        uint64_t);
    static constexpr uint32_t lg_omega = __lg(omega);
    static_assert(omega == 64u);

    Bit_Presum(vector<uint64_t> mask_)
            : n(mask_.size()), mask(move(mask_)), presum(n+1)
                {
        build();
    }
    Bit_Presum(uint32_t bits, bool init_val = 0)
            : n((bits>>lg_omega) + 1),
              mask(n, init_val ? ~uint64_t{0} : uint64_t{0}),
              presum(n+1) {
        if (init_val) mask.back()<<=((n<<lg_omega) - bits);
        build();
    }
    // popcount l <= i < r
    uint32_t query(uint32_t l, uint32_t r) const {
        if (__builtin_expect(r < l, false)) return 0;
        return query(r) - query(l);
    }
    // popcount 0 <= i < x
    uint32_t query(uint32_t x) const {
        uint32_t high = x>>lg_omega, low = x & ((uint64_t
            {1}<<lg_omega) - 1);
        uint32_t ret = presum_query(high);
        ret += __builtin_popcountll(mask[high]& ((uint64_t{1}
             << low)-1));
        return ret;
    }

    void update_pre_build(uint32_t x, bool val) {
        uint32_t high = x>>lg_omega, low = x & ((1u<<lg_omega
            ) - 1);
        mask[high] = (mask[high] & ~(uint64_t{1} << low)) | (
            uint64_t{val}<<low);
    }
    void do_build() {
        build();
    }

    friend ostream& operator<<(ostream&o, Bit_Presum const&b)
         {
        for (auto const& e : b.mask) {
            stringstream ss;
            ss << bitset<omega>(e);
            auto s = ss.str();
            reverse(s.begin(), s.end());
            o << s << "|";
        }
        o << " : ";
        for (auto const&e:b.presum) o << e << " ";
        o << "\n";
        return o;
    }

private:
    void presum_build() {
        for (uint32_t x = 1; x <= n; ++x) {
            presum[x] += presum[x-1];
        }
    }
    // sum 0 <= i < x
    uint32_t presum_query(uint32_t x) const {
        return presum[x];
    }
    void build() {
        for (uint32_t x = 0; x < n; ++x) {
            presum[x+1] = __builtin_popcountll(mask[x]);
        }
        presum_build();
    }

    const uint32_t n;
    vector<uint64_t> mask;
    vector<uint32_t> presum;
};
// }}}

template<typename T, typename Bit_Ds = Bit_Presum>
class WaveletMatrix {
public:
    static_assert(is_integral<T>::value);
    static constexpr uint32_t height = CHAR_BIT * sizeof(T);

    WaveletMatrix(vector<T> v): n(v.size()), data(height, n)
         {
        build(move(v));
    }
    // count l <= i < r s.t. A <= val[i] < B
    uint32_t range_count(int l, int r, T A, T B) const {
        assert(0 <= l && r <= n);
        return count_lower(l, r, B) - count_lower(l, r, A);
    }
    // count l <= i < r s.t. A <= val[i]
    uint32_t range_count_up(int l, int r, T A) const {
        assert(0 <= l && r <= n);
        if (__builtin_expect(l>r, false)) return uint32_t{0};
        return (r-l) - count_lower(l, r, A);
    }
    // k from 0
    // range: [l, r-1]
    T k_th(int l, int r, int k) const {
        assert(0 <= k && k < n);
        return get_kth(l, r, k);
    }

    // internal functions {{{
private:
    void build(vector<T> v) {
        m_index.resize(height);
        T const a = numeric_limits<T>::min();
        for (int h = height-1; h>=0;--h) {
            T const b = a + (T{1}<<(max(0, h-1))) - !h + (T
                {1}<<(max(0, h-1)));
            for (int i=0;i<n;++i) {
                data[h].update_pre_build(i, v[i]<b);
            }
            data[h].do_build();
            const int m = stable_partition(v.begin(), v.end()
                , [&b](T const&x) {return x < b;}) - v.begin
                ();
            for (int i=m;i<n;++i) {
                v[i] = v[i] - (T{1}<<(max(0, h-1))) + !h - (T
                    {1}<<(max(0, h-1)));
            }
            m_index[h] = m;
        }
    }
    /// count l <= i < r s.t. val[i] < B
    uint32_t count_lower(int l, int r, T const&B) const {
        assert(0 <= l && r <= n);
        if (__builtin_expect(r<l, false)) return 0;
        uint32_t ret = 0;
        int h = height;
        T a = numeric_limits<T>::min();
        while(h > 0) {
            --h;
  bool go_left = B < a + (T{1}<<(max(0, h-1))) - !h + (T
        {1}<<(max(0, h-1)));
            const int low_l = data[h].query(l), low_r = data[
                h].query(r);
            if (go_left) {
                l = low_l;
                r = low_r;
            } else {
```

```
            a = a + (T{1}<<(max(0, h-1))) - !h + (T{1}<<(
                max(0, h-1)));
            ret+= low_r-low_l;
            l = m_index[h] + l-low_l;
            r = m_index[h] + r-low_r;
        }
    }
    return ret;
}
T get_kth(int l, int r, int k) const {
    assert(0 <= l && r <= n);
    assert(0 <= k && k < r-l);
    int h = height;
    T a = numeric_limits<T>::min();
    while (h > 0) {
        --h;
        const int low_l = data[h].query(l), low_r = data[
            h].query(r), low_lr = low_r-low_l;
        bool go_left = k < low_lr;
        if (go_left) {
            l = low_l;
            r = low_r;
        } else {
            a+= T{1}<<h;
            k-= low_lr;
            l = m_index[h] + l-low_l;
            r = m_index[h] + r-low_r;
        }
    }
    return a;
}

const int n;
vector<int> m_index;
vector<Bit_Ds> data;
// }}}
};
// }}}
```

## 1.7 fenwickTree(BIT)

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

const int N = 1e6 + 5;
const int LOGN = log(N) + 1;
int bit[N];
int a[N];
int n; // n is size of array.

void initialize() { // create bit in O(N)
    for(int i = 1 ; i <= n; ++i) {
        bit[i] += a[i];
        if (i + (i&-i) <= n) bit[i+(i&-i)] += bit[i];
    }
}

void update(int i, int val) {
    for(; i <= n; i += i&(-i))
        bit[i] += val;
}

int get(int i) {
    int res = 0;
    for(; i > 0; i -= i&(-i))
        res += bit[i];
    return res;
}

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

int bit_search(int v) {
    int sum = 0;
    int pos = 0;
    for(int i = LOGN; i >= 0; --i) {
        if (pos + (1<<i) < N && sum + bit[pos + (1<<i)] < v)
            {
            pos += 1<<i;
            sum += bit[pos];
        }
    }
    return pos + 1;
    // +1 because 'pos' will have position of largest value
        less than 'v'
}

int main() {

}
```

## 1.8 fullHash

## 1.9 lazySegtree

```
// Lazy Segment Tree, copied from AtCoder {{{
// Source: https://github.com/atcoder/ac-library/blob/master
    /atcoder/lazysegtree.hpp
// Doc: https://atcoder.github.io/ac-library/master/
    document_en/lazysegtree.html
//
// Notes:
// - Index of elements from 0
// - Range queries are [l, r-1]
// - composition(f, g) should return f(g())
//
// Tested:
// - https://oj.vnoi.info/problem/qmax2
// - https://oj.vnoi.info/problem/lites
// - (range set, add, mult, sum) https://oj.vnoi.info/
    problem/segtree_itmix
// - (range add (i-L)*A + B, sum) https://oj.vnoi.info/
    problem/segtree_itladder
// - https://atcoder.jp/contests/practice2/tasks/practice2_l
// - https://judge.yosupo.jp/problem/range_affine_range_sum

int ceil_pow2(int n) {
    int x = 0;
    while ((1U << x) < (unsigned int)(n)) x++;
    return x;
}
template<
    class S,                // node data type
    S (*op) (S, S),         // combine 2 nodes
    S (*e) (),              // identity element
    class F,                // lazy propagation tag
    S (*mapping) (F, S),    // apply tag F on a node
    F (*composition) (F, F), // combine 2 tags
    F (*id)()               // identity tag
>
struct LazySegTree {
    LazySegTree() : LazySegTree(0) {}
    explicit LazySegTree(int n) : LazySegTree(vector<S>(n, e
        ())) {}
    explicit LazySegTree(const vector<S>& v) : _n((int) v.
        size()) {
        log = ceil_pow2(_n);
        size = 1 << log;
        d = std::vector<S>(2 * size, e());
        lz = std::vector<F>(size, id());
        for (int i = 0; i < _n; i++) d[size + i] = v[i];
        for (int i = size - 1; i >= 1; i--) {
            update(i);
```

```
    }
}

// 0 <= p < n
void set(int p, S x) {
    assert(0 <= p && p < _n);
    p += size;
    for (int i = log; i >= 1; i--) push(p >> i);
    d[p] = x;
    for (int i = 1; i <= log; i++) update(p >> i);
}

// 0 <= p < n
S get(int p) {
    assert(0 <= p && p < _n);
    p += size;
    for (int i = log; i >= 1; i--) push(p >> i);
    return d[p];
}

// Get product in range [l, r-1]
// 0 <= l <= r <= n
// For empty segment (l == r) -> return e()
S prod(int l, int r) {
    assert(0 <= l && l <= r && r <= _n);
    if (l == r) return e();

    l += size;
    r += size;

    for (int i = log; i >= 1; i--) {
        if (((l >> i) << i) != l) push(l >> i);
        if (((r >> i) << i) != r) push((r - 1) >> i);
    }

    S sml = e(), smr = e();
    while (l < r) {
        if (l & 1) sml = op(sml, d[l++]);
        if (r & 1) smr = op(d[--r], smr);
        l >>= 1;
        r >>= 1;
    }

    return op(sml, smr);
}

S all_prod() {
    return d[1];
}
```

```
// 0 <= p < n
void apply(int p, F f) {
    assert(0 <= p && p < _n);
    p += size;
    for (int i = log; i >= 1; i--) push(p >> i);
    d[p] = mapping(f, d[p]);
    for (int i = 1; i <= log; i++) update(p >> i);
}

// Apply f on all elements in range [l, r-1]
// 0 <= l <= r <= n
void apply(int l, int r, F f) {
    assert(0 <= l && l <= r && r <= _n);
    if (l == r) return;

    l += size;
    r += size;

    for (int i = log; i >= 1; i--) {
        if (((l >> i) << i) != l) push(l >> i);
        if (((r >> i) << i) != r) push((r - 1) >> i);
    }

    {
        int l2 = l, r2 = r;
        while (l < r) {
            if (l & 1) all_apply(l++, f);
            if (r & 1) all_apply(--r, f);
            l >>= 1;
            r >>= 1;
        }
        l = l2;
        r = r2;
    }

    for (int i = 1; i <= log; i++) {
        if (((l >> i) << i) != l) update(l >> i);
        if (((r >> i) << i) != r) update((r - 1) >> i);
    }
}

// Binary search on SegTree to find largest r:
//   f(op(a[l] .. a[r-1])) = true (assuming empty array
    is always true)
//   f(op(a[l] .. a[r])) = false (assuming op(..., a[n])
    , which is out of bound, is always false)
template <bool (*g)(S)> int max_right(int l) {
    return max_right(l, [](S x) { return g(x); });
}
template <class G> int max_right(int l, G g) {
```

```
    assert(0 <= l && l <= _n);
    assert(g(e()));
    if (l == _n) return _n;
    l += size;
    for (int i = log; i >= 1; i--) push(l >> i);
    S sm = e();
    do {
        while (l % 2 == 0) l >>= 1;
        if (!g(op(sm, d[l]))) {
            while (l < size) {
                push(l);
                l = (2 * l);
                if (g(op(sm, d[l]))) {
                    sm = op(sm, d[l]);
                    l++;
                }
            }
            return l - size;
        }
        sm = op(sm, d[l]);
        l++;
    } while ((l & -l) != l);
    return _n;
}

// Binary search on SegTree to find smallest l:
//   f(op(a[l] .. a[r-1])) = true  (assuming empty array
     is always true)
//   f(op(a[l-1] .. a[r-1])) = false (assuming op(a[-1],
     ..), which is out of bound, is always false)
template <bool (*g)(S)> int min_left(int r) {
    return min_left(r, [](S x) { return g(x); });
}
template <class G> int min_left(int r, G g) {
    assert(0 <= r && r <= _n);
    assert(g(e()));
    if (r == 0) return 0;
    r += size;
    for (int i = log; i >= 1; i--) push((r - 1) >> i);
    S sm = e();
    do {
        r--;
        while (r > 1 && (r % 2)) r >>= 1;
        if (!g(op(d[r], sm))) {
            while (r < size) {
                push(r);
                r = (2 * r + 1);
                if (g(op(d[r], sm))) {
                    sm = op(d[r], sm);
                    r--;
```

```
                }
            }
            return r + 1 - size;
        }
        sm = op(d[r], sm);
    } while ((r & -r) != r);
    return 0;
}


private:
    int _n, size, log;
    vector<S> d;
    vector<F> lz;

    void update(int k) {
        d[k] = op(d[2*k], d[2*k+1]);
    }
    void all_apply(int k, F f) {
        d[k] = mapping(f, d[k]);
        if (k < size) lz[k] = composition(f, lz[k]);
    }
    void push(int k) {
        all_apply(2*k, lz[k]);
        all_apply(2*k+1, lz[k]);
        lz[k] = id();
    }
};
// }}}

// Examples {{{
// https://onlinejudge.u-aizu.ac.jp/courses/library/3/DSL/2/
    DSL_2_D
// https://onlinejudge.u-aizu.ac.jp/courses/library/3/DSL/2/
    DSL_2_E
// https://onlinejudge.u-aizu.ac.jp/courses/library/3/DSL/2/
    DSL_2_F
// https://onlinejudge.u-aizu.ac.jp/courses/library/3/DSL/2/
    DSL_2_G
// https://onlinejudge.u-aizu.ac.jp/courses/library/3/DSL/2/
    DSL_2_H
// https://onlinejudge.u-aizu.ac.jp/courses/library/3/DSL/2/
    DSL_2_I
// supports:
// - set a(l -> r) to val; val > NOT_SET
// - add a(l -> r) += val
// - find sum a(l -> r)
// - find min a(l -> r)
struct RangeSetAddMinSumOps {
    struct S { long long sum, min, sz; };
    static S op(S l, S r) { return S { l.sum + r.sum, min(l.
        min, r.min), l.sz + r.sz }; }
    static S e() { return S {0LL, INT_MAX, 0}; }

    static const long long NOT_SET = -1000111000;
    struct F { long long set, add; };

    static S mapping(F f, S s) {
        if (f.set == NOT_SET) {
            return S {
                s.sum + f.add * s.sz,
                s.min + f.add,
                s.sz,
            };
        }
        return S {
            (f.set + f.add) * s.sz,
            f.set + f.add,
            s.sz,
        };
    }
    static F composition(F f, F g) {
        if (f.set == NOT_SET) {
            return F { g.set, g.add + f.add };
        }
        return f;
    }
    static F id() {
        return F { NOT_SET, 0 };
    }
};
// }}}
```

## 1.10 lca-rmq

```
// L[i] = level
// L[root] = -1
// LCA[0][root] = -1
const int MN = 100111;
int V, LCA[22][MN], L[MN];
long long Rmax[22][MN];
#define LL long long
void initLCA () {
    FOR (lg, 1, 19) {
        REP (i, V) {
            if (LCA[lg - 1][i] == -1) continue;
            LCA[lg][i] = LCA[lg - 1][LCA[lg - 1][i]];
            Rmax[lg][i] = max (Rmax[lg - 1][LCA[lg - 1][i]],
                Rmax[lg - 1][i]);
        }
    }
}

LL query (LL a, LL b) {
    LL ret = 0;
    if (L[a] < L[b]) swap (a, b);

    FORD(lg,19,0) {
        if (LCA[lg][a] != -1 && L[LCA[lg][a]] >= L[b]) {
            ret = max (ret, Rmax[lg][a]);
            a = LCA[lg][a];
        }
    }
    if (a == b) return ret; // if LCA, return a
    FORD(lg,19,0) {
        if (LCA[lg][a] != LCA[lg][b]) {
            ret = max (ret, Rmax[lg][a]);
            ret = max (ret, Rmax[lg][b]);
            a = LCA[lg][a];
            b = LCA[lg][b];
        }
    }
    ret = max (ret, Rmax[0][a]);
    ret = max (ret, Rmax[0][b]);
    return ret; // if LCA, return LCA[0][a]
}
```

## 1.11 orderedSet

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

## 1.12 pbds-faster-map

```
// From https://codeforces.com/blog/entry/60737

// Code copied from https://codeforces.com/contest/1006/
    submission/41804666
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

unsigned hash_f(unsigned x) {
```

```
    x = ((x >> 16) ^ x) * 0x45d9f3b;
    x = ((x >> 16) ^ x) * 0x45d9f3b;
    x = (x >> 16) ^ x;
    return x;
}
struct chash {
    int operator()(int x) const { return hash_f(x); }
};

gp_hash_table<int, int, chash> mp;


// alternative hash function:
// Code copied from https://ideone.com/LhpILA
const ll TIME = chrono::high_resolution_clock::now().
     time_since_epoch().count();
const ll SEED = (ll)(new ll);
const ll RANDOM = TIME ^ SEED;
const ll MOD = (int)1e9+7;
const ll MUL = (int)1e6+3;

struct chash{
    ll operator()(ll x) const { return std::hash<ll>{}((x ^
         RANDOM) % MOD * MUL); }
};
```

## 1.13 segmentTree-fast

```
#include<cstdio>
const int N = 1e5; // limit for array size
int n; // array size
int t[2 * N];

void build() { // build the tree
  for (int i = n - 1; i > 0; --i) t[i] = t[i<<1] + t[i
       <<1|1];
}

void modify(int p, int value) { // set value at position p
  for (t[p += n] = value; p > 1; p >>= 1) t[p>>1] = t[p] + t
       [p^1];
}

int query(int l, int r) { // sum on interval [l, r)
  int res = 0;
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l&1) res += t[l++];
    if (r&1) res += t[--r];
  }
  return res;
}

int main() {
  scanf("%d", &n);
  for (int i = 0; i < n; ++i) scanf("%d", t + n + i);
  build();
  modify(0, 1);
  printf("%d\n", query(3, 11));
  return 0;
}
```

