Team Notebook

Universidad Mayor de San SimÃ³n - Perritos Malvados

1 A. To Order Nico 3
1.1 mo's in tree's 3
1.2 or statements like 2 sat problem 3
1.3 poly definitions 4
1.4 polynomial sum lazy segtree problem 4
2 Data Structures 4
2.1 custom hash pair 4
2.2 custom hash 5
2.3 disjoint set union 5
2.4 fenwick tree 2d 5
2.5 fenwick tree 5
2.6 general iterative segment tree 5
2.7 general lazy tree 6
2.8 lazy sum tree 7
2.9 min sparse table 7
2.10 mo's hilbert curve 7
2.11 mo's 8
2.12 multiorderedset 8
2.13 orderedset 9
2.14 persistantsegmenttree 9
2.15 priority queue 9
2.16 rope 9
3 Dp 9
3.1 convex hull trick 9
3.2 knuths optimization 10
3.3 linear recurrence cayley halmiton 10
3.4 linear recurrence matrix exponciation 10
3.5 longest increasing subsequence 11
3.6 multiple knacksack optimizacion 11
3.7 optimizing pragmas for bitset 11
3.8 separation optimization 11
4 Flow 11
4.1 hungarian 11
4.2 max flow 12
4.3 min cost max flow 12
4.4 min cut 13
5 Geometry 13
5.1 closest pair of points 13
5.2 convex hull 14
5.3 heron formula 14
5.4 point in convex polygon 14
5.5 point in general polygon 15
5.6 polygon diameter 15
5.7 segment intersection 16
6 Graphs 17
6.1 strongly connected components 17
6.2 topological sort 17
6.3 two sat 17
7 Mateo 18
7.1 2 sat 18
7.2 2d bit 18
7.3 aho corasick 19
7.4 centroid descomposition 19
7.5 chulltrick 19
7.6 closest pair of points 20
7.7 dp dc 20
7.8 dp dc amortizado 20
7.9 dsu rollback 20
7.10 euler walk 21
7.11 hld 21
7.12 implicit segment tree 22
7.13 isomorfismo arboles 22
7.14 khun 23
7.15 knapsack optimization 23
7.16 manacher 23
7.17 mo's on trees 24
7.18 mo's 24
7.19 parallel binary search 24
7.20 parallel dsu 25
7.21 puentes 25
7.22 puntos de articulacion 25
7.23 sort c clockwise 26
7.24 sos dp 26
7.25 suffix array nuevo 26
8 Math 26
8.1 catalan convolution 26
8.2 catalan 27
8.3 combinatorics 27
8.4 count primes with pi function 27
8.5 fast fibonacci 28
8.6 floor sums 28
8.7 optimized polard rho 28
8.8 subfactorial 29
8.9 ternary search 29
9 Strings 29
9.1 fast hashing 29
9.2 kmp automaton 29
9.3 kmp 29
9.4 todo add manacher, ahocorasick, suffix array .
30
10 Tree 30
10.1 euler tour sum 30
10.2 k th parent 30

10.3 lowest common ancestor ... 30
10.4 todo add dsu, centroid, hld ... 31
11 Utils ... 31
11.1 bit tricks ... 31
11.2 io int128 ... 31
11.3 pragmas ... 31
11.4 randoms ... 31
11.5 string streams ... 31

1 A. To Order Nico

1.1 mo's in tree's

```
#include <bits/stdc++.h>
using namespace std;
typedef vector<int> vi;
typedef vector<vi> vvi;

map<int, int> getID;
map<int, int>::iterator it;

const int LOGN = 20;
int id, bs, N;
int counter[50050];
int A[50050], P[100050];
int res[100050];
int st[50050], ed[50050];
int DP[20][50050], level[50050];
bool flag[50050];
bool seen[50050];
vvi edges;

struct Q {
   int l, r, p, id;
   bool operator < (const Q& other) const {
      return (l / bs < other.l / bs || (l / bs == other.l /
bs && r < other.r));
   }//operator <
} q[100050];

void DFS0(const int u) {
   seen[u] = 1;
   P[id] = u;
   st[u] = id++;
   for (auto& e : edges[u]) {
      if (!seen[e]) {
         DP[0][e] = u;
         level[e] = level[u] + 1;
         DFS0(e);
      }//if
   }//for
   P[id] = u;
   ed[u] = id++;
}//DFS0

void prep(const int r) {
   level[r] = 0;
   for (int i = 0; i < LOGN; i++)
      DP[i][r] = r;
   id = 0;
   DFS0(r);
   for (int i = 1; i < LOGN; i++)
      for (int j = 1; j <= N; j++)
         DP[i][j] = DP[i - 1][DP[i - 1][j]];
}//prep

int LCA(int a, int b) {
   if (level[a] > level[b])
      swap(a, b);
   int diff = level[b] - level[a];
   for (int i = 0; i < LOGN; i++)
      if (diff & (1 << i))
         b = DP[i][b]; //move 2^i parents upwards
   if (a == b)
      return a;
   for (int i = LOGN - 1; i >= 0; i--)
      if (DP[i][a] != DP[i][b])
         a = DP[i][a], b = DP[i][b];
   return DP[0][a];
}//LCA

int main() {
   int Q, n1, n2, L, R, a, v = 1, tot;
   scanf("%d %d", &N, &Q);
   edges.assign(N + 5, vi());
   bs = sqrt(N);
   for (int i = 1; i <= N; i++) {
      scanf("%d", &a);
      A[i] = ((it = getID.find(a)) != getID.end()) ? it-
>second : (getID[a] = v++);
   }//for
   for (int i = 0; i < N - 1; i++) {
      scanf("%d %d", &n1, &n2);
      edges[n1].push_back(n2);
      edges[n2].push_back(n1);
   }//for
   prep(1);
   for (int i = 0; i < Q; i++) {
      scanf("%d %d", &n1, &n2);
      if (st[n1] > st[n2])
         swap(n1, n2);
      q[i].p = LCA(n1, n2);
      if (q[i].p == n1)
         q[i].l = st[n1], q[i].r = st[n2];
      else
         q[i].l = ed[n1], q[i].r = st[n2];
      q[i].id = i;
   }//for
   sort(q, q + Q);
   L = 0; R = -1; tot = 0;
   for (int i = 0; i < Q; i++) {
      while (R < q[i].r) {
         if (!flag[P[++R]])
            tot += (++counter[A[P[R]]] == 1);
         else
            tot -= (--counter[A[P[R]]] == 0);
         flag[P[R]] = !flag[P[R]];
      }//while
      while (R > q[i].r) {
         if (!flag[P[R]])
            tot += (++counter[A[P[R]]] == 1);
         else
            tot -= (--counter[A[P[R]]] == 0);
         flag[P[R]] = !flag[P[R]];
         R--;
      }//while
      while (L < q[i].l) {
         if (!flag[P[L]])
            tot += (++counter[A[P[L]]] == 1);
         else
            tot -= (--counter[A[P[L]]] == 0);
         flag[P[L]] = !flag[P[L]];
         L++;
      }//while
      while (L > q[i].l) {
         if (!flag[P[--L]])
            tot += (++counter[A[P[L]]] == 1);
         else
            tot -= (--counter[A[P[L]]] == 0);
         flag[P[L]] = !flag[P[L]];
      }//while
      res[q[i].id] = tot + (q[i].p != P[q[i].l] && !
counter[A[q[i].p]]);
   }//for
   for (int i = 0; i < Q; i++)
      printf("%d\n", res[i]);
   return 0;
}//main
```

1.2 or statements like 2 sat problem

```
// Return the smaller lexicographic array of size n that
satities a_i | a_j = z
// a_i | a_i = z is allowed.
// there must exists a solution.
vector<ll> f(ll n, vector<tuple<ll,ll,ll>> &statements) {
    ll m = statements.size();
    vector<vector<pair<ll,ll>>> adj(n + 1);
    const ll bits = 30;
    vector<ll> taken(n+1, (1 << bits) - 1), answer(n+1, (1
<< bits) - 1);
    for (int i = 0; i < m; i++) {
        ll x, y, z;
        tie(x, y, z) = statements[i];

        answer[x] &= z;
        answer[y] &= z;
        if (x == y) {
            taken[x] = 0;
            continue;
        }
        taken[x] &= z;
        taken[y] &= z;
        adj[x].pb({y, z});
        adj[y].pb({x, z});
    }
    for (int x = 1; x <= n; x++) {
        for (int i = 0; i < bits; i++) {
            if (!((taken[x] >> i) & 1)) continue;
            ll allHave = true;
            for (auto y : adj[x]) {
                if ((y.S >> i) & 1) {
                    allHave &= ((taken[y.F] >> i) & 1) ||
((answer[y.F] >> i) & 1);
                }
            }
            taken[x] -= 1 << i;
            if (allHave) {
                answer[x] -= 1 << i;
                for (auto y : adj[x]) {
                    if ((y.S >> i) & 1) {
                        taken[y.F] |= 1 << i;
                        taken[y.F] ^= 1 << i;
                    }
                }
            }
        }
    }
    answer.erase(answer.begin());
    return answer;
}
```

1.3 poly definitions

```
A(x) = Sum i=0 to n ( a_i * x^i )  y B(x) Sum i=0 to m ( b_i
* x^i)

A(x)*B(x) Sum i=0 to (n+m) Sum j=0 to (n+m)  (a_j)*(b_i-
j) x^i

const ld PI = acos(-1);
```

1.4 polynomial sum lazy segtree problem

```
/* Polynomial Queries, queries
1. Increase [a,b] by 1, second by 2, third by 3, and so on
2. Sum of [a,b]
Use:

cin >> nums[i],tree.update(i, { nums[i] })
For 1: tree.apply(l,r,{0,1});
For 2: tree.query(l,r).sum
*/
struct Node { ll sum = 0; };
struct Func { ll add, ops; };
Node e() { return {0}; };
Func id() { return {0, 0}; }
Node op(Node a, Node b) { return {a.sum + b.sum }; }
ll f(ll x) { return x * (x+1)/2; }
Node mapping(Node node, Func lazy, ll sz) {
    return { node.sum + sz*lazy.add + lazy.ops*f(sz) };
}
Func composicion(Func prev, Func actual) {
    Func ans = { prev.add + actual.add, prev.ops +
actual.ops };
    return ans;
}
Func sumF(Func f, ll x) { return {f.add + x*f.ops,
f.ops }; }

struct lazytree {
    int n;
    vector<Node> nodes;
    vector<Func> lazy;
    void init(int nn) {
        n = nn;
        int size = 1;
        while (size < n) { size *= 2; }
        ll m = size * 2;
        nodes.assign(m, e());
        lazy.assign(m, id());
    }

    void push(int i, int sl, int sr) {
        nodes[i] = mapping(nodes[i], lazy[i], sr-sl+1);
        if (sl != sr) {
            ll cnt = (sr+sl)/2-sl+1; // changed
            lazy[i * 2 + 1] =
composicion(lazy[i*2+1],lazy[i]);
            lazy[i * 2 + 2] =
composicion(lazy[i*2+2],sumF(lazy[i],cnt));
        }
        lazy[i] = id();
    }

    void apply(int i, int sl, int sr, int l, int r, Func f)
{
        push(i, sl, sr);
        if (l <= sl && sr <= r) {
            lazy[i] = sumF(f, abs(sl-l)); //Changed
            push(i,sl,sr);
        } else if (sr < l || r < sl) {
        } else {
            int mid = (sl + sr) >> 1;
            apply(i * 2 + 1, sl, mid, l, r, f);
            apply(i * 2 + 2, mid + 1, sr, l, r, f);
            nodes[i] = op(nodes[i*2+1],nodes[i*2+2]);
        }
    }

    void apply(int l, int r, Func f) {
        assert(l <= r);
        assert(r < n);
        apply(0, 0, n - 1, l, r, f);
    }

    void update(int i, Node node) {
        assert(i < n);
        update(0, 0, n-1, i, node);
    }

    void update(int i, int sl, int sr, int pos, Node node) {
        if (sl <= pos && pos <= sr) {
            push(i,sl,sr);
            if (sl == sr) {
                nodes[i] = node;
            } else {
                int mid = (sl + sr) >> 1;
                update(i * 2 + 1, sl, mid, pos, node);
                update(i * 2 + 2, mid + 1, sr, pos, node);
                nodes[i] = op(nodes[i*2+1], nodes[i*2+2]);
            }
        }
    }

    Node query(int i, int sl, int sr, int l, int r) {
        push(i,sl,sr);
        if (l <= sl && sr <= r) {
            return nodes[i];
        } else if (sr < l || r < sl) {
            return e();
        } else {
            int mid = (sl + sr) >> 1;
            auto a = query(i * 2 + 1, sl, mid, l, r);
            auto b = query(i * 2 + 2, mid + 1, sr, l, r);
            return  op(a,b);
        }
    }

    Node query(int l, int r) {
        assert(l <= r);
        assert(r < n);
        return query(0, 0, n - 1, l, r);
    }
};
```

2 Data Structures

2.1 custom hash pair

```
// Example: unordered_set<pair<ll,ll>, HASH> exists;
// It's better to convine with other custom hash
struct HASH{
  size_t operator()(const pair<ll,ll>&x)const{
    return hash<ll>()(((ll)x.first)^(((ll)x.second)<<32));
  }
};
```

2.2 custom hash

```
// Avoid hashing hacks and improve performance of hash
structures
// e.g. unordered_map<ll,ll,custom_hash>
struct custom_hash {
    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().count();
        x ^= FIXED_RANDOM;
        return x ^ (x >> 16);
    }
};

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);
    }
};
```

2.3 disjoint set union

```
/* Disjoint Set Union
This uses union by rank, it is longer to code but faster
than normal Disjoint Set Union. The time complexity per
operation
is faster than O(log n)

It finds if 2 nodes in a graph are connected or not (have
same 'find(x)' value),
and the size of the connected component, the graph starts
with no edges,
then you can add edges with 'group(nodeA, nodeB)' method.
*/

struct union_find {
    vi link, score, size;
    int n;
    void init(int nn) {
        link.resize(nn);
        score.resize(nn);
        size.resize(nn);
```

```
        this->n = nn;
        for (int i = 0; i < n; i++) {
            link[i] = i;
            score[i] = 0;
            size[i] = 1;
        }
    }
    int find(int x) {
        if (link[x] == x) return x;
        return (link[x] = find(link[x]));
    }
    void group(int a, int b) {
        int pa = find(a);
        int pb = find(b);
        if (pa != pb) {
            if (score[pa] >= score[pb]) {
                link[pb] = pa;
                size[pa] += size[pb];
                if (score[pa] == score[pb]) score[pa]++;
            } else {
                link[pa] = pb;
                size[pb] += size[pa];
            }
        }
    }
};
```

2.4 fenwick tree 2d

```
struct BIT2D { // 1-indexed
    vector<vl> bit;
    ll n, m;

    BIT2D(ll n, ll m) : bit(n+1, vl(m+1)), n(n), m(m) {}

    ll lsb(ll i) { return i & -i; }

    void add(int row, int col, ll x) {
        for (int i = row;i<=n;i+=lsb(i))
            for (int j = col;j<=m;j+=lsb(j))
                bit[i][j] += x;
    }

    ll sum(int row, int col) {
        ll res = 0;
        for (int i = row;i>0;i-=lsb(i))
            for (int j = col;j>0;j-=lsb(j))
                res += bit[i][j];
        return res;
    }

    ll sum(int x1, int y1, int x2, int y2) {
        return sum(x2,y2) - sum(x1-1,y2) - sum(x2,y1-1) +
sum(x1-1,y1-1);
    }
```

```

    void set(int x, int y, ll val) {
        add(x,y,val-sum(x,y,x,y));
    }
};
```

2.5 fenwick tree

```
struct FenwickTree {
    vector<int> bit;
    int n;

    FenwickTree(int n) {
        this->n = n;
        bit.assign(n, 0);
    }

    FenwickTree(vector<int> a) : FenwickTree(a.size()) {
        for (size_t i = 0; i < a.size(); i++)
            add(i, a[i]);
    }

    int sum(int r) {
        int ret = 0;
        for (; r >= 0; r = (r & (r + 1)) - 1)
            ret += bit[r];
        return ret;
    }

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

    void add(int idx, int delta) {
        for (; idx < n; idx = idx | (idx + 1))
            bit[idx] += delta;
    }

    // TODO: Lower Bound
};
```

2.6 general iterative segment tree

```
// >>>>>>>>> Implement
struct Node { ll x = 0; };

Node e() { return Node(); } // null element

Node op(Node &a, Node &b) { // operation
    Node c;
    c.x = a.x + b.x;
    return c;
}
// <<<<<<<
```

```
struct segtree {
    vector<Node> t;
    ll n;

    void init(int n) {
        t.assign(n * 2, e());
        this->n = n;
    }

    void init(vector<Node>& s) {
        n = s.size();
        t.assign(n * 2, e());
        for (int i = 0; i < n; i++) {
            t[i+n] = s[i];
        }
        build();
    }

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

    // set value at position p
    void update(int p, const Node& value) {
        for (t[p += n] = value; p >>= 1; ) t[p] =
op(t[p<<1], t[p<<1|1]);
    }

    // sum on interval [l, r]
    Node query(int l, int r) {
        r++; // make this inclusive
        Node resl=e(), resr=e(); // null element
        for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
            if (l&1) resl = op(resl, t[l++]);
            if (r&1) resr = op(t[--r], resr);
        }
        return op(resl, resr);
    }

    Node get(int i) {
        return query(i, i); // improve to o(1)
    }

    // TODO: implement left, right binary search
};
```

