TEAMBOOK TRUCHO

POLLOS PRIMO

November 8, 2024

Contents

1 estdatos

1.1 MACROS

```
#include <bits/stdc++.h>
using namespace std;
// #include <ext/pb_ds/assoc_container.hpp>
// #include <ext/pb_ds/assoc_container.hpp>
// #include <ext/pb_ds/tree_policy.hpp>
// #include <ext/rope>
#define int long long
#define uset unordered_set
#define f first
#define sst stringstream
#define s second
#define umap unordered_map
#define mp make_pair
#define pb push_back
#define sz(x) (int)(x).size()
#define all(a) (a).begin(), (a).end()
#define rall(a) (a).rbegin(), (a).rend()
#define floatigual(a, b) (fabs(a - b) < EPS)
#define mod(a) md(a, MOD)
#define fore(i, a, n) for(int i = (a); i < (n); i++)
#define forb(i, n) for (int i = (n) - 1; i >= 0; i--)
#define FORDD(i, a, b) for (int i = (b) - 1; i >= (a); --i)
#define techo(a, b) (a / b + (a % b != 0))
#define popcount(x) __builtin_popcountll(x);
using namespace std;
// using namespace __gnu_pbds;
// using namespace __gnu_cxx;
typedef long double ld;
typedef unsigned long long ull;
typedef pair<int, int> pii;
typedef vector<int> vi;
typedef vector<bool> vbol;
// typedef
// tree<int,null_type,less<int>,rb_tree_tag,tree_order_statistics_node_update>
// ordered_set; find_by_order kth largest  order_of_key < mt19937
// rng(chrono::steady_clock::now().time_since_epoch().count()); rng
const int tam = 200010;
const int MOD = 1000000007;
const int MOD1 = 998244353;
const double DINF = 1e100;
const double EPS = 1e-9;
const double PI = acos(-1);
// Modificar la constante para la criba
const int constante = 500;
vector<bool> criba(constante + 1);
vector<int> primos;
void eratostenes() {
    criba[0] = criba[1] = true;
    for (int i = 2; i <= constante; ++i) {
        if (!criba[i]) {
            primos.push_back(i);
            for (int j = i * i; j <= constante; j += i) {
                criba[j] = 1;
            }
        }
    }
}
int binpow(int a, int b) {
    if (b == 0) {
        return 1;
    } else if (b == 1) {
        return a;
    } else if (b < 0) {
        return 1 / binpow(a, -b);
    } else if (b % 2 == 0) {
        int we = binpow(a, b / 2);
        return we * we;
    } else {
        return a * binpow(a, b - 1);
    }
}
int gauss(int n) {
    int res = (((n % MOD) * ((n + 1) % MOD)) % MOD) / 2;
    return res;
}
int expMod(int base, int exponente, int mod) {
    int res = 1;
    base %= mod;
    while (exponente > 0) {
        if (exponente % 2 == 1) res = (res * base) % mod;
        exponente >>= 1;
        base = (base * base) % mod;
    }
    return res;
}
class UnionFind {
    vector<int> parents;
    vector<int> sizes;

   public:
    UnionFind(int size) : parents(size), sizes(size, 1) {
        for (int i = 0; i < size; i++) {
            parents[i] = i;
        }
    }
    int find(int x) {
        return parents[x] == x ? x : (parents[x] = find(parents[x]));
    }
    bool join(int x, int y) {
        int x_root = find(x);
        int y_root = find(y);
        if (x_root == y_root) {
            return false;
        }
        if (sizes[x_root] < sizes[y_root]) {
            swap(x_root, y_root);
        }
        sizes[x_root] += sizes[y_root];
        parents[y_root] = x_root;
        return true;
    }
    bool connected(int x, int y) { return find(x) == find(y); }
};
```

```

void solve() {

}
signed main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    int t;cin>>t;while(t--)solve();
    solve();
}
//PLUS ULTRA RECARGADO!!!
```

MACROS

1.2 bit

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

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

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

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

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

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

bit

1.3 bit2d

```
/*2D BIT is basically a BIT where each element is another BIT.
Updating by adding v on (x, y) means it's effect will be found
throughout the rectangle [(x, y), (max_x, max_y)],
and query for (x, y) gives you the result of the rectangle
[(0, 0), (x, y)], assuming the total rectangle is
```

```
[(0, 0), (max_x, max_y)]. So when you query and update on
this BIT,you have to be careful about how many times you are
subtracting a rectangle and adding it. Simple set union formula
works here.

So if you want to get the result of a specific rectangle
[(x1, y1), (x2, y2)], the following steps are necessary:

Query(x1,y1,x2,y2) = getSum(x2, y2)-getSum(x2, y1-1) -
                                        getSum(x1-1, y2)+getSum(x1-1, y1-1)

Here 'Query(x1,y1,x2,y2)' means the sum of elements enclosed
in the rectangle with bottom-left corner's co-ordinates
(x1, y1) and top-right corner's co-ordinates - (x2, y2)

Constraints -> x1<=x2 and y1<=y2

        /\
y |
        |                --------(x2,y2)
        |                |       |
        |                |       |
        |                |       |
        |                ---------
        |        (x1,y1)
        |
        |___________________________
(0, 0)                           x-->

In this program we have assumed a square matrix. The
program can be easily extended to a rectangular one. */

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

#define N 4 // N-->max_x and max_y

// A structure to hold the queries
struct Query
{
        int x1, y1; // x and y co-ordinates of bottom left
        int x2, y2; // x and y co-ordinates of top right
};

// A function to update the 2D BIT
void updateBIT(int BIT[][N+1], int x, int y, int val)
{
        for (; x <= N; x += (x & -x))
        {
                // This loop update all the 1D BIT inside the
                // array of 1D BIT = BIT[x]
                for (int yy=y; yy <= N; yy += (yy & -yy))
                        BIT[x][yy] += val;
        }
        return;
}

// A function to get sum from (0, 0) to (x, y)
int getSum(int BIT[][N+1], int x, int y)
```

```
{
        int sum = 0;

        for(; x > 0; x -= x&-x)
        {
                // This loop sum through all the 1D BIT
                // inside the array of 1D BIT = BIT[x]
                for(int yy=y; yy > 0; yy -= yy&-yy)
                {
                        sum += BIT[x][yy];
                }
        }
        return sum;
}

// A function to create an auxiliary matrix
// from the given input matrix
void constructAux(int mat[][N], int aux[][N+1])
{
        // Initialise Auxiliary array to 0
        for (int i=0; i<=N; i++)
                for (int j=0; j<=N; j++)
                        aux[i][j] = 0;

        // Construct the Auxiliary Matrix
        for (int j=1; j<=N; j++)
                for (int i=1; i<=N; i++)
                        aux[i][j] = mat[N-j][i-1];

        return;
}

// A function to construct a 2D BIT
void construct2DBIT(int mat[][N], int BIT[][N+1])
{
        // Create an auxiliary matrix
        int aux[N+1][N+1];
        constructAux(mat, aux);

        // Initialise the BIT to 0
        for (int i=1; i<=N; i++)
                for (int j=1; j<=N; j++)
                        BIT[i][j] = 0;

        for (int j=1; j<=N; j++)
        {
                for (int i=1; i<=N; i++)
                {
                        // Creating a 2D-BIT using update function
                        // everytime we/ encounter a value in the
                        // input 2D-array
                        int v1 = getSum(BIT, i, j);
                        int v2 = getSum(BIT, i, j-1);
                        int v3 = getSum(BIT, i-1, j-1);
                        int v4 = getSum(BIT, i-1, j);

                        // Assigning a value to a particular element
                        // of 2D BIT
                        updateBIT(BIT, i, j, aux[i][j]-(v1-v2-v4+v3));
                }
        }

        return;
}

// A function to answer the queries
void answerQueries(Query q[], int m, int BIT[][N+1])
{
        for (int i=0; i<m; i++)
        {
                int x1 = q[i].x1 + 1;
                int y1 = q[i].y1 + 1;
                int x2 = q[i].x2 + 1;
                int y2 = q[i].y2 + 1;

                int ans = getSum(BIT, x2, y2)-getSum(BIT, x2, y1-1)-
                                getSum(BIT, x1-1, y2)+getSum(BIT, x1-1, y1-1);

                printf ("Query(%d, %d, %d, %d) = %d\n",
                                q[i].x1, q[i].y1, q[i].x2, q[i].y2, ans);
        }
        return;
}

// Driver program
int main()
{
        int mat[N][N] = {{1, 2, 3, 4},
                                        {5, 3, 8, 1},
                                        {4, 6, 7, 5},
                                        {2, 4, 8, 9}};

        // Create a 2D Binary Indexed Tree
        int BIT[N+1][N+1];
        construct2DBIT(mat, BIT);

        /* Queries of the form - x1, y1, x2, y2
        For example the query- {1, 1, 3, 2} means the sub-matrix-
        y
        /\
3 |      1 2 3 4          Sub-matrix
2 |      5 3 8 1          {1,1,3,2}       --->    3 8 1
1 |      4 6 7 5                                                                6 7 5
0 |      2 4 8 9
        |
--|------ 0 1 2 3 ----> x
        |

        Hence sum of the sub-matrix = 3+8+1+6+7+5 = 30

        */

        Query q[] = {{1, 1, 3, 2}, {2, 3, 3, 3}, {1, 1, 1, 1}};
        int m = sizeof(q)/sizeof(q[0]);

        answerQueries(q, m, BIT);

        return(0);
}
```