## 1.14 segmentTree-merge

```
// CPP program to implement k-th order statistics
#include <bits/stdc++.h>
using namespace std;

const int MAX = 4e5 + 5; // max size of segtree = 4*N

// Constructs a segment tree and stores tree[]
void buildTree(int treeIndex, int l, int r, vector<pair<int,
     int>> &a,
               vector<int> tree[]) {

  /* l => start of range,
        r => ending of a range
        treeIndex => index in the Segment Tree/Merge
                                 Sort Tree */

  /* leaf node */
  if (l == r) {
    tree[treeIndex].push_back(a[l].second);
    return;
  }

  int mid = (l + r) / 2;

  /* building left subtree */
  buildTree(2 * treeIndex, l, mid, a, tree);

  /* building left subtree */
  buildTree(2 * treeIndex + 1, mid + 1, r, a, tree);

  /* merging left and right child in sorted order */
  merge(tree[2 * treeIndex].begin(), tree[2 * treeIndex].end
       (),
        tree[2 * treeIndex + 1].begin(), tree[2 * treeIndex +
             1].end(),
        back_inserter(tree[treeIndex]));
}

// Returns the Kth smallest number in query range
int queryRec(int segmentStart, int segmentEnd, int
     queryStart, int queryEnd,
             int treeIndex, int K, vector<int> tree[]) {
  /*
         segmentStart => start of a Segment,
         segmentEnd => ending of a Segment,
         queryStart => start of a query range,
         queryEnd => ending of a query range,
         treeIndex => index in the Segment
                                      Tree/Merge Sort Tree,
         K => kth smallest number to find */

  if (segmentStart == segmentEnd)
    return tree[treeIndex][0];

  int mid = (segmentStart + segmentEnd) / 2;

  // finds the last index in the segment
  // which is <= queryEnd
  int last_in_query_range = (upper_bound(tree[2 * treeIndex
       ].begin(),
                                         tree[2 * treeIndex].end
                                              (), queryEnd) -
                             tree[2 * treeIndex].begin());

  // finds the first index in the segment
  // which is >= queryStart
  int first_in_query_range =
      (lower_bound(tree[2 * treeIndex].begin(), tree[2 *
           treeIndex].end(),
                   queryStart) -
       tree[2 * treeIndex].begin());

  int M = last_in_query_range - first_in_query_range;

  if (M >= K) {

    // Kth smallest is in left subtree,
    // so recursively call left subtree for Kth
    // smallest number
    return queryRec(segmentStart, mid, queryStart, queryEnd,
         2 * treeIndex, K,
                    tree);
  }

  else {
```

```
    // Kth smallest is in right subtree,
    // so recursively call right subtree for the
    // (K-M)th smallest number
    return queryRec(mid + 1, segmentEnd, queryStart, queryEnd
        ,
                    2 * treeIndex + 1, K - M, tree);
  }
}

// A wrapper over query()
int query(int queryStart, int queryEnd, int K, int n, vector
     <pair<int, int>> &a,
         vector<int> tree[]) {

  return queryRec(0, n - 1, queryStart - 1, queryEnd - 1, 1,
       K, tree);
}

// Driver code
int main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  cout.tie(0);
  int n, q;
  cin >> n >> q;
  int arr[n];
  for (int i = 0; i < n; ++i)
    cin >> arr[i];
  // vector of pairs of form {element, index}
  vector<pair<int, int>> v;
  for (int i = 0; i < n; i++) {
    v.push_back(make_pair(arr[i], i));
  }

  // sort the vector
  sort(v.begin(), v.end());

  // Construct segment tree in tree[]
  vector<int> tree[MAX];
  buildTree(1, 0, n - 1, v, tree);

  // Answer queries
  // kSmallestIndex hold the index of the kth smallest
       number
  for (int i = 0; i < q; ++i) {
    int l, r;
    cin >> l >> r;
    int mid = (l + r) / 2 - l + 1;
    int median = query(l, r, mid, n, v, tree);
    cout << arr[median] << endl;
  }

  return 0;
}
```

## 1.15 segmentTree2D

```
#include <stdio.h>
#include <iostream>
#include <algorithm>
using namespace std;

int Max[4096][4096];

struct dir {
    int ll, rr, id;
    dir (int L, int R, int X)
        { ll=L, rr=R, id=X; }
    dir left() const
        { return dir(ll, (ll+rr)/2, id*2); }
    dir right() const
        { return dir((ll+rr)/2+1, rr, id*2+1); }
    inline bool irrelevant(int L, int R) const
        { return ll>R || L>rr || L>R; }
};

void maximize(int &a, int b)
    { a=max(a, b); }

void maximize(const dir &dx, const dir &dy, int x, int y,
     int k, bool only_y) {
    if (dx.irrelevant(x, x) || dy.irrelevant(y, y)) return;
    maximize(Max[dx.id][dy.id], k);
    if (!only_y && dx.ll != dx.rr) {
        maximize(dx.left(), dy, x, y, k, false);
        maximize(dx.right(), dy, x, y, k, false);
    }
    if (dy.ll != dy.rr) {
        maximize(dx, dy.left(), x, y, k, true);
        maximize(dx, dy.right(), x, y, k, true);
    }
}

int max_range(const dir &dx, const dir &dy, int lx, int rx,
     int ly, int ry) {
    if (dx.irrelevant(lx, rx) || dy.irrelevant(ly, ry))
        return 0;
    if (lx<=dx.ll && dx.rr<=rx) {
        if (ly<=dy.ll && dy.rr<=ry) return Max[dx.id][dy.id];
        int Max1 = max_range(dx, dy.left(), lx, rx, ly, ry);
        int Max2 = max_range(dx, dy.right(), lx, rx, ly, ry);
        return max(Max1, Max2);
    } else {
        int Max1 = max_range(dx.left(), dy, lx, rx, ly, ry);
        int Max2 = max_range(dx.right(), dy, lx, rx, ly, ry);
        return max(Max1, Max2);
    }
}

const int M=100005, N=1003;
int m, k, x[M], y[M], z[M];

main() {
    scanf("%d%d", &m, &k);
    for (int i=1; i<=m; i++)
    scanf("%d%d%d", &x[i], &y[i], &z[i]);

    dir dx(0, N+N, 1), dy(0, N+N, 1);
    for (int i=m; i>=1; i--) {
        #define actual(x, y, k) x+y-k, x+y+k, x-y-k+N, x-y+k+
             N
        int F = max_range(dx, dy, actual(x[i], y[i], k)) + z[
             i];
        maximize(dx, dy, x[i]+y[i], x[i]-y[i]+N, F, false);
    }
    cout << max_range(dx, dy, actual(0, 0, k)) << endl;
}
```

## 1.16 splay-tree

```
// SplayTreeById
//
// Note:
// - op() must be commutative, otherwise reverse queries won
     't work.
//   To fix it, need to store aggregate data from right->left
//   See https://judge.yosupo.jp/submission/53778 (and look
     at invsum)
//
// Tested:
// - (cut, join)    https://vn.spoj.com/problems/CARDS/
// - (keys, reverse) https://oj.vnoi.info/problem/twist
// - (insert, prod) https://oj.vnoi.info/problem/qmax3vn
// - (insert, delete) https://vn.spoj.com/problems/QMAX4/
// - (insert, delete) https://vn.spoj.com/problems/CARDSHUF/
// - (lazy)         https://judge.yosupo.jp/problem/
     dynamic_sequence_range_affine_range_sum
```

```
// - (lazy)        https://oj.vnoi.info/problem/upit
template<class K, class S, class F>
struct node_t {
    using Node = node_t<K, S, F>;

    std::array<Node*, 2> child;
    Node *father;
    int size;

    // Whether we will need to reverse this subtree.
    // Handling reverse operations requires some specialized
        code,
    // so I couldn't put this in F
    bool reverse;

    K key;
    S data;
    F lazy;
};
template<
    class K,                            // key
    class S,                            // node aggregate
        data
    S (*op) (S, K, S),                  // for recomputing
        data of a node
    pair<K, S> (*e) (),                 // identity data
    class F,                            // lazy propagation
        tag
    pair<K, S> (*mapping) (F, node_t<K, S, F>*), // apply tag
         F on a node
    F (*composition) (F, F),            // combine 2 tags
    F (*id)()                           // identity tag
>
struct SplayTreeById {
    using Node = node_t<K, S, F>;

    Node *nil, *root;

    SplayTreeById() {
        initNil();
        root = nil;
    }
    SplayTreeById(const vector<K>& keys) {
        initNil();
        root = createNode(keys, 0, (int) keys.size());
    }

    vector<K> getKeys() {
        vector<K> keys;
        traverse(root, keys);
        return keys;
    }

    // k in [0, n-1]
    Node* kth(int k) {
        auto res = _kth(root, k);
        splay(res);
        root = res;
        return res;
    }

    // Return <L, R>:
    // - L contains [0, k-1]
    // - R contains [k, N-1]
    // Modify tree
    pair<Node*, Node*> cut(int k) {
        if (k == 0) {
            return {nil, root};
        } else if (k == root->size) {
            return {root, nil};
        } else {
            Node *left = kth(k - 1); // kth already splayed
            Node* right = left->child[1];
            left->child[1] = right->father = nil;
            pushUp(left);
            return {left, right};
        }
    }

    // Return <X, Y, Z>:
    // - X contains [0, u-1]
    // - Y contains [u, v-1]
    // - Z contains [v, N-1]
    // This is useful for queries on [u, v-1]
    // Modify tree
    tuple<Node*, Node*, Node*> cut(int u, int v) {
        auto [xy, z] = cut(v);
        root = xy;
        auto [x, y] = cut(u);
        return {x, y, z};
    }

    // Make this tree x + y
    void join(Node *x, Node *y) {
        if (x == nil) {
            root = y;
            return;
        }
        while (1) {
            pushDown(x);
            if (x->child[1] == nil) break;
            x = x->child[1];
        }
        splay(x);
        setChild(x, y, 1);
        pushUp(x);
        root = x;
    }

    // reverse range [u, v-1]
    void reverse(int u, int v) {
        assert(0 <= u && u <= v && v <= root->size);
        if (u == v) return;

        auto [x, y, z] = cut(u, v);
        y->reverse = true;
        join(x, y);
        join(root, z);
    }

    // apply F on range [u, v-1]
    void apply(int u, int v, const F& f) {
        assert(0 <= u && u <= v && v <= root->size);
        if (u == v) return;

        auto [x, y, z] = cut(u, v);
        y->lazy = composition(f, y->lazy);
        std::tie(y->key, y->data) = mapping(f, y);

        join(x, y);
        join(root, z);
    }

    // Insert before pos
    // pos in [0, N]
    void insert(int pos, K key) {
        assert(0 <= pos && pos <= root->size);
        // x: [0, pos-1]
        // y: [pos, N-1]
        auto [x, y] = cut(pos);
        auto node = createNode(key);
        setChild(node, x, 0);
        setChild(node, y, 1);
        pushUp(node);
        root = node;
    }

    // Delete pos; pos in [0, N-1]
    K erase(int pos) {
        assert(0 <= pos && pos < root->size);
```

```
        // x = [0, pos-1]
        // y = [pos, pos]
        // z = [pos+1, N-1]
        auto [x, y, z] = cut(pos, pos+1);
        join(x, z);
        return y->key;
    }

    // aggregated data of range [l, r-1]
    S prod(int l, int r) {
        auto [x, y, z] = cut(l, r);
        auto res = y->data;
        join(x, y);
        join(root, z);
        return res;
    }

// private:
    void initNil() {
        nil = new Node();
        nil->child[0] = nil->child[1] = nil->father = nil;
        nil->size = 0;
        nil->reverse = false;
        std::tie(nil->key, nil->data) = e();
        nil->lazy = id();
    }
    void pushUp(Node* x) {
        if (x == nil) return;
        x->size = x->child[0]->size + x->child[1]->size + 1;
        x->data = op(x->child[0]->data, x->key, x->child[1]->
            data);
    }
    void pushDown(Node* x) {
        if (x == nil) return;

        if (x->reverse) {
            for (auto c : x->child) {
                if (c != nil) {
                    c->reverse ^= 1;
                }
            }
            std::swap(x->child[0], x->child[1]);
            x->reverse = false;
        }

        for (auto c : x->child) {
            if (c != nil) {
                std::tie(c->key, c->data) = mapping(x->lazy, c
                    );
                c->lazy = composition(x->lazy, c->lazy);
            }
            // For problem like UPIT, where we want to push
                different
            // lazy tags to left & right children, may need
                to modify
            // code here
            // (query L R X: a(L) += X, a(L+1) += 2X, ...)
            // e.g. for UPIT:
            // x->lazy.add_left += (1 + c->size) * x->lazy.
                step;
        }

        x->lazy = id();
    }
    Node* createNode(K key) {
        Node *res = new Node();
        res->child[0] = res->child[1] = res->father = nil;
        res->key = key;
        res->size = 1;
        res->data = e().second;
        res->lazy = id();
        return res;
    }
    void setChild(Node *x, Node *y, int d) {
        x->child[d] = y;
        if (y != nil) y->father = x;
    }
    // Assumption: x is father of y
    int getDirection(Node *x, Node *y) {
        assert(y->father == x);
        return x->child[0] == y ? 0 : 1;
    }
    // create subtree from keys[l, r-1]
    Node* createNode(const vector<K>& keys, int l, int r) {
        if (l >= r) { // empty
            return nil;
        }
        int mid = (l + r) / 2;
        Node *p = createNode(keys[mid]);
        Node *left = createNode(keys, l, mid);
        Node *right = createNode(keys, mid + 1, r);

        setChild(p, left, 0);
        setChild(p, right, 1);

        pushUp(p);
        return p;
    }
    void traverse(Node* x, vector<K>& keys) {
        if (x == nil) return;
        pushDown(x);
        traverse(x->child[0], keys);
        keys.push_back(x->key);
        traverse(x->child[1], keys);
    }
    /**
     * Before:
     *    y
     *    |
     *    x
     *  /
     * z
     *  \
     *  zchild
     *
     * After:
     *    y
     *    |
     *    z
     *     \
     *      x
     *     /
     *  zchild
     */
    void rotate(Node *x, int d) {
        Node *y = x->father;
        Node *z = x->child[d];
        setChild(x, z->child[d ^ 1], d);
        setChild(y, z, getDirection(y, x));
        setChild(z, x, d ^ 1);
        pushUp(x);
        pushUp(z);
    }
    // Make x root of tree
    Node *splay(Node *x) {
        if (x == nil) return nil;
        while (x->father != nil) {
            Node *y = x->father;
            Node *z = y->father;
            int dy = getDirection(y, x);
            int dz = getDirection(z, y);
            if (z == nil) {
                rotate(y, dy);
            } else if (dy == dz) {
                rotate(z, dz);
                rotate(y, dy);
            } else {
                rotate(y, dy);
                rotate(z, dz);
```

```
            }
        }
        return x;
    }

    Node* _kth(Node* p, int k) {
        pushDown(p);
        // left: [0, left->size - 1]
        if (k < p->child[0]->size) {
            return _kth(p->child[0], k);
        }
        k -= p->child[0]->size;
        if (!k) return p;
        return _kth(p->child[1], k - 1);
    }
};

////////// Below: example usage
// Splay tree only need to store keys (no aggregated value / no lazy update)
struct KeyOnlyOps {
    struct S{};
    struct F{};
    using Node = node_t<int, S, F>;

    static S op(__attribute__((unused)) S left, __attribute__((unused)) int key, __attribute__((unused)) S right) {
        return {};
    }
    static pair<int, S> e() {
        return {-1, {}};
    }
    static pair<int, S> mapping(__attribute__((unused)) F f, Node* node) {
        return {node->key, {}};
    }
    static F composition(__attribute__((unused)) F f, __attribute__((unused)) F g) {
        return {};
    }
    static F id() {
        return {};
    }
};

/* Example:
    SplayTreeById<
        int,
        KeyOnlyOps::S,
        KeyOnlyOps::op,
        KeyOnlyOps::e,
        KeyOnlyOps::F,
        KeyOnlyOps::mapping,
        KeyOnlyOps::composition,
        KeyOnlyOps::id
    > tree(keys);
 */


// For query get max of keys in range
// No lazy update tags
struct MaxQueryOps {
    static const int INF = 1e9 + 11;
    struct F{};
    using Node = node_t<int, int, F>;

    static int op(const int& left, int key, const int& right) {
        return max({left, key, right});
    }
    static pair<int, int> e() {
        return {-1, -INF};
    }
    static pair<int, int> mapping(__attribute__((unused)) const F& f, Node* node) {
        return {node->key, node->data};
    }
    static F composition(__attribute__((unused)) const F& f, __attribute__((unused)) const F& g) {
        return {};
    }
    static F id() {
        return {};
    }
};
/* Example:
    SplayTreeById<
        int,
        int,
        MaxQueryOps::op,
        MaxQueryOps::e,
        MaxQueryOps::F,
        MaxQueryOps::mapping,
        MaxQueryOps::composition,
        MaxQueryOps::id
    > tree;
 */

// For queries a[i] <- a[i]*mult + add
struct RangeAffineOps {
    struct S {
        long long sum, sz;
    };
    struct F {
        long long a, b;
    };
    using Node = node_t<int, S, F>;

    static const int MOD = 998244353;
    static S op(const S& left, int key, const S& right) {
        return S {
            (left.sum + key + right.sum) % MOD,
            left.sz + 1 + right.sz,
        };
    }
    static pair<int, S> e() {
        return {0, {0, 0}};
    }
    static pair<int, S> mapping(const F& f, Node* node) {
        return {
            (f.a * node->key + f.b) % MOD,
            S {
                (f.a * node->data.sum + f.b * node->data.sz) % MOD,
                node->data.sz,
            }
        };
    }
    static F composition(const F&f, const F& g) {
        return F {
            f.a * g.a % MOD,
            (f.a * g.b + f.b) % MOD,
        };
    }
    static F id() {
        return F {1, 0};
    }
};

/* Example
    SplayTreeById<
        int,
        RangeAffineOps::S,
        RangeAffineOps::op,
        RangeAffineOps::e,
        RangeAffineOps::F,
        RangeAffineOps::mapping,
        RangeAffineOps::composition,
        RangeAffineOps::id
```

```
   > tree(keys);
*/
```