2.7 general lazy tree

```
// TODO: Make this iterative
struct Node { // Structure
    ll mn;
    ll size = 1;

    Node(ll mn):mn(mn) {
    }
};

struct Func { // Applied function
    ll a = 0;
};

Node e() { // op(x, e()) = x
    Node a(INT64_MAX); // neutral element
    return a;
};

Func id() { // mapping(x, id()) = x
    Func l = {0}; // identify func
    return l;
}

Node op(Node &a, Node &b) { // associative property
    Node c = e();           // binary operation
    c.size = a.size + b.size;
    c.mn = min(a.mn, b.mn);
    return c;
}

Node mapping(Node node, Func &lazy) {
    node.mn += lazy.a; // appling function
    return node;
}

Func composicion(Func &prev, Func &actual) {
    prev.a = prev.a + actual.a; // composing funcs
    return prev;
}

struct lazytree {
    int n;
    vector<Node> nodes;
    vector<Func> lazy;

    void init(int nn) {
        n = nn;
        int size = 1;
        while (size < n) {
            size *= 2;
        }
        ll m = size *2;
        nodes.assign(m, e());
        lazy.assign(m, id());
    }

    void push(int i, int sl, int sr) {
        nodes[i] = mapping(nodes[i], lazy[i]);
        if (sl != sr) {
            lazy[i * 2 + 1] =
composicion(lazy[i*2+1],lazy[i]);
            lazy[i * 2 + 2] =
composicion(lazy[i*2+2],lazy[i]);
        }
        lazy[i] = id();
    }

    void apply(int i, int sl, int sr, int l, int r, Func f)
{
        push(i, sl, sr);
        if (l <= sl && sr <= r) {
            lazy[i] = f;
            push(i,sl,sr);
        } else if (sr < l || r < sl) {
        } else {
            int mid = (sl + sr) >> 1;
            apply(i * 2 + 1, sl, mid, l, r, f);
            apply(i * 2 + 2, mid + 1, sr, l, r, f);
            nodes[i] = op(nodes[i*2+1],nodes[i*2+2]);
        }
    }

    void apply(int l, int r, Func f) {
        assert(l <= r);
        assert(r < n);
        apply(0, 0, n - 1, l, r, f);
    }

    void update(int i, Node node) {
        assert(i < n);
        update(0, 0, n-1, i, node);
    }

    void update(int i, int sl, int sr, int pos, Node node) {
        if (sl <= pos && pos <= sr) {
            push(i,sl,sr);
            if (sl == sr) {
                nodes[i] = node;
            } else {
                int mid = (sl + sr) >> 1;
                update(i * 2 + 1, sl, mid, pos, node);
                update(i * 2 + 2, mid + 1, sr, pos, node);
                nodes[i] = op(nodes[i*2+1], nodes[i*2+2]);
            }
        }
    }

    Node query(int i, int sl, int sr, int l, int r) {
        push(i,sl,sr);
        if (l <= sl && sr <= r) {
            return nodes[i];
        } else if (sr < l || r < sl) {
            return e();
        } else {
            int mid = (sl + sr) >> 1;
            auto a = query(i * 2 + 1, sl, mid, l, r);
            auto b = query(i * 2 + 2, mid + 1, sr, l, r);
            return  op(a,b);
        }
    }

    Node query(int l, int r) {
        assert(l <= r);
        assert(r < n);
        return query(0, 0, n - 1, l, r);
    }
};
```

2.8 lazy sum tree

```
struct lazytree {
    int n;
    vl sum;
    vl lazySum;

    void init(int nn) {
        sum.clear();
        n = nn;
        int size = 1;
        while (size < n) {
            size *= 2;
        }
        sum.resize(size * 2);
        lazySum.resize(size * 2);
    }

    void update(int i, int sl, int sr, int l, int r, ll
diff) {
        if (lazySum[i]) {
            sum[i] += (sr - sl + 1) * lazySum[i];
            if (sl != sr) {
                lazySum[i * 2 + 1] += lazySum[i];
                lazySum[i * 2 + 2] += lazySum[i];
            }
            lazySum[i] = 0;
        }
        if (l <= sl && sr <= r) {
            sum[i] += (sr - sl + 1) * diff;
            if (sl != sr) {
                lazySum[i * 2 + 1] += diff;
                lazySum[i * 2 + 2] += diff;
            }
        } else if (sr < l || r < sl) {
        } else {
            int mid = (sl + sr) >> 1;
            update(i * 2 + 1, sl, mid, l, r, diff);
            update(i * 2 + 2, mid + 1, sr, l, r, diff);
            sum[i] = sum[i * 2 + 1] + sum[i * 2 + 2];
        }
    }
```

```
    void update(int l, int r, ll diff) {
        assert(l <= r);
        assert(r < n);
        update(0, 0, n - 1, l, r, diff);
    }

    ll query(int i, int sl, int sr, int l, int r) {
        if (lazySum[i]) {
            sum[i] += lazySum[i] * (sr - sl + 1);
            if (sl != sr) {
                lazySum[i * 2 + 1] += lazySum[i];
                lazySum[i * 2 + 2] += lazySum[i];
            }
            lazySum[i] = 0;
        }
        if (l <= sl && sr <= r) {
            return sum[i];
        } else if (sr < l || r < sl) {
            return 0;
        } else {
            int mid = (sl + sr) >> 1;
            return query(i * 2 + 1, sl, mid, l, r) + query(i
* 2 + 2, mid + 1, sr, l, r);
        }
    }

    ll query(int l, int r) {
        assert(l <= r);
        assert(r < n);
        return query(0, 0, n - 1, l, r);
    }
};
```

2.9 min sparse table

```
using Type = int;

// Gets the minimum in a range [l,r] in O(1)
// Preprocesing is O(n log n)
struct min_sparse {

    int log;
    vector<vector<Type>> sparse;

    void init(vector<Type> &nums) {
        int n = nums.size();
        log = 0;
        while (n) log++, n/=2;
        n = nums.size();
        sparse.assign(n, vector<Type>(log, 0));
        for (int i = 0; i < n; i++) sparse[i][0] = nums[i];
        for (int l = 1; l < log; l++) {
            for (int j = 0; j + (1 << l) - 1 < n; j++) {
                sparse[j][l] = min(sparse[j][l-1],
sparse[j+(1 << (l-1))][l-1]);
```

```
            }
        }
    }

    Type query(int x, int y) {
        int n = y - x + 1;
        int logg = -1;
        while (n) logg++, n/=2; // TODO: improve this with
fast builtin
        return min(sparse[x][logg], sparse[y-(1 << logg)+1]
[logg]);
    }
};
```

2.10 mo's hilbert curve

```
/*
Hilbert Curve for Mo's,

This is a better ordering for Mo (review Mo's algorithm),
sometimes it take the half time, other times it takes
more time. So it needs a revision to identify when it's
better!!!
https://codeforces.com/blog/entry/61203
*/

inline int64_t gilbertOrder(int x, int y, int pow, int
rotate) {
  if (pow == 0) {
    return 0;
  }
  int hpow = 1 << (pow-1);
  int seg = (x < hpow) ? (
    (y < hpow) ? 0 : 3
  ) : (
    (y < hpow) ? 1 : 2
  );
  seg = (seg + rotate) & 3;
  const int rotateDelta[4] = {3, 0, 0, 1};
  int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
  int nrot = (rotate + rotateDelta[seg]) & 3;
  int64_t subSquareSize = int64_t(1) << (2*pow - 2);
  int64_t ans = seg * subSquareSize;
  int64_t add = gilbertOrder(nx, ny, pow-1, nrot);
  ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add
- 1);
  return ans;
}

struct Query {
  int l, r, idx;
  int64_t ord;

  inline void calcOrder() {
    ord = gilbertOrder(l, r, 21, 0); // n,q <= 1e5
  }
};

inline bool operator<(const Query &a, const Query &b) {
  return a.ord < b.ord;
}
```

2.11 mo's

```
/*
Mo's algorithm O((N + Queries)*SQRT(N))

It answers queries offline using SQRT Descomposition,
sometimes get TLE when you have Set, Map, various iteration
when adding,
So you need to implement efficiently (with arrays) in O(1).
You can answer
questions like in a range, the number of distincts values.
*/
const int BLOCK_SIZE = 430; // For se 1e5=310, for 2e5=430

struct query {
    int l, r, idx;

    bool operator <(query &other) const {
        return MP(l / BLOCK_SIZE, r) < MP(other.l /
BLOCK_SIZE, other.r);
    }
};


void add(int idx);
void remove(int idx);
ll getAnswer();

vector<ll> mo(vector<query> queries) {
    vector<ll> answers(queries.size());
    int l = 0;
    int r = -1;
    sort(all(queries));
    each(q, queries) {
        while (q.l < l) add(--l);
        while (r < q.r) add(++r);
        while (l < q.l) remove(l++);
        while (q.r < r) remove(r--);
        answers[q.idx] = getAnswer();
    }
    return answers;
}

vl nums; //init
ll ans = 0;
int cnt[1000001];
void add(int idx) {
  // update ans, when adding an element
}
void remove(int idx) {
  // update ans, when removing an element
}
ll getAnswer() { return ans; }
```

2.12 multiorderedset

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

using namespace __gnu_pbds;

struct multiordered_set {
    tree<ll,
        null_type,
        less_equal<ll>, // this is the trick
        rb_tree_tag,
        tree_order_statistics_node_update> oset;

    //this function inserts one more occurrence of (x) into
the set.
    void insert(ll x) {
        oset.insert(x);
    }

    //this function checks weather the value (x) exists in
the set or not.
    bool exists(ll x) {
        auto it = oset.upper_bound(x);
        if (it == oset.end()) {
            return false;
        }
        return *it == x;
    }

    //this function erases one occurrence of the value (x).
    void erase(ll x) {
        if (exists(x)) {
            oset.erase(oset.upper_bound(x));
        }
    }

    //this function returns the value at the index (idx)..(0
indexing).
    ll find_by_order(ll pos) {
        return *(oset.find_by_order(pos));
    }

    //this function returns the first index of the value
(x)..(0 indexing).
    int first_index(ll x) {
        if (!exists(x)) {
            return -1;
        }
        return (oset.order_of_key(x));
    }

    //this function returns the last index of the value
(x)..(0 indexing).
    int last_index(ll x) {
        if (!exists(x)) {
            return -1;
        }
        if (find_by_order(size() -1) == x) {
            return size() - 1;
        }
        return first_index(*oset.lower_bound(x)) -1;
    }

    //this function returns the number of occurrences of the
value (x).
    int count(ll x) {
        if (!exists(x)) {
            return -1;
        }
        return last_index(x) - first_index(x) + 1;
    }

    // Count the numbers between [l, r]
    int count(ll l, ll r) {
        auto left = oset.upper_bound(l);
        if (left == oset.end() || *left>r) {
            return 0;
        }
        auto right = oset.upper_bound(r);
        if (right != oset.end()) {
            right =
oset.find_by_order(oset.order_of_key(*right));
        }
        if (right == oset.end() || *right >r) {
            if (right == oset.begin()) return 0;
            right--;
        }
        return last_index(*right)-first_index(*left)+1;
    }

    //this function clears all the elements from the set.
    void clear() {
        oset.clear();
    }

    //this function returns the size of the set.
    ll size() {
        return (ll)oset.size();
    }
};
```

2.13 orderedset

```
// Same as set, but you can get the order of an element or
the
// element in given sorted position in O(log n)
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

#define oset tree<ll, null_type,less<ll>,
rb_tree_tag,tree_order_statistics_node_update>
//find_by_order(k) order_of_key(k)
```

2.14 persistantsegmenttree

```
struct Vertex {
    Vertex *l, *r;
    ll sum;
    Vertex(int val) : l(nullptr), r(nullptr), sum(val) {}
    Vertex(Vertex *l, Vertex *r) : l(l), r(r), sum(0) {
        if (l) sum += l->sum;
        if (r) sum += r->sum;
    }
};
Vertex* build(vector<ll>& a, int tl, int tr) {
    if (tl == tr)
        return new Vertex(a[tl]);
    int tm = (tl + tr) / 2;
    return new Vertex(build(a, tl, tm), build(a, tm+1, tr));
}
ll get_sum(Vertex* v, int tl, int tr, int l, int r) {
    if (l > r)
        return 0;
    if (l == tl && tr == r)
        return v->sum;
    int tm = (tl + tr) / 2;
    return get_sum(v->l, tl, tm, l, min(r, tm))
         + get_sum(v->r, tm+1, tr, max(l, tm+1), r);
}
Vertex* update(Vertex* v, int tl, int tr, int pos, int
new_val) {
    if (tl == tr)
        return new Vertex(new_val);
    int tm = (tl + tr) / 2;
    if (pos <= tm)
        return new Vertex(update(v->l, tl, tm, pos,
new_val), v->r);
    else
        return new Vertex(v->l, update(v->r, tm+1, tr, pos,
new_val));
}
Vertex* build(vector<ll> &a) {
    return build(a,0,a.size()-1);
}
ll get_sum(Vertex *v,ll n,int l, int r) {
    return get_sum(v,0,n-1,l,r);
}
Vertex* update(Vertex* v,ll n, int pos, int newV) {
    return update(v,0,n-1,pos,newV);
}
Vertex* copy(Vertex* v) {
    return new Vertex(v->l,v->r);
}
```

2.15 priority queue

```
template<class T> using pql =
priority_queue<T,vector<T>,greater<T>>;// less first
template<class T> using pqg = priority_queue<T>; // greater
first
```

2.16 rope

```
/**
 * Description: insert element at $i$-th position, cut a
substring and
  * re-insert somewhere else. At least 2 times slower than
handwritten treap.
 * Time: O(\log N) per operation? not well tested
 * Source: https://codeforces.com/blog/entry/10355
 * Verification: CEOI 2018 Day 2 Triangles
  * https://szkopul.edu.pl/problemset/problem/AzKAZ2
RDiVTjeWSBolwoC5zl/site/?key=statement
  * vector is faster for this problem ...
 */

#include <ext/rope>
using namespace __gnu_cxx;
void ropeExample() {
  rope<int> v(5,0); // initialize with 5 zeroes
  FOR(i,sz(v)) v.mutable_reference_at(i) = i+1;
  FOR(i,5) v.pb(i+1); // constant time pb
  rope<int> cur = v.substr(1,2);
  v.erase(1,3); // erase 3 elements starting from 1st
element
  for (rope<int>::iterator it = v.mutable_begin();
    it != v.mutable_end(); ++it) pr((int)*it,' ');
  ps(); // 1 5 1 2 3 4 5
  v.insert(v.mutable_begin()+2,cur); // or just 2
  v += cur; FOR(i,sz(v)) pr(v[i],' ');
  ps(); // 1 5 2 3 1 2 3 4 5 2 3
}

/*
Tested in cses https://cses.fi/problemset/task/1749/
https://cses.fi/problemset/result/10105636/ 2 times slower
than ordered set
It's better to create rope with an given size and value

TLE with 2*10^5
Need revision!!

push_back() - O(log N).
pop_back() - O(log N)
insert(int x, crope r1): O(log N) and Worst O(N)
substr(int x, int l): O(log N)
replace(int x, int l, crope r1): O(log N).
*/
```

3 Dp

3.1 convex hull trick

```
/**
 * Author: Simon Lindholm
 * Description: Container where you can add lines of the
form kx+m, and query maximum values at points x.
 *  Useful for dynamic programming (``convex hull trick'').
 * Time: O(\log N)
 * Status: stress-tested
 */

 // For minimum you can multiply by -1 'k' and 'm' when
adding, and the answer when querying.
 // Tested in https://atcoder.jp/contests/dp/submissions/
55836691
#pragma once

struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y =
erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

3.2 knuths optimization

```
/*
Knuth's Optimization

For dp[i][j] = min_{i<=k<j} dp[i][k] + dp[k+1][j] + cost[i]
[j]

Optimizing that from O(n^3) to O(n^2), it's required to hold
the following:
opt[i][j] = optimal splitting point of dp[i][j], the 'k' the
minimizes the above definition

opt[i][j-1] <= opt[i][j] <= opt[i+1][j]

You can demostrate that by the following:

a<=b<=c<=d
cost(b,c) <= cost(a,d), // an contained interval is <= of
the interval
cost(a,c)+cost(b,d) <= cost(a,d)+cost(b,c) // partial
intersection <= total intersection

Complexity: O(n^2)
*/
void test_case() {
    cin >> n;nums.assign(n,0);pf.assign(n+1,0); // READ
input!!!
    for (int i =0;i<n;i++) cin>>nums[i], pf[i+1] = pf[i] +
nums[i];

    for (int i = 0;i<n;i++) { // base case
        dp[i][i] = 0; // depends of the dp!!!!
        opt[i][i] = i;
    }

    for (int i =n-2;i>=0;i--) {
        for (int j = i+1;j<n;j++) {
            dp[i][j] = inf; // set to inf, any option is
better, or use -1 or a flag!!
            ll cost = sum(i,j); // depends of problem
            for (int k = opt[i][j-1]; k<= min(j-1ll,opt[i+1]
[j]);k++) {
                ll actual = dp[i][k] + dp[k+1][j] + cost;
                if (actual < dp[i][j]) { // if flag '-1'
used, change here!!
                    dp[i][j] = actual;
                    opt[i][j] = k;
                }
            }
        }
    }

    cout << dp[0][n-1] << "\n";
}
```

