Team notebook

Universidad de la Amazonia

October 14, 2025

Contents

1 Data Structures

1.1 Eulerian Path

```
// Camino de Euler / Ciclo de Euler

//Validar que el ciclo inicie con el mismo src
//Validar que el camino termine con n - 1 en caso de ser obligatorio

// Declarar: 'euler<true> E(n);' si se desea
// dirigido y con 'n' vrtices.
// Las funciones devuelven un par con un booleano
// que indica si tiene el ciclo/camino solicitado,
// y un vector de {vrtice, ID de la arista para alcanzar el vrtice}.
```

```cpp
// Si es get_path, en la primera posicin el ID ser -1.
// get_path(src) intenta encontrar un camino o ciclo euleriano
// comenzando en el vrtice 'src'.
// Si encuentra un ciclo, el primer y el ltimo vrtice sern 'src'.
// Si es un P3, un posible retorno sera [0, 1, 2, 0]
// get_cycle() encuentra un ciclo euleriano si el grafo es euleriano.
// Si es un P3, un posible retorno sera [0, 1, 2]
// (el vrtice inicial no se repite).

// O(n+m)

using pii = pair<int, int>;

template <bool directed = false>
struct euler {
    int n;
    vector<vector<pii>> g;
    vi used;

    euler(int n_) : n(n_), g(n) {}
    void add(int a, int b) {
        int at = sz(used);
        used.push_back(0);
        g[a].emplace_back(b, at);
        if(!directed) g[b].emplace_back(a, at);
    }
    // #warning chamar para o src certo!
    pair<bool, vector<pii>> get_path(int src) {
        if(!sz(used)) return {true, {}};
        vi beg(n, 0);
        for(int& i : used) i = 0;
        // {{vertice, anterior}, label}
        vector<pair<pii, int>> ret, st = {{{src, -1}, -1}};
        while(sz(st)) {
            int at = st.back().first.first;
            int& it = beg[at];
            while(it < sz(g[at]) and used[g[at][it].second]) it++;
            if(it == sz(g[at])) {
                if(sz(ret) and ret.back().first.second != at)
                    return {false, {}};
                ret.push_back(st.back()), st.pop_back();
            } else {
                st.push_back({{g[at][it].first, at}, g[at][it].second});
                used[g[at][it].second] = 1;
            }
        }
        if(sz(ret) != sz(used) + 1) return {false, {}};
        vector<pii> ans;
        for(auto i : ret) ans.emplace_back(i.first.first, i.second);
        reverse(all(ans));
        return {true, ans};
    }
    pair<bool, vector<pii>> get_cycle() {
        if(!sz(used)) return {true, {}};
        int src = 0;
        while(!sz(g[src])) src++;
        auto ans = get_path(src);
        if(!ans.first or ans.second[0].first != ans.second.back().first)
            return {false, {}};
        ans.second[0].second = ans.second.back().second;
        ans.second.pop_back();
        return ans;
    }
};

int main() {
    int nodos, m;
    cin >> nodos >> m;
    euler<true> ee(nodos);
    forn(i, m) {
        int a, b;
        cin >> a >> b;
        a--, b--;
        ee.add(a, b); // agregar arista
    }

    pair<bool, vector<pii>> ans = ee.get_path(0);
    //se valida que haya camino y que termine en n - 1
    if(ans.first && ans.second.back().first == n - 1) {
        for(auto [u, id] : ans.second) {
            cout << u + 1 << " ";
        }
        cout << ln;
    } else {
        cout << "IMPOSSIBLE" << ln;
    }
    return 0;
}

/*
```

```
    EULERIAN PATH
    Un camino euleriano es un camino de aristas
    que visita todas las aristas de un grafo
    exactamente una vez.

    UNDIRECTED GRAPH

    o bien cada vertice tiene un grado par o
    exactamente dos vertices tienen un grado impar.

    DIRECTED GRAPH

    como mximo un vrtice tiene (grado de salida) - (grado de entrada) = 1
    y como mximo un vrtice tiene (grado de entrada) - (grado de salida)
        = 1,
    y todos los dems vrtices tienen grados de entrada y salida iguales.
*/

// EULERIAN PATH DIRECTED GRAPH O(E)

/*
    Eulerian CIRCUIT
    Un circuito euleriano es un camino euleriano
    que comienza y termina en el mismo vrtice.

    UNDIRECTED GRAPH

    Cada vertice tiene un grado par.

    DIRECTED GRAPH

    Cada vrtice tiene el mismo grado de entrada y de salida.
*/
```

1.2 Fenwick Tree

```
//[1, n]
//rangos [l, r]

template <typename T>
struct BIT {
    vector<T> ft;
    int N;
    BIT(int n): ft(n + 1), N(n) {}

    BIT(const vector<T>& a): ft(sz(a) + 1), N(sz(a)) {
        forn(i, sz(a)) {
            upd(i + 1, a[i]);
        }
    }

    T qry(int i) {
        T ans = 0;
        for (; i; i -= i & -i) ans += ft[i];
        return ans;
    }

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

    void upd(int i, T v) {
        for (; i < sz(ft); i += i & -i) ft[i] += v;
    }
    //This is equivalent to calculating
    //lower_bound on prefix sums array
    int bit_search(int v) { // O(log(N))
        int sum = 0;
        int pos = 0;
        int LOGN = int(log2(N));
        for (int i = LOGN; i >= 0; i--) {
            ll acum = sum + ft[pos + (1 << i)];
            if (pos + (1 << i) < N && acum < v) {
                sum += ft[pos + (1 << i)];
                pos += (1 << i);
            }
        }
        return pos + 1;
    }
    //sin palabras x
};
```

1.3 Fenwick$_{r}ange_{u}pd$

```
// [1, n]
//
template <typename T>
struct ft_range {
    vector<T> ft1, ft2;

    ft_range(int n) {
        ft1.assign(n + 1, 0);
        ft2.assign(n + 1, 0);
    }

    ft_range(vector<T> &a) : ft1(sz(a) + 1), ft2(sz(a) + 1) {
        forab(i, 1, sz(a) + 1)
        update(i, i, a[i]);
    }

    T query(vector<T> & ft, int i) {
        T sum = 0;
        for(; i; i -= (i & -i)) sum += ft[i];
        return sum;
    }
    void update(vector<T> & ft, int i, int v) {
        for(; i < sz(ft); i += (i & -i))
            ft[i] += v;
    }
    void update(int i, int j, T v) {
        update(ft1, i, v);
        update(ft1, j + 1, -v);
        update(ft2, i, v * (i - 1));
        update(ft2, j + 1, -v * j);
    }
    T query(int i) {
        return query(ft1, i) * i - query(ft2, i);
    }
    T query(int i, int j) {
        return query(j) - query(i - 1);
    }
};

int main() {
    int n;
    cin >> n;
    vector<int> v(n + 1);
    forab(i, 1, n + 1) cin >> v[i];
    ft_range<int> bit(v);
    int q;
    cin >> q;
    int l, r, val;
    while(q--) {
        char op;
        cin >> op;
        if(op == 'q') {
            cin >> l >> r;
            cout << bit.query(l, r) << ln;
        } else {
            cin >> l >> r >> val;
            bit.update(l, r, val);
        }
    }
    return 0;
}
```

1.4 Heavy Light Decomposition

```
//tomado de https://github.com/brunomaletta

//Segtree lazy code here

const int MAXN = 2e5 + 10;
vi g[MAXN];

int val[MAXN]; // value of each node
vll st_val(MAXN); // STree array
STree<ll> st(MAXN); // segtree

struct HLD {
        vi in, tam, par, hp;
        int timer;

        HLD(int root, int n){
        timer = 0;
        in.resize(n); //tin
        tam.resize(n); //subtree size
        par.resize(n); //parent
        hp.resize(n); //heavy path

        build_hld(root);
```

```
    st.build(1, 0, sz(st_val) - 1, st_val);
    }

//save all values first in st_val and then build segtree
//or keep updating the segment tree

    void build_hld(int u, int p = -1, int f = 1) {
           st_val[in[u] = timer++] = val[u];
    //st.upd(in[k], in[k], val[k]);
           tam[u] = 1;
           //dont forget the &
           //with edges use [v, w] and check pair(first, second)
           for (auto &v : g[u]) if (v != p) {
                  par[v] = u;
                  hp[v] = (v == g[u][0] ? hp[u] : v);
                  build_hld(v, u, f);
                  tam[u] += tam[v];

                  if (tam[v] > tam[g[u][0]] || g[u][0] == p) swap(v,
                      g[u][0]);
           }
           if (p * f == -1) build_hld(hp[u] = u, -1, timer = 0);
    }

    ll query_path(int a, int b) {
    //if (a == b) return 0;
           if (in[a] < in[b]) swap(a, b);

           if (hp[a] == hp[b]){
       return st.query(in[b], in[a]); // in[b] + 1 if the value is on
            the edge
           }
           return st.query(in[hp[a]], in[a]) + query_path(par[hp[a]],
               b);
    }

    void update_path(int a, int b, int x) {
    //if(a == b) return;
           if (in[a] < in[b]) swap(a, b);

           if (hp[a] == hp[b]){
       return (void) st.upd(in[b], in[a], x); // in[b] + 1 if the
            value is on the edge
           }
           st.upd(in[hp[a]], in[a], x);
           update_path(par[hp[a]], b, x);
    }

    ll query_subtree(int a) {
           return st.query(in[a], in[a] + tam[a] - 1);
    }

    void update_subtree(int a, int x) {
           st.upd(in[a], in[a] + tam[a] - 1, x);
    }

    int lca(int a, int b) {
           if (in[a] < in[b]) swap(a, b);
           return hp[a] == hp[b] ? b : lca(par[hp[a]], b);
    }
};

/*
If the value is on the edge,
the node that stores the value is the one at the greater depth,
and in queries and updates the higher one is excluded.
*/
```

1.5 MO's on Tree

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

const int N = 3e5 + 9;
//tomado de
//https://github.com/ShahjalalShohag/code-library/

// unique elements on the path from u to v
vector<int> g[N];
int st[N], en[N], T, par[N][20], dep[N], id[N * 2];
void dfs(int u, int p = 0) {
    st[u] = ++T;
    id[T] = u;
    dep[u] = dep[p] + 1;
    par[u][0] = p;
    for (int k = 1; k < 20; k++)
        par[u][k] = par[par[u][k - 1]][k - 1];
```

```
    for (auto v : g[u])
        if (v != p)
            dfs(v, u);
    en[u] = ++T;
    id[T] = u;
}
int lca(int u, int v) {
    if (dep[u] < dep[v])
        swap(u, v);
    for (int k = 19; k >= 0; k--)
        if (dep[par[u][k]] >= dep[v])
            u = par[u][k];
    if (u == v)
        return u;
    for (int k = 19; k >= 0; k--)
        if (par[u][k] != par[v][k])
            u = par[u][k], v = par[v][k];
    return par[u][0];
}

int cnt[N], a[N], ans;
inline void add(int u) {
    int x = a[u];
    if (cnt[x]++ == 0)
        ans++;
}
inline void rem(int u) {
    int x = a[u];
    if (--cnt[x] == 0)
        ans--;
}
bool vis[N];
inline void yo(int u) {
    if (!vis[u])
        add(u);
    else
        rem(u);
    vis[u] ^= 1;
}

const int B = 320;
struct query {
    int l, r, id;
    bool operator<(const query &x) const {
        if (l / B == x.l / B)
            return r < x.r;
        return l / B < x.l / B;
    }
} Q[N];

int res[N];
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    int n, q;
    while (cin >> n >> q) {
        for (int i = 1; i <= n; i++)
            cin >> a[i];
        map<int, int> mp;
        for (int i = 1; i <= n; i++) {
            if (mp.find(a[i]) == mp.end())
                mp[a[i]], mp[a[i]] = mp.size();
            a[i] = mp[a[i]];
        }
        for (int i = 1; i < n; i++) {
            int u, v;
            cin >> u >> v;
            g[u].push_back(v);
            g[v].push_back(u);
        }
        T = 0;
        dfs(1);
        for (int i = 1; i <= q; i++) {
            int u, v;
            cin >> u >> v;
            if (st[u] > st[v])
                swap(u, v);
            int lc = lca(u, v);
            if (lc == u)
                Q[i].l = st[u], Q[i].r = st[v];
            else
                Q[i].l = en[u], Q[i].r = st[v];
            Q[i].id = i;
        }
        sort(Q + 1, Q + q + 1);
        ans = 0;
        int l = 1, r = 0;
        for (int i = 1; i <= q; i++) {
            int L = Q[i].l, R = Q[i].r;
```

```
            if (R < l) {
                while (l > L)
                    yo(id[--l]);
                while (l < L)
                    yo(id[l++]);
                while (r < R)
                    yo(id[++r]);
                while (r > R)
                    yo(id[r--]);
            } else {
                while (r < R)
                    yo(id[++r]);
                while (r > R)
                    yo(id[r--]);
                while (l > L)
                    yo(id[--l]);
                while (l < L)
                    yo(id[l++]);
            }
            int u = id[l], v = id[r], lc = lca(u, v);
            if (lc != u && lc != v)
                yo(lc); // take care of the lca separately
            res[Q[i].id] = ans;
            if (lc != u && lc != v)
                yo(lc);
        }
        for (int i = 1; i <= q; i++)
            cout << res[i] << '\n';
        for (int i = 0; i <= n; i++) {
            g[i].clear();
            vis[i] = cnt[i] = 0;
            for (int k = 0; k < 20; k++)
                par[i][k] = 0;
        }
    }
    return 0;
}
/*
    MOs on tree
    dado un arbol de N nodos
    cada nodo tiene un valor
    nos preguntan
    cuantos valores diferentes
    hay en el camino de (u, v)
*/
```

1.6 Mo's

```
int S, n, q;

struct query {
    int l, r, idx;
    query(int l, int r, int idx): l(l), r(r), idx(idx) {}

    bool operator < (const query &x) const {
        if (l / S != x.l / S) return l / S < x.l / S;
        return (l / S & 1) ? r < x.r: r > x.r;
    }
};

const int MAXN = 1e6 + 1;
ll sum = 0;
vector<query> qu;
vector<ll> ans;
vi feq(MAXN, 0);
vll a;

void add(int idx) {
    ll val = a[idx];
    sum = sum - (val * feq[val] * feq[val]);
    feq[val]++;
    sum = sum + (val * feq[val] * feq[val]);
}

/*
Very common pattern:
Subtract power of element from total
Update count
Add back power of element to total
*/
void del(int idx) {
    ll val = a[idx];
    sum = sum - (val * feq[val] * feq[val]);
    feq[val]--;
    sum = sum + (val * feq[val] * feq[val]);
}
```

```
ll get_ans() {
    return sum;
}

void mo_s() {
    S = sqrt(n);
    sort(all(qu));
    ans.resize(q);
    int l = 0, r = -1;
    for (query &it: qu) {
        while (r < it.r) add(++r);
        while (l > it.l) add(--l);
        while (r > it.r) del(r--);
        while (l < it.l) del(l++);
        ans[it.idx] = get_ans();
    }
}

int main() {
    cin >> n >> q;
    a = vll(n);
    forn(i, n) cin >> a[i];

    int l, r;
    forn(i, q) {
        cin >> l >> r;
        l--;
        r--;
        qu.pb({l, r, i});
    }
    mo_s();
    forn(i, q) cout << ans[i] << ln;
    return 0;
}
```

1.7 Path sum with Fenwick

```
//use fenwick code here

//O(n*logn)
int n, m;
const int MAXN = 2e5 + 5;
const int LOG = 17;
vi g[MAXN];
int up[MAXN][LOG + 1];
int tin[MAXN], tout[MAXN];
int deep[MAXN];
int timer = 1;

void dfs(int u, int p) {
    up[u][0] = p;
    tin[u] = timer++;
    deep[u] = deep[p] + 1;
    forab(k, 1, LOG) {
        up[u][k] = up[up[u][k - 1]][k - 1];
    }
    for(int v: g[u]) {
        if(v == p) continue;
        dfs(v, u);
    }
    tout[u] = timer++;
}

int lca(int u, int v) {
    if (deep[u] < deep[v]) swap(u, v);
    int df = abs(deep[u] - deep[v]);
    for (int k = LOG - 1; k >= 0; k--) {
        if (df & (1LL << k)) {
            u = up[u][k];
        }
    }
    if (u == v) return u;
    for (int k = LOG - 1; k >= 0; k--) {
        if (up[u][k] != up[v][k]) {
            u = up[u][k];
            v = up[v][k];
        }
    }
    return up[u][0];
}

//add val to all nodes in path (a, b)
void upd_path(int a, int b, int val, BIT<int> &ft) {
    if(tin[b] < tin[a]) swap(a, b);
    int lc = lca(a, b);
    ft.upd(tin[a], + val);
```

```
    ft.upd(tin[b], + val);
    ft.upd(tout[lc], - val);
    ft.upd(tin[lc], - val);
}

//query the node value
int get_val_node(int u, BIT<int> &ft) {
    int x = ft.qry(tout[u] - 1) - ft.qry(tin[u] - 1);
    return x;
}

int main() {
    cin >> n >> m;
    forn(i, n - 1) {
        int a, b;
        cin >> a >> b;
        g[a].pb(b);
        g[b].pb(a);
    }
    dfs(1, 1);
    BIT<int> ft(timer + 1);
    forn(i, m) {
        int a, b;
        cin >> a >> b;
        upd_path(a, b, 1, ft);
    }

    forab(i, 1, n + 1) {
        cout << get_val_node(i, ft) << " ";
    }
    cout << ln;
    return 0;
}
```

1.8 Prefix sum 2D

```
const int MAXN = 1005;
int a[MAXN][MAXN];
int pref[MAXN][MAXN];

int main() {
    int n, m;
    cin >> n >> m;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            pref[i][j] =
                pref[i - 1][j] + pref[i][j - 1] - pref[i - 1][j - 1] +
                    a[i][j];
        }
    }
    int q;
    cin >> q;
    while (q--) {
        int x1, y1, x2, y2;
        cin >> x1 >> y1 >> x2 >> y2;
        int ans = 0;
        // for (int i = x1; i <= x2; i++) {
        //   for (int j = y1; j <= y2; j++) {
        //     ans += a[i][j];
        //   }
        // }
        ans = pref[x2][y2] - pref[x1 - 1][y2] - pref[x2][y1 - 1] +
              pref[x1 - 1][y1 - 1];
        cout << ans << ln;
    }
    return 0;
}
```

1.9 SegmentTree 2D

```
template<typename T>
struct STree {
   int n, m;
   T neutro = T(0);
   vector<vector<T>> st;

   STree(vector<vector<T>> &a) {
      n = sz(a);
      m = sz(a[0]);
```

```
    st = vector<vector<T>>(2 * n, vector<T>(2 * m, neutro));
    build(a);
  }

  inline T oper(T a, T b) {
    return a + b;
  }

  void build(vector<vector<T>> &a) {
    forn (i, n) forn (j, m)
    st[i + n][j + m] = a[i][j];

    forn (i, n) rform (j, m - 1, 1, 1)
    st[i + n][j] = oper(st[i + n][j << 1], st[i + n][j << 1 | 1]);

    rform (i, n - 1, 1, 1) forn (j, 2 * m)
    st[i][j] = oper(st[i << 1][j], st[i << 1 | 1][j]);
  }
  //esquina izq arriba y esquina der abajo [row, col]
  T qry(int x1, int y1, int x2, int y2) { // [x1, y1] (x2, y2)
    T ans = neutro;
    for(int i0 = x1 + n, i1 = x2 + n; i0 < i1; i0 >>= 1, i1 >>= 1) {
      int t[4], q = 0;
      if (i0 & 1) t[q++] = i0++;
      if (i1 & 1) t[q++] = --i1;
      forn (k, q)
      for (int j0 = y1 + m, j1 = y2 + m; j0 < j1; j0 >>= 1, j1 >>= 1) {
        if(j0 & 1) ans = oper(ans,st[t[k]][j0++]);
        if(j1 & 1) ans = oper(ans,st[t[k]][--j1]);
      }
    }
    return ans;
  }

  void upd(int l, int r, T val) {
    st[l + n][r + m] = val;
    for (int j = r + m; j > 1; j >>= 1)
      st[l + n][j >> 1] = oper(st[l + n][j], st[l + n][j ^ 1]);

    for (int i = l + n; i > 1; i >>= 1)
      for (int j = r + m; j; j >>= 1)
        st[i >> 1][j] = oper(st[i][j], st[i ^ 1][j]);
  }
};
```

1.10 SegmentTree Iterativo

```
// [1, n]
template<typename T>
struct STree {
    vector<T> tree;
    int n;
    T neutro = T(0);

    T oper(T a, T b) {
        return a + b;
    }

    STree(vector<T> &a) {
        int tam = 1;
        while(1LL << (tam) < sz(a)) tam++;
        n = 1LL << tam;
        tree.resize(n * 2);
        forab(i, 1, sz(a) + 1) upd(i, a[i - 1]);
    }

    void upd(int i, T x) {
        i += n;
        tree[i] = x;
        while (i > 1) {
            i /= 2;
            // The index of the left child is s * 2.
            // The index of the right child is s * 2 + 1.
            tree[i] = oper(tree[i * 2], tree[i * 2 + 1]);
        }
    }

    T qry(int l, int r) {
        T res = 0;
        l += n;
        r += n;
        while (l <= r) {
            if (l % 2 == 1) res = oper(res, tree[l++]);
            if (r % 2 == 0) res = oper(res, tree[r--]);
            l /= 2;
            r /= 2;
        }
        return res;
```

```
    }

    //lower bound in the prefix sum
    int lb(T k) {
        int s = 1;
        while (s < n) {
            if (tree[s * 2] >= k) {
                s = s * 2;
            } else {
                k -= tree[s * 2];
                s = s * 2 + 1;
            }
        }

        return s - n;
    }
};
```

1.11 SegmentTree Lazy Iterativo

```
// tomado de https://github.com/brunomaletta

const int LOG = 17; //calcular
const int MAXN = 2e5;

//[0, n - 1]
template<typename T>
struct STree {
    T seg[2 * MAXN], lazy[2 * MAXN];

    int n;

    T oper(T a, T b) {
        return a + b;
    }

    // suma x en la posicin p de tamao tam
    void poe(int p, T x, int tam, bool prop=1) {
        seg[p] += x * tam;
        if (prop and p < n) lazy[p] += x;
        //######
    }

    // actualiza todos los padres de la hoja p
    void sobe(int p) {
        for (int tam = 2; p /= 2; tam *= 2) {
            seg[p] = oper(seg[2 * p], seg[2 * p + 1]);
            poe(p, lazy[p], tam, 0);
        }
    }

    // propaga la ruta desde la raz hasta la hoja p
    void prop(int p) {
        int tam = 1 << (LOG - 1);
        for (int s = LOG; s; s--, tam /= 2) {
            int i = p >> s;
            if (lazy[i]) {
                poe(2 * i, lazy[i], tam);
                poe(2 * i + 1, lazy[i], tam);
                lazy[i] = 0;//####
            }
        }
    }

    void build(int n2, vector<T> &v) {
        n = n2;
        for (int i = 0; i < n; i++) seg[ n + i] = v[i];
        for (int i = n - 1; i; i--) seg[i] = oper(seg[2 * i], seg[2 * i +
            1]);
        for (int i = 0; i < 2 * n; i++) lazy[i] = 0; //##########
    }

    T query(int a, int b) {
        T ret = 0; //#######
        for (prop(a += n), prop(b += n); a <= b; ++a /= 2, --b /= 2) {
            if (a % 2 == 1) ret = oper(ret, seg[a]);
            if (b % 2 == 0) ret = oper(ret, seg[b]);
        }
        return ret;
    }

    void upd(int a, int b, T x) {
        int a2 = a += n, b2 = b += n, tam = 1;
        for (; a <= b; ++a /= 2, --b /= 2, tam *= 2) {
            if (a % 2 == 1) poe(a, x, tam);
            if (b % 2 == 0) poe(b, x, tam);
        }
```

```
        sobe(a2), sobe(b2);
    }
};


/*

    STree<int> st;
    vector<int> a = {1, 2, 3, 1, 1, 1, 1, 1};
    st.build(sz(a), a);
*/
```

1.12 SegmentTree Lazy

```
template<typename T>
struct STree {
    #define L(v) v << 1
    #define R(v) v << 1 | 1
    //L(v) = 2 * v
    //R(v) = 2 * v + 1

    int n;
    vector<T> st, lazy;
    T neutro = T(0);

    STree(int m) {
        n = m;
        st.resize(n * 4);
        lazy.resize(n * 4);
    }

    STree(vector<T> &a) {
        n = sz(a);
        st.resize(n * 4);
        lazy.resize(n * 4);
        build(1, 0, n - 1, a);
    }

    T oper(T a, T b) {
        return a + b;
    }

    void build(int v, int tl, int tr, vector<T> &a) {
        if(tl == tr) {
            st[v] = a[tl];
            return;
        }
        int tm = (tl + tr) / 2;
        build(L(v), tl, tm, a);
        build(R(v), tm + 1, tr, a);
        st[v] = oper(st[L(v)], st[R(v)]);
    }

    void push(int v, int tl, int tr) {
        if (!lazy[v]) return;
        st[v] += (tr - tl + 1) * lazy[v];
        if (tl != tr) {
            lazy[L(v)] += lazy[v];
            lazy[R(v)] += lazy[v];
        }
        lazy[v] = 0;
    }

    void upd(int v, int tl, int tr, int l, int r, T val) {
        push(v, tl, tr);
        if(tr < l || tl > r) return;
        if (tl >= l && tr <= r) {
            lazy[v] = val;
            push(v, tl, tr);
            return;
        }
        int tm = (tl + tr) / 2;
        upd(L(v), tl, tm, l, r, val);
        upd(R(v), tm + 1, tr, l, r, val);
        st[v] = oper(st[L(v)], st[R(v)]);
    }