## 1.17 treap

```
#include <bits/stdc++.h>
#define elif else if
#define NIL &leaf
using namespace std;
const int INF=1<<30;
struct node {
    node *p, *l,*r;
    int key,pr;
    };
node *root,leaf;
node* newnode(node* parent,int key) {
    node *x=new node;
    x->p=parent;
    x->l=x->r=NIL;
    x->pr=rand();
    x->key=key;
    if(parent!=NIL) {
        if(parent->key>key) parent->l=x;
        else parent->r=x;
        }
    else root=x;
    return x;
    }
void init() {
    leaf.l=leaf.r=leaf.p=NIL;
    leaf.key=-INF;
    leaf.pr=-1;
    root=NIL;
    }
void link(node* x,node* y) {
    if(y==root) root=x;
    elif(y==y->p->l) y->p->l=x;
    else y->p->r=x;
    x->p=y->p;
    y->p=x;
    }
void uptree(node* x) {
    node *parent=x->p;
    link(x,parent);
    if(x==parent->l) {
        parent->l=x->r;
        if(parent->l!=NIL) parent->l->p=parent;
        x->r=parent;
        }
    else {
        parent->r=x->l;
        if(parent->r!=NIL) parent->r->p=parent;
        x->l=parent;
        }
    }
void insert(int key) {
    node* x=root,*parent=NIL;
    while(x!=NIL) {
        parent=x;
        if(key==x->key) return;
        if(key<x->key) x=x->l;
        else x=x->r;
        }
    x=newnode(parent,key);

    while(x!=root&&x->pr>x->p->pr) {
        uptree(x);
        }
    }
node* find(int key) {
    node* x=root;
    while(x!=NIL) {
        if(key==x->key) return x;
        if(key<x->key) x=x->l;
        else x=x->r;
        }
    return NIL;
    }
void delall(node* x) {
    if(x==root) root=NIL;
    elif(x==x->p->l) x->p->l=NIL;
    else x->p->r=NIL;
    }
bool del(int key) {
    node* x=find(key);
    if(x==NIL) return false;
    while(x->l!=NIL&&x->r!=NIL) {
        if(x->l->pr>x->r->pr) uptree(x->l);
        else uptree(x->r);
        }
    if(x->l!=NIL) link(x->l,x);
    elif(x->r!=NIL) link(x->r,x);
    else delall(x);
    free(x);
    return true;
    }
node* Min() {
    node* x=root;
    while(x->l!=NIL) {
        x=x->l;
        }
    return x;
    }
node* Max() {
    node* x=root;
    while(x->r!=NIL) {
        x=x->r;
        }
    return x;
    }
node* succ(int key) {
    node* ans=NIL;
    node* x=root;
    while(x!=NIL) {
        if(key<x->key) {
            ans=x;
            x=x->l;
            }
        else x=x->r;
        }
    return ans;
    }
node* pred(int key) {
    node* ans=NIL;
    node* x=root;
    while(x!=NIL) {
        if(key>x->key) {
            ans=x;
            x=x->r;
            }
        else x=x->l;
        }
    return ans;
    }
char sss[30];
void dfs(node *x) {
    if(x==NIL) return ;
    printf("%d->>>",x->key);
    dfs(x->l);
    printf("||||| %d->>>",x->key);
    dfs(x->r);
    }
main() {
    //freopen("out","w",stdout);
    ios_base::sync_with_stdio(false);
    srand(time(NULL));
    init();
    int n;
    int x;
```

```
while(cin>>n&&n) {
  //  dfs(root);
   // puts("");
    if(n==1) {
        cin>>x;
        insert(x);
        }
    elif(n==2) {
        cin>>x;
        del(x);
        }
    elif(n==3) {

        if(root==NIL) {
            puts("empty");
            continue;
            }
        printf("%d\n",Min()->key);
        }
    elif(n==4) {

        if(root==NIL) {
            puts("empty");
            continue;
            }
        printf("%d\n",Max()->key);
        }
    elif(n==5) {
        cin>>x;

        if(root==NIL) {
            puts("empty");
            continue;
            }
        node* f=succ(x);
        if(f!=NIL) printf("%d\n",f->key);
        else puts("no");
        }
    elif(n==6) {
        cin>>x;

        if(root==NIL) {
            puts("empty");
            continue;
            }
        node* f=find(x);
        if(f==NIL) f=succ(x);
        if(f!=NIL) printf("%d\n",f->key);
        else puts("no");
        }
        elif(n==7) {
            cin>>x;

            if(root==NIL) {
                puts("empty");
                continue;
                }
            node* f=pred(x);
            if(f!=NIL) printf("%d\n",f->key);
            else puts("no");

            }
        elif(n==8) {
            cin>>x;

            if(root==NIL) {
                puts("empty");
                continue;
                }
            node* f=find(x);
            if(f==NIL) f=pred(x);
            if(f!=NIL) printf("%d\n",f->key);
            else puts("no");
            }
        }

    }

// https://sites.google.com/site/kc97ble/container/treap-cpp
```

# 2 Geometry

## 2.1 circle

```
struct Circle : Point {
    double r;
    Circle(double _x = 0, double _y = 0, double _r = 0) :
        Point(_x, _y), r(_r) {}
    Circle(Point p, double _r) : Point(p), r(_r) {}

    bool contains(Point p) { return (*this - p).len() <= r +
        EPS; }

    double area() const { return r*r*M_PI; }

    // definitions in https://en.wikipedia.org/wiki/Circle
    // assumption: 0 <= theta <= 2*PI
    // theta: angle in radian
    double sector_area(double theta) const {
        return 0.5 * r * r * theta;
    }

    // assumption: 0 <= theta <= 2*PI
    // theta: angle in radian
    double segment_area(double theta) const {
        return 0.5 * r * r * (theta - sin(theta));
    }
};
istream& operator >> (istream& cin, Circle& c) {
    cin >> c.x >> c.y >> c.r;
    return cin;
}
ostream& operator << (ostream& cout, const Circle& c) {
    cout << '(' << c.x << ", " << c.y << ") " << c.r;
    return cout;
}

// Find common tangents to 2 circles
// Tested:
// - http://codeforces.com/gym/100803/ - H
// Helper method
void tangents(Point c, double r1, double r2, vector<Line> &
     ans) {
    double r = r2 - r1;
    double z = sqr(c.x) + sqr(c.y);
    double d = z - sqr(r);
    if (d < -EPS) return;
    d = sqrt(fabs(d));
    Line l((c.x * r + c.y * d) / z,
           (c.y * r - c.x * d) / z,
           r1);
    ans.push_back(l);
}
// Actual method: returns vector containing all common
     tangents
vector<Line> tangents(Circle a, Circle b) {
    vector<Line> ans; ans.clear();
    for (int i=-1; i<=1; i+=2)
        for (int j=-1; j<=1; j+=2)
            tangents(b-a, a.r*i, b.r*j, ans);
    for(int i = 0; i < (int) ans.size(); ++i)
        ans[i].c -= ans[i].a * a.x + ans[i].b * a.y;

    vector<Line> ret;
    for(int i = 0; i < (int) ans.size(); ++i) {
        if (std::none_of(ret.begin(), ret.end(), [&] (Line l)
             { return areSame(l, ans[i]); })) {
            ret.push_back(ans[i]);
```

```
        }
    }
    return ret;
}

// Circle & line intersection
// Tested:
// - http://codeforces.com/gym/100803/ - H
vector<Point> intersection(Line l, Circle cir) {
    double r = cir.r, a = l.a, b = l.b, c = l.c + l.a*cir.x +
         l.b*cir.y;
    vector<Point> res;

    double x0 = -a*c/(a*a+b*b), y0 = -b*c/(a*a+b*b);
    if (c*c > r*r*(a*a+b*b)+EPS) return res;
    else if (fabs(c*c - r*r*(a*a+b*b)) < EPS) {
        res.push_back(Point(x0, y0) + Point(cir.x, cir.y));
        return res;
    } else {
        double d = r*r - c*c/(a*a+b*b);
        double mult = sqrt (d / (a*a+b*b));
        double ax,ay,bx,by;
        ax = x0 + b * mult;
        bx = x0 - b * mult;
        ay = y0 - a * mult;
        by = y0 + a * mult;

        res.push_back(Point(ax, ay) + Point(cir.x, cir.y));
        res.push_back(Point(bx, by) + Point(cir.x, cir.y));
        return res;
    }
}

// helper functions for commonCircleArea
double cir_area_solve(double a, double b, double c) {
    return acos((a*a + b*b - c*c) / 2 / a / b);
}
double cir_area_cut(double a, double r) {
    double s1 = a * r * r / 2;
    double s2 = sin(a) * r * r / 2;
    return s1 - s2;
}
// Tested: http://codeforces.com/contest/600/problem/D
double commonCircleArea(Circle c1, Circle c2) { //return the
     common area of two circle
    if (c1.r < c2.r) swap(c1, c2);
    double d = (c1 - c2).len();
    if (d + c2.r <= c1.r + EPS) return c2.r*c2.r*M_PI;
    if (d >= c1.r + c2.r - EPS) return 0.0;
    double a1 = cir_area_solve(d, c1.r, c2.r);
    double a2 = cir_area_solve(d, c2.r, c1.r);
    return cir_area_cut(a1*2, c1.r) + cir_area_cut(a2*2, c2.r
        );
}

// Check if 2 circle intersects. Return true if 2 circles
    touch
bool areIntersect(Circle u, Circle v) {
    if (cmp((u - v).len(), u.r + v.r) > 0) return false;
    if (cmp((u - v).len() + v.r, u.r) < 0) return false;
    if (cmp((u - v).len() + u.r, v.r) < 0) return false;
    return true;
}

// If 2 circle touches, will return 2 (same) points
// If 2 circle are same --> be careful
// Tested:
// - http://codeforces.com/gym/100803/ - H
// - http://codeforces.com/gym/100820/ - I
vector<Point> circleIntersect(Circle u, Circle v) {
    vector<Point> res;
    if (!areIntersect(u, v)) return res;
    double d = (u - v).len();
    double alpha = acos((u.r * u.r + d*d - v.r * v.r) / 2.0 /
         u.r / d);

    Point p1 = (v - u).rotate(alpha);
    Point p2 = (v - u).rotate(-alpha);
    res.push_back(p1 / p1.len() * u.r + u);
    res.push_back(p2 / p2.len() * u.r + u);
    return res;
}
```

## 2.2 convexHull

```
// Add lines a*x + b, must be in:
// - increasing order of a
// - decreasing order of b
// Get y = max(a*x + b)
//
// Tested:
// - http://codeforces.com/contest/678/standings/friends/
    true
// - https://oj.vnoi.info/problem/segtree_itds2

const long long INF = 1e18 + 11;
struct Line {
    long long a, b;
    long long f(long long x) {
        return a*x + b;
    }
};
bool operator < (const Line& f, const Line& g) {
    if (f.a != g.a) return f.a < g.a;
    return f.b > g.b;
}

struct Hull {
    vector<double> x;
    vector<Line> segs;

    void insert(Line l) {
        if (segs.empty()) {
            x.push_back(-INF);
            segs.push_back(l);
            return;
        }
        double xNew = -INF;
        while (!segs.empty()) {
            if (segs.back().a == l.a) {
                assert(segs.back().b >= l.b);
                return;
            }
            xNew = intersection(segs.back(), l);
            if (xNew < x.back()) {
                remove();
            } else break;
        }

        segs.push_back(l);
        x.push_back(xNew);
    }

    long long get(long long x0) {
        if (segs.empty()) {
            return -INF;
        }
        auto i = upper_bound(x.begin(), x.end(), x0) - x.
            begin() - 1;
        return segs[i].f(x0);
    }

private:
    void remove() {
        segs.pop_back();
        x.pop_back();
    }
    double intersection(const Line& f, const Line& g) {
        return 1.0 * (f.b - g.b) / (g.a - f.a);
```

```
    }
};
```

## 2.3 n-segment-intersects

```
// Given N segments.
// Check (and returns the indices) if there are 2 segments intersect.
//
// NOTES:
// - Must set Segment.id. Otherwise it will be impossible to debug..
// - Floating point number? copy from here:
//   https://cp-algorithms.com/geometry/intersecting_segments.html
//
// TESTED:
// - http://acm.timus.ru/problem.aspx?space=1&num=1469
// - http://vn.spoj.com/problems/VMLINES

int cmp(int x, int y) {
    if (x == y) return 0;
    if (x < y) return -1;
    return 1;
}
struct Point {
    int x, y;

    Point() { x = y = 0; }
    Point(int x, int y) : x(x), y(y) {}

    Point operator - (const Point& a) const {
        return Point(x - a.x, y - a.y);
    }
    int operator % (const Point& a) const {
        return x*a.y - y*a.x;
    }
};
istream& operator >> (istream& cin, Point& p) {
    cin >> p.x >> p.y;
    return cin;
}

struct Segment {
    Point p, q;
    int id;

    double get_y(int x) const {
        if (p.x == q.x) return p.y;
        return p.y + (q.y - p.y) * (x - p.x) / (double) (q.x - p.x);
    }
};
istream& operator >> (istream& cin, Segment& s) {
    cin >> s.p >> s.q;
    return cin;
}

bool intersect1d(int l1, int r1, int l2, int r2) {
    if (l1 > r1) swap(l1, r1);
    if (l2 > r2) swap(l2, r2);

    return max(l1, l2) <= min(r1, r2);
}
int ccw(Point a, Point b, Point c) {
    return cmp((b - a) % (c - a), 0);
}

bool intersect(const Segment& a, const Segment& b) {
    return intersect1d(a.p.x, a.q.x, b.p.x, b.q.x)
        && intersect1d(a.p.y, a.q.y, b.p.y, b.q.y)
        && ccw(a.p, a.q, b.p) * ccw(a.p, a.q, b.q) <= 0
        && ccw(b.p, b.q, a.p) * ccw(b.p, b.q, a.q) <= 0;
}

bool operator < (const Segment& a, const Segment& b) {
    int x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.x));
    return a.get_y(x) < b.get_y(x) - 1e-9;
}

struct Event {
    int x;
    int tp, id;

    Event() {}
    Event(int x, int tp, int id) : x(x), tp(tp), id(id) {}

    bool operator < (const Event& e) const {
        if (x != e.x) return x < e.x;
        return tp > e.tp;
    }
};

set<Segment> s;
vector< set<Segment> :: iterator> where;
set<Segment> :: iterator get_prev(set<Segment>::iterator it) {
    return it == s.begin() ? s.end() : --it;
}
set<Segment> :: iterator get_next(set<Segment>::iterator it) {
    return ++it;
}

pair<int,int> solve(const vector<Segment>& a) {
    int n = SZ(a);
    vector<Event> e;
    REP(i,n) {
        e.push_back(Event(min(a[i].p.x, a[i].q.x), +1, i));
        e.push_back(Event(max(a[i].p.x, a[i].q.x), -1, i));
    }
    sort(ALL(e));

    s.clear();
    where.resize(SZ(a));
    REP(i,SZ(e)) {
        int id = e[i].id;
        if (e[i].tp == +1) {
            set<Segment>::iterator next = s.lower_bound(a[id]), prev = get_prev(next);
            if (next != s.end() && intersect(*next, a[id])) {
                return make_pair(next->id, id);
            }
            if (prev != s.end() && intersect(*prev, a[id])) {
                return make_pair(prev->id, id);
            }
            where[id] = s.insert(next, a[id]);
        } else {
            set<Segment>::iterator next = get_next(where[id]), prev = get_prev(where[id]);
            if (next != s.end() && prev != s.end() && intersect(*next, *prev)) {
                return make_pair(prev->id, next->id);
            }
            s.erase(where[id]);
        }
    }
    return make_pair(-1, -1);
}
```

## 2.4 pollard

```
#include <bits/stdc++.h>
namespace Pollard {
    template<typename num_t>
    num_t mulmod(num_t a, num_t b, num_t p) {
        a %= p; b %= p;
```

```
    num_t q = (num_t) ((long double) a * b / p);
    num_t r = a * b - q * p;
    while (r < 0) r += p;
    while (r >= p) r -= p;
    return r;
    /*
    num_t r = 0;
    int block = 1;
    num_t base = 1LL << block;
    for (; b; b >>= block) {
        r = (r + a * (b & (base - 1))) % p;
        a = a * base % p;
    }
    return r;
    */
}
template<typename num_t>
num_t powmod(num_t n, num_t k, num_t p) {
    num_t r = 1;
    for (; k; k >>= 1) {
        if (k & 1) r = mulmod(r, n, p);
        n = mulmod(n, n, p);
    }
    return r;
}
template<typename num_t>
int rabin(num_t n) {
    if (n == 2) return 1;
    if (n < 2 || !(n & 1)) return 0;
    const num_t p[9] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    num_t a, d = n - 1, mx = 4;
    int i, r, s = 0;
    while (!(d & 1)) {++s; d >>= 1;}
    for (i = 0; i < mx; i++) {
        if (n == p[i]) return 1;
        if (!(n % p[i])) return 0;
        a = powmod(p[i], d, n);
        if (a != 1) {
            for (r = 0; r < s && a != n - 1; r++) a =
                mulmod(a, a, n);
            if (r == s) return 0;
        }
    }
    return 1;
}
template<typename num_t>
inline num_t f(num_t a, num_t b, num_t n) {
    return (mulmod(a, a, n) + b) % n;
}
template<typename num_t>
    void factorize(num_t n, vector<num_t>& facs) {
        static int init_seed = 0;
        if (!init_seed) {
            init_seed = 1;
            srand(2311);
        }
        if (n == 1) {
            return;
        }
        if (rabin(n)) {
            facs.push_back(n);
            return;
        }
        if (n == 4) {
            facs.push_back(2);
            facs.push_back(2);
            return;
        }
        while (1) {
            num_t a = rand() & 63, x = 2, y = 2;
            while (1) {
                x = f(x, a, n), y = f(f(y, a, n), a, n);
                if (x == y) break;
                num_t p = __gcd(n, y <= x ? x - y : y - x);
                if (p > 1) {
                    factorize(p, facs), factorize(n / p, facs)
                        ;
                    return;
                }
            }
        }
    }
}

vector<long long> get_divisors(map<long long, int> prime) {
    vector<long long> res(1, 1);
    for (auto it : prime) {
        int n = res.size();
        long long val = 1;
        for (int i = 0; i < it.second; ++i) {
            val *= it.first;
            for (int j = 0; j < n; ++j) {
                res.push_back(res[j] * val);
            }
        }
    }
    return res;
}
```