3.3 linear recurrence cayley halmiton

```
// O(N^2 log K)

const int mod = 1e9 + 7;

template <int32_t MOD>
struct modint {
  int32_t value;
  modint() = default;
  modint(int32_t value_) : value(value_) {}
  inline modint<MOD> operator + (modint<MOD> other) const
{ int32_t c = this->value + other.value; return
modint<MOD>(c >= MOD ? c - MOD : c); }
  inline modint<MOD> operator * (modint<MOD> other) const
{ int32_t c = (int64_t)this->value * other.value % MOD;
return modint<MOD>(c < 0 ? c + MOD : c); }
  inline modint<MOD> & operator += (modint<MOD> other)
{ this->value += other.value; if (this->value >= MOD) this-
>value -= MOD; return *this; }
  modint<MOD> pow(uint64_t k) const {
    modint<MOD> x = *this, y = 1;
    for (; k; k >>= 1) {
      if (k & 1) y *= x;
      x *= x;
    }
    return y;
  }
  modint<MOD> inv() const { return pow(MOD - 2); }  // MOD
must be a prime
};

using mint = modint<mod>;

vector<mint> combine (int n, vector<mint> &a, vector<mint>
&b, vector<mint> &tr) {
  vector<mint> res(n * 2 + 1, 0);
  for (int i = 0; i < n + 1; i++) {
    for (int j = 0; j < n + 1; j++) res[i + j] += a[i] *
b[j];
  }
  for (int i = 2 * n; i > n; --i) {
    for (int j = 0; j < n; j++) res[i - 1 - j] += res[i] *
tr[j];
  }
  res.resize(n + 1);
  return res;
};
// transition -> for(i = 0; i < x; i++) f[n] += tr[i] * f[n-
i-1]
// S contains initial values, k is 0 indexed
mint LinearRecurrence(vector<mint> &S, vector<mint> &tr,
long long k) {
  int n = S.size(); assert(n == (int)tr.size());
  if (n == 0) return 0;
  if (k < n) return S[k];
  vector<mint> pol(n + 1), e(pol);
  pol[0] = e[1] = 1;
  for (++k; k; k /= 2) {
    if (k % 2) pol = combine(n, pol, e, tr);
    e = combine(n, e, e, tr);
  }
  mint res = 0;
  for (int i = 0; i < n; i++) res += pol[i + 1] * S[i];
  return res;
}

void test_case() {
    ll n;
    cin >> n; // Fibonacci
    vector<mint> initial = {0, 1}; // F0, F1
    vector<mint> tr = {1, 1};
    cout << LinearRecurrence(initial, tr, n).value << "\n";
}
```

3.4 linear recurrence matrix exponciation

```
// Solves F_n = C_n-1 * F_n-1 + ... + C_0*F_0 + p + q*n +
r*n^2
// O((n+3)^3*log(k))
// Also solves (k steps)-min path of a matrix in same
complexity
// Tested and for more details see: https://codeforces.com/
blog/entry/80195
const int MOD = 1e9 + 7;
const int N = 10 + 3; // 10 is MAX N, 3 is for p,q,r
inline ll add(ll x, ll y) { return (x+y)%MOD; }
inline ll mul(ll x, ll y) { return (x*y)%MOD; }

// const ll inf = ll(1e18) + 5; // for k-min path
struct Mat {
    array<array<ll,N>,N> mt;
    Mat(bool id=false) {
        for (auto &x : mt) fill(all(x),0);
        if (id) for (int i=0;i<N;i++) mt[i][i]=1;
  //for (auto &x : mt) fill(all(x),inf); // For k-min path
        //if (id) for (int i =0;i<N;i++) mt[i][i]=0;
    }
    inline Mat operator * (const Mat &b) {
        Mat ans;
        for (int k=0;k<N;k++)for(int i=0;i<N;i++)for(int
j=0;j<N;j++)
                ans.mt[i][j]=add(ans.mt[i][j],mul(mt[i]
[k],b.mt[k][j]));
                //ans.mt[i][j] = min(ans.mt[i][j],mt[i]
[k]+b.mt[k][j]); // For K-min Path

        return ans;
    }
    inline Mat pow(ll k) {
        Mat ans(true),p=*this; // Note '*'!!
        while (k) {
            if (k&1) ans = ans*p;
            p=p*p;
            k>>=1;
        }
        return ans;
    }
    string db() { // Optional for debugging
        string ans;
        for (int i =0;i<mt.size();i++)for (int
j=0;j<mt.size();j++)
                ans +=to_string(mt[i][j]),ans+="
\n"[j==N-1];
        return "\n"+ans;
    }
};

// Important!!! Remember to set N = MAX_N + 3
// Solves F_n = C_n-1 * F_n-1 + ... + C_0*F_0 + p + q*n +
r*n^2
// f = {f_0,f_1,f_2,f_3,...,f_n}
// c = {c_0,c_1,c_2,c_3,...,c_n}
ll fun(vl f, vl c, ll p, ll q, ll r, ll k) {
    ll n = c.size();
    if (k < n) return f[k];
    reverse(all(c)),reverse(all(f));
    Mat mt,st;
    for (int i = 0;i<n;i++) mt.mt[0][i]=c[i];
    for (int i = 1;i<n;i++) mt.mt[i][i-1]=1;
    for (int i = 0;i<n;i++) st.mt[i][0]=f[i];

    vl extra = {p,q,r}; // To extend here with
1*p,i*q,i*i*r,etc
    for (int i=0;i<extra.size();i++) {
        st.mt[n+i][0]=1; //1,i,i*i,i*i*i
        mt.mt[0][n+i]=extra[i];//p,q,r
        mt.mt[n+i][n]=1; //pascal
        for (int j=1;j<=i;j++) { //pascal
            st.mt[n+i][0]*=n;//1,i*i,i*i*i
            mt.mt[n+i][n+j]=mt.mt[n+i-1][n+j]+mt.mt[n+i-1]
[n+j-1];
        }
    }
    return (mt.pow(k-(n-1))*st).mt[0][0];
}
```

3.5 longest increasing subsequence

```
// Find the Longest Increasing Subsequence of an array in
O(n log n)
// 1 2 3 5 10 2 -1 100 500 -> input
// 1 2 3 5 10 100 500 -> Lis
int lis(vl &nums) {
    vl best;
    int n = nums.size();
    for (int i = 0; i < n; i++) {
        // For non-decreasing
        // int idx = upper_bound(all(best), nums[i]) -
best.begin();

        // For increasing
        int idx = lower_bound(all(best), nums[i]) -
best.begin();
        if (idx == best.size()) {
            best.pb(nums[i]);
        } else {
            best[idx] = min(best[idx], nums[i]);
        }
    }

    return best.size();
}

// Also you can do this with Segment Tree in O(n log n)
```

3.6 multiple knacksack optimizacion

```
/*
Multiple Knacksack

You have a knacksack of a capacity, and 'n' objects with
value, weight, and a number of copies that you can buy of
that object.

Maximize the value without exceding the capacity of the
knacksack.

Time complexity is O(W*N*sum)
W = capacity, N = number of objects,
sum is: for (int i =0, sum = 0;i<n;i++) sum +=
log2(copies[i])

Tested in https://cses.fi/problemset/task/1159/
n<=100, capacity<=10^5, copies[i]<=1000
*/
ll multipleKnacksack(vl &value, vl& weight, vl&copies, ll
capacity) {
    vl vs,ws;
    ll n = value.size();
    for (int i = 0;i<n;i++) {
        ll h=value[i],s=weight[i],k=copies[i];
        ll p = 1;
        while (k>p) {
            k-=p; // Binary Grouping Optimization
            vs.pb(s*p);
            ws.pb(h*p);
            p*=2;
        }
        if (k) {
            vs.pb(s*k);
            ws.pb(h*k);
        }
    }
    vl dp(capacity+1);
    // 0-1 knacksack
    for (int i =0;i<ws.size();i++) {
        for (int j = capacity;j>=ws[i];j--) {
            dp[j] = max(dp[j],dp[j-ws[i]] + vs[i]);
        }
    }
    return dp[capacity];
}
```

3.7 optimizing pragmas for bitset

```
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")

// The difference to TLE and AC in https://cses.fi/
problemset/task/2137/
```

3.8 separation optimization

```
No se si tiene un nombre, pero en varios problemas vi este
truco:

dp[x] = f(x,y) donde se necesita un optimo 'y'.

Truco: si puedes separarlo en f(x,y) en f(x) (+) f(y),
puedes guardar en una estructa optima f(y) como un segment
tree.
```

4 Flow

4.1 hungarian

```
// Halla el maximo match en un grafo bipartito con pesos
(min cost)  O(V ^ 3)

typedef ll T;
const T inf = 1e18;

struct hung {
    int n, m;
    vector<T> u, v; vector<int> p, way;
    vector<vector<T>> g;

    hung(int n, int m):
        n(n), m(m), g(n+1, vector<T>(m+1, inf-1)),
        u(n+1), v(m+1), p(m+1), way(m+1) {}

    void set(int u, int v, T w) { g[u+1][v+1] = w; }

    T assign() {
        for (int i = 1; i <= n; ++i) {
            int j0 = 0; p[0] = i;
            vector<T> minv(m+1, inf);
            vector<char> used(m+1, false);
            do {
                used[j0] = true;
                int i0 = p[j0], j1; T delta = inf;
                for (int j = 1; j <= m; ++j) if (!used[j]) {
                    T cur = g[i0][j] - u[i0] - v[j];
                    if (cur < minv[j]) minv[j] = cur, way[j] = j0;
                    if (minv[j] < delta) delta = minv[j], j1 = j;
                }
                for (int j = 0; j <= m; ++j)
                    if (used[j]) u[p[j]] += delta, v[j] -= delta;
                    else minv[j] -= delta;
                j0 = j1;
            } while (p[j0]);
            do {
                int j1 = way[j0]; p[j0] = p[j1]; j0 = j1;
            } while (j0);
        }
        return -v[0];
    }
};
```

4.2 max flow

```
//#define int long long // take care int overflow with this
//#define vi vector<long long>
struct Dinitz{
    const int INF = 1e9 + 7;
    Dinitz(){}
    Dinitz(int n, int s, int t) {init(n, s, t);}

    void init(int n, int s, int t)
    {
        S = s, T = t;
        nodes = n;
        G.clear(), G.resize(n);
        Q.resize(n);
    }
    struct flowEdge
    {
        int to, rev, f, cap;
    };

    vector<vector<flowEdge> > G;

    // AÃ±ade arista (st -> en) con su capacidad
    void addEdge(int st, int en, int cap) {
        flowEdge A = {en, (int)G[en].size(), 0, cap};
        flowEdge B = {st, (int)G[st].size(), 0, 0};
        G[st].pb(A);
        G[en].pb(B);
    }

    int nodes, S, T; // asignar estos valores al armar el grafo G
                    // nodes = nodos en red de flujo. Hacer G.clear(); G.resize(nodes);
    vi work, lvl;
    vi Q;

    bool bfs() {
        int qt = 0;
        Q[qt++] = S;
        lvl.assign(nodes, -1);
        lvl[S] = 0;
        for (int qh = 0; qh < qt; qh++) {
            int v = Q[qh];
            for (flowEdge &e : G[v]) {
                int u = e.to;
                if (e.cap <= e.f || lvl[u] != -1) continue;
                lvl[u] = lvl[v] + 1;
                Q[qt++] = u;
            }
        }
        return lvl[T] != -1;
    }

    int dfs(int v, int f) {
        if (v == T || f == 0) return f;
        for (int &i = work[v]; i < G[v].size(); i++) {
            flowEdge &e = G[v][i];
            int u = e.to;
            if (e.cap <= e.f || lvl[u] != lvl[v] + 1) continue;
            int df = dfs(u, min(f, e.cap - e.f));
            if (df) {
                e.f += df;
                G[u][e.rev].f -= df;
                return df;
            }
        }
        return 0;
    }

    int maxFlow() {
        int flow = 0;
        while (bfs()) {
            work.assign(nodes, 0);
            while (true) {
                int df = dfs(S, INF);
                if (df == 0) break;
                flow += df;
            }
        }
        return flow;
    }
};

void test_case() {
    ll n, m, s, t;
    cin >> n >> m >> s >> t;
    Dinitz flow;
    flow.init(n, s, t);
    for (int i =0; i < m; i++) {
        ll a, b, c;
        cin >> a >> b >> c;
        flow.addEdge(a, b, c);
    }
    ll f = flow.maxFlow(); // max flow

    vector<tuple<ll,ll,ll>> edges; // edges used with flow
    for (int i = 0; i < n; i++) {
        for (auto edge : flow.G[i]) {
            if (edge.f > 0) {
                edges.pb({i, edge.to, edge.f});
            }
        }
    }
}
```

4.3 min cost max flow

```
// O(min(E^2*V^2, E*V*FLOW))
// Min Cost Max Flow Dinits
struct CheapDinitz{
    const int INF = 1e9 + 7;

    CheapDinitz() {}
    CheapDinitz(int n, int s, int t) {init(n, s, t);}

    int nodes, S, T;
    vi dist;
    vi pot, curFlow, prevNode, prevEdge, Q, inQue;

    struct flowEdge{
        int to, rev, flow, cap, cost;
    };
    vector<vector<flowEdge>> G;
    void init(int n, int s, int t)
    {
        nodes = n, S = s, T = t;
        curFlow.assign(n, 0), prevNode.assign(n, 0),
prevEdge.assign(n, 0);
        Q.assign(n, 0), inQue.assign(n, 0);
        G.clear();
        G.resize(n);
    }

    void addEdge(int s, int t, int cap, int cost)
    {
        flowEdge a = {t, (int)G[t].size(), 0, cap, cost};
        flowEdge b = {s, (int)G[s].size(), 0, 0, -cost};
        G[s].pb(a);
        G[t].pb(b);
    }

    void bellmanFord()
    {
        pot.assign(nodes, INF);
        pot[S] = 0;
        int qt = 0;
        Q[qt++] = S;
        for (int qh = 0; (qh - qt) % nodes != 0; qh++)
        {
            int u = Q[qh % nodes];
            inQue[u] = 0;
            for (int i = 0; i < (int)G[u].size(); i++)
            {
                flowEdge &e = G[u][i];
                if (e.cap <= e.flow) continue;
                int v = e.to;
                int newDist = pot[u] + e.cost;
                if (pot[v] > newDist)
                {
                    pot[v] = newDist;
                    if (!inQue[v])
                    {
                        Q[qt++ % nodes] = v;
                        inQue[v] = 1;
                    }
                }
            }
        }
    }

    ii MinCostFlow()
    {
        bellmanFord();
        int flow = 0;
        int flowCost = 0;
        while (true) // always a good start for an
algorithm :v
        {
            set<ii> s;
            s.insert({0, S});
            dist.assign(nodes, INF);
            dist[S] = 0;
            curFlow[S] = INF;
            while (s.size() > 0)
            {
                int u = s.begin() -> s;
                int actDist = s.begin() -> f;
                s.erase(s.begin());
                if (actDist > dist[u]) continue;
                for (int i = 0; i < (int)G[u].size(); i++)
                {
                    flowEdge &e = G[u][i];
                    int v = e.to;
                    if (e.cap <= e.flow) continue;
                    int newDist = actDist + e.cost + pot[u]
- pot[v];
                    if (newDist < dist[v])
                    {
                        dist[v] = newDist;
                        s.insert({newDist, v});
                        prevNode[v] = u;
                        prevEdge[v] = i;
                        curFlow[v] = min(curFlow[u], e.cap -
e.flow);
                    }
                }
            }
            if (dist[T] == INF)
                break;
            for (int i = 0; i < nodes; i++)
                pot[i] += dist[i];
            int df = curFlow[T];
            flow += df;
            for (int v = T; v != S; v = prevNode[v])
            {
                flowEdge &e = G[prevNode[v]][prevEdge[v]];
                e.flow += df;
                G[v][e.rev].flow -= df;
                flowCost += df * e.cost;
            }
        }
        return {flow, flowCost};
    }
};
```

