Competitive Programming Notebook

Miguel Nogueira

Contents

1 Dynamic programming

1.1 Optimal-selection

```
/*
    Optimal Selection
    Tens n escolhas pra fazer em k intervalos de tempo,
    escolhe o melhor subconjunto tal que alguma heuristica
    eh maximizada ao longo de todos os timestamps
*/

int optimal_selection(int n, int k, int w[][]) {
    int f[(1 << n)][n + 1];
    for (int i = 0; i < (1 << n); i++) {
        for (int j = 0; j < n; j++) {
            f[i][j] = 0;
        }
    }
    for (int i = 0; i < k; i++) {
        f[(1 << i)][0] = w[i][0];
    }
    for (int day = 1; day < n; day++) {
        for (int mask = 0; mask < 8; mask++) {
            f[mask][day] = f[mask][day - 1];
            for (int y = 0; y < k; y++) {
                if (mask & (1 << y)) {
                    f[mask][day] = max(f[mask][day], f[mask^(1 << k)][day - 1] + w[k][day]);
                }
            }
        }
    }
    return f[(1 << n) - 1][n - 1];
}
```

1.2 Longest-increasing-subsequence

```
/*
    Longest Increasing Subsequence
    Encontra o tamanho e recupera uma LIS de um vetor `a`
    Complexidade: O(n log n)
*/

vector<int> lis(vector<int> const& a) {
    int n = a.size();
    vector<int> d(n+1, INF), pos(n+1, -1), prev(n, -1);
    d[0] = -INF;

    for (int i = 0; i < n; i++) {
        int l = lower_bound(d.begin(), d.end(), a[i]) - d.begin();
        if (a[i] < d[l]) {
            d[l] = a[i];
            pos[l] = i;
            prev[i] = pos[l-1];
        }
    }

    int len = 0;
    while (d[len] < INF) {
        len++;
    }
    len--;

    vector<int> result;
    int curr_pos = pos[len];
    while (curr_pos != -1) {
        result.push_back(a[curr_pos]);
        curr_pos = prev[curr_pos];
    }
    reverse(result.begin(), result.end());

    return result;
}
```

1.3 Digit

```
/*
Digit DP
Calcula a soma dos digitos de todos os numeros entre 0 e `number`
para intervalo [a, b] -> solve(b) - solve(a - 1)
*/

const int MAX_DIGITS = 10;
ll dp[MAX_DIGITS][180][2];
vector<int> number;
ll f(int pos, ll sum, int smaller) {
    if (pos == number.size()) return sum;
    ll &ans = dp[pos][sum][smaller];
    if (~ans) return ans;
    ans = 0;
    for (int i=0; i <= (smaller ? 9: number[pos]); i++) {
        bool smaller_now = (smaller || i < number[pos]);
        ans += f(pos + 1, sum + i, smaller_now);
    }
    return dp[pos][sum][smaller] = ans;
}

/*

    Se nao tiver inversa:
    const int MAX_DIGITS = 20;
    const int MAX_K = 20;
    ll dp[MAX_DIGITS][MAX_K][2][2]; //

    int d, k;
    vector<int> number_a, number_b;

    ll solve(int pos, int cnt, bool smaller_than_b, bool greater_than_a){
        if(pos == number_a.size()) return (cnt == k);
        ll &ans = dp[pos][cnt][smaller_than_b][greater_than_a];
        if(~ans) return ans;
        ans = 0;
        for(int i = (greater_than_a ? 0 : number_a[pos]); i <= (smaller_than_b ? 9 : number_b[pos]); i++){
            bool is_smaller_now = (smaller_than_b || (i < number_b[pos]));
            bool is_greater_now = (greater_than_a || (i > number_a[pos]));
            int new_cnt = cnt + (i == d);
            ans += solve(pos+1, new_cnt, is_smaller_now, is_greater_now);
        }
        return ans;
    }

*/

vector<int> ntovec(int num)  {
    if (num == 0) return {0};
    vector<int> v;
    for (; num >0; num /= 10) v.push_back(num % 10);
    reverse(begin(v), end(v));
    return v;
}
```

```
ll solve(int n) {
    if (n < 0) return 0;
    number = ntovec(n);
    memset(dp, -1, sizeof dp);
    return f(0, 0, false);
}

ll ans(int a, int b) {
    return solve(b) - solve(a - 1);
}
```

1.4 Knapsack

```
/*
    Knapsack Problem -
    Given a set of items with value i and cost j, and you have limited budget
    find the subset of items you can take where total value is maximal
    Variations covered:
        - 0/1 Knapsack - Only one copie of each item can be taken
        - Bounded Knapsack - Each item has a number k[i] of copies

    If item retrieval is unecessary prefer 1D knapsack
*/

/*
    0/1 Knapsack - One copy of each item
*/

int f[n + 1][cap + 1], weight[n], value[n];

// Time: O(nW)
// Space : O(nW)
int knapsack_2D() {
    for (int i = 1; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            f[i][w] = f[i - 1][w];
            if (w >= weight[i - 1]) {
                f[i][w] = max(f[i][w], f[i - 1][w - weight[i - 1]] + value[i - 1]);
            }
        }
    }
    return f[n][W];
}

// Time O(nW)
// Space O(w)
int knapsack_1D() {
    for (int i = 0; i < n; i++) {
        for (int w = W; w >= weight[i]; i--) {
            f[w] = max(f[w], f[w - weight[i]] + value[i]);
        }
    }
    return f[W];
}

/*
Unbounded Knapsack - Infinite copies of each item
*/

// Time O(nW)
// Space O(nw)
int unbounded_knapsack_2D() {
    for (int i = 1; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            f[i][w] = f[i - 1][w];  // Not taking item
            if (w >= weight[i - 1]) {
                f[i][w] = max(f[i][w], f[i][w - weight[i - 1]] + value[i - 1]);
            }
        }
    }
    return f[n][W];
}

// Time O(nW)
// Space O(W)
int unbounded_knapsack_1D() {
    for (int i = 0; i < n; i++) {
        for (int w = weight[i]; w <= W; w++) {  // Forward loop allows reuse
            f[w] = max(f[w], f[w - weight[i]] + value[i]);
        }
    }
    return f[W];
}

/*
    Bounded knapsack - Bounded number of copies of each item
*/

// Time: O(nWk) suitable for small k
// Space: O(nW)
int bounded_knapsack_2D() {
    for (int i = 1; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            f[i][w] = f[i - 1][w];  // Not taking item
            for (int k = 1; k <= count[i - 1] && k * weight[i - 1] <= w; k++) {
                f[i][w] = max(f[i][w], f[i - 1][w - k * weight[i - 1]] + k * value[i - 1]);
            }
        }
    }
    return f[n][W];
}

// Time: O(nW) any k
// Space: O(W)
int bounded_knapsack_1D() {
    for (int i = 0; i < n; i++) {
        for (int k = 1; count[i] > 0; k *= 2) {
            int take = min(k, count[i]);
            count[i] -= take;
            for (int w = W; w >= take * weight[i]; w--) {
                f[w] = max(f[w], f[w - take * weight[i]] + take * value[i]);
            }
        }
    }
    return f[W];
}
```

1.5 Sos

```
// F[mask] = sum of values of all submasks of mask
for (int i = 0; i < n; i++) {
    for (int mask = 0; mask < (1 << n); mask++) {
        if (mask & (1 << i)) {
            dp[mask] += dp[mask ^ (1 << i)];
        }
    }
}
```