## 2.5 smallestEnclosingClosure

```
// Smallest enclosing circle:
// Given N points. Find the smallest circle enclosing these
    points.
// Amortized complexity: O(N)
//
// Tested:
// - https://www.spoj.com/problems/ALIENS/
// - https://www.spoj.com/problems/QCJ4/
// - https://www.acmicpc.net/problem/2626
// - https://oj.vnoi.info/problem/icpc22_mt_l

struct SmallestEnclosingCircle {
    Circle getCircle(vector<Point> points) {
        assert(!points.empty());

        random_shuffle(points.begin(), points.end());
        Circle c(points[0], 0);
        int n = points.size();

        for (int i = 1; i < n; i++)
            if ((points[i] - c).len() > c.r + EPS)
            {
                c = Circle(points[i], 0);
                for (int j = 0; j < i; j++)
                    if ((points[j] - c).len() > c.r + EPS)
                    {
                        c = Circle((points[i] + points[j]) / 2,
                              (points[i] - points[j]).len() /
                             2);
                        for (int k = 0; k < j; k++)
                            if ((points[k] - c).len() > c.r +
                                EPS)
                                c = getCircumcircle(points[i],
                                    points[j], points[k]);
                    }
            }

        return c;
    }

    // NOTE: This code work only when a, b, c are not
        collinear and no 2 points are same --> DO NOT
    // copy and use in other cases.
    Circle getCircumcircle(Point a, Point b, Point c) {
        assert(a != b && b != c && a != c);
        assert(ccw(a, b, c));
```

```
        double d = 2.0 * (a.x * (b.y - c.y) + b.x * (c.y - a.
            y) + c.x * (a.y - b.y));
        assert(fabs(d) > EPS);
        double x = (a.norm() * (b.y - c.y) + b.norm() * (c.y
            - a.y) + c.norm() * (a.y - b.y)) / d;
        double y = (a.norm() * (c.x - b.x) + b.norm() * (a.x
            - c.x) + c.norm() * (b.x - a.x)) / d;
        Point p(x, y);
        return Circle(p, (p - a).len());
    }
};
```

# 3 Graph

## 3.1 BellmanFord

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

#define ar array
#define vt vector
#define all(v) (v).begin(), (v).end()
#define pb push_back
#define ll long long
#define ld long double
#define ii pair<int, int>
#define iii pair<int, ii>
#define fi first
#define se second
#define FORIT(i, s) for (auto it=(s.begin()); it!=(s.end());
     ++it)
#define F_OR(i, a, b, s) for (int i=(a); (s)>0? i<(b) : i>(b
    ); i+=(s))
#define F_OR1(n) F_OR(i, 0, n, 1)
#define F_OR2(i, e) F_OR(i, 0, e, 1)
#define F_OR3(i, b, e) F_OR(i, b, e, 1)
#define F_OR4(i, b, e, s) F_OR(i, b, e, s)
#define GET5(a, b, c, d, e, ...) e
#define F_ORC(...) GET5(__VA_ARGS__, F_OR4, F_OR3, F_OR2,
    F_OR1)
#define FOR(...) F_ORC(__VA_ARGS__)(__VA_ARGS__)
#define EACH(x, a) for(auto& x: a)

const int d4x[] = {-1, 0, 1, 0},
        d4y[] = {0, -1, 0, 1},
        d8x[] = {-1, -1, -1, 0, 0, 1, 1, 1},
        d8y[] = {-1, 0, 1, -1, 1, -1, 0, 1},
        N = 2e5+1;
const ll oo = LLONG_MAX;
int n, // number of vertices
    m, // number of edges
    s, // start vertex
    e; // end vertex
vt<vt<ii>> G; // adjancency list of edge, G is directed
    weighted graph
ll d[N]; // distance from s_v to e_v

void bellmanFord(vt<vt<ii>> G){
    fill_n(d, sizeof(d)/sizeof(d[0]), oo);
    d[s]=0;
    FOR(u, 1, n+1){
        EACH(e, G[u]){
            int v(e.se), uv(e.fi);
            d[v]=min(d[v], d[u]+uv);
        }
    }
    FOR(u, 1, n+1){
        EACH(e, G[u]){
            int v(e.se), uv(e.fi);
            if (d[v]>d[u]+uv) d[v]=-oo; // v is in a negative
                cycle
        }
    }

}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    // freopen("test.inp", "r", stdin);
    // freopen("test.out", "w", stdout);

    cin >> n >> m;
    G = vt<vt<ii>>(n+1);
    FOR(m){
        int u, v, w;
        cin >> u >> v >> w;
        G[u].pb(ii(w, v));
    }
    cin >> s >> e;
    bellmanFord(G);
    cout << d[e];
}
```

## 3.2 Dijkstra

```
const int oo = 1e9;
vvii G;
vi d;

void dijkstra(){
    int n = G.size();
    priority_queue<ii, vii, greater<ii>> pq;
    pq.push(ii(0, 1));

    while(pq.size()){
        int u = pq.top().se,
            du = pq.top().fi;
        pq.pop();
        if (du!=d[u]) continue;

        FOR(i, G[u].size()){
            int v = G[u][i].se,
                uv = G[u][i].fi;
            if (d[v] > du+uv) d[v] = du+uv, pq.push({d[v], v
                 });
        }
    }
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    // freopen("test.inp", "r", stdin);
    // freopen("test.out", "w", stdout);

    int n, m;
    G = vvii(n+1);
    d = vi(n+1, oo);
    while(m--){
        int u, v, w;
        cin >> u >> v >> w;
        G[u].pb({w, v});
        G[v].pb({w, u});
    }
    dijkstra();
}
```

## 3.3 FloydWarshall

```
const int d4x[] = {-1, 0, 1, 0},
          d4y[] = {0, -1, 0, 1},
          d8x[] = {-1, -1, -1, 0, 0, 1, 1, 1},
          d8y[] = {-1, 0, 1, -1, 1, -1, 0, 1},
          N = 2e3+1,
          oo = 1e9;
int n, // number of vertices
    m, // number of edges
    s, // start vertex
    e; // end vertex
ll d[N][N]; // distance from x to y

void floydWarshall(){
    FOR(i, 1, n+1){
        FOR(j, 1, n+1){
            FOR(k, 1, n+1){
                d[i][j] = min(d[i][j], d[i][k]+d[k][j]);
            }
        }
    }
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    // freopen("test.inp", "r", stdin);
    // freopen("test.out", "w", stdout);

    cin >> n >> m;
    FOR(i, 1, n+1){
        FOR(j, 1, n+1) d[i][j] = oo;
    }
    FOR(i, 1, n+1) d[i][i]=0;
    FOR(m){
        int u, v, w;
        cin >> u >> v >> w;
        d[u][v] = w;
    }
    cin >> s >> e;
    floydWarshall();
    cout << d[s][e];
}
```

## 3.4 MinMaxPathDAG

```
const ll oo = LLONG_MAX;
int n, m, cnt;
vt<vt<ii>> G; // G is a DAG
vt<bool> vs;
vt<int> topo;
vt<ll> d;

void dfs(int u){
    vs[u]=true;
    // cout << u << '\n';
    EACH(e, G[u]){
        int w(e.fi), v(e.se);
        if (!vs[v]){
            dfs(v);
        }
    }
    topo[--cnt]=u;
}

void topoSort(){
    topo = vt<int>(n, 0);
    cnt = n;
    vs = vt<bool>(n+1, false);
    FOR(i, 1, n+1){
        if (!vs[i]) dfs(i);
    }
}

ll shortestPathDAG(vt<vt<ii>> G){
    vt<ll> d(n+1, oo);
    d[1] = 0LL;
    EACH(u, topo){
        EACH(e, G[u]){
            int w(e.fi), v(e.se);
            d[v] = min(d[v], d[u]+w);
        }
    }
    return d[n];
}

ll longestPathDAG(vt<vt<ii>> G){
    vt<vt<ii>> G_ = G;
    FOR(i, 1, n+1){
        EACH(e, G_[i])
            e.fi*=-1;
    }
    return -1*shortestPathDAG(G_);
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    // freopen("test.inp", "r", stdin);
    // freopen("test.out", "w", stdout);

    cin >> n >> m;
    G = vt<vt<ii>>(n+1);
    vs = vt<bool>(n+1, false);
    FOR(m){
        int u, v, w;
        cin >> u >> v >> w;
        G[u].pb(ii(w, v));
    }
    topoSort();
    cout << shortestPathDAG(G) << " " << longestPathDAG(G);
}
```

## 3.5 Tarjan

```
#include <stdio.h>

#include <algorithm>
#include <iostream>
#include <stack>
#include <vector>
using namespace std;

const int N = 100005;
const int oo = 0x3c3c3c3c;

int n, m, Num[N], Low[N], cnt = 0;
vector<int> a[N];
stack<int> st;
int Count = 0;

void visit(int u) {
    Low[u] = Num[u] = ++cnt;
    st.push(u);

    for (int v : a[u])
        if (Num[v])
            Low[u] = min(Low[u], Num[v]);
        else {
            visit(v);
            Low[u] = min(Low[u], Low[v]);
        }

    if (Num[u] == Low[u]) { // found one
        Count++;
        int v;
```

```
        do {
            v = st.top();
            st.pop();
            Num[v] = Low[v] = oo; // remove v from graph
        } while (v != u);
    }
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++) {
        int x, y;
        scanf("%d%d", &x, &y);
        a[x].push_back(y);
    }

    for (int i = 1; i <= n; i++)
        if (!Num[i]) visit(i);

    cout << Count << endl;
}
```

## 3.6 TopologicalSort

```
const int d4x[] = {-1, 0, 1, 0},
          d4y[] = {0, -1, 0, 1},
          d8x[] = {-1, -1, -1, 0, 0, 1, 1, 1},
          d8y[] = {-1, 0, 1, -1, 1, -1, 0, 1};
int n, m, cnt;
vt<int> res;
vt<vt<int>> G;
vt<bool> vs;

void dfs(int u){
    EACH(v, G[u]){
        if (!vs[v]){
            vs[v] = true;
            dfs(v);
        }
    }
    res[--cnt]=u;
}

void topoSort(){
    res = vt<int>(n);
    cnt=n;
    vs = vt<bool>(n+1, false);
    FOR(i, 1, n+1){
        if (!vs[i]){
            vs[i]=true;
            dfs(i);
        }
    }
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    freopen("test.inp", "r", stdin);
    freopen("test.out", "w", stdout);

    cin >> n >> m;
    G = vt<vt<int>>(n+1);
    FOR(m){
        int u, v;
        cin >> u >> v;
        G[u].pb(v);
    }
    topoSort();
    EACH(u, res) cout << u << '\n';

}
```

## 3.7 bipartie-edge-coloring

```
// Copied from https://judge.yosupo.jp/submission/11755
// Source: Benq
//
// Tested:
// - https://codeforces.com/contest/600/problem/F
// - https://judge.yosupo.jp/problem/bipartite_edge_coloring
// - https://oj.vnoi.info/problem/nkdec

// Credit: Benq
// returns vector of {vertex, id of edge to vertex}
// the second element of the first pair is always -1
template<int N, bool directed> struct Euler {
    vector<pair<int, int>> adj[N];
    vector<pair<int, int>>::iterator iter[N];
    bool in_vertex[N];
    vector<int> nodes;
    vector<bool> used;
    Euler() { for (int i = 0; i < N; i++) in_vertex[i] = 0; }
    vector<int> ans;
    void clear() {
        for (auto &t: nodes) adj[t].clear(), in_vertex[t] =
            0;
        nodes.clear(); used.clear(); ans.clear();
    }
    void add(int x) {
        if (in_vertex[x]) return;
        in_vertex[x] = 1;
        nodes.push_back(x);
    }
    void add_edge(int a, int b) {
        int m = used.size();
        used.push_back(0);
        add(a); add(b);
        adj[a].emplace_back(b, m);
        if (!directed) adj[b].emplace_back(a, m);
    }
    void go(int src) {
        vector<pair<pair<int, int>,int>> ret, s = {{{src,
            -1}, -1}};
        // {{vertex, prev vertex}, edge label}
        while (s.size()) {
            int x = s.back().first.first;
            auto& it = iter[x], en = end(adj[x]);
            while (it != en && used[it->second]) it ++;
            if (it == en) { // no more edges out of vertex
                if ((int)ret.size() && ret.back().first.second
                      != x) exit(5);
                ret.push_back(s.back()), s.pop_back();
            } else {
                s.push_back({{it->first,x},it->second});
                used[it->second] = 1;
            }
        }
        for (int i = 0; i < (int)ret.size() - 1; i++) ans.
            push_back(ret[i].second);
        assert((int)ans.size() % 2 == 0);
    }
    array<vector<int>, 2> tour() {
        for (auto &v: nodes) {
            assert(adj[v].size() % 2 == 0);
            iter[v] = begin(adj[v]);
        }
        for (auto &v: nodes) for (auto &e: adj[v]) if (!used[
            e.second]) go(v);
        array<vector<int>, 2> res;
        for (int i = 0; i < (int)ans.size(); i++) res[i % 2].
            push_back(ans[i]);
        return res;
    }
};

typedef array<int, 2> T;
```

```
struct EdgeColoring {
    int n; vector<T> ed;
    Euler<N * 2, 0> E; // at least 2 * n
    array<vector<int>,2> split(vector<int> lab) { // k is
         even, split into two parts
        E.clear();
        for (auto &t: lab) E.add_edge(ed[t][0], ed[t][1]);
        auto v = E.tour(); // get half edges on each
        for (int i = 0; i < 2; i++) for (auto &t: v[i]) t =
             lab[t];
        return v;
    }
    vector<int> match(vector<int> lab) { // find perfect
         matching in MlogM
        assert((int)lab.size() && (int)lab.size() % n == 0);
        int k = (int)lab.size() / n;
        int p = 0;
        while ((1 << p) < n * k) p ++;
        int a = (1 << p) / k;
        int b = (1 << p) - k * a;
        vector<int> cnt_good((int)lab.size(),a), cnt_bad(n,b)
             ; // now each edge is adjacent to 2^t
        for (; p; --p) { // divide by two!!
            E.clear(); vector<int> tmp;
            for (int i = 0; i < n * k; i++) {
                if (cnt_good[i] & 1) E.add_edge(ed[lab[i]][0],
                     ed[lab[i]][1]), tmp.push_back(i);
                cnt_good[i] /= 2;
            }
            int num_lab = tmp.size();
            for (int i = 0; i < n; i++) {
                if (cnt_bad[i] & 1) E.add_edge(i, n + i), tmp.
                     push_back(i);
                cnt_bad[i] /= 2;
            }
            array<vector<int>, 2> x = E.tour();
            T cnt = T();
            for (int i = 0; i < 2; i++) for (auto &t: x[i])
                 cnt[i] += t >= num_lab;
            if (cnt[0] > cnt[1]) swap(x[0], x[1]);
            for (auto &t: x[0]) {
                if (t < num_lab) cnt_good[tmp[t]] ++;
                else cnt_bad[tmp[t]] ++;
            }
        }
        vector<int> v;
        for (int i = 0; i < (int) lab.size(); i++) if (
             cnt_good[i]) v.push_back(lab[i]);
        assert((int)v.size() == n);
        return v;
    }
    vector<bool> used;
    vector<vector<int>> edge_color(vector<int> lab) { //
         regular bipartite graph!
        assert((int)lab.size() % n == 0);
        int k = (int)lab.size() / n;
        if (k == 0) return {};
        if (k == 1) return {lab};
        if ( __builtin_popcount(k) == 1) {
            array<vector<int>,2> p = split(lab);
            vector<vector<int>> a = edge_color(p[0]), b =
                 edge_color(p[1]);
            a.insert(end(a), b.begin(), b.end());
            return a;
        }
        if (k % 2 == 0) {
            array<vector<int>, 2> p = split(lab);
            auto a = edge_color(p[0]);
            int cur = k/2;
            while ( __builtin_popcount(cur) > 1) {
                cur ++;
                p[1].insert(end(p[1]),a.back().begin(), a.back
                     ().end());
                a.pop_back();
            }
            auto b = edge_color(p[1]);
            a.insert(end(a),b.begin(), b.end());
            return a;
        } else {
            vector<int> v = match(lab);
            for (auto &t: v) used[t] = 1;
            vector<int> LAB;
            for (auto &t: lab) if (!used[t]) LAB.push_back(t)
                 ;
            for (auto &t: v) used[t] = 0;
            auto a = edge_color(LAB);
            a.push_back(v);
            return a;
        }
    }
    // returns edge chromatic number, ans contains the edge
         coloring(colors are 1 indexed)
    // supports multiple edges
    // 0 indexed, O(M log M)
    int solve(vector<T> _ed, vector<int> &ans) {
        if (_ed.empty()) {
            return 0;
        }
        T side = T();
        for (auto &t: _ed) for (int i = 0; i < 2; i++) side[i
             ] = max(side[i], t[i]+1);
        vector<int> deg[2], cmp[2], sz[2];
        for (int i = 0; i < 2; i++) deg[i].resize(side[i]),
             cmp[i].resize(side[i]);
        for (auto &t: _ed) for (int i = 0; i < 2; i++) deg[i
             ][t[i]] ++;
        int k = 0;
        for (int i = 0; i < 2; i++) for (auto &t: deg[i]) k =
             max(k, t);
        for (int s = 0; s < 2; s++) {
            for (int i = 0; i < side[s]; ) {
                sz[s].push_back(0);
                while (i < side[s] && sz[s].back() + deg[s][i]
                     <= k) {
                    cmp[s][i] = (int)sz[s].size() - 1;
                    sz[s].back() += deg[s][i++];
                }
            }
        }
        for (int i = 0; i < 2; i++) while (sz[i].size() < sz[
             i ^ 1].size()) sz[i].push_back(0);
        n = sz[0].size();
        for (auto &t: _ed) ed.push_back({cmp[0][t[0]], n +
             cmp[1][t[1]]});
        int ind = 0;
        for (int i = 0; i < n; i++) {
            while (sz[0][i] < k) {
                while (sz[1][ind] == k) ind ++;
                sz[0][i] ++, sz[1][ind] ++;
                ed.push_back({i, n + ind});
            }
        }
        used.resize(n * k);
        vector<int> lab(n * k);
        iota(lab.begin(), lab.end(),0);
        auto tmp = edge_color(lab);
        ans.resize(_ed.size());
        for (int i = 0; i < (int) tmp.size(); i++) {
            for (auto x: tmp[i]) if (x < (int) _ed.size())
                 ans[x] = i + 1;
        }
        return tmp.size();
    }
};
```

## 3.8 dfsTree

### 3.8.1 Biconnected-component