bit2d

1.4 hashtrucho

```
random_device rd;
mt19937_64 gen(rd());
map<ull, ull> mapping;
set<ull> usados = { 0 };

for (auto &c : v) {
    ull random;
    if (!mapping.count(c)) {
        do { random = gen(); } while (usados.count(random));
        usados.insert(random);
        mapping[c] = random;
    } else {
        random = mapping[c];
    }
    c = random;
}

//buscar los macros para esto
```

hashtrucho

1.5 lazytree

```
struct Node {
    ll mn;
    ll size = 1;

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

struct Func {
    ll a = 0;
};

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

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

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

```
}

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

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

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

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

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

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

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

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

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

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

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

lazytree

1.6 minimalmacros

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define f first
#define sst stringstream
#define s second
#define pb push_back
#define sz(x) (int)(x).size()
#define all(a) (a).begin(), (a).end()
#define rall(a) (a).rbegin(), (a).rend()
#define fore(i, a, n) for(int i = (a); i < (n); i++)
#define forb(i, n) for (int i = (n) - 1; i >= 0; i--)
#define popcount(x) __builtin_popcountll(x);
typedef pair<int, int> pii;
typedef vector<int> vi;
const int MOD = 1000000007;
const double EPS = 1e-9;
const double PI = acos(-1);
const int INF = 1e18;
//PLUS ULTRA RECARGADO!!!
void solve() {

}
signed main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    int t;cin>>t;while(t--)solve();
    solve();
}

/*ordered set:
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

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

minimalmacros

1.7 minsparce

```
using Type = int;
//xd ?????
struct min_sparse {

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

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

    Type query(int x, int y) {
        int n = y - x + 1;
        int logg = -1;
        while (n) logg++, n/=2;
        return min(sparse[x][logg], sparse[y-(1 << logg)+1][logg]);
    }
};
```

1.8 segtreegeneral

```cpp
// >>>>>>> Implement
// Example of a Segment tree of Xor
struct Node {
    ll a = 0;
};

Node e() {
    Node node;
    return node;
}

Node op(Node a, Node b) {
    Node node;
    node.a = a.a ^ b.a;
    return node;
}
// >>>>>>>> Implement

struct segtree {
    vector<Node> nodes;
    ll n;

    void init(int n) {
        auto a = vector<Node>(n, e());
        init(a);
    }

    void init(vector<Node>& initial) {
        nodes.clear();
        n = initial.size();
        int size = 1;
        while (size < n) {
            size *= 2;
        }
        nodes.resize(size * 2);
        build(0, 0, n-1, initial);
    }

    void build(int i, int sl, int sr, vector<Node>& initial) {
        if (sl == sr) {
            nodes[i] = initial[sl];
        } else {
            ll mid = (sl + sr) >> 1;
            build(i*2+1, sl, mid, initial);
            build(i*2+2, mid+1,sr,initial);
            nodes[i] = op(nodes[i*2+1], nodes[i*2+2]);
        }
    }

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

    void update(int pos, Node node) {
        update(0, 0, n - 1, pos, node);
    }

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

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

    Node get(int i) {
        return query(i, i);
    }
};
```

segtreegeneral

1.9 segtreeterativo

```
// >>>>>>>>> Implement
struct Node { //VALOR NEUTRO

int x = 0; };

Node e() { return Node(); }

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

struct segtree {
    vector<Node> t;
    int n;

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

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

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

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

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

    Node get(int i) {
        return query(i, i);
    }
};
```

segtreeterativo

1.10 struclazy

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

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

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

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

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

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

struclazy

1.11 structsegtree

```cpp
template <typename T>
struct segtree {
    int n;
    vector<T> tree;
    T neutral;
    void init(int nn, T neutralx) {
        neutral = neutralx;
        n = nn;
        int size = 1;
        while (size < n) size *= 2;
        tree.assign(size * 2, neutral);
    }
    segtree(int nn, T neutral) { init(nn, neutral); }

    T combine(T a, T b) {
        //Cambiar por la operacion que se necesite
        return a xor b;
    }

    void update(int i, int sl, int sr, int pos, T diff) {
        if (sl <= pos && pos <= sr) {
            if (sl == sr) {
                tree[i] = diff;
            } else {
                int mid = (sl + sr) / 2;
                update(i * 2 + 1, sl, mid, pos, diff);
                update(i * 2 + 2, mid + 1, sr, pos, diff);
                tree[i] = combine(tree[i * 2 + 1], tree[i * 2 + 2]);
            }
        }
    }

    void update(int pos, T diff) {
        update(0, 0, n - 1, pos, diff);
    }

    T query(int i, int sl, int sr, int l, int r) {
        if (l <= sl && sr <= r) {
            return tree[i];
        } else if (sr < l || r < sl) {
            return neutral;
        } else {
            int mid = (sl + sr) / 2;
            T a = query(i * 2 + 1, sl, mid, l, r);
            T b = query(i * 2 + 2, mid + 1, sr, l, r);
            return combine(a, b);
        }
    }

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

structsegtree

2 grafos

2.1 SCC

```
Dado un grafo dirigido halla las componentes fuertemente conexas (SCC).

const int inf = 1e9;
const int MX = 1e5+5; //Cantidad maxima de nodos
vector<int> g[MX]; //Lista de adyacencia
stack<int> st;
int low[MX], pre[MX], cnt;
int comp[MX]; //Almacena la componente a la que pertenece cada nodo
int SCC; //Cantidad de componentes fuertemente conexas
int n, m; //Cantidad de nodos y aristas

void tarjan(int u) {
    low[u] = pre[u] = cnt++;
    st.push(u);

    for (auto &v : g[u]) {
        if (pre[v] == -1) tarjan(v);
        low[u] = min(low[u], low[v]);
    }
    if (low[u] == pre[u]) {
        while (true) {
            int v = st.top(); st.pop();
            low[v] = inf;
            comp[v] = SCC;
            if (u == v) break;
        }
        SCC++;
    }
}

void init() {
    cnt = SCC = 0;
    for (int i = 0; i <= n; i++) {
        g[i].clear();
        pre[i] = -1; //no visitado
    }
}