    T query(int v, int tl, int tr, int l, int r) {
        push(v, tl, tr);
        if(tl > r || tr < l) return neutro;
        if (l <= tl && tr <= r) return st[v];
        int tm = (tl + tr) / 2;
        return oper(query(L(v), tl, tm, l, r), query(R(v), tm + 1, tr, l,
            r));
    }

    void upd(int l, int r, T val) {
        upd(1, 0, n - 1, l, r, val);
```

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

1.13 SegmentTree Regular Bracket

```
struct node {
    int start, end, maxLen;
    /*{start, end}*/
};
struct STregularBracket {
    vector<node> seg;
    int size;

    STregularBracket(string& ch) {
        size = ch.length();
        seg.resize(4 * size);
        build(1, 1, size - 1, ch);
    }

    void build(int idx, int s, int e, string& ch) {
        if (s == e) {
            if (ch[s] == '(') {
                seg[idx] = { 1, 0 };
            } else {
                seg[idx] = { 0, 1 };
            }
            return;
        }
        build(idx << 1, s, (s + e) / 2, ch);
        build(idx << 1 | 1, (s + e) / 2 + 1, e, ch);
        seg[idx] = { seg[idx << 1 | 1].start, seg[idx << 1].end };
        int dif = seg[idx << 1].start - seg[idx << 1 | 1].end;
        int mini = min(seg[idx << 1].start, seg[idx << 1 | 1].end);
        seg[idx].maxLen += mini * 2 + seg[idx << 1 | 1].maxLen + seg[idx
            << 1].maxLen;
        if (dif > 0) {
            seg[idx].start += dif;
        } else {
            seg[idx].end -= dif;
        }
    }

    node query(int idx, int s, int e, int l, int r) {
        if (l > e || s > r) {
            return { 0, 0 };
        }
        if (s >= l && e <= r) {
            return seg[idx];
        }
        node p1 = query(idx << 1, s, (s + e) / 2, l, r);
        node p2 = query(idx << 1 | 1, (s + e) / 2 + 1, e, l, r);
        node ans = { p2.start, p1.end };
        int dif = p1.start - p2.end;
        ans.maxLen += p1.maxLen + p2.maxLen;
        ans.maxLen += min(p1.start, p2.end) * 2;
        if (dif > 0) {
            ans.start += dif;
        } else {
            ans.end -= dif;
        }
        return ans;
    }
    // [1, n]
    node query(int l, int r) {
        return query(1, 1, size - 1, l, r);
    }
};
// https://codeforces.com/contest/380/problem/C

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

    string w;
    cin >> w;
    w = "0" + w; // se agrega un comodin al inicio
    STregularBracket seg(w);
    int q, l, r;
    cin >> q;
    while (q--) {
        cin >> l >> r;
        cout << (seg.query(l, r).maxLen) << ln;
    }
    return 0;
```

```
}
```

1.14 SegmetTree

```
template <typename T>
struct STree {
    int n;
    vector<T> st;
    T neutro = T(0);

    STree(vector<T>& a) {
        n = sz(a);
        st.resize(n * 4);
        build(1, 0, n - 1, a);
    }

    T oper(T a, T b) {
        return max(a, b);
    }

    void build(int v, int tl, int tr, vector<T>& a) {
        if (tl == tr) {
            st[v] = a[tl];
            return;
        }
        int tm = (tr + tl) / 2;
        build(v * 2, tl, tm, a);
        build(v * 2 + 1, tm + 1, tr, a);
        st[v] = oper(st[v * 2], st[v * 2 + 1]);
    }

    T query(int v, int tl, int tr, int l, int r) {
        if (tl > r || tr < l)
            return neutro;
        if (l <= tl && tr <= r)
            return st[v];
        int tm = (tl + tr) / 2;
        return oper(query(v * 2, tl, tm, l, r), query(v * 2 + 1, tm + 1,
            tr, l, r));
    }

    void upd(int v, int tl, int tr, int pos, T val) {
        if (tl == tr) {
            st[v] = val;
            return;
        }
        int tm = (tr + tl) / 2;
        if (pos <= tm)
            upd(v * 2, tl, tm, pos, val);
        else
            upd(v * 2 + 1, tm + 1, tr, pos, val);
        st[v] = oper(st[v * 2], st[v * 2 + 1]);
    }

    int count_qry(int v, int tl, int tr, int l, int r, T x) {
        if (tl > r || tr < l)
            return 0;
        if (l <= tl && tr <= r) {
            if (st[v] <= x) {
            /*
            >  (st[v] <= x, max(a,b))
            >= (st[v] < x, max(a,b))
            <  (st[v] >= x, min(a,b))
            <= (st[v] > x, min(a,b))
            */
                return 0;
            }
            if (tl == tr)
                return 1;
        }
        int tm = (tl + tr) / 2;
        return count_qry(v * 2, tl, tm, l, r, x) + count_qry(v * 2 + 1, tm
            + 1, tr, l, r, x);
    }
    int count_qry(int a, int b, T x) {
        return count_qry(1, 0, n - 1, a, b, x);
    }
    void upd(int pos, T val) {
        upd(1, 0, n - 1, pos, val);
    }
    T query(int l, int r) {
        return query(1, 0, n - 1, l, r);
    }
};
```

1.15 Sparse Table

```
struct STable {
    int n, K;
    vector<vi> st;

    STable(const vi &a) {
        n = sz(a);
        K = int(log2(n)) + 1;
        st.assign(n + 1, vi(K));
        forn (i, n) st[i][0] = a[i];
        forn (j, K - 1)
        for (int i = 0; i + (1 << (j + 1)) <= n; ++i)
            st[i][j + 1] = oper(st[i][j], st[i + (1 << j)][j]);
    }

    int oper(int a, int b) {
        return __gcd(a, b);
    }

    int query(int l, int r) {
        int k = 31 - __builtin_clz(r - l + 1);
        return oper(st[l][k], st[r - (1 << k) + 1][k]);
    }
};
```

1.16 Trie XOR

```
struct Node {
    Node* childs[2];
};

struct Trie {
    Node* root;

    Trie() {
        root = new Node();
    }

    void insert(int x) {
        Node* cur = root;
        int i = 32;
        while (i--) {
            int bit = (x >> i) & 1;
            if (cur->childs[bit] == NULL) {
                cur->childs[bit] = new Node();
            }
            cur = cur->childs[bit];
        }
    }

    // max xor entre un elemento y query
    int maxXorQuery(int query) {
        Node* cur = root;
        int ans = 0;
        int i = 32;
        while (i--) {
            int bit = (query >> i) & 1;
            if (cur->childs[1 - bit] != NULL) {
                ans = ans | (1 << i);
                cur = cur->childs[1 - bit];
            } else {
                cur = cur->childs[bit];
            }
        }
        return ans;
    }

    // maximo xor entre dos elementos en un arreglo
    int maxXor(vi &arr) {
        int n = sz(arr);
        int max_val = 0;
        insert(arr[0]);
        for (int i = 1; i < n; i++) {
            max_val = max(maxXorQuery(arr[i]), max_val);
            insert(arr[i]);
        }
        return max_val;
    }
};
```

1.17 Union Find

```
struct UnionFind {
    vi p, c_sz;
    vi maxi, mini;
```

```
    UnionFind(int n) {
        p = vi(n);
        c_sz = vi(n);
        maxi = vi(n);
        mini = vi(n);
        forn(i, n) {
            p[i] = i, c_sz[i] = 1;
            maxi[i] = i;
            mini[i] = i;
        }
    }
    // Path compression optimization
    int find(int a) {
        return p[a] = (p[a] == a) ? a : find(p[a]);
    }
    // Union by size
    void unite(int x, int y) {
        int a = find(x);
        int b = find(y);
        if (a == b)return;
        if(c_sz[a] < c_sz[b]) swap(a, b);
        p[b] = a;
        c_sz[a] += c_sz[b];
        mini[a] = min(mini[a], mini[b]);
        maxi[a] = max(maxi[a], maxi[b]);
    }

    bool isSame(int a, int b) {
        return find(a) == find(b);
    }
    int comp_sz(int u) {
        return c_sz[find(u)];
    }
};
```

1.18 dsu persistent

```
//Tomando de:
//https://github.com/ShahjalalShohag/code-library/blob/main/Data%20Structures/DSU%20Partially%20Persistent.cpp
const int MAXN = 3e5 + 9;

struct DSU {
    vector<vector<pair<int, int>>> par;
    int time = 0; // initial time
    DSU(int n) : par(n + 1, {{-1, 0}}) {}
    bool merge(int u, int v) {
        ++time;
        if ((u = root(u, time)) == (v = root(v, time)))
            return 0;
        if (par[u].back().first > par[v].back().first)
            swap(u, v);
        par[u].push_back({par[u].back().first + par[v].back().first,
            time});
        par[v].push_back({u, time}); // par[v] = u
        return 1;
    }
    bool same(int u, int v, int t) { return root(u, t) == root(v, t); }

    int root(int u, int t) { // root of u at time t
        if (par[u].back().first >= 0 && par[u].back().second <= t)
            return root(par[u].back().first, t);
        return u;
    }

    int size(int u, int t) { // size of the component of u at time t
        u = root(u, t);
        int l = 0, r = sz(par[u]) - 1, ans = 0;
        while (l <= r) {
            int mid = l + r >> 1;
            if (par[u][mid].second <= t)
                ans = mid, l = mid + 1;
            else
                r = mid - 1;
        }
        return -par[u][ans].first;
    }
};
int a[MAXN];

int main() {
    int n, m, q;
    cin >> n >> m >> q;
    DSU d(n);
    for(int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;
        d.merge(u, v);
```

```
        a[d.time] = i;
    }
    while(q--) {
        int u, v;
        cin >> u >> v;
        int ans = -1, l = 0, r = r = d.time;
        while(l <= r) {
            int mid = (l + r) >> 1;
            if(d.same(u, v, mid)) ans = a[mid], r = mid - 1;
            else l = mid + 1;
        }
        cout << ans << ln;
    }
    return 0;
}

/*
    N nodos disjuntos

    van dando aristas conectando nodos (u, v)

    me piden responder el momento mas temprano cuando (u, v)
    estan en la misma componente (existe un camino)

    se puede responder onnline
*/
```

1.19 ordered$_set$

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

//template when all elements are distinct
template <class T> using ordered_set = tree<T, null_type,
less<T>, rb_tree_tag,tree_order_statistics_node_update>;
//template when duplicate elements are also used
template <class T> using ordered_multiset = tree<T, null_type,
less_equal<T>, rb_tree_tag,tree_order_statistics_node_update>;

//Find # of elements smaller than X in the set
//st.order_of_key(x); // O(log n)
//Element present at the k-th index in the set
//(*st.find_by_order(k)); //// O(log n)

int main() {
  ordered_multiset<int> st;
  int arr [] = {1, 10, 2, 7, 3};
  forn(i, 5) st.insert(arr[i]);
  // st = {1 2 3 7 10}
  forn(i, 5){
    int x = (*st.find_by_order(i));
    cout << (st.order_of_key(arr[i])) << ln;
  }
  return 0;
}
```

1.20 sqrt decomposition

```
//minimum range queries with updates

const int oo = 1e9 + 50;
const int MAXN = 2e5 + 5;

const int S = sqrt(MAXN);//block size
const int B = MAXN / S + 1; //# de bloques

int a[MAXN];
int blocks[B];

int get_idx(int i){
    //es el indice donde inicia el
    //bloque en el que cae i
    return (i / S) * S;
}

int main() {
    int n, q;
    cin >> n >> q;
    forn(i, n) cin >> a[i];

    fill(blocks, blocks + B, oo);
```

```
    forn(i, n){
        blocks[i / S] = min(blocks[i / S], a[i]);
    }

    while(q--){
        int o;
        cin >> o;
        if(o == 1){
            //modify
            int id, v;
            cin >> id >> v;
            id--;
            a[id] = v;
            blocks[id / S] = oo;

            int l = get_idx(id);
            int r = min(n, l + S);

            forab(i, l, r){
                blocks[i / S] = min(blocks[i / S], a[i]);
            }

        }else{
            int l, r;
            cin >> l >> r;
            l--, r--;
            int mn = oo;
            int i = l;
            while(i <= r){
                if(i % S == 0 && i + S < r){
                    mn = min(mn, blocks[i / S]);
                    i += S;
                }else{
                    mn = min(mn, a[i]);
                    i++;
                }
            }
            cout << mn << ln;
        }
    }
    return 0;
}
```

2 Dynamic Programming

2.1 Counting paths in a DAG

```
int main() {
    cin >> n >> m;
    int a, b;
    vi grado(n + 1, 0);
    forn(i, m) {
        cin >> a >> b;
        g[a].pb(b);
        grado[b]++;
    }
    vi topo = topo_sort(grado);
    int source = 1;
    int dest = n;
    //cuidado con overflow
    int dp[n] = { 0 };
    dp[dest] = 1;
    // traverse in reverse order
    rforn (i, sz(topo)) { // O(n + m)
        forn (j, sz(g[topo[i]])) {
            dp[topo[i]] += dp[g[topo[i]][j]];
        }
    }
    //number of paths from source
    //to dest
    cout << dp[source] << ln;
    return 0;
}
```

2.2 Counting tilings

```
//counting tilings
const int MOD = 1e9 + 7;
int dp[1001][1<<10];
int n, m;

void fill_colum(int column, int idx, int cur_mask, int next_mask) {
   if (idx == n) { // he llenado toda la columna
```

```
      dp[column + 1][next_mask] = (dp[column + 1][next_mask] +
          dp[column][cur_mask]) % MOD;
      return;
   }
   if ((cur_mask) & (1 << idx)) { // si el tile actual esta full
      fill_colum(column, idx + 1, cur_mask, next_mask);
   } else {
      // horizontal
      fill_colum(column, idx + 1, cur_mask, next_mask | (1 << idx));
      // vertical
      if (idx + 1 < n && (!(cur_mask & (1 << (idx + 1))))) {
         fill_colum(column, idx + 2, cur_mask, next_mask);
      }
   }
}

int main() {
   ios_base::sync_with_stdio(0);
   cin.tie(0);
   cout.tie(0);
   cin >> n >> m;
   dp[0][0] = 1;
   forn(column, m) { // todas las coumnas
      forn(mask, (1 << n)) { // todas las posibles mask
         if (dp[column][mask] > 0) {
            // hay forma de llenar la i-esima columna
            fill_colum(column, 0, mask, 0);
         }
      }
   }
   cout << dp[m][0] << ln;

   return 0;
}
```

2.3 Generating sums

```
const int MAX_N = 101;
const int MAX_SUM = 1e5 + 1;
bool dp[MAX_N + 1][MAX_SUM + 1];

int main() {
   //dada una cantidad de monedas, decir todas
   //sumas que se pueden formar con ellas
   int n;
   cin >> n;
   vi coins(n);
   for(auto &i: coins) cin >> i;
   dp[0][0] = true;
   forab (i, 1, n + 1) {
      forab (curSum, 0, MAX_SUM + 1) {
         dp[i][curSum] = dp[i - 1][curSum];
         int prevSum = curSum - coins[i - 1];
         if (prevSum >= 0 && dp[i - 1][prevSum]) {
            dp[i][curSum] = true;
         }
      }
   }
   vi ans;
   forab (sum, 1, MAX_SUM + 1) {
      if (dp[n][sum])ans.pb(sum);
   }
   cout << sz(ans) << ln;
   for (int sum : ans)
      cout << sum << " ";
   cout << ln;
   return 0;
}
```

2.4 LIS logn

```
//longest increasing subsecuence
//O(n log n)
const ll oo = 1e18;
int main() {
   int n;
   cin >> n;
   vll a(n);
   for(auto &i: a) cin >> i;
   vll dp(n + 1, oo);
   dp[0] = -oo;
   forab(i, 0, n) {
      int l = upper_bound(all(dp), a[i]) - dp.begin();
      if(dp[l - 1] < a[i] && a[i] < dp[l]) {
```

```
            dp[l] = a[i];
        }
    }
    ll ans = 0;
    forab(l, 0, n + 1) {
        if(dp[l] < oo) ans = l;
    }
    cout << ans << ln;
    return 0;
}
```

2.5 Traveling Salesman Problem

```
//shortest route
//it is a closed cycle where it
//ends at the same point it starts

// it visits each node exactly once
const int oo = 1e9;
int n, m;
const int N = 16;
vector<vi> g(N, vi(N, oo)); // directed graph
int dp[1 << N][N];

int go(int mask, int u) {
    if(mask == ((1 << n) - 1)) {
        return g[u][0] != oo ? g[u][0] : oo;
    }
    if(dp[mask][u] != - 1) return dp[mask][u];

    int ans = oo;
    forn(v, n) {
        if((mask & (1 << v)) == 0) {
            int aux = go(mask | (1 << v), v) + g[u][v];
            ans = min(ans, aux);
        }
    }
    return dp[mask][u] = ans;
}

int main() {
    cin >> n >> m;
    int a, b, w;
    forn(i, m) {
        cin >> a >> b >> w;
        g[a][b] = w;
    }
    memset(dp, -1, sizeof dp);
    int ans = go(1, 0); // mask, node
    cout << (ans == oo ? -1: ans) << ln;
    return 0;
}
```

2.6 convex hull trick

```
struct line {
    long long m, b;
    line (long long a, long long c) : m(a), b(c) {}
    long long eval(long long x) {
        return m * x + b;
    }
};

long double inter(line a, line b) {
    long double den = a.m - b.m;
    long double num = b.b - a.b;
    return num / den;
}

/**
 *  min m_i * x_j + b_i, for all i.
 *     x_j <= x_{j + 1}
 *     m_i >= m_{j + 1}
 * */
struct ordered_cht {
    vector<line> ch;
    int idx; // id of last "best" in query
    ordered_cht() {
        idx = 0;
    }

    void insert_line(long long m, long long b) {
        line cur(m, b);
        // new line's slope is less than all the previous
```

```
        while (ch.size() > 1 &&
               (inter(cur, ch[ch.size() - 2]) >= inter(cur, ch[ch.size()
                   - 1]))) {
            // f(x) is better in interval [inter(ch.back(), cur), inf)
            ch.pop_back();
        }

        ch.push_back(cur);
    }

    long long eval(long long x) { // minimum
        // current x is greater than all the previous x,
        // if that is not the case we can make binary search.
        idx = min<int>(idx, ch.size() - 1);
        while (idx + 1 < (int)ch.size() && ch[idx + 1].eval(x) <=
            ch[idx].eval(x))
            idx++;

        return ch[idx].eval(x);
    }
};


/**
 *  Dynammic convex hull trick
 * */

typedef long long int64;
typedef long double float128;

const int64 is_query = -(1LL<<62), inf = 1e18;

struct Line {
    int64 m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if (!s) return 0;
        int64 x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
```

```
struct HullDynamic : public multiset<Line> { // will maintain upper hull
    for maximum
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            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 (float128)(x->b - y->b)*(z->m - y->m) >= (float128)(y->b -
            z->b)*(y->m - x->m);
    }
    void insert_line(int64 m, int64 b) {
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        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));
    }

    int64 eval(int64 x) {
        auto l = *lower_bound((Line) {
            x, is_query
        });
        return l.m * x + l.b;
    }
};

/**
 *  Problems:
 *    http://codeforces.com/problemset/problem/319/C
 *    http://codeforces.com/contest/311/problem/B
 *    https://csacademy.com/contest/archive/task/squared-ends
 *    http://codeforces.com/contest/932/problem/F
 * */
```

2.7 digit dp

```cpp
//digit dp
//[0, a]

//What if we want [a, b]
//solve(b) - solve(a - 1)

// dado un rango contar cuando numeros hay que tengan
// la siguiente forma 22338899
// es decir, el digito en la posicion 2n-1 == 2n

const ll MOD = 1e4 + 7;

int n;
const int D = 1e5 + 5; //digitos del numero
ll dp[D][10][2][2];
short num[D];

ll add(ll a, ll b){
    return ((a % MOD) + (b % MOD)) % MOD;
}
ll sub(ll a, ll b){
    return ((a % MOD) - (b % MOD) + MOD) % MOD;
}

//prev = digito usado previamente
// eq = para saber el max_digit que puedo usar
// start = es true cuando he usado un digito diferente a cero
// (0000022) es valido porque representa el 22

ll go(int idx, int prev, bool eq, bool start){
    if(idx == n){
        return start;
    }
    ll &ans = dp[idx][prev][eq][start];
    if(ans != -1) return ans;

    ans = 0;
    int max_digit = (eq ? num[idx] : 9);

    if(idx % 2 == 0){
        //si es par puedo iniciar con cualquier digito
        for(int d = 0; d <= max_digit; d++){
            //si tengo ceros a la izq y me quedan 2 para colocar
            //solo puedo poner uno diferente a cero
            if(!start && d == 0 && idx == n - 2) continue;

            bool new_eq = eq && d == num[idx];
            int new_start = start || (d != 0);

            if(!new_start){
                ans = add(ans, go(idx + 1, 0, new_eq, 0));
            }else{
                ans = add(ans, go(idx + 1, d, new_eq, 1));
            }
        }
    }else{
        // si es impar, tengo que poner el mismo usado anteriormente
        if(start){
            if(prev <= max_digit){
                ans = add(ans, go(idx + 1, prev, eq && prev == max_digit,
                    1));
            }
        }else{
            //si no he iniciado pero puedo continuar con 0s
            if(0 <= max_digit){
                ans = add(ans, go(idx + 1, 0, eq && 0 == max_digit, 0));
            }
        }
    }
    return ans;
}

string subtract_one(const string& s) {
    string res = s;
    int n = sz(res);
    int i = n - 1;

    while (i >= 0 && res[i] == '0') {
        res[i] = '9';
        i--;
    }

    if (i >= 0) {
        res[i]--;
    }

    int start = 0;
    while (start < n - 1 && res[start] == '0') {
        start++;
```

```
    }
    return res.substr(start);
}


ll solve(string &s){
    n = sz(s);
    if(sz(s) % 2 == 1){
        //en este problema la cadena
        //siempre debe ser par
        s = "0" + s;
        n = sz(s);
    }

    forn(i, n) num[i] = short(s[i] - '0');
    memset(dp, -1, sizeof dp);
    ll ans = go(0, 0, 1, 0);
    return ans;
}
int main() {
    string a, b;
    cin >> a >> b;

    string ax = subtract_one(a);
    ll aa = solve(ax);
    ll bb = solve(b);

    cout << sub(bb, aa) << ln;
    return 0;
}
```

2.8 digit permutation

```
//we have two integers N and M,
//we need to find how many numbers
// obtained by rearranging
//digits of N are divisible by M.

int getdigit(char c) {
  return int(c - '0');
}

string s;
ll m;
ll dp[101][(1 << 18) + 1];
ll base = 10;

// go(0, 0);
ll go(ll rem, int mask) {
  if(rem == 0LL && mask == ((1 << sz(s)) - 1)) {
    return 1;
  }
  if(dp[rem][mask] != -1) return dp[rem][mask];
  ll ans = 0;
  forn(i, sz(s)) {
    if(!(mask & (1 << i))) {
      int digit = getdigit(s[i]);
      //leading zeros
      if(mask == 0 && digit == 0) continue;
      ans += go(((rem * base) + digit) % m, mask | (1 << i));
    }
  }
  return dp[rem][mask] = ans;
}

int main() {
  cin >> s >> m;
  vi cnt(10, 0);
  ll repetidos = 1;
  for(char c: s) {
    repetidos *= ++cnt[c - '0'];
  }
  vector<vll> dp(101, vll(1 << 18, 0));
  dp[0][0] = 1LL;
  forn(mask, (1 << sz(s))) {
    forn(i, sz(s)) {
      if(!(mask & (1 << i))) {
        int digit = getdigit(s[i]);
        if(mask == 0 && digit == 0) continue;
        forn(rem, m) {
          dp[((1LL * rem * base) + digit) % m][mask | (1 << i)] +=
              dp[rem][mask];
        }
      }
    }
  }
  ll ans = dp[0][(1 << sz(s))- 1];
  cout << (ans / repetidos) << ln;
```

```
  return 0;
}
```

2.9 kadane

```
//max subarray sum
ll kadane(vll &a, int l, int r){
    ll mx = 0LL;
    ll cur = 0LL;
    for(int i = l; i < r; i++){
        cur = max(cur + a[i], a[i]);
        mx = max(mx, cur);
    }
    return mx;
}
```

3 Flows

3.1 Dinic

```
typedef pair<int, int> ii;

struct FlowEdge {
    int v, u;
    ll cap, flow = 0;
    FlowEdge(int _v, int _u, ll _cap) : v(_v), u(_u), cap(_cap) {}
};

struct Dinic { // O(V^2 * E)
    const ll flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> g;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int _n, int _s, int _t) : n(_n), s(_s), t(_t) {
        g.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

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

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

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

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

    vector<ii> min_cut() {
        //llamar flow();
        vector<ii> cut;
        for (auto &e : edges)
            if (level[e.v] != -1 && level[e.u] == -1 && e.cap > 0)
                cut.pb({e.v, e.u});
        return cut;
    }
};
//Dinic dd(n + 2, s, t);

//int nodos = n + 5;
//Dinic dd(nodos, nodos - 2, nodos - 1);


/* Max Independent Paths (vertex-disjoint)
  Two paths from S to T are independent if they
  do not share any vertex apart from S and T.

  Construc the network flow, each edge with 1 capacity,
  and each node with 1 capacity (each node can only be used once),
  then run a max flow algo.
*/

/*
  Max Edge-disjoint Paths
  Two paths from S to T are edge-disjoint
  if they do not share any edge
  (but they can share nodes).
  this is equivalent to find the (vertex-disjoint)
  but we do not have vertex capacity
*/

// Min Vertex Cover: vertices de L con level[v]==-1 y vertices de R con
    level[v]>0
// Max Independent Set: vertices NO tomados por el Min Vertex Cover
```