4.4 min cut

```
/*
Minimum Cut, need revision for general cases!!!

Tested in https://cses.fi/problemset/task/1695/
  with undirected graph with capacity 1

1. Run Max Flow Algorithm
2. Get reachable vertices from source in the residual graph!
   using BFS or DFS, call this reachable vertices = S
3. Iterate S, iterate edges of S in normal graph that not
belongs to S,
   thats the mincut edges of the answer.
*/

void test_case() {
    ll n, m; cin >> n >> m;
    vvl g(n);
    Dinitz f(n,0,n-1); // add Dinitz for maxFlow!!
    for (int i =0;i<m;i++) {
        ll x, y;cin >>x >> y;x--,y--;
        g[x].pb(y); g[y].pb(x); //this example the graph is
bidirectional
        f.addEdge(x,y,1); f.addEdge(y,x,1);
    }
    f.maxFlow(); // step 1

    vvl residual(n); // Step 2
    for (int i =0;i<n;i++) for (auto j:f.G[i])
        if ((j.f<0 && j.cap==0) || (j.f==0 && j.cap > 0))
            residual[i].pb(j.to); // residual graph

    set<ll> vis;
    function<void(int)> dfs = [&](int x) {
        vis.insert(x); // dfs on residual from source
        for (auto y : residual[x]) if (!vis.count(y))
dfs(y);
    };
    dfs(0);

    vector<pair<ll,ll>> ans; // step 3
    for (auto x : vis)
        for (auto y : g[x])
            if (!vis.count(y))
                ans.pb({x,y});

    cout << ans.size() << "\n";
    for (auto e : ans) {
        cout << e.F + 1 << " " << e.S + 1 << "\n";
    }
}
```

5 Geometry

5.1 closest pair of points

```
// It seems O(n log n), not sure but it worked for 50000
// This algorithms is not the best, TLE in CSES
// https://cses.fi/problemset/task/2194

#define x first
#define y second
long long dist2(pair<int, int> a, pair<int, int> b) {
  return 1LL * (a.x - b.x) * (a.x - b.x) + 1LL * (a.y - b.y)
* (a.y - b.y);
}
pair<int, int> closest_pair(vector<pair<int, int>> a) {
  int n = a.size();
  assert(n >= 2);
  vector<pair<pair<int, int>, int>> p(n);
  for (int i = 0; i < n; i++) p[i] = {a[i], i};
  sort(p.begin(), p.end());
  int l = 0, r = 2;
  long long ans = dist2(p[0].x, p[1].x);
  pair<int, int> ret = {p[0].y, p[1].y};
  while (r < n) {
    while (l < r && 1LL * (p[r].x.x - p[l].x.x) * (p[r].x.x
- p[l].x.x) >= ans) l++;
    for (int i = l; i < r; i++) {
      long long nw = dist2(p[i].x, p[r].x);
      if (nw < ans) {
        ans = nw;
        ret = {p[i].y, p[r].y};
      }
    }
    r++;
  }
  return ret;
}

// Tested: https://vjudge.net/solution/52922194/
ccPUXODAMWTzpzCEvXbV
void test_case() {
    ll n;
    cin >> n;
    vector<pair<int,int>> points(n);
    for (int i = 0;i<n;i++) cin >> points[i].x >>
points[i].y;
    auto ans = closest_pair(points);
    cout << fixed << setprecision(6);
    if (ans.F > ans.S) swap(ans.F,ans.S);
    ld dist = sqrtl(dist2(points[ans.F],points[ans.S]));
    cout << ans.F << " " << ans.S << " " << dist << endl;
}
```

5.2 convex hull

```
// Given a Polygon, find its convex hull polygon
// O(n)
struct pt {
    ll x, y;

    pt operator - (pt p) { return {x-p.x, y-p.y}; }

    bool operator == (pt b) { return x == b.x && y == b.y; }
    bool operator != (pt b) { return !((*this) == b); }
    bool operator < (const pt &o) const { return y < o.y ||
(y == o.y && x < o.x); }
};

ll cross(pt a, pt b) { return a.x*b.y - a.y*b.x; } // x =
180 -> sin = 0
ll orient(pt a, pt b, pt c) { return cross(b-a,c-a); }//
clockwise = -
ld norm(pt a) { return a.x*a.x + a.y*a.y; }
ld abs(pt a) { return sqrt(norm(a)); }


struct polygon {
    vector<pt> p;
    polygon(int n) : p(n) {}

    void delete_repetead() {
        vector<pt> aux;
        sort(p.begin(), p.end());
        for(pt &i : p)
            if(aux.empty() || aux.back() != i)
                aux.push_back(i);
        p.swap(aux);
    }

    int top = -1, bottom = -1;
    void normalize() { /// polygon is CCW
        bottom = min_element(p.begin(), p.end()) -
p.begin();
        vector<pt> tmp(p.begin()+bottom, p.end());
        tmp.insert(tmp.end(), p.begin(), p.begin()+bottom);
        p.swap(tmp);
        bottom = 0;
        top = max_element(p.begin(), p.end()) - p.begin();
    }

    void convex_hull() {
        sort(p.begin(), p.end());
        vector<pt> ch;
        ch.reserve(p.size()+1);
        for(int it = 0; it < 2; it++) {
            int start = ch.size();
            for(auto &a : p) {
                /// if colineal are needed, use < and remove
repeated points
                while(ch.size() >= start+2 &&
orient(ch[ch.size()-2], ch.back(), a) <= 0)
                    ch.pop_back();
                ch.push_back(a);
            }
            ch.pop_back();
            reverse(p.begin(), p.end());
        }
        if(ch.size() == 2 && ch[0] == ch[1]) ch.pop_back();
        /// be careful with CH of size < 3
        p.swap(ch);
    }

    ld perimeter() {
        ld per = 0;
        for(int i = 0, n = p.size(); i < n; i++)
            per += abs(p[i] - p[(i+1)%n]);
        return per;
    }
};
```

5.3 heron formula

```
ld triangle_area(ld a, ld b, ld c) {
    ld s = (a + b + c) / 2;
    return sqrtl(s * (s - a) * (s - b) * (s - c));
}
```

5.4 point in convex polygon

```
// Check if a point is in, on, or out a convex Polygon
// in O(log n)
ll IN = 0;
ll ON = 1;
ll OUT = 2;
vector<string> ANS = {"IN", "ON", "OUT"};

#define pt pair<ll,ll>
#define x first
#define y second

pt sub(pt a, pt b) { return {a.x - b.x, a.y - b.y}; }
ll cross(pt a, pt b) { return a.x*b.y - a.y*b.x; } // x =
180 -> sin = 0
ll orient(pt a, pt b, pt c) { return
cross(sub(b,a),sub(c,a)); }// clockwise = -

// poly is in clock wise order
ll insidePoly(vector<pt> &poly, pt query) {
    ll n = poly.size();
    ll left = 1;
    ll right = n - 2;
    ll ans = -1;
    if (!(orient(poly[0], poly[1], query) <= 0
         && orient(poly[0], poly[n-1], query) >= 0)) {
        return OUT;
    }
    while (left <= right) {
        ll mid = (left + right) / 2;
        if (orient(poly[0], poly[mid], query) <= 0) {
            left = mid + 1;
            ans = mid;
        } else {
            right = mid - 1;
        }
    }
    left = ans;
    right = ans + 1;
    if (orient(poly[left], query, poly[right]) < 0) {
        return OUT;
    }
    if (orient(poly[left], poly[right], query) == 0
        || (left == 1 && orient(poly[0], poly[left], query)
== 0)
        || (right == n-1 && orient(poly[0], poly[right],
query) == 0)) {
        return ON;
    }
    return IN;
}
```

5.5 point in general polygon

```
// Use insidepoly(poly, point)
// Returns if a point is inside=0, outside=1, onedge=2
// tested https://vjudge.net/solution/45869791/BIPDAUMWyupUW
18AlWgd
// Seems to be O(n)??
int inf = 1 << 30;
int INSIDE = 0;
int OUTSIDE = 1;
int ONEDGE = 2;
int COLINEAR = 0;
int CW = 1;
int CCW = 2;
typedef long double ld;
struct point {
    ld x, y;
    point(ld xloc, ld yloc) : x(xloc), y(yloc) {}
    point() {}
    point& operator= (const point& other) {
        x = other.x, y = other.y;
        return *this;
    }
    int operator == (const point& other) const {
        return (abs(other.x - x) < .00001 && abs(other.y -
y) < .00001);
    }
    int operator != (const point& other) const {
        return !(abs(other.x - x) < .00001 && abs(other.y -
y) < .00001);
    }
    bool operator< (const point& other) const {
        return (x < other.x ? true : (x == other.x && y <
other.y));
    }
};

struct vect { ld i, j; };

struct segment {
    point p1, p2;
    segment(point a, point b) : p1(a), p2(b) {}
    segment() {}
};

long double crossProduct(point A, point B, point C) {
    vect AB, AC;
    AB.i = B.x - A.x;
    AB.j = B.y - A.y;
    AC.i = C.x - A.x;
    AC.j = C.y - A.y;
    return (AB.i * AC.j - AB.j * AC.i);
}

int orientation(point p, point q, point r) {
    int val = int(crossProduct(p, q, r));
    if(val == 0) {
        return COLINEAR;
    }
    return (val > 0) ? CW : CCW;
}

bool onSegment(point p, segment s) {
    return (p.x <= max(s.p1.x, s.p2.x) && p.x >= min(s.p1.x,
s.p2.x) &&
            p.y <= max(s.p1.y, s.p2.y) && p.y >= min(s.p1.y,
s.p2.y));
}

vector<point> intersect(segment s1, segment s2) {
    vector<point> res;
    point a = s1.p1, b = s1.p2, c = s2.p1, d = s2.p2;

    if(orientation(a, b, c) == 0 && orientation(a, b, d) ==
0 &&
       orientation(c, d, a) == 0 && orientation(c, d, b) ==
0) {
        point min_s1 = min(a, b), max_s1 = max(a, b);
        point min_s2 = min(c, d), max_s2 = max(c, d);

        if(min_s1 < min_s2) {
            if(max_s1 < min_s2) {
                return res;
            }
        }
        else if(min_s2 < min_s1 && max_s2 < min_s1) {
            return res;
        }

        point start = max(min_s1, min_s2), end = min(max_s1,
max_s2);
        if(start == end) {
            res.push_back(start);
        }
        else {
            res.push_back(min(start, end));
            res.push_back(max(start, end));
        }
        return res;
    }
    ld x1 = (b.x - a.x);
    ld y1 = (b.y - a.y);
    ld x2 = (d.x - c.x);
    ld y2 = (d.y - c.y);
    ld u1 = (-y1 * (a.x - c.x) + x1 * (a.y - c.y)) / (-x2 *
y1 + x1 * y2);
    ld u2 = (x2 * (a.y - c.y) - y2 * (a.x - c.x)) / (-x2 *
y1 + x1 * y2);

    if(u1 >= 0 && u1 <= 1 && u2 >= 0 && u2 <= 1) {
        res.push_back(point((a.x + u2 * x1), (a.y + u2 *
y1)));
    }
    return res;
}

int insidepoly(vector<point> poly, point p) {
    bool inside = false;
    point outside(inf, p.y);
    vector<point> intersection;

    for(unsigned int i = 0, j = poly.size()-1; i <
poly.size(); i++, j = i-1) {
        if(p == poly[i] || p == poly[j]) {
            return ONEDGE;
        }
        if(orientation(p, poly[i], poly[j]) == COLINEAR &&
onSegment(p, segment(poly[i], poly[j]))) {
            return ONEDGE;
        }
        intersection = intersect(segment(p, outside),
segment(poly[i], poly[j]));
        if(intersection.size() == 1) {
            if(poly[i] == intersection[0] && poly[j].y <=
p.y) {
                continue;
            }
            if(poly[j] == intersection[0] && poly[i].y <=
p.y) {
                continue;
            }
            inside = !inside;
        }
    }

    return inside ? INSIDE : OUTSIDE;
}
//
```

5.6 polygon diameter

```
// Given a set of points, it returns
// the diameter (the biggest distance between 2 points)
// tested: https://open.kattis.com/submissions/13937489
const double eps = 1e-9;
int sign(double x) { return (x > eps) - (x < -eps); }

struct PT {
    double x, y;
    PT() { x = 0, y = 0; }
    PT(double x, double y) : x(x), y(y) {}
    PT operator - (const PT &a) const { return PT(x - a.x, y
- a.y); }
    bool operator < (PT a) const { return sign(a.x - x) ==
0 ? y < a.y : x < a.x; }
    bool operator == (PT a) const { return sign(a.x - x) ==
0 && sign(a.y - y) == 0; }

};
inline double dot(PT a, PT b) { return a.x * b.x + a.y *
b.y; }
inline double dist2(PT a, PT b) { return dot(a - b, a -
b); }
inline double dist(PT a, PT b) { return sqrt(dot(a - b, a -
b)); }
inline double cross(PT a, PT b) { return a.x * b.y - a.y *
b.x; }
inline int orientation(PT a, PT b, PT c) { return
sign(cross(b - a, c - a)); }

double diameter(vector<PT> &p) {
    int n = (int)p.size();
    if (n == 1) return 0;
    if (n == 2) return dist(p[0], p[1]);
    double ans = 0;
    int i = 0, j = 1;
    while (i < n) {
        while (cross(p[(i + 1) % n] - p[i], p[(j + 1) % n] -
p[j]) >= 0) {
          ans = max(ans, dist2(p[i], p[j]));
          j = (j + 1) % n;
        }
        ans = max(ans, dist2(p[i], p[j]));
        i++;
    }
    return sqrt(ans);
}
vector<PT> convex_hull(vector<PT> &p) {
  if (p.size() <= 1) return p;
  vector<PT> v = p;
    sort(v.begin(), v.end());
    vector<PT> up, dn;
    for (auto& p : v) {
        while (up.size() > 1 && orientation(up[up.size() -
2], up.back(), p) >= 0) {
            up.pop_back();
        }
        while (dn.size() > 1 && orientation(dn[dn.size() -
2], dn.back(), p) <= 0) {
            dn.pop_back();
        }
        up.push_back(p);
        dn.push_back(p);
    }
    v = dn;
    if (v.size() > 1) v.pop_back();
    reverse(up.begin(), up.end());
    up.pop_back();
    for (auto& p : up) {
        v.push_back(p);
    }
    if (v.size() == 2 && v[0] == v[1]) v.pop_back();
    return v;
}

void test_case() {
    ll n;
    cin >>n;
    vector<PT> p(n);
    for (int i = 0;i<n;i++) cin >> p[i].x >> p[i].y;
    p = convex_hull(p);
    cout << fixed<<setprecision(10) << diameter(p) << "\n";
}
```

5.7 segment intersection

```
// No the best algorithm, find a better one!!
// Given two segment, finds the intersection point.

// LINE if they are parallel and mulitple intersection??
// POINT with the intersection point
// NONE if not intersection
struct line {
    ld a, b; // first point
    ld x, y; // second point

    ld m()  { return (a - x)/(b - y); }
    bool horizontal() { return b == y; }
    bool vertical() { return a == x; }

    void intersects(line &o) {
        if (horizontal() && o.horizontal()) {
            if (y == o.y) cout << "LINE\n";
            else cout << "NONE\n";
            return;
        }
        if (vertical() && o.vertical()) {
            if (x == o.x) cout << "LINE\n";
            else cout << "NONE\n";
            return;
        }
        if (!horizontal() && !o.horizontal()) {
            ld ma = m();
            ld mb = o.m();
            if (ma == mb) {
                ld someY = (o.x - x)/ma + y;
                if (abs(someY - o.y) <= 0.000001) {
                    cout << "LINE\n";
                } else {
                    cout << "NONE\n";
                }
            } else {
                ld xx = (x*mb - o.x*ma + ma*mb*(o.y - y))/
(mb - ma);
                ld yy = (xx - x)/ma + y;
                cout << "POINT " << fixed << setprecision(2)
<< xx << " " << yy << "\n";
            }
        } else {
            if (!horizontal()) {
                ld xx;
                if (x == a) {
                    xx = x;
                } else {
                    xx = (o.y - y)/m() + x;
                }
                ld yy = o.y;
                cout << "POINT "<< fixed << setprecision(2)
<< xx << " " << yy << "\n";
            } else {
                ld xx;
                if (x == a) {
                    xx = x;
                } else {
                    xx = (y - o.y)/o.m() + o.x;
                }
                ld yy = y;
                cout << "POINT "<< fixed << setprecision(2)
<< xx << " " << yy << "\n";
            }
        }

    }

};

void test_case() {
    line l[2];
    for (int i = 0; i < 2; i++) {
        ld x, y, a, b;
        cin >> x >> y >> a >> b;
        l[i].a = x;
        l[i].b = y;
        l[i].x = a;
        l[i].y = b;
    }
    l[0].intersects(l[1]);
}
```

6 Graphs

6.1 strongly connected components

```
/*
 Tarjan t(graph); provides you the SCC of that graph
 passing the adjancency list of the graph (as vector<vl>)

 This is 0-indexed, (but you can have node 0 as dummy-node)
 Use t.comp[x] to get the component of node x
 SCC is the total number of components
 adjComp() gives you the adjacency list of strongly
components
*/
struct Tarjan {
    vl low, pre, comp;
    ll cnt, SCC, n;
    vvl g;
    const int inf = 1e9;

    Tarjan(vvl &adj) {
        n = adj.size();
        g = adj;
        low = vl(n);
        pre = vl(n,-1);
        cnt = SCC = 0;
        comp = vl(n,-1);

        for (int i = 0;i<n;i++)
            if (pre[i] == -1) tarjan(i);
    }

    stack<int> st;
    void tarjan(int u) {
        low[u] = pre[u] = cnt++;
        st.push(u);
        for (auto &v : g[u]) {
            if (pre[v] == -1) tarjan(v);
            low[u] = min(low[u],low[v]);
        }
        if (low[u] == pre[u]) {
            while (true) {
                int v = st.top();st.pop();
                low[v] = inf;
                comp[v] = SCC;
                if (u == v) break;
            }
            SCC++;
        }
    }
    vvl adjComp() {
        vvl adj(SCC);
        for (int i = 0;i<n;i++) {
            for (auto j : g[i]) {
                if (comp[i] == comp[j]) continue;
                adj[comp[i]].pb(comp[j]);
            }
        }
        for (int i = 0;i<SCC;i++) {
            sort(all(adj[i]));
            adj[i].erase(
                unique(all(adj[i])),
                adj[i].end());
        }
        return adj;
    }
};

/* Another way is with with Kosaraju:
    1. Find topological order of G
    2. Run dfs in topological order in reverse Graph
       to find o connected component
*/
```