// example
void test_case() {
    cin >> n >> m;
    init();
    rep(i, 0, m) {
        int x, y;
        cin >> x >> y;
        g[x].pb(y);
    }
    rep(i, 1, n + 1) {
        if (pre[i] == -1) {
            tarjan(i);
        }
    }
}
```

SCC

2.2 bfsgrillapath

```
int n, m;
vector<string> mat(1000 + 10);
vector<vector<bool>> visi(1000 + 10, vector<bool>(1000 + 10));
//se  guarda en camino[i][j] desde donde llego
vector<vector<char>> camino(1000 + 10, vector<char>(1000 + 10));
vector<pair<int, int>i> direcciones = { { -1, 0 }, { 1, 0 }, { 0, -1 }, { 0, 1 } };
vector<char> direchar = { 'U', 'D', 'L', 'R' };
bool esval(int nx, int ny) {
    return nx >= 0 && ny >= 0 && nx < n && ny < m && !visi[nx][ny] &&
           mat[nx][ny] != '#';
}
bool bfs(pair<int, int>i ini, pair<int, int>i &fin) {
    queue<pair<int, int>i> q;
    q.push(ini);
    visi[ini.first][ini.second] = true;

    while (!q.empty()) {
        auto [x, y] = q.front();
        q.pop();

        for (int i = 0; i < 4; i++) {
            int nx = x + direcciones[i].first;
            int ny = y + direcciones[i].second;

            if (esval(nx, ny)) {
                q.push({ nx, ny });
                visi[nx][ny] = true;
                camino[nx][ny] = direchar[i];
                if (mat[nx][ny] == 'B') {
                    fin = { nx, ny };
                    return true;
                }
            }
        }
    }

    return false;
}

void solve() {
    cin >> n >> m;
    pair<int, int>i ini, fin;
    FOR(i, n) {
        cin >> mat[i];
        FOR(j, m) {
            if (mat[i][j] == 'A') { ini = { i, j }; }
        }
    }
```

```cpp

    if (bfs(ini, fin)) {
        cout << "YES\n";
        string cam;
        pair<int, int>i actual = fin;
        while (mat[actual.first][actual.second] != 'A') {
            char dirr = camino[actual.first][actual.second];
            cam += dirr;
            if (dirr == 'U')
                actual.first++;
            else if (dirr == 'D')
                actual.first--;
            else if (dirr == 'L')
                actual.second++;
            else if (dirr == 'R')
                actual.second--;
        }

        reverse(all(cam));
        cout << (int)cam.size() << '\n' << cam << '\n';
    } else {
        cout << "NO\n";
    }
}
```

bfsgrillapath

2.3 bfspath

```cpp
void solve() {
    int n,m;cin>>n>>m;
    vector<int> g[n+1];
    for(int i = 0; i<m; i++){
        int a,b;cin>>a>>b;
        g[a].push_back(b);
        g[b].push_back(a);
    }
    vector<int> dist(n+1,1e9);
    vector<int> parent(n+1,-1);
    queue<int> q;
    q.push(1);
    dist[1]=0;
    while(!q.empty()){
        int u=q.front();
        q.pop();
        for(auto v:g[u]){
            if(dist[v]>dist[u]+1){
                dist[v]=dist[u]+1;
                parent[v]=u;
                q.push(v);
            }
        }
    }
    if(dist[n]==1e9){
        cout<<"IMPOSSIBLE\n";
        return;
    }
    cout<<dist[n]+1<<"\n";
    vector<int> ans;
    int u=n;
    while(u!=-1){
        ans.push_back(u);
        u=parent[u];
    }
    reverse(all(ans));
    for(auto x:ans)cout<<x<<" ";
    cout<<"\n";
}
```

bfspath

2.4 bipartitecheck

```
bool check(vector<vector<int>>& g, int n){
    vector<int> color(n + 1, -1);
    vector<bool> visi(n + 1, false);
    for (int i = 1; i <= n; i++) {
        if (!visi[i]) {
            queue<int> cola;
            cola.push(i);
            color[i] = 1;
            while (!cola.empty()) {
                int act = cola.front();
                cola.pop();
                if (visi[act]) continue;
                visi[act] = true;
                for (int &vecino : g[act]) {
                    if (color[vecino] == color[act]) {
                        return false;
                    } else if (!visi[vecino]) {
                        color[vecino] = 1 - color[act];
                        cola.push(vecino);
                    }
                }
            }
        }
    }
    return true;
}
```

bipartitecheck

2.5 ciclonegativobf

```
// This uses Bellmanford algorithm to find a negative cycle
// O(n*m) m=edges, n=nodes
void test_case() {
    ll n, m;
    cin >> n >> m;
    vector<ll> dist(n+1);
    vector<ll> p(n+1);
    vector<tuple<ll,ll,ll>> edges(m);
    for (int i =0; i < m; i ++) {
        ll x, y, z;
        cin >> x >> y >> z;
```

```cpp
        edges[i] = {x, y, z};
    }

    ll efe = -1;
    for (int i = 0; i < n; i++) {
        efe = -1;
        for (auto pp : edges) {
            ll x,y,z;
            tie(x,y,z) = pp;
            if (dist[x] + z < dist[y]) {
                dist[y] = dist[x] + z;
                p[y] = x;1
                efe = y;
            }
        }
    }
    if (efe == -1) {
        cout << "NO\n";
    } else {
        cout << "YES\n";
        ll x = efe;
        for (int i = 0; i < n; i++) {
            x = p[x];
        }
        vector<ll> cycle;
        ll y = x;
        while (cycle.size() == 0 || y != x) {
            cycle.pb(y);
            y = p[y];
        }
        cycle.pb(x);
        reverse(all(cycle));
        for (int i =0; i < cycle.size(); i++) {
            cout << cycle[i]<<’ ’;
        }
    }
}
```

ciclonegativobf

2.6 detectarciclosdirigido

```cpp
vector<vector<int>> adj(2e5+5);
vector<int> visited(2e5);
bool ok = false; // if cycle was found ok is true
vector<int> cycle;
void dfs(int x, vector<int> &st) {
    if (ok || visited[x] == 2) {
        return;
    } else if (visited[x] == 1) {
        cycle.pb(x);
        while (st.back() != x) {
            cycle.pb(st.back());
            st.pop_back();
        }
        cycle.pb(x);
        reverse(aint(cycle));
        ok = true;
        return;
    }
    visited[x] = 1;
    st.pb(x);
    for (auto y : adj[x]) {
        dfs(y, st);
    }
    st.pop_back();
    visited[x] = 2;
}

void test_case() {
    int n, m;
    cin >> n >> m;

    for (int i =0; i < m;i ++) {
        int x, y;
        cin >> x >> y;
        adj[x].pb(y);
    }

    vector<int> st;
    for (int i = 1; i <= n; i++) {
        dfs(i, st);
    }

    if (ok) {
        //haycilo e imprimir
    }
}
```

detectarciclosdirigido

2.7 dijkstra

```
vector<int> dijkstra(vector<vector<pair<int, int>>> &grafo, int n, int origen) {
    vector<int> distancia(n + 1, INF);
    distancia[origen] = 0;
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
    pq.push({0, origen});

 //   vector<int> padre(n + 1);
    while (!pq.empty()) {
        auto [peso, nodo] = pq.top();
        pq.pop();
        if (distancia[nodo] < peso) {
            continue;
        }

        for (auto vecino : grafo[nodo]) {
            int destino = vecino.first;
            int costo = vecino.second;

            if (distancia[nodo] + costo < distancia[destino]) {
                distancia[destino] = distancia[nodo] + costo;
                pq.push({distancia[destino], destino});
            //    padre[destino] = nodo;
            }
        }
    }

    return distancia;
}
```

dijkstra

2.8 floydwarshall

```
const int inf = 1e9;
vector<vector<pair<int, int>>> adj;
int distance[n][n];

void floydWarshaint() {
        for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                        distance[i][j] = inf;
                }
        }
        for (int i = 0; i < n; i++) {
                for (auto p : adj[i]) {
                        int b = p.first;
                        int w = p.second;
                        distance[i][b] = w;
                }
        }
        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]);
                        }
                }
        }
}
```

floydwarshall

2.9 kruskal