```
// Input graph: vector< vector<int> > a, int n
// Note: 0-indexed
// Usage: BiconnectedComponent bc; (bc.components is the
    list of components)
//
// This is biconnected components by edges (1 vertex can
    belong to
// multiple components). For vertices biconnected component,
     remove
// bridges and find components
int n;
vector<vector<int>> g;
struct BiconnectedComponent {
    vector<int> low, num, s;
    vector< vector<int> > components;
    int counter;

    BiconnectedComponent() : low(n, -1), num(n, -1), counter
        (0) {
        for (int i = 0; i < n; i++)
            if (num[i] < 0)
                dfs(i, 1);
    }

    void dfs(int x, int isRoot) {
        low[x] = num[x] = ++counter;
        if (g[x].empty()) {
            components.push_back(vector<int>(1, x));
            return;
        }
        s.push_back(x);

        for (int i = 0; i < (int) g[x].size(); i++) {
            int y = g[x][i];
            if (num[y] > -1) low[x] = min(low[x], num[y]);
            else {
                dfs(y, 0);
                low[x] = min(low[x], low[y]);

                if (isRoot || low[y] >= num[x]) {
                    components.push_back(vector<int>(1, x));
                    while (1) {
                        int u = s.back();
                        s.pop_back();
                        components.back().push_back(u);
                        if (u == y) break;
                    }
                }
            }
        }
    }
};
```

### 3.8.2 BridgeArticulation

```
const int d4x[] = {-1, 0, 1, 0},
          d4y[] = {0, -1, 0, 1},
          d8x[] = {-1, -1, -1, 0, 0, 1, 1, 1},
          d8y[] = {-1, 0, 1, -1, 1, -1, 0, 1},
          N = 2e5+1;
int n, m, cnt, low[N], num[N], numChild[N];
bool isArt[N];
vt<vt<int>> G;
vt<ii> bridges;

void dfs(int u, int p){
    low[u] = num[u] = ++cnt;
    EACH(v, G[u]){
        if (!num[v]){
            ++numChild[u];
            dfs(v, u);
            low[u] = min(low[u], low[v]);
        } else {
            if (v != p){
                low[u] = min(low[u], num[v]);
            }
        }
        if (low[u]==num[u]){
            if (numChild[u]>1) isArt[u]=true;
        }
        if (num[u]<low[v]){
            bridges.pb({u, v});
            isArt[u]=true;
        }
    }
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    // freopen("test.inp", "r", stdin);
    // freopen("test.out", "w", stdout);
    cin >> n >> m;
    G = vt<vt<int>>(n+1);
    FOR(m){
        int u, v;
        cin >> u >> v;
        G[u].pb(v);
        // G[v].pb(u); // if undirected
    }
    FOR(i, 1, n+1) if (!num[i]) dfs(i, -1);
    FOR(i, 1, n+1) cout << i << " " << num[i] << " " << low[i
        ] << '\n';
    EACH(b, bridges) cout << b.fi << " " << b.se << '\n';
    FOR(i,1 , n+1) if (isArt[i]) cout << i << '\n';
}
```

### 3.8.3 StronglyConnected

```
// Index from 0
// Usage:
// DirectedDfs tree;
// Now you can use tree.scc
//
// Note: reverse(tree.scc) is topo sorted
//
// Tested:
// - (requires scc to be topo sorted) https://judge.yosupo.
    jp/problem/scc
// - https://cses.fi/problemset/task/1686/
struct DirectedDfs {
    vector<vector<int>> g;
    int n;
    vector<int> num, low, current, S;
    int counter;
    vector<int> comp_ids;
    vector< vector<int> > scc;

    DirectedDfs(const vector<vector<int>>& _g) : g(_g), n(g.
        size()),
           num(n, -1), low(n, 0), current(n, 0), counter(0),
                comp_ids(n, -1) {
        for (int i = 0; i < n; i++) {
            if (num[i] == -1) dfs(i);
        }
    }

    void dfs(int u) {
        low[u] = num[u] = counter++;
        S.push_back(u);
```

```
        current[u] = 1;
        for (auto v : g[u]) {
            if (num[v] == -1) dfs(v);
            if (current[v]) low[u] = min(low[u], low[v]);
        }
        if (low[u] == num[u]) {
            scc.push_back(vector<int>());
            while (1) {
                int v = S.back(); S.pop_back(); current[v] =
                     0;
                scc.back().push_back(v);
                comp_ids[v] = ((int) scc.size()) - 1;
                if (u == v) break;
            }
        }
    }

    // build DAG of strongly connected components
    // Returns: adjacency list of DAG
    std::vector<std::vector<int>> build_scc_dag() {
        std::vector<std::vector<int>> dag(scc.size());
        for (int u = 0; u < n; u++) {
            int x = comp_ids[u];
            for (int v : g[u]) {
                int y = comp_ids[v];
                if (x != y) {
                    dag[x].push_back(y);
                }
            }
        }
        return dag;
    }
};
```

## 3.9 heavylight-adamat

```
// HeavyLight {{{
// Index from 0
// Best used with SegTree.h
//
// Usage:
// HLD hld(g, root);
// // build segment tree. Note that we must use hld.order[i]
// vector<T> nodes;
// for (int i = 0; i < n; i++)
//   nodes.push_back(initial_value[hld.order[i]])
// SegTree<S, op, e> st(nodes);
//
// // Update path
// hld.apply_path(from, to, is_edge_or_vertex, [&] (int l,
    int r) {
//   st.apply(l, r+1, F);
// });
//
// // Query path
// hld.prod_path_commutative<S, op, e> (from, to,
    is_edge_or_vertex, [&] (int l, int r) {
//   return st.prod(l, r+1);
// });
//
// Tested:
// - (vertex, path) https://judge.yosupo.jp/problem/
    vertex_add_path_sum
// - (vertex, path, non-commutative) https://judge.yosupo.jp
    /problem/vertex_set_path_composite
// - (vertex, subtree) https://judge.yosupo.jp/problem/
    vertex_add_subtree_sum
// - (vertex, path, non-commutative, 1-index) https://oj.
    vnoi.info/problem/icpc21_mt_l
// - (vertex, path) https://oj.vnoi.info/problem/qtree3
//
// - (edge, path) https://oj.vnoi.info/problem/qtreex
// - (edge, path) https://oj.vnoi.info/problem/lubenica
// - (edge, path) https://oj.vnoi.info/problem/pwalk
// - (edge, path, lazy) https://oj.vnoi.info/problem/kbuild
// - (edge, path, lazy) https://oj.vnoi.info/problem/
    onbridge
//
// - (lca) https://oj.vnoi.info/problem/fselect
// - (kth_parent) https://cses.fi/problemset/task/1687
#include<bits/stdc++.h>
using namespace std;
struct HLD {
    HLD(const vector<vector<int>>& _g, int root)
            : n(_g.size()), g(_g),
            parent(n), depth(n), sz(n),
            dfs_number(0), nxt(n), in(n), out(n), order(n)
    {
        assert(0 <= root && root < n);

        // init parent, depth, sz
        // also move most heavy child of u to g[u][0]
        depth[root] = 0;
        dfs_sz(root, -1);

        // init nxt, in, out
        nxt[root] = root;
        dfs_hld(root);
    }

    int lca(int u, int v) const {
        assert(0 <= u && u < n);
        assert(0 <= v && v < n);
        while (true) {
            if (in[u] > in[v]) swap(u, v); // in[u] <= in[v]
            if (nxt[u] == nxt[v]) return u;
            v = parent[nxt[v]];
        }
    }

    // return k-th parent
    // if no such parent -> return -1
    int kth_parent(int u, int k) const {
        assert(0 <= u && u < n);
        if (depth[u] < k) return -1;

        while (true) {
            int v = nxt[u];
            if (in[u] - k >= in[v]) return order[in[u] - k];
            k -= in[u] - in[v] + 1;
            u = parent[v];
        }
    }

    // return k-th vertex on path from u -> v (0 <= k)
    // if k > distance -> return -1
    int kth_vertex_on_path(int u, int v, int k) const {
        assert(0 <= u && u < n);
        assert(0 <= v && v < n);

        int l = lca(u, v);
        int ul = depth[u] - depth[l];
        if (k <= ul) return kth_parent(u, k);
        k -= ul;
        int vl = depth[v] - depth[l];
        if (k <= vl) return kth_parent(v, vl - k);
        return -1;
    }

    int dist(int u, int v) const {
        assert(0 <= u && u < n);
        assert(0 <= v && v < n);
        int l = lca(u, v);
        return depth[u] + depth[v] - 2*depth[l];
    }

    // apply f on vertices on path [u, v]
    // edge = true -> apply on edge
    //
```

```
// f(l, r) should update segment tree [l, r] INCLUSIVE
void apply_path(int u, int v, bool edge, const function<
     void(int, int)> &f) {
    assert(0 <= u && u < n);
    assert(0 <= v && v < n);
    if (u == v && edge) return;

    while (true) {
        if (in[u] > in[v]) swap(u, v); // in[u] <= in[v]
        if (nxt[u] == nxt[v]) break;
        f(in[nxt[v]], in[v]);
        v = parent[nxt[v]];
    }
    if (u == v && edge) return;
    f(in[u] + edge, in[v]);
}

// get prod of path u -> v
// edge = true -> get on edges
//
// f(l, r) should query segment tree [l, r] INCLUSIVE
// f must be commutative. For non-commutative, use
     getSegments below
template<class S, S (*op) (S, S), S (*e)()>
S prod_path_commutative(
        int u, int v, bool edge,
        const function<S(int, int)>& f) const {
    assert(0 <= u && u < n);
    assert(0 <= v && v < n);
    if (u == v && edge) {
        return e();
    }
    S su = e(), sv = e();
    while (true) {
        if (in[u] > in[v]) { swap(u, v); swap(su, sv); }
        if (nxt[u] == nxt[v]) break;
        sv = op(sv, f(in[nxt[v]], in[v]));
        v = parent[nxt[v]];
    }
    if (u == v && edge) {
        return op(su, sv);
    } else {
        return op(su, op(sv, f(in[u] + edge, in[v])));
    }
}

// f(l, r) modify seg_tree [l, r] INCLUSIVE
void apply_subtree(int u, bool edge, const function<void(
     int, int)>& f) {
    assert(0 <= u && u < n);
    f(in[u] + edge, out[u] - 1);
}

// f(l, r) queries seg_tree [l, r] INCLUSIVE
template<class S>
S prod_subtree_commutative(int u, bool edge, const
     function<S(S, S)>& f) {
    assert(0 <= u && u < n);
    return f(in[u] + edge, out[u] - 1);
}

// Useful when functions are non-commutative
// Return all segments on path from u -> v
// For this problem, the order (u -> v is different from
     v -> u)
vector< pair<int,int> > getSegments(int u, int v) const {
    assert(0 <= u && u < n);
    assert(0 <= v && v < n);
    vector< pair<int,int> > upFromU, upFromV;

    int fu = nxt[u], fv = nxt[v];
    while (fu != fv) { // u and v are on different chains
        if (depth[fu] >= depth[fv]) { // move u up
            upFromU.push_back({u, fu});
            u = parent[fu];
            fu = nxt[u];
        } else { // move v up
            upFromV.push_back({fv, v});
            v = parent[fv];
            fv = nxt[v];
        }
    }
    upFromU.push_back({u, v});
    reverse(upFromV.begin(), upFromV.end());
    upFromU.insert(upFromU.end(), upFromV.begin(),
         upFromV.end());
    return upFromU;
}

// return true if u is ancestor
bool isAncestor(int u, int v) {
    return in[u] <= in[v] && out[v] <= out[u];
}

// private:
    int n;
    vector<vector<int>> g;
    vector<int> parent; // par[u] = parent of u. par[root] =
         -1
    vector<int> depth;  // depth[u] = distance from root -> u
    vector<int> sz;     // sz[u] = size of subtree rooted at
         u
    int dfs_number;
    vector<int> nxt;    // nxt[u] = vertex on heavy path of u
         , nearest to root
    vector<int> in, out; // subtree(u) is in range [in[u],
         out[u]-1]
    vector<int> order;  // euler tour

    void dfs_sz(int u, int fu) {
        parent[u] = fu;
        sz[u] = 1;
        // remove parent from adjacency list
        auto it = std::find(g[u].begin(), g[u].end(), fu);
        if (it != g[u].end()) g[u].erase(it);

        for (int& v : g[u]) {
            depth[v] = depth[u] + 1;
            dfs_sz(v, u);

            sz[u] += sz[v];
            if (sz[v] > sz[g[u][0]]) swap(v, g[u][0]);
        }
    }

    void dfs_hld(int u) {
        order[dfs_number] = u;
        in[u] = dfs_number++;

        for (int v : g[u]) {
            nxt[v] = (v == g[u][0] ? nxt[u] : v);
            dfs_hld(v);
        }
        out[u] = dfs_number;
    }
};
// }}}
int main() {

}
```

## 3.10 maxflowDinic

```
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(
         cap) {}
};
```

```
struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid
             ++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u] || edges[id].cap -
                 edges[id].flow < 1)
                continue;
            long long tr = dfs(u, min(pushed, edges[id].cap -
                 edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

## 3.11 minSpanningTree

```
#include <vector>
#include <iostream>
#include <unordered_map>
using namespace std;

struct DSU {
    vector<int> lab;

    DSU(int n) : lab(n+1, -1) {}

    int getRoot(int u) {
        if (lab[u] < 0) return u;
        return lab[u] = getRoot(lab[u]);
    }

    bool merge(int u, int v) {
        u = getRoot(u); v = getRoot(v);
        if (u == v) return false;
        if (lab[u] > lab[v]) swap(u, v);
        lab[u] += lab[v];
        lab[v] = u;
        return true;
    }

    bool same_component(int u, int v) {
        return getRoot(u) == getRoot(v);
    }

    int component_size(int u) {
        return -lab[getRoot(u)];
    }
};
// }}}
using ll = long long;
struct Edge {
    int u, v;
    ll c;
};
bool operator < (const Edge& a, const Edge& b) {
    return a.c < b.c;
}
ostream& operator << (ostream& out, const Edge& e) {
    out << e.u << " - " << e.v << " [" << e.c << ']';
    return out;
}
std::pair<ll, std::vector<Edge>> mst(
        int n,
        std::vector<Edge> edges) {
    std::sort(edges.begin(), edges.end());

    DSU dsu(n + 1); // tolerate 1-based index
    ll total = 0;
    vector<Edge> tree;
    for (const auto& e : edges) {
        const auto [u, v, c] = e;
        if (dsu.merge(u, v)) {
            total += c;
            tree.push_back(e);
        }
    }
    return {total, tree};
}
int main() {
    int V, E; cin >> V >> E;
    unordered_map<string, int> nameMap;
    vector<string> people(V);
```

```
    for(int i = 0; i < V; ++i) {
        cin >> people[i];
        nameMap[people[i]] = i;
    }
    vector<Edge> adj;
    for(int i = 0; i < E; ++i) {
        string person1, person2;
        int weight;
        cin >> person1 >> person2 >> weight;
        adj.push_back({nameMap[person1], nameMap[person2],
            weight});
        adj.push_back({nameMap[person2], nameMap[person1],
            weight});
    }
    auto result = mst(V, adj);
//    cout << result.second;
    int sum = 0;
    for(auto &it: result.second) {
        cout << people[it.u] << " " << people[it.v] <<'\n';
        sum += it.c;
    }
    cout << sum << '\n';
}
```

# 4 Math

## 4.1 BigNum

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

using namespace std;

#define vt vector
#define pb push_back
#define ll long long
#define vi vt<int>
#define FORIT(i, s) for (auto it=(s.begin()); it!=(s.end());
      ++it)
#define F_OR(i, a, b, s) for (int i=(a); (s)>0? i<(b) : i>(b
    ); i+=(s))
#define F_OR1(n) F_OR(i, 0, n, 1)
#define F_OR2(i, e) F_OR(i, 0, e, 1)
#define F_OR3(i, b, e) F_OR(i, b, e, 1)
#define F_OR4(i, b, e, s) F_OR(i, b, e, s)
#define GET5(a, b, c, d, e, ...) e
#define F_ORC(...) GET5(__VA_ARGS__, F_OR4, F_OR3, F_OR2,
    F_OR1)
#define FOR(...) F_ORC(__VA_ARGS__)(__VA_ARGS__)
#define EACH(x, a) for(auto& x: a)