2 General

2.1 Progressions

```cpp
ll nthTermAP(ll a, ll d, int n) {
    return a + (n - 1) * d;
}

ll sumAP(ll a, ll d, int n) {
    return (n / 2LL) * (2LL * a + (n - 1) * d);
}

ll nthTermGP(ll a, ll r, int n) {
    return a * pow(r, n - 1);
}

ll sumGP(ll a, ll r, int n) {
    if (r == 1) return a * n; // Special case for r=1
    return a * (1 - pow(r, n)) / (1 - r);
}
```

2.2 Chminmax

```cpp
template <class T, class U>
inline bool chmax(T &a, const U &b) {
    return a < b ? (a = b, true) : false;
}

template <class T, class U>
inline bool chmin(T &a, const U &b) {
    return a > b ? (a = b, true) : false;
}
```

2.3 Mo

```cpp
/*
    Mo's algorithm
    Answer OFFLINE range queries in O((n + q) sqrt (n))
*/

int len;  // roughly sqrt n
struct Query {
    int l, r, idx;
    bool operator<(const Query& other) const {
        int block_a = l / len, block_b = other.l / len;
        if (block_a != block_b)
            return block_a < block_b;
        return (block_a & 1) ? (r > other.r) : (r < other.r);
    }
};

int get_ans() {
}
void add(int idx) {
}
void remove(int idx) {
}

template <typename T>
void mo(vector<Query> queries) {
    sort(all(queries));
    ans.assign(queries.size(), 0);
    int cur_l = 0, cur_r = -1;
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        ans[q.idx] = get_ans();  // get answer
    }
}
```

2.4 Two Sat

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

/*
2-satisfiability testado em (https://judge.yosupo.jp/problem/two_sat)
N variÃąveis e M clausulas de (and, or, xor, nor,etc) para serem satisfeitas
ao mesmo tempo, determina se existe e dÃą uma resposta.
O(N + M) com Kosaraju
*/

struct two_sat {
    int n;
    vector<vector<int>> graph, reverse_graph;
    vector<int> components, topo_order, answer;
    vector<bool> visited;

    two_sat() {}
    two_sat(int _n): n(_n) {
        graph.assign(2 * n, {});
        reverse_graph.assign(2 * n, {});
        components.assign(2 * n, 0);
        visited.assign(2 * n, false);
        answer.assign(n, 0);
    }

    inline int var(int i, bool val) {
        return 2 * i + (val ? 0 : 1);
    }

    void add_edge(int u, int v) {
        graph[u].push_back(v);
        reverse_graph[v].push_back(u);
    }

    void add_or(int i, bool f, int j, bool g) {
        add_edge(var(i, !f), var(j, g));
        add_edge(var(j, !g), var(i, f));
    }

    // Ambos true
    void add_and(int i, bool f, int j, bool g) {
        add_edge(var(i, !f), var(i, f));
        add_edge(var(j, !g), var(j, g));
    }

    // Exatamente um deve ser true
    void add_xor(int i, bool f, int j, bool g) {
        add_or(i, f, j, g);
        add_or(i, !f, j, !g);
    }

    // Ambos falsos
```

```cpp
    void add_nor(int i, bool f, int j, bool g) {
        add_edge(var(i, f), var(i, !f));
        add_edge(var(j, g), var(j, !g));
    }


    void dfs(int u) {
        visited[u] = true;
        for (int v: graph[u]) if (!visited[v]) dfs(v)
;
        topo_order.push_back(u);
    }

    void scc(int u, int id) {
        visited[u] = true;
        components[u] = id;
        for (int v: reverse_graph[u]) if (!visited[v
]) scc(v, id);
    }

    bool satisfiable() {
        fill(visited.begin(), visited.end(), false);
        topo_order.clear();
        for (int i = 0; i < 2 * n; i++)
            if (!visited[i]) dfs(i);

        fill(visited.begin(), visited.end(), false);
        reverse(topo_order.begin(), topo_order.end())
;

        int id = 0;
        for (int v: topo_order)
            if (!visited[v]) scc(v, id++);

        for (int i = 0; i < n; i++) {
            if (components[var(i, 0)] == components[
var(i, 1)])
                return false;
            answer[i] = components[var(i, 0)] <
components[var(i, 1)];
        }
        return true;
    }
};
```

2.5 Set-xor-m

```cpp
/*
    Set Xor Min/Max

    C = maior valor
    insere um numero em log(C)
    faz a query de um numero descendo na trie em log(
C)

    pra pegar o menor/maior xor desse numero com os
    outros em log
*/

const int mxbit = 32;
const int mxn = 2e5 + 5;

int trie[mxbit * mxn][2], z = 1;
memset(trie[0], -1, sizeof trie);

auto insert = [&] (int x, int msb) {
    bool val; int cur = 0;
    for (int b = msb; b >= 0; b--) {
        val = x & (1 << b);
        if (trie[cur][val] == -1) {
            memset(trie[z], -1, sizeof trie[z]);
            trie[cur][val] = z++;
        }
        cur = trie[cur][val];
    }
};

auto query = [&] (int x, int msb)->int {
    bool val; int cur = 0,  res = 0;
    for (int b = msb; b >= 0; b--) {
        val = x & (1 << b);
        // troca os dois ifs pra fazer xor min
        if (trie[cur][!val] != -1) cur = trie[cur][!
val], res |= (1 << b);
        else if (trie[cur][val] != -1) cur = trie[cur
][val];
    }
    return res;
};
```

2.6 Gray Code

```cpp
// Generate gray code sequence for n bits
for (int i = 0; i < (1 << n); i++) {
    int gray = i ^ (i >> 1);
    // Process gray code
}
```

2.7 Rng

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

int uniform(int l, int r){
    uniform_int_distribution<int> uid(l, r);
    return uid(rng);
}
```