3.2 Hopcroft Karp

```
struct mbm {// O(E * sqrt(V))
    int n, m; //left and right sz
    vi mat, inv, d;
    vector<vi> g;
    vb cover[2];

    mbm(int nl, int nr) : n(nl), m(nr) {
        mat = vi(n, -1);
        inv = vi(m, -1);

        d = vi(n);
        g = vector<vi>(n);

        //mvc
        cover[0].assign(n, true);
        cover[1].assign(m, false);
    }
    //u->[0, n) v->[0, m)
    void add(int u, int v) {
        g[u].pb(v);
    }

    bool bfs() {
        bool aug = false;
        queue<int> q;
        forab(u, 0,n) if (mat[u] < 0) q.push(u);
        else d[u] = -1;
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (auto v : g[u]) {
                if (inv[v] < 0) aug = true;
```

```
            else if (d[inv[v]] < 0) d[inv[v]] = d[u] + 1,
                q.push(inv[v]);
        }
    }
    return aug;
}

bool dfs(int u) {
    for (auto v : g[u]) if (inv[v] < 0) {
            mat[u] = v, inv[v] = u;
            return true;
        }
    for (auto v : g[u]) if (d[inv[v]] > d[u] && dfs(inv[v])) {
            mat[u] = v, inv[v] = u;
            return true;
        }
    d[u] = 0;
    return false;
}

int go() {
    int ans = 0;
    while (bfs()) forab(u,0,n) if (mat[u] < 0) ans += dfs(u);
    return ans;
}
void dfs2(int u) {
    cover[0][u] = false;
    for (auto v : g[u]) if (!cover[1][v]) {
            cover[1][v] = true;
            dfs2(inv[v]);
        }
}
pair<vi, vi> mvc() {
    int mf = go();
    forab(u, 0,n) if (mat[u] < 0) dfs2(u);
    vi L, R;
    forab(i, 0, n) if (cover[0][i]) L.pb(i);
    forab(i, 0, m) if (cover[1][i]) R.pb(i);
    assert(mf == sz(L) + sz(R));
    return {L, R};
}

};

int main() {
    //get matching
    forab(u, 0, n) {
        if(dd.mat[u] != -1) {
            cout << u + 1 << " " << dd.mat[u] + 1 << ln;
        }
    }
    return 0;
}

/*Maximum Independent Set:
Subset of nodes
with the following propriety:
For any two nodes u, v
are not adjacent
(There is no direct edge between nodes u and v).
*/

/*Minimum Vertex Cover:
A vertex cover is a subset of the nodes
that together touch all the edges.
*/

//Max Independet Set = |V| - size of the MCBM

/*Min Path Cover (only in DAG)
 Find the minimum number of paths to cover
 each vertex on a graph
 */
```

3.3 Min$_c ost_m ax_f low$

```
#define INF 0x3f3f3f

template<typename T> struct mcmf {
    struct edge {
        int to, rev, flow, cap; // para, id da reversa, fluxo, capacidade
        bool res; // se eh reversa
        T cost; // custo da unidade de fluxo
        edge() : to(0), rev(0), flow(0), cap(0), cost(0), res(false) {}
        edge(int to_, int rev_, int flow_, int cap_, T cost_, bool res_)
            : to(to_), rev(rev_), flow(flow_), cap(cap_), res(res_),
                cost(cost_) {}
    };
```

```
vector<vector<edge>> g;
vi par_idx, par;
T inf;
vector<T> dist;

mcmf(int n) : g(n), par_idx(n), par(n),
    inf(numeric_limits<T>::max()/3) {}

void add(int u, int v, int w, T cost) { // de u pra v com cap w e
    custo cost
    edge a = edge(v, sz(g[v]), 0, w, cost, false);
    edge b = edge(u, sz(g[u]), 0, 0, -cost, true);

    g[u].pb(a);
    g[v].pb(b);
}

vector<T> spfa(int s) { // nao precisa se nao tiver custo negativo
    deque<int> q;
    vb is_inside(sz(g), 0);
    dist = vector<T>(sz(g), inf);

    dist[s] = 0;
    q.pb(s);
    is_inside[s] = true;

    while (!q.empty()) {
        int v = q.front();
        q.pop_front();
        is_inside[v] = false;

        for (int i = 0; i < sz(g[v]); i++) {
            auto [to, rev, flow, cap, res, cost] = g[v][i];
            if (flow < cap and dist[v] + cost < dist[to]) {
                dist[to] = dist[v] + cost;

                if (is_inside[to]) continue;
                if (!q.empty() and dist[to] > dist[q.front()])
                    q.push_back(to);
                else q.push_front(to);
                is_inside[to] = true;
            }
        }
    }
    return dist;
}
bool dijkstra(int s, int t, vector<T>& pot) {
    priority_queue<pair<T, int>, vector<pair<T, int>>, greater<>> q;
    dist = vector<T>(sz(g), inf);
    dist[s] = 0;
    q.emplace(0, s);
    while (sz(q)) {
        auto [d, v] = q.top();
        q.pop();
        if (dist[v] < d) continue;
        for (int i = 0; i < sz(g[v]); i++) {
            auto [to, rev, flow, cap, res, cost] = g[v][i];
            cost += pot[v] - pot[to];
            if (flow < cap and dist[v] + cost < dist[to]) {
                dist[to] = dist[v] + cost;
                q.emplace(dist[to], to);
                par_idx[to] = i, par[to] = v;
            }
        }
    }
    return dist[t] < inf;
}

pair<int, T> min_cost_flow(int s, int t, int flow = INF) {
    vector<T> pot(sz(g), 0);
    pot = spfa(s); // mudar algoritmo de caminho minimo aqui

    int f = 0;
    T ret = 0;
    while (f < flow and dijkstra(s, t, pot)) {
        for (int i = 0; i < sz(g); i++)
            if (dist[i] < inf) pot[i] += dist[i];

        int mn_flow = flow - f, u = t;
        while (u != s) {
            mn_flow = min(mn_flow,
                        g[par[u]][par_idx[u]].cap -
                            g[par[u]][par_idx[u]].flow);
            u = par[u];
        }

        ret += pot[t] * mn_flow;

        u = t;
```

```
            while (u != s) {
                g[par[u]][par_idx[u]].flow += mn_flow;
                g[u][g[par[u]][par_idx[u]].rev].flow -= mn_flow;
                u = par[u];
            }

            f += mn_flow;
        }

        return make_pair(f, ret);
    }

    // Opcional: retorna as arestas originais por onde passa flow = cap
    vector<pair<int,int>> recover() {
        vector<pair<int,int>> used;
        for (int i = 0; i < sz(g); i++) for (edge e : g[i])
                if(e.flow == e.cap && !e.res) used.push_back({i, e.to});
        return used;
    }
};

//nodos = (n + m) + 10;
// s = n - 1;
// t = n - 2;
```

4 Geometry

4.1 Circle

```
//Require struct Point and functions (dot, cross, dist, perp, etc.)

// Vector perpendicular (rotado 90 CCW)
Point perp(const Point& a) {
    return Point(-a.y, a.x);
}

//Estructura de lnea ( til  para intersecciones)
struct Line {
    Point p;  // punto en la lnea
    Point v;  // vector director

    Line() {}
    Line(Point _p, Point _v) : p(_p), v(_v) {}

    // Proyeccin de un punto sobre la lnea
    Point proj(const Point& a) const {
        return p + v * ( (a - p).dot(v) / v.norm2() );
    }

    // Distancia cuadrada de un punto a la lnea
    double sqDist(const Point& a) const {
        return ((a - p).cross(v) * (a - p).cross(v)) / v.norm2();
    }
};

//Circuncentro de un tringulo (pasa por A, B, C)
Point circumCenter(Point a, Point b, Point c) {
    b = b - a, c = c - a; // coordenadas relativas a A
    double d = cross(b, c);
    assert(fabs(d) > EPS); // no existe circunferencia si son colineales
    Point res = a + perp(b * c.norm2() - c * b.norm2()) * (0.5 / d);
    return res;
}

//Signo generalizado
template <typename T>
int sgn(T x) { return (x > 0) - (x < 0); }

//Interseccin  crculo - lnea
// Devuelve:
//   0     no toca
//   1     tangente
//   2     interseccin en dos puntos
int circleLine(Point o, double r, Line l, pair<Point, Point>& out) {
    double h2 = r * r - l.sqDist(o);
    if (h2 < -EPS) return 0; // no intersecan

    Point p = l.proj(o);
    if (h2 < EPS) { // tangente
        out = {p, p};
        return 1;
    }
    Point h = l.v * (sqrt(h2) / l.v.norm());
    out = {p - h, p + h};
    return 2;
}
```

```cpp
// Interseccin entre dos crculos
// Devuelve 0, 1 o 2 dependiendo del nmero de intersecciones
int circleCircle(Point o1, double r1, Point o2, double r2, pair<Point,
    Point>& out) {
    Point d = o2 - o1;
    double dist2 = d.norm2();
    double distv = sqrt(dist2);

    if (distv < EPS) return 0; // concntricos  sin puntos definidos

    double pd = (dist2 + r1 * r1 - r2 * r2) / (2 * distv); // distancia
        al punto base
    double h2 = r1 * r1 - pd * pd;
    if (h2 < -EPS) return 0; // no se cruzan

    Point base = o1 + d * (pd / distv);
    if (h2 < EPS) { // tangentes
        out = {base, base};
        return 1;
    }

    Point h = perp(d) * (sqrt(h2) / distv);
    out = {base - h, base + h};
    return 2;
}

// Tangentes comunes entre dos crculos
// inner = false   tangentes externas
// inner = true    tangentes internas
// Devuelve 0, 1 o 2 pares de puntos (tangencias)
int tangents(Point o1, double r1, Point o2, double r2, bool inner,
    vector<pair<Point, Point>>& out) {
    if (inner) r2 = -r2;
    Point d = o2 - o1;
    double dr = r1 - r2;
    double d2 = d.norm2();
    if (d2 < EPS) return 0; // crculos concntricos    sin tangentes
    double h2 = d2 - dr * dr;
    if (h2 < -EPS) return 0; // sin tangentes reales

    for (double sign : {-1.0, 1.0}) {
        Point v = (d * dr + perp(d) * sqrt(max(0.0, h2)) * sign) / d2;
        out.push_back({o1 + v * r1, o2 + v * r2});
    }
    return (h2 > EPS ? 2 : 1);
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    // --- Circuncentro ---
    Point A(0,0), B(4,0), C(0,3);
    Point O = circumCenter(A, B, C);
    cout << "Circuncentro: (" << O.x << ", " << O.y << ")\n";

    // --- Crculo- Lnea  ---
    Line l(Point(0,0), Point(1,1)); // y = x
    pair<Point,Point> inter;
    int t1 = circleLine(Point(2,0), 2.0, l, inter);
    cout << "Interseccin  crculo - lnea : " << t1 << " puntos\n";
    if (t1) cout << "(" << inter.first.x << "," << inter.first.y << ") "
                 << "(" << inter.second.x << "," << inter.second.y <<
                     ")\n";

    // --- Crculo- Crculo  ---
    pair<Point,Point> cc;
    int t2 = circleCircle(Point(0,0), 3, Point(4,0), 3, cc);
    cout << "Interseccin  crculo - crculo : " << t2 << " puntos\n";
    if (t2) cout << "(" << cc.first.x << "," << cc.first.y << ") "
                 << "(" << cc.second.x << "," << cc.second.y << ")\n";

    // --- Tangentes ---
    vector<pair<Point,Point>> tangs;
    int t3 = tangents(Point(0,0), 2, Point(6,0), 2, false, tangs);
    cout << "Tangentes externas: " << t3 << "\n";
    for (auto& p : tangs)
        cout << "T1(" << p.first.x << "," << p.first.y << ") "
             << "T2(" << p.second.x << "," << p.second.y << ")\n";

    return 0;
}
```

4.2 Convex Hull

```cpp
#define sq(x) ((x)*(ll)(x))
```

```
struct pt { // punto
    ll x, y;
    pt(ll x_ = 0, ll y_ = 0) : x(x_), y(y_) {}
    bool operator < (const pt p) const {
        if (x != p.x) return x < p.x;
        return y < p.y;
    }
    bool operator == (const pt p) const {
        return x == p.x and y == p.y;
    }
    pt operator + (const pt p) const {
        return pt(x+p.x, y+p.y);
    }
    pt operator - (const pt p) const {
        return pt(x-p.x, y-p.y);
    }
    pt operator * (const ll c) const {
        return pt(x*c, y*c);
    }
    ll operator * (const pt p) const {
        return x*(ll)p.x + y*(ll)p.y;
    }
    ll operator ^ (const pt p) const {
        return x*(ll)p.y - y*(ll)p.x;
    }
    friend istream& operator >> (istream& in, pt& p) {
        return in >> p.x >> p.y;
    }
};


class ConvexHull {
public:
    int n;
    vector<pt> compute(vector<pt> &pts) {
        n = sz(pts);
        // sort points by x-axis, and by the y-axis when there's a tie.
        sort(pts.begin(), pts.end(), [](pt &a, pt &b) {
            if (b.x < a.x) return false;
            if (b.x == a.x) return a.y > b.y;
            return true;
        });
        auto low_p = lower_hull(pts);
        auto upp_p = upper_hull(pts);
        vector<pt> hull;
        for(auto i: low_p) hull.pb(i);
        for(auto i: upp_p) hull.pb(i);

        sort(all(hull));
        hull.erase(unique(all(hull)), hull.end());
        return hull;
    }

    vector<pt> lower_hull(vector<pt> &pts) {
        vector<pt> stk;
        for (int i = 0; i < n; i++) {
            while (sz(stk) > 1 && is_cwm(stk[sz(stk) - 2], stk.back(),
                pts[i]))
                stk.pop_back();
            stk.pb(pts[i]);
        }
        return stk;
    }
    vector<pt> upper_hull(vector<pt> &pts) {
        vector<pt> stk;
        for (int i = n - 1; i >= 0; i--) {
            while (sz(stk) > 1 && is_cwm(stk[sz(stk) - 2], stk.back(),
                pts[i]))
                stk.pop_back();
            stk.pb(pts[i]);
        }
        return stk;
    }
    //isClockWiseMovement
    bool is_cwm(pt &a, pt &b, pt &c) {
        return cross_p(a, b, c) > 0;
    }
    //cross product
    ll cross_p(pt &a, pt &b, pt &c) {
        return ((a.x - b.x) * (c.y - b.y)) - ((a.y - b.y) * (c.x - b.x));
    }
};

int main() {
    int n;
    cin >> n;
    vector<pt> pts(n);
    forn(i, n) {
```

```
        cin >> pts[i];
    }

    ConvexHull convexHull;
    vector<pt > ans = convexHull.compute(pts);

    cout << sz(ans) << ln;
    for (auto &p: ans)
        cout << p.x << " " << p.y << ln;

    return 0;
}

//https://cses.fi/problemset/task/2195/
```

4.3 Geomotry3D

```
// geometry3d.hpp
#pragma once
#include <bits/stdc++.h>
using namespace std;

typedef double ld;
const ld DINF = 1e18;
const ld EPS = 1e-9;
inline ld sq(ld x) { return x*x; }
inline bool eq(ld a, ld b) { return fabs(a - b) <= EPS; }

struct pt {
    ld x, y, z;
    pt(ld x_ = 0, ld y_ = 0, ld z_ = 0) : x(x_), y(y_), z(z_) {}
    bool operator < (const pt& p) const {
        if (!eq(x, p.x)) return x < p.x;
        if (!eq(y, p.y)) return y < p.y;
        if (!eq(z, p.z)) return z < p.z;
        return false;
    }
    bool operator == (const pt& p) const {
        return eq(x, p.x) && eq(y, p.y) && eq(z, p.z);
    }
    pt operator + (const pt& p) const { return pt(x + p.x, y + p.y, z +
        p.z); }
    pt operator - (const pt& p) const { return pt(x - p.x, y - p.y, z -
        p.z); }
    pt operator * (const ld c) const { return pt(x * c, y * c, z * c); }
    pt operator / (const ld c) const { return pt(x / c, y / c, z / c); }
    // producto escalar
    ld operator * (const pt& p) const { return x * p.x + y * p.y + z *
        p.z; }
    // producto cruz
    pt operator ^ (const pt& p) const {
        return pt(y * p.z - z * p.y,
                  z * p.x - x * p.z,
                  x * p.y - y * p.x);
    }
    ld norm2() const { return x*x + y*y + z*z; }
    ld norm() const { return sqrt(norm2()); }
    friend istream& operator >> (istream& in, pt& p) {
        return in >> p.x >> p.y >> p.z;
    }
    friend ostream& operator << (ostream& out, const pt& p) {
        out << "(" << p.x << "," << p.y << "," << p.z << ")";
        return out;
    }
};

inline ld DEG_TO_RAD(ld n) { return n * acos(-1.0) / 180.0; }
inline ld RAD_TO_DEG(ld n) { return n * 180.0 / acos(-1.0); }

struct line {
    pt p, q; // segment from p to q, or line through p in direction (q-p)
    line() {}
    line(pt p_, pt q_) : p(p_), q(q_) {}
    friend istream& operator >> (istream& in, line& r) { return in >> r.p
        >> r.q; }
};

struct plane {
    array<pt,3> p;  // tres puntos que definen el plano
    array<ld,4> eq; // ax + by + cz + d = 0
    plane() {}
    plane(pt a, pt b, pt c) : p({a,b,c}) { build(); }
    void build() {
        pt v1 = p[1] - p[0], v2 = p[2] - p[0];
        pt n = v1 ^ v2;
        // eq = {a, b, c, d} con d tal que a*x + b*y + c*z + d = 0
        eq[0] = n.x; eq[1] = n.y; eq[2] = n.z;
```

```
        eq[3] = -(n * p[0]);
    }
    friend istream& operator >> (istream& in, plane& P) {
        in >> P.p[0] >> P.p[1] >> P.p[2];
        P.build();
        return in;
    }
};

// convierte de coordenadas esfricas (rho, theta, phi) a cartesianas
// theta = rotacin en xy, phi = ngulo desde z (radianes)
inline pt convert(ld rho, ld theta, ld phi) {
    return pt(sin(phi) * cos(theta) * rho,
              sin(phi) * sin(theta) * rho,
              cos(phi) * rho);
}

// proyeccin de punto p en la recta r
inline pt proj(const pt& p, line r) {
    if (r.p == r.q) return r.p;
    pt dir = r.q - r.p;
    pt v = p - r.p;
    ld t = (v * dir) / dir.norm2();
    return r.p + dir * t;
}

// proyeccin de punto p en el plano P
inline pt proj(const pt& p, const plane& P) {
    pt v1 = P.p[1] - P.p[0], v2 = P.p[2] - P.p[0];
    pt n = v1 ^ v2;
    if (n.norm2() < EPS) return P.p[0]; // plano degenerado
    // proyectar: p' = p - n * ((n(p - p0)) / |n|^2)
    ld t = (n * (p - P.p[0])) / n.norm2();
    return p - n * t;
}

// distancia euclidiana
inline ld dist(const pt& a, const pt& b) {
    return (a - b).norm();
}

// distancia de un punto a una recta
inline ld distline(const pt& p, const line& r) {
    return dist(p, proj(p, r));
}
```

```
// distancia de punto a segmento
inline ld distseg(const pt& p, const line& r) {
    pt a = r.p, b = r.q;
    pt ab = b - a;
    pt ap = p - a;
    ld d1 = ap * ab;
    if (d1 <= 0) return dist(p, a);
    ld d2 = (p - b) * (a - b); // (p-b)(a-b)
    if (d2 <= 0) return dist(p, b);
    return distline(p, r);
}

// distancia con signo punto-plano: ax + by + cz + d
inline ld sdist(const pt& p, const plane& P) {
    return P.eq[0]*p.x + P.eq[1]*p.y + P.eq[2]*p.z + P.eq[3];
}
inline ld distplane(const pt& p, const plane& P) { return
    fabs(sdist(p,P)) / sqrt(sq(P.eq[0]) + sq(P.eq[1]) + sq(P.eq[2])); }

// si punto pertenece al segmento (tolerancia EPS)
inline bool isinseg(const pt& p, const line& r) {
    return eq(distseg(p, r), 0.0);
}

// --- PUNTO EN POLGONO 3D ---
// Proyecta polgono y punto al plano principal del polgono y aplica
    ray-casting 2D.
// Devuelve true si p est dentro o en el borde del polgono (polgono
    simple, no necesariamente convexo).
inline bool pointInPolygon3D(const pt& p, const vector<pt>& poly) {
    int n = poly.size();
    assert(n >= 3);
    // construir plano del polgono
    plane P(poly[0], poly[1], poly[2]);
    // proyectar punto y polgono al sistema de coordenadas 2D local
    // Elegimos dos ejes ortonormales u,v en el plano
    pt u = (poly[1] - poly[0]);
    if (u.norm() < EPS) return false;
    u = u / u.norm();
    pt nrm = (poly[1] - poly[0]) ^ (poly[2] - poly[0]);
    if (nrm.norm() < EPS) return false;
    nrm = nrm / nrm.norm();
    pt v = nrm ^ u; // v es ortogonal a u en el plano
```

```
    auto to2d = [&](const pt& q)->pair<ld,ld> {
        pt rel = q - poly[0];
        return { rel * u, rel * v }; // coordenadas (x,y)
    };

    pair<ld,ld> pp = to2d(proj(p, P)); // proyectamos p al plano del
        polgono
    vector<pair<ld,ld>> poly2(n);
    for (int i = 0; i < n; ++i) poly2[i] = to2d(poly[i]);

    // Punto en segmento (2D)
    auto onSeg2D = [&](pair<ld,ld> a, pair<ld,ld> b, pair<ld,ld> q)->bool
        {
        ld minx = min(a.first, b.first)-EPS, maxx = max(a.first,
            b.first)+EPS;
        ld miny = min(a.second, b.second)-EPS, maxy = max(a.second,
            b.second)+EPS;
        // area (a,b,q)
        ld cross = (b.first - a.first) * (q.second - a.second) - (b.second
            - a.second) * (q.first - a.first);
        return fabs(cross) <= EPS && q.first >= minx && q.first <= maxx &&
            q.second >= miny && q.second <= maxy;
    };

    // Ray casting 2D
    bool inside = false;
    for (int i = 0, j = n-1; i < n; j = i++) {
        auto a = poly2[i], b = poly2[j];
        if (onSeg2D(a,b,pp)) return true;
        bool intersect = ((a.second > pp.second) != (b.second > pp.second))
                    && (pp.first < (b.first - a.first) * (pp.second -
                        a.second) / (b.second - a.second) + a.first);
        if (intersect) inside = !inside;
    }
    return inside;
}

// distancia de punto a polgono (polgono simple en 3D)
// si la proyeccin del punto cae dentro del polgono, la distancia es la
    distancia perpendicular al plano,
// si no, la mnima distancia a aristas
inline ld distpol(const pt& p, const vector<pt>& poly) {
    int n = poly.size();
    assert(n >= 3);
    plane P(poly[0], poly[1], poly[2]);
    pt pproj = proj(p, P);
    if (pointInPolygon3D(p, poly)) return dist(p, pproj);

    ld best = DINF;
    for (int i = 0; i < n; ++i) {
        int j = (i+1)%n;
        best = min(best, distseg(p, line(poly[i], poly[j])));
    }
    return best;
}

// Interseccin segmento-plano
enum RETCODE { BOTH, ONE, PARAL, CONCOR };
inline pair<RETCODE, pt> intersect(const plane& P, const line& r) {
    ld d1 = sdist(r.p, P);
    ld d2 = sdist(r.q, P);
    if (eq(d1, 0) && eq(d2, 0)) return {BOTH, r.p};
    if (eq(d1, 0)) return {ONE, r.p};
    if (eq(d2, 0)) return {ONE, r.q};
    if ((d1 > 0 && d2 > 0) || (d1 < 0 && d2 < 0)) {
        if (eq(d1 - d2, 0)) return {PARAL, pt()};
        return {CONCOR, pt()};
    }
    ld frac = d1 / (d1 - d2);
    pt res = r.p + (r.q - r.p) * frac;
    return {ONE, res};
}

// Rotacin de p alrededor del eje definido por u (vector que pasa por
    origen) por ngulo a (radianes).
// Usamos frmula de Rodrigues:
// p_rot = p*cos(a) + (u x p)*sin(a) + u*(up)*(1 - cos(a)), con u
    normalizado.
inline pt rotate(const pt& p, pt u, ld a) {
    ld un = u.norm();
    if (un < EPS) return p; // eje invlido
    u = u / un;
    ld cosa = cos(a), sina = sin(a);
    pt term1 = p * cosa;
    pt term2 = (u ^ p) * sina;
    pt term3 = u * ((u * p) * (1 - cosa));
    return term1 + term2 + term3;
}
```

```
// ------------------- Ejemplo de uso rpido -------------------
#ifdef GEOMETRY3D_TEST_MAIN
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    // test bsicos
    pt A(0,0,0), B(1,0,0), C(0,1,0), P(0.2,0.2,1);
    plane pl(A,B,C);
    cout << "Distancia punto-plano (P): " << distplane(P, pl) << "\n";
    cout << "Proyeccin de P en plano: " << proj(P, pl) << "\n";

    vector<pt> poly = {A, B, pt(1,1,0), C}; // cuadrado en z=0
    cout << "P en polgono? " << (pointInPolygon3D(P, poly) ? "si" : "no")
         << "\n";
    pt P2(0.5,0.5,0.5);
    cout << "distpol(P2) = " << distpol(P2, poly) << "\n";

    // rotar punto (1,0,0) 90 grados alrededor del eje z
    pt p(1,0,0), axis(0,0,1);
    pt r = rotate(p, axis, DEG_TO_RAD(90.0));
    cout << "Rotado 90 deg alrededor de z: " << r << "\n";

    return 0;
}
#endif
```