/*
    Treat BigNum as a vector<int>, char from _size-1 -> 0.
    BASE: each character of BigNum is a integer in range [0,
        BASE).
*/
const int BASE = 1e5;

void fix(vi &x){
    /*
        fix to the right form after operator
    */
    x.pb(0);
    FOR(x.size()-1){
        x[i+1] += x[i]/BASE;
        x[i] %= BASE;
        if (x[i]<0) x[i]+=BASE, --x[i+1];
    }
    while(x.size()>1 && !x.back()) x.pop_back();
}

vi operator + (vi x, const vi &y){
    x.resize(max(x.size(), y.size()));
    FOR(y.size()) x[i] += y[i];
    return fix(x), x;
}

vi operator - (vi x, const vi &y){
    x.resize(max(x.size(), y.size()));
    FOR(y.size()) x[i] -= y[i];
    return fix(x), x;
}

vi operator * (vi x, int k){
    assert(k<BASE);
    EACH(xi, x) xi *= k;
    return fix(x), x;
}

vi operator * (const vi &x, const vi &y){
    vi z(x.size()+y.size()+1);
    FOR(x.size()) FOR(j, y.size()){
        z[i+j] += x[i]*y[j];
        z[i+j+1] += z[i+j]/BASE;
        z[i+j] %= BASE;
    }
    return fix(z), z;
}

vi operator / (vi x, int k){
    assert(k<BASE);
    for(int i=x.size()-1, r=0; i>=0; --i) r=r*BASE+x[i], x[i
        ]=r/k;
    return fix(x), x;
}

bool operator < (const vi &x, const vi &y){
    if (x.size()!=y.size()) return x.size()<y.size();
    FOR(i, x.size()-1, -1, -1) if (x[i]!=y[i]) return x[i]<y[
        i];
    return false;
}

istream &operator>>(istream &cin, vi &a) {
    string s;
    cin >> s;
    a.clear();
    a.resize(s.size()/4+1);
    FOR(s.size()){
        int x = (s.size()-1-i)/4; // <- log10(BASE)=4
        a[x] = a[x]*10+(s[i]-'0');
    }
    return fix(a), cin;
}

ostream &operator<<(ostream &cout, const vi &a) {
    printf("%d", a.back());
    FOR(i, a.size()-2, -1, -1) printf("%04d", a[i]);
    return cout;
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    // freopen("test.inp", "r", stdin);
    // freopen("test.out", "w", stdout);
}
```

## 4.2 euler-totient

```
void phi_1_to_n(int n) {
    vector<int> phi(n+1);
    phi[0] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; ++i) {
        phi[i] = i-1;
    }
```

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

ll phi_euler(ll n)
{
 ll res = n;
 for (ll i = 2; i * i <= n; ++i)
 {
  if (n % i == 0)
  {
   while (n % i == 0)
    n /= i;
   res -= res / i;
  }
 }
 if (n > 1)
  res -= res / n;
 return res;
}
```

## 4.3 extendedEuclid

```
struct Triple {
    ll d,x,y;
};

Triple extendedEuclid(ll A, ll B) {
    if (B == 0) return {A, 1, 0};
    else {
        Triple ext = extendedEuclid(B, A%B);
        return {ext.d, ext.y, ext.x - (A/B) * ext.y};
    }
}
```

## 4.4 get$_d ivisor$

```
template <typename _Tp>
vector<_Tp> get_divisor(_Tp n)
{
 vector<_Tp> arr;
 factorize(n, arr);
 map<_Tp, int> prime;
 for (_Tp i : arr)
  prime[i]++;
 vector<_Tp> res(1, 1);
 for (auto it : prime)
 {
  int k = res.size();
  _Tp val = 1;
  for (int i = 0; i < it.second; ++i)
  {
   val *= it.first;
   for (int j = 0; j < k; ++j)
   {
    res.push_back(res[j] * val);
   }
  }
 }
 return res;
}
```

## 4.5 leastPrimeFactor

```
const int N = 5*1e6+5;

vi prime;
vi lpf(N, 2);

void sieve() {
    prime.assign(1,2);
    lpf[1] = -2;
    for(int i = 3; i <= N; i+=2) {
        if (lpf[i] == 2) prime.push_back(lpf[i] = i);
        for(int j = 0; j < (int) prime.size() && prime[j] <=
             lpf[i] && i * prime[j] <= N; ++j) {
            lpf[prime[j]*i] = prime[j];
        }
    }
}
```

## 4.6 matrix

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

using namespace std;

const int mod = 111539786;

using type = int;

struct Matrix {
    vector <vector <type> > data;

    int row() const { return data.size(); }

    int col() const { return data[0].size(); }

    auto & operator [] (int i) { return data[i]; }

    const auto & operator[] (int i) const { return data[i]; }

    Matrix() = default;

    Matrix(int r, int c): data(r, vector <type> (c)) { }

    Matrix(const vector <vector <type> > &d): data(d) { }

    friend ostream & operator << (ostream &out, const Matrix
         &d) {
        for (auto x : d.data) {
            for (auto y : x) out << y << ' ';
            out << '\n';
        }
        return out;
    }

    static Matrix identity(long long n) {
        Matrix a = Matrix(n, n);
        while (n--) a[n][n] = 1;
        return a;
    }

    Matrix operator * (const Matrix &b) {
        Matrix a = *this;
        assert(a.col() == b.row());
        Matrix c(a.row(), b.col());
        for (int i = 0; i < a.row(); ++i)
            for (int j = 0; j < b.col(); ++j)
                for (int k = 0; k < a.col(); ++k){
                    c[i][j] += 1ll * a[i][k] % mod * (b[k][j]
                         % mod) % mod;
                    c[i][j] %= mod;
                }
        return c;
    }

    Matrix pow(long long exp) {
        assert(row() == col());
        Matrix base = *this, ans = identity(row());
        for (; exp > 0; exp >>= 1, base = base * base)
```

```
            if (exp & 1) ans = ans * base;
        return ans;
    }
};

int main(){
    Matrix a({
        {1, 1},
        {1, 0}
    });

    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        Matrix tmp = a.pow(n - 1);
        cout << (tmp[0][0] + tmp[0][1]) % mod << '\n';
    }
}
```

## 4.7 miller

```
pair<ll, ll> factor(ll n)
{
 ll s = 0;
 while ((n & 1) == 0)
 {
  s++;
  n >>= 1;
 }
 return {s, n};
}
ll pow(ll a, ll d, ll n)
{
 ll r = 1;
 a = a % n;
 while (d > 0)
 {
  if (d & 1)
   r = (r * a) % n;
  d >>= 1;
  a = (a * a) % n;
 }
 return r;
}
bool test_a(ll s, ll d, ll n, ll a)
{
 if (n == a)
  return true;
 ll p = pow(a, d, n);
 if (p == 1)
  return true;
 for (; s > 0; s--)
 {
  if (p == n - 1)
   return true;
  p = p * p % n;
 }
 return false;
}
bool miller(ll n)
{
 if (n < 2)
  return false;
 if ((n & 1) == 0)
  return n == 2;
 ll s, d;
 tie(s, d) = factor(n - 1);

 if (n < 1373653)
 {
  return test_a(s, d, n, 2) && test_a(s, d, n, 3);
 }
 else if (n < 4759123141)
 {
  return test_a(s, d, n, 2) && test_a(s, d, n, 7) && test_a(
      s, d, n, 61);
 }
 else
 {
  return test_a(s, d, n, 2) && test_a(s, d, n, 3) && test_a(
      s, d, n, 5) && test_a(s, d, n, 7) && test_a(s, d, n,
      11) && test_a(s, d, n, 13) && test_a(s, d, n, 17) &&
      test_a(s, d, n, 19) && test_a(s, d, n, 23);
 }
}
```

## 4.8 mod-operator

```
long long addMod(long long a, long long b, long long m) {
    return ((a+b) % m + m) % m;}

long long mulMod(long long a, long long b, long long m) {
    long long res = 0;
    for (a %= m, b %= m; b > 0; a = addMod(a, a, m), b >>= 1)
        if (b & 1) res = addMod(res, a, m);
    return res;
}

long long powMod(long long a, long long n, long long m) {
    long long res = 1;
    for (a %= m; n > 0; a = mulMod(a, a, m), n >>= 1)
        if (n & 1) res = mulMod(res, a, m);
    return res;
}
long long powMod2(long long a, long long n, long long m) {
    long long res = 1;
    for (a %= m; n > 0; a = a*a % m, n >>= 1)
        if (n & 1) res = res * a % m;
    return res;
}
```

## 4.9 modint

```
// ModInt {{{
template <int MD> struct ModInt {
  using ll = long long;
  int x;

  constexpr ModInt() : x(0) {}
  constexpr ModInt(ll v) { _set(v % MD + MD); }
  constexpr static int mod() { return MD; }
  constexpr explicit operator bool() const { return x != 0;
      }

  constexpr ModInt operator+(const ModInt &a) const {
    return ModInt()._set((ll)x + a.x);
  }
  constexpr ModInt operator-(const ModInt &a) const {
    return ModInt()._set((ll)x - a.x + MD);
  }
  constexpr ModInt operator*(const ModInt &a) const {
    return ModInt()._set((ll)x * a.x % MD);
  }
  constexpr ModInt operator/(const ModInt &a) const {
    return ModInt()._set((ll)x * a.inv().x % MD);
  }
  constexpr ModInt operator-() const { return ModInt()._set(
      MD - x); }

  constexpr ModInt &operator+=(const ModInt &a) { return *
      this = *this + a; }
  constexpr ModInt &operator-=(const ModInt &a) { return *
      this = *this - a; }
  constexpr ModInt &operator*=(const ModInt &a) { return *
      this = *this * a; }
```

```
constexpr ModInt &operator/=(const ModInt &a) { return *
    this = *this / a; }

friend constexpr ModInt operator+(ll a, const ModInt &b) {
  return ModInt()._set(a % MD + b.x);
}
friend constexpr ModInt operator-(ll a, const ModInt &b) {
  return ModInt()._set(a % MD - b.x + MD);
}
friend constexpr ModInt operator*(ll a, const ModInt &b) {
  return ModInt()._set(a % MD * b.x % MD);
}
friend constexpr ModInt operator/(ll a, const ModInt &b) {
  return ModInt()._set(a % MD * b.inv().x % MD);
}

constexpr bool operator==(const ModInt &a) const { return
    x == a.x; }
constexpr bool operator!=(const ModInt &a) const { return
    x != a.x; }

friend std::istream &operator>>(std::istream &is, ModInt &
    x) {
  ll val;
  is >> val;
  x = ModInt(val);
  return is;
}
constexpr friend std::ostream &operator<<(std::ostream &os
    , const ModInt &x) {
  return os << x.x;
}

constexpr ModInt pow(ll k) const {
  ModInt ans = 1, tmp = x;
  while (k) {
    if (k & 1)
      ans *= tmp;
    tmp *= tmp;
    k >>= 1;
  }
  return ans;
}

constexpr ModInt inv() const {
  if (x < 1000111) {
    _precalc(1000111);
    return invs[x];
  }
  int a = x, b = MD, ax = 1, bx = 0;
  while (b) {
    int q = a / b, t = a % b;
    a = b;
    b = t;
    t = ax - bx * q;
    ax = bx;
    bx = t;
  }
  assert(a == 1);
  if (ax < 0)
    ax += MD;
  return ax;
}

static std::vector<ModInt> factorials, inv_factorials,
    invs;
constexpr static void _precalc(int n) {
  if (factorials.empty()) {
    factorials = {1};
    inv_factorials = {1};
    invs = {0};
  }
  if (n > MD)
    n = MD;
  int old_sz = factorials.size();
  if (n <= old_sz)
    return;

  factorials.resize(n);
  inv_factorials.resize(n);
  invs.resize(n);

  for (int i = old_sz; i < n; ++i)
    factorials[i] = factorials[i - 1] * i;
  inv_factorials[n - 1] = factorials.back().pow(MD - 2);
  for (int i = n - 2; i >= old_sz; --i)
    inv_factorials[i] = inv_factorials[i + 1] * (i + 1);
  for (int i = n - 1; i >= old_sz; --i)
    invs[i] = inv_factorials[i] * factorials[i - 1];
}

static int get_primitive_root() {
  static int primitive_root = 0;
  if (!primitive_root) {
    primitive_root = [&]() {
      std::set<int> fac;
      int v = MD - 1;
      for (ll i = 2; i * i <= v; i++)
        while (v % i == 0)
          fac.insert(i), v /= i;
      if (v > 1)
        fac.insert(v);
      for (int g = 1; g < MD; g++) {
        bool ok = true;
        for (auto i : fac)
          if (ModInt(g).pow((MD - 1) / i) == 1) {
            ok = false;
            break;
          }
        if (ok)
          return g;
      }
      return -1;
    }();
  }
  return primitive_root;
}

private:
  // Internal, DO NOT USE.
  // val must be in [0, 2*MD)
  constexpr inline __attribute__((always_inline)) ModInt &
      _set(ll v) {
    x = v >= MD ? v - MD : v;
    return *this;
  }
};
template <int MD> std::vector<ModInt<MD>> ModInt<MD>::
    factorials = {1};
template <int MD> std::vector<ModInt<MD>> ModInt<MD>::
    inv_factorials = {1};
template <int MD> std::vector<ModInt<MD>> ModInt<MD>::invs =
     {0};
// }}}
```

## 4.10 primeFactor

```
vector<long long> trial_division3(long long n) {
    vector<long long> factorization;
    for (int d : {2, 3, 5}) {
        while (n % d == 0) {
            factorization.push_back(d);
            n /= d;
        }
    }
    static array<int, 8> increments = {4, 2, 4, 2, 4, 6, 2,
        6};
    int i = 0;
    for (long long d = 7; d * d <= n; d += increments[i++]) {
```

```
        while (n % d == 0) {
            factorization.push_back(d);
            n /= d;
        }
        if (i == 8)
            i = 0;
    }
    if (n > 1)
        factorization.push_back(n);
    return factorization;
}
```

## 4.11 rabin

```
template <typename _Tp>
int rabin(_Tp n)
{
 if (n == 2)
  return 1;
 if (n < 2 || !(n & 1))
  return 0;
 const _Tp p[9] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
 _Tp a, d = n - 1, mx = 4;
 int i, r, s = 0;
 while (!(d & 1))
 {
  ++s;
  d >>= 1;
 }
 for (i = 0; i < mx; i++)
 {
  if (n == p[i])
   return 1;
  if (!(n % p[i]))
   return 0;
  a = powmod(p[i], d, n);
  if (a != 1)
  {
   for (r = 0; r < s && a != n - 1; r++)
    a = mulmod(a, a, n);
   if (r == s)
    return 0;
  }
 }
 return 1;
}
```

# 5 String

## 5.1 Aho-Corasick

```
// Tested:
// - https://open.kattis.com/problems/stringmultimatching
// - https://icpc.kattis.com/problems/firstofhername
// - https://oj.vnoi.info/problem/binpal
//
// Notes:
// - Node IDs from 0 to aho.sz.
// - Characters should be normalized to [0, MC-1].
// - For each node of AhoCorasick, we store a linked list
    containing all queries "associated" with this node.
//  The reason is that, when we reach a node in AhoCorasick,
     it's possible to match several queries at once.
//  (this happens when queries are suffix of others, e.g. C,
     BC, ABC).
//  This also means 1 node maps to several queries, and 1
    query maps to several nodes.
//  However I believe that the sum of length of all linked
    list is O(N) -- TODO: Source / proof required.
#include<bits/stdc++.h>
#include<string.h>
#include<assert.h>

const int MN = 1000111; // MN > total length of all patterns
const int MC = 26; // Alphabet size.

// Start of Linked list
struct Node {
    int x; Node *next;
} *nil;
struct List {
    Node *first, *last;
    List() { first = last = nil; }
    void add(int x) {
        Node *p = new Node;
        p->x = x; p->next = nil;
        if (first == nil) last = first = p;
        else last->next = p, last = p;
    }
};
// End of linked list
//
struct Aho {
    int qu[MN], suffixLink[MN];
    List leaf[MN];
    int link[MN][MC];
    int sz;
    bool calledBuildLink;

    void init() {
        calledBuildLink = false;
        sz = 0;
        memset(suffixLink, 0, sizeof suffixLink);
        leaf[0] = List();
        memset(link[0], -1, sizeof link[0]);
    }
    int getChild(int type, int v, int c) {
        if (type == 2) assert(calledBuildLink);

        if (link[v][c] >= 0) return link[v][c];
        if (type == 1) return 0;
        if (!v) return link[v][c] = 0;
        return link[v][c] = getChild(type, suffixLink[v], c);
    }
    void buildLink() {
        calledBuildLink = true;
        int first, last;
        qu[first = last = 1] = 0;
        while (first <= last) {
            int u = qu[first++];
            for(int c = 0; c < MC; ++c) {
                int v = link[u][c]; if (v < 0) continue;
                qu[++last] = v;
                if (u == 0) suffixLink[v] = 0;
                else suffixLink[v] = getChild(2, suffixLink[u
                     ], c);

                if (leaf[suffixLink[v]].first != nil) {
                    if (leaf[v].first == nil) {
                        leaf[v].first = leaf[suffixLink[v]].
                             first;
                        leaf[v].last = leaf[suffixLink[v]].last
                             ;
                    }
                    else {
                        leaf[v].last->next = leaf[suffixLink[v
                             ]].first;
                        leaf[v].last = leaf[suffixLink[v]].last
                             ;
                    }
                }
            }
        }
    }
} aho;
// Usage:
int main() {
```

```
    aho.init(); // Initialize
    // Foreach query, insert one character at a time:
                int p = 0;
                while (k--) {
                    int x; scanf("%d", &x);
                    int t = aho.getChild(1, p, x);
                    if (t > 0) p = t;
                    else {
                        ++aho.sz;
                        aho.leaf[aho.sz] = List();
                        memset(aho.link[aho.sz], -1, sizeof aho
                             .link[aho.sz]);

                        aho.link[p][x] = aho.sz;
                        p = aho.sz;
                    }
                }
                aho.leaf[p].add(i);
    // Init back link
            aho.buildLink();
    // After this stage, we should use aho.getChild(2, node,
         c) to jump
}
```

## 5.2 KMP-online

```
// C++ program to implement a
// real time optimized KMP
// algorithm for pattern searching

#include <iostream>
#include <set>
#include <string>
#include <unordered_map>

using std::string;
using std::unordered_map;
using std::set;
using std::cout;

// Function to print
// an array of length len
void printArr(int* F, int len,
   char name)
{
 cout << '(' << name << ')'
  << "contain: [";

 // Loop to iterate through
 // and print the array
 for (int i = 0; i < len; i++) {
  cout << F[i] << " ";
 }
 cout << "]\n";
}

// Function to print a table.
// len is the length of each array
// in the map.
void printTable(
 unordered_map<char, int*>& FT,
 int len)
{
 cout << "Failure Table: {\n";

 // Iterating through the table
 // and printing it
 for (auto& pair : FT) {

  printArr(pair.second,
    len, pair.first);
 }
 cout << "}\n";
}

// Function to construct
// the failure function
// corresponding to the pattern
void constructFailureFunction(
 string& P, int* F)
{

 // P is the pattern,
 // F is the FailureFunction
 // assume F has length m,
 // where m is the size of P

 int len = P.size();

 // F[0] must have the value 0
 F[0] = 0;