3 Geometry

3.1 Convex-hull

```cpp
struct Point {
    ll x, y;
    Point(ll x = 0, ll y = 0) : x(x), y(y) {}
    Point operator+(const Point& a) const { return
    Point(x + a.x, y + a.y); }
    Point operator-(const Point& a) const { return
    Point(x - a.x, y - a.y); }
    ll operator*(const Point& a) const { return (x *
    a.x + y * a.y); }  // DOT product  // norm //
    lenght^2 // inner
    ll operator%(const Point& a) const { return (x *
    a.y - y * a.x); }  // Cross // Vector product
    Point operator*(ll c) const { return Point(x * c,
     y * c); }
    Point operator/(ll c) const { return Point(x / c,
     y / c); }

    /*******************
    // FOR DOUBLE POINT //
    const ld EPS = 1e-9;
    bool eq(ld a, ld b){ return abs(a-b) < EPS; } //
    ==
    bool lt(ld a, ld b){ return a + EPS < b;    } //
    <
    bool gt(ld a, ld b){ return a > b + EPS;    } //
    >
    bool le(ld a, ld b){ return a < b + EPS;    } //
    <=
    bool ge(ld a, ld b){ return a + EPS > b;    } //
    >=
    bool operator==(const PT&a) const{ return eq(x, a
    .x) && eq(y, a.y); }               // for double
    point
```

```cpp
    bool operator< (const PT&a) const{ return eq(x, a.x) ? lt(y, a.y) : lt(x, a.x); }   // for double point
    bool operator<<(PT&a){ PT&p=*this; return eq(p%a, 0) ? lt(p*p, a*a) : lt(p%a, 0);  } //angle(this) < angle(a)
    //Change LL to LD and uncomment this
    //Also, consider replacing comparisons with these functions
    *********************/

    bool operator==(const Point& a) const { return x == a.x && y == a.y; }
    bool operator<(const Point& a) const { return x != a.x ? x < a.x : y < a.y; }
    bool operator<<(const Point& a) const {
        Point p = *this;
        return (p % a == 0) ? (p * p < a * a) : (p % a < 0);
    }  // angle(p) < angle(a)
};

vector<Point> ch(vector<Point> pts, bool sorted = false) {
    if (!sorted) sort(begin(pts), end(pts));
    pts.resize(unique(begin(pts), end(pts)) - begin(pts));
    if (pts.size() <= 1) return pts;
    int s = 0, n = pts.size();
    vector<Point> h(2 * n + 1);
    for (int i = 0; i < n; h[s++] = pts[i++])
        while (s > 1 && (pts[i] - h[s - 2]) % (h[s - 1] - h[s - 2]) > 0)
            s--;
    for (int i = n - 2, t = s; ~i; h[s++] = pts[i--])
        while (s > t && (pts[i] - h[s - 2]) % (h[s - 1] - h[s - 2]) > 0)
            s--;
    h.resize(s - 1);
    return h;
}

/* Checks if a point is inside the convex hull: O(log(n))*/

bool inside_triangle(Point a, Point b, Point c, Point point) {
    ll s1 = abs((b - a) % (c - b));
    ll area1 = abs((point - a) % (point - b));
    ll area2 = abs((point - b) % (point - c));
    ll area3 = abs((point - c) % (point - a));
    ll s2 = area1 + area2 + area3;
    return s1 == s2;
}
bool strictly_inside_triangle(Point a, Point b, Point c, Point point) {
    ll area = abs((b - a) % (c - a));
    ll area1 = abs((point - a) % (point - b));
    ll area2 = abs((point - b) % (point - c));
    ll area3 = abs((point - c) % (point - a));
    return area1 != 0 && area2 != 0 && area3 != 0 && area1 + area2 + area3 == area;
}

bool is_inside(vector<Point>& hull, Point p) {
    int n = hull.size();
    if (n == 1) return (hull.front() == p);

    int l = 1, r = n - 1;
    while (abs(r - l) > 1) {
        int mid = (r + l) / 2;
        Point to_mid = hull[mid] - hull[0];
        Point to_p = p - hull[0];
        if ((to_p % (to_mid)) < 0)
            r = mid;
        else
            l = mid;
    }
    return strictly_inside_triangle(hull[0], hull[l], hull[r], p);
}

void solve() {
    int n;
    cin >> n;
    vector<Point> a(n);
    for (int i = 0; i < n; i++) {
        int x, y;
        cin >> x >> y;
        a[i] = {x, y};
    }
    vector<Point> ans = ch(a);
    cout << ans.size() << '\n';
    for (auto [x, y] : ans) {
        cout << x << ' ' << y << '\n';
    }
}
```

3.2 General

```cpp

ld dist  (Point a, Point b){ return sqrtl((a-b)*(a-b)); }                 // distance from A to B
ld angle (Point a, Point b){ return acos((a*b) / sqrtl(a*a) / sqrtl(b*b)); } //Angle between A and B
Point rotate(Point p, double ang){ return Point(p.x*cos(ang) - p.y*sin(ang), p.x*sin(ang) + p.y*cos(ang)); } //Left rotation. Angle in radian

ll Area(vector<Point>& p){
  ll area = 0;
  for(int i=2; i < p.size(); i++)
    area += (p[i]-p[0]) % (p[i-1]-p[0]);
  return abs(area) / 2LL;
}

// Intersecao entre duas retas definidas por a1 + td1 e a2 + td2
// se retas forem paralelas d1 % d2 = 0
Point intersect(Point a1, Point d1, Point a2, Point d2){
  return a1 + d1 * (((a2 - a1)%d2) / (d1%d2));
}

ld dist_pt_line(Point a, Point l1, Point l2){
    return abs( ((a-l1) % (l2-l1)) / dist(l1, l2)  );
}

ld dist_pt_segm(Point a, Point s1, Point s2){
  if(s1 == s2) return dist(s1, a);

  Point d = s2 - s1;
  ld t = max(0.0L, min(1.0L, ((a-s1)*d) / sqrtl(d*d)));

  return dist(a, s1+(d*t));
}
```

4 Number theory

4.1 Millerrho

```cpp
// O(1)
```

```cpp
ll fmul(ll a, ll b, ll M) {
    ll k = a * b - M * (long long)(1.L / M * a * b);
    return k + M * (k < 0) - M * (k >= (ll)M);
}

// O(log(e))
ll fexp(ll b, ll e, ll MOD) {
    ll ans = 1;
    while (e) {
        if (e & 1) ans = fmul(ans, b, MOD);
        b = fmul(b, b, MOD);
        e >>= 1;
    }
    return ans;
}

// ~ O(log n)
bool isPrime(ll n) {
    if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
    ll bases[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
       s = __builtin_ctzll(n - 1), d = n >> s;

    for (ll base : bases) {
        ll p = fexp(base % n, d, n), i = s;

        while (p != 1 && p != n - 1 && base % n && i--)
            p = fmul(p, p, n);

        if (p != n - 1 && i != s) return false;
    }
    return true;
}

// O(n^(1/4))
ll rho(ll n) {
    ll x = 0, y = 0, t = 30, prd = 2, i = 1, q;
    auto f = [&](ll x) { return fmul(x, x, n) + i; };
    while (t++ % 40 || __gcd(prd, n) == 1) {
        if (x == y) x = ++i, y = f(x);
        if ((q = fmul(prd, max(x, y) - min(x, y), n))) prd = q;
        x = f(x), y = f(f(y));
    }
    return __gcd(prd, n);
}

// O(n^(1/4))
vector<ll> factor(ll n) {
    if (n == 1) return {};
    if (isPrime(n)) return {n};
    ll x = rho(n);
    auto l = factor(x), r = factor(n / x);
    for (ll x : r) l.push_back(x);
    return l;
}
```

4.2 Sieve-of-erasthotenes

```cpp
/*
Sieve of Erasthotenes
    Consulta rapida de numeros primos
    Complexidade: O(nlog(log(n)))
    Calcula o maior divisor primo de cada numero
*/

bool prime[LIM];
int big_prime[LIM];
void sieve() {
    memset(prime, 1, sizeof prime);
    prime[0] = prime[1] = false;
    for (int i = 2; i < LIM; i++) {
        if (prime[i]) {
            big_prime[i] = i;
            for (int j = i * 2; j < LIM; j += i)
                prime[j] = false, big_prime[j] = i;
        }
    }
}

// Retorna os divisores de `n` O(sqrt(n))
vector<int> divisores(int n)
{
    vector<int> d;
    for (int i = 1; i * i <= n; i++) {
        if (n % i == 0) {
            d.push_back(i);
            if (i != n / i) d.push_back(n / i);
        }
    }
    d.push_back(n);
    return d;
}

// Fatoracao prima de `n` com sieve O(log(n))
vector<int> sieve_factorization(int n) {
    vector<int> primes;
    while (n > 1) {
        primes.push_back(big_prime[n]);
        n /= big_prime[n];
    }
    return primes;
}

// Fatoracao prima em O(sqrt(n))
vector<pair<int, int>> prime_factorization(int n) {
    vector<pair<int, int>> primes;
    for (int i = 2; i * i <= n; i++) {
        int cnt = 0;
        while (n % i == 0)
            n /= i, cnt++;
        if (cnt > 0)
            primes.push_back({i, cnt});
    }
    if (n > 1)
        primes.push_back({n, 1});
    return primes;
}

// Soma dos divisores de todos os numero de 1 ateh LIM - 1
ll sumDivisors[LIM];
void sum_div()
{
    for (int i = 1; i < LIM; i++) {
        for (int j = i; j < LIM; j += i) {
            sumDivisors[j] += i;
        }
    }
}

// Numero dos divisores de todos os numero de 1 ateh LIM - 1
ll numDivisors[LIM];
void num_div()
{
    for (int i = 1; i < LIM; i++) {
        for (int j = i; j < LIM; j += i) {
            numDivisors[j]++;
        }
    }
}
```