4.4 IntersecctionPolygonConvex

```
//use struct polygon and point from Polygon.cpp and Point_and_Segments.cpp

bool inside(const Point &a, const Point &b, const Point &p)
{
    return orientation(a, b, p) >= 0;
}

Point lineIntersection(const Point &a, const Point &b, const Point &c,
     const Point &d)
{
    Point r = b - a, s = d - c;
    double t = (c - a).cross(s) / r.cross(s);
    return a + r * t;
}

vector<Point> convexIntersection(const vector<Point> &A, const
     vector<Point> &B)
{
    vector<Point> output = A;

    for (int i = 0; i < (int)B.size(); ++i)
    {
        Point a = B[i], b = B[(i + 1) % B.size()];
        vector<Point> input = output;
        output.clear();

        for (int j = 0; j < (int)input.size(); ++j)
        {
            Point p = input[j], q = input[(j + 1) % input.size()];
            bool pin = inside(a, b, p), qin = inside(a, b, q);

            if (pin && qin)
                output.push_back(q);
            else if (pin && !qin)
                output.push_back(lineIntersection(a, b, p, q));
            else if (!pin && qin)
            {
                output.push_back(lineIntersection(a, b, p, q));
                output.push_back(q);
            }
        }
    }

    if (output.size() > 1 && output.front() == output.back())
        output.pop_back();
    return output;
}

int main()
{

    vector<Point> A = {{0, 0}, {4, 0}, {4, 4}, {0, 4}};
    vector<Point> B = {{2, -1}, {5, 2}, {2, 5}, {-1, 2}};
    vector<Point> inter = convexIntersection(A, B);
    Polygon interPoly(inter);
    cout << "rea de interseccin: " << interPoly.area() << "\n";
}
```

4.5 Point$_a nd_S egments$

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

const double EPS = 1e-9;
const double PI = acos(-1.0);

int sgn(double x) {
    return (x > EPS) - (x < -EPS);
}

struct Point {
    double x, y;
    Point() : x(0), y(0) {}
    Point(double _x, double _y) : x(_x), y(_y) {}
    Point operator+(const Point& p) const { return Point(x + p.x, y +
        p.y); }
    Point operator-(const Point& p) const { return Point(x - p.x, y -
        p.y); }
    Point operator*(double k) const { return Point(x * k, y * k); }
    Point operator/(double k) const { return Point(x / k, y / k); }
    bool operator<(const Point& p) const {
        if (sgn(x - p.x) != 0) return x < p.x;
        return y < p.y;
    }
    bool operator==(const Point& p) const {
        return sgn(x - p.x) == 0 && sgn(y - p.y) == 0;
    }
    double dot(const Point& p) const { return x * p.x + y * p.y; }
    double cross(const Point& p) const { return x * p.y - y * p.x; }
    double norm2() const { return x * x + y * y; }
    double norm() const { return sqrt(norm2()); }
    //rotate point by angle (in radians) around the origin
    Point rotate(double ang) const {
        double c = cos(ang), s = sin(ang);
        return Point(x * c - y * s, x * s + y * c);
    }
    //angle align with x axis
    double angle() const { return atan2(y, x); }
};

//calculete the distance between two points (distance eucludiana)
double dist(const Point& a, const Point& b) {
    return hypot(a.x - b.x, a.y - b.y);
}

// > 0    giro antihorario
// < 0    giro horario
// = 0    colineales
double orientation(const Point& a, const Point& b, const Point& c) {
    return (b - a).cross(c - a);
}

//distance from point p to line ab
double distancePointLine(const Point& a, const Point& b, const Point& p) {
    return fabs((b - a).cross(p - a)) / dist(a, b);
}
//distance from point p to segment ab
double distancePointSegment(const Point& a, const Point& b, const Point&
     p) {
    Point ap = p - a, ab = b - a;
    double t = ap.dot(ab) / ab.norm2();
    if (t < 0.0) return dist(p, a);
    if (t > 1.0) return dist(p, b);
    Point proj = a + ab * t;
    return dist(p, proj);
}

// Check if segments ab and cd intersect
bool segmentsIntersect(const Point& a, const Point& b, const Point& c,
     const Point& d) {
    double o1 = orientation(a, b, c);
    double o2 = orientation(a, b, d);
    double o3 = orientation(c, d, a);
    double o4 = orientation(c, d, b);

    if (sgn(o1) * sgn(o2) < 0 && sgn(o3) * sgn(o4) < 0)
        return true;
    auto onSegment = [&](const Point& p, const Point& q, const Point& r) {
        return min(p.x, r.x) - EPS <= q.x && q.x <= max(p.x, r.x) + EPS &&
               min(p.y, r.y) - EPS <= q.y && q.y <= max(p.y, r.y) + EPS;
    };

    if (sgn(o1) == 0 && onSegment(a, c, b)) return true;
    if (sgn(o2) == 0 && onSegment(a, d, b)) return true;
    if (sgn(o3) == 0 && onSegment(c, a, d)) return true;
    if (sgn(o4) == 0 && onSegment(c, b, d)) return true;

    return false;
```

```
}

int main() {
    Point A(0, 0), B(4, 4), C(1, 2), D(3, 0);
    Point p1(0,0),p2(1,1),p3(-1,-5);

    cout << "Orientacin A,B,C: " << orientation(A, B, C) << "\n";
    cout << "Distancia de C a lnea AB: " << distancePointLine(A, B, C) <<
        "\n";
    cout << "Segmentos AB y CD se cruzan: " << (segmentsIntersect(A, B,
        C, D) ? "S" : "No") << "\n";
    cout<<"Orientacin p1,p2,p3: "<<orientation(p1,p2,p3)<<"\n";
    return 0;
}
```

4.6 Polygon

```
// for use this structure, include Point_and_Segments.cpp
struct Polygon {
    vector<Point> P;

    Polygon() {}
    Polygon(const vector<Point>& pts) : P(pts) {}

    int n() const { return (int)P.size(); }

    void close() {
        if (P.empty()) return;
        if (!(P.front() == P.back())) P.push_back(P.front());
    }
    double area() const {
        double A = 0;
        for (int i = 0; i < n(); i++) {
            int j = (i + 1) % n();
            A += P[i].cross(P[j]);
        }
        return fabs(A) / 2.0;
    }

    // Signed area (positive = CCW)
    double signedArea() const {
        double A = 0;
        for (int i = 0; i < n(); i++) {
            int j = (i + 1) % n();
            A += P[i].cross(P[j]);
        }
        return A / 2.0;
    }

    //clockwise (CW) if signed area < 0
    bool isCCW() const { return signedArea() > 0; }

    //determine if a point is inside, outside or on the border of the
        polygon
    //  0 -> out, 1 -> in, 2 -> on the border
    int containsPoint(const Point& q) const {
        bool inside = false;
        for (int i = 0, j = n() - 1; i < n(); j = i++) {
            const Point& a = P[i];
            const Point& b = P[j];

            if (fabs(orientation(a, b, q)) < EPS &&
                min(a.x, b.x) - EPS <= q.x && q.x <= max(a.x, b.x) + EPS &&
                min(a.y, b.y) - EPS <= q.y && q.y <= max(a.y, b.y) + EPS)
                return 2; // En el borde

            bool intersect = ((a.y > q.y) != (b.y > q.y)) &&
                             (q.x < (b.x - a.x) * (q.y - a.y) / (b.y - a.y +
                                 EPS) + a.x);
            if (intersect) inside = !inside;
        }
        return inside ? 1 : 0;
    }
};

// Convex Hull using Andrew's monotone chain algorithm
vector<Point> convexHull(vector<Point> pts) {
    int n = pts.size(), k = 0;
    if (n <= 1) return pts;

    sort(pts.begin(), pts.end());
    vector<Point> H(2 * n);

    //lower hull
    for (int i = 0; i < n; ++i) {
        while (k >= 2 && orientation(H[k-2], H[k-1], pts[i]) <= 0) k--;
        H[k++] = pts[i];
    }
```

```
    //upper hull
    for (int i = n - 2, t = k + 1; i >= 0; --i) {
        while (k >= t && orientation(H[k-2], H[k-1], pts[i]) <= 0) k--;
        H[k++] = pts[i];
    }

    H.resize(k - 1);
    return H;
}

// calculte the diameter of a convex polygon (the farthest pair of points)
double polygonDiameter(const vector<Point>& P) {
    int n = P.size();
    if (n == 1) return 0;
    if (n == 2) return dist(P[0], P[1]);

    double maxDist = 0;
    int j = 1;
    for (int i = 0; i < n; ++i) {
        int ni = (i + 1) % n;
        while (fabs((P[ni] - P[i]).cross(P[(j + 1) % n] - P[i])) >
               fabs((P[ni] - P[i]).cross(P[j] - P[i])))
            j = (j + 1) % n;
        maxDist = max(maxDist, dist(P[i], P[j]));
    }
    return maxDist;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    vector<Point> pts = {
        {0,0}, {1,2}, {2,1}, {4,4}, {0,4}, {3,0}
    };

    //Convex Hull
    vector<Point> hull = convexHull(pts);
    cout << "Convex Hull:\n";
    for (auto &p : hull) cout << "(" << p.x << "," << p.y << ") ";
    cout << "\n";

    Polygon poly(hull);
    cout << "rea: " << poly.area() << "\n";

    // poin in polygon test
    Point test(2, 2);
    int inside = poly.containsPoint(test);
    cout << "El punto (2,2) est: "
         << (inside == 1 ? "Dentro" : inside == 2 ? "En el borde" :
             "Fuera") << "\n";

    cout << "Dimetro del polgono: " << polygonDiameter(hull) << "\n";
}
```

4.7 SweepLine

```
//use struct polygon and point from Polygon.cpp and Point_and_Segments.cpp

// Sweep Line Algorithm to detect if any segments intersect
struct Segment
{
    Point p, q;
    int id;
    Segment(Point _p, Point _q, int _id) : p(_p), q(_q), id(_id) {}
};


struct SweepCmp
{
    double x;
    bool operator()(const Segment &a, const Segment &b) const
    {
        auto getY = [&](const Segment &s, double X)
        {
            if (fabs(s.p.x - s.q.x) < EPS)
                return s.p.y;
            return s.p.y + (s.q.y - s.p.y) * (X - s.p.x) / (s.q.x - s.p.x);
        };
        double y1 = getY(a, x), y2 = getY(b, x);
        if (fabs(y1 - y2) > EPS)
            return y1 < y2;
        return a.id < b.id;
    }
};

struct Event
```

```
{
    double x;
    int type; // 0 = inicio, 1 = fin
    Segment seg;
    Event(double _x, int _t, Segment _s) : x(_x), type(_t), seg(_s) {}
    bool operator<(const Event &e) const
    {
        if (fabs(x - e.x) > EPS)
            return x < e.x;
        return type < e.type;
    }
};

bool anySegmentIntersection(vector<Segment> &segs)
{
    vector<Event> events;
    for (auto &s : segs)
    {
        if (s.q < s.p)
            swap(s.p, s.q);
        events.push_back(Event(s.p.x, 0, s));
        events.push_back(Event(s.q.x, 1, s));
    }
    sort(events.begin(), events.end());

    set<Segment, SweepCmp> active;
    SweepCmp cmp;

    auto next_it = [&](auto it)
    { return it == prev(active.end()) ? active.end() : next(it); };
    auto prev_it = [&](auto it)
    { return it == active.begin() ? active.end() : prev(it); };

    for (auto &ev : events)
    {
        cmp.x = ev.x;
        if (ev.type == 0)
        { // Insertar segmento
            auto it = active.insert(ev.seg).first;
            auto it_prev = prev_it(it), it_next = next_it(it);
            if (it_prev != active.end() && segmentsIntersect(it_prev->p,
                it_prev->q, it->p, it->q))
                return true;
            if (it_next != active.end() && segmentsIntersect(it_next->p,
                it_next->q, it->p, it->q))
                return true;
        }
        else
        { // Eliminar
            auto it = active.find(ev.seg);
            if (it == active.end())
                continue;
            auto it_prev = prev_it(it), it_next = next_it(it);
            if (it_prev != active.end() && it_next != active.end() &&
                segmentsIntersect(it_prev->p, it_prev->q, it_next->p,
                    it_next->q))
                return true;
            active.erase(it);
        }
    }
    return false;
}

int main()
{
    vector<Segment> segs = {
        {{0, 0}, {4, 4}, 1},
        {{1, 4}, {4, 1}, 2},
        {{5, 5}, {7, 7}, 3}};
    cout << (anySegmentIntersection(segs) ? "S hay interseccin" : "No hay
        interseccin") << "\n";

    return 0;
}
```

4.8 closestPoints

```
// use Point_and_Segments.cpp
#include <bits/stdc++.h>

double closestPairRec(vector<Point> &pts, int l, int r)
{
    if (r - l <= 3)
    {
        double minD = 1e18;
        for (int i = l; i < r; ++i)
            for (int j = i + 1; j < r; ++j)
                minD = min(minD, dist(pts[i], pts[j]));
```

```
        sort(pts.begin() + l, pts.begin() + r, [](Point a, Point b)
            { return a.y < b.y; });
        return minD;
    }

    int m = (l + r) / 2;
    double midX = pts[m].x;
    double d = min(closestPairRec(pts, l, m), closestPairRec(pts, m, r));

    vector<Point> temp;
    merge(pts.begin() + l, pts.begin() + m, pts.begin() + m, pts.begin()
        + r,
         back_inserter(temp), [](Point a, Point b)
         { return a.y < b.y; });
    copy(temp.begin(), temp.end(), pts.begin() + l);

    vector<Point> strip;
    for (int i = l; i < r; ++i)
        if (fabs(pts[i].x - midX) < d)
            strip.push_back(pts[i]);

    for (int i = 0; i < (int)strip.size(); ++i)
        for (int j = i + 1; j < (int)strip.size() && (strip[j].y -
            strip[i].y) < d; ++j)
            d = min(d, dist(strip[i], strip[j]));

    return d;
}

double closestPair(vector<Point> pts)
{
    sort(pts.begin(), pts.end(), [](Point a, Point b)
        { return a.x < b.x; });
    return closestPairRec(pts, 0, pts.size());
}

int main()
{

    vector<Point> P = {{0, 0}, {3, 4}, {7, 7}, {4, 3}, {1, 1}};
    cout << "Distancia mnima: " << closestPair(P) << "\n";
    return 0;
}
```

4.9 geometry$_{u}$*tils*

```
/*
PI = 3.141592653589793...
RAD_TO_DEG(x) = x * 180 / PI
DEG_TO_RAD(x) = x * PI / 180
Circunferencia completa = 2 * PI * r
 rea  del  crculo = PI *  r
*/

ld DEG_TO_RAD(ld deg) { return deg * PI / 180.0; }
ld RAD_TO_DEG(ld rad) { return rad * 180.0 / PI; }

// -------------------- CRCULO Y ARCO --------------------

/*
d = 2r
l = longitud de la cuerda del arco ( d)
   =  ngulo  central en radianes
*/

ld arco_por_cuerda(ld d, ld l) {
    // longitud del arco correspondiente a una cuerda de longitud l
    // devuelve (arco / permetro)
    return (d * acos(l/d)) / (PI * d);
}

ld longitud_arco(ld r, ld angulo_en_grados) {
    return 2 * PI * r * (angulo_en_grados / 360.0);
}

ld area_sector(ld r, ld angulo_en_grados) {
    return (PI * r * r * angulo_en_grados) / 360.0;
}

ld area_segmento(ld r, ld angulo_en_radianes) {
    //  rea  del segmento circular (sector - tringulo)
    return 0.5 * r * r * (angulo_en_radianes - sin(angulo_en_radianes));
}
// -------------------- TRINGULOS --------------------
ld area_heron(ld a, ld b, ld c) {
    ld s = (a + b + c) / 2;
    return sqrtl(s * (s - a) * (s - b) * (s - c));
}
```

```
ld area_triangulo(pt a, pt b, pt c) {
    return fabsl(cross(b - a, c - a)) / 2.0;
}

ld circunradio(ld a, ld b, ld c) {
    ld A = area_heron(a, b, c);
    return (a * b * c) / (4.0 * A);
}

ld inradio(ld a, ld b, ld c) {
    ld A = area_heron(a, b, c);
    ld s = (a + b + c) / 2;
    return A / s;
}

/*
Teorema del seno:
    a / sin(A) = b / sin(B) = c / sin(C) = 2R

Teorema del coseno:
     c  =  a  +  b  - 2ab * cos(C)

 Pitgoras :
     c  =  a  +  b   (si C = 90 )

Distancia punto a plano (3D):
    d = |Ax + By + Cz + D| / sqrt(A + B +  C )

Vector normal a un plano (3D):
    n = (p2 - p1)   (p3 - p1)

 ngulo  entre dos vectores:
       = arccos( (uv) / (|u| |v|) )
*/
```

4.10 isFigure

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

// ===== CONFIGURACIN BASE =====
#define endl '\n'
typedef long double ld;

const ld EPS = 1e-9;
#define eq(a,b) (fabsl((a)-(b)) < EPS)

struct pt {
    ld x, y;
    pt(ld _x=0, ld _y=0): x(_x), y(_y) {}
    pt operator+(pt p) const { return {x+p.x, y+p.y}; }
    pt operator-(pt p) const { return {x-p.x, y-p.y}; }
    pt operator*(ld d) const { return {x*d, y*d}; }
    bool operator==(pt p) const { return eq(x,p.x) && eq(y,p.y); }
    void read() { cin >> x >> y; }
};

ld dot(pt a, pt b) { return a.x*b.x + a.y*b.y; }
ld cross(pt a, pt b) { return a.x*b.y - a.y*b.x; }
ld norm2(pt a) { return a.x*a.x + a.y*a.y; }
ld dist2(pt a, pt b) { return norm2(a - b); }
bool parallel(pt a, pt b) { return eq(cross(a,b), 0); }

// ===== CLASIFICADORES =====

// cuadrado: lados iguales + ngulos rectos
bool isSquare(pt a, pt b, pt c, pt d) {
    ld ab = dist2(a,b), bc = dist2(b,c), cd = dist2(c,d), da = dist2(d,a);
    return eq(ab,bc) && eq(bc,cd) && eq(cd,da)
        && eq(dot(b-a, d-a), 0);
}

// rectngulo: ngulos rectos, lados opuestos iguales
bool isRectangle(pt a, pt b, pt c, pt d) {
    return parallel(a-b, c-d) && parallel(b-c, d-a)
        && eq(dot(b-a, b-c), 0);
}

// rombo: lados iguales
bool isRhombus(pt a, pt b, pt c, pt d) {
    ld ab = dist2(a,b), bc = dist2(b,c), cd = dist2(c,d), da = dist2(d,a);
    return eq(ab,bc) && eq(bc,cd) && eq(cd,da);
}

// paralelogramo: lados opuestos paralelos
bool isParallelogram(pt a, pt b, pt c, pt d) {
    return parallel(a-b, c-d) && parallel(b-c, d-a);
}
```

```
// trapecio: un par de lados paralelos
bool isTrapezium(pt a, pt b, pt c, pt d) {
    return parallel(a-b, c-d) ^ parallel(b-c, d-a);
}

// cometa (kite): dos pares de lados adyacentes iguales
bool isKite(pt a, pt b, pt c, pt d) {
    ld ab = dist2(a,b), bc = dist2(b,c), cd = dist2(c,d), da = dist2(d,a);
    return (eq(ab,bc) && eq(cd,da)) || (eq(ab,da) && eq(bc,cd));
}

// ===== PROGRAMA PRINCIPAL =====
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    pt a,b,c,d;
    a.read(); b.read(); c.read(); d.read();

    if (isSquare(a,b,c,d)) cout << "square" << endl;
    else if (isRectangle(a,b,c,d)) cout << "rectangle" << endl;
    else if (isRhombus(a,b,c,d)) cout << "rhombus" << endl;
    else if (isParallelogram(a,b,c,d)) cout << "parallelogram" << endl;
    else if (isTrapezium(a,b,c,d)) cout << "trapezium" << endl;
    else if (isKite(a,b,c,d)) cout << "kite" << endl;
    else cout << "none" << endl;

    return 0;
}
```

4.11 point

```
/*
razones trigonometricas

CO = cateto opuesto
CA = cateto adyacente
H = hipotenusa

sin(angulo) = CO / H
cos(angulo) = CA / H
tan(angulo) = CO / CA

sosecante(angulo) = H / CO
secante(angulo) = H / CA
cotangente(angulo) = CA / CO
*/

using ld = long double;

struct point {
    ll x, y, id;
    point(ll a = 0, ll b = 0, int i = -1) {
        x = a;
        y = b;
        id = i;
    }

    point operator -(const point b) {
        return point(x - b.x, y - b.y, -1);
    }

    //cross(a, b) producto escalar
    ll operator ^(const point b) {
        return x * b.y - y * b.x;
    }

    ll operator *(const point b) {
        return x * b.x + y * b.y;
    }
};

ld norm(point a) { // Modulo
    return sqrt(a * a);
}

// ======= POLAR SORT =======
int ret[2][2] = {{3, 2}, {4, 1}};
inline int quad(point p) {
    return ret[p.x >= 0][p.y >= 0];
}

//points sorted in counterclockwise order
bool comp(point a, point b) { // ccw
    int qa = quad(a), qb = quad(b);
    return (qa == qb ? (a ^ b) > 0 : qa < qb);
```

```
}

// ======= POLAR SORT =======

// ang(a1,b1) <= ang(a2,b2)
bool angle_less(point a1, point b1, point a2, point b2) {
    point p1((a1 * b1), abs((a1 ^ b1)));
    point p2((a2 * b2), abs((a2 ^ b2)));
    return (p1 ^ p2) <= 0;
}

int main() {
    //dados n vectores
    //todos con origen en (0, 0)
    //hallar dos vectores diferentes
    //que forman el angulo minimo
    int n;
    cin >> n;
    vector<point> pts(n);

    forn(i, n) {
        ll x, y;
        cin >> x >> y;
        pts[i] = point(x, y, i + 1);
    }

    sort(all(pts), comp);
    array<point, 2> ans = {pts[0], pts[n - 1]};
    forab(i, 1, n) {
        if(angle_less(ans[0], ans[1], pts[i], pts[i - 1])) {
            ans = {pts[i], pts[i - 1]};
        }
    }
    cout << ans[0].id << " " << ans[1].id << ln;
    return 0;
}
```

5 Graph

5.1 2 edge connected comp

```
typedef pair<int, int> pii;

//Given a undirected graph
//you can remove exactly
//one edge from the graph.
//Your task is to minimize
//the number of pairs of vertices (u, v)
//between which there exists a path in this graph

int n, m;
const int MAXN = 1e5 + 5;

int timer, tagTree;
vector<pii> g[MAXN];
//2-edge-connected component tree
//(Bridge Tree)
vi tree[MAXN];
vb vis, isBridge;
vi tin, low;
vi id; // u pertenece a la comp id[u]
vector<array<int, 3>> edges;

//optional
vi cntNodos; //nodos de la componente i
vi tam; // subtree size

//Tarjan
void dfs1(int u, int p) {
    tin[u] = low[u] = ++timer;
    vis[u] = true;
    for(auto &[to, id]: g[u]) {
        if(to == p) continue;
        if(vis[to]) {
            low[u] = min(low[u], tin[to]);
        } else {
            dfs1(to, u);
            low[u] = min(low[u], low[to]);
            if(low[to] > tin[u]) {
                isBridge[id] = true;
            }
        }
    }
}

//assing id
```

```
void dfs2(int u) {
    vis[u] = 1;
    id[u] = tagTree;
    for(auto &[to, id]: g[u]) {
        //skip bridges
        if(isBridge[id]) continue;
        if(!vis[to]) dfs2(to);
    }
}

//build edge tree
void build() {
    timer = 0;
    tagTree = 0;
    dfs1(1, 0);
    fill(all(vis), 0);
    forab(i, 1, n + 1) {
        if(!vis[i]) {
            tagTree++;
            dfs2(i);
        }
    }
    int bridges = 0;
    forab(i, 1, m + 1) {
        if(isBridge[i]) {
            auto [u, v, idx] = edges[i];
            tree[id[u]].pb(id[v]);
            tree[id[v]].pb(id[u]);
        }
    }
}

//do something in bridge tree
// subtree size of each comp
void dfs3(int u, int p) {
    tam[u] = cntNodos[u];
    for(int v: tree[u]) {
        if(v == p) continue;
        dfs3(v, u);
        tam[u] += tam[v];
    }
}

int main() {
    int t;
    cin >> t;
    while(t--) {
        cin >> n >> m;
        vis = vb(n + 1, 0);
        isBridge = vb(m + 1, 0);

        tin = vi(n + 1);
        low = vi(n + 1);
        id = vi(n + 1);
        cntNodos = vi(n + 1, 0);
        tam = vi(n + 1, 0);

        edges = vector<array<int, 3>>(m + 1);
        int a, b;
        forab(id, 1, m + 1) {
            cin >> a >> b;
            g[a].pb({b, id});
            g[b].pb({a, id});
            edges[id] = {a, b, id};
        }

        build();

        forab(u, 1, n + 1) {
            cntNodos[id[u]]++;
        }
        dfs3(1, 0); //subtree size
        ll mx_remove = 0;
        forab(v, 1, tagTree + 1) {
            mx_remove = max(mx_remove, (ll) (n - tam[v]) * tam[v]);
        }
        ll cant = 1LL * n * (n - 1LL) / 2;
        cout << (cant - mx_remove) << ln;
        if(t) {
            forab(i, 1, n + 1) {
                g[i].clear();
                tree[i].clear();
            }
        }
    }
    return 0;
}

//minimun number of edges
//such that there are no
```

```
//bridges in the new graph
//ans = ceil(leaf / 2)

//if we can remove only once
//edge, we can reduce the number
//of bridges to
//(bridges - tree diameter)
```