 // The index, we are parsing P[1..j]
 int j = 1;
 int l = 0;

 // Loop to iterate through the
 // pattern
 while (j < len) {

  // Computing the failure function or
  // lps[] similar to KMP Algorithm
  if (P[j] == P[l]) {
   l++;
   F[j] = l;
   j++;
  }
  else if (l > 0) {
   l = F[l - 1];
  }
  else {
   F[j] = 0;
   j++;
  }
 }
}

// Function to construct the failure table.
// P is the pattern, F is the original
// failure function. The table is stored in
// FT[][]
void constructFailureTable(
 string& P,
 set<char>& pattern_alphabet,
 int* F,
 unordered_map<char, int*>& FT)
{
 int len = P.size();

 // T is the char where we mismatched
 for (char t : pattern_alphabet) {

  // Allocate an array
  FT[t] = new int[len];
  int l = 0;
  while (l < len) {
   if (P[F[l]] == t)

    // Old failure function gives
    // a good shifting
    FT[t][l] = F[l] + 1;
   else {

    // Move to the next char if
    // the entry in the failure
    // function is 0
    if (F[l] == 0)
     FT[t][l] = 0;
```

```
    // Fill the table if F[l] > 0
    else
     FT[t][l] = FT[t][F[l] - 1];
   }
   l++;
  }
 }
}

// Function to implement the realtime
// optimized KMP algorithm for
// pattern searching. T is the text
// we are searching on and
// P is the pattern we are searching for
void KMP(string& T, string& P,
  set<char>& pattern_alphabet)
{

 // Size of the pattern
 int m = P.size();

 // Size of the text
 int n = T.size();

 // Initialize the Failure Function
 int F[m];

 // Constructing the failure function
 // using KMP algorithm
 constructFailureFunction(P, F);
 printArr(F, m, 'F');

 unordered_map<char, int*> FT;

 // Construct the failure table and
 // store it in FT[][]
 constructFailureTable(
  P,
  pattern_alphabet,
  F, FT);
 printTable(FT, m);

 // The starting index will be when
 // the first match occurs
 int found_index = -1;

 // Variable to iterate over the
 // indices in Text T
 int i = 0;

 // Variable to iterate over the
 // indices in Pattern P
 int j = 0;

 // Loop to iterate over the text
 while (i < n) {
  if (P[j] == T[i]) {

   // Matched the last character in P
   if (j == m - 1) {
    found_index = i - m + 1;
    break;
   }
   else {
    i++;
    j++;
   }
  }
  else {
   if (j > 0) {

    // T[i] is not in P's alphabet
    if (FT.find(T[i]) == FT.end())

     // Begin a new
     // matching process
     j = 0;

    else
     j = FT[T[i]][j - 1];

    // Update 'j' to be the length of
    // the longest suffix of P[1..j]
    // which is also a prefix of P

    i++;
   }
   else
    i++;
  }
 }

 // Printing the index at which
 // the pattern is found
 if (found_index != -1)
  cout << "Found at index "
   << found_index << '\n';
 else
  cout << "Not Found \n";

 for (char t : pattern_alphabet)

  // Deallocate the arrays in FT
  delete[] FT[t];

 return;
}

// Driver code
int main()
{
 string T = "cababababcababaca";
 string P = "ababaca";
 set<char> pattern_alphabet
  = { 'a', 'b', 'c' };
 KMP(T, P, pattern_alphabet);
}
/*
The new preprocessing step has a running time complexity of
     O(|\Sigma_P| \cdot M), where \Sigma_P is the alphabet
     set of pattern P, M is the size of P.
The whole modified KMP algorithm has a running time
     complexity of O(|\Sigma_P| \cdot M + N). The auxiliary
     space usage of O(|\Sigma_P| \cdot M).
The running time and space usage look like worse than the
     original KMP algorithm. However, if we are searching
     for the same pattern in multiple texts or the alphabet
     set of the pattern is small, as the preprocessing step
     only needs to be done once and each character in the
     text will be compared at most once (real-time). So, it
     is more efficient than the original KMP algorithm and
     good in practice.
*/
```

## 5.3 KMP

```
// C++ program for implementation of KMP pattern searching
// algorithm
#include <bits/stdc++.h>

void computeLPSArray(char* pat, int M, int* lps);

// Prints occurrences of txt[] in pat[]
void KMPSearch(char* pat, char* txt)
{
 int M = strlen(pat);
 int N = strlen(txt);
```

```
 // create lps[] that will hold the longest prefix suffix
 // values for pattern
 int lps[M];

 // Preprocess the pattern (calculate lps[] array)
 computeLPSArray(pat, M, lps);

 int i = 0; // index for txt[]
 int j = 0; // index for pat[]
 while ((N - i) >= (M - j)) {
  if (pat[j] == txt[i]) {
   j++;
   i++;
  }

  if (j == M) {
   printf("Found pattern at index %d ", i - j);
   j = lps[j - 1];
  }

  // mismatch after j matches
  else if (i < N && pat[j] != txt[i]) {
   // Do not match lps[0..lps[j-1]] characters,
   // they will match anyway
   if (j != 0)
    j = lps[j - 1];
   else
    i = i + 1;
  }
 }
}

// Fills lps[] for given pattern pat[0..M-1]
void computeLPSArray(char* pat, int M, int* lps)
{
 // length of the previous longest prefix suffix
 int len = 0;

 lps[0] = 0; // lps[0] is always 0

 // the loop calculates lps[i] for i = 1 to M-1
 int i = 1;
 while (i < M) {
  if (pat[i] == pat[len]) {
   len++;
   lps[i] = len;
   i++;
  }
  else // (pat[i] != pat[len])
  {
   // This is tricky. Consider the example.
   // AAACAAAA and i = 7. The idea is similar
   // to search step.
   if (len != 0) {
    len = lps[len - 1];

    // Also, note that we do not increment
    // i here
   }
   else // if (len == 0)
   {
    lps[i] = 0;
    i++;
   }
  }
 }
}

// Driver program to test above function
int main()
{
 char txt[] = "ABABDABACDABABCABAB";
 char pat[] = "ABABCABAB";
 KMPSearch(pat, txt);
 return 0;
}
```

## 5.4 Palindrome-tree-eerTree

```
// Palindrome Tree {{{
// Notes:
// - number of *distinct* palindrome substring <= N
// Tested:
// - https://oj.vnoi.info/problem/icpc22_mn_d
template<int MAXC = 26>
struct PalindromicTree {
    PalindromicTree(const string& str)
            : _sz(str.size() + 5),
              next(_sz, vector<int> (MAXC, 0)),
              link(_sz, 0), qlink(_sz, 0),
              cnt(_sz, 0), right_id(_sz, 0),
              len(_sz, 0), s(_sz, 0) {
        init();
        for (int i = 0; i < (int) str.size(); ++i) {
            add(str[i], i);
        }
        count();
    }
    int _sz;

    // returns vector of (left, right, frequency)
    vector<tuple<int,int,int>> get_palindromes() {
        vector<tuple<int,int,int>> res;
        dfs(0, res);
        dfs(1, res);
        return res;
    }

    void dfs(int u, vector<tuple<int,int,int>>& res) {
        if (u > 1) { // u = 0 and u = 1 are two empty nodes
            res.emplace_back(right_id[u] - len[u] + 1,
                 right_id[u], cnt[u]);
        }
        for (int i = 0; i < MAXC; ++i) {
            if (next[u][i]) dfs(next[u][i], res);
        }
    }

    int last, n, p;
    vector<vector<int>> next, dlink;
    vector<int> link, qlink, cnt, right_id, len, s;

    int newnode(int l, int right) {
        len[p] = l;
        right_id[p] = right;
        return p++;
    }
    void init() {
        p = 0;
        newnode(0, -1), newnode(-1, -1);
        n = last = 0;
        s[n] = -1, link[0] = 1;
    }
    int getlink(int x) {
        while (s[n - len[x] - 1] != s[n]) {
            if (s[n - len[link[x]] - 1] == s[n]) x = link[x];
            else x = qlink[x];
        }
        return x;
    }
    void add(char c, int right) {
        c -= 'a';
        s[++n] = c;
        int cur = getlink(last);
        if (!next[cur][(int) c]) {
            int now = newnode(len[cur] + 2, right);
            link[now] = next[getlink(link[cur])][(int) c];
            next[cur][(int) c] = now;
```

```
            if (s[n - len[link[now]]] == s[n - len[link[link[
                now]]]]) {
                qlink[now] = qlink[link[now]];
            }
            else {
                qlink[now] = link[link[now]];
            }
        }
        last = next[cur][(int) c];
        cnt[last]++;
    }
    void count() {
        for (int i = p - 1; i >= 0; i--) {
            cnt[link[i]] += cnt[i];
        }
    }
};
// }}}
```

## 5.5 SuffixArray

```
// Suffix Array {{{
// Source: http://codeforces.com/contest/452/submission
    /7269543
// Efficient Suffix Array O(N*logN)

// String index from 0
// Usage:
// string s; (s[i] > 0)
// SuffixArray sa(s);
// Now we can use sa.SA and sa.LCP
// sa.LCP[i] = max common prefix suffix of sa.SA[i-1] and sa
    .SA[i]
//
// Notes:
// - Number of distinct substrings = |S| * (|S| + 1) / 2 -
    sum(LCP)
//
// Tested:
// - (build SA) https://judge.yosupo.jp/problem/suffixarray
// - (LCP) https://judge.yosupo.jp/problem/
    number_of_substrings
// - (LCP - kth distinct substr) https://cses.fi/problemset/
    task/2108
// - (LCP - longest repeated substr) https://cses.fi/
    problemset/task/2106/
#include <algorithm>
// #include <cassert>
#include <iostream>
#include <string>
#include <vector>
using namespace std;
struct SuffixArray {
  string a;
  int N, m;
  vector<int> SA, LCP, x, y, w, c;

  SuffixArray(string _a, int _m = 256)
      : a(" " + _a), N(a.length()), m(_m), SA(N), LCP(N), x(N
          ), y(N),
        w(max(m, N)), c(N) {
    a[0] = 0;
    DA();
    kasaiLCP();
#define REF(X)

    \
  {

      \
    rotate(begin(X), begin(X) + 1, end(X));
                                        \
    X.pop_back();

        \
  }
    REF(SA);
    REF(LCP);
    a = a.substr(1, a.size());
    for (int i = 0; i < (int)SA.size(); ++i)
      --SA[i];
#undef REF
  }

  inline bool cmp(const int u, const int v, const int l) {
    return (y[u] == y[v] &&
            (u + l < N && v + l < N ? y[u + l] == y[v + l] :
                false));
  }

  void Sort() {
    for (int i = 0; i < m; ++i)
      w[i] = 0;
    for (int i = 0; i < N; ++i)
      ++w[x[y[i]]];
    for (int i = 0; i < m - 1; ++i)
      w[i + 1] += w[i];
    for (int i = N - 1; i >= 0; --i)
      SA[--w[x[y[i]]]] = y[i];
  }
  void DA() {
    for (int i = 0; i < N; ++i)
      x[i] = a[i], y[i] = i;
    Sort();
    for (int i, j = 1, p = 1; p < N; j <<= 1, m = p) {
      for (p = 0, i = N - j; i < N; ++i)
        y[p++] = i;
      for (int k = 0; k < N; ++k)
        if (SA[k] >= j)
          y[p++] = SA[k] - j;
      Sort();
      for (swap(x, y), p = 1, x[SA[0]] = 0, i = 1; i < N; ++i
          )
        x[SA[i]] = cmp(SA[i - 1], SA[i], j) ? p - 1 : p++;
    }
  }
  void kasaiLCP() {
    for (int i = 0; i < N; ++i)
      c[SA[i]] = i;
    for (int i = 0, j, k = 0; i < N; LCP[c[i++]] = k)
      if (c[i] > 0)
        for (k ? k-- : 0, j = SA[c[i] - 1]; a[i + k] == a[j +
            k]; k++)
          ;
      else
        k = 0;
  }
};

// Example:
// given string S and Q queries pat_i, for each query, count
    how many
// times pat_i appears in S
// O(min(|S|, |pat|) * log(|S|)) per query
//
// Tested:
// - (yes / no) https://cses.fi/problemset/task/2102
// - (count) https://cses.fi/problemset/task/2103
// - (position; need RMQ) https://cses.fi/problemset/task
    /2104
int count_occurrence(const string &s, const vector<int> &sa,
                     const string &pat) {
  int n = s.size(), m = pat.size();
  // assert(n == (int)sa.size());
  if (n < m)
    return 0;
  auto f = [&](int start) { // compare s[start:...] and pat
      [0: ...]
    for (int i = 0; start + i < n && i < m; ++i) {
```

```
      if (s[start + i] < pat[i])
        return true;
      if (s[start + i] > pat[i])
        return false;
    }
    return n - start < m;
  };
  auto g = [&](int start) {
    for (int i = 0; start + i < n && i < m; ++i) {
      if (s[start + i] > pat[i])
        return false;
    }
    return true;
  };
  auto l = std::partition_point(begin(sa), end(sa), f);
  auto r = std::partition_point(l, end(sa), g);
  return std::distance(l, r);
}

int main() {
  string s;
  cin >> s;
  int k;
  cin >> k;
  vector<string> patterns;
  for (int i = 0; i < k; ++i) {
    string x;
    cin >> x;
    patterns.push_back(x);
  }
  SuffixArray sfa(s);
  for (int i = 0; i < k; ++i) {
    int ans = count_occurrence(s, sfa.SA, patterns[i]);
    cout << (ans == 0 ? "NO" : "YES") << '\n';
  }
}
```

## 5.6 maine-lorentz

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

int get_z(vector<int> const& z, int i) {
    if (0 <= i && i < (int)z.size())
        return z[i];
    else
        return 0;
}

vector<pair<int, int>> repetitions;

void convert_to_repetitions(int shift, bool left, int cntr,
     int l, int k1, int k2) {
    for (int l1 = max(1, l - k2); l1 <= min(l, k1); l1++) {
        if (left && l1 == l) break;
        int l2 = l - l1;
        int pos = shift + (left ? cntr - l1 : cntr - l - l1 +
              1);
        repetitions.emplace_back(pos, pos + 2*l - 1);
    }
}

void find_repetitions(string s, int shift = 0) {
    int n = s.size();
    if (n == 1)
        return;

    int nu = n / 2;
    int nv = n - nu;
    string u = s.substr(0, nu);
    string v = s.substr(nu);
    string ru(u.rbegin(), u.rend());
    string rv(v.rbegin(), v.rend());

    find_repetitions(u, shift);
    find_repetitions(v, shift + nu);

    vector<int> z1 = z_function(ru);
    vector<int> z2 = z_function(v + '#' + u);
    vector<int> z3 = z_function(ru + '#' + rv);
    vector<int> z4 = z_function(v);

    for (int cntr = 0; cntr < n; cntr++) {
        int l, k1, k2;
        if (cntr < nu) {
            l = nu - cntr;
            k1 = get_z(z1, nu - cntr);
            k2 = get_z(z2, nv + 1 + cntr);
        } else {
            l = cntr - nu + 1;
            k1 = get_z(z3, nu + 1 + nv - 1 - (cntr - nu));
            k2 = get_z(z4, (cntr - nu) + 1);
        }
        if (k1 + k2 >= l)
            convert_to_repetitions(shift, cntr < nu, cntr, l,
                  k1, k2);
    }
}
```

## 5.7 manacher

## 5.8 suffix-automation

```
struct Node {
    int len, link; // len = max length of suffix in this
         class
    int next[33];
};
Node s[MN * 2];
set< pair<int,int> > order; // in most application we'll
     need to sort by len
struct Automaton {
    int sz, last;
    Automaton() {
  order.clear();
        sz = last = 0;
        s[0].len = 0;
        s[0].link = -1;
        ++sz;
        // need to reset next if necessary
    }
    void extend(char c) {
        c = c - 'A';
        int cur = sz++, p;
        s[cur].len = s[last].len + 1;
        order.insert(make_pair(s[cur].len, cur));

        for(p = last; p != -1 && !s[p].next[c]; p = s[p].link
             )
            s[p].next[c] = cur;
        if (p == -1) s[cur].link = 0;
        else {
```

```
            int q = s[p].next[c];
            if (s[p].len + 1 == s[q].len) s[cur].link = q;
            else {
                int clone = sz++;
                s[clone].len = s[p].len + 1;
                memcpy(s[clone].next, s[q].next, sizeof(s[q].
                     next));
                s[clone].link = s[q].link;
                order.insert(make_pair(s[clone].len, clone));

                for(; p != -1 && s[p].next[c] == q; p = s[p].
                     link)
                    s[p].next[c] = clone;
                s[q].link = s[cur].link = clone;
            }
        }
        last = cur;
    }
};
// Construct:
// Automaton sa; for(char c : s) sa.extend(c);
// 1. Number of distinct substr:
//    - Find number of different paths --> DFS on SA
//    - f[u] = 1 + sum( f[v] for v in s[u].next
// 2. Number of occurrences of a substr:
//    - Initially, in extend: s[cur].cnt = 1; s[clone].cnt =
    0;
//    - for(it : reverse order)
//        p = nodes[it->second].link;
//        nodes[p].cnt += nodes[it->second].cnt
// 3. Find total length of different substrings:
//    - We have f[u] = number of strings starting from node u
//    - ans[u] = sum(ans[v] + d[v] for v in next[u])
// 4. Lexicographically k-th substring
//    - Based on number of different substring
// 5. Smallest cyclic shift
//    - Build SA of S+S, then just follow smallest link
// 6. Find first occurrence
//    - firstpos[cur] = len[cur] - 1, firstpos[clone] =
    firstpos[q]
```

## 5.9 zfunc

```
// z[i] = length of longest common prefix of s[0..N] and s[i
    ..N]
//
// Tested:
// - https://judge.yosupo.jp/problem/zalgorithm
// - (string matching) https://oj.vnoi.info/problem/substr
#include <string>
#include <vector>
using namespace std;

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

// Examples:
// Find all occurrences of p in t

/**
   string s = p + "_" + t;
   auto z = zfunc(s);

   REP(i,SZ(t)) {
       if (z[i + SZ(p) + 1] == SZ(p)) {
           cout << 1+i << ' ';
       }
   }
   cout << endl;
*/
```