6.2 topological sort

```
// Find the topological order of a graph in O(n)
const int N = 1e5;
vector<vector<ll>> adj(N + 10);
vector<ll> visited(N +10);
bool cycle = false; // reports if doesn't exists a
topological sort
vector<ll> topo;

void dfs(ll x) {
    if (visited[x] == 2) {
        return;
    } else if (visited[x] == 1) {
        cycle = true;
        return;
    }
    visited[x] = 1;
    for (auto y : adj[x]) dfs(y);
    visited[x] = 2;
    topo.pb(x);
}

void test_case() {
    ll n, m; cin >> n >> m;
    for (int i =0; i < m; i++) {
        ll x, y; cin >> x >> y;
        adj[x].pb(y);
    }
    for (int i = 1; i <= n; i++) dfs(i);
    reverse(topo.begin(), topo.end());
    if (cycle) {
        cout << "IMPOSSIBLE\n";
    } else {
        for (int i =0; i < n; i++) {
            cout << topo[i] << " \n" [i == n - 1];
        }
    }
}
```

6.3 two sat

```
/*
2-Sat (Boolean satisfiability problem with 2-clause
literals)
Complexity: O(n)
Tested: https://cses.fi/problemset/task/1684

To find a solution that makes this true with N boolean vars
as form:
  (x or y) and (~x or y) and (z or ~x) and d and (x => y)

Call s.satisfiable() to see if solution, and sat2.value to
see
values of variables
*/

struct sat2 {
  int n;
  vector<vector<vector<int>>> g;
  vector<int> tag;
  vector<bool> seen, value;
  stack<int> st;
  sat2(int n) : n(n), g(2, vector<vector<int>>(2*n)),
tag(2*n), seen(2*n), value(2*n) { }
  int neg(int x) { return 2*n-x-1; }
  void add_or(int u, int v) { implication(neg(u), v); }
  void make_true(int u) { add_edge(neg(u), u); }
  void make_false(int u) { make_true(neg(u)); }
  void eq(int u, int v) {
    implication(u, v);
    implication(v, u);
  }
  void diff(int u, int v) { eq(u, neg(v)); }
  void implication(int u, int v) {
    add_edge(u, v);
    add_edge(neg(v), neg(u));
  }
  void add_edge(int u, int v) {
    g[0][u].push_back(v);
    g[1][v].push_back(u);
  }
  void dfs(int id, int u, int t = 0) {
    seen[u] = true;
    for(auto& v : g[id][u])
```

```
      if(!seen[v])
        dfs(id, v, t);
    if(id == 0) st.push(u);
    else tag[u] = t;
  }
  void kosaraju() {
    for(int u = 0; u < n; u++) {
      if(!seen[u]) dfs(0, u);
      if(!seen[neg(u)]) dfs(0, neg(u));
    }
    fill(seen.begin(), seen.end(), false);
    int t = 0;
    while(!st.empty()) {
      int u = st.top(); st.pop();
      if(!seen[u]) dfs(1, u, t++);
    }
  }
  bool satisfiable() {
    kosaraju();
    for(int i = 0; i < n; i++) {
      if(tag[i] == tag[neg(i)]) return false;
      value[i] = tag[i] > tag[neg(i)];
    }
    return true;
  }
};
```

7 Mateo

7.1 2 sat

```
/*
indexado en 0
Time complexity: O(N)
Se puede usar desde index 0 en los nodos y la inicializacion
tampoco es estricta e.g. sat2 S(n+5)

Notas.- En problemas de direccionar aristas e.g. grado
salida = grado entrada
*/
struct sat2 {
  int n;
  vector<vector<vector<int>>> g;
  vector<int> tag;
  vector<bool> seen, value;
  stack<int> st;
  sat2(int n) : n(n), g(2, vector<vector<int>>(2*n)),
tag(2*n), seen(2*n), value(2*n) { }
  int neg(int x) { return 2*n-x-1; }
  void add_or(int u, int v) { implication(neg(u), v); }
  void make_true(int u) { add_edge(neg(u), u); }
  void make_false(int u) { make_true(neg(u)); }
  void eq(int u, int v) {
    implication(u, v);
    implication(v, u);
  }
  void diff(int u, int v) { eq(u, neg(v)); }
  void implication(int u, int v) {
    add_edge(u, v);
    add_edge(neg(v), neg(u));
  }
  void add_edge(int u, int v) {
    g[0][u].push_back(v);
    g[1][v].push_back(u);
  }
  void dfs(int id, int u, int t = 0) {
    seen[u] = true;
    for(auto& v : g[id][u])
      if(!seen[v])
        dfs(id, v, t);
    if(id == 0) st.push(u);
    else tag[u] = t;
  }
  void kosaraju() {
    for(int u = 0; u < n; u++) {
      if(!seen[u]) dfs(0, u);
      if(!seen[neg(u)]) dfs(0, neg(u));
    }
    fill(seen.begin(), seen.end(), false);
    int t = 0;
    while(!st.empty()) {
      int u = st.top(); st.pop();
      if(!seen[u]) dfs(1, u, t++);
    }
  }
  bool satisfiable() {
    kosaraju();
    for(int i = 0; i < n; i++) {
      if(tag[i] == tag[neg(i)]) return false;
      value[i] = tag[i] > tag[neg(i)];
    }
    return true;
  }
};
```

7.2 2d bit

```
#include<bits/stdc++.h>
#define lcm(a,b) (a/__gcd(a,b))*b
#define fast
ios_base::sync_with_stdio(false);cin.tie(0);cout.tie(0);
#define ll long long int
#define vi vector<int>
#define vll vector<ll>
#define pb push_back
#define F first
#define S second
#define mp make_pair
//"\n"
//__builtin_popcount(x)
// a+b=2*(a&b) + (a^b)
using namespace std;
const int tam=1005;
int n,q;
int T[tam][tam];
void update(int x, int y, int val){
    x++;y++;
    for(;x<tam;x+=x&-x){
        for(int l=y;l<tam;l+=l&-l)T[x][l]+=val;
    }
}
int query(int x, int y){
    x++;y++;
    int res=0;
    for(;x>0;x-=x&-x){
        for(int l=y;l>0;l-=l&-l)res+=T[x][l];
    }
    return res;
}
int main()
{
    cin>>n>>q;
    string s;
    vector<string>M;
    for(int i=0;i<n;i++){
        cin>>s;
        M.pb(s);
        for(int l=0;l<n;l++){
            if(s[l]=='*'){
                update(i,l,1);
            }
        }
    }
    while(q--){
        int c,x1,x2,y1,y2;
        cin>>c;
        if(c==1){
            cin>>x1>>y1;
            x1--;y1--;
            if(M[x1][y1]=='*'){
                M[x1][y1]='.';
                update(x1,y1,-1);
            }else{
                M[x1][y1]='*';
                update(x1,y1,1);
            }
        }else{
            cin>>x1>>y1>>x2>>y2;
            x1--;y1--;x2--;y2--;
            cout<<query(x2,y2)-query(x2,y1-1)-
query(x1-1,y2)+query(x1-1,y1-1)<<endl;
        }
    }
    return 0;
}
```

7.3 aho corasick

```
// Notas.- Cuando formo el suffix tree inverso
// cuando quiero ver cuantas veces aparece un nodo en un
string s, entonces hago caminar en el aho corasick y en cada
paso chequedar suffix links si llegan
// a veces se puede armar el suffix tree y luego con euler
tour y st puedo ver cuantas veces se toco este nodo

struct vertex {
  map<char,int> next,go;
  int p,link;
  char pch;
  vector<int> leaf; // se puede cambiar por int, en ese caso
int leaf y leaf(0) en constructor
  vertex(int p=-1, char pch=-1):p(p),pch(pch),link(-1){}
};
vector<vertex> t;
void aho_init(){ //do not forget!!
  t.clear();t.pb(vertex());
}
void add_string(string s, int id){
  int v=0;
  for(char c:s){
    if(!t[v].next.count(c)){
      t[v].next[c]=t.size();
      t.pb(vertex(v,c));
    }
    v=t[v].next[c];
  }
  t[v].leaf.pb(id);
}
int go(int v, char c);
int get_link(int v){
  if(t[v].link<0)
    if(!v||!t[v].p)t[v].link=0;
    else t[v].link=go(get_link(t[v].p),t[v].pch);
  return t[v].link;
}
int go(int v, char c){
  if(!t[v].go.count(c))
    if(t[v].next.count(c))t[v].go[c]=t[v].next[c];
    else t[v].go[c]=v==0?0:go(get_link(v),c);
  return t[v].go[c];
}
```

7.4 centroid descomposition

```
/*
La altura del Centroid Tree es log(N).
El camino entre cualquier par de nodos (A,B) pasa por un
centroide ancestro de ambos (LCA en el Centroid Tree).
Para problemas donde se hace update(nodo) y query(nodo).
Minimizando algo por ejemplo, entonces solo actualizas los
log(N) ancestros de nodo.
y para query(nodo) preguntas por cada ancestro de nodo, de
esta forma revisas todos los caminos entre (nodo, algun otro
nodo)

Time Complexity: O(N log(N))
*/

const int tam = 200005;
vi G[tam];
int del[tam], sz[tam];
int n;

void init(int nodo, int ant) {
  sz[nodo] = 1;
  for (auto it : G[nodo]) {
    if (it == ant || del[it]) continue;
    init(it, nodo);
    sz[nodo] += sz[it];
  }
}

int centroid(int nodo, int ant, int desired) {
  for (auto it : G[nodo]) {
    if (it == ant || del[it]) continue;
    if (sz[it] * 2 >= desired) return centroid(it, nodo,
desired);
  }
  return nodo;
}

int get_centroid(int nodo) {
  init(nodo, -1);
  int desired = sz[nodo];
  return centroid(nodo, -1, desired);
}

void DC(int nodo) {
  int c = get_centroid(nodo);
  del[c] = 1;
  // aqui haces pre/calculo ?
  // update dfs(nodo)
  for (auto it : G[c]) {
    if (del[it]) continue;
    // sigues con calculo, a veces si tienes que contar para
cada nodo caminos que pasan sobre el
    // y no solamente cantidad de caminos puedes hacer
    // delete dfs(it)
    // contar (it)
    // update dfs(it)
  }
  // * reinicias tus arreglos *
  for (auto it : G[c]) {
    if (del[it]) continue;
    DC(it, c);
  }
}
```

7.5 chulltrick

```
/// Complexity: O(|N|*log(|N|))
typedef ll T;
const T is_query = -(1LL<<62);
struct line {
  T m, b;
  mutable multiset<line>::iterator it, end;
  bool operator < (const line &rhs) const {
    if(rhs.b != is_query) return m < rhs.m;
    auto s = next(it);
    if(s == end) return 0;
    return b - s->b < (long double)(s->m - m) * rhs.m;
  }
};
struct CHT : public multiset<line> {
  bool bad(iterator y) {
    auto z = next(y);
    if(y == begin()) {
      if(z == end()) return false;
      return y->m == z->m && y->b <= z->b;
    }
    auto x = prev(y);
    if(z == end()) return y->m == x->m && y->b == x->b;
    return (long double)(x->b - y->b)*(z->m - y->m) >= (long
double)(y->b - z->b)*(y->m - x->m);
  }
  void add(T m, T b) {
    auto y = insert({m, b});
    y->it = y; y->end = end();
    if(bad(y)) { erase(y); return; }
    while(next(y) != end() && bad(next(y))) erase(next(y));
    while(y != begin() && bad(prev(y)))erase(prev(y));
  }
  T eval(T x) { /// for maximum
    auto l = *lower_bound({x, is_query});
    return l.m*x+l.b;
  }
};

// for minimum, you must change (b, m) to (-b, -m)

vector<ld> get_intersections(CHT &cht) {
    vector<ld> res;
    for(auto it = cht.begin(); it != cht.end(); it++) {
        if(next(it) == cht.end()) break;
        if(it->m == next(it)->m) continue;
        res.pb((ld)(next(it)->b - it->b) / (it->m -
next(it)->m));
    }
    return res;
}
```

7.6 closest pair of points

```
/*
Retorna indices (index 0) de los puntos mas cercanos.
Tiempo: O(n log n)
*/
long long dist2(pair<int, int> a, pair<int, int> b) {
  return 1LL * (a.F - b.F) * (a.F - b.F) + 1LL * (a.S - b.S)
* (a.S - b.S);
}

pair<int, int> closest_pair(vector<pair<int, int>> a) {
  int n = a.size();
  assert(n >= 2);
  vector<pair<pair<int, int>, int>> p(n);
  for (int i = 0; i < n; i++) p[i] = {a[i], i};
  sort(p.begin(), p.end());
  int l = 0, r = 2;
  long long ans = dist2(p[0].F, p[1].F);
  pair<int, int> ret = {p[0].S, p[1].S};
  while (r < n) {
    while (l < r && 1LL * (p[r].F.F - p[l].F.F) * (p[r].F.F
- p[l].F.F) >= ans) l++;
    for (int i = l; i < r; i++) {
      long long nw = dist2(p[i].F, p[r].F);
      if (nw < ans) {
        ans = nw;
        ret = {p[i].S, p[r].S};
      }
    }
    r++;
  }
  return ret;
}
```

7.7 dp dc

```
const int tam=8005;
const ll INF=1e17;
ll locura[tam];
ll pref[tam];
ll dp[805][tam];
ll riesgo(int l, int r){
  if(l>r)return 0;
  return (pref[r]-pref[l-1])*(r-l+1);
}
// solve dp retorna k
ll solvedp(int g,int pos, int izq, int der){
  dp[g][pos]=INF;
  int k;
  for(int i=izq;i<=der;i++){
    ll curr=dp[g-1][i]+riesgo(i+1,pos);
    if(curr<dp[g][pos]){
      dp[g][pos]=curr;
      k=i;
    }
  }
  return k;
}
void solve(int g,int l, int r, int izq, int der){
  if(l>r)return;
  if(l==r){
    solvedp(g,l,izq,der);
    return;
  }
  int mid=(l+r)/2;
  int k=solvedp(g,mid,izq,der);
  solve(g,mid+1,r,k,der);
  solve(g,l,mid-1,izq,k);
}
int main(){
  // puedo aplicar D&C pq la transicion es dp[G][i]=dp[G-1]
[algo] + C(G,i)
  // la funcion no es decreciente nunca respecto a k
  // algo de G,i <= algo de G,i+1
  int L,G,x;
  cin>>L>>G;
  if(G>L)G=L;
  for(int i=1;i<=L;i++){
    cin>>locura[i];
    pref[i]=pref[i-1]+locura[i];
  }
  for(int i=1;i<=L;i++){
    dp[1][i]=riesgo(1,i);// caso base cuando solo tomo un
guardia
  }
  for(int i=2;i<=G;i++){
    solve(i,1,L,1,L);
  }
  cout<<dp[G][L]<<endl;

  return 0;
}
// https://www.hackerrank.com/contests/ioi-2014-practice-
contest-2/challenges/guardians-lunatics-ioi14/problem
```

7.8 dp dc amortizado

```
const int tam=100005;
ll a[tam];
ll cnt[tam];
const ll INF=1e16;
ll dp[25][tam];//G y pos
ll TOT=0;
int L=1,R;
void add(int x){TOT+=cnt[x]++;}
void del(int x){TOT-=--cnt[x];}
ll query(int l,int r){
  while(L>l) add(a[--L]);
  while(R<r) add(a[++R]);
  while(L<l) del(a[L++]);
  while(R>r) del(a[R--]);
  return TOT;
}
int solvedp(int g,int pos, int izq, int der){
  int k=0;
  dp[g][pos]=INF;
  for(int i=izq;i<=min(der,pos-1);i++){
    ll curr=dp[g-1][i]+query(i+1,pos);
    if(curr<dp[g][pos]){
      dp[g][pos]=curr;
      k=i;
    }
  }
  return k;
}
void solve(int g,int l, int r, int izq, int der){
  if(l>r)return;
  int mid=(l+r)/2;
  int k=solvedp(g,mid,izq,der);
  solve(g,l,mid-1,izq,k);
  solve(g,mid+1,r,k,der);
}
int main(){
  fast
  fast
  ll n,k;
  cin>>n>>k;
  ll acum=0;
  for(int i=1;i<=n;i++){
    cin>>a[i];
    acum+=cnt[a[i]];cnt[a[i]]++;
    dp[1][i]=acum;
  }
  memset(cnt,0,sizeof(cnt));
  for(int i=2;i<=k;i++){
    solve(i,1,n,1,n);
  }
  cout<<dp[k][n]<<endl;
  return 0;
}
```