4.3 Extended-euclidean-algorithm

```cpp
/*
    Algoritmo Estendido de Euclides (Extended GCD)

    Complexidade: O(log(min(a, b)))

    Calcula os coeficientes x e y da equacao diofantina:
        ax + by = gcd(a, b)

    Para resolver a equacao ax + by = c, onde c eh um valor dado:
    - Primeiro, eh necessario que c % gcd(a, b) == 0.
    - Se sim, as solucÃţes sao:
        x *= c / gcd(a, b)
        y *= c / gcd(a, b)
    - Solucao geral eh
        x(t) = x0 + (b/gcd(a,b)) * t
        y(t) = y0 - (a/gcd(a,b)) * t

*/

int extendedGCD(int a, int b, int &x, int &y){
    if(!b){
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = extendedGCD(b, a%b, x1, y1);
    x = y1;
    y = x1 - y1*(a/b);
    return d;
}
```

4.4 Matrix-exponentiation

```cpp
/*
    Exponenciacao Rapida de Matrizes  O(mÂş log (b))
    Calcula recorrÃłncias lineares
*/
typedef vector<vector<i64>> matrix;

matrix init(int size) {
    matrix mat(size, vector<i64>(size));
    return mat;
}

vector<i64> vecmul(matrix m, vector<i64> vec, i64 sz, i64 mod) {
    assert(vec.size() == sz);
    vector<i64> ans;
    for (int i = 0; i < sz; i++) {
        for (int j = 0; j < sz; i++) {
            ans[i] += (vec[j] * (m[i][j]));
        }
    }
    return ans;
}

matrix matmul(matrix m1, matrix m2, i64 mod, i64 sz) {
    matrix ans;
    for (int i = 0; i < sz; i++) {
        for (int j = 0; j < sz; j++) {
            for (int k = 0; k < sz; k++) {
                ans[i][j] = (ans[i][j] + 1LL * (m1[i][k] % mod) * (m2[k][j] % mod)) % mod;
            }
        }
    }
    return ans;
}

// O(log(b))
matrix fexp(matrix p, i64 b, i64 mod, i64 sz) {
    matrix ans;
    for(int i = 0; i < sz; i++) ans[i][i] = 1;
    while(b){
        if(b & 1) ans = matmul(ans, p, mod, sz);
        p = matmul(p, p, mod, sz);
        b >>= 1;
    }
    return ans;
}
```

4.5 Fast-exponentiation

```cpp
/*
    Fast Exponentiation
    Calcula a^b mod m em O(log(n))
*/

ll fexp(ll a, ll b, ll MOD){
    ll ans = 1;
    while(b) {
        if(b & 1) ans = (ans * a) % MOD;
        a = (a * a) % MOD;
        b >>= 1;
    }
    return ans;
}

/*
Calcula o Inverso Modular de um numero `a` mod `p`
pelo pequeno teorema de fermat.
*/

ll inv(ll a, ll p){
    return fexp(a, p - 2);
}

/*
    Calcula N escolhe K mod P
*/

ll fact[1000000]; // Preh computar fatoriais
ll comb(ll n, ll k, ll p) {
    return ((fact[n] * inv(fact[k], p) % p) * inv(fact[n - k], p)) % p;
}
```

5 Graph

5.1 Dijsktra

```cpp

/*
Dijkstra - Single Source Shortest Path
Complexidade O(n log (n))
*/
vector<ll> dist(maxn, INF);
vector<pii> g[maxn];
vector<ll> dijkstra() {
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    pq.push({0, 0});
    dist[0] = 0;
    while(!pq.empty()) {
        auto [cost, from] = pq.top();
        pq.pop();
        if (dist[from] != cost) continue;
        for (const auto&[w, to]: g[from]) {
            if (dist[from] + w < dist[to]) {
                dist[to] = dist[from] + w;
```

```
                pq.push({dist[to], to });
            }
        }
    }
    return dist;
}
```

5.2 Lca com rmq

```
/*
    Least Common Ancestor in a Tree
    Computa o menor ancestral de dois nodes a e b
    tbm suporta queries de distancia entre dois nodes

    testado em: https://cses.fi/problemset/result/12501773/ e https://cses.fi/problemset/task/1135
*/

template<typename T>
class LCA {
    private:
        int n; const vector<vector<T>> &g;
        SparseTable<pair<T, T>> rmq;
        vector<T> tin, et, depth;
        int timer = 0;

        // O(n)
        void dfs(int u = 0, int p = -1) {
            tin[u] = timer;
            et[timer++] = u;
            for (int v: g[u]) {
                if (v != p) {
                    depth[v] = depth[u] + 1;
                    dfs(v, u);
                    et[timer++] = u;
                }
            }
        }

    public:
        // Build O(nlogn)
        LCA(vector<vector<T>> & _g): n(_g.size()), g(_g), tin(n), et(2 * n), depth(n), rmq(vector<pair<T ,T>>(1)) {
            dfs();
            vector<pair<T, T>> arr(2 * n);
            for (int i = 0; i <  2 * n; i++) {arr[i] = {depth[et[i]], et[i]}; };
            rmq = SparseTable<pair<T, T>>(arr);
        }

        // O(1)
        T query(int a, int b) {
            if (tin[a] > tin[b]) swap(a, b);
            return rmq.query(tin[a], tin[b]).second;
        }

        // O(1)
        T dist(int a, int b) {
            return depth[a] + depth[b] - 2 * depth[query(a, b)];
        }
};
```

5.3 Bridges

```
/*
    Acha todas as bridges em O(N + M)
*/

int n, m;
const int mxn = 1e5 + 5;
vector<int> g[mxn];
int tin[mxn], low[mxn];
vector<pii> bridges;
int timer = 1;

void dfs(int u, int p) {
    tin[u] = timer++;
    low[u] = tin[u];
    int ch = 0;
    for (int v : g[u])
        if (v != p) {
            if (tin[v])  // lowlink direta
                low[u] = min(tin[v], low[u]);
            else {
                dfs(v, u);
                low[u] = min(low[v], low[u]);
                if (tin[u] < low[v]) bridges.push_back({u, v});
            }
        }
}
```

5.4 Bellman-ford

```
/*
Bellmand Ford Single Source Shortest Path
    Complexidade O(VE)
    Encontra ciclos negativos
*/

struct Edge {
    int from, to, cost;
    Edge(int _f, int _t, int _c): from(_f), to(_t), cost(_c) {}
};
vector<ll> BellmanFord(int n, vector<Edge> &g, int src) {
    vector<ll> distance(n, INF);
    distance[src] = 0;
    for (int u = 0; u < n - 1; u++) {
        for (auto edge : g) {
            auto [from, to, cost] = edge;
            distance[to] = min(distance[to], distance[from] + cost);
        }
    }

    vector<int> negative_cycle(n);
    for (auto edge : g) {
        auto [from, to, cost] = edge;
        if (distance[from] + cost < distance[to]) {
            distance[to] = -INF;
            negative_cycle[to] = true;
        }
    }

    // propaga ciclo negativo e encontra os nos afetados - O(VE)
    for (int u = 0; u < n; u++) {
        if (negative_cycle[u]) {
            queue<int> q;
            q.push(u);
            while (!q.empty()) {
                int node = q.front();
                q.pop();
                for (auto [from, to, cost] : g) {
                    if (from == node && !negative_cycle[to]) {
                        negative_cycle[to] = true;
                        q.push(to);
                    }
                }
            }
```

```
        }
    }

    // Marca os nos afetados por ciclos negativos
    for (int i = 0; i < n; i++) {
        if (negative_cycle[i]) {
            distance[i] = -INF;
        }
    }

    return distance;
}
```