5.2 Belman Ford

```
int n, m;
const int MAXN = 2505;
const int MAXM = 5005;
const ll oo = 1e18 + 5;

array<ll, 3> edges[MAXM];
int par[MAXN];
ll dis[MAXN];

int main() {
    cin >> n >> m;
    forn(i, m) {
        int a, b, w;
        cin >> a >> b >> w;
        edges[i] = {a, b, -w};
    }

    //negative cycle
    vb change(n + 1, 0);
    forab(i, 1, n + 1) dis[i] = oo;
    dis[1] = 0;
    int x;
    forn(i, n) {
        x = -1;
        for(auto [a, b, w]: edges) {
            if(dis[a] < oo) {
                if(dis[b] > dis[a] + w) {
                    if(i == n - 1) change[b] = 1;
                    dis[b] = max(-oo, dis[a] + w);
                    par[b] = a;
                    x = b;
                }
            }
        }
    }
    bool cycle = (x != -1);
    bool valid_cycle = 0;

    if(cycle) {
        /*
        int y = x;
        forn(i, n) y = par[y];
        vi path;
        for(int cur = y;; cur = par[cur]){
          path.pb(cur);
          if(cur == y && sz(path) > 1) break;
        }
        */
        dfs1(1);//normal graph
        dfs2(n);//reverse graph

        forab(i, 1, n + 1) {
            if(change[i]) {
                if(vis1[i] && vis1[n]) {
                    if(vis2[i] && vis2[1]) {
                        valid_cycle = 1;
                        break;
                    }
                }
            }
        }
    }

    if(valid_cycle) {
        puts("-1");
    } else {
        printf("%lld\n", -dis[n]);
    }
    return 0;
}

//shortest path
forab(i, 1, n + 1) dis[i] = oo;
int source = 1;
int dest = n;
par[source] = -1;
d[source] = 0;
```

```
forn (i, n - 1) {
    for (auto [a, b, w] : edges) {
        if (dis[a] < oo) {
            dis[b] = min(dis[b], dis[a] + w);
            par[b] = a;
        }
    }
}
if (dis[dest] == oo) {
    //no path
} else {
    vi path;
    for (int cur = dest; cur != -1; cur = par[cur])
        path.pb(cur);
    reverse(all(path));
}
```

5.3 Bipartite check

```
const int N = 2e5+5;
vi g[N];
vi color;
bool bipartite = 1;

void dfs(int u, int c) {
    if(~color[u]) {
        if(color[u] ^ c) {
            bipartite = 0;
        }
        return;
    }
    color[u] = c;
    for(int next: g[u]) {
        dfs(next, c^1);
    }
}

int main() {
    int n, m;
    cin >> n >> m;
    color = vi(n + 1, -1);
    int a, b;
    forn(i, m) {
        cin >> a >> b;
        g[a].pb(b);
        g[b].pb(a);
    }
    for(int i = 1; i < n + 1 && bipartite; i++) {
        if(color[i] < 0) dfs(i, 1);
    }
    if(bipartite) {
        //color of each node
        forab(i, 1, n + 1) {
            cout << color[i] + 1 <<" ";
        }
    } else {
        cout << "No bipartite" << ln;
    }
    return 0;
}

/*validar distancias
entre dos nodos A y B
si es par o impar
if(!bipartite){
        dist pares e impares;
}else{
        if(color[a] == color[b]){
                dist par
        }
        if(color[a] != color[b]){
                dist impar
        }
}*/
```

5.4 Bridges and articulation Points

```
typedef pair<int, int> ii;

// Tarjan's Algorithm
// Finding bridges in a graph in O(N + M)
// Undirected graph.

const int MAXN = (int) 1e5 + 5; // cant nodos
vi g[MAXN];
vi tin, low;
```

```
int timer = 0;
vector<ii> bridges;
vb is_articulation, vis;
int n, m;

void dfs(int v, int p = -1){
  vis[v] = true;
  tin[v] = low[v] = timer++;
  int children = 0;
  for (int u : g[v]) {
    if (u == p) continue;
    if (vis[u]) {
      low[v] = min(low[v], tin[u]);
    } else {
      dfs(u, v);
      low[v] = min(low[v], low[u]);
      if (low[u] >= tin[v] && p != -1)
        is_articulation[v] = true;
      ++children;
      if (low[u] > tin[v])
        bridges.pb({min(u, v), max(u, v)});
    }
  }
  if (p == -1 && children > 1)
    is_articulation[v] = true;
}

int main(){
  cin >> n >> m;
  tin = vi(n + 1, -1);
  low = vi(n + 1, -1);
  vis = vb(n + 1, 0);
  is_articulation = vb(n + 1, 0);

  int a, b;
  forn(i, m) {
    cin >> a >> b;
    g[a].pb(b); g[b].pb(a);
  }
  // find bridges and articulation points
  forab (i, 1, n + 1) {
    if (!vis[i]) dfs(i);
  }
  // print articulation points
  forab (i, 1, n + 1) {
    if (is_articulation[i]) {
      cout << i << ln;
    }
  }
  // print bridges
  for (auto [u, v] : bridges) {
    cout << u << " " << v;
  }
  return 0;
}
```

5.5 Centroid decomposition

```
const int MAXN = 2e5 + 5;
vi g[MAXN];
int tam[MAXN];
bool is_removed[MAXN];

int dfs_sz(int u, int p) {
    tam[u] = 1;
    for(int v: g[u]) {
        if(v == p || is_removed[v]) continue;
        dfs_sz(v, u);
        tam[u] += tam[v];
    }
    return tam[u];
}

int find_centroid(int u, int p, int tsz) {
    for(int v: g[u]) {
        if(v == p || is_removed[v]) continue;
        if(tam[v] * 2 > tsz) {
            return find_centroid(v, u, tsz);
        }
    }
    return u;
}

void yo(vi &path, int u, int p, int deep) {
    path.pb(deep);
    for(int v: g[u]) {
        if(v == p || is_removed[v]) continue;
```

```
        yo(path, v, u, deep + 1);
    }
}

ll decomp(int root, int k) {
    int c = find_centroid(root, root, dfs_sz(root, root));
    is_removed[c] = 1;

    //resolver para subarbol de centroid
    ll ans = 0;
    vi cnt(tam[root] + 1, 0);
    cnt[0] = 1;

    for(int v : g[c]) {
        if(is_removed[v]) continue;
        vi path;
        yo(path, v, v, 0);
        for (int d : path) if (k - d - 1 >= 0 && k - d - 1 < tam[root])
                ans += cnt[k - d - 1];
        for (int d : path) cnt[d + 1]++;

    }

    for (int v: g[c]) if (!is_removed[v]) ans += decomp(v, k);
    is_removed[c] = 0;
    return ans;
}

int main() {
    int n, k;
    cin >> n >> k;
    forn(i, n - 1) {
        int a, b;
        cin >> a >> b;
        g[a].pb(b);
        g[b].pb(a);
    }
    cout << 1LL * decomp(1, k) << ln;
    return 0;
}
```

5.6 DAG reachability dynamic

```
//Tomado de :
//
    https://github.com/ShahjalalShohag/code-library/blob/main/Graph%20Theory/DAG

/*add_edge(s, t): insert edge (s,t) to the network if it does not make a
    cycle
is_reachable(s, t): return true iff there is a path s --> t
Algorithm by G.F. ITALIANO
Complexity: amortized O(n) per update*/
// 0-indexed
struct Italiano {
    int n;
    vector<vector<int>> par;
    vector<vector<vector<int>>> child;
    Italiano(int n)
        : n(n), par(n, vector<int>(n, -1)), child(n,
            vector<vector<int>>(n)) {}
    bool is_reachable(int s, int t) { return s == t || par[s][t] >= 0; }
    bool add_edge(int s, int t) {
        if (is_reachable(t, s))
            return 0; // break DAG condition
        if (is_reachable(s, t))
            return 1; // no-modification performed
        for (int p = 0; p < n; ++p) {
            if (is_reachable(p, s) && !is_reachable(p, t))
                meld(p, t, s, t);
        }
        return 1;
    }
    void meld(int root, int sub, int u, int v) {
        par[root][v] = u;
        child[root][u].push_back(v);
        for (int c : child[sub][v]) {
            if (!is_reachable(root, c))
                meld(root, sub, v, c);
        }
    }
};

// add edges one by one. if it breaks DAG law then print it

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

```
    cin >> n >> m;
    Italiano t(n);
    while (m--) {
        int u, v;
        cin >> u >> v;
        --u;
        --v;
        if (t.is_reachable(v, u))
            cout << u + 1 << " " << v + 1 << ln;
        else
            t.add_edge(u, v);
    }
    cout << "0 0" << ln;
    return 0;
}

/*
    dado una lista de aristas dirigidas
    agregar solo las necesarias para que no hayan ciclos

    decir cuales aristas NO fueron agregadas
*/
```

5.7 DSU on tree

```
// O(n log(n))
int n, q;
const int MAXN = 2e5 + 5;
vi g[MAXN];
int a[MAXN];//val of each node
int tam[MAXN];//subtree size
int ans[MAXN];//ans to queries
vector<array<int, 2>> queries[MAXN];
vector<map<int, int>> st(MAXN);

void dfs_sz(int u, int p){
    tam[u] = 1;
    for(int v: g[u]){
        if(v == p) continue;
        dfs_sz(v, u);
        tam[u] += tam[v];
    }
}

void dfs(int u, int p){
    int big = -1, mx = -1;
    for(int v: g[u]){
        if(v == p) continue;
        if(tam[v] > mx){
            mx = tam[v];
            big = v;
        }
    }

    for(int v: g[u]){
        if(v == p || v == big) continue;
        dfs(v, u);
    }

    if(big != -1){
        dfs(big, u);
        swap(st[u], st[big]);
        //ahora u tiene el set mas grande
    }

    //small to large
    for(int v: g[u]){
        if(v == p || v == big) continue;
        //en caso de un merge mas complejo
        //hacer una funcion merge(u, v)
        for(auto [c, occ] : st[v]){
            st[u][c] += occ;
        }
    }

    st[u][a[u]]++;//incluir u
    //ans queries
    for(auto [c, idx]: queries[u]){
        ans[idx] = st[u].count(c) ? st[u][c] : 0;
    }
}

int main() {
    cin >> n >> q;
    forab(i, 1, n + 1) cin >> a[i];
    //read tree
```

```
    //en el subarbol de u
    // cuantos nodos tienen color c?
    forn(i, q){
        int u, c;
        cin >> u >> c;
        queries[u].pb({c, i});
    }

    dfs_sz(1, 0);
    dfs(1, 0);

    forn(i, q){
        cout << ans[i] << ln;
    }
    return 0;
}
```

5.8 Detect cycle

```
//directed graph
const int MAXN = 2e5 + 5;
vi g[MAXN];
int color[MAXN];
int in_cycle[MAXN];
int par[MAXN];

void dfs(int u){
    if(color[u] == 2) return;
    color[u] = 1;
    for(int v: g[u]){
        if(color[v] == 0){
            par[v] = u;
            dfs(v);
        }else if(color[v] == 1){
            //traverse the cycle
            int ini = u, fin = v;
            int cur = u;
            in_cycle[cur] = 1;
            while(cur != v){
                cur = par[cur];
                in_cycle[cur] = 1;
            }
            in_cycle[v] = 1;
        }
    }
    color[u] = 2;
}

/*
    forab(i, 1, n + 1) {
        in_cycle[i] = 0;
        color[i] = 0;
    }
    forab(i, 1, n + 1) {
        if(color[i] == 0) dfs(i);
    }
*/

//undirected graph
const int MAXN = 2e5 + 5;
int color[MAXN];
int mark[MAXN];
int par[MAXN];
vi g[MAXN];

void dfs(int u, int p, int &cy){
    if(color[u] == 2){
        return;
    }
    if(color[u] == 1){
        cy++;
        int cur = p;
        mark[cur] = cy;
        while(cur != u){
            cur = par[cur];
            mark[cur] = cy;
        }
        return;
    }
    par[u] = p;
    color[u] = 1;
    for(int v: g[u]){
        if(v == par[u]) continue;
        dfs(v, u, cy);
    }
    color[u] = 2;
```

```
}

/*
    forab(i, 1, n + 1){
        color[i] = 0;
        par[i] = i;
        mark[i] = -1;
    }
    int cycles = 0;
    dfs(1, 1, cycles);

    forab(i, 1, n + 1){
        if(mark[i] != -1){
            found = true;
        }
    }
*/
```

5.9 Dijkstra

```
struct edge {
    int u;
    ll w;
    bool operator < (const edge &x) const {
        return x.w < w;
    }
};

int n, m;
const int MAXN = 2e5 + 5;
const ll oo = (1LL << 62);
vector<edge> g[MAXN];
vi par(MAXN);

vll dijkstra(int s) {
    priority_queue<edge> pq;
    vll dis(n + 1, oo);
    pq.push(edge{s, 0});
    dis[s] = 0;
    par[s] = -1;
    while (sz(pq)) {
        auto [u, cur_dis] = pq.top();
        pq.pop();
        ll min_dis = dis[u];
        if(cur_dis != min_dis) continue;
        for (auto [v, w] : g[u]) {
            if (dis[u] + w < dis[v]) {
                dis[v] = dis[u] + w;
                par[v] = u;
                pq.push(edge{v, dis[v]});
            }
        }
    }
    return dis;
}

int main() {
    cin >> n >> m;
    forn(i, m) {
        int a, b, w;
        cin >> a >> b >> w;
        g[a].pb({b, w});
        g[b].pb({a, w});
    }
    vll dis = dijkstra(1);
    if(dis[n] != oo) {
        vi path;
        for (int i = n; i != -1; i = par[i]) {
            path.pb(i);
        }
        rforn (i, sz(path)) {
            cout << path[i] << " ";
        }
        cout << ln;
    } else {
        cout << "-1" << ln;
    }
    return 0;
}

//validar si una arista
//esta en el camino mas corto
//desde (a, b)
vll da = dijkstra(1);
ll minDis = da[n];
vll db = dijkstra(n);
vector<ll> shortes_path;
```

```
for (auto [u, v, w] : edges) {
    if(da[u] + w + db[v] == minDis) {
        //edge is on shortest path
        //do something
    }
    if(da[v] + w + db[u] == minDis) {
        //edge is on shortest path
        //do something
    }
}
```

5.10 Directed MST

```
const int N = 3e5 + 9;

const long long inf = 1e18;

template <typename T> struct PQ {
    long long sum = 0;
    priority_queue<T, vector<T>, greater<T>> Q;
    void push(T x) {
        x.w -= sum;
        Q.push(x);
    }
    T pop() {
        auto ans = Q.top();
        Q.pop();
        ans.w += sum;
        return ans;
    }
    int size() { return Q.size(); }
    void add(long long x) { sum += x; }
    void merge(PQ &&x) {
        if (size() < x.size())
            swap(sum, x.sum), swap(Q, x.Q);
        while (x.size()) {
            auto tmp = x.pop();
            tmp.w -= sum;
            Q.push(tmp);
        }
    }
};
```

```
struct edge {
    int u, v;
    long long w;
    bool operator>(const edge &rhs) const { return w > rhs.w; }
};
struct DSU {
    vector<int> par;
    DSU(int n) : par(n, -1) {}
    int root(int i) { return par[i] < 0 ? i : par[i] = root(par[i]); }
    void set_par(int c, int p) { par[c] = p; }
};
// returns parents of each vertex
// each edge should be distinct
// it assumes that a solution exists (all vertices are reachable from
    root)
// 0 indexed
// Takes ~300ms for n = 2e5
vector<int> DMST(int n, int root, const vector<edge> &edges) {
    vector<int> u(2 * n - 1, -1), par(2 * n - 1, -1);
    edge par_edge[2 * n - 1];
    vector<int> child[2 * n - 1];

    PQ<edge> Q[2 * n - 1];
    for (auto e : edges)
        Q[e.v].push(e);
    for (int i = 0; i < n; i++)
        Q[(i + 1) % n].push({i, (i + 1) % n, inf});

    int super = n;
    DSU dsu(2 * n - 1);
    int head = 0;
    while (Q[head].size()) {
        auto x = Q[head].pop();
        int nxt_root = dsu.root(x.u);
        if (nxt_root == head)
            continue;
        u[head] = nxt_root;
        par_edge[head] = x;
        if (u[nxt_root] == -1)
            head = nxt_root;
        else {
            int j = nxt_root;
            do {
                Q[j].add(-par_edge[j].w);
                Q[super].merge(move(Q[j]));
```

```
            assert(u[j] != -1);
            child[super].push_back(j);
            j = dsu.root(u[j]);
        } while (j != nxt_root);
        for (auto u : child[super])
            par[u] = super, dsu.set_par(u, super);
        head = super++;
    }
}
vector<int> res(2 * n - 1, -1);
queue<int> q;
q.push(root);
while (q.size()) {
    int u = q.front();
    q.pop();
    while (par[u] != -1) {
        for (auto v : child[par[u]]) {
            if (v != u) {
                res[par_edge[v].v] = par_edge[v].u;
                q.push(par_edge[v].v);
                par[v] = -1;
            }
        }
        u = par[u];
    }
}
res[root] = root;
res.resize(n);
return res;
}
int main() {
    int n, m, root;
    cin >> n >> m >> root;
    vector<edge> edges(m);
    for (auto &e : edges)
        cin >> e.u >> e.v >> e.w;
    auto res = DMST(n, root, edges);

    unordered_map<int, int> W[n];
    for (auto u : edges)
        W[u.v][u.u] = u.w;

    long long ans = 0;
    for (int i = 0; i < n; i++)
        if (i != root)
            ans += W[i][res[i]];
    cout << ans << '\n';
    for (auto x : res)
        cout << x << ' ';
    cout << '\n';
    return 0;
}
/*
    Directed MST

    dado un grafo dirigido, un nodo S
    y M aristas con pesos
    hallar el MST tal que todos
    los nodos sean alcanzables desde S
*/
```

5.11 Dynamic diameter online

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

//Tomado de:
//https://github.com/ShahjalalShohag/code-library/

const int N = 1e5 + 9;
//edges weights must need to be non-negative

// f[x] = distance from root to x
// x <= y <= z
// max{f[x] - 2f[y] + f[z]}
// a --> f[x]
// b --> -2f[y]
// c --> f[x] - 2f[y]
// d --> -2f[y] + f[z]
// e --> f[x] - 2f[y] + f[z]
struct node {
    long long a = 0, b = 0, c = 0, d = 0, e = 0;
    node operator + (const node &oth) const {
        node ret;
        ret.a = max(a, oth.a);
        ret.b = max(b, oth.b);
        ret.c = max(max(c, oth.c), a + oth.b);
        ret.d = max(max(d, oth.d), b + oth.a);
```

```
        ret.e = max(max(e, oth.e), max(c + oth.a, a + oth.d));
        return ret;
    }
};
struct ST {
#define lc (n << 1)
#define rc ((n << 1) | 1)
    long long lazy[4 * N * 2];
    node t[4 * N * 2];
    ST() {
        memset(lazy, 0, sizeof lazy);
    }
    inline void push(int n, int b, int e) {
        if(lazy[n] == 0) return;
        long long v = lazy[n];
        t[n].a += v, t[n].b -= v + v, t[n].c -= v, t[n].d -= v;
        if (b != e) lazy[lc] += v, lazy[rc] += v;
        lazy[n] = 0;
    }
    inline void pull(int n) {
        t[n] = t[lc] + t[rc];
    }
    void upd(int n, int b, int e, int i, int j, long long v) {
        push(n, b, e);
        if(j < b || e < i) return;
        if(i <= b && e <= j) {
            lazy[n] += v;
            push(n, b, e);
            return;
        }
        int mid = (b + e) >> 1;
        upd(lc, b, mid, i, j, v);
        upd(rc, mid + 1, e, i, j, v);
        pull(n);
    }
} t;

long long W[N];
vector<pair<int, int>> g[N];
int T, st[N * 2], en[N * 2], pos[N];
void dfs (int u, int p = 0) {
    st[u] = ++T;
    for (auto e : g[u]) {
        int v = e.first, i = e.second;
        if (v == p) continue;
        pos[i] = v;
        dfs(v, u);
        ++T;
    }
    en[u] = T;
}
int32_t main() {
    int n, q;
    long long mod;
    cin >> n >> q >> mod;
    for (int i = 1; i < n; i++) {
        int u, v;
        long long w;
        cin >> u >> v >> w;
        g[u].push_back({v, i});
        g[v].push_back({u, i});
        W[i] = w;
    }
    dfs(1);
    for (int i = 1; i < n; ++i) t.upd(1, 1, T, st[pos[i]], en[pos[i]],
        W[i]);
    long long last = 0;
    while (q--) {
        int d;
        long long e;
        cin >> d >> e;
        d = 1 + (d + last) % (n - 1);
        e = (e + last) % mod;
        t.upd(1, 1, T, st[pos[d]], en[pos[d]], e - W[d]);
        last = t.t[1].e, W[d] = e;
        cout << last << '\n';
    }
    return 0;
}
/*


Dynamic diameter tree

You are given a weighted undirected tree on n
 vertices and a list of q updates.
 Each update changes the weight of one edge.
 The task is to output the diameter of the tree after each update.

*/
```

5.12 Flattenig Tree

```
//Warning
//STree recursive [l, r] from 0
//STree iterative [l, r + 1) from 0
template<typename T>
struct STree {
    int n;
    vector<T> st, lazy;
    T neutro = T(0ll);

    STree(int m) {
        n = m;
        st.resize(n * 4);
        lazy.resize(n * 4);
    }

    STree(vector<T> &a) {
        n = sz(a);
        st.resize(n * 4);
        lazy.resize(n * 4);
        build(1, 0, n - 1, a);
    }

    T oper(T a, T b) {
        return a | b;
    }

    void build(int v, int tl, int tr, vector<T> &a) {
        if(tl == tr) {
            st[v] = 1ll << a[tl];
            return;
        }
        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, a);
        build(v * 2 + 1, tm + 1, tr, a);
        st[v] = oper(st[v * 2], st[v * 2 + 1]);
    }

    void push(int v, int tl, int tr) {
        if (!lazy[v]) return;
        st[v] = 1ll << lazy[v];
        if (tl != tr) {
            lazy[v * 2] = lazy[v * 2 + 1] = lazy[v];
        }
        lazy[v] = 0;
    }

    void upd(int v, int tl, int tr, int l, int r, T val) {
        push(v, tl, tr);
        if(tr < l || tl > r) return;
        if (tl >= l && tr <= r) {
            lazy[v] = val;
            push(v, tl, tr);
            return;
        }
        int tm = (tl + tr) / 2;
        upd(v * 2, tl, tm, l, r, val);
        upd(v * 2 + 1, tm + 1, tr, l, r, val);
        st[v] = oper(st[v * 2], st[v * 2 + 1]);
    }

    T query(int v, int tl, int tr, int l, int r) {
        push(v, tl, tr);
        if(tl > r || tr < l) return neutro;
        if (l <= tl && tr <= r) return st[v];
        int tm = (tl + tr) / 2;
        return oper(query(v * 2, tl, tm, l, r), query(v * 2 + 1, tm + 1,
            tr, l, r));
    }

    void upd(int l, int r, T val) {
        upd(1, 0, n - 1, l, r, val);
    }
    T query(int l, int r) {
        return query(1, 0, n - 1, l, r);
    }
};

const int MAXN = 4e5 +5;
vi g[MAXN];
int tin[MAXN];
int tout[MAXN];
vll aplanado;
int timer = 0;

int n, q;
vi color;
void dfs(int node, int par) {
    tin[node] = timer;
```

```
    //se usa el color como el aplanado
    //se puede cambiar segun sea
    aplanado[timer] = color[node];
    timer++;
    for(int next: g[node]) {
        if(next != par) {
            dfs(next, node);
        }
    }
    tout[node] = timer - 1;
}

//https://codeforces.com/contest/620/problem/E
int main() {
    cin >> n >> q;
    color = vi(n + 1);
    aplanado = vll(n + 1);
    forab(i, 1, n + 1) {
        cin >> color[i];
    }
    int a, b;
    forn(i, n - 1) {
        cin >> a >> b;
        g[a].pb(b);
        g[b].pb(a);
    }
    dfs(1, 0);
    int oper, v, c;
    STree<ll> st(aplanado);
    while(q--) {
        cin >> oper;
        if(oper == 1) {
            cin >> v >> c;
            // l = tin[v]
            // r = tout[v]
            // (l, r) = subArbol del nodo v
            //cambiamos el color del subarbol de v
            st.upd(tin[v], tout[v], c);
        } else {
            cin >> v;
            ll ans = st.query(tin[v], tout[v]);
            //cantidad de colores diferentes en
            //el subarbol de v
            cout << __builtin_popcountll(ans) << ln;
        }
    }
    return 0;
}
```

5.13 Floyd Warshall

```
const ll oo = 1e18 / 10;
int main() {
    int n, m, q;
    cin >> n >> m >> q;
    vector<vll> g(n, vll(n));
    forn(i, n) {
        forab(j, i + 1, n) {
            g[i][j] = g[j][i] = oo;
        }
    }
    int a, b;
    ll w;
    forn(i, m) {
        cin >> a >> b >> w;
        a--;
        b--;
        g[a][b] = g[b][a] = min(g[a][b], w);
    }
    //Floyd Warshall O(n^3)
    forn(k, n) {
        forn (i, n) {
            forn (j, n) {
                g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
            }
        }
    }
    while(q--) {
        cin >> a >> b;
        a--;
        b--;
        cout << (g[a][b] == oo ? -1: g[a][b]) <<ln;
    }
    return 0;
}
```

5.14 Kosaraju

```
const int MAXN = 1e5 + 5;
int n, m;
vi g[MAXN];
vi rg[MAXN];
vb vis(MAXN);
vi id(MAXN);
int tagSCC;
vector<vi> SCC;
vi curComp;

void dfs1(int u, stack<int> &st) {
    vis[u] = 1;
    for(int v: g[u]) {
        if(!vis[v]) dfs1(v, st);
    }
    st.push(u);
}
void dfs2(int u) {
    vis[u] = 1;
    id[u] = tagSCC;
    curComp.pb(u);
    //u esta en la SCC id[u]
    for(int v: rg[u]) {
        if(!vis[v])dfs2(v);
    }
}

int main() {
    cin >> n >> m;
    int a, b;
    forn(i, m) {
        cin >> a >> b;
        g[a].pb(b);
        rg[b].pb(a);
    }
    stack<int> st;
    forab(i, 1, n + 1) {
        if(!vis[i])dfs1(i, st);
    }
    vis = vb(n + 1, 0);
    while(!st.empty()) {
        int u = st.top();
        st.pop();
        if(vis[u]) continue;
        tagSCC++;
        curComp.clear();
        dfs2(u);
        SCC.pb(curComp);
    }
    //build DAG
    const int MAXDAG = tagSCC + 1;
    vi dag[MAXDAG];
    forab(u, 1, n + 1) {
        for(int v: g[u]) {
            if(id[u] == id[v]) continue;
            dag[id[u]].pb(id[v]);
        }
    }
    //print SCC
    int cp = 1;
    for(vi i: SCC) {
        printf("SCC #%d: ", cp++);
        for(int u: i) {
            printf("%d ", u);
        }
        printf("\n");
    }
    return 0;
}
```