7.9 dsu rollback

```
/*
Para sacar checkpoint int CP = st.size()
Para rollback rollback(CP)
LLamar a init(n) al inicio

Note.- index 1 de los nodos, cuidado con los indices de las
aristas al hacer Dynamic Connectivity
dynamic connectivity se realiza sobre los indices de las
```

```
queries simulando el paso del tiempo
y las aristas viven en ciertos rangos de tiempo (se simula
con dfs y segment tree)

Time Complexity: O(log(n)) para find y union
*/

struct RB_DSU {
    vi P;
    vi sz;
    stack<int> st;
    int scc;

    void init(int n) {
        P.resize(n+1);
        sz.resize(n+1, 1);
        scc = n;
        for (int i = 1; i <= n; i++) P[i] = i;
    }

    int _find(int a) {
        if (P[a] == a)
            return a;
        return _find(P[a]);
    }

    void _union(int a, int b) {
        a = _find(a);
        b = _find(b);
        if (a == b) return;
        if (sz[a] > sz[b]) swap(a, b);
        P[a] = b;
        sz[b] += sz[a];
        scc--;
        st.push(a);
    }

    void rollback(int t) {
        while (st.size() > t) {
            int a = st.top();
            st.pop();
            sz[P[a]] -= sz[a];
            P[a] = a;
            scc++;
        }
    }
};
```

7.10 euler walk

```
/*
La entrada es un vector (dest, index global de la arista) en
dirigidos
para grafos no dirigidos las aristas de ida y vuelta tienen
el mismo index global.
Retorna un vector de nodos en el Eulerian path/cycle
con src como nodo inicial. Si no hay solucion, retorna un
vector vacio.

Para obtener indices de aristas, anhadir .second a s y ret o
usar mapa.
Para ver si existe respuesta, ver si ret.size() == nedges +
1
Para ver si existe camino euleriano con (start, end) tambien
ver si ans.back() == end

Un grafo dirigido tiene un camino euleriano si:

Tiene exactamente un vertice con outDegree - inDegree = 1
Tiene exactamente un vertice con inDegree - outDegree = 1
Todos los demas vertices tienen inDegree = outDegree
El recorrido empieza en el vertice con outDegree - inDegree
= 1
Correr desde este nodo y no necesito verficar lo demas (si
no hay tal nodo correr desde uno con grado de salida > 0)

Nota.- Volverlo global D,its,eu si corres varias veces (para
cada componente conexa)
Time complexity: O(V + E)
*/
vi eulerWalk(vector<vector<pii>> &gr, int nedges, int src =
1) {
  int n = gr.size();
  vi D(n), its(n), eu(nedges), ret, s = {src}; // cambiar eu
a mapa<int,bool> si las aristas no son [0,nedges]
  D[src]++; // para permitir Euler Paths, no solo ciclos
  while (!s.empty()) {
    int x = s.back(), y, e, &it = its[x], end =
gr[x].size();
    if (it == end) {
      ret.pb(x);
      s.pop_back();
      continue;
    }
    tie(y, e) = gr[x][it++];
    if (!eu[e]) {
      D[x]--, D[y]++;
      s.pb(y);
      eu[e] = 1;
    }
  }
  for (int x : D) if (x < 0 || ret.size() != nedges + 1)
return {};
  return {ret.rbegin(), ret.rend()};
}
```

7.11 hld

```
// El camino entre dos nodos pasa por maximo log n aristas
livianas
// los ids (en el arreglo) de los nodos de un subarbol son
contiguos entonces puedes hacer updates a todo el subarbol
[id[nodo],id[nodo]+sz[nodo]-1]
// en dp que solo importa el estado de atras se puede hacer
dp[lvl][estado] para ahorrar memoria y primero me muevo por
los livianos
// y luego por el pesado sin cambiar el lvl pq ya no importa

// heavy light decomposition
const int tam=200005;
int v[tam];
int bigchild[tam],padre[tam],depth[tam];
int sz[tam],id[tam],tp[tam];
int T[4*tam];
vi G[tam];
int n;
int query(int lo, int hi) {
  int ra = 0, rb = 0;
  for (lo += n, hi += n + 1; lo < hi; lo /= 2, hi /= 2) {
    if (lo & 1) ra = max(ra, T[lo++]);
    if (hi & 1) rb = max(rb, T[--hi]);
  }
  return max(ra, rb);
}
void update(int idx, int val) {
  T[idx += n] = val;
  for (idx /= 2; idx; idx /= 2) T[idx] = max(T[2 * idx], T[2
* idx + 1]);
}
void dfs_size(int nodo, int ant){
  sz[nodo]=1;
  int big=-1;
  int who=-1;
  padre[nodo]=ant;
  for(auto it : G[nodo]){
    if(it==ant)continue;
    depth[it]=depth[nodo]+1;
    dfs_size(it,nodo);
    sz[nodo]+=sz[it];
    if(sz[it]>big){
      big=sz[it];
      who=it;
    }
  }
  bigchild[nodo]=who;
}
int num=0;
void dfs_hld(int nodo, int ant, int top){
  id[nodo]=num++;
  tp[nodo]=top;
  if(bigchild[nodo]!=-1){
    dfs_hld(bigchild[nodo],nodo,top);
  }
  for(auto it : G[nodo]){
    if(it==ant || it==bigchild[nodo])continue;
    dfs_hld(it,nodo,it);
  }
}
int queryPath(int a, int b){
  int res=0;
  while(tp[a]!=tp[b]){
    if(depth[tp[a]]<depth[tp[b]])swap(a,b);
    res=max(res,query(id[tp[a]],id[a]));
    a=padre[tp[a]];
  }
  if(depth[a]>depth[b])swap(a,b);
  res=max(res,query(id[a],id[b]));
  return res;
}
int main(){
  int c,q,a,b;
  cin>>n>>q;
  for(int i=1;i<=n;i++)cin>>v[i];
  for(int i=0;i<n-1;i++){
    cin>>a>>b;
    G[a].pb(b);
    G[b].pb(a);
  }
  dfs_size(1,0);
  dfs_hld(1,0,1);
  for(int i=1;i<=n;i++){
    update(id[i],v[i]);
  }
  while(q--){
    cin>>c;
    cin>>a>>b;
    if(c==1){
      update(id[a],b);
    }else{
      printf("%d ", queryPath(a,b));
    }
  }
  return 0;
}
```

7.12 implicit segment tree

```
// Node *T = new Node;
// query(T, 0, top, 0, top);  top = 1e9 e.g.
// update(T, 0, top, y1, y2);
struct Node {
    int valor;
    int lazy;
    Node *L, *R;
    Node() : valor(0), lazy(0), L(NULL), R(NULL) {}
    void propagate(int b, int e) {
        if (lazy == 0) return;
        lazy = 0;
        valor = (e - b + 1) - valor;
        if (b == e) return;
        if (!L) L = new Node();
        if (!R) R = new Node();
        L->lazy ^= 1;
        R->lazy ^= 1;

        // esta vaina no es necesaria solo cuando da MLE
        if (L && L->lazy == 0 && L->valor == 0) {
            delete L;
            L = NULL;
        }
        if (R && R->lazy == 0 && R->valor == 0) {
            delete R;
            R = NULL;
        }
    }
};

void update(Node *nodo, int b, int e, int izq, int der) {
    nodo->propagate(b, e);
    if (b > der || e < izq) return;
    if (b >= izq && e <= der) {
        nodo->lazy ^= 1;
        nodo->propagate(b, e);
        return;
    }
    int mid = (b + e) / 2;
    if (!nodo->L) nodo->L = new Node();
    if (!nodo->R) nodo->R = new Node();
    update(nodo->L, b, mid, izq, der);
    update(nodo->R, mid + 1, e, izq, der);
    nodo->valor = nodo->L->valor + nodo->R->valor;
}

int query(Node *nodo, int b, int e, int izq, int der) {
    if (b > der || e < izq) return 0;
    nodo->propagate(b, e);
    if (b >= izq && e <= der) return nodo->valor;
    int mid = (b + e) / 2;
    return query(nodo->L, b, mid, izq, der) + query(nodo->R,
mid + 1, e, izq, der);
}
```

7.13 isomorfismo arboles

```
#include <bits/stdc++.h>
#define vi vector<int>
#define pb push_back
#define S second
#define F first
using namespace std;
struct Tree{
  int n;
  vi sz;
  vector<vi>G;
  vi centroids;
  vector<vi>level;
  vi prev;
  Tree(int x){
    n=x;
    sz.resize(x+1);
    G.assign(n+1,vi());
    prev.resize(n+1);
  }
  void addEdge(int a, int b){
    G[a].pb(b);G[b].pb(a);
  }
  void centroid(int nodo, int ant){
    bool ok=1;
    for(auto it : G[nodo]){
      if(it==ant)continue;
      if(sz[it]>n/2){
        ok=false;
      }
      centroid(it,nodo);
    }
    int atras=n-sz[nodo];
    if(atras>n/2)ok=false;
    if(ok)centroids.pb(nodo);
  }
  void initsz(int nodo, int ant){
    sz[nodo]=1;
    for(auto it : G[nodo]){
      if(it!=ant){
        initsz(it,nodo);
        sz[nodo]+=sz[it];
      }
    }
  }
  void initLevels(int nodo){
    level.clear();
    vi aux;aux.pb(nodo);
    int pos=0;
    level.pb(aux);
    prev[nodo]=-1;
    while(true){
      aux.clear();
      for(auto it : level[pos]){
        for(auto j : G[it]){
          //cout<<"apagare la luz  "<<j<<endl;
          if(j==prev[it])continue;
          aux.pb(j);
          prev[j]=it;
        }
      }
      if(aux.size()==0)break;
      level.pb(aux);
      pos++;
    }
  }
};
```

```
bool check(Tree A, int a, Tree B, int b){
  A.initLevels(a);B.initLevels(b);
  if(A.level.size()!=B.level.size())return false;
  int hashA[A.n+5];
  int hashB[A.n+5];//hash del subarbol rooteado en i

  vector<vi>EA,EB;//le paso los hash de todos los hijos de i
me
                  //servira para formar el hash del subarbol
i
  EA.resize(A.n+1);EB.resize(A.n+1);

  for(int h=A.level.size()-1;h>=0;h--){
    map<vi,int>ind;
    for(auto it : A.level[h]){
      sort(EA[it].begin(),EA[it].end());
      ind[EA[it]]=0;
    }
    for(auto it : B.level[h]){
      sort(EB[it].begin(),EB[it].end());
      ind[EB[it]]=0;
    }

    int num=0;
    for(auto it : ind){
      it.S=num;
      ind[it.F]=num;
      num++;
    }
    //paso a sus padres
    for(auto it : A.level[h]){
      hashA[it]=ind[EA[it]];

      if(h>0)EA[A.prev[it]].pb(hashA[it]);
    }
    for(auto it : B.level[h]){
      hashB[it]=ind[EB[it]];
      if(h>0)EB[B.prev[it]].pb(hashB[it]);
    }
  }
  return hashA[a]==hashB[b];
}
bool isomorphic(Tree A, Tree B){
  A.initsz(1,-1);B.initsz(1,-1);
  A.centroid(1,-1);B.centroid(1,-1);
  vi CA=A.centroids,CB=B.centroids;
  if(CA.size()!=CB.size())return false;
  for(int i=0;i<CB.size();i++){
    if(check(A,CA[0],B,CB[i])){
      return true;
    }
  }
  return false;
}
int main() {
  int t,n,a,b;
  cin>>t;
  while(t--){
    cin>>n;
    Tree A(n);
    Tree B(n);
    for(int i=1;i<n;i++){
      cin>>a>>b;
      A.addEdge(a,b);
    }
    for(int i=1;i<n;i++){
      cin>>a>>b;
      B.addEdge(a,b);
    }
    if(isomorphic(A,B)){
      cout<<"YES"<<"\n";
    }else{
      cout<<"NO"<<"\n";
    }
  }
}
```

7.14 khun

```
// algoritmo de khun para grafos bipartitos  O(nm)
const int tam = 100;
vi G[tam]; // pueden tener mismo indices nodos de distintos
grupos
bool vis[tam];
int pareja[tam];// pareja de los nodos de la derecha

bool khun(int nodo) {
  if (vis[nodo]) return false;
  vis[nodo] = 1;
  for (auto it : G[nodo]) {
    if (pareja[it] == -1 || khun(pareja[it])) {
      pareja[it] = nodo;
      return true;
    }
  }
  return false;
}

int main() {
  int m, a, b;
  cin >> m;
  for (int i = 0; i < m; i++) {
    cin >> a >> b; // de izquierda a derecha
    G[a].pb(b);
  }
  memset(pareja, -1, sizeof(pareja));
  int match = 0;
  for (int i = 1; i <= n; i++) {
    memset(vis, false, sizeof(vis)); // no olvidar
    if (khun(i)) match++; // camino aumentante
  }
  return 0;
}
```

7.15 knapsack optimization

```
bitset<100001> posi;
posi[0] = 1;
for (int t : comps) posi |= posi << t;
for (int i = 1; i <= n; ++i) cout << posi[i];

// cuando suma maxima es tam = 2e5
// entonces la cantidad de numeros diferentes es sqrt(2e5)
// lo que hago es dejar como maximo 2 repeticiones en cada
valor
// entonces cada dos i's le paso uno a 2*i y me queda solo
sqrt(n) numeros
// ya que cada i solo aparece maximo 2 veces

for(int i=1;i<tam;i++){
  if(cant[i]>=3){
    int mv=cant[i]/2;
    if(cant[i]%2==0)mv--;
    cant[i]-=mv*2;
    cant[2*i]+=mv;
  }
}

bitset<tam> dp;
dp[0]=1;
for(int i=1;i<tam;i++){// importante empezar en 1
  for(int l=0;l<cant[i];l++){
    dp|=dp<<i;
  }
}
```

7.16 manacher

```
/*
f = 1 para pares, 0 impar
a a a a a a
1 2 3 3 2 1   f = 0 impar
0 1 2 3 2 1   f = 1 par centrado entre [i-1,i]
Time: O(n)
*/
void manacher(string &s, int f, vi &d) {
  int l = 0, r = -1, n = s.size();
  d.assign(n, 0);
  for (int i = 0; i < n; i++) {
    int k = (i > r ? (1 - f) : min(d[l + r - i + f], r - i +
f)) + f;
    while (i + k - f < n && i - k >= 0 && s[i + k - f] ==
s[i - k]) ++k;
    d[i] = k - f; --k;
    if (i + k - f > r) l = i - k, r = i + k - f;
  }
}
```

7.17 mo's on trees

```
// Si en el rango un nodo aparece dos veces entonces no se
toma en cuenta (se cancela)
// Para una query en camino [u,v], IN[u]<=IN[v]
// Si LCA(u,v) = u -> Rango Query [IN[u],IN[v]]
// Si No -> Rango Query [OUT[u],IN[v]] + [IN[LCA],IN[LCA]]
(o sea falta considerar el LCA)

// Cuando las consultas son sobre las aristas
// Si LCA(u,v) = u -> Rango Query [IN[u]+1,IN[v]]
// Si No -> Rango Query [OUT[u],IN[v]]

const int tam = 100005;
vector<pair<int, int>> G[tam];
int dp[20][tam];// esto para LCA
int tiempo = -1;
int IN[tam];// tiempo de entrada
int OUT[tam];// tiempo de salida
int A[3*tam];// los nodos en orden del dfs
int depth[tam];
int valor[tam];// valor del nodo/arista

void dfs(int nodo, int ant, int llega, int d) {
    depth[nodo] = d+1;
    dp[0][nodo] = ant;
    valor[nodo] = llega;
    IN[nodo] = ++tiempo;
    A[IN[nodo]] = nodo;
    for (auto it : G[nodo]) {
        int v = it.first;
        int val = it.second;
        if (v == ant) continue;
        dfs(v, nodo, val, d+1);
    }
    OUT[nodo] = ++tiempo;
    A[OUT[nodo]] = nodo;
}
```

7.18 mo's

```
// Complexity: O(|N+Q|*sqrt(|N|)*|meter/quitar|)
// Requiere meter(), quitar()

vector<pair<pair<int,int>,int> >Q;// {{izq,der},id}
int tami = 300; // o sqrt(n)+1
bool comp(pair<pair<int,int>,int> a,pair<pair<int,int>,int>
b){
    if(a.F.F/tami!=b.F.F/tami){
        return a.F.F/tami<b.F.F/tami;
    }
    return a.F.S<b.F.S;
}
// main
sort(Q.begin(),Q.end(),comp);
int L=0,R=-1;
int respuesta=0;
for(int i=0;i<q;i++){
    int izq=Q[i].F.F;
    int der=Q[i].F.S;
    int ind=Q[i].S;
    while(L>izq)meter(--L);
    while(R<der)meter(++R);
    while(R>der)quitar(R--);
    while(L<izq)quitar(L++);
    res[ind]=respuesta;
}
```