5.5 Floyd-warshall

```
/*
    Floyd Warshall - All Pairs Shortest Path
    Funciona apenas em matrizes
    Complexidade O(nÂş)
*/

vector<vector<ll>> FloydWarshall(int n, vector<vector<int>> &graph) {
    // precomputa distÃćncias O(nÂš)
    vector<vector<ll>> distance(n, vector<ll>(n, INF));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) {
            distance[i][j] = 0;
            } else if (graph[i][j] != -1) {
                distance[i][j] = graph[i][j];
            }
        }
    }

    // O(nÂş)
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                distance[i][j] = min(distance[i][j], distance[i][k] + distance[k][j]);
            }
        }
    }
}
```

5.6 Kosaraju

```
/*
    Kosaraju's algorithm
    Find strongly connected components in a directed graph in O(n)
    with two dfs passes
*/
#include <bits/stdc++.h>
using namespace std;

class Kosaraju {
    private:
        const int n;
        vector<bool> visited;
    public:

    vector<int> dfs(int v, vector<vector<int>> &adj, vector<int> &output) {
        visited[v] = true;
        for (auto u: adj[v]) {
            if (!visited[u]) dfs(u, adj, output);
        }
        output.push_back(v);
    }

    vector<vector<int>> scc(vector<vector<int>> & adj, vector<vector<int>> &adj_transp) {
        int n = adj.size();
        vector<vector<int>> components;
        vector<int> order; visited.assign(n, false);
        for (int i = 0; i < n; i++) {
            if(!visited[i]) dfs(i, adj, order);
        }
        visited.assign(n, false);
        reverse(order.begin(), order.end());
        for (int v: order) {
            if (!visited[v]) {
                vector<int> component;
                dfs(v, adj_transp, component);
                components.push_back(component);
            }
        }
        return components;
    }
};
```

5.7 Articulation

```
/*
    Acha todos os articulation points do grafo em O(N + M)
*/

int n, m;
const int mxn = 1e5 + 5;
vector<int> g[mxn];
int tin[mxn], low[mxn];
vector<int> art;
int timer = 1;

void dfs(int u, int p) {
    tin[u] = timer++;
    low[u] = tin[u];
    int ch = 0;
    int fw = 0;
    for (int v : g[u])
        if (v != p) {
            if (tin[v])  // lowlink direta
                low[u] = min(tin[v], low[u]);
            else {
                dfs(v, u);
                fw++;
                low[u] = min(low[v], low[u]);
                ch = max(low[v], ch);
            }
        }
    if (u == p && fw > 1)
        art.push_back(u);
    else if (u != p && ch && tin[u] <= ch)
        art.push_back(u);
}
```

5.8 Kahn

```
/*
    Kahn Topological Sorting
    Complexidade - O(V + E)

    Encontra a ordenacao topologica e detecta ciclos ao
    mesmo tempo
*/
vector<int> KahnToposort(int n, vector<int> *graph) {
    vector<int> in_degree(n);
    for (int i = 0; i < n; i++) {
        for (int to : graph[i]) {
```

```
            in_degree[to]++;
        }
    }
    queue<int> q;
    for (int i = 0; i < n; i++) {
        if (in_degree[i] == 0)
            q.push(i);
    }

    int idx = 0;
    vector<int> order(n);
    while (!q.empty()) {
        int u = q.front(); q.pop();
        order[idx++] = u;
        for (int v: graph[u]) {
            in_degree[v]--;
            if (in_degree[v] == 0) {
                q.push(v);
            }
        }
    }

    if (idx != n) {
        return {};  // cycle detected
    }

    return order;
}
```

5.9 Dinic

```
/*
O(VÂšE) in general graphs. u
nit capacity networks, it's O(min(V^2/3, E^1/2))
(source/sink only connected to one side of a bipartite graph), it's O(EV).
usually much faster than worst case
*/
struct Edge {
    int u, v;
    ll cap, flow = 0;
    Edge(int u, int v, ll cap) : u(u), v(v), cap(cap) {}
};

template<typename T>
struct Dinic {
    vector<Edge> edges;
    vector<vector<int>> adj;
    int n, s, t, m = 0;
    vector<int> lvl, ptr;
    queue<int> q;
    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        lvl.resize(n);
        ptr.resize(n);
    }
    void add_edge(int u, int v, ll cap, ll other = 0) {
        edges.emplace_back(u, v, cap);
        edges.emplace_back(v, u, other);
        adj[u].push_back(m++);
        adj[v].push_back(m++);
    }
    bool bfs() {
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int id : adj[u]) {
                if (edges[id].cap == edges[id].flow) continue;
                if (lvl[edges[id].v] != -1) continue;
                lvl[edges[id].v] = lvl[u] + 1;
                q.push(edges[id].v);
            }
        }
        return lvl[t] != -1;
    }

    ll dfs(int u, ll pushed) {
        if (pushed == 0) return 0;
        if (u == t) return pushed;
        for (int &cid = ptr[u]; cid < (int)adj[u].size(); cid++) {
            int id = adj[u][cid];
            int v = edges[id].v;
            if (lvl[v] != lvl[u] + 1) continue;
            ll tr = dfs(v, 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 (1) {
            fill(lvl.begin(), lvl.end(), -1);
            lvl[s] = 0;
            q.push(s);
            if (!bfs()) break;
            fill(ptr.begin(), ptr.end(), 0);
            while (ll p = dfs(s, inf)) {
                f += p;
            }
        }
        return f;
    }
};
```

5.10 Mcmf

```
/*
Min cost max flow
Unit: O(VEÂšlog v)
General O(F (E log V))
*/

template <typename Cap, typename Cost>
struct MCMF {
    const Cost INF = numeric_limits<Cost>::max();
    struct Edge {
        int to;
        Cap cap, flow;
        Cost cost;
        Edge(int to, Cap cap, Cost cost) : to(to), cap(cap), flow(0), cost(cost) {}
        Cap res() const { return cap - flow; }
    };
    int m = 0, n;
    vector<Edge> edges;
    vector<vector<int>> g;
    vector<Cap> neck;
    vector<Cost> dist, pot;
    vector<int> from;
    MCMF(int n) : n(n), g(n), neck(n), pot(n) {}
    void add_edge(int u, int v, Cap cap, Cost cost) {
        if (u != v) {
            edges.emplace_back(v, cap, cost);
            edges.emplace_back(u, 0, -cost);
            g[u].emplace_back(m++);
            g[v].emplace_back(m++);
        }
```

```
    }
    void spfa(int s) {
        vector<bool> inq(n, false);
        queue<int> q({s});
        while (!q.empty()) {
            auto u = q.front();
            q.pop();
            inq[u] = false;
            for (auto e : g[u]) {
                auto ed = edges[e];
                if (ed.res() == 0) continue;
                Cost w = ed.cost + pot[u] - pot[ed.to];
                if (pot[ed.to] > pot[u] + w) {
                    pot[ed.to] = pot[u] + w;
                    if (!inq[ed.to]) {
                        inq[ed.to] = true;
                        q.push(ed.to);
                    }
                }
            }
        }
    }
    bool dijkstra(int s, int t) {
        dist.assign(n, INF);
        from.assign(n, -1);
        neck[s] = numeric_limits<Cap>::max();
        using ii = pair<Cost, int>;
        priority_queue<ii, vector<ii>, greater<ii>> pq;
        pq.push({dist[s] = 0, s});
        while (!pq.empty()) {
            auto [d_u, u] = pq.top();
            pq.pop();
            if (dist[u] != d_u) continue;
            for (auto i : g[u]) {
                auto ed = edges[i];
                Cost w = ed.cost + pot[u] - pot[ed.to];
                if (ed.res() > 0 && dist[ed.to] > dist[u] + w) {
                    from[ed.to] = i;
                    pq.push({dist[ed.to] = dist[u] + w, ed.to});
                    neck[ed.to] = min(neck[u], ed.res());
                }
            }
        }
        return dist[t] < INF;
    }
    pair<Cap, Cost> mcmf(int s, int t, Cap k = numeric_limits<Cap>::max()) {
        Cap flow = 0;
        Cost cost = 0;
        spfa(s);
        while (flow < k && dijkstra(s, t)) {
            Cap amt = min(neck[t], k - flow);
            for (int v = t; v != s; v = edges[from[v] ^ 1].to) {
                cost += edges[from[v]].cost * amt;
                edges[from[v]].flow += amt;
                edges[from[v] ^ 1].flow -= amt;
            }
            flow += amt;
            fix_pot();
        }
        return {flow, cost};
    }
    void fix_pot() {
        for (int u = 0; u < n; ++u) {
            if (dist[u] < INF) {
                pot[u] += dist[u];
            }
        }
    }
};
```