5.15 LCA binary lifting

```
int n, m;
const int MAXN = 3e5 + 5;
const int LOG = 21; //calcular

vi g[MAXN];
int tin[MAXN];
int tout[MAXN];
int deep[MAXN];

int timer = 1;
int up[MAXN][LOG];
int max_up[MAXN][LOG];
```

```
map<pair<int, int>, int> weights;

void dfs(int u, int p) {
  tin[u] = timer++;
  deep[u] = deep[p] + 1;

  // initialize binary lifting arrays
  up[u][0] = p;
  max_up[u][0] = weights[ {p, u}];

  for (int v : g[u]) {
    if (v != p) {
      dfs(v, u);
    }
  }
  tout[u] = timer - 1;
}

bool is_ancestor(int x, int y) {
  return tin[x] <= tin[y] && tout[y] <= tout[x];
}

int lca(int x, int y) {
  if (is_ancestor(x, y)) return x;

  for (int i = LOG - 1; i >= 0; i--) {
    if (!is_ancestor(up[x][i], y)) {
      x = up[x][i];
    }
  }
  return up[x][0];
}

void build(int root) {
  dfs(root, root);
  forab(k, 1, LOG) {
    forab(u, 1, n + 1) {
      up[u][k] = up[up[u][k - 1]][k - 1];
      // take max of weights from left and right
      max_up[u][k] = max(max_up[u][k - 1], max_up[up[u][k - 1]][k - 1]);
    }
  }
}
//get max weight edge from (u, lcaU)
int get_max_up(int u, int v) {
  if(deep[u] < deep[v]) swap(u, v);
  int df = abs(deep[v] - deep[u]);
  int res = 0;
  forn(i, LOG) {
    if(df & (1 << i)) {
      //up the lowest node
      res = max(res, max_up[u][i]);
      u = up[u][i];
    }
  }
  return res;
}

bool cmp(array<int, 4> &a, array<int, 4> &b) {
  return a[2] < b[2];
}

int main() {
  cin >> n >> m;
  vector<array<int, 4>> edges(m);
  forn(i, m) {
    int a, b, w;
    cin >> a >> b >> w;
    edges[i] = {a, b, w, i};
    weights[ {a, b}] = w;
    weights[ {b, a}] = w;
  }

  //Kruskal
  sort(all(edges), cmp);
  UnionFind uf(n + 1);
  ll mst = 0;
  for(auto [a, b, w, id]: edges) {
    if(!uf.isSame(a, b)) {
      uf.unite(a, b);
      mst += w;
      g[a].pb(b);
      g[b].pb(a);
    }
  }

  build(1);
  vll ans(m);
```

```
  for(auto [a, b, w, id]: edges) {
    int LCA = lca(a, b);
    int mx_a = get_max_up(a, LCA);
    int mx_b = get_max_up(b, LCA);

    ll res = mst - max(mx_a, mx_b) + w;
    ans[id] = res;
  }

  forn(i, m) cout << ans[i] << ln;
  return 0;
}
```

5.16 LCA

```
// Lowest Common Ancestor
// Se usa el Euler tour para encontrar un nodo que
//  est  entre u y v que tiene la altura mas baja
struct LCA {
  vi height, euler, first, st;//SegmentTree
  vector<vi> table; // Sparse Table
  vector<bool> vis;
  int n;
  bool SegTree;

  // LCA with segment tree -> option = 1
  // LCA with Sparse Table -> option = 2
  LCA(vector<vi>& g, int root, int option) {
    n = g.size();
    height.resize(n);
    first.resize(n);
    euler.reserve(n * 2);
    vis.assign(n, false);
    dfs(g, root, 0);
    if (option == 1) {
      SegTree = true;
      int m = euler.size();
      st.resize(m * 4);
      build(1, 0, m - 1);
    } else if(option == 2) {
      SegTree = false;
      build();
    }
  }

  void dfs(vector<vi>& g, int node, int h) {
    vis[node] = true;
    height[node] = h;
    first[node] = euler.size();
    euler.pb(node);
    for (auto to : g[node]) {
      if (!vis[to]) {
        dfs(g, to, h + 1);
        euler.pb(node);
      }
    }
  }

  void build(int node, int b, int e) {
    if (b == e) {
      st[node] = euler[b];
    } else {
      int mid = (b + e) / 2;
      build(node << 1, b, mid);
      build(node << 1 | 1, mid + 1, e);
      int l = st[node << 1], r = st[node << 1 | 1];
      st[node] = (height[l] < height[r]) ? l : r;
    }
  }

  int query(int node, int b, int e, int L, int R) {
    if (b > R || e < L)
      return -1;
    if (b >= L && e <= R)
      return st[node];
    int mid = (b + e) >> 1;

    int left = query(node << 1, b, mid, L, R);
    int right = query(node << 1 | 1, mid + 1, e, L, R);
    if (left == -1)
      return right;
    if (right == -1)
      return left;
    return height[left] < height[right] ? left : right;
  }

  int lca(int u, int v) {
    int left = first[u], right = first[v];
```

```
    if (left > right)
      swap(left, right);
    if (SegTree) {
      return query(1, 0, euler.size() - 1, left, right);
    } else {
      return query(left, right);
    }
  }

  void build() {
    int N = euler.size();
    int K = int(log2(N)) + 1;
    table.assign(N + 1, vi(K));
    forn(i, N) {
      table[i][0] = euler[i];
    }
    forn(j, K - 1) {
      for (int i = 0; i + (1 << (j + 1)) <= N; ++i) {
        int a = table[i][j];
        int b = table[i + (1 << j)][j];
        table[i][j + 1] = height[a] < height[b] ? a : b;
      }
    }
  }

  int query(int l, int r) {
    int k = 31 - __builtin_clz(r - l + 1);
    int a = table[l][k];
    int b = table[r - (1 << k) + 1][k];
    return height[a] < height[b] ? a : b;
  }
};

int main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  cout.tie(0);
  int n, q;
  cin >> n >> q;
  vector<vi> g(n + 1, vi());
  int a, b;
  for (int i = 0; i < n - 1; i++) {
    cin >> a >> b;
    g[a].pb(b);
    g[b].pb(a);
  }
  LCA lca(g, 1, 2);
  while (q--) {
    cin >> a >> b;
    int LCA = lca.lca(a, b);
    cout << lca.height[a] + lca.height[b] - (2 * lca.height[LCA]) << ln;
  }

  return 0;
}
```

5.17 Manhattan MST

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

const int N = 2e5 + 9;

int n;
vector<pair<int, int>> g[N];
struct PT {
    int x, y, id;
    bool operator<(const PT &p) const {
        return x == p.x ? y < p.y : x < p.x;
    }
} p[N];
struct node {
    int val, id;
} t[N];
struct DSU {
    int p[N];
    void init(int n) {
        for (int i = 1; i <= n; i++)
            p[i] = i;
    }
    int find(int u) {
        return p[u] == u ? u : p[u] = find(p[u]);
    }
    void merge(int u, int v) {
        p[find(u)] = find(v);
    }
} dsu;
struct edge {
```

```
    int u, v, w;
    bool operator<(const edge &p) const {
        return w < p.w;
    }
};
vector<edge> edges;
int query(int x) {
    int r = 2e9 + 10, id = -1;
    for (; x <= n; x += (x & -x))
        if (t[x].val < r)
            r = t[x].val, id = t[x].id;
    return id;
}
void modify(int x, int w, int id) {
    for (; x > 0; x -= (x & -x))
        if (t[x].val > w)
            t[x].val = w, t[x].id = id;
}
int dist(PT &a, PT &b) {
    return abs(a.x - b.x) + abs(a.y - b.y);
}
void add(int u, int v, int w) {
    edges.push_back({u, v, w});
}
long long Kruskal() {
    dsu.init(n);
    sort(edges.begin(), edges.end());
    long long ans = 0;
    for (edge e : edges) {
        int u = e.u, v = e.v, w = e.w;
        if (dsu.find(u) != dsu.find(v)) {
            ans += w;
            g[u].push_back({v, w});
            // g[v].push_back({u, w});
            dsu.merge(u, v);
        }
    }
    return ans;
}
void Manhattan() {
    for (int i = 1; i <= n; ++i)
        p[i].id = i;
    for (int dir = 1; dir <= 4; ++dir) {
        if (dir == 2 || dir == 4) {
            for (int i = 1; i <= n; ++i)
                swap(p[i].x, p[i].y);
        } else if (dir == 3) {
            for (int i = 1; i <= n; ++i)
                p[i].x = -p[i].x;
        }
        sort(p + 1, p + 1 + n);
        vector<int> v;
        static int a[N];
        for (int i = 1; i <= n; ++i)
            a[i] = p[i].y - p[i].x, v.push_back(a[i]);
        sort(v.begin(), v.end());
        v.erase(unique(v.begin(), v.end()), v.end());
        for (int i = 1; i <= n; ++i)
            a[i] = lower_bound(v.begin(), v.end(), a[i]) - v.begin() + 1;
        for (int i = 1; i <= n; ++i)
            t[i].val = 2e9 + 10, t[i].id = -1;
        for (int i = n; i >= 1; --i) {
            int pos = query(a[i]);
            if (pos != -1)
                add(p[i].id, p[pos].id, dist(p[i], p[pos]));
            modify(a[i], p[i].x + p[i].y, i);
        }
    }
}
int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> p[i].x >> p[i].y;
    Manhattan();
    cout << Kruskal() << '\n';
    for (int u = 1; u <= n; u++) {
        for (auto x : g[u])
            cout << u - 1 << ' ' << x.first - 1 << '\n';
    }
    return 0;
}
/*

    Manhattan MST

    dados N puntos en el plano,
    su distancia entre ellos es
    |xi - xj| + |yi - yj|
```

```
    calcular el MST
*/
```

5.18 Prims

```
//MST
//O(E log N )
struct edge {
   int v;
   ll w;
   int idx;

   bool operator < (const edge &x) const {
      return w > x.w;
   }
};

int n, m;
const int MAXN = 3e5 + 5;
vector<edge> g[MAXN];
vector<array<ll, 4>> edges;

vector<vector<edge>> prims(int src){
   vector<vector<edge>> tree(n + 1);
   priority_queue<edge> pq;

   vb vis(n + 1, 0);
   vis[src] = 1;

   for (edge &i: g[src]) {
      if (!vis[i.v]){
         pq.push(i);
      }
   }

   ll mst = 0;
   while (sz(pq)) {
      edge u = pq.top();
      pq.pop();

      if (!vis[u.v]) {
         //arista del MST
         auto [uu, vv, ww, id] = edges[u.idx];
         tree[uu].pb(edge{vv, ww, id});
         tree[vv].pb(edge{uu, ww, id});
         mst += ww;
         vis[u.v] = 1;

         for (edge &v : g[u.v]) {
            if (!vis[v.v]) {
               pq.push(v);
            }
         }
      }
   }
   return tree;
}

int main() {
   cin >> n >> m;
   edges.resize(m);
   forn(i, m){
      int a, b, w;
      cin >> a >> b >> w;
      g[a].pb({b, w, i});
      g[b].pb({a, w, i});
      edges[i] = {a, b, w, i};
   }

   vector<vector<edge>> tree = prims(1);
   return 0;
}
```

5.19 Tarjan

```
int n, m;
const int MAXN = 1e5 + 5;
vi g[MAXN];

struct Tarjan {
    vi low, num, comp;
    stack<int> st;
    int n, scc, cont;
    const int oo = int(1e9);
```

```
    Tarjan(int n) {
        this->n = n;
        low.resize(n);
        num.assign(n, -1);
        comp.resize(n);
        scc = cont = 0;
    }

    void dfs(int v) {
        low[v] = num[v] = cont++;
        st.push(v);
        for (int &u : g[v]) {
            if (num[u] == -1) dfs(u);
            low[v] = min(low[v], low[u]);
        }
        if (low[v] == num[v]) {
            int u;
            do {
                u = st.top();
                st.pop();
                low[u] = oo;
                comp[u] = scc;
            } while (u != v);
            scc++;
        }
    };

    vector<vi> getComp() {
        forab (i, 1, n)
        if (num[i] == -1) dfs(i);
        vector<vi> cc(scc);
        forab(i, 1, n) {
            cc[comp[i]].pb(i);
        }
        return cc;
    }
};

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> n >> m;
    int a, b;
    forn(i, m) {
        cin >> a >> b;
        g[a].pb(b);
    }
    Tarjan t(n + 1);
    vector<vi> components = t.getComp();
    for(vi cc: components) {
        for(int u: cc) {
            cout << u << " ";
        }
        cout << ln;
    }
    return 0;
}
```

5.20 Tree Diameter

```
using ii = pair<int, int>;

int n;
const int MAXN = 2e5 + 5;
vi g[MAXN];

ii dfs(int u, int p){
    ii ans = make_pair(0, u);
    for(int v: g[u]){
        if(v == p) continue;
        ans = max(ans, dfs(v, u));
    }
    ans.first++;
    return ans;
}

int main() {
    auto [dis, nodo] = dfs(1, 0);
    auto [diam, raiz] = dfs(nodo, 0);

    cout << diam - 1 << ln;
    return 0;
}
```

5.21 Tree distances

```
const int MAXN = 2e5 + 5;
int n;
ll down[MAXN], up[MAXN];
vector<pair<int, ll>> tree[MAXN];

void dfs_down(int u, int parent) {
    down[u] = 0;
    for (auto [v, w] : tree[u]) {
        if (v != parent) {
            dfs_down(v, u);
            down[u] = max(down[u], down[v] + w);
        }
    }
}

void dfs_up(int u, int parent) {
    int max1 = -1, max2 = -1; // Las dos mayores distancias hacia abajo
        desde u
    // Encuentra las dos mayores distancias hacia abajo
    for (auto [v, w] : tree[u]) {
        if (v != parent) {
            int total_dist = down[v] + w;
            if (total_dist > max1) {
                max2 = max1;
                max1 = total_dist;
            } else if (total_dist > max2) {
                max2 = total_dist;
            }
        }
    }

    //up[v] para cada hijo v
    for (auto [v, w] : tree[u]) {
        if (v != parent) {
            if (down[v] + w == max1) {
                up[v] = max(up[u] + w, max2 + w);
            } else {
                up[v] = max(up[u] + w, max1 + w);
            }
            dfs_up(v, u);
        }
    }
}

int main() {
    //build tree

    dfs_down(1, 1);
    up[1] = 0;
    dfs_up(1, 1);

    forab(i, 1, n + 1) {
        ll best = max(up[i], down[i]);
        if(i == n) printf("%lld\n", best);
        else printf("%lld ", best);
    }
    return 0;
}
```

5.22 Two Sat

```
//si A y  A  estan en la misma SCC,
//entonces no hay solucion.
struct sat2 {
    int n;
    vector<vector<vector<int>>> g;
    vector<bool> vis, val;
    vector<int> comp;
    stack<int> st;

    sat2(int n) : n(n), g(2, vector<vector<int>>(2*n)), vis(2*n),
        val(2*n), comp(2*n) { }

    int neg(int x) {
        return 2*n-x-1;
    }
    void make_true(int u) {
        add_edge(neg(u), u);
    }
    void make_false(int u) {
        make_true(neg(u));
    }
    void add_or(int u, int v) {
        implication(neg(u), v);
```

```
}
void diff(int u, int v) {
    eq(u, neg(v));
}
void eq(int u, int v) {
    implication(u, v);
    implication(v, u);
}
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].pb(v);
    g[1][v].pb(u);
}

void dfs(int id, int u, int t = 0) {
    vis[u] = true;
    for (auto &v : g[id][u])
        if (!vis[v]) dfs(id, v, t);
    if (id) comp[u] = t;
    else st.push(u);
}

void kosaraju() {
    for (int u = 0; u < n; u++) {
        if (!vis[u]) dfs(0, u);
        if (!vis[neg(u)]) dfs(0, neg(u));
    }
    vis.assign(2*n, false);
    int t = 0;
    while (!st.empty()) {
        int u = st.top();
        st.pop();
        if (!vis[u]) dfs(1, u, t++);
    }
}

bool check() {
    kosaraju();
    for (int i = 0; i < n; i++) {
        if (comp[i] == comp[neg(i)]) return false;
        val[i] = comp[i] > comp[neg(i)];
        }
        return true;
    }
};

int main() {
    int t, n, m;
    cin >> t;
    int caso = 1;
    while(t--) {
        cin >> n >> m;
        sat2 ts(n + 1);
        forn(i, m) {
            int a, b;
            cin >> a >> b;
            ts.implication(a, ts.neg(b));
            ts.implication(ts.neg(a), b);
        }
        bool ok = ts.check();
        if(ok)puts("si hay solucion");
        else puts("no hay solucion");
    }
    return 0;
}
```

5.23 bfs grid

```
//find shortest path from A to B
int n, m;
const int MAXN = 1002;
char grid[MAXN][MAXN];

//der, abajo, izq, arriba
vi dx = {0, 1, 0, -1};
vi dy = {1, 0, -1, 0};

bool valid(int i, int j) {
    if(i < 0 || i >= n || j < 0 || j >= m) {
        return 0;
    }
    if(grid[i][j] == '#') return 0;
    return 1;
}
```

```
string path;
bool bfs(int i, int j) {
    queue<array<int, 3>> q;
    vector<vb> vis(n, vb(m, 0));
    char br[n][m];
    q.push({i, j, 0});
    vis[i][j] = 1;

    while(sz(q)) {
        auto &[x, y, dis] = q.front();
        q.pop();
        if(grid[x][y] == 'B') { //found
            while(true) { //build path
                path.pb(br[x][y]);
                if(path.back() == 'R') y--;
                if(path.back() == 'D') x--;
                if(path.back() == 'L') y++;
                if(path.back() == 'U') x++;
                if(x == i && y == j) break;
                if(!valid(x, y)) break;
            }
            return 1;
        }

        forn(k, 4) { // dir
            int nx = x + dx[k];
            int ny = y + dy[k];
            if(valid(nx, ny) && !vis[nx][ny]) {
                if(k == 0) br[nx][ny] = 'R';
                if(k == 1) br[nx][ny] = 'D';
                if(k == 2) br[nx][ny] = 'L';
                if(k == 3) br[nx][ny] = 'U';
                vis[nx][ny] = 1;
                q.push({nx, ny, dis + 1});
            }
        }
    }
    return 0;
}
```

5.24 functional graph

```
const int MAXN = 2e5 + 5;
vi rg[MAXN]; // reverse graph
vll valor(MAXN);
vb vis(MAXN, 0);
vi p(MAXN); //parent
vi in_cycle(MAXN, 0);

void mark(int u) {
    if(vis[u]) return;
    vis[u] = 1;
    for(int v: rg[u]) {
        mark(v);
    }
}

int cycle(int u) {
    int x = p[u];
    int y = p[p[u]];
    while(x != y) {
        x = p[x];
        y = p[p[y]];
    }
    //traverse the cycle
    ll min_cycle = valor[x];
    x = p[x];
    while(x != y) {
        in_cycle[x] = 1;
        min_cycle = min(min_cycle, valor[x]);
        x = p[x];
    }
    mark(x);
    return min_cycle;
}

int main() {
    int n;
    cin >> n;
    forab(i, 1, n + 1) {
        cin >> valor[i];
    }

    forab(i, 1, n + 1) {
        cin >> p[i];
        rg[p[i]].pb(i);
```

```
    }

    ll ans = 0;
    forab(i, 1, n + 1) {
        if(!vis[i]) {
            ans += cycle(i);
        }
    }
    cout << ans << ln;
    return 0;
}

//When we are swapping 2
//elements that belong to one cycle,
//we are splitting our cycle into 2
//parts.

//min swaps(a[i], a[a[i]]) to sort
//the permutation = n - #cycles
```

5.25 topo sort

```
vi topo_sort(vi &grado) {
    vi topo;
    queue<int> q;
    forab(i, 1, n + 1)
    if(grado[i] == 0) q.push(i);
    while(sz(q)) {
        int u = q.front();
        q.pop();
        topo.pb(u);
        for(int v: g[u]) {
            if(--grado[v] == 0) {
                q.push(v);
            }
        }
    }
    if(sz(topo) == n) return topo;
    else return vi();
}
```

6 Math

6.1 Binary

```
//numeros desde [0, n] en binario
vi pre(int n) {
  vi bin;
  bin.pb(0);
  forab(i, 1, n + 1) {
    int j = 32;
    int x = i;
    string s;
    bool turn = 0;
    while (j--) {
      int bit = (x >> j) & 1;
      if (bit)
        turn = 1;
      if (turn) {
        s += (bit ? "1" : "0");
      }
    }
    int X = stoi(s);
    bin.pb(X);
  }
  return bin;
}

string binString(ll x) {
  int j = 32;//bits
  string s;
  bool turn = 0;
  while (j--) {
    int bit = (x >> j) & 1;
    if (bit) turn = 1;
    if (turn) s += (bit ? "1" : "0");
  }
  if(sz(s) == 0) s+="0";
  return s;
}
//dado un N decir si se puede formar
//por la multiplicacion de numeros
//en su representacion binaria
//N = bin * bin
```

```
//121 = 11 * 11
int main() {
  int t;
  cin >> t;
  vi bin = pre(32);
  int MAX = 1e5 + 10;
  vb sol(MAX, 0);
  sol[1] = 1;
  forab(i, 1, MAX) {
    if(sol[i]) {
      for(int x : bin) {
        if((ll) i * x < MAX) sol[i*x] = 1;
      }
    }
  }
  while(t--) {
    ll n;
    cin >> n;
    cout << (sol[n] ? "YES" :"NO" ) <<ln;
  }
  return 0;
}
```

6.2 Divisors of a number

```
//todos los divisores n en O(sqrt(n))
vll divisors(ll n) {
    vll div;
    for (ll i = 1; i * i <= n; i++) {
        if (n % i == 0) {
            div.pb(i);
            if ((n/i) != i)
                div.pb(n / i);
        }
    }
    return div;
}
```

6.3 ExtendedEuclid

```
ll euclid(ll a, ll b, ll &x, ll &y) {
    if(b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    ll xi, yi;
    ll g = euclid(b, a % b, xi, yi);
    x = yi;
    y = xi - yi * (a / b);
    return g;
}

int main() {
    //hallar Ax + By + C = 0
    ll a, b, c;
    cin >> a >> b >> c;
    ll x, y;
    ll g = euclid(a, b, x, y);
    if(c % g) cout << "-1" << ln;
    else cout << (-(c / g) * x) << " " << (-(c / g) * y) << ln;
    return 0;
}
```

6.4 Factorials

```
// define MOD
ll MOD = 1e9 + 7;

//use binpow() method here

const int MAXN = 1e6 + 1;
ll fact[MAXN];
ll ifact[MAXN];

void cal_factorials(int n) {
    fact[0] = fact[1] = 1;
    for (int i = 2; i <= n; i++) {
        fact[i] = (fact[i - 1] * i) % MOD;
    }

    //fact_inv O(n)
```

```
    ifact[n] = binpow(fact[n], MOD - 2);
    for (int i = n - 1; i >= 0; i--) {
        ifact[i] = ((i + 1) * ifact[i + 1]) % MOD;
    }
}

ll binomial_coefficient(int n, int k) {
    return fact[n] * ifact[k] % MOD * ifact[n - k] % MOD;
}

int main() {
    int t;
    cin >> t;
    int caso = 1;
    cal_factorials(MAXN);
    while(t--) {
        int n, k;
        cin >> n >> k;
        ll ans = binomial_coefficient(n, k);
        printf("Case %d: %lld\n", caso++, ans);
    }
    return 0;
}
```

6.5 Factorization sieve

```
const int MAX = 10e6;
int primediv[MAX]; // 10^6
vll primes;

void sieve() {
    forn(i, MAX) primediv[i] = i;
    int root = sqrt(MAX) + 1;
    forab(i, 2, MAX) {
        if (primediv[i] != i) continue;
        primes.pb(i);
        if (i > root) continue;
        form(j, i * i, MAX, i) primediv[j] = i;
    }
}

map<ll, int> factorize(ll n) { // n <= 10^12
    map<ll, int> factors;
    for (int i = 0; i < sz(primes) && n >= MAX; ++i) {
        while (n % primes[i] == 0) {
            factors[primes[i]]++;
            n /= primes[i];
        }
    }
    if (n >= MAX) {
        factors[n]++;
        return factors;
    }
    while (n > 1) {
        factors[primediv[n]]++;
        n /= primediv[n];
    }
    return factors;
}
```

6.6 FibLog

```
typedef pair<ll, ll> pll;
//first = n-esimo termino
//second = siguiente termino
pll fib_log(ll n, ll mod){
  if (n == 0) return {0, 1};
  auto [a, b] = fib_log(n >> 1, mod);
  ll c = a * (2*b - a + mod) % mod;
  ll d = ((a*a % mod) + (b*b % mod)) % mod;
  if (n & 1) return {d, (c + d) % mod};
  else return {c, d};
}
```

6.7 Linear Sieve

```
//O(n)
const int MAXN = 10000000;
int lp[MAXN + 1]; //minimo factor primo
vi pr;
```

```
void cribaLineal() {
    for (int i = 2; i <= MAXN; ++i) {
        if (lp[i] == 0) {
            lp[i] = i;
            pr.push_back(i);
        }
        for (int j= 0; j < sz(pr) && pr[j] <= lp[i] && i * pr[j] <= MAXN;
             ++j)
            lp[i * pr[j]] = pr[j];
    }
}
```

6.8 Math utils