7.19 parallel binary search

```
#include<bits/stdc++.h>
#define lcm(a,b) (a/__gcd(a,b))*b
#define fast
ios_base::sync_with_stdio(false);cin.tie(0);cout.tie(0);
#define ll long long int
#define vi vector<int>
#define vll vector<ll>
#define pb push_back
#define F first
#define S second
#define mp make_pair
//salida rapida "\n"
//DECIMALES fixed<<sp(n)<<x<<endl;
//gcd(a,b)= ax + by
//lCB x&-x
//set.erase(it) - ersases the element present at the
required index//auto it = s.find(element)
//set.find(element) - iterator pointing to the given element
if it is present else return pointer pointing to set.end()
//set.lower_bound(element) - iterator pointing to element
greater than or equal to the given element
//set.upper_bound(element) - iterator pointing to element
greater than the given element
// | ^
//__builtin_popcount(x)
using namespace std;
const int tam=300030;
const ll INF=1e16;
unsigned long long T[2*tam];
ll n,m,k;
vector<vll>G;
ll E[tam];
ll res[tam];
vector<pair<pair<ll,ll>,ll > >Q;//estas son las queries
void update(int pos, int val){
    while(pos<=m){
        T[pos]+=val;
        pos+=(pos&-pos);
    }
}
ll query(ll pos){
   unsigned long long res=0;
   while(pos>0){
       res+=T[pos];
       pos-=(pos&-pos);
   }
   return res;
}

void parallel(ll b,ll e, vll q){
    if(q.size()==0 or e<b)return ;
    ll mid=(b+e)/2;
    //memset(T,0,sizeof T);
    for(int i=b;i<=mid;i++){
        ll l=Q[i].F.F,r=Q[i].F.S,val=Q[i].S;
        update(l,val);
        if(r<l){
            update(1,val);
            update(m+1,-val);
        }
        update(r+1,-val);
    }
    vll A,B;
    for(int i=0;i<q.size();i++){
        ll sum=0ll;
        for(auto it : G[q[i]]){
            sum+=query(it);
            if (sum >= 1e10) break;
        }
        if(sum>=E[q[i]]){
            A.pb(q[i]);
            res[q[i]]=min(res[q[i]],mid+1);
        }else{
            B.pb(q[i]);
        }
    }
    parallel(mid+1,e,B);
    for(int i=b;i<=mid;i++){
        int l=Q[i].F.F,r=Q[i].F.S,val=-Q[i].S;
        update(l,val);
        if(r<l){
            update(1,val);
            update(m+1,-val);
        }
        update(r+1,-val);
    }
    parallel(b,mid-1,A);
}
int main()
{
    fast
    ll x;
    cin>>n>>m;
    G.assign(n+1,vll());
    for(int i=1;i<=m;i++){
        cin>>x;
        G[x].pb(i);
    }
    for(int i=1;i<=n;i++){
        cin>>E[i];
    }
    ll l,r,val;
    cin>>k;
    for(int i=0;i<k;i++){
        cin>>l>>r>>val;
        Q.pb({{l,r},val});
    }
    vll aux;
    for(int i=0;i<=n;i++)res[i]=k+1;
    for(int i=1;i<=n;i++)aux.pb(i);
    parallel(0,k-1,aux);
    for(int i=1;i<=n;i++){
        if(res[i]==k+1){
            cout<<"NIE"<<"\n";
        }else{
            cout<<res[i]<<"\n";
        }
    }
    return 0;
}

//parallel binary search
// Complexity : O (Q+N) log N * Log Q (log M es por las
queries y update de BIT, N tamanio array, Q numero updates
donde aplico D&C)
//https://oj.uz/problem/view/POI11_met
```

7.20 parallel dsu

```
// Para establecer que dos substrings/subarreglos son
iguales
// Mantener conjuntos las posiciones que obligatoriamente
tienen que ser iguales
// para s[p1, p1+1, ..., p1+len-1] = s[p2, p2+1, ...,
p2+len-1]
// es qeuivalente a hacer _union(p1, p2), _union(p1+1,
p2+2) ...
// Nota .- Para problemas de palindromos puedes concatenar
el reverse del string al final
// index-0

struct DSU {
    vi P;
    void init(int N) {
        P.resize(N+2);
        for(int i=0;i<=N+1;i++)P[i]=i;
    }
    int _find(int nodo) {
        if(P[nodo]==nodo)return nodo;
        return P[nodo]=_find(P[nodo]);
    }
    void _union(int a, int b) {
        a=_find(a); b=_find(b);
        P[b]=a;// para palindromos (primera mitad padres)
    }
};

int n; // tamanio del string
DSU nivel[22];

// s[p1, p1+1, ..., p1+len-1] = s[p2, p2+1, ..., p2+len-1]
void equal(int p1, int p2, int len){// para definir dos
substrings iguales
    int k=0;
    while((1<<(k+1))<=len) k++;
    nivel[k]._union(p1, p2);
    nivel[k]._union(p1+len-(1<<k), p2+len-(1<<k));
}

void build(){// no olvidar llamar
    for(int k=20;k>=1;k--){
        for(int i=0;i<=n-(1<<k);i++){
            int j = nivel[k]._find(i);
            nivel[k-1]._union(i, j);
            nivel[k-1]._union(i+(1<<(k-1)), j+(1<<(k-1)));
        }
    }
}

int inv(int pos){// para palindromos
    return n - 1 - pos;
}

// main
for(int i=0;i<=20;i++){
    nivel[i].init(n);
}
for(int i=0;i<ns;i++){// para palindromos
    equal(i, inv(i), 1);
}
```

7.21 puentes

```
// si es grafo con aristas multiples (a,b) , (a,b)
// entonces usar una mapa de pares y si una arista aparece
dos veces no puede ser puente
const int tam=2e5+5;
set<pair<int, int>> st; // puente arista entre (a, b)
vi G[tam];
int arc[tam], IN[tam];
int tiempo=0;
void dfs(int nodo, int ant){
  tiempo++;
  IN[nodo] = arc[nodo] = tiempo;
  for(auto it : G[nodo]){
    if(it == ant) continue;
    if(IN[it]){
      arc[nodo] = min(arc[nodo], IN[it]);
    } else {
      dfs(it, nodo);
      arc[nodo] = min(arc[nodo], arc[it]);
      if(arc[it] > IN[nodo]){
        st.insert({nodo, it});
      }
    }
  }
}
```

7.22 puntos de articulacion

```
vector<vi> G;
vi vis, arc;
vector<bool> check;
int num=0;
void dfs(int nodo, int ant){
  num++;
  vis[nodo]=arc[nodo]=num;
  int hijos=0;
  for(auto it : G[nodo]){
    if(ant==it)continue;
    if(vis[it]){
      // si ya fue visitado entonces es un puente hacia
"atras"
      arc[nodo]=min(arc[nodo],vis[it]);
    }else{
      hijos++;
      dfs(it,nodo);
      arc[nodo]=min(arc[nodo],arc[it]);// para ver si su
padre de nodo es punto, por la pila recursiva
      if(ant!=-1 && arc[it]>=vis[nodo]){
        // entra al if si el puente mayor esta debajo del
nodo
        check[nodo]=1;
      }
    }
  }
  if(ant==-1 && hijos>1){
    // esto no cuenta los vecinos, si no los "subconjuntos"
que une la raiz
    check[nodo]=1;
  }
}
int main()
{
  int n, m, a, b;
  cin >> n >> m;
  arc.resize(n+1); vis.resize(n+1);
  check.assign(n+1, false);
  G.assign(n+1, vi());
  for(int i=0; i<m; i++){
    cin >> a >> b;
    G[a].pb(b);
    G[b].pb(a);
  }
  dfs(1, -1);
  for(int i=1; i<=n; i++){
    cout << check[i] << " ";
  }
  return 0;
}
```

7.23 sort c clockwise

```
bool up(Point a) {
  return a.y > 0 || (a.y == 0 && a.x >= 0);
}
bool cmp(Point a, Point b) {
  if (up(a) != up(b)) return up(a) > up(b);
  return cross(a, b) > 0;
}


// this starts from the half line x<=0, y=0
int group(Point a) {
  if (a.y < 0) return -1;
  if (a.y == 0 && a.x >= 0) return 0;
  return 1;
}
bool cmp(Point a, Point b) {
  if (group(a) == group(b)) return cross(a, b) > 0;
  return group(a) < group(b);
}
```

7.24 sos dp

```
// iterative version
for (int mask = 0; mask < (1 << N); ++mask) {
  dp[mask][-1] = A[mask];  // handle base case separately
(leaf states)
  for (int i = 0; i < N; ++i) {
    if (mask & (1 << i))
      dp[mask][i] = dp[mask][i - 1] + dp[mask ^ (1 << i)][i
- 1];
    else
      dp[mask][i] = dp[mask][i - 1];
  }
  F[mask] = dp[mask][N - 1];
}

// memory optimized, super easy to code.
for (int i = 0; i < (1 << N); ++i)
  F[i] = A[i];
```

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

7.25 suffix array nuevo

```
const int alpha = 400;
struct suffix_array { // s MUST not have 0 value
  vector<int> sa, rank, lcp;
  suffix_array(string s) {
    s.push_back('$'); // always add something less to input,
so it stays in pos 0
    int n = s.size(), mx = max(alpha, n)+2;
    vector<int> a(n), a1(n), c(n+1), c1(n+1), head(mx),
cnt(mx);
    rank = lcp = a;
    for(int i = 0; i < n; i++) c[i] = s[i], a[i] = i,
cnt[ c[i] ]++;
    for(int i = 0; i < mx-1; i++) head[i+1] = head[i] +
cnt[i];
    for(int k = 0; k < n; k = max(1ll, k<<1)) {
      for(int i = 0; i < n; i++) {
        int j = (a[i] - k + n) % n;
        a1[ head[ c[j] ]++ ] = j;
      }
      swap(a1, a);
      for(int i = 0, x = a[0], y, col = 0; i < n; i++, x =
a[i], y = a[i-1]) {
        c1[x] = (i && c[x] == c[y] && c[x+k] == c[y+k]) ?
col : ++col;
        if(!i || c1[x] != c1[y]) head[col] = i;
      }
      swap(c1, c);
      if(c[ a[n-1] ] == n) break;
    }
    swap(sa, a);
    for(int i = 0; i < n; i++) rank[ sa[i] ] = i;
    for(int i = 0, k = 0, j; i < n; lcp[ rank[i++] ] = k)
{ /// lcp[i, i+1]
      if(rank[i] == n-1) continue;
      for(k = max(0ll, k-1), j = sa[ rank[i]+1 ]; s[i+k] ==
s[j+k]; k++);
    }
  }
  int& operator[] ( int i ){ return sa[i]; }
};

//      012345  6
//      ababba  $
//      5. a
//      0. ababba
//      2. abba
//      4. ba
```

```
//      1. babba
//      3. bba
// sa   = 6 5 0 2 4 1 3
// lcp  = 0 1 2 0 2 1 0
// rank = 2 5 3 6 4 1 0   posicion del sufixx i en el sa
// lcp[i] = lcp(sa[i],sa[i+1])
```

8 Math

8.1 catalan convolution

```
/*
Return Catalan Convolution.

Convolution for k=3
((( A ) B ) C ) D

Where A + B + C + D = N, for N + 1
*/

const int MOD = 1e9 + 7;
ll mul(ll x, ll y) { return (x*y)%MOD; }
ll pot(ll x, ll y) {
    if(y==0) return 1;
    ll ans = pot(x,y/2);
    ans = mul(ans,ans);
    if (y&1)ans=mul(ans,x);
    return ans;
}
ll inv(ll x) { return pot(x, MOD-2); }


// mxN it the double of the max input N, plus max K
const int mxN = 2e6 + 1e6 + 10;
vl fact(mxN,1);
ll cnk(ll n, ll k) {
    if (k < 0 || n < k) return 0;
    ll nOverK = mul(fact[n],inv(fact[k]));
    return mul(nOverK,inv(fact[n-k]));
}

void init() {
    for (int i =1;i<=mxN;i++) {
        fact[i] = mul(fact[i-1],i);
    }
}

// for parethesis example
// number of n+k pairs having k open parethesis at beginning

// (cnk(2n+k,n)*(k+1))/(n+k+1)
ll catalanCov(ll n, ll k) {
    ll up = mul(cnk(2*n+k,n),(k+1)%MOD);
    ll down = (n+k+1)%MOD;
    return mul(up,inv(down));
}

/*
6
(()

ans: 2
*/
// size, and prefix
ll countParenthesisWithPrefix(ll n, string &p) {
    if (n&1) return 0;
    ll k = 0;
    for (auto c : p) {
        if (c=='(') k++;
        else k--;
        if (k<0) return 0;
    }
    n=(n-(ll)p.size()-k)/2;
    return catalanCov(n,k);
}
```

8.2 catalan

```
/*Catalan, counts the number of ways of:
( A ) B, where |A|+|B| = N, for N+1
*/

const int MOD = 1e9 + 7;
ll mul(ll x, ll y) { return (x*y)%MOD; }
ll pot(ll x, ll y) {
    if(y==0) return 1;
    ll ans = pot(x,y/2);
    ans = mul(ans,ans);
    if (y&1)ans=mul(ans,x);
    return ans;
}
ll inv(ll x) { return pot(x, MOD-2); }

// mxN it the double of the max input 'n'
const int mxN = 2e6 + 10;
vl fact(mxN,1);
void init() {
    for (int i =1;i<=mxN;i++) {
        fact[i] = mul(fact[i-1],i);
    }
}

ll catalan(ll n) {
    if (n<0) return 0;
    ll up = fact[2*n];
    ll down = mul(fact[n],fact[n+1]);
    return mul(up,inv(down));
}
```

8.3 combinatorics

```
// if k == 0 then 1
// if k negative or no enough choices then 0
// O(min(n, n-k)) lineal
ll nck(ll n, ll k) {
    if (k < 0 || n < k) return 0;
    k = min(k, n-k);
    ll ans = 1;
    for (int i = 1; i <= k; i++) {
        ans = ans * (n-i+1) / i;
    }
    return ans;
}
```

8.4 count primes with pi function

```
// sprime.count_primes(n);
// O(n^(2/3))
// PI(n) = Count prime numbers until n inclusive

struct count_primers_struct {
    vector<int> primes;
    vector<int> mnprimes;
    ll ans;
    ll y;
    vector<pair<pair<ll, int>, char>> queries;

    ll count_primes(ll n) {
        // this y is actually n/y
        // also no logarithms, welcome to reality, this y is
the best for n=10^12 or n=10^13
        y = pow(n, 0.64);
        if (n < 100) y = n;

        // linear sieve
        primes.clear();
        mnprimes.assign(y + 1, -1);
        ans = 0;
        for (int i = 2; i <= y; ++i) {
            if (mnprimes[i] == -1) {
                mnprimes[i] = primes.size();
                primes.push_back(i);
            }
            for (int k = 0; k < primes.size(); ++k) {
                int j = primes[k];
                if (i * j > y) break;
                mnprimes[i * j] = k;
                if (i % j == 0) break;
            }
        }
        if (n < 100) return primes.size();
        ll s = n / y;

        for (int p : primes) {
            if (p > s) break;
            ans++;
        }
        // pi(n / y)
        int ssz = ans;

        // F with two pointers
        int ptr = primes.size() - 1;
        for (int i = ssz; i < primes.size(); ++i) {
            while (ptr >= i && (ll)primes[i] * primes[ptr] >
n)
                --ptr;
            if (ptr < i) break;
            ans -= ptr - i + 1;
        }

        // phi, store all queries
        phi(n, ssz - 1);

        sort(queries.begin(), queries.end());
        int ind = 2;
        int sz = primes.size();

        // the order in fenwick will be reversed, because
prefix sum in a fenwick is just one query
        fenwick fw(sz);
        for (auto qq : queries) {
            auto na = qq.F;
            auto sign = qq.S;
            auto n = na.F;
            auto a = na.S;
            while (ind <= n)
                fw.add(sz - 1 - mnprimes[ind++], 1);
            ans += (fw.ask(sz - a - 2) + 1) * sign;
        }
        queries.clear();
        return ans - 1;
    }

    void phi(ll n, int a, int sign = 1) {
        if (n == 0) return;
        if (a == -1) {
            ans += n * sign;
            return;
        }
        if (n <= y) {
            queries.emplace_back(make_pair(n, a), sign);
            return;
        }
        phi(n, a - 1, sign);
        phi(n / primes[a], a - 1, -sign);
    }

    struct fenwick {
        vector<int> tree;
        int n;

        fenwick(int n = 0) : n(n) {
            tree.assign(n, 0);
        }

        void add(int i, int k) {
            for (; i < n; i = (i | (i + 1)))
                tree[i] += k;
        }

        int ask(int r) {
            int res = 0;
            for (; r >= 0; r = (r & (r + 1)) - 1)
                res += tree[r];
            return res;
        }
    };
} ;

count_primers_struct sprime;
```

8.5 fast fibonacci

```
// Fast Fibonacci O(log n)
// Use fib(n).F to get the at nth position
pair<ll,ll> fib (ll n) {
    if (n == 0)
        return {0, 1};

    auto p = fib(n >> 1);
    ll c = (p.F * (2*p.S - p.F + MOD)%MOD)%MOD;
    ll d = (p.F * p.F + p.S * p.S)%MOD;
    if (n & 1)
        return {d, (c + d)%MOD};
    else
        return {c, d};
}
/* Fib properties
Addition Rule: F_n+k = F_k * F_n+1 + F_k-1 * F_n
F_2n= Fn * (F_n+1 + F_n-1)

GCD Identity: GCD(F_m,F_n) = F_gcd(m,n)
Cassinis' identity: F_n-1 * F_n+1 - F_n*F_n = (-1)^n
*/
```