5.11 Lca

```
const int mxn = 2e5 + 5;
const int LOG = 22;
int n, q;
int tin[mxn], tout[mxn];
vector<vector<int>> up;  // up[v][k] = 2^k-esimo ancestor de v
vector<int> g[mxn];
int lvl[mxn];
int timer = 0;
void dfs(int u, int p) {
    tin[u] = ++timer;
    lvl[u] = lvl[p] + 1;
    up[u][0] = p;
    for (int i = 1; i <= LOG; i++) {
        up[u][i] = up[up[u][i - 1]][i - 1];
    }
    for (int v : g[u]) {
        if (v != u && !tin[v])
            dfs(v, u);
    }
    tout[u] = ++timer;
}

bool is_ancestor(int u, int v) {
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int a, int b) {
    if (is_ancestor(a, b)) return a;
    if (is_ancestor(b, a)) return b;
    for (int i = LOG; i >= 0; i--) {
        if (!is_ancestor(up[a][i], b)) {
            a = up[a][i];
        }
    }
    return up[a][0];
}
```

5.12 Binary-lifting

```
void preprocess(int n) {
    for (int v = 0; v < n; v++)
        up[v][0] = parent[v];
    for (int i = 1; i < log2dist; i++) {
        for (int v = 0; v < n; v++) {
            if (v != 0) depth[v] = depth[parent[v]] + 1;
            up[v][i] = up[up[v][i - 1]][i - 1];
        }
    }
}

void dfs(int u, int p = 0) {
    for (int v : tree[u]) {
        if (v != p) {
            dfs(v, u);
            parent[v] = u;
        }
    }
}

int kth_ancestor(int node, int k) {
    if (depth[node] < k) return -1;
    for (int i = 0; i < log2dist; i++) {
        if (k & (1 << i)) {
```

```
            node = up[node][i];
        }
    }
    return node + 1;
}
```

5.13 Kruskal

```
struct Edge
{
    int u, v, w;
    Edge() {}
    Edge(int a, int b, int c): u(a), v(b), w(c) {}
    bool operator<(const Edge &s) const { return w < s.w; }
};

/*
    Encontra o custo da Arvore Geradora Minima
    Complexidade O(E log E)
    find(u) e unite(u, v) de Union-Find
*/

ll Kruskal(vector<Edge> &g) {
    sort(begin(g), end(g));
    ll total = 0;
    for (auto [u, v, w]: g) {
        if (find(u) != find(v)) {
            unite(u, v);
            total += w;
        }
    }
    return total;
}
```

6 String

6.1 Suffix-array

```
/*
    Suffix Array
    ConstrÃşi o suffix array e o lcp array em O(n log n) pro sa + O(n) pro lcp
*/


vector<int> build_sa(string &s) {
    s += '$';
    int n = s.size();
    const int ALPHA = 256;
    vector<int> p(n), c(n), cnt(max(ALPHA, n));
    for (int i = 0; i < n; i++) cnt[s[i]]++;
    for (int i = 1; i < ALPHA; i++) cnt[i] += cnt[i - 1];
    for (int i = 0; i < n; i++) p[--cnt[s[i]]] = i;
    c[p[0]] = 0; int classes = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i - 1]]) classes++;
        c[p[i]] = classes - 1;
    }
    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; h++) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0) pn[i] += n;
        }
        fill(cnt.begin(), cnt.begin() + classes, 0);
        for (int i = 0; i < n; i++) cnt[c[pn[i]]]++;
        for (int i = 1; i < classes; i++) cnt[i] += cnt[i - 1];
        for (int i = n - 1; i >= 0; i--) p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        int new_classes = 1;
        for (int i = 1; i < n; i++) {
            pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n]};
            pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 << h)) % n]};
            if (cur != prev) ++new_classes;
            cn[p[i]] = new_classes - 1;
        }
        c.swap(cn);
        swap(classes, new_classes);
    }
    s.pop_back();
    p.erase(p.begin());
    return p;
}

vector<int> build_lcp(string &s, vector<int> &sa) {
    int n = s.size();
    vector<int> rank(n, 0);
    for (int i = 0; i < n; i++) rank[sa[i]] = i;
    int k = 0;
    vector<int> lcp(n - 1, 0);
    for (int i = 0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0; continue;
        }
        int j = sa[rank[i] + 1];
        while (i + k < n and j + k < n and s[i + k] == s[j + k]) k++;
        lcp[rank[i]] = k;
        if(k) k--;
    }
    return lcp;
}
```

6.2 Double-hash

```
/*
    Double Polynomial Hashing
    Prehcalculo - O(n)
    Substring hash queries - O(1)
    Hash(l, m - 1) calcula o hash da substring incluindo o l de tamanho m
*/

const int MOD1 = 188'888'881;
const int MOD2 = 1e9 + 7;
const int base = 137;

ll pow1[MAXN];
ll pow2[MAXN];

// O(n) - Chamar antes
void calc_pow() {
    pow1[0] = pow2[0] = 1;
    for (int i = 1; i < MAXN; i++)
        pow1[i] = (pow1[i - 1] * base) % MOD1,
        pow2[i] = (pow2[i - 1] * base) % MOD2;
}

struct Hashing {
    vector<pair<ll, ll>> pref;
    // O(1)
    Hashing(string &s) {
        pref = vector<pair<ll, ll>>(s.size() + 1, {0, 0});
        for (int i = 0; i < s.size(); i++)
            pref[i + 1].first = ((pref[i].first * base) % MOD1 + s[i]) % MOD1,
                      pref[i + 1].second = ((pref[i].second * base) % MOD2 + s[i]) % MOD2;
```

```
    }

    // O(1)
    ll operator()(int a, int b) {
        ll h1 = (MOD1 + pref[b + 1].first - (pref[a].first * pow1[b - a + 1]) % MOD1) % MOD1;
        ll h2 = (MOD2 + pref[b + 1].second - (pref[a].second * pow2[b - a + 1]) % MOD2) % MOD2;
        return (h1 << 32) | h2;
    }
};
```