```
ll MOD = 1e9 + 7;

ll inv_mod(ll a){
  //binpow con MOD igual
  return binpow(a, MOD - 2);
}

ll suma(ll a, ll b){
  return ((a % MOD) + (b % MOD)) % MOD;
}

ll resta(ll a, ll b){
  return ((a % MOD) - (b % MOD) + MOD) % MOD;
}

ll multi(ll a, ll b){
  return ((a % MOD) * (b % MOD)) % MOD;
}

ll divi(ll a, ll b){
  return multi(a, inv_mod(b));
}

//GCD
ll gcd(ll a, ll b) {
  return b == 0 ? (a < 0 ? -a : a) : gcd(b, a % b);
}
//LCM
ll lcm(ll a, ll b) {
  ll lcm = (a / gcd(a, b)) * b;
  return lcm > 0 ? lcm : -lcm;
}
//lcm(A, B, C, ....) =
//p1 max(a1, b1, c1, ...) *
//p2 max(a2, b2, c2, ...) *
//p3 max(a3, b3, c3, ...) * ... ...* pk max(ak, bk, ck, ...)
//pi = prime, ai = exponent

//cuantas veces B puede restar a A
x = floor(A/B + 0.0)

//distancia de X a Y dando saltos de tamao D
int dis(int x, int y, int d) {
  return (abs(x - y) + (d - 1)) / d;
}
```

6.9 Miller Rabin

```
bool probably_prime(ll n, ll a, ll d, int s){
    ll x = binpow(a, d, n);
    if(x == 1 || x+1 == n) return true;
    forn(r, s){
        x = mulmod(x,x,n);
        if(x == 1) return false;
        if(x+1 == n) return true;
    }
    return false;
}

bool miller_rabin(ll n){//check (n is prime)?
    if(n < 2) return false;
    const int a[] = {2,3,5,7,11,13,17,19,23};
    int s = -1;
    ll d = n-1;
    while(!d&1) d >>= 1, s++;

    forn(i, 9){
        if(n == a[i]) return true;
        if(!probably_prime(n, a[i], d, s))
            return false;
    }
```

```
    return true;
}
```

6.10 Modular Inverse

```
int inv[MAXN];

void modular_inverse_range(int m){
    inv[0] = 0; inv[1] = 1;
    forab(i, 2, MAXN)
        inv[i] = (-(m/i)*inv[m%i] + m) % m;
}

int modular_inverse_binpow(int a, int m){
    return binpow(a, phi(m)-1, m);
}

int modular_inverse_extEuclid(int a, int m){
    int x, y;
    int g = extEuclid(a, m, x, y);
    if(g != 1)
        return -1;
    x = (x % m + m) % m;
    return x;
}

vi inversos(vi a, int m){
    vi inv;
    int v = 1;
    forn(i, sz(a)){
        inv.pb(v);
        v = (v * a[i]) % m;
    }

    int x, y;
    extEuclid(v, m, x, y);
    x = (x % m + m) % m;

    rforn(i, sz(a)){
        inv[i] = inv[i] * x;
        x = (x * a[i]) % m;
    }
    return inv;
}
```

6.11 Phi Euler

```
typedef unsigned long long ull;

int phi(int n) {
  int result = n;
  for(int i = 2; i * i <= n; ++i) {
    if(n % i) continue;
    while(n % i == 0)
      n /= i;
    result -= result / i;
  }

  if(n > 1) result -= result / n;
  return result;
}

//number of integers
//which are coprime
//gcd(a, b) == 1

vi phi_1_to_n(int n) {
  vi phi;
  forab(i, 0, n + 1) phi.pb(i);

  for(int i = 2; i <= n; ++i) {
    if(phi[i] != i) continue;

    for(int j = i; j <= n; j += i)
      phi[j] -= phi[j] / i;
  }
  return phi;
}

vi phi_1_to_n2(int n) {
  vi phi;
  forab(i, 0, n + 1) phi.pb(i - 1);
  phi[1] = 1;

  for(int i = 2; i <= n; ++i) {
```

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


int main() {
  int t; cin >> t;
  int caso = 1;
  int MAXN = 5e6;

  vi phi = phi_1_to_n(MAXN);

  //sumatoria en rango de ph[i] * phi[i]
  vector<ull> pf(MAXN + 1, 0);
  forab(i, 2, MAXN + 1){
    pf[i] = pf[i - 1] + (ull(phi[i]) * ull(phi[i]));
  }

  while(t--){
    int a, b;
    cin >> a >> b;
    ull ans = pf[b] - pf[a - 1];
    printf("Case %d: %llu\n", caso++, ans);
  }
  return 0;
}
```

6.12 Segmented Sieve

```
vector<int> prime; // sqrt(MAX R)

vector<ll> segmented_criba(ll l, ll r) {
    l = max(l, 2ll);
    vector<bool> vis(r - l + 1);
    for (int &pp : prime) {
        if ((ll) pp * pp > r) break;
        ll mn = (l + pp - 1) / pp;
        if (mn == 1ll) mn++;
        mn *= pp;
        for (ll i = mn; i <= r; i += pp) {
            vis[i - l] = true;
        }
    }
    vector<ll> ans;
    forn (i, sz(vis)) if (!vis[i]) ans.push_back(l + i);
    return ans;
}
```

6.13 Sieve

```
const int MAX = int(1e6);
bitset<MAX + 5> bs;
vi prime;

void sieve() {
    bs.set();
    bs[0] = bs[1] = 0;
    form (i, 2, MAX + 1, 1) {
        if (bs[i]) {
            prime.pb(i);
            for (ll j = (ll) i * i; j <= MAX; j += i) {
                bs[j] = 0;
            }
        }
    }
}
```

6.14 Ternary Search

```
double f(double x) {
  return x;
}

double ternary_search(double l, double r) {
  double eps = 1e-9;
  while (r - l > eps) {
    double m1 = l + (r - l) / 3;
    double m2 = r - (r - l) / 3;
    double f1 = f(m1);
    double f2 = f(m2);
    if (f1 < f2) l = m1;
    else r = m2;
```

```
  }
  return f(l);
}
```

6.15 $_{int128}$

```
using i128 = __int128;

bool cmp(i128 x, i128 y) { return x > y; }

i128 read() {
    i128 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(i128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x > 9) print(x / 10);
    putchar(x % 10 + '0');
}
```

6.16 binpow

```
//a^b % MOD
ll binpow(ll a, ll b, ll m) {
    a %= m;
    ll res = 1;
    while (b > 0) {
        if (b & 1)
            res = (res * a) % m;
        a = (a * a) % m;
        b >>= 1;
    }
    return res;
}

ll binpow(ll a, ll b) {
    ll res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a;
        a = a * a;
        b >>= 1;
    }
    return res;
}
```

6.17 mint

```
const ll MOD = 1e9 + 7;

struct mint {
    ll v;

    mint() : v(0) {}
    mint(ll x) : v(x % MOD) {
        if (v < 0) v += MOD;
    }

    mint &operator++() { //a++
        v++;
        if (v == MOD) v = 0;
        return *this;
    }

    mint &operator--() { //a--
        if (v == 0) v = MOD;
        v--;
        return *this;
    }
```

```
mint &operator+=(const mint &other) { //a += b
    v += other.v;
    if (v >= MOD) v -= MOD;
    return *this;
}
mint &operator-=(const mint &other) { //a -= b
    v -= other.v;
    if (v < 0) v += MOD;
    return *this;
}
mint &operator*=(const mint &other) { //a *= b
    v = ll(ll(v) * ll(other.v) % MOD);
    return *this;
}
mint &operator/=(const mint &other) { // a /= b
    return *this *= other.inv();
}

friend mint operator+(const mint &a, const mint &b) {
    return mint(a) += b;
}
friend mint operator-(const mint &a, const mint &b) {
    return mint(a) -= b;
}
friend mint operator*(const mint &a, const mint &b) {
    return mint(a) *= b;
}
friend mint operator/(const mint &a, const mint &b) {
    return mint(a) /= b;
}

friend mint power(mint a, ll p) { // pow(a, p)
    mint res = 1;
    while (p > 0) {
        if (p % 2 == 1)
            res *= a;
        p >>= 1;
        a *= a;
    }
    return res;
}

static ll minv(ll a, ll m) {
    a %= m;
    assert(a);
    return a == 1 ? 1 : ll(m - ll(minv(m, a)) * ll(m) / a);
}

friend bool operator==(const mint &a, const mint &b) {
    return a.v == b.v;
}
friend bool operator!=(const mint &a, const mint &b) {
    return a.v != b.v;
}
friend bool operator>(const mint &a, const mint &b) {
    return a.v > b.v;
}
friend bool operator<(const mint &a, const mint &b) {
    return a.v < b.v;
}

mint neg() const {
    mint res;
    res.v = v ? MOD - v : 0;
    return res;
}

friend mint neg(const mint &m) {
    return m.neg();
}

mint inv() const {
    mint res;
    res.v = minv(v, MOD);
    return res;
}

friend mint inv(const mint &m) {
    return m.inv();
}

mint operator-() const {
    return neg();
}
mint operator+() const {
    return mint(*this);
}
};
```

6.18 oper with string

```
//multiply big numbers
string multyply(string a, int b){
  int carry = 0;
  string ans = "";
  forn(i, sz(a)){
    carry = ((a[i] - '0') * b + carry);
    ans += carry % 10 + '0';
    carry /= 10;
  }
  while(carry != 0){
    ans += carry % 10 + '0';
    carry /= 10;
  }
  return ans;
}

string res = "1";
for(auto [p, ex] : fac){
  forn(i, ex){
    res = multyply(res, p);
  }
}
reverse(all(res));
```

6.19 polynomial

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

using ll = long long;
const double EPS = 1e-9;
const ll mod = 1000000007;

typedef int tp;

template <class T = tp>
struct poly
{
    vector<T> c;
    poly() {}
    poly(initializer_list<T> v) : c(v) {}
    poly(const vector<T> &v) : c(v) {}
    poly(int n) : c(n) {}

    T &operator[](int i) { return c[i]; }
    const T &operator[](int i) const { return c[i]; }

    poly operator+(const poly &o) const
    {
        int m = c.size(), n = o.c.size();
        poly res(max(m, n));
        for (int i = 0; i < m; i++)
            res[i] += c[i];
        for (int i = 0; i < n; i++)
            res[i] += o.c[i];
        return res;
    }

    poly operator*(tp k) const
    {
        poly res(c.size());
        for (int i = 0; i < (int)c.size(); i++)
            res[i] = c[i] * k;
        return res;
    }

    poly operator*(const poly &o) const
    {
        int m = c.size(), n = o.c.size();
        poly res(m + n - 1);
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                res[i + j] += c[i] * o.c[j];
        return res;
    }

    poly operator-(const poly &o) const { return *this + (o * -1); }

    T operator()(tp v) const
    {
        T sum = 0;
        for (int i = (int)c.size() - 1; i >= 0; --i)
            sum = sum * v + c[i];
        return sum;
    }
};
```

```
// -------- Mtodos adicionales --------

// races enteras posibles
set<tp> roots(poly<> p)
{
    set<tp> r;
    while (!p.c.empty() && !p.c.back())
        p.c.pop_back();
    if (p.c.empty())
        return r;
    if (p(0) == 0)
        r.insert(0);

    tp a0 = 0, an = abs(p[p.c.size() - 1]);
    for (int k = 0; !a0;)
        a0 = abs(p[k++]);

    vector<tp> ps, qs;
    for (int i = 1; i * i <= a0; i++)
        if (a0 % i == 0)
        {
            ps.push_back(i);
            ps.push_back(a0 / i);
        }
    for (int i = 1; i * i <= an; i++)
        if (an % i == 0)
        {
            qs.push_back(i);
            qs.push_back(an / i);
        }

    for (auto pt : ps)
        for (auto qt : qs)
            if (pt % qt == 0)
            {
                tp x = pt / qt;
                if (p(x) == 0)
                    r.insert(x);
                if (p(-x) == 0)
                    r.insert(-x);
            }
    return r;
}
```

```
// Divisin sinttica (Ruffini)
pair<poly<>, tp> ruffini(poly<> p, tp r)
{
    int n = p.c.size() - 1;
    vector<tp> b(n);
    b[n - 1] = p[n];
    for (int k = n - 2; k >= 0; --k)
        b[k] = p[k + 1] + r * b[k + 1];
    return {poly<>(b), p[0] + r * b[0]};
}

// Divisin polinmica (double)
pair<poly<>, poly<>> polydiv(poly<> p, poly<> q)
{
    int n = p.c.size() - q.c.size() + 1;
    vector<tp> b(n);
    for (int k = n - 1; k >= 0; --k)
    {
        b[k] = p.c.back() / q.c.back();
        for (int i = 0; i < (int)q.c.size(); i++)
            p[i + k] -= b[k] * q[i];
        p.c.pop_back();
    }
    while (!p.c.empty() && fabs(p.c.back()) < EPS)
        p.c.pop_back();
    return {poly<>(b), p};
}

// Interpolacin de Lagrange
poly<> interpolate(vector<tp> x, vector<tp> y)
{
    poly<> q = {1}, S = {0};
    for (tp a : x)
        q = poly<>({-a, 1}) * q;
    for (int i = 0; i < (int)x.size(); i++)
    {
        auto Li = ruffini(q, x[i]).first;
        Li = Li * (1.0 / Li(x[i])); // para int habra que usar fracciones
        S = S + Li * y[i];
    }
    return S;
}

// Construccin a partir de races
vector<ll> coef(vector<ll> roots, bool first = true)
```

```
{
    int l = roots.size() + 1;
    vector<ll> c(10002, 0), m(10002, 0);
    c[0] = 1;
    for (int k = 0; k < (int)roots.size(); k++)
    {
        for (int i = 1; i < l; i++)
            m[i] = c[i] + c[i - 1] * roots[k];
        for (int i = 1; i < l; i++)
            c[i] = m[i];
    }
    ll sign = first ? 1 : -1;
    for (int i = 0; i < (int)roots.size(); i++)
    {
        sign *= -1LL;
        m[i + 1] *= sign;
    }
    return m;
}

// mdulo seguro
inline ll modn(ll x)
{
    x %= mod;
    if (x < 0)
        x += mod;
    return x;
}

// Interpolacin rpida modular
ll interpolateAndEvaluate(ll k, int inix, vector<ll> &y)
{
    ll den = 1, num = 1;
    int len = inix + y.size() - 1;
    for (int i = inix; i < len; i++)
    {
        num = (num * (k - (i + 1))) % mod;
        den = (den * modn(-1ll * i)) % mod;
    }
    auto divmod = [&](ll a, ll b)
    { return a * 1LL * std::pow(b, mod - 2) % mod; }; // suponiendo mod
        primo

    ll res = (y[0] * divmod(num, den)) % mod;
    return res;
}

// -------- Ejemplo --------
int main()
{
    poly<> p = {-6, 11, -6, 1}; //-6 + 11x - 6x^2 + x^3
    cout << "p(1) = " << p(1) << "\n";

    auto r = roots(p);
    cout << "Raices enteras: ";
    for (auto x : r)
        cout << x << " ";
    cout << "\n";

    auto d = ruffini(p, 1);
    cout << "Division por (x-1): resto=" << d.second << "\n";

    return 0;
}
```

7 Probability

7.1 Dice probability

```
/*
    se tira un dado N veces, el dado tiene los
    numeros del 1 al 6, cual es la probabilidad
    de que la suma caiga entre [a, b]
*/

using ld = long double;

int n, a, b;
const int MAXN = 101;
ld dp[MAXN][605];
bool vis[MAXN][605];

ld go(int i, int s){
    if(i == n){
        if(s >= a && s <= b){
            return 1.0;
```

```
        }
        return 0;
    }
    if(vis[i][s]){
        return dp[i][s];
    }
    ld ans = 0;
    for(int j : {1, 2, 3, 4, 5, 6}){
        ans += (1 / 6.0) * go(i + 1, s + j);
    }
    vis[i][s] = 1;
    return dp[i][s] = ans;
}

int main() {
    cin >> n >> a >> b;
    ld ans = go(0, 0);
    printf("%.6LF\n", ans);
    return 0;
}

//teorema de la probabilidad total
// P(x) = P1 * P(x | 1) +
//        P2 * P(x | 2) +
//        P3 * P(x | 3) + ...

// Pi = probabilidad de que suceda i
// P(x | i)
//probabilidad de x dado que el evento i ya ha ocurrido

/*
    tener cuidado cuando piden
    imprimir un numero flotante
    con (rounding half to even)
    usar python para no tener
    problemas de precision
*/
```

7.2 Rob

```
/*
    hay N bancos y se quiere robar
    la maxima cantidad de dinero posible

    limit = prob tolerable de ser atrapado.

    si decide robar el banco i
    roba money[i] dinero
    hay prob p[i] de ser atrapado


    todas las prob son independientes.

*/

int n;
double limit;
const int MAXN = 105;

int money[MAXN];
double p[MAXN];

double dp[100 * 100 + 5];

int main() {
    cin >> limit >> n;
    int total = 0;
    forn(i, n) {
        cin >> money[i] >> p[i];
        total += money[i];
    }
    memset(dp, 0, sizeof dp);
    dp[0] = 1.0; //max prob de robar 0 de dinero sin ser capturado
    forn(i, n) {
        for(int val = total; val >= 0; val--) {
            //maximizar la probabilidad de NO ser atrapado
            if(val - money[i] >= 0) {
                dp[val] = max(dp[val], dp[val - money[i]] * (1.0 - p[i]));
            }
        }
    }
    int ans = 0;
    for(int val = total; val >= 0; val--) {
        //prob de ser atrapado
        if(1.0 - dp[val] < limit) {
            ans = val;
```

```
            break;
        }
    }
    cout << ans << ln;

    return 0;
}
```

8 Strings

8.1 AhoCorasick

```
//https://github.com/JJuanJr/Notebook-ICPC/blob/main/String/AhoCorasick.cpp
Complejidad:
build(|patrones| * M)
Complejidad:
query(|text|)


const int M = 26;
struct node {
    vector<int> child;
    int p = -1;
    char c = 0;
    int suffixLink = -1, endLink = -1;
    int id = -1;
    //int cnt = 0; Para contar patrones repetidos
    node(int p = -1, char c = 0) : p(p), c(c) {
        child.resize(M, -1);
    }
};

struct AhoCorasick {
    vector<node> t;
    vector<int> lenghts;
    int wordCount = 0;

    AhoCorasick() {
        t.emplace_back();
    }

    void add(const string & s) {
        int u = 0;
        for(char c : s) {
            if(t[u].child[c-'a'] == -1) {
                t[u].child[c-'a'] = t.size();
                t.emplace_back(u, c);
            }
            u = t[u].child[c-'a'];
        }
        t[u].id = wordCount++;
        lenghts.push_back(s.size());
    }

    void link(int u) {
        if(u == 0) {
            t[u].suffixLink = 0;
            t[u].endLink = 0;
            return;
        }
        if(t[u].p == 0) {
            t[u].suffixLink = 0;
            if(t[u].id != -1) t[u].endLink = u;
            else t[u].endLink = t[t[u].suffixLink].endLink;
            return;
        }
        int v = t[t[u].p].suffixLink;
        char c = t[u].c;
        while(true) {
            if(t[v].child[c-'a'] != -1) {
                t[u].suffixLink = t[v].child[c-'a'];
                break;
            }
            if(v == 0) {
                t[u].suffixLink = 0;
                break;
            }
            v = t[v].suffixLink;
        }
        if(t[u].id != -1) t[u].endLink = u;
        else t[u].endLink = t[t[u].suffixLink].endLink;
    }

    void build() {
        queue<int> Q;
        Q.push(0);
```

```
        while(!Q.empty()) {
            int u = Q.front();
            Q.pop();
            link(u);
            for(int v = 0; v < M; ++v)
                if(t[u].child[v] != -1)
                    Q.push(t[u].child[v]);
        }
    }

    int match(const string & text) {
        int u = 0;
        int ans = 0;
        for(int j = 0; j < text.size(); ++j) {
            int i = text[j] - 'a';
            while(true) {
                if(t[u].child[i] != -1) {
                    u = t[u].child[i];
                    break;
                }
                if(u == 0) break;
                u = t[u].suffixLink;
            }
            int v = u;
            while(true) {
                v = t[v].endLink;
                if(v == 0) break;
                ++ans;
                int idx = j + 1 - lenghts[t[v].id];
                // cout << "Found word #" << t[v].id << " at position " <<
                    idx << "\n";
                v = t[v].suffixLink;
            }
        }
        return ans;
    }
};
```

8.2 Expression parsing

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

bool delim(char c) {
    return c == ' ';
}

bool is_op(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/';
}

bool is_unary(char c) {
    return c == '+' || c == '-';
}

int priority (char op) {
    if (op < 0) // unary operator
        return 3;
    if (op == '+' || op == '-')
        return 1;
    if (op == '*' || op == '/')
        return 2;
    return -1;
}

void process_op(stack<int>& st, char op) {
    if (op < 0) {
        int l = st.top();
        st.pop();
        switch (-op) {
        case '+':
            st.push(l);
            break;
        case '-':
            st.push(-l);
            break;
        }
    } else {
        int r = st.top();
        st.pop();
        int l = st.top();
        st.pop();
        switch (op) {
        case '+':
            st.push(l + r);
            break;
        case '-':
            st.push(l - r);
```

```
            break;
        case '*':
            st.push(l * r);
            break;
        case '/':
            st.push(l / r);
            break;
        }
    }
}

int evaluate(string s) {
    stack<int> st;
    stack<char> op;
    bool may_be_unary = true;
    for (int i = 0; i < (int)s.size(); i++) {
        if (delim(s[i]))
            continue;

        if (s[i] == '(') {
            op.push('(');
            may_be_unary = true;
        } else if (s[i] == ')') {
            while (op.top() != '(') {
                process_op(st, op.top());
                op.pop();
            }
            op.pop();
            may_be_unary = false;
        } else if (is_op(s[i])) {
            char cur_op = s[i];
            if (may_be_unary && is_unary(cur_op))
                cur_op = -cur_op;
            while (!op.empty() && (
                        (cur_op >= 0 && priority(op.top()) >=
                            priority(cur_op)) ||
                        (cur_op < 0 && priority(op.top()) >
                            priority(cur_op))
                    )) {
                process_op(st, op.top());
                op.pop();
            }
            op.push(cur_op);
            may_be_unary = true;
        } else {
            int number = 0;
            while (i < (int)s.size() && isalnum(s[i]))
                number = number * 10 + s[i++] - '0';
            --i;
            st.push(number);
            may_be_unary = false;
        }
    }

    while (!op.empty()) {
        process_op(st, op.top());
        op.pop();
    }
    return st.top();
}
int32_t main() {
    cout << evaluate("12-3-(3 - 4)");
    return 0;
}
```

8.3 KMP

```
vector<int> prefix_function(string &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;
}

int kmp(string &s, string &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;
}
```

8.4 Palindrome with hash

```
//usar Hash_str aqui

int n;

//retorna si tiene centro
//si es par el centro es unico
//si es impar el centro es doble EJ: bbaabb

int check(int mid, bool even, hash_str &h1, hash_str &h2) {
    forab(i, 0, n) {
        int left = i - mid + (even ? 1 : 0);
        int right = i + mid;
        if (left >= 0 && right < n) {
            if (is_pal(left, right, h1, h2)) return i;
        }
    }
    return -1;
}

//indicar si se busca una cadena par o impar
string solve(string &s, bool even, hash_str &h1, hash_str &h2) {
    int pos = 0;
    int l = -1, r = n;
    while (l + 1 < r) {
        int m = l + (r - l) / 2;
        int centro = check(m, even, h1, h2);
        if (centro != -1) {
            l = m;
            pos = centro;
        } else {
            r = m;
        }
    }

    int len = even ? 2 * l : 2 * l + 1;
    int start = even ? pos - l + 1 : pos - l;

    start = max(0, start);
    len = max(0, len);
    return s.substr(start, len);
}


int main() {
    string s;
    cin >> s;
    n = sz(s);
    hash_str h1(s);
    reverse(all(s));
    string rev = s;
    hash_str h2(rev);

    reverse(all(s));


    string odd = solve(s, 0, h1, h2);
    string even = solve(s, 1, h1, h2);

    if(sz(odd) < sz(even)) swap(even, odd);
    cout << odd << ln;
    return 0;
}
```

8.5 String Hash 2D

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

const int N = 3e5 + 9;

//Tomado de
//https://github.com/ShahjalalShohag/code-library/
struct Hashing {
    vector<vector<int>> hs;
    vector<int> PWX, PWY;
```