```
int kruskal(vector<tuple<int,int,int>> &edges, int nodes) {
    union_find uf(nodes+1);
    //peso, u v
    sort(all(edges));
   // reverse(aint(edges)); // for maxst
    int answer = 0;
    for (auto edge : edges)  {
        int cost, a, b;
        tie(cost, a, b) = edge;
        if (uf.conectar(a, b))
            answer += cost;
    }
    return answer;
}
```

kruskal

2.10 kthsspath

```
// Using djisktra, finds the k shortesth paths from 1 to n
// 2    n10    ^5, 1      m210      ^5, 1    weight10    ^9, 1    k10
// complexity seems O(k*m)
#define P pair<int,int>
void test_case() {
    int n, m, k;
    cin >> n >> m >> k;
    vector<int> visited(n+1, 0);
    vector<vector<pair<int,int>>> adj(n+1);
    for (int i = 0; i < m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        adj[a].pb({b, c});
    }
    vector<int> ans;
    priority_queue<P,vector<P>, greater<P>> q;
    q.push({0, 1});
    int kk = k;
    while (q.size()) {
        int x = q.top().S;
        int z = q.top().F;
        q.pop();
        if (visited[x] >= kk) {
            continue;
        }
        visited[x]++;
        if (x == n) {
            ans.pb(z);
            k--;
            if (k == 0) break;
        }
        for (auto yy : adj[x]) {
            q.push({yy.S + z, yy.F});
        }
    }
    for (int i = 0; i < ans.size(); i++) {
        cout << ans[i]<<’ ’;
    }
}
```

kthsspath

2.11 multliavabfs

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

int n, m;
vector<pair<int, int>> monstruos;
vector<vector<int>> lava(1000 + 10, vector<int>(1000 + 10, INT_MAX));
```

```
pair<int, int> inicio, fin;
// 0 = U   1 = D   2 = R    3 = L
// Estos indices se usan para calc
vector<pair<int, int>> mov = { { -1, 0 }, { 1, 0 }, { 0, 1 }, { 0, -1 } };
char calc(int x) {
    char res;
    switch (x) {
        case 0:
            res = ’U’;
            break;
        case 1:
            res = ’D’;
            break;
        case 2:
            res = ’R’;
            break;
        case 3:
            res = ’L’;
            break;
    }
    return res;
}
vector<vector<char>> direcciones(1000 + 10, vector<char>(1000 + 10, ’#’));
void reconstruircamino() {
    vector<char> res;
    auto [x, y] = fin;
    while (direcciones[x][y] != ’$’) {
        char aux = direcciones[x][y];
        res.push_back(aux);
        if (aux == ’U’) {
            x++;
        } else if (aux == ’D’) {
            x--;
        } else if (aux == ’L’) {
            y++;
        } else {
            y--;
        }
    }
    cout << (int)res.size() << ’\n’;
    reverse(res.begin(), res.end());
    for (char& it : res) { cout << it; }
    cout << ’\n’;
}
bool esvalida(int x, int y, int tiempo) {
    if (x < 0 or y < 0 or x >= n or y >= m) { return false; }
    if (lava[x][y] <= tiempo) { return false; }
    return true;
}

bool esSalida(int x, int y, int tiempo) {
    if (!esvalida(x, y, tiempo)) return false;
    if (x == 0 or y == 0 or x == n - 1 or y == m - 1) return true;
    return false;
}

bool bfsDeSalida() {
    queue<pair<pair<int, int>, int>> q;
    q.push(make_pair(inicio, 0));
    direcciones[inicio.first][inicio.second] = ’$’;
    while (!q.empty()) {
        int cx = q.front().first.first;
        int cy = q.front().first.second;
        int tiempo = q.front().second;
        tiempo++;
        q.pop();
        for (int i = 0; i < 4; i++) {
            auto mv = mov[i];
            int tx = cx + mv.first;
            int ty = cy + mv.second;
            if (esSalida(tx, ty, tiempo)) {
                direcciones[tx][ty] = calc(i);
                fin = { tx, ty };
                return true;
            }
            if (esvalida(tx, ty, tiempo)) {
                direcciones[tx][ty] = calc(i);
                lava[tx][ty] = tiempo;
                q.push({ { tx, ty }, tiempo });
            }
        }
    }
    return false;
}

void precalculoLava() {
    queue<pair<pair<int, int>, int>> q;
    for (auto m : monstruos) { q.push(make_pair(m, 0)); }
    while (!q.empty()) {
        int cx = q.front().first.first;
        int cy = q.front().first.second;
        int tiempo = q.front().second;
        tiempo++;
        q.pop();

        for (auto mv : mov) {
            int tx = cx + mv.first;
            int ty = cy + mv.second;
            if (esvalida(tx, ty, tiempo)) {
                lava[tx][ty] = tiempo;
                q.push({ { tx, ty }, tiempo });
            }
        }
    }
}

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cin >> n >> m;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            char c;
            cin >> c;
            if (c == ’#’) {
                lava[i][j] = 0;
            } else if (c == ’M’) {
                lava[i][j] = 0;

                monstruos.push_back({ i, j });
            } else if (c == ’A’) {
                lava[i][j] = 0;
                inicio = { i, j };
            } else {
                lava[i][j] = INT_MAX;
            }
        }
    }
    if (inicio.first == 0 or inicio.second == 0 or inicio.first == n - 1 or
        inicio.second == m - 1) {
        cout << "YES" << endl;
        cout << 0;
        return 0;
    }
    precalculoLava();

    if (!bfsDeSalida()) {
        cout << "NO";
        return 0;
    }
    cout << "YES" << endl;
    reconstruircamino();
}
```

multliavabfs

2.12 ssspnegativo

```
// Find the minimum distance from any i to j, with negative weights.
// dist[i][j] == -inf, there some negative loop from i to j
// dist[i][j] == inf, from i cannot reach j
// otherwise the min dist from i to j

// take care of the max a path from i to j, it has to be less than inf
const ll inf = INT32_MAX;
void test_case() {
    ll n, m; // nodes, edges
    vector<vector<ll>> dist(n, vector<ll>(n, inf));
    for (int i = 0; i < n; i++) dist[i][i] = 0;
    for (int i = 0; i < m; i++) {
        ll a, b, w;
        cin >> a >> b >> w; // negative weights
        dist[a][b] = min(dist[a][b], w);
    }
    // floid warshall
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (dist[i][k] == inf || dist[k][j] == inf) continue;
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }
    // find negative cycles for a node
    for (int i = 0; i < n; i++) {
        if (dist[i][i] < 0) dist[i][i] = -inf;
```

```cpp
    }
    // find negative cycles betweens a routes from i to j
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n ; j++) {
            for (int k = 0; k < n; k++) {
                if (dist[k][k] < 0 && dist[i][k] != inf && dist[k][j] != inf) {
                    dist[i][j] = -inf;
                }
            }
        }
    }
}
```

ssspnegativo

2.13 topsort

```cpp
const int N = 1e5;
vector<vector<ll>> adj(N + 10);
vector<ll> visited(N +10);
bool cycle = false; // reports if doesn't exists a topological sort
vector<ll> topo;

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

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

topsort

2.14 unionfind

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

unionfind

3 DP

3.1 Knapsack

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

//dp[i][j] = max(dp[i-1][j], dp[i-1][j-value])
void solve(){
     int n,x; cin>>n>>x;
     vector<int> v(n);
     vector<int> w(n);
     for(int i = 0; i < n; i++)cin>>w[i];
     for(int i = 0; i < n; i++)cin>>v[i];
     vector<vector<int>> dp(n+1, vector<int> (x+1, 0));
     for(int i = 1; i <= n; i++){
         for(int j = 0; j <= x; j++){
             int w1 = w[i-1];
             int v1 = v[i-1];
             int t = (j >= w1? dp[i-1][j-w1]+v1:0);
             int nt = dp[i-1][j];
             dp[i][j] = max(t, nt);
         }
     }
     cout<<dp[n][x]<<endl;
}

signed main() {
    solve();
    return 0;
}
```