6.3 Manacher

```
/*
    Manacher's algorithm
    Acha o raio do maior palindromo centralizado em i pra cada i
    so acha palindromo impar
    se for pra achar par tb bota um caracter entre cada:
    b$a$a$b
    b$a$a$b
    1124211
*/
vector<int> manacher(string &S){
  vector<int> R(S.size());
  int i = 0, j = 0;
  while (i < S.size()) {
    while (i-j >= 0 && i+j < S.size() && S[i-j] == S[i+j]) ++j;
    R[i] = j;
    int k = 1;
    while (i-k >= 0 && k+R[i-k] < j) R[i+k] = R[i-k], ++k;
    i += k; j -= k;
  }
  return R;
}
```

6.4 Trie

```
const int ALPHA = 26; // tamanho do alfabeto
/*
    Trie - arvore de Prefixos
    maxn - Soma do tamanho de todas as strings
*/
int trie[maxn][ALPHA], word_end[maxn], z = 1;

// Add(P) - O(|P|)
void add(string &s) {
    int cur = 0;
    for(int i = 0; i < s.size(); i++) {
        if (trie[cur][s[i] - 'a'] == -1) {
            memset(trie[z], -1, sizeof trie[z]);
            trie[cur][s[i] - 'a'] = z++;
        }
        cur = trie[cur][s[i] - 'a'];
    }
    word_end[cur]++;
}

// Query(P) - O(|P|)
int query(string &s){
    int cur = 0;
    for(int i = 0; i < s.size(); i++){
        if(trie[cur][s[i] - 'a'] == -1) return 0;
        cur = trie[cur][s[i] - 'a'];
    }
    return word_end[cur];
}

// Sempre inicializar antes
void init(){
    memset(trie[0], -1, sizeof trie[0]);
    memset(word_end, 0, sizeof word_end);
    z = 1;
}
```

7 Data structures

7.1 Fenwick-tree

```
/*
    Fenwick Tree - Range Queries
*/

struct Fenwick {
    int n;
    vector<ll> bit;
    Fenwick(int _n = 0) { init(_n); }
    void init(int _n) {
        n = _n;
        bit.assign(n + 1, 0);
    }
    // add val at position pos (0-based)
    void add(int pos, int val = 1) {
        for (int i = pos + 1; i <= n; i += i & -i) bit[i] += val;
    }
    // sum [0..pos] (0-based)
    ll sum(int pos) const {
        if (pos < 0) return 0;
        ll s = 0;
        for (int i = pos + 1; i > 0; i -= i & -i) s += bit[i];
        return s;
    }
};
```

7.2 Segment-tree-lazy

```
template <typename T>
class LazySegmentTree {
private:
    const int sz;
    vector<T> tree;
    vector<T> lazy;

    void apply(int v, int len, T add) {
        tree[v] += add * len;
        lazy[v] += add;
    }

    void pushdown(int v, int l, int r) {
        int m = (l + r) / 2;
        apply(2 * v, m - l + 1, lazy[v]);
        apply(2 * v + 1, r - m, lazy[v]);
        lazy[v] = 0;
    }

    void build(vector<T> &a, int v, int l, int r) {
        if (l == r) {
            tree[v] = a[l];
        } else {
            int m = l + (r - l) / 2;
            build(a, v * 2, l, m);
            build(a, v * 2 + 1, m + 1, r);
            tree[v] = tree[v * 2] + tree[v * 2 + 1];
        }
    }

    void range_add(int v, int l, int r, int ql, int qr, int add) {
```

```cpp
        if (qr < l || ql > r) {
            return;
        }
        if (ql <= l and r <= qr) {
            apply(v, r - l + 1, add);
        } else {
            pushdown(v, l, r);
            int m = (l + r) / 2;
            range_add(2 * v, l, m, ql, qr, add);
            range_add(2 * v + 1, m + 1, r, ql, qr, add);
            tree[v] = tree[2 * v] + tree[2 * v + 1];
        }
    }

    T range_sum(int v, int l, int r, int ql, int qr) {
        if (qr < l || ql > r) return 0;
        if (ql <= l and r <= qr) return tree[v];
        pushdown(v, l, r);
        int m = (l + r) / 2;
        return range_sum(2 * v, l, m, ql, qr) + range_sum(2 * v + 1, m + 1, r, ql, qr);
    }

public:
    LazySegmentTree(int n) : sz(n), lazy(4 * n), tree(4 * n) {}

    void add(int ql, int qr, int add) {
        range_add(1, 0, sz - 1, ql, qr, add);
    }

    T qry(int ql, int qr) {
        return range_sum(1, 0, sz - 1, ql, qr);
    }

    void build_seg(vector<T> &a) {
        build(a, 1, 0, sz - 1);
    }
};

/*
    Range sum Lazy Segment Tree
    Allows for range updates and range queries
    Query - O(log(n))
    Update - O(log(n))
    Apply - O(1)
    Build - O(n)
*/
```

7.3 Sparse-table

```cpp
/*
    Range (Idempotent Function) Query
    Build - O(n log n)
    Query - O(1)
    Nao suporta updates, para queries de funcoes tipo soma eh melhor so usar uma seg mesmo
    Testado em: https://judge.yosupo.jp/problem/staticrmq
*/

template<typename T> class SparseTable {
    private:
        int n, k;
        vector<vector<T>> st;
    public:
    SparseTable(const vector<T> & v) {
        n = v.size(); k = 31 - __builtin_clz(n) + 1;
        st.resize(k); st[0] = v;
        for (int i = 1; i < k; i++) {
            st[i].resize(n - (1 << i) + 1);
            for (int j = 0; j + (1 << i) <= n; j++)
                st[i][j] = min(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
        }
    }
    T query(int l, int r) {
        int p = 31 - __builtin_clz(r - l + 1);
        return min(st[p][l], st[p][r - (1 << p) + 1]);
    }
};
```

7.4 Hash Roubado

```cpp
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
struct chash {
    const int RANDOM = (long long)(make_unique<char>().get()) ^ chrono::high_resolution_clock::now().time_since_epoch().count();
    static unsigned long long hash_f(unsigned long long x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    static unsigned hash_combine(unsigned a, unsigned b) { return a * 31 + b; }
    int operator()(int x) const { return hash_f(x) ^ RANDOM; }
};
gp_hash_table<int, int, chash> d1;
```

7.5 Treap

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

mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count());
struct node {
    int cnt, weight, left, right;
    ll sum = 0;
    int val, neg;
    node (int v): cnt(1), weight(rng()), left(-1), right(-1), sum(v), val(v), neg(0) {}
};

vector<node> tree;

void push_lazy(int i) {
    if (tree[i].neg) {
        tree[i].val *= -1;
        tree[i].sum *= -1;
        tree[i].neg = 0;
        // resolve lazy negations and push to children
        if (tree[i].left >= 0) tree[tree[i].left].neg ^= 1;
        if (tree[i].right >= 0) tree[tree[i].right].neg ^= 1;
    }
}

// subtree size
int cnt(int i) { return i == -1 ? 0 : tree[i].cnt; };

ll sum(int i) {
    if (i >= 0 and tree[i].neg) push_lazy(i);
    return i == -1 ? 0 : tree[i].sum;
}
```

```cpp

void flip(int i) {
    tree[i].neg ^= 1;
    push_lazy(i);
}

void update_cnt(int i) {
    tree[i].cnt = 1 + cnt(tree[i].left) + cnt(tree[i].right);
    tree[i].sum = tree[i].val + sum(tree[i].left) + sum(tree[i].right);
}