8.6 floor sums

```
// from atcoder
// floor_sum(n,m,a,b) = sum{0}to{n-1} [(a*i+b)/m]
// O(log m), mod 2^64, n<2^32, m<2^32

constexpr long long safe_mod(long long x, long long m) {
    x %= m;
    if (x < 0) x += m;
    return x;
}

unsigned long long floor_sum_unsigned(unsigned long long n,
                                      unsigned long long m,
                                      unsigned long long a,
                                      unsigned long long b)
{
    unsigned long long ans = 0;
    while (true) {
        if (a >= m) {
            ans += n * (n - 1) / 2 * (a / m);
            a %= m;
        }
        if (b >= m) {
            ans += n * (b / m);
            b %= m;
        }

        unsigned long long y_max = a * n + b;
        if (y_max < m) break;
        // y_max < m * (n + 1)
        // floor(y_max / m) <= n
        n = (unsigned long long)(y_max / m);
        b = (unsigned long long)(y_max % m);
        swap(m, a);
    }
    return ans;
}

long long floor_sum(long long n, long long m, long long a,
long long b) {
    assert(0 <= n && n < (1LL << 32));
    assert(1 <= m && m < (1LL << 32));
    unsigned long long ans = 0;
    if (a < 0) {
        unsigned long long a2 = safe_mod(a, m);
        ans -= 1ULL * n * (n - 1) / 2 * ((a2 - a) / m);
        a = a2;
    }
    if (b < 0) {
        unsigned long long b2 = safe_mod(b, m);
        ans -= 1ULL * n * ((b2 - b) / m);
        b = b2;
    }
    return ans + floor_sum_unsigned(n, m, a, b);
}
```

8.7 optimized polard rho

```
// Fast factorization with big numbers, use fact method
// Seems to be O(log^3(n)) !!! Need revision
#define fore(i, b, e) for(int i = b; i < e; i++)
ll gcd(ll a, ll b){return a?gcd(b%a,a):b;}
ll mulmod(ll a, ll b, ll m) {
  ll r=a*b-(ll)((long double)a*b/m+.5)*m;
  return r<0?r+m:r;
}
ll expmod(ll b, ll e, ll m){
  if(!e)return 1;
  ll q=expmod(b,e/2,m);q=mulmod(q,q,m);
  return e&1?mulmod(b,q,m):q;
}
bool is_prime_prob(ll n, int a){
  if(n==a)return true;
  ll s=0,d=n-1;
  while(d%2==0)s++,d/=2;
  ll x=expmod(a,d,n);
  if((x==1)||(x+1==n))return true;
  fore(_,0,s-1){
    x=mulmod(x,x,n);
    if(x==1)return false;
    if(x+1==n)return true;
  }
  return false;
}
bool rabin(ll n){ // true iff n is prime
  if(n==1)return false;
  int ar[]={2,3,5,7,11,13,17,19,23};
  fore(i,0,9)if(!is_prime_prob(n,ar[i]))return false;
  return true;
}
// optimized version: replace rho and fact with the
following:
const int MAXP=1e6+1; // sieve size
int sv[MAXP]; // sieve
ll add(ll a, ll b, ll m){return (a+=b)<m?a:a-m;}
ll rho(ll n){
  static ll s[MAXP];
  while(1){
    ll x=rand()%n,y=x,c=rand()%n;
    ll *px=s,*py=s,v=0,p=1;
    while(1){
      *py++=y=add(mulmod(y,y,n),c,n);
      *py++=y=add(mulmod(y,y,n),c,n);
      if((x=*px++)==y)break;
      ll t=p;
      p=mulmod(p,abs(y-x),n);
      if(!p)return gcd(t,n);
      if(++v==26){
        if((p=gcd(p,n))>1&&p<n)return p;
        v=0;
      }
    }
    if(v&&(p=gcd(p,n))>1&&p<n)return p;
  }
}
void init_sv(){
  fore(i,2,MAXP)if(!sv[i])for(ll j=i;j<MAXP;j+=i)sv[j]=i;
}
```

```
void fact(ll n, map<ll,int>& f){ // call init_sv first!!!
  for(auto&& p:f){
    while(n%p.F==0){
      p.S++; n/=p.F;
    }
  }
  if(n<MAXP)while(n>1)f[sv[n]]++,n/=sv[n];
  else if(rabin(n))f[n]++;
  else {ll q=rho(n);fact(q,f);fact(n/q,f);}
}
```

8.8 subfactorial

```
/* Denote as !n or derangement numbers
  Count the number of permutations where no element is in
  the originial position, formally p[i] != i

  it can be seen as f(n) = n!-sum_i=1_to_n { cnk(n,i)*f(n-
i) }
  f(0)=1, f(1) = 1

  1 0 1 2 9 44 265 1,854 14,833 133,496

  n! = sumi=0,i<=n,{cnk(n,i)!i}
  d[i] = (d[i-1]+d[i-2])*(i-1)
*/
const int mxN = 2e6 + 10; // max number
ll add(ll x, ll y) { return (x+y)%MOD; }
ll mul(ll x, ll y) { return (x*y)%MOD; }
vl subFact(mxN);
void init() {
    subFact[0] = 1;
    subFact[1] = 0;
    for (int i = 2;i<mxN;i++) {
        subFact[i] =
mul(add(subFact[i-1],subFact[i-2]),i-1);
    }
}
```

8.9 ternary search

```
// this is for find minimum point in a parabolic
// O(log3(n))
// TODO: Improve to generic!!!
ll left = 0;
ll right = n - 1;
while (left + 3 < right) {
    ll mid1 = left + (right - left) / 3;
    ll mid2 = right - (right - left) / 3;
    if (f(b, lines[mid1]) <= f(b, lines[mid2])) {
        right = mid2;
    } else {
        left = mid1;
    }
}
ll target = -4 * a * c;
ll ans = -1; // find the answer, in this case any works.
for (ll mid = left; mid <= right; mid++) {
    if (f(b, lines[mid]) + target < 0) {
        ans = mid;
    }
}
```

9 Strings

9.1 fast hashing

```
const int N = 1e6 + 9; // Max size

int power(long long n, long long k, const int mod) {
  int ans = 1 % mod;
  n %= mod;
  if (n < 0) n += mod;
  while (k) {
    if (k & 1) ans = (long long) ans * n % mod;
    n = (long long) n * n % mod;
    k >>= 1;
  }
  return ans;
}

const int MOD1 = 127657753, MOD2 = 987654319;
const int p1 = 137, p2 = 277;
int ip1, ip2;
pair<int, int> pw[N], ipw[N];
void init() { // Call init() first!!!
  pw[0] =  {1, 1};
  for (int i = 1; i < N; i++) {
    pw[i].first = 1LL * pw[i - 1].first * p1 % MOD1;
    pw[i].second = 1LL * pw[i - 1].second * p2 % MOD2;
  }
  ip1 = power(p1, MOD1 - 2, MOD1);
  ip2 = power(p2, MOD2 - 2, MOD2);
  ipw[0] =  {1, 1};
  for (int i = 1; i < N; i++) {
    ipw[i].first = 1LL * ipw[i - 1].first * ip1 % MOD1;
    ipw[i].second = 1LL * ipw[i - 1].second * ip2 % MOD2;
  }
}
struct Hashing {
  int n;
  string s; // 0 - indexed
  vector<pair<int, int>> hs; // 1 - indexed
  Hashing() {}
  Hashing(string _s) {
    n = _s.size();
    s = _s;
    hs.emplace_back(0, 0);
    for (int i = 0; i < n; i++) {
      pair<int, int> p;
      p.first = (hs[i].first + 1LL * pw[i].first * s[i] %
MOD1) % MOD1;
      p.second = (hs[i].second + 1LL * pw[i].second * s[i] %
MOD2) % MOD2;
      hs.push_back(p);
    }
  }
  pair<int, int> get_hash(int l, int r) { // 1-indexed
    assert(1 <= l && l <= r && r <= n);
    pair<int, int> ans;
    ans.first = (hs[r].first - hs[l - 1].first + MOD1) * 1LL
* ipw[l - 1].first % MOD1;
    ans.second = (hs[r].second - hs[l - 1].second + MOD2) *
1LL * ipw[l - 1].second % MOD2;
    return ans;
  }
  pair<int,int> get(int l, int r) { // 0-indexed
    return get_hash(l+1,r+1);
  }
  pair<int, int> get_hash() {
    return get_hash(1, n);
  }
};
```

9.2 kmp automaton

```
// Very useful for some DP's with strings
// aut[i][j], you are in 'i' position, and choose character
'j', the next position.

const int MAXN = 1e5 + 5, alpha = 26;
const char L = 'A';
int aut[MAXN][alpha]; // aut[i][j] = a donde vuelvo si estoy
en i y pongo una j

void build(string &s) {
    int lps = 0;
    aut[0][s[0]-L] = 1;
    int n = s.size();
    for (int i = 1; i < n+1; i++) {
        for (int j = 0; j < alpha; j++) aut[i][j] = aut[lps]
[j];
        if (i < n) {
            aut[i][s[i]-L] = i + 1;
            lps = aut[lps][s[i]-L];
        }
    }
}
```

9.3 kmp

```
// Given string s, and patter p, count and find
// occurences of p in s. O(n)
struct KMP {
    int kmp(vector<ll> &s, vector<ll> &p) {
        int n = s.size(), m = p.size(), cnt = 0;
        vector<int> pf = prefix_function(p);
        for(int i = 0, j = 0; i < n; i++) {
            while(j && s[i] != p[j]) j = pf[j-1];
            if(s[i] == p[j]) j++;
            if(j == m) {
                cnt++;
                j = pf[j-1];
            }
        }
        return cnt;
    }

    vector<int> prefix_function(vector<ll> &s) {
        int n = s.size();
        vector<int> pf(n);
        pf[0] = 0;
        for (int i = 1, j = 0; i < n; i++) {
            while (j && s[i] != s[j]) j = pf[j-1];
            if (s[i] == s[j]) j++;
            pf[i] = j;
        }
        return pf;
    }
};
```

9.4 todo add manacher, ahocorasick, suffix array

10 Tree

10.1 euler tour sum

```
/* Euler Tour Sum Path O(n log n)
Given a Tree, and values in each node, process this queries:
 - Calculate the sum of values in the Path 1 to a 'given
node'.
 - Update value of a node

Also you can extend this to sum path from A to B with
updates.
Just add LCA and sum(0,A) + sum(0,B) - 2*(sum(0,lca)-
values[lca])

Tested: https://cses.fi/problemset/task/1138/
*/
void test_case() {
    ll n, m; cin >> n >> m;
    vector<vl> adj(n+1); // 1-indexed
    vl nums(n+1), tin(n+1), tout(n+1);
    FenwickTree tree(2*n+2); // Add Fenwick Tree 0-indexed

    for (int i = 1; i <= n; i++) cin >> nums[i];
    for (int i = 0; i < n-1; i++) {
        ll x, y; cin >> x >> y;
        adj[x].pb(y); adj[y].pb(x);
    }

    ll time = 0;
    function<void(int,int)> dfs =[&](int x, int p) {
        tin[x] = time++;
        for (auto y : adj[x]) if (y != p) dfs(y, x);
        tout[x] = time++;
        tree.add(tin[x], nums[x]);
        tree.add(tout[x], -nums[x]);
    };
    dfs(1, 0);
    for (int i =0; i < m; i++) {
        ll t, x;
        cin >> t >> x;
        if (t == 1) { // update
            ll y; cin >> y;
            ll diff = y - nums[x];
            nums[x] = y;
            tree.add(tin[x], diff);
            tree.add(tout[x], -diff);
        } else { // query
            cout << tree.sum(0, tin[x]) << "\n";
        }
    }
}
```

10.2 k th parent

```
/* K-th Parent.cpp
Given and Tree, and Q queries to see the K-Parent of a node
*/
const int LOG = 20;
vvl parent(LOG, vl(2e5 + 10, -1));

void test_case() { // 1-based indexed
    ll n, q; cin >> n >> q;

    for (int i = 0; i < n - 1; i++)
        cin >> parent[0][i+2];

    for (int j = 1; j < LOG; j++) {
        for (int i = 1; i <= n; i++) {
            if (parent[j-1][i] == -1) continue;
            parent[j][i] = parent[j-1][parent[j-1][i]];
        }
    }
    for (int i = 0; i < q; i++) { // Queries
        ll x, k;
        cin >> x >> k;
        ll ans = x;
        ll y = 0;
        while (k) {
            if (k&1) {
                ans = parent[y][ans];
            }
            if (ans == -1) break;
            k /= 2;
            y++;
        }
        cout << ans << "\n";
    }
}
```

10.3 lowest common ancestor

```
// Give a rooted tree, find the Lowest Common Ancestor node
// of A and B.
// Tested https://cses.fi/problemset/task/1688/
vector<vector<ll>> up;
vector<ll> depth;
const int LOG = 18; // for 2e5

void init(vector<vector<ll>> children, ll n) {
    up.assign(LOG,vector<ll>(n,0));
    depth.assign(n,0);
    function<void(int)> dfs = [&](int x) {
        for (auto y : children[x]) {
            up[0][y] = x;
            depth[y] = depth[x] +1;
            dfs(y);
        }
    };
    int root = 0; dfs(root);

    for (int i = 1; i < LOG; i++)
        for (int j = 0; j < n; j++)
            up[i][j] = up[i-1][up[i-1][j]];
}

ll kParent(ll x, ll k) {
    ll i = 0;
    while (k) {
        if (k & 1) x = up[i][x];
        k >>= 1;
        i++;
    }
    return x;
}

ll query(ll x, ll y) {
    if (depth[x] < depth[y]) swap(x, y);
    x = kParent(x, depth[x] - depth[y]);
    if (x == y) {
        return x;
    }
    for (int i = LOG - 1; i >= 0; i--) {
        if (up[i][x] != up[i][y]) {
            x = up[i][x];
            y = up[i][y];
        }
    }
    return up[0][x];
}

void test_case() {
    ll n, q; cin >> n >> q;
    vvl children(n);
    for (int i = 1; i < n; i++) {
        ll p; cin >> p; p--;
        children[p].pb(i);
    }
    init(children, n);

    for (int i = 0; i < q; i++) {
        ll x, y;cin >> x >> y;
        x--,y--;
        cout << query(x, y) + 1 << "\n";
    }
}
```

10.4 todo add dsu, centroid, hld

11 Utils

11.1 bit tricks

```
y = x & (x-1)  // Turn off rightmost 1bit
y = x & (-x)   // Isolate rightmost 1bit
y = x | (x-1)  // Right propagate rightmost 1bit(fill in 1s)
y = x | (x+1)  // Turn on rightmost 0bit
y = ~x & (x+1) // Isolate rightmost 0bit
// If x is of long type, use __builtin_popcountl(x)
// If x is of long long type, use __builtin_popcountll(x)
// 1. Counts the number of oneâ€™s(set bits) in an integer.
__builtin_popcount(x)
// 2. Checks the Parity of a number. Returns true(1) if the
// number has odd number of set bits, else it returns
// false(0) for even number of set bits.
 __builtin_parity(x)
// 3. Counts the leading number of zeros of the integer.
__builtin_clz(x)
// 4. Counts the trailing number of zeros of the integer.
__builtin_ctz(x)
// 5. Returns 1 + the index of the least significant 1-bit.
__builtin_ffs(x) // If x == 0, returns 0.
// Iterate over non empty subsets of bitmask
```

```
for(int s=m;s;s=(s-1)&m) // Decreasing order
for(int s=0;s=s-m&m;) // Increasing order
```

11.2 io int128

```
// Read and Print integers of 128 bits
__int128 read() {
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}
void print(__int128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x > 9) print(x / 10);
    putchar(x % 10 + '0');
}

void print(__int128 x) {
    if (x < 0) {
        cout << "-";
        x = -x;
    }
    if (x > 9) print(x / 10);
    cout << char((int)(x % 10) + '0');
}
```

11.3 pragmas

```
//#pragma GCC target("popcnt")
//It's worth noting that after adding  __builtin_popcount()
is replaced to corresponding machine instruction (look at
the difference). In my test this maked x2 speed up.
bitset::count() use __builtin_popcount() call in
implementation, so it's also affected by this.
#pragma GCC target ("avx2")
#pragma GCC optimization ("O3")
#pragma GCC optimization ("unroll-loops")
#pragma GCC target("popcnt")
#pragma GCC
target("avx,avx2,sse3,ssse3,sse4.1,sse4.2,tune=native")
#pragma GCC optimize(3)
#pragma GCC optimize("O3")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")
```

11.4 randoms

```
// Get random numbers between [a, b]
mt19937
mt_rng(chrono::steady_clock::now().time_since_epoch().count());
// also for ll exists mt19937_64
ll randint(ll a, ll b) {
    return uniform_int_distribution<ll>(a, b)(mt_rng);
}
```

11.5 string streams

```
// For some complex reading of input
// st is the same as a cin, but you pass the string
string line;
getline(cin, line);
stringstream st(line);
vl in;
ll x;
```

```
while (st >> x) {
    in.pb(x);
}
```