Knapsack

3.2 KnapsackOptimization

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

 //Knapsack Optimization memory O(2*W)
void solve(){
     int n,x; cin>>n>>x;
     vector<int> v(n);
     vector<int> w(n);
     for(int i = 0; i < n; i++)cin>>w[i];
     for(int i = 0; i < n; i++)cin>>v[i];
     vector<int> ant(x+1, 0);
     for(int i = 1; i <= n; i++){
        vector<int> act(x+1, 0);
         for(int j = 0; j <= x; j++){
             int w1 = w[i-1];
             int v1 = v[i-1];
             int t = (j >= w1? ant[j-w1]+v1:0); //dp[i-1][j-value]
             int nt = ant[j]; //dp[i-1][j]
             act[j]= max(t, nt);
         }
         ant = act;
     }
     cout<<ant[x]<<endl;
}

signed main() {
    solve();
    return 0;
}
```

KnapsackOptimization

3.3 LCS

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

int const N = 1000;
int memo[N][N];
pair<int, int> path[N+1][N+1];
//-1 no calculated
int dp(string &s1, string &s2, int l, int r){
       if(l >= (int)s1.size()|| r >= (int)s2.size()){
          path[l][r] = {-1, -1}; return 0;
       }
       int &a = memo[l][r];
       if(a == -1){
          a = 0;
          if(s1[l] == s2[r]){
             path[l][r] = {l+1, r+1};
             a = dp(s1,s2,l+1, r+1)+1;
          }else{
             int op1 = dp(s1,s2,l+1,r);
             int op2 = dp(s1,s2,l,r+1);
             if(op1 > op2) path[l][r] = {l+1,r};
             else path[l][r] = {l, r+1};
             a = max(op1, op2);
          }
       }
       return a;
}

// Construct answer after DP
string construct(string &s1, string &s2) {
    int i, j;
    i = j = 0;
    string ans;
    while (i != -1 && j != -1) {
        if (s1[i] == s2[j]) {
            ans.push_back(s1[i]);
        }
        pair<int, int> aux = path[i][j];
        i = aux.first;
        j = aux.second;
    }
    return ans;
}


signed main(){
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    string s1,s2;cin>>s1>>s2;
    memset(memo, -1, sizeof(memo));
    cout<<dp(s1,s2,0,0)<<endl;
    cout<<construct(s1, s2)<<endl;
    return 0;
}
```

LCS

3.4 LIS

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


vector<int> lis(vector<int> &v){
    int n = v.size();
    vector<int> dp;
    dp.push_back(-1e9);
    vector<int> curr(n);
    for (int i = 0; i < n; i++) {
        int l = 0, r = dp.size()-1;
        int pos = dp.size();
        while (l <= r) {
            int m = l + (r-l) / 2;
            if(dp[m] >= v[i]){
                pos = m;
                r = m - 1;
            }else{
                l = m + 1;
            }
        }
        curr[i] = pos;
        if (pos == dp.size()) {
            dp.push_back(v[i]);
        } else {
            dp[pos] = v[i];
        }
    }
    vector<int> ans;
    int x = dp.size() - 1;
    for (int i = n - 1; i >= 0; i--) {
        if (curr[i] == x) {
            ans.push_back(v[i]);
            x--;
        }
    }
    reverse(ans.begin(), ans.end());
    return ans;
}


signed main() {
    ios::sync_with_stdio(0);cin.tie(0); cout.tie(0);
    int n; cin>>n;vector<int> arr(n);
    for(int i = 0; i < n; i++) cin>>arr[i];
    vector<int> ans = lis(arr);
    cout<<ans.size()<<endl;
    for(auto x: ans){
        cout<<x<<" ";
    }
    return 0;
}
```

LIS

3.5 MCM(MatrixChainMultiplicated)

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

int const N = 105;
int memo[N][N];
int dp(vector<int> &arr, int l, int r){
    if(l+1 == r) return 0;//one matrix
    if(l+2 == r) return arr[l]*arr[l+1]*arr[l+2]; //two matrix
    int &a = memo[l][r];
    if(a == -1){
       a = 1e17;
       for(int i = l+1; i < r; i++){
          int precio = arr[l]*arr[i]*arr[r];
          a = min(a, dp(arr,l,i)+dp(arr,i,r)+precio);
       }
    }
    return a;
}

signed main(){
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    int n; cin>>n;
    vector<int> arr(n);
    for(int i = 0; i < n; i++) cin>>arr[i];
    memset(memo, -1, sizeof(memo));
    cout<<dp(arr,0, n-1);
    return 0;
}
```

MCM(MatrixChainMultiplicated)

3.6 MCMaplicationn

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

int memo1[501][501];
int memo2[501][501];

int dp(int l, int r, vector<int> &arr){
    if(l+1 == r && arr[r] != arr[l]) return 0;
    if(l+1 == r && arr[r] == arr[l]) return arr[l]+1;
    if(l == r) return arr[l];
    int &a = memo1[l][r];
    if(a == -1){
       a = 0;
       for(int i = l; i < r; i++){
           memo1[l][i] = dp(l, i,arr);
           memo1[i+1][r] = dp(i+1, r,arr);
           if(memo1[l][i] == memo1[i+1][r] && memo1[l][i] != 0){
              memo1[l][r] = memo1[l][i]+1;
           }
       }
    }
    return a;
}

int dp2(int l, int r, vector<int> &arr){
    if (l == r) return 1;
    if (memo2[l][r] != -1) return memo2[l][r];
    if (memo1[l][r] != 0) return memo2[l][r] = 1;
    int mini = r - l + 1;
    for (int i = l; i < r; ++i) {
        mini = min(mini, dp2(l, i,arr) + dp2(i + 1, r,arr));
    }
    return memo2[l][r] = mini;
}

signed main(){
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
        memset(memo1, -1, sizeof(memo1));
    memset(memo2, -1, sizeof(memo2));
    int n; cin>>n;
    vector<int>arr(n+1);
    for(int i = 1; i<= n; i++)cin>>arr[i];
    dp(1, n, arr);
    cout<<dp2(1, n, arr);
        return 0;
}
```

MCMaplicationn

3.7 OptimalGridTraversal

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

using namespace std;
int const MAX_N = 100;
int const MAX_M = 100;
int G[MAX_N][MAX_M];
int n,m;
int memo[MAX_N][MAX_M];

int f(int i, int j){
    int &a = memo[i][j];
    if(a != -1) return a;
    if(i == n-1 && j == m-1) a = G[i][j];
    else{
       a = 0;
       if(i < n-1) a = max(a, G[i][j]+f(i+1,j));
       if(j < m-1) a = max(a, G[i][j]+f(i,j+1));
    }
    return a;
}

//llamar despues de llamar a la funcion f
string ans = "";
void caminoOptimo(int i, int j){
     if(i == n-1 && j == m-1) return;
     if(i < n-1 && memo[i][j] == G[i][j]+memo[i+1][j]){
        ans+='A'; //hacia abajo
        caminoOptimo(i+1,j);
     }else{
        ans += 'D'; //hacia derecha
        caminoOptimo(i,j+1);
     }
}
//igualmente despues de llamar a la funcion f
int cont[MAX_N][MAX_M];
int cant(){
    for(int i = n-1; i >= 0; --i){
        for(int j = m-1; j>= 0; --j){
            if(i == n-1 && j == m-1) cont[i][j] =1;
            else{
               cont[i][j] = 0;
               if(i < n-1 && memo[i][j] == G[i][j]+memo[i+1][j])
                 cont[i][j] += cont[i+1][j];
                if(j < m-1 && memo[i][j] == G[i][j]+memo[i][j+1])
                cont[i][j] += cont[i][j+1];
            }
        }
    }
    return cont[0][0];
}

string sol = "";
//usa el contar solucion y la funcion f
void getk(int i, int j, int k) {
    if (i == n-1 && j == m-1) return;
    if (i < n-1 && memo[i][j] == G[i][j] + memo[i+1][j]){
        if(cont[i+1][j] >= k){
            sol += 'A';
            getk(i+1, j, k);
            return;
        }
        k -= cont[i+1][j];
    }
    if (j < m-1 && memo[i][j] == G[i][j] + memo[i][j+1]) {
        sol += 'D';
        getk(i, j+1, k);
    }
}
signed main(){
       memset(memo, -1, sizeof(memo));
       cin>>n>>m;
       for(int i = 0; i < n; i++){
           for(int j = 0; j < m; j++){
               cin>>G[i][j];
           }
       }
       cout<<f(0,0)<<endl;
       caminoOptimo(0,0);
       cout<<ans<<endl;
       cout<<cant()<<endl;
       getk(0,0,3);
       cout<<sol<<endl;
}
```