// split treap at index k
void split(int n, int k, int &l ,int &r) {
    if (n == -1) { l = r = -1; return; }
    push_lazy(n); // always resolve pending lazy udpates before opreations
    if (cnt(tree[n].left) < k) {
        split(tree[n].right, k - cnt(tree[n].left) - 1, tree[n].right, r), l = n;
    } else {
        split(tree[n].left, k, l, tree[n].left), r = n;
    }
    update_cnt(n);
}

void merge(int l, int r, int &n) {
    if (l == -1 || r == -1) {n = (l == -1 ? r : l); return; }
    push_lazy(l), push_lazy(r); // reoslve pending
    if (tree[l].weight > tree[r].weight) {
        merge(tree[l].right, r, tree[l].right), n = l;
    } else {
        merge(l, tree[r].left, tree[r].left), n = r;
    }
    update_cnt(n);
}

void solve() {
    int n, q; cin >> n >> q;
    vector<int> a(n);
    for (int &x: a) cin >> x;
    ll cur_sum = 0;
    int rt = -1;
    for (int i = 0; i < n; i++) {
        cur_sum += i % 2 == 1 ? -a[i] : a[i];
        tree.push_back(node(i % 2 == 1 ? -a[i] : a[i]));
        merge(rt, tree.size() - 1, rt);
    }
    while(q--) {
        int l, r; cin >> l >> r; --l, --r;
        int left, mid, right;
        // mid = [0, r] right -> [r + 1, ]
        split(rt, r + 1, mid, right);
        // left -> [0, l - 1] mid [l, r]
        split(mid, l, left, mid);
        // take off the sum of mid interval
        cur_sum -= sum(mid);
        int k, shift;
        // k [0, 1] shift [1, r]
        split(mid, 1, k, shift);
        // flip signs odd -> even and even -> odd because of shifting
        flip(shift);
        if (l % 2 != r % 2) flip(k);
        // merge [shift, k] into mid
        merge(shift, k, mid);
        // merge and add sum back
        cur_sum += sum(mid);
        // merge left and mid into rt
        merge(left, mid, rt);
        // merge rt and right into rt'
        merge(rt, right, rt);
        if (cur_sum > 0) cout << "FISH\n";
        else if (cur_sum == 0) cout << "TIE\n";
        else cout << "MAN\n";
    }
}

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    int tt = 1; // cin >> tt;
    while(tt--) {
        solve();
    }
}
```

7.6 Dsu-rollback

```cpp
class DSU {
private:
    vector<int> p, sz;
    vector<pair<int &, int>> history;

public:
    DSU(int n) : p(n), sz(n, 1) { iota(p.begin(), p.end(), 0); }

    int get(int x) { return x == p[x] ? x : get(p[x]); }

    void unite(int a, int b) {
        a = get(a);
        b = get(b);
        if (a == b) {
            return;
        }
        if (sz[a] < sz[b]) {
            swap(a, b);
        }

        history.push_back({sz[a], sz[a]});
        history.push_back({p[b], p[b]});
        p[b] = a;
        sz[a] += sz[b];
    }
    int snapshot() { return history.size(); }
    void rollback(int until) {
        while (snapshot() > until) {
            history.back().first = history.back().second;
            history.pop_back();
        }
    }
};
```

7.7 Union-find

```cpp
/*
    Disjoint Set Union with path compression
    Complexidade:
        - find(u) O(alpha(n))
        - unite(u) O(alpha(n))
*/

template <typename T>
struct UnionFind {
    vector<int> par, sz;

```

```
    // O(n)
    UnionFind(int n) {
        par.assign(n, 0);
        sz.assign(n, 1);
        iota(par.begin(), par.end(), 0);
    }

    // O(alpha(n)) ~ O(1)
    int find(int x) { return par[x] == x ? x : (par[x] = find(par[x])); }

    // O(alpha(n)) ~ O(1)
    bool unite(int x, int y) {
        int x_root = find(x);
        int y_root = find(y);
        if (x_root == y_root) { return false; }
        if (sz[x_root] < sz[y_root]) { swap(x_root, y_root); }
        sz[x_root] += sz[y_root];
        par[y_root] = x_root;
        return true;  // (some condition met for component);
    }
};
```

7.8 Pbds

```
#include <ext/pb_ds/assoc_container.hpp>  // Common file
#include <ext/pb_ds/tree_policy.hpp>      // Including tree_order_statistics_node_update
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
             tree_order_statistics_node_update>
    ordered_set;
ordered_set X;
X.insert(1);
X.find_by_order(0);  // iterador pra kesimo maior elemento
X.order_of_key(-5);  // numero de elementos estritamente menor q chave
end(X), begin(X);
```

7.9 Segment-tree

```
// 0-Based Segment Tree - Range Queries
int n;
struct SegmentTree {
    private:
        vector<int> tree;
        const int NEUTRAL = 1e9 + 9;
    public:

    SegmentTree(int _n) {
        tree.resize(4 * _n);
    }

    int join(int a, int b) {}

    // O(n) a is a 0-based array
    void build(vector<int> &a, int l = 0, int r = n - 1, int v = 0) {
        if (l == r) {
            tree[v] = a[l];
            return;
        } else {
            int mid = l + (r - l) / 2;
            build(a, l, mid, v * 2 + 1);
            build(a, mid + 1, r, v * 2 + 2);
            tree[v] = join(tree[v * 2 + 1], tree[v * 2 + 2]);
        }
    }

    // O(log(n))
    void update(int pos, int val, int l = 0, int r = n - 1, int v = 0) {
        if (l == r) {
            tree[v] = val;
            return;
        } else {
            int mid = l + (r - l) / 2;
            if (pos <= mid) {
                update(pos, val, l, mid, v * 2 + 1);
            } else {
                update(pos, val, mid + 1, r, v * 2 + 2);
            }
            tree[v] = join(tree[v * 2 + 1], tree[v * 2 + 2]);
        }
    }

    // O(log(n))
    int query(int a, int b, int l = 0, int r = n - 1, int v = 0) {
        if (b < l || a > r) return NEUTRAL;
        if (a <= l && r <= b) return tree[v];
        int mid = l + (r - l) / 2;
        int left = query(a, b, l, mid, v * 2 + 1);
        int right = query(a, b, mid + 1, r, v * 2 + 2);
        return join(left, right);
    }
};
```

7.10 Merge-sort-tree

```
/*
    Geralmente queries em O(nlogÂšn) sem update
*/

int n;
vector<int> tree[4 * maxn], a;
void build(int l = 0, int r = n - 1, int v = 0) {
    if (l == r) {
        tree[v].push_back(a[l]);
    } else {
        int m = (l + r) / 2;
        build(l, m, v * 2 + 1);
        build(m + 1, r, v * 2 + 2);
        int i = 0, j = 0;
        while (i < tree[v * 2 + 1].size() and j < tree[v * 2 + 2].size()) {
            if (tree[v * 2 + 1][i] < tree[v * 2 + 2][j])
                tree[v].push_back(tree[v * 2 + 1][i++]);
            else
                tree[v].push_back(tree[v * 2 + 2][j++]);
        }
        while (i < tree[v * 2 + 1].size())
            tree[v].push_back(tree[v * 2 + 1][i++]);
        while (j < tree[v * 2 + 2].size())
            tree[v].push_back(tree[v * 2 + 2][j++]);
    }
}

int query(int a, int b, int k, int l = 0, int r = n - 1, int v = 0) {
    if (b < l || a > r) return 0;
    if (l >= a and r <= b) {
        // answer query
    }
```

```
    int m = (l + r) / 2;
    int half1 = query(a, b, k, l, m, v * 2 + 1);
    int half2 = query(a, b, k, m + 1, r, v * 2 + 2);
    return half1 + half2;
}
```