```
    int n, m;
    static const int PX = 3731, PY = 2999, mod = 998244353;
    Hashing() {}
    Hashing(vector<string>& s) {
        n = (int)s.size(), m = (int)s[0].size();
        hs.assign(n + 1, vector<int>(m + 1, 0));
        PWX.assign(n + 1, 1);
        PWY.assign(m + 1, 1);
        for (int i = 0; i < n; i++) PWX[i + 1] = 1LL * PWX[i] * PX % mod;
        for (int i = 0; i < m; i++) PWY[i + 1] = 1LL * PWY[i] * PY % mod;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                hs[i + 1][j + 1] = s[i][j] - 'a' + 1;
            }
        }
        for (int i = 0; i <= n; i++) {
            for (int j = 0; j < m; j++) {
                hs[i][j + 1] = (hs[i][j + 1] + 1LL * hs[i][j] * PY % mod)
                    % mod;
            }
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j <= m; j++) {
                hs[i + 1][j] = (hs[i + 1][j] + 1LL * hs[i][j] * PX % mod)
                    % mod;
            }
        }
    }
    int get_hash(int x1, int y1, int x2, int y2) { // 1-indexed
        assert(1 <= x1 && x1 <= x2 && x2 <= n);
        assert(1 <= y1 && y1 <= y2 && y2 <= m);
        x1--;
        y1--;
        int dx = x2 - x1, dy = y2 - y1;
        return (1LL * (hs[x2][y2] - 1LL * hs[x2][y1] * PWY[dy] % mod +
            mod) % mod -
                1LL * (hs[x1][y2] - 1LL * hs[x1][y1] * PWY[dy] % mod +
                    mod) % mod * PWX[dx] % mod + mod) % mod;
    }
    int get_hash() {
        return get_hash(1, 1, n, m);
    }
};
int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    int t;
    cin >> t;
    while (t--) {
        int n, m;
        cin >> n >> m;
        vector<string> a(n);
        for (int i = 0; i < n; i++) cin >> a[i];
        Hashing H(a);
        int x, y;
        cin >> x >> y;
        vector<string> b(x);
        for (int i = 0; i < x; i++) cin >> b[i];
        auto z = Hashing(b).get_hash();
        int ans = 0;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                if (i + x - 1 <= n && j + y - 1 <= m && H.get_hash(i, j, i
                    + x - 1, j + y - 1) == z) ans++;
            }
        }
        cout << ans << '\n';
    }
    return 0;
}
/*
    dada una matriz de n * m
    busca un patron de x * y
*/
```

8.6 String Hash

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

const ll MOD = 991831889LL;

//9982443539LL
//1000000007LL
//1000000009LL

const ll C = uniform_int_distribution<ll>(0.1 * MOD, 0.9 * MOD)(rng);
```

```
struct hash_str {
    vll h, p;
    hash_str(const string &s) : h(sz(s) + 1), p(sz(s) + 1) {
        p[0] = 1;
        h[0] = 0;
        forn (i, sz(s)) {
            h[i + 1] = (C * h[i] + s[i]) % MOD;
            p[i + 1] = (C * p[i]) % MOD;
        }
    }
    // Returns hash of interval s[a ... b] (where 0 <= a <= b < sz(s))
    ll get(int a, int b) {
        return (h[b + 1] - ((h[a] * p[b - a + 1]) % MOD) + MOD) % MOD;
    }
};


bool is_pal(int l, int r, hash_str &h1, hash_str &h2) {
    int l2 = sz(h1.h) - 2 - r;
    int r2 = sz(h1.h) - 2 - l;
    return (h1.get(l, r) == h2.get(l2, r2));
}

//how many substrings of a given string are palindromes.
ll count_palindromes(string &s, hash_str &h1, hash_str &h2) {
    ll ans = 0;

    forn(i, sz(s)) {
        // Palindromes odd length
        int l = 0, r = min(i + 1, sz(s) - i);
        while (l < r) {
            ll mid = (l + r + 1) / 2;
            if (is_pal(i - mid + 1, i + mid - 1, h1, h2)) {
                l = mid;
            } else {
                r = mid - 1;
            }
        }
        ans += l;

        //Palindromes even length
        l = 0, r = min(i + 1, sz(s) - i - 1);
        while (l < r) {
            ll mid = (l + r + 1) / 2;
            if (is_pal(i - mid + 1, i + mid, h1, h2)) {
                l = mid;
            } else {
                r = mid - 1;
            }
        }
        ans += l;
    }

    return ans;
}

int main() {
    string s;
    cin >> s;

    //normal hash
    hash_str h1(s);

    //reverse hash
    string rev_s = s;
    reverse(all(rev_s));
    hash_str h2(rev_s);

    ll ans = count_palindromes(s, h1, h2);
    cout << ans << ln;
    return 0;
}
```

8.7 Suffix Array

```
typedef long long ll;
typedef vector<int> vi;
typedef vector<bool> vb;
typedef vector<ll> vll;

struct SuffixArray {
    int K = 256; // alphabet size
    int n;
    vi sa, lcp, c, cnt;

    vi build(string& str) {
        string s = str;
```

```
s += "$";
n = sz(s);
sa = vi(n);
c = vi(2 * n);
cnt = vi((max(n, K)));
forab (k, 0, n) {
    cnt[(int)(s[k])]++;
}
forab (k, 1, K) {
    cnt[k] += cnt[k - 1];
}
forab (k, 0, n) {
    sa[--cnt[(int)(s[k])]] = k;
}
c[sa[0]] = 0;
int classes = 1;
forab (k, 1, n) {
    if (s[sa[k]] != s[sa[k - 1]]) {
        classes++;
    }
    c[sa[k]] = classes - 1;
}
vi pn(n);
vi cn(2 * n);

for (int h = 0; (1 << h) < n; h++) {
    forab (k, 0, n) {
        pn[k] = sa[k] - (1 << h);
        if (pn[k] < 0) {
            pn[k] += n;
        }
    }
    fill(all(cnt), 0);
    forab (k, 0, n) {
        cnt[c[pn[k]]]++;
    }
    forab (k, 1, classes) {
        cnt[k] += cnt[k - 1];
    }
    for (int k = n - 1; k >= 0; k--) {
        sa[--cnt[c[pn[k]]]] = pn[k];
    }
    cn[sa[0]] = 0;
    classes = 1;
    forab (k, 1, n) {
        int cur1 = c[sa[k]];
        int cur2 = c[sa[k] + (1 << h)];
        int prev1 = c[sa[k - 1]];
        int prev2 = c[sa[k - 1] + (1 << h)];
        if (cur1 != prev1 || cur2 != prev2) {
            classes++;
        }
        cn[sa[k]] = classes - 1;
    }
    swap(c, cn);// swap c and cn
}
return sa;
}

/*
    * Longest Common Preffix
    * Use Kasai algorithm to build LCP array
    * LCP is an array in which every index
    * tracks how many characters two sorted
    * adjacent suffixes have in common.
    */
void kasai(string& str) {
    string s = str;
    s += "$";
    int n = sz(s);
    lcp = vi(n);
    vi inv(n);
    forab (i, 0, n)
    inv[sa[i]] = i;
    for (int i = 0, k = 0; i < n; i++) {
        if (inv[i] == n - 1) {
            k = 0;
            continue;
        }
        int j = sa[inv[i] + 1];
        while ((i + k < n) && (j + k < n) && s[i + k] == s[j + k])
            k++;
        lcp[inv[i]] = k;
        if (k > 0)
            k--;
    }
}

/*
    * n = s.length();
```

```
    * unique subStrings of s = (n*(n + 1)/2) - ( lcp[i]);
    */
ll uniqueSubStrings(string& str) {
    string s = str;
    build(s);
    kasai(s);
    int n = s.length();
    ll ans = n - sa[0];
    forab (i, 1, lcp.size()) {
        ans += (n - sa[i]) - lcp[i - 1];
    }
    return ans;
}

/*
    * To find the LCS of two Strings
    * Let's combine two strings into one through the symbol "sharp"
        s1#s2
    * identify the suffixes which start in positions wich are inside
        the s1
    * identify the suffixes which start in positions wich are inside
        the s2
    * then let's build SA and LCP of combined string
    *
    * then we need to find two suffixes,
    * one should be inside s1 and other should be inside s2
    *
    * such that the length of their common prefix is a big as possible
    */
//longest common substring
string lcs(string& s1, string& s2) {
    string combined = s1;

    int leftS1 = 0;
    combined += ("#");
    int rightS1 = combined.length() - 1;

    int leftS2 = combined.length();
    combined += (s2);
    int rightS2 = combined.length();

    build(combined);
    kasai(combined);
    int MAX = 0;
    int start = -1;

    forab (i, 0, sa.size() - 1) {
        // if sa[i] inside s1 && sa[i + 1] inside s2
        if (sa[i] >= leftS1 && sa[i] < rightS1 && sa[i + 1] >= leftS2
            && sa[i + 1] < rightS2) {
            if (lcp[i] > MAX) {
                MAX = lcp[i];
                start = i;
            }
            // else sa[i] inside s2 && sa[i + 1] inside s1
        } else if (sa[i + 1] >= leftS1 && sa[i + 1] < rightS1 && sa[i]
            >= leftS2 && sa[i] < rightS2) {
            if (lcp[i] > MAX) {
                MAX = lcp[i];
                start = i;
            }
        }
    }
    if (start == -1) {
        return "";
    } else {
        for (int i = sa[start]; i < sa[start] + MAX; i++) {
            cout << combined[i];
        }
        cout << ln;
        // string lcs = combined.substr(sa[start], sa[start] + MAX);
        return "";
    }
}

int match(string& s, string& w) {
    int n = sz(s);
    int l = 1;
    int r = n;
    int lower = -1;
    while (l <= r) {
        int mid = l + (r - l) / 2;
        if (compare(s, sa[mid], w) >= 0) {
            lower = mid;
            r = mid - 1;
        } else {
            l = mid + 1;
        }
    }
    l = 1;
```

```
        r = n;
        int upper = -1;
        while (l <= r) {
            int mid = l + (r - l) / 2;
            if (compare(s, sa[mid], w) <= 0) {
                upper = mid;
                l = mid + 1;
            } else {
                r = mid - 1;
            }
        }
        upper++;
        if (lower == -1 || compare(s, sa[lower], w) != 0) {
            return -1; // no aparece
        } else {
            //cantidad de veces que aparece
            return upper - lower + 1;
        }
    }

    int compare(string& s, int x, string& w) {
        int n = sz(s);
        int qn = sz(w);

        int i = 0;
        while (i + x < n && i < qn && s[i + x] == w[i])
            i++;
        if (i >= qn) {
            return 0;
        } else if (i + x >= n) {
            return -1;
        }
        return (int)s[i + x] - (int)w[i];
    }
};

int main() {
    string s, t;
    cin >> s >> t;
    SuffixArray sf;
    sf.lcs(s, t);
    return 0;
}
```

8.8 Trie

```
int maxlen;
int cnt;
struct Node {
    int cnt;
    Node* child[26];
};

struct Trie {
    Node* root;
    Trie() {
        root = new Node();
    }

    void insert(const string &s) {
        Node* cur = root;
        int len = 0;
        forn(i, sz(s)) {
            ++len;
            int c = s[i] - 'a';
            if (cur->child[c] == NULL) {
                cur->child[c] = new Node();
            }
            cur->child[c]->cnt++;
            cur = cur->child[c];
        }
    }

    void dfs(Node* u, int dep) {
        if(u->cnt >=3 && dep > maxlen) {
            maxlen = dep;
            cnt = u->cnt;
        }
        forn(v, 26)
        if(u->child[v]!=NULL)
            dfs(u->child[v], dep + 1);
    }

    pair<int, int> query(const string &s) {
        Node* cur = root;
        forn (i, sz(s)) {
            int c = s[i] - 'a';
            if (cur->child[c] == NULL) {
```

```
            return {i, cur->cnt};
        }
        cur = cur->child[c];
    }
    return {sz(s), cur->cnt};
}

};
```

8.9 Z function

```
Dada una string s, devuelve un vector Z
donde Z[i] representa el prefijo
de mayor longitud de s, que tambien
es prefijo del sufijo de s que inicia
en i. 01234567
Ejemplo:
aabzaaba "aab" es un prefijo de s
y "aaba" es un sufijo de s, Z[4] = 3.

Otra definicion: Dado un string s retorna un vector z
donde z[i] es igual al mayor
numero de caracteres desde s[i]
que coinciden con los caracteres desde s[0]

Complejidad: O(|n|)
vector<int> z_function (string &s) {
    int n = s.size();
    vector<int> z(n);
    for (int i = 1, x = 0, y = 0; i < n; i++) {
        z[i] = max(0, min(z[i-x], y-i+1));
        while (i+z[i] < n && s[z[i]] == s[i+z[i]]) {
            x = i, y = i+z[i], z[i]++;
        }
    }
    return z;
}
```

8.10 manacher

```
Devuelve un vector p donde,
para cada i, p[i] es igual al largo del palindromo mas
largo con centro en i.
Tener en cuenta que el string debe tener el siguiente formato:
%#s[0]#s[1]#...#s[n-1]#$ (s es el string original y n es el largo del
    string)

Complejidad: O(|n|)
vector<int> manacher(string s) {
    int n = s.size();
    vector<int> p(n, 0);
    int c = 0, r = 0;
    for (int i = 1; i < n-1; i++) {
        int j = c - (i-c) ;
        if (r > i) p[i] = min(r-i , p[j]);
        while (s[i+1+p[i]] == s[i-1-p[i]])
            p[i]++;
        if (i+p[i] > r) {
            c = i;
            r = i+p[i];
        }
    }
    return p;
}

Recibe el string original.

vector<int> impar(string s){
    int n = sz(s);
    vector<int> d1 (n);
    int l=0, r=-1;
    for (int i=0; i<n; ++i) {
        int k = i>r ? 1 : min(d1[l+r-i], r-i+1);
        while (i+k < n && i-k >= 0 && s[i+k] == s[i-k])
            ++k;
        d1[i] = k;
        if (i+k-1 > r)
            l = i-k+1, r = i+k-1;
    }
    return d1;
}

vector<int> par(string s){
    int n = sz(s);
    vector<int> d2 (n);
    l=0, r=-1;
```

```
    for (int i=0; i<n; ++i) {
        int k = i>r ? 0 : min (d2[l+r-i+1], r-i+1);
        while (i+k < n && i-k-1 >= 0 && s[i+k] == s[i-k-1])
            ++k;
        d2[i] = k;
        if (i+k-1 > r)
            l = i-k, r = i+k-1;
    }
    return d2;
}
```

8.11 minimum expression

```
Dado un string s devuelve el indice donde comienza
la rotacin lexicograficamente menor de s.

Complejidad:
O(|n|)
//Factorizacion de lyndon
int minimum_expression(string s) {
    s = s+s; // si no se concatena devuelve el indice del sufijo menor
    int len = s.size(), i = 0, j = 1, k = 0;
    while (i+k < len && j+k < len) {
        if (s[i+k] == s[j+k]) k++;
        else if (s[i+k] > s[j+k]) i = i+k+1, k = 0; // <- here
        // cambiar por < para maximum
        else j = j+k+1, k = 0;
        if (i == j) j++;
    }
    return min(i, j);
}
```

8.12 palindrome subarrays

```
// how many subarrays s[l,r]
// exists such that you
//can reorder its characters
// to get a palindrome.
const int N = 1e6 + 5;
const int MASK = (1<<26);
int F[MASK];

int main() {
  int n;
  cin >> n;
  string s;
  cin >> s;
  ll ans = 0;
  int cur_mask = 0;
  F[0]++;
  forn(r, n) {
    cur_mask ^= 1 <<(s[r] - 'a');
    ans+=F[cur_mask];//caso par
    forn(y, 26) {
      ans+=F[cur_mask ^ (1 << y)];//caso impar
    }
    F[cur_mask]++;
  }
  cout << ans << ln;
  return 0;
}
```

8.13 palindrome substr range

```
  //numero de substrings
  //que son palindromos
  //en el rango [l, r]
  string s; cin >> s;
  int n = sz(s);
  const int MAXN = 5000 + 10;
  vector<vb> isPal(MAXN, vb(MAXN));
  vector<vi> dp(MAXN, vi(MAXN));
  forn(i, n){
    isPal[i][i] = 1;
    dp[i][i] = 1;
    isPal[i + 1][i] = 1;
  }
  forab(len, 2, n + 1){
    forn(i, n - len + 1){
      isPal[i][i + len - 1] = isPal[i + 1][i + len - 2] & s[i] == s[i +
          len - 1];
      dp[i][i + len - 1] = dp[i][i + len - 2] +
```

```
      dp[i + 1][i + len - 1] - dp[i + 1][i + len - 2] +
       isPal[i][i + len - 1];
    }
  }
  int q; cin >> q;
  while(q--){
    int l, r; cin >> l >> r;
    l--; r--;
    cout << dp[l][r] << ln;
  }
```

8.14 psa

```
vector<vi> prefix(string &s) {
  vector<vi> psa(26, vi(sz(s) + 1));
  forab(i, 1, sz(s) + 1) {
    forn(j, 26) {
      psa[j][i] = psa[j][i - 1];
    }
    char c = s[i - 1];
    psa[c - 'a'][i]++;
  }
  return psa;
}
//freq de char c en el rango [l, r]
//int query = psa[c -'a'][r] - psa[c-'a'][l];
```

8.15 string utils

```
//String completa a miniscula
transform(all(in), in.begin(), ::tolower);


//A es subsecuencia de B?
bool is_subsecuence(vi &a, vi &b){
  int at = 0;
  for(auto i: b){
    if(i == a[at]) at++;
    if(at == sz(a)) return 1;
  }
  return 0;
}
```

9 utils

9.1 Bits

```
Operaciones a nivel de bits.

n & 1          -> Verifica si n es impar o no
n & (1<<k)     -> Verifica si el k-esimo bit esta encendido o no
n | (1<<k)     -> Enciende el k-esimo bit
n & ~(1<<k)    -> Apaga el k-esimo bit
n ^ (1<<k)     -> Invierte el k-esimo bit
~n                -> Invierte todos los bits
n & -n         -> Devuelve el bit encendido mas a la derecha
~n & (n+1)     -> Devuelve el bit apagado mas a la derecha
n | (n+1)      -> Enciende el bit apagado mas a la derecha
n & (n-1)      -> Apaga el bit encendido mas a la derecha

__builtin_popcountll(x) -> Cuantos bits tiene encendidos


//sumatoria de bits encendidos en el rango [l, r]
//sumatoria de log2 en el rango [l, r]

ll count_bits(ll n){
    ll ans = 0;
    // se trabaja con cada bit independientemente
    forn(i, 61){
        ll total = (1LL << (i + 1));
        ll pattern = (n / total);
        ans += (pattern * total / 2);
        ll remain = n % total;
        if(remain - total / 2 > 0) ans += (remain - total / 2);

    }
    return ans;
}

ll count_logs(ll n){
    if(n == 0) return 0;
    ll cur = 0;
}
```

```
    int i = 0;
    ll ans = 0;
    while(cur < n){
        cur += (1LL << i);
        ans += i * (1LL << i);
        i++;
    }
    i--;
    ll res = abs(cur - n + 1) * i;
    return ans - res;
}

    ll l, r;
    cin >> l >> r;
    ll sum_bits = count_bits(r + 1) - count_bits(l);
    ll sum_logs = count_logs(r + 1) - count_logs(l);
    cout << sum_logs + sum_bits << ln;
```

9.2 debug

```
#include "debug.h"

string __print(string s){
       return '"' + s + '"';
}

string __print(char c){
       string ret = "'";
       ret += c;
       ret += "'";
       return ret;
}

string __print(bool c){
       return (c ? "true" : "false");
}

string __print(int a){
       return to_string(a);
}

string __print(long a){
       return to_string(a);
}

string __print(long long a){
       return to_string(a);
}


string __print(float a){
       return to_string(a);
}

string __print(double a){
       return to_string(a);
}

string __print(long double a){
       return to_string(a);
}

template<typename A, typename B>
string __print(pair<A, B> p){
       return "(" + __print(p.first) + ", " + __print(p.second) + ")";
}

template<typename T, size_t Z>
string __print(array<T, Z> a){
       bool sep = false;
       string ret = "{";
       for(const T &x : a){
              if(sep) ret += ", ";
              sep = true;
              ret += __print(x);
       }
       ret += "}";
       return ret;
}

template<typename T>
string __print(vector<T> v){
       bool sep = false;
       string ret = "{";
       for(const T &x : v){
              if(sep) ret += ", ";
              sep = true;
              ret += __print(x);
```

```
	}
	ret += "}";
	return ret;
}

template<typename T>
string __print(stack<T> s){
	bool sep = false;
	string ret = "{";
	while(!s.empty()){
		if(sep) ret += ", ";
		sep = true;
		ret += __print(s.top()); s.pop();
	}
	ret += "}";
	return ret;
}

template<typename T>
string __print(queue<T> q){
	bool sep = false;
	string ret = "{";
	while(!q.empty()){
		if(sep) ret += ", ";
		sep = true;
		ret += __print(q.front()); q.pop();
	}
	ret += "}";
	return ret;
}

template<typename T>
string __print(priority_queue<T> pq){
	bool sep = false;
	string ret = "{";
	while(!pq.empty()){
		if(sep) ret += ", ";
		sep = true;
		ret += __print(pq.top()); pq.pop();
	}
	ret += "}";
	return ret;
}

template<typename T>
string __print(priority_queue<T, vector<T>, greater<T>> pq){
	bool sep = false;
	string ret = "{";
	while(!pq.empty()){
		if(sep) ret += ", ";
		sep = true;
		ret += __print(pq.top()); pq.pop();
	}
	ret += "}";
	return ret;
}

template<typename T>
string __print(deque<T> v){
	bool sep = false;
	string ret = "{";
	for(const T &x : v){
		if(sep) ret += ", ";
		sep = true;
		ret += __print(x);
	}
	ret += "}";
	return ret;
}

template<typename T>
string __print(set<T> v){
	bool sep = false;
	string ret = "{";
	for(const T &x : v){
		if(sep) ret += ", ";
		sep = true;
		ret += __print(x);
	}
	ret += "}";
	return ret;
}

template<typename T>
string __print(multiset<T> v){
	bool sep = false;
	string ret = "{";
	for(const T &x : v){
		if(sep) ret += ", ";
		sep = true;
```

```
                ret += __print(x);
        }
        ret += "}";
        return ret;
}

template<typename T>
string __print(unordered_set<T> v){
        bool sep = false;
        string ret = "{";
        for(const T &x : v){
                if(sep) ret += ", ";
                sep = true;
                ret += __print(x);
        }
        ret += "}";
        return ret;
}

template<typename F, typename S>
string __print(map<F, S> v){
        bool sep = false;
        string ret = "{";
        for(const auto& x : v){
                if(sep) ret += ", ";
                sep = true;
                ret += __print(x);
        }
        ret += "}";
        return ret;
}

template<typename F, typename S>
string __print(multimap<F, S> v){
        bool sep = false;
        string ret = "{";
        for(const auto& x : v){
                if(sep) ret += ", ";
                sep = true;
                ret += __print(x);
        }
        ret += "}";
        return ret;
}
```

```
void debug() { cerr << '\n'; }


template <typename Head, typename... Tail>
void debug(Head H, Tail... T){
        cerr << ' ' << __print(H);
        debug(T...);
}

#define dbg(...) cerr << "[" << #__VA_ARGS__ << "] => ",
    debug(__VA_ARGS__)
```

9.3 helps

```
typedef pair<int, int> pii;

bool cmp(pii &a, pii &b) {
  if(a.second == b.second) return a.first < b.first;
  return a.second > b.second;
}
//for set
set<pii, decltype(&cmp)> q(&cmp);


//funciones dentro del main
function<ll(int, int, int)> suma = [&](int a, int b, int c) {
  return ll(a + b + c);
};
cout << suma(11, 11, 1) << ln;

//input / output
string x;
getline(cin, x);//lee linea completa

//imprime long double con 6 decimales
printf("%.6LF\n", value);//long double

//impimir string con printf
string aux = "texto";
printf("%s\n", aux.c_str());
```

```
//compresion de coordenadas
sort(all(ax));
ax.erase(unique(all(ax)), ax.end());
```

9.4 main

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

#define ln '\n'
#define all(x) x.begin(), x.end()
#define forn(i, n) for(int i = 0; i < n; i++)
#define forab(i, a, b) for (int i = a; i < b; i++)
#define pb push_back
#define sz(x) int(x.size())
#define rforn(i, n) for (int i = n-1; i >= 0; --i)
#define form(i, n, m, x) for (int i = n; i < m; i += x)
#define rform(i, n, m, x) for (int i = n; i >= m; i -= x)

#ifdef LOCAL
#include "debug.h"
#else
#define dbg(...)
#endif

typedef long long ll;
typedef vector<int> vi;
typedef vector<bool> vb;
typedef vector<ll> vll;

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    #ifdef LOCAL
        freopen("in.txt", "r", stdin);
        //freopen("out.txt", "w", stdout);
    #endif

    return 0;
}
```

9.5 main2

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

using namespace std;

#define forn(i,n) for (int i = 0; i < n; i++)
#define rforn(i,n) for (int i = n-1; i >= 0; i--)
#define fornP(i,a,b) for (int i = a; i < b; i++)
#define endl '\n'
#define ff first
#define ss second
#define pb push_back
#define ll long long
#define all(x) x.begin(),x.end()
#define sz(n) (int)n.size()
#define read(x) for(auto &el : x) cin >> el;
#define show(x) for (auto &i: x) cout << i << " "; cout << endl;
#define FAST ios_base::sync_with_stdio(0); cin.tie(0);cout.tie(0);
//
const int mod = 1e9 + 7;

void solve(){
    // code here
}
int main() {

    FAST
    int test = 1;
    //cin >> test;
    while (test--) solve();

    return 0;
}
```

9.6 python

```
from functools import cmp_to_key

def next_int(): return int(input())
def read_array(): return map(int, input().split())
def read_list(): return list(map(int, input().split()))
```

```python
def comparator(a, b):
  if a < b:
    return -1
  else:
    return 1


t = next_int()
for _ in range(t):
  n = next_int()
  a = read_list()
  a = sorted(a, key=cmp_to_key(comparator))

  i = 0
  j = n - 1

  #while i <= j:
  #if i == j:
  #for i in range(n)
  print(int(ans))

#vector
arr = []
a.append(10)

#deque
from collections import deque
q = deque()
q.append(("a", 8))
q.popleft()

#set
st = set()
st.add(1)

#mapa
mp = dict()
mp["str"] = "x"
mp["k"] = 21
```

9.7 template

```cpp
//g++ -std=c++17 -Wall -Wextra -O2 -DLOCAL main.cpp -o main && ./main <
    in.txt
#include <bits/stdc++.h>
using namespace std;

void _print() {
  cerr << "]" << endl;
}
template<typename T, typename... V>
void _print(T t, V... v) {
  cerr << t;
  if (sizeof...(v)) cerr << ", ";
  _print(v...);
}

#ifdef LOCAL
#define dbg(x...) cerr << "[" << #x << "]: ["; _print(x)
#else
#define dbg(x...)
#define endl '\n'
#endif

#define pb push_back
#define sz(x) int(x.size())
#define all(x) x.begin(), x.end()
#define forn(i, n) for (int i = 0; i < n; ++i)
#define rforn(i, n) for (int i = n-1; i >= 0; --i)
#define forab(i, a, b) for (int i = a; i < b; ++i)
#define form(i, n, m, x) for (int i = n; i < m; i += x)
#define rform(i, n, m, x) for (int i = n; i >= m; i -= x)

typedef long long ll;
typedef vector<int> vi;
typedef vector<bool> vb;
typedef vector<ll> vll;
typedef pair<int, int> ii;
int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);
  #ifdef LOCAL
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
  #endif LOCAL
```

```
//codes here
return 0;
}
```