# ICPC MX 2025 Reference

**Enrique Calderón, Luis Salazar, Gustavo Valenzuela**

Last Updated: September 5, 2025

# Table of Contents

# 1 Data Types

| Type | Range | Bytes |
|---|---|---|
| bool | true or false | 1 bit |
| signed char | -128 to 127 | 1 |
| unsigned char | 0 to 255 | 1 |
| short int | -32,768 to 32,767 | 2 |
| unsigned short int | 0 to 65,535 | 2 |
| unsigned int | 0 to 4,294,967,295 | 4 |
| int | -2,147,483,648 to 2,147,483,647 | 4 |
| long int | -2,147,483,648 to 2,147,483,647 | 4 |
| unsigned long int | 0 to 4,294,967,295 | 4 |
| long long int | -(2ˆ63) to (2ˆ63)-1 | 8 |
| unsigned long long int | 0 to 18,446,744,073,709,551,615 | 8 |
| float | -3.4E38 to 3.4E38 | 4 |
| double | -1.7E308 to 1.7E308 | 8 |
| long double | -1.1E4932 to 1.1E4932 | 12 |

# 2 General algorithms

## 2.1 Sparse Table

### 2.1.1 Prerequisites

- Immutable array
- Associative function for O(n log n) range query
- Overlap friendly function O(1) range query (max, min, gcd, lcm)

### 2.1.2 Implementation

```cpp
class SparseTable {
    vector<vector<int>> st;
    vector<int> logs;

public:
    SparseTable(vector<int>& arr) {
        int n = arr.size();
        int maxLog = 0;
        while ((1 << maxLog) <= n) maxLog++;

        st.assign(maxLog, vector<int>(n));
        logs.assign(n + 1, 0);

        // Precompute logs
        for (int i = 2; i <= n; i++) {
            logs[i] = logs[i / 2] + 1;
        }

        // Fill first row
        for (int i = 0; i < n; i++) {
            st[0][i] = arr[i];
        }

        // Fill table
        for (int j = 1; j < maxLog; j++) {
            for (int i = 0; i + (1 << j) <= n; i++) {
                st[j][i] = max(st[j-1][i], st[j-1][i + (1 << (j-1)
                    )]);
            }
        }
    }

    // O(1) range maximum query
    int query(int l, int r) {
        int j = logs[r - l + 1];
        return max(st[j][l], st[j][r - (1 << j) + 1]);
    }
};
```

# 3 Geometry

## 3.1 Constants

### 3.1.1 PI

```cpp
#define PI acos(-1.0)
```

### 3.1.2 Epsilon

```cpp
#define EPS 1e-9
```

## 3.2 Conversions

### 3.2.1 Degree/Radian conversions

```cpp
double DEG_to_RAD(double d) { return d * PI / 180.0; }
double RAD_to_DEG(double r) { return r * 180.0 / PI; }
```

## 3.3 Structures

### 3.3.1 Point

```cpp
struct point_i {
    int x, y;
    point_i() { x = y = 0; }
    point_i(int _x, int _y) : x(_x), y(_y) {}
};

struct point {
    double x, y;
    point() { x = y = 0.0; }
    point(double _x, double _y) : x(_x), y(_y) {}
};
```

### 3.3.2 Line

```cpp
struct line {
    double a, b, c;
    // ax + by + c = 0
};
```

## 3.4 Circle

### 3.4.1 Check if point is inside circle

```cpp
int insideCircle(point_i p, point_i c, int r) {
    int dx = p.x - c.x, dy = p.y - c.y;
    int Euc = dx * dx + dy * dy, rSq = r * r;
    return Euc < rSq ? 0 : Euc == rSq ? 1 : 2;
    // 0 = inside, 1 = on border, 2 = outside
}
```

### 3.4.2 Arc Length

To calculate the arc use `L = r * theta` where theta is in radians.

## 3.5 Triangle

### 3.5.1 Area using Heron's formula

```cpp
double triangleArea(double a, double b, double c) {
    double s = (a + b + c) / 2.0; // semiperimeter
    return sqrt(s * (s - a) * (s - b) * (s - c));
}
```

### 3.5.2 Distance between points

```cpp
double dist(point p1, point p2) {
    return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (
        p1.y - p2.y));
}
```

### 3.5.3 Perimeter

```cpp
double perimeter(double a, double b, double c) {
    return a + b + c;
}

double perimeter(point a, point b, point c) {
    return dist(a, b) + dist(b, c) + dist(c, a);
}
```

## 3.6 Save int as real number

For more precision you can use scanf:

```cpp
int a, b;
scanf("%d.%d", &a, &b);
```

# 4 C++ Functions

## 4.1 Common STL Algorithms

### Sorting Algorithms

| Function | Parameters | Description |
|---|---|---|
| sort | begin, end, [comp] | Standard unstable sort (O(n log n)) |
| stable_sort | begin, end, [comp] | Stable sort preserves element order |
| is_sorted | begin, end, [comp] | Checks if range is sorted (returns bool) |
| nth_element | begin, nth, end, [comp] | Partitions around nth element |

## Searching Functions

| Function | Parameters | Description |
|---|---|---|
| lower_bound | begin, end, val, [comp] | First element ≤ value |
| upper_bound | begin, end, val, [comp] | First element ¿ value |
| binary_search | begin, end, val, [comp] | Existence check in sorted range |
| find | begin, end, val | Linear search for value |
| find_if | begin, end, pred | Find first matching predicate |

## Sequence Operations

| Function | Parameters | Description |
|---|---|---|
| reverse | begin, end | Reverse elements in-place |
| rotate | begin, mid, end | Rotate elements left |
| next_permutation | begin, end | Generate next permutation |
| unique | begin, end, [pred] | Remove consecutive duplicates |
| remove | begin, end, val | Remove elements equal to value |

## Numerical Functions

| Function | Parameters | Description |
|---|---|---|
| accumulate | begin, end, init, [op] | Sum/accumulate elements |
| partial_sum | begin, end, dest, [op] | Compute prefix sums |
| __gcd | a, b | Greatest common divisor (C++17) |
| lcm | a, b | Least common multiple (C++17) |
| iota | begin, end, val | Fill with consecutive values |

## Memory/Array Operations

| Function | Parameters | Description |
|---|---|---|
| memset | ptr, value, count | Fill memory with byte value |
| fill | begin, end, value | Fill range with value |
| fill_n | begin, count, value | Fill N elements with value |
| copy | src_b, src_e, dest | Copy range to destination |
| copy_if | src_b, src_e, dest, pred | Copy elements matching predicate |

## Utility Functions

| Function | Parameters | Description |
|---|---|---|
| swap | a, b | Swap two values |
| max_element | begin, end, [comp] | Find maximum element |
| min_element | begin, end, [comp] | Find minimum element |
| count | begin, end, val | Count element occurrences |
| all_of | begin, end, pred | Check all elements satisfy condition |

## Mathematical / Bitwise Builtins

| Function | Parameters | Description |
|---|---|---|
| __builtin_popcount | x (int) | Count number of 1-bits |
| __builtin_popcountll | x (long long) | Count number of 1-bits (64-bit) |
| __builtin_clz | x (unsigned int) | Count leading zeros |
| __builtin_clzll | x (unsigned long long) | Leading zeros (64-bit) |
| __builtin_ctz | x (unsigned int) | Count trailing zeros |
| __builtin_ctzll | x (unsigned long long) | Trailing zeros (64-bit) |
| __builtin_parity | x | Return 1 if #bits is odd |
| __builtin_ffs | x | Position of least significant 1-bit (1-indexed) |
| __lg | x | Floor of $\log_2(x)$ (index of highest bit) |

## Priority Queues and Heaps

| Function | Parameters | Description |
|---|---|---|
| priority_queue | [type], [container], [comp] | Max-heap by default |
| make_heap | begin, end, [comp] | Turn range into heap |
| push_heap | begin, end, [comp] | Push element into heap |
| pop_heap | begin, end, [comp] | Pop max element into end |
| sort_heap | begin, end, [comp] | Heap sort |

### Set / Map Utilities

| **Operation** | **Usage** | **Description** |
|---|---|---|
| s.lower_bound(x) | set/map | First element $\geq$ x |
| s.upper_bound(x) | set/map | First element > x |
| s.equal_range(x) | multiset/map | Pair of lower/upper bound |
| s.erase(it) | iterator | Erase element at iterator |
| s.find(x) | key | Iterator to key or end |

### String Functions

| **Function** | **Parameters** | **Description** |
|---|---|---|
| stoi, stol, stoll | string, [pos], [base] | Convert string $\rightarrow$ int/long/ll |
| stoul, stoull | string, [pos], [base] | Convert string $\rightarrow$ unsigned |
| stod, stof, stold | string | Convert string $\rightarrow$ double/float/-long double |
| to_string | value | Convert number $\rightarrow$ string |
| substr | pos, len | Substring |
| find | str, pos | Find first occurrence |
| rfind | str, pos | Find last occurrence |

### Random Number Utilities

| **Type / Function** | **Usage** | **Description** |
|---|---|---|
| mt19937 rng | chrono::steady_clock::now() | Fast random generator |
| uniform_int_distribution | dist(a,b)(rng) | Random int in [a,b] |
| shuffle | begin, end, rng | Random shuffle |

### Other Useful Utilities

| **Function** | **Parameters** | **Description** |
|---|---|---|
| chrono::high_resolution_clock | now() | Get precise current time |
| __int128 | value | 128-bit integer (manual I/O needed) |
| bitset¡N¿ | ops: &, —, ˆ, ¡¡, ¿¿ | Fixed-size bitset manipulation |
| tuple | get$< i >$(t) | Store and access heterogenous data |
| pair | first, second | Store pair of values |

## 5 Binary search in the answer

```cpp
// Standard binary search (iterative)
int binary_search(vector<int>& arr, int target) {
    int left = 0, right = arr.size() - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) return mid;
        if (arr[mid] < target) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}

// Lower bound (first element >= target)
int lower_bound(vector<int>& arr, int target) {
    int left = 0, right = arr.size();
    while (left < right) {
        int mid = left + (right - left) / 2;
        arr[mid] < target ? left = mid + 1
                          : right = mid;
    }
    return left;
}

// Upper bound (first element > target)
int upper_bound(vector<int>& arr, int target) {
    int left = 0, right = arr.size();
    while (left < right) {
        int mid = left + (right - left) / 2;
        arr[mid] <= target ? left = mid + 1
                           : right = mid;
    }
    return left;
```

```cpp
}

// Binary search on real numbers (e.g. sqrt)
double sqrt_precision(double n, double eps=1e-6) {
    double left = 0, right = n;
    for (int i = 0; i < 100; ++i) { // or while (right-left > eps)
        double mid = (left + right) / 2;
        if (mid*mid < n) left = mid;
        else right = mid;
    }
    return left;
}

// Binary search on answer space (monotonic condition)
int find_min_valid(vector<int>& nums, int k) {
    auto is_valid = [&](int x) {
        /* condition check */
    };

    int left = 0, right = 1e9; // adjust bounds
    while (left < right) {
        int mid = left + (right - left) / 2;
        is_valid(mid) ? right = mid
                      : left = mid + 1;
    }
    return left;
}
```

# 6 Data Structures

## 6.1 Fenwick Tree

```cpp
struct FenwickTree {
    vector<int> bit;  // binary indexed tree
    int n;

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

    FenwickTree(vector<int> const &a) : FenwickTree(a.size()){
    for (int i = 0; i < n; i++) {
        bit[i] += a[i];
        int r = i | (i + 1);
        if (r < n) bit[r] += bit[i];
    }
}

    FenwickTree(vector<int> const &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;
    }
};
```

## 6.2 Fenwick Minimum

```cpp
struct FenwickTreeMin {
    vector<int> bit;
    int n;
    const int INF = (int)1e9;

    FenwickTreeMin(int n) {
        this->n = n;
        bit.assign(n, INF);
    }

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

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

    void update(int idx, int val) {
        for (; idx < n; idx = idx | (idx + 1))
```

```
            bit[idx] = min(bit[idx], val);
    }
};
```

## 6.3 1-Indexed Fenwick Tree

```
struct FenwickTreeOneBasedIndexing {
    vector<int> bit;  // binary indexed tree
    int n;

    FenwickTreeOneBasedIndexing(int n) {
        this->n = n + 1;
        bit.assign(n + 1, 0);
    }

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

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

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

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

## 6.4 Fenwick 2D (Sum query)

```
struct Fenwick2D {
    vector<vector<int>> tree;
    int rows, cols;

    Fenwick2D(int r, int c) : rows(r), cols(c),
        tree(r + 1, vector<int>(c + 1)) {}

    // Update: add delta to (x, y) (1-based)
    void update(int x, int y, int delta) {
        for(int i = x; i <= rows; i += lsb(i))
            for(int j = y; j <= cols; j += lsb(j))
                tree[i][j] += delta;
    }

    // Query sum from (1,1) to (x,y)
    int query(int x, int y) {
        int sum = 0;
        for(int i = x; i > 0; i -= lsb(i))
            for(int j = y; j > 0; j -= lsb(j))
                sum += tree[i][j];
        return sum;
    }

    // Range sum from (x1,y1) to (x2,y2)
    int range_query(int x1, int y1, int x2, int y2) {
        return query(x2, y2) - query(x1-1, y2)
             - query(x2, y1-1) + query(x1-1, y1-1);
    }

    int lsb(int i) { return i & -i; }
};
```

## 6.5 Fenwick 2D (Counting in range)

```
struct Fenwick2DPerType {
    int rows, cols;
    unordered_map<int, Fenwick2D> trees;  // Map from type to 2D
        Fenwick Tree

    Fenwick2DPerType(int r, int c) : rows(r), cols(c) {}

    // Update: add 'delta' objects of type 't' at position (x, y)
    void update(int t, int x, int y, int delta) {
        if (trees.find(t) == trees.end()) {
            trees[t] = Fenwick2D(rows, cols);
        }
        trees[t].update(x, y, delta);
    }

    // Query: count of type 't' in rectangle [x1,y1] to [x2,y2]
    int query(int t, int x1, int y1, int x2, int y2) {
        if (trees.find(t) == trees.end()) return 0;
        return trees[t].range_query(x1, y1, x2, y2);
    }
};

// Requires the base Fenwick2D implementation from previous answer
```

```cpp
struct Fenwick2D {
    vector<vector<int>> tree;
    int rows, cols;

    Fenwick2D(int r, int c) : rows(r), cols(c),
        tree(r + 1, vector<int>(c + 1)) {}

    void update(int x, int y, int delta) { /* same as before */ }

    int query(int x, int y) { /* same as before */ }

    int range_query(int x1, int y1, int x2, int y2) { /* same as
        before */ }

    int lsb(int i) { return i & -i; }
};
```

## 6.6 Fenwick Tree Range Update - Point Query

```cpp
// Range Update - Point Query (1-based indexing)
struct FenwickRUQ {
    int n;
    std::vector<int> bit;

    FenwickRUQ(int size) : n(size + 1), bit(size + 2) {}

    // Add val to range [l, r] (1-based)
    void range_add(int l, int r, int val) {
        add(l, val);
        add(r + 1, -val);
    }

    // Get value at position idx (1-based)
    int point_query(int idx) {
        int res = 0;
        for(; idx > 0; idx -= idx & -idx)
            res += bit[idx];
        return res;
    }

private:
    void add(int idx, int val) {
        for(; idx < n; idx += idx & -idx)
            bit[idx] += val;
    }
};
```

## 6.7 Fenwick Tree - Range update and query

```cpp
// Range Update - Range Query (1-based indexing)
struct FenwickRURQ {
    int n;
    std::vector<int> B1, B2;

    FenwickRURQ(int size) : n(size + 1), B1(size + 2), B2(size +
        2) {}

    // Add val to range [l, r] (1-based)
    void range_add(int l, int r, int val) {
        add(B1, l, val);
        add(B1, r + 1, -val);
        add(B2, l, val * (l - 1));
        add(B2, r + 1, -val * r);
    }

    // Get sum of range [l, r] (1-based)
    int range_sum(int l, int r) {
        return prefix_sum(r) - prefix_sum(l - 1);
    }

private:
    void add(std::vector<int>& b, int idx, int val) {
        for(; idx < n; idx += idx & -idx)
            b[idx] += val;
    }

    int sum(const std::vector<int>& b, int idx) {
        int total = 0;
        for(; idx > 0; idx -= idx & -idx)
            total += b[idx];
        return total;
    }

    int prefix_sum(int idx) {
        return sum(B1, idx) * idx - sum(B2, idx);
    }
};
```

## 6.8 Segment Tree (Iterative)

```cpp
int segtree[2*100000 + 5];

    void build(vector<int> &arr, int n){
        for(int i=0; i<n; i++)
            segtree[n+i] = arr[i];

```

```
        for(int i=n-1; i>=1; i--)
            segtree[i] = max(segtree[2*i], segtree[2*i+1]);
    }

    void update(int pos, int value, int n){
        pos+=n;
        segtree[pos] = value;

        while(pos>1){
            pos>>=1;
            segtree[pos] = max(segtree[2*pos],segtree[2*pos+1]);
        }
    }

    int query(int l, int r, int n){
        l+=n;
        r+=n;

        int mx = INT_MIN;

        while(l <= r){
            if(l % 2 == 1) mx = max(mx, segtree[l++]);
            if(r % 2 == 0) mx = max(mx, segtree[r--]);
            l >>= 1;
            r >>= 1;
        }

        return mx;
    }
```

## 6.9 Segment Tree (Sum query)

```
ll  t[4*MAX];

// Shout-out to CP algo for the SegTree implementation: https://cp-algorithms.com/data_structures/segment_tree.html#memory-efficient-implementation

void buildSegTree(vector<ll> &a, int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = a[tl];
    } else {
        int tm = (tl + tr) / 2;
        buildSegTree(a, v*2, tl, tm);
        buildSegTree(a, v*2+1, tm+1, tr);
        t[v] = t[v*2] + t[v*2+1];
    }
}


ll sum(int v, int tl, int tr, int l, int r) {
    if (l > r)
        return 0;
    if (l == tl && r == tr) {
        return t[v];
    }
    int tm = (tl + tr) / 2;
    return sum(v*2, tl, tm, l, min(r, tm))
           + sum(v*2+1, tm+1, tr, max(l, tm+1), r);
}

void update(int v, int tl, int tr, int pos, ll new_val) {
    if (tl == tr) {
        t[v] = new_val;
    } else {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v*2, tl, tm, pos, new_val);
        else
            update(v*2+1, tm+1, tr, pos, new_val);
        t[v] = t[v*2] + t[v*2+1];
    }
}
```

## 6.10 Segment Tree (Minimum query)

```
ll  t[4*MAX];

// Shout-out to CP algo for the SegTree implementation: https://cp-algorithms.com/data_structures/segment_tree.html#memory-efficient-implementation

void buildSegTree(vector<ll> &a, int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = a[tl];
    } else {
        int tm = (tl + tr) / 2;
        buildSegTree(a, v*2, tl, tm);
        buildSegTree(a, v*2+1, tm+1, tr);
        t[v] = min(t[v*2], t[v*2+1]); // Change to minimum
    }
}


ll query(int v, int tl, int tr, int l, int r) {
    if (l > r)
        return LLONG_MAX; // Return maximum possible value for empty range
```

```cpp
    if (l == tl && r == tr) {
        return t[v];
    }
    int tm = (tl + tr) / 2;
    return min(query(v*2, tl, tm, l, min(r, tm)),
               query(v*2+1, tm+1, tr, max(l, tm+1), r));
}

void update(int v, int tl, int tr, int pos, ll new_val) {
    if (tl == tr) {
        t[v] = new_val;
    } else {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v*2, tl, tm, pos, new_val);
        else
            update(v*2+1, tm+1, tr, pos, new_val);
        t[v] = min(t[v*2], t[v*2+1]); // Change to minimum
    }
}
```

## 6.11 Segment Tree Lazy Propagation

```cpp
typedef long long ll;
typedef vector<int> vec;
typedef vector<pair<int,int>> vpii;
const ll mod=1e9+7;
const int MAX=1e5+3;
const int limit=2e5+3;
const int TAM=2e5+1;
ll t[4*TAM];
ll op[4*TAM];
int type[4*TAM];
//ascii https://elcodigoascii.com.ar/

void propagate(int root,int l,int r)
{
    if(type[root]==1)
    {
        t[root]+=op[root]*(r+1-l);
        if(l!=r){
            op[2*root]+=op[root];
            op[2*root+1]+=op[root];
            type[2*root+1]=max(1,type[2*root+1]);
            type[2*root]=max(1,type[2*root]);
        }
    }
    else
    {
        if(type[root]==2){
            t[root]=op[root]*(r+1-l);
            if(l!=r){
                op[2*root]=op[root];
                op[2*root+1]=op[root];
                type[2*root+1]=2;
                type[2*root]=2;
            }
        }
    }
    op[root]=0;
    type[root]=0;
}

void build(int root,int l,int r,vector<ll> &arr)
{
    if(l==r)
    {
        t[root]=arr[l];
        op[root]=0;
        type[root]=0;
        return;
    }
    int mid=(l+r)/2;
    build(2*root,l,mid,arr);
    build(2*root+1,mid+1,r,arr);
    t[root]=t[2*root]+t[2*root+1];
    op[root]=0;
    type[root]=0;
}

void sum(int root,int l,int r,int a,int b,ll val)
{
    propagate(root,l,r);
    if(a>b) return;
    if(l==a && r==b)
    {
        op[root]=val;
        type[root]=1;
        propagate(root,l,r);
        return;
    }
    int mid=(l+r)/2;
    sum(2*root,l,mid,a,min(b,mid),val);
    sum(2*root+1,mid+1,r,max(mid+1,a),b,val);
    t[root]=t[2*root]+t[2*root+1];
}

void setR(int root,int l,int r,int a,int b,ll val)
{
    propagate(root,l,r);
```

```
    if(a>b) return;
    if(l==a && r==b)
    {
        op[root]=val;
        type[root]=2;
        propagate(root,l,r);
        return;
    }
    int mid=(l+r)/2;
    setR(2*root,l,mid,a,min(b,mid),val);
    setR(2*root+1,mid+1,r,max(mid+1,a),b,val);
    t[root]=t[2*root]+t[2*root+1];
}

ll consult(int root,int l,int r, int a,int b)
{
    propagate(root,l,r);
    if(a>b) return 0;
    if(l==a && r==b){
        return t[root];
    }
    int mid=(l+r)/2;
    return consult(2*root,l,mid,a,min(b,mid))+
    consult(2*root+1,mid+1,r,max(mid+1,a),b);
}
```

## 6.12 Segment Tree 2D

```
typedef long long ll;
typedef vector<int> vec;
const ll mod=1e9+7;
const int TAM=1e3+1;
//ascii https://elcodigoascii.com.ar/
vector<vector<int>> forest(TAM,vector<int> (TAM));
ll t[4*TAM][4*TAM];
int n;

void buildNode(int root,int l,int r,int node,vector<int> &arr){
    if(l==r)
    {
        t[node][root]=arr[l];
        return;
    }
    int mid=(l+r)/2;
    buildNode(2*root,l,mid,node,arr);
    buildNode(2*root+1,mid+1,r,node,arr);
    t[node][root]=t[node][2*root]+t[node][2*root+1];
}

void build(int root,int l,int r,vector<vector<int>> &arr)
{
    if(l==r)
    {
        buildNode(1,0,n-1,root,arr[l]);
        return;
    }
    int mid=(l+r)/2;
    build(2*root,l,mid,arr);
    build(2*root+1,mid+1,r,arr);
    FO(i,4*TAM) t[root][i]=t[2*root][i]+t[2*root+1][i];

}

void updateNode(int root,int l,int r,int y,int node,int val)
{
    if(l==r)
    {
        t[node][root]=val;
        return;
    }
    int mid=(l+r)/2;
    if(y>mid)
    {
        updateNode(2*root+1,mid+1,r,y,node,val);
    }
    else{
        updateNode(2*root,l,mid,y,node,val);
    }
    t[node][root]=t[node][2*root]+t[node][2*root+1];
}

void update(int root,int l,int r,int x,int y,int val)
{
    if(l==r)
    {
        updateNode(1,0,n-1,y,root,val);
        return;
    }
    int mid=(l+r)/2;
    if(x>mid)
    {
        update(2*root+1,mid+1,r,x,y,val);
    }
    else{
        update(2*root,l,mid,x,y,val);
    }
    int i=0,j=n-1,Ndt=1,mid_aux;
    while(i!=j)
    {
        mid_aux=(i+j)/2;
```

```
        t[root][Ndt]=t[2*root][Ndt]+t[2*root+1][Ndt];
        if(y>mid_aux){
            i=mid_aux+1;
            Ndt=2*Ndt+1;
        }
        else{
            j=mid_aux;
            Ndt*=2;
        }
    }
    t[root][Ndt]=t[2*root][Ndt]+t[2*root+1][Ndt];
}

ll consultNode(int root,int l,int r,int node,int y1,int y2)
{
    if(y1>y2) return 0;
    if(l==y1 && r==y2) return t[node][root];
    int mid=(l+r)/2;
    return consultNode(2*root,l,mid,node,y1,min(y2,mid))+
    consultNode(2*root+1,mid+1,r,node,max(mid+1,y1),y2);
}

ll consult(int root,int l,int r, int x1,int x2,int y1,int y2)
{
    if(x1>x2) return 0;
    if(l==x1 && r==x2) return consultNode(1,0,n-1,root,y1,y2);
    int mid=(l+r)/2;
    return consult(2*root,l,mid,x1,min(x2,mid),y1,y2)+
    consult(2*root