OptimalGridTraversal

3.8 OptimalGridTraversalOptimization

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

using namespace std;
int const MAX_N = 100;
int const MAX_M = 100;
int G[MAX_N][MAX_M];
int n,m;
int memo[2][MAX_M];

int calc_dp_optimized(){
    for(int i = n-1; i>= 0; i--){
        for(int j = m-1; j >= 0; --j){
            if(i == n-1 && j == m-1) memo[i%2][j] = G[i][j];
            else{
              memo[i%2][j] = 0;
              if(i < n-1){
                 memo[i%2][j] = max(memo[i%2][j], G[i][j]+memo[1-i%2][j]);
              }
              if(j < m-1){
                 memo[i%2][j] = max(memo[i%2][j], G[i][j]+memo[i%2][j+1]);
              }
            }
        }
    }
    return memo[0][0];
}

signed main(){
       memset(memo, -1, sizeof(memo));
       cin>>n>>m;
       for(int i = 0; i < n; i++){
           for(int j = 0; j < m; j++){
               cin>>G[i][j];
           }
       }
       cout<<calc_dp_optimized()<<endl;
}
```

OptimalGridTraversalOptimization

3.9 SOS

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

void sos() {
    int n; cin>>n;
    vector<int> arr(1<<n);
    for(int i = 0; i <(1<<n); i++) cin>>arr[i];
    int sos[1 << n] = {0};
    for (int x = 0; x < (1 << n); x++) {
        sos[x] = arr[0];
        for (int i = x; i > 0; i = (i - 1) & x)
            sos[x] += arr[i];
    }

    for(int i = 0; i <(1<<n); i++) {
        cout<<sos[i]<< " ";
    }
}

signed main(){
    sos();
    return 0;
}
```

SOS

4 sortingsearchinggreedy

4.1 biseccionesgeneral

```
double bisection(double a, double b) {
  if (eval(a) * eval(b) >= 0) {
    // no establecemos bien los limites de evaluacion
    return -INF;
  }

  double c = a;
  while ((b - a) >= EPS) {
    // punto medio:
    c = (a + b) / 2;

    // si encontramos justo la solucion
    if (eval(c) == 0.0)
      break;

    // biseccion:
    else if (eval(c) * eval(a) < 0)
      b = c;
    else
      a = c;
  }
  // el valor de una raiz esta en c
  return c;
}
```

biseccionesgeneral

4.2 cantdesubarrsumax

```
int cantsesubarrsumaequis(vector<int>& arr, int x, int n)
{
    // Map to store the frequency of prefix sums
    map<int, int> prefSums;

    prefSums[0] = 1;
    int pref = 0;
    int cnt = 0;

    // Calculate the prefix sum at every index, and find the
    // count of subarrays with sum = pref - x
    for (int i = 0; i < n; ++i) {
        pref += arr[i];
        cnt += (prefSums[pref - x]);
        prefSums[pref]++;
    }
    return cnt;
}
```

cantdesubarrsumax

4.3 cantidaddesumasdistintas

```
//cantidad de sumas distintas de exactamente k de longitud (equivalente a cantidad de sumas ordenadas)
int dp(int n, int k, const vector<int>& arr) {
    int m = arr.size();

    // Crear una tabla DP para almacenar los resultados
    vector<vector<vector<int>>> dp(n + 1, vector<vector<int>>(k + 1, vector<int>(m + 1, 0)));

    // Caso base: Si x == 0 y k == 0, hay exactamente 1 forma (no seleccionar ning n  n mero)
    for (int i = 0; i <= m; i++) {
        dp[0][0][i] = 1;  // 1 forma de hacer suma 0 con 0 elementos
    }

    // Rellenar la tabla iterativamente
    for (int x = 1; x <= n; x++) {
        for (int kk = 1; kk <= k; kk++) {
            for (int i = 0; i < m; i++) {
                // Opci n 1: No tomar el  n mero arr[i], simplemente tomamos el valor de la fila anterior
                dp[x][kk][i + 1] = dp[x][kk][i];

                // Opci n 2: Tomar el  n mero arr[i], restamos arr[i] de x y reducimos k
                if (x >= arr[i]) {
                    dp[x][kk][i + 1] += dp[x - arr[i]][kk - 1][i];
                }
            }
        }
    }
    // La respuesta final est   en dp[n][k][m], que es la forma de sumar n con k  n meros usando todos los numeros
    return dp[n][k][m];
}


//reduccion de memoria (NO COMPROBADO)
int dp(int n, int k, const vector<int>& arr) {
    int m = arr.size();

    // Crear una tabla 2D de DP: dp[x][k] donde x es la suma y k es el  n mero de elementos seleccionados
    vector<vector<int>> dp(n + 1, vector<int>(k + 1, 0));

    // Caso base: hay exactamente 1 forma de hacer suma 0 con 0 elementos
    dp[0][0] = 1;

    // Rellenar la tabla iterativamente
    for (int i = 0; i < m; i++) {  // Iteramos sobre cada elemento en arr
        // Iteramos hacia atr s para no sobrescribir los valores de la misma iteraci n
        for (int x = n; x >= arr[i]; x--) {  // Solo consideramos sumas donde x >= arr[i]
            for (int kk = k; kk >= 1; kk--) {  // Solo consideramos hasta k elementos
                dp[x][kk] += dp[x - arr[i]][kk - 1];
            }
        }
    }

    // La respuesta final est  en dp[n][k], que es la forma de sumar n con k elementos
    return dp[n][k];
}
```

cantidaddesumasdistintas

4.4 cantsubarraydistintos

```
int cantsubarraydistinct(int* arr, int N, int K) {
    // left and right pointers to mark the start and end of
    // the sliding window
    int left = 0, right = 0;
    // Variable to count how many different numbers we have
    // in the window
    int distinct_count = 0;
    // Variable to store the final result
    int result = 0;

    // Map to keep track of how many times each number
    // appears in the window
    unordered_map<int, int> frequency;

    // Slide the window tiint the window tiint the right
    // pointer does not reach the end of the array
    while (right < N) {
        // Check if the current number is new or if its
        // count is zero
        if (frequency.find(arr[right]) == frequency.end() ||
            frequency[arr[right]] == 0)
            distinct_count++;

        // Update the count of the current number
        frequency[arr[right]]++;

        // If there are more than K distinct numbers, shrink
        // the window from the left
        while (distinct_count > K) {
            // Decrease the count of the number going out of
            // the window
            frequency[arr[left]]--;
            // If its count becomes zero, then there wiint be
            // one less distinct number in the window
            if (frequency[arr[left]] == 0) distinct_count--;
            // Move the left pointer to the right to shrink
            // the window
            left++;
        }

        // Calculate the number of subarrays that end at the
        // current position
        result += right - left + 1;

        // Move the right edge of the window to the right to
        // expand it
        right++;
    }
    // Return the result
    return result;
}
```

cantsubarraydistintos

4.5 cantsubarrdiv

```
// Function to count the number of subarrays divisible by n
int solve(vector<int>& arr, int n)
{
    // Map to store the frequency of prefix sums % n
    map<int, int> contResid;

    contResid[0] += 1;
    int residuo = 0;
    int cnt = 0;

    // Iterate over aint the index and add the count of
    // subarrays with sum divisible by n
    for (int i = 0; i < n; ++i) {
        // Since arr[i] can be negative, we add n to the
        // residuo to avoid negative residuos
        residuo = ((residuo + arr[i]) % n + n) % n;
        cnt += contResid[residuo];
        contResid[residuo] += 1;
    }
    return cnt;
}
```

cantsubarrdiv

4.6 maxsubarraysumentreab

```
void MaximumSubarraySumentreab(int N, int A, int B, vector<int>& arr) {
    // Initialize a deque to store indices in increasing
    // order of prefix sum values
    deque<int> dq;
    // Initialize a prefixSum array to store cumulative sums
    vector<int> prefixSum(N + 1);
    // Initialize the answer to track the maximum sum
    int ans = intONG_MIN;
    // Calculate cumulative sums
    for (int i = 1; i <= N; i++) {
        prefixSum[i] += prefixSum[i - 1] + arr[i - 1];
    }
    // Loop through the first (B-1) indices to initialize
    // deque
    for (int i = 1; i < B; i++) {
        // Maintain deque in increasing order of prefix sum
        // values
        while (!dq.empty() && prefixSum[dq.front()] <= prefixSum[i]) {
            dq.pop_front();
        }
        dq.push_front(i);
    }
    // Loop through each starting index i from 0 to (n-a)
    for (int i = 0; i <= (N - A); i++) {
        // Maintain deque in increasing order of prefix sum
        // values
        while (i + B <= N && !dq.empty() &&
               prefixSum[dq.front()] <= prefixSum[i + B]) {
            dq.pop_front();
        }
        // Push the right end index to the front of deque
        if (i + B <= N) dq.push_front(i + B);
        // If the index of maximum element outside the
        // current window , pop elements from the back of
        // the deque until the back index(index of maximum
        // element) is within the current window.
        while (!dq.empty() && dq.back() < (A + i)) { dq.pop_back(); }
        // Update the answer by taking the maximum of the
        // current answer and the difference between the
        // prefix sum at the back(maximum element) of the
        // deque and the prefix sum at index i
        ans = max(ans, prefixSum[dq.back()] - prefixSum[i]);
    }
    // Print the final answer
    cout << ans << "\n";
}
```

maxsubarraysumentreab

4.7 secuencialargaunica

```
// la longitud del subarray con numeros sucesivos mas larga (sin repetir
// numeros)
int LI-SUBARRAYsinrepetir(int n, vector<int> arr) {
    int l = 0, ans = 0;
    // mapa para guardar la ultima ocurrencia de un numero
    map<int, int> mp;

    // two pointers
    for (int r = 0; r < n; r++) {
        // Si el elemento actual no esta en el mapa
        if (mp.find(arr[r]) == mp.end())
            mp.insert({ arr[r], r });
        else {
            // if el numero actual esta en el mapa y esta en la ventana
            if (mp[arr[r]] >= l) l = mp[arr[r]] + 1;
            // actualizar la ultima ocurrencia del caracter en el indice actual
            mp[arr[r]] = r;
        }
        //maximizar la respuesta
        ans = max(ans, r - l + 1);
    }
    return ans;
}
```

secuencialargaunica

4.8 sumade4valroes

```
// function to find a quadruplet whose sum = X
void suma4(vector<int> &arr, int X, int N) {
    // vector to store the values along with their indices
    vector<vector<int>> vec(N, vector<int>(2));

    for (int i = 0; i < N; i++) {
        vec[i][0] = arr[i];
        vec[i][1] = i + 1;
    }

    // Sort the vector in increasing order of the values
    sort(vec.begin(), vec.end());

    // Iterate for aint possible values of first element
    for (int ptr1 = 0; ptr1 < N - 3; ptr1++) {
        // Iterate for aint possible values of second element
        for (int ptr2 = ptr1 + 1; ptr2 < N - 2; ptr2++) {
            // Maintain two pointers for the third and
            // fourth element
            int ptr3 = ptr2 + 1;
            int ptr4 = N - 1;
            while (ptr3 < ptr4) {
                int currentSum =
                    vec[ptr1][0] + vec[ptr2][0] + vec[ptr3][0] + vec[ptr4][0];
                if (currentSum == X) {
                    cout << vec[ptr1][1] << " " << vec[ptr2][1] << " "
                         << vec[ptr3][1] << " " << vec[ptr4][1] << "\n";
                    return;
                }
                // Decrease the currentSum by moving ptr4 to
                // ptr4 - 1
                else if (currentSum > X) {
                    ptr4--;
                } else if (currentSum < X) {
                    ptr3++;
                }
            }
        }
    }
    cout << "IMPOSSIBLE";
}
```

sumade4valroes

5 Math

5.1 convexHullGrahamScan

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

using namespace std;
struct Point {
    int x, y;
};

// to  the first point Used in compare function of qsort()
Point p0;

// A utility function to find next to top in a stack
Point nextToTop(stack<Point> &S) {
    Point p = S.top();
    S.pop();
    Point res = S.top();
    S.push(p);
    return res;
}

// A utility function to swap two points
void swap(Point &p1, Point &p2) {
    Point temp = p1;
    p1 = p2;
    p2 = temp;
}

// A utility function to return square of distance
// between p1 and p2
int distSq(Point p1, Point p2) {
    return (p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y);
}

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are collinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r) {
    int val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);

    if (val == 0)
        return 0;             // collinear
    return (val > 0) ? 1 : 2; // clock or counterclock wise
}

// A function used by library function qsort() to sort an array of
// points with respect to the first point
int compare(const void *vp1, const void *vp2) {
    Point *p1 = (Point *)vp1;
    Point *p2 = (Point *)vp2;

    // Find orientation
    int o = orientation(p0, *p1, *p2);
    if (o == 0)
```

```
        return (distSq(p0, *p2) >= distSq(p0, *p1)) ? -1 : 1;

    return (o == 2) ? -1 : 1;
}

// Prints convex hull of a set of n points.
void convexHull(Point points[], int n) {
    // Find the bottommost point
    int ymin = points[0].y, min = 0;
    for (int i = 1; i < n; i++) {
        int y = points[i].y;

        // Pick the bottom-most or choose the left
        // most point in case of tie
        if ((y < ymin) || (ymin == y && points[i].x < points[min].x))
            ymin = points[i].y, min = i;
    }

    // Place the bottom-most point at first position
    swap(points[0], points[min]);

    // Sort n-1 points with respect to the first point.
    // A point p1 comes before p2 in sorted output if p2
    // has larger polar angle (in counterclockwise
    // direction) than p1
    p0 = points[0];
    qsort(&points[1], n - 1, sizeof(Point), compare);

    // If two or more points make same angle with p0,
    // Remove all but the one that is farthest from p0
    // Remember that, in above sorting, our criteria was
    // to keep the farthest point at the end when more than
    // one points have same angle.
    int m = 1; // Initialize size of modified array
    for (int i = 1; i < n; i++) {
        // Keep removing i while angle of i and i+1 is same
        // with respect to p0
        while (i < n - 1 && orientation(p0, points[i], points[i + 1]) == 0)
            i++;

        points[m] = points[i];
        m++; // Update size of modified array
    }

    // If modified array of points has less than 3 points,
    // convex hull is not possible
    if (m < 3)
        return;

    // Create an empty stack and push first three points
    // to it.
    stack<Point> S;
    S.push(points[0]);
    S.push(points[1]);
    S.push(points[2]);

    // Process remaining n-3 points
    for (int i = 3; i < m; i++) {
        // Keep removing top while the angle formed by
        // points next-to-top, top, and points[i] makes
        // a non-left turn
        while (S.size() > 1 &&
               orientation(nextToTop(S), S.top(), points[i]) != 2)
            S.pop();
        S.push(points[i]);
    }
}
void solve() {
    Point points[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4},
                      {0, 0}, {1, 2}, {3, 1}, {3, 3}};
    int n = sizeof(points) / sizeof(points[0]);
    convexHull(points, n);
}

signed main() {
    std::ios::sync_with_stdio(false);
    cin.tie(0);
    int t;

    t = 1;
    // memset(dp, -1, sizeof(dp));
    //  cin >> t;
    while (t--) {
        solve();
    }
}
void solve() {
    //
}

signed main() {
    std::ios::sync_with_stdio(false);
    cin.tie(0);
    int t;

    t = 1;
    // memset(dp, -1, sizeof(dp));
    //  cin >> t;
    while (t--) {
        solve();
    }
}
```

convexHullGrahamScan

5.2 convexHullJarvisMarch

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

using namespace std;

/**
.---------------------------------------------------.
/
                                                                     /
/
                                                                          /
```

```
|
                                                         |
|
                                   |
|
                                                                                  |
|
                                                |
'---------------------------------------------------'
**/

// A C++ program to find convex hull of a set of points. Refer
// https://www.geeksforgeeks.org/orientation-3-ordered-points/
// for explanation of orientation()

struct Point {
    int x, y;
};

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are collinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r) {
    int val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);

    if (val == 0)
        return 0;             // collinear
    return (val > 0) ? 1 : 2; // clock or counterclock wise
}

// Prints convex hull of a set of n points.
void convexHull(Point points[], int n) {
    // There must be at least 3 points
    if (n < 3)
        return;

    // Initialize Result
    vector<Point> hull;

    // Find the leftmost point
    int l = 0;
    for (int i = 1; i < n; i++)
        if (points[i].x < points[l].x)
            l = i;

    // Start from leftmost point, keep moving counterclockwise
    // until reach the start point again. This loop runs O(h)
    // times where h is number of points in result or output.
    int p = l, q;
    do {
        // Add current point to result
        hull.push_back(points[p]);

        // Search for a point 'q' such that orientation(p, q,
        // x) is counterclockwise for all points 'x'. The idea
        // is to keep track of last visited most counterclock-
        // wise point in q. If any point 'i' is more counterclock-
        // wise than q, then update q.
        q = (p + 1) % n;
        for (int i = 0; i < n; i++) {
            // If i is more counterclockwise than current q, then
            // update q
            if (orientation(points[p], points[i], points[q]) == 2)
                q = i;
        }

        // Now q is the most counterclockwise with respect to p
        // Set p as q for next iteration, so that q is added to
        // result ’hull’
        p = q;

    } while (p != l); // While we don’t come to first point

    // Print Result
    for (int i = 0; i < hull.size(); i++)
        cout << "(" << hull[i].x << ", " << hull[i].y << ")\n";
}

// Driver program to test above functions
int main() {
    Point points[] = {{0, 3}, {2, 2}, {1, 1}, {2, 1}, {3, 0}, {0, 0}, {3, 3}};
    int n = sizeof(points) / sizeof(points[0]);
    convexHull(points, n);
    return 0;
}
```

convexHullJarvisMarch

6 AlgoritmosGeneral

6.1 Combinatorics

```
#include <bits/stdc++.h>
#define int long long

using namespace std;

const int MOD =  1e9 +7;
const int N = 200005;
int fact[N];
int invfact[N];

int binpow(int a, int b, int m) {
    int res = 1;
    while (b > 0) {
        if (b & 1) {
            res = (res * a) % m;
        }
        a = (a * a) % m;
        b >>= 1;
    }
    return res;
}

```

```
int inversoFermat(int a, int m){
    return binpow(a, m-2, m);
}

void procesar(){
     fact[0] = fact[1] = 1;
     invfact[0] = invfact[1] = inversoFermat(1, MOD);
     for(int i = 2; i < N; i++){
         fact[i] = i*fact[i-1]%MOD;
         invfact[i] = inversoFermat(fact[i], MOD);
     }
}

int nCk(int n, int k){
    if(k == n) return 1;
    if(k > n) return 0;
    int res = fact[n] * invfact[n-k] % MOD * invfact[k] % MOD;
    return res;
}

signed main(){
    int n,k; cin>>n>>k;
    procesar();
    cout<<nCk(n,k);
}
```

Combinatorics

6.2 TernarySearch

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

//   ar vector que contiene a la funcion

void ternarySearchMaximo() {
    int l = 0;
    r = n - 1, m1, m2;
    while (r - l > 2) {
        m1 = l + (r - l) / 3;
        m2 = r - (r - l) / 3;

        int fm1 = ar[m1]; // resultado funcion evaluada en m1
        int fm2 = ar[m2];

        if (fm1 == fm2) {
            l = m1;
            r = m2;

        } else if (fm1 < fm2) {
            l = m1;
        } else {
            r = m2;
        }
    }
    int ans = INT_MIN;
    for (int i = l; i <= r; i++) {
        if (ar[i] > ans) {
            ans = ar[i];
        }
    }
}
void ternarySearchMin() {
    int l = 0;
    r = n - 1, m1, m2;
    while (r - l > 2) {
        m1 = l + (r - l) / 3;
        m2 = r - (r - l) / 3;

        int fm1 = ar[m1]; // resultado funcion evaluada en m1
        int fm2 = ar[m2];

        if (fm1 == fm2) {
            l = m1;
            r = m2;

        } else if (fm1 < fm2) {
            l = m1;
        } else {
            r = m2;
        }
    }
    int ans = INT_MAX;
    for (int i = l; i <= r; i++) {
        if (ar[i] < ans) {
            ans = ar[i];
        }
    }
}
```

TernarySearch

6.3 TernarySearchReales

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

using namespace std;

/**
.---------------------------------------------------.
/
                                                                                   /
/
                                                                                         /
/
                                                                    /
/
                                          /
/
                                                                                                   /
```

```
/
                                                /
'---------------------------------------------------'
**/

double f(double x) {
    return 0;
    // escribir la funcion;
}

void ternarySearchMaximo() {
    double eps = 1e-6;
    int l = -100, r = 100, m1, m2;
    while (r - l > 2) {
        m1 = l + (r - l) / 3;
        m2 = r - (r - l) / 3;

        int fm1 = ar[m1]; // resultado funcion evaluada en m1
        int fm2 = ar[m2];

        if (fm1 == fm2) {
            l = m1;
            r = m2;

        } else if (fm1 < fm2) {
            l = m1;
        } else {
            r = m2;
        }
    }
    int ans = INT_MIN;
    for (int i = l; i <= r; i++) {
        if (ar[i] > ans) {
            ans = ar[i];
        }
    }
}
i
```

TernarySearchReales

7 datastructures

7.1 FenwikTree

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


int lsb(int x) {return x & -x;}
template <typename T>
struct FenwickTree{
    //indexado a 1
       vector<T> bit;
        FenwickTree(int N): bit(N+1, 0) {};
       void add(int i, T val){
             while(i < bit.size()){
                    bit[i] = (bit[i]+val)%MOD;
                    i += lsb(i);
              }
        }
        T sum(int i){
              T ret = 0;
              while(i > 0){
               ret = (ret +  bit[i]) %MOD;
               i -= lsb(i);
              }
              return ret;
        }
        T sum(int l, int r){
          return sum(r)-sum(l-1);
        }
};

signed main(){
     return 0;
}
```

FenwikTree