# 6 Test

## 6.1 binaryTrie

```
// #include <cassert>
#include <bits/stdc++.h>
// #include <array>
// #include <iostream>
// #include <vector>
// #include <pair>
using namespace std;

// Binary Trie
// Based on https://judge.yosupo.jp/submission/72657
// Supports:
// - get min / max / kth element
// - given K, find x: x^K is min / max / kth
//
// Notes:
// - high mem usage. If just need kth_element
//   -> use OrderedSet.h if MAX_VALUE is ~10^6
//   -> use STL/order_statistic.cpp if MAX_VALUE is big /
    custom type
//
// Tested:
// - (insert, remove, min xor) https://judge.yosupo.jp/
    problem/set_xor_min
// - (insert, max xor) https://cses.fi/problemset/task/1655/
template<
    class Val = long long, // values stored in Trie
    class Count = long long, // frequency of values
    int B = (sizeof(Val) * 8 - 2) // max number of bit
> struct BinaryTrie {
    struct Node {
        std::array<int, 2> child;
        Count count;
        Node() : child{-1, -1}, count(0) {}
    };

    BinaryTrie() : nodes{Node()} {} // create root node

    // Number of elements in the trie
    Count size() {
        return nodes[0].count;
    }

    void insert(Val x, Count cnt = 1) {
        update(x, cnt);
    }
    void remove(Val x, Count cnt = 1) {
        update(x, -cnt);
    }

    // return X: X ^ xor_val is minimum
    pair<Val, Node> min_element(Val xor_val = 0) {
        //assert(0 < size());
        return kth_element(0, xor_val);
    }

    // return X: X ^ xor_val is maximum
    pair<Val, Node> max_element(Val xor_val = 0) {
        //assert(0 < size());
        return kth_element(size() - 1, xor_val);
    }
```

```
    // return X: X ^ xor_val is K-th (0 <= K < size())
    pair<Val, Node> kth_element(Count k, Val xor_val = 0) {
        //assert(0 <= k && k < size());
        int u = 0;
        Val x = 0;
        for (int i = B - 1; i >= 0; i--) {
            int b = get_bit(xor_val, i);
            int v0 = get_child(u, b);
            if (nodes[v0].count <= k) {
                k -= nodes[v0].count;
                u = get_child(u, 1-b);
                x |= 1LL << i;
            } else {
                u = v0;
            }
        }
        return {x, nodes[u]};
    }

    // return frequency of x
    Count count(Val x) {
        int u = 0;
        for (int i = B - 1; i >= 0; i--) {
            int b = get_bit(x, i);
            if (nodes[u].child[b] == -1) {
                return 0;
            }
            u = get_child(u, b);
        }
        return nodes[u].count;
    }

// private:
    vector<Node> nodes;

    int get_child(int p, int b) {
        //assert(0 <= p && p < (int) nodes.size());
        //assert(0 <= b && b < 2);
        if (nodes[p].child[b] == -1) {
            nodes[p].child[b] = nodes.size();
            nodes.push_back(Node{});
        }
        return nodes[p].child[b];
    }

    void update(Val x, Count cnt) {
        int u = 0;
        for (int i = B - 1; i >= 0; i--) {
            nodes[u].count += cnt;
            //assert(nodes[u].count >= 0); // prevent over
                delete
            int b = get_bit(x, i);
            u = get_child(u, b);
        }
        nodes[u].count += cnt;
        //assert(nodes[u].count >= 0); // prevent over delete
    }

    inline int get_bit(Val v, int bit) {
        return (v >> bit) & 1;
    }
};
int main() {
    int n; cin >> n;
    vector<long long> a(n);
    BinaryTrie<long long, long long> bt;
    bt.insert(0);
    long long preXor = 0LL;
    long long res = LLONG_MIN;
    for (int i = 0; i < n; ++i) {
        cin >> a[i];
        preXor ^= a[i];
        bt.insert(preXor);
        res = max(res, bt.max_element(preXor).first);
    }
    cout << res;
    // long long xor_sum = 0LL;
    // for(int i = 0 ; i < n; ++i) {
    //     xor_sum ^= a[i];
    //     auto tmp = bt.max_element(xor_sum);
    //     mx.push_back(tmp.first);
    //
    // }
    // mx.push_back(bt.max_element().first);
    // auto ans = bt.max_element();
    // cout << ans.first;
    // for(auto &it: mx) cout << it << " ";
    // cout << *max_element(begin(mx), end(mx));
}
```

# 7 Tree

## 7.1 diameter

```
/*
Call diameter() -> return number of edge create diameter of
    tree
Just use dilemma: diameter = max(dfs(random_node), dfs(
    max_leaf))
*/


const int N = 3e5+5;
bitset<N> visited;
vector<int> adj[N];
int n;
int x; // use to track the farthest node

void dfs_utils(int u, int count, int &max_count) {
    visited[u] = true;
    count++;
    for(auto &v: adj[u]) {
        if (!visited[v]) {
            if (count > max_count) {
                x = v;
                max_count = count;
            }
            dfs_utils(v, count, max_count);
        }
    }
}

void dfs(int node, int &max_count) {
    int count = 0;
    visited.reset();
    dfs_utils(node, count, max_count);
}

int diameter() {
    int max_count = INT_MIN;
    dfs(1, max_count);
    dfs(x, max_count);
    return max(0, max_count);
}
```

## 7.2 diameter$_{s}hort$

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

const int N = 2*1e5+5;
vector<int> adj[N];

pair<int, int> mx;

void dfs(int next, int root, int len) {
```

```
    if (len > mx.first) mx = {len, next};
    for(auto &u : adj[next]) {
        if (u == root) continue;
        dfs(u, next, len+1);
    }
}

int main() {
    mx = {0,0};
    int n; cin >> n;
    for (int i = 0; i < n-1; i++)
    {
        int l, r;
        cin >> l >> r;
        adj[l].push_back(r);
        adj[r].push_back(l);
    }
    dfs(1, 0, 0);
    dfs(mx.second, 0, 0);
    cout << mx.first;
}
```

## 7.3 lowestCommonAncestor

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

const int N = 5e5+5;
vector<int> g[N];
int n, q;
int h[N], up[N][20];

void dfs(int u) {
    for (auto v: g[u]) {
        if (v == up[u][0]) continue; // v = ancestor of u
        h[v] = h[u] + 1;
        up[v][0] = u;
        for(int j = 1; j < 20; ++j) {
            up[v][j] = up[up[v][j-1]][j-1];
        }
        dfs(v);
    }
}

int lca(int u, int v) {
    if (h[u] != h[v]) {
        if (h[u] < h[v]) swap(u, v); // Without lost of
            generality
        // find ancestor u' of u so h[u'] = h[v]
        int k = h[u] - h[v];
        for(int j = 0; (1<<j) <= k; ++j) {
            if (k >> j & 1) u = up[u][j];
        }
    }
    if (u == v) return u;
    int k = __lg(h[u]);
    for(int j = k; j >= 0; --j) {
        if (up[u][j] != up[v][j]) { // if ancestor 2^j th of
            u and v is different
            u = up[u][j], v = up[v][j];
        }
    }
    return up[u][0];
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin >> n >> q;
    for(int i = 1; i < n; ++i) {
        int x; cin >> x;
        // g[i].push_back(x);
        // g[x].push_back(i);
        g[i].emplace_back(x);
        g[x].emplace_back(i);
    }
    // for(int i = 0; i < n; ++i) {
    //     cout << "Number " << i << ":\n ";
    //     for(auto &it: g[i]) cout << it << " ";
    //     cout << "\n";
    // }
    dfs(0);
    for(int i = 0; i < q; ++i) {
        int u, v; cin >> u >> v;
        cout << lca(u, v) << "\n";
    }
    return 0;
}
```

## 7.4 suffixArray

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

#include <algorithm>
#include <iostream>
using namespace std;

const int N = 200005;
int n, sa[N], ra[N], rb[N], G;
char a[N];

bool cmp(int x, int y) {
    if (ra[x] != ra[y]) return ra[x] < ra[y];
    return ra[x + G] < ra[y + G];
}

int main() {
    scanf("%s", a + 1);
    n = strlen(a + 1);

    for (int i = 1; i <= n; i++) {
        sa[i] = i;
        ra[i] = a[i];
    }

    for (G = 1; G <= n; G *= 2) {
        sort(sa + 1, sa + n + 1, cmp);
        for (int i = 1; i <= n; i++)
            rb[sa[i]] = rb[sa[i - 1]] + cmp(sa[i - 1], sa[i])
                ;
        for (int i = 1; i <= n; i++)
            ra[i] = rb[i];
        if (ra[sa[n]] == n) break;
    }

    for (int i = 1; i <= n; i++)
        printf("%d\n", sa[i] - 1);
}

// https://sites.google.com/site/kc97ble/1-3-day-so-va-xau/
    suffix-array
```

## 7.5 trie

```
#include <stdio.h>
#include <vector>
using namespace std;

// trie

class trie {
  public :
    struct node {
        int a[64];
        int value;
```

```
        int& operator[] (int i){ return a[i%64]; }
        node() { for (int i=0; i<64; i++) a[i]=0; value=0; }
    };

    vector <node> a;

    int& operator[] (char *s){
        int pos=0, i, c;

        for (i=0; c=s[i]; i++)
        {
            if (a[pos][c]==0) {
                a.push_back(node());
                a[pos][c] = a.size()-1;
            }
            pos=a[pos][c];
        }
        return a[pos].value;
    }

    void clear(){ a.clear(); a.push_back(node()); }
    trie(){ clear(); }
};

trie tr;

// main
int main(){
    int cnt=0;
    char s[2309];
    for(;;){
        gets(s);
        if (tr[s]==0) tr[s]=++cnt;
        printf("%s = %d\n", s, tr[s]);
    }
    return 0;
}

// https://sites.google.com/site/kc97ble/container/trie-cpp
```

# 8 buffer-reader

```
// Buffered reader {{{
//
#include<iostream>
namespace IO {
    const int BUFSIZE = 1<<14;
    char buf[BUFSIZE + 1], *inp = buf;
    bool reacheof;
    char get_char() {
        if (!*inp && !reacheof) {
            memset(buf, 0, sizeof buf);
            int tmp = fread(buf, 1, BUFSIZE, stdin);
            if (tmp != BUFSIZE) reacheof = true;
            inp = buf;
        }
        return *inp++;
    }
    template<typename T>
    T get() {
        int neg = 0;
        T res = 0;
        char c = get_char();
        while (!std::isdigit(c) && c != '-' && c != '+') c =
            get_char();
        if (c == '+') { neg = 0; }
        else if (c == '-') { neg = 1; }
        else res = c - '0';

        c = get_char();
        while (std::isdigit(c)) {
            res = res * 10 + (c - '0');
            c = get_char();
        }
        return neg ? -res : res;
    }
};
// }}}
```

# 9 hash

```
#define long long long
const int N = 1000006, BASE = 1000000007;
int m, n;
char a[N], b[N];
long A[N], B[N], M[N];

void hash_build(char a[], int n, long H[]) {
    for (int i = 1; i <= n; i++)
        H[i] = (H[i - 1] * M[1] + a[i]) % BASE;
}

long hash_range(long H[], int L, int R) {
    return (H[R] - H[L - 1] * M[R - L + 1] + 1LL * BASE *
        BASE) % BASE;
}

// https://sites.google.com/site/kc97ble/1-3-day-so-va-xau/
    hash

struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::
            steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
```

# 10 simd

```
#pragma once

//#pragma GCC target("avx2")
#pragma GCC target("avx2,avx512f,avx512vl")
#pragma GCC optimize("O3")
#pragma GCC optimize("unroll-loops")
#include <immintrin.h>

using m256 = __m256i;

#define ALIGN __attribute__((aligned(64)))

#define SET(x) _mm256_set1_epi32(x)
#define SET64(x) _mm256_set1_epi64x(x)
#define LOAD(p) _mm256_loadu_si256((__m256i*)(p))
#define STORE(p, A) _mm256_storeu_si256((__m256i*)(p), A)

#define AND(a, b) _mm256_and_si256(a, b)
#define OR(a, b) _mm256_or_si256(a, b)
#define XOR(a, b) _mm256_xor_si256(a, b)

#define ADD(a, b) _mm256_add_epi32(a, b)
#define SUB(a, b) _mm256_sub_epi32(a, b)
#define CMP(a, b) _mm256_cmpgt_epi32(a, b)
```

```
#define GETMOD(a, MOD) SUB(a, AND(CMP(a, MOD), MOD))
#define MADD(a, b, MOD) GETMOD(ADD(a, b), MOD)
#define MSUB(a, b, MOD) GETMOD(SUB(ADD(a, MOD), b), MOD)

#define SETLO(a) _mm256_shuffle_epi32(a, 0xA0)
#define SETHI(a) _mm256_shuffle_epi32(a, 0xF9)
#define CAST64(a) AND(a, SET64(0xFFFFFFFF))
#define ADD64(a, b) _mm256_add_epi64(a, b)
```

# 11 template-bak

```
/**
  *   author: delus
**/

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


// Disable this pragma by default because of debugging
// 2 pragma lines give compiler information to use SIMD
     instruction for optimize code.
// #pragma GCC target("avx2")
// #pragma GCC optimize("O3")


#define vi vector<int>
#define vl vector<long long>
#define vb vector<bool>
#define ll long long
#define ii pair<int, int>
#define vii vector<ii>
#define all(x) x.begin(), x.end()
#define FORIT(i, s) for (auto it=(s.begin()); it!=(s.end());
      ++it)
#define F_OR(i, a, b, s) for (int i=(a); (s)>0? i<(int) (b)
     : i > (int) (b); i+=(s))
#define F_OR1(n) F_OR(i, 0, n, 1)
#define F_OR2(i, e) F_OR(i, 0, e, 1)
#define F_OR3(i, b, e) F_OR(i, b, e, 1)
#define F_OR4(i, b, e, s) F_OR(i, b, e, s)
#define GET5(a, b, c, d, e, ...) e
#define F_ORC(...) GET5(__VA_ARGS__, F_OR4, F_OR3, F_OR2,
     F_OR1)
#define FOR(...) F_ORC(__VA_ARGS__)(__VA_ARGS__)
#define FOR1(n) F_OR(i, 1, n+1, 1)
#define EACH(x, a) for(auto& x: a)
#define BUG(x)                  \
    {                           \
        cout << #x << " = " << x; \
    }
#define IO                                  \
    {                                       \
        freopen("input.txt", "r", stdin); \
        freopen("output.txt", "w", stdout); \
    }
#define IOS ios::sync_with_stdio(0); cin.tie(0); cout.tie(0)
     ;
template <class T>
void print(T &x)
{
    for (auto &it : x)
    {
        cout << it << " ";
    }
    cout << "\n";
};
template <class T>
void printPair(T &x)
{
    for (auto &it : x)
    {
        cout << "(" << it.first << ", " << it.second <<") ";
    }
    cout << "\n";
};

int dx[] = {1,1,0,-1,-1,-1, 0, 1};
int dy[] = {0,1,1, 1, 0,-1,-1,-1}; // S,SE,E,NE,N,NW,W,SW
     neighbors

int solve() {

    return 0;
}

int main()
{
    IOS;
    solve();
}
```

# 12 template

```
#include <bits/stdc++.h>
// #include<ext/pb_ds/assoc_container.hpp>
// #include<ext/pb_ds/tree_policy.hpp>
// #include<bits/extc++.h>

using namespace std;
// using namespace __gnu_pbds;
// typedef tree<int, null_type, less<int>, rb_tree_tag,
// tree_order_statistics_node_update> ordered_tree;
// typedef tree<int, null_type,
// less_equal<int>, rb_tree_tag,
     tree_order_statistics_node_update>
// multi_ordered_tree; tree.find_by_order(x); tree.
     order_of_key(x); remove
// element in multi_ordered_tree: tree.erase(--tree.
     lower_bound(x));
#define ar array
#define vt vector
#define all(v) begin(v), end(v)
#define pb push_back
#define ll long long
#define ld long double
#define ii pair<int, int>
#define iii pair<int, ii>
#define vb vt<bool>
#define vc vt<char>
#define vi vt<int>
#define vl vt<ll>
#define vvb vt<vb>
#define vvc vt<vc>
#define vvi vt<vi>
#define vvl vt<vl>
#define vii vt<ii>
#define fi first
#define se second
#define FORIT(i, s) for (auto it = (s.begin()); it != (s.end
     ()); ++it)
#define F_OR(i, a, b, s)
                                                          \
  for (int i = (a); (s) > 0 ? i < (int)(b) : i > (int)(b); i
       += (s))
#define F_OR1(n) F_OR(i, 0, n, 1)
#define F_OR2(i, e) F_OR(i, 0, e, 1)
#define F_OR3(i, b, e) F_OR(i, b, e, 1)
#define F_OR4(i, b, e, s) F_OR(i, b, e, s)
#define GET5(a, b, c, d, e, ...) e
#define F_ORC(...) GET5(__VA_ARGS__, F_OR4, F_OR3, F_OR2,
     F_OR1)
#define FOR(...) F_ORC(__VA_ARGS__)(__VA_ARGS__)
#define FOR1(n) F_OR(i, 1, n + 1, 1)
#define EACH(x, a) for (auto &x : a)
```

```
#define IOS

    \
  ios_base::sync_with_stdio(0);
                                          \
  cin.tie(0);

      \
  cout.tie(0);

const int d4x[] = {-1, 0, 1, 0}, d4y[] = {0, -1, 0, 1},
        d8x[] = {-1, -1, -1, 0, 0, 1, 1, 1},
        d8y[] = {-1, 0, 1, -1, 1, -1, 0, 1};
template <class T> void print(T &x) {
  for (auto &it : x) {
    cerr << it << " ";
  }
  cerr << "\n";
};
template <class T> void printPair(T &x) {
  for (auto &it : x) {
    cerr << "(" << it.first << ", " << it.second << ") ";
  }
  cerr << "\n";
};
int solve() {
   return 0;
}
int main() {
  IOS;
#ifndef ONLINE_JUDGE
  freopen("in", "r", stdin);
  freopen("out", "w", stdout);
#else
  // online submission
#endif

  solve();
  return 0;
}
```

# 13 template1

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

int solve() {

   return 0;
}

int main() {
   ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
   solve();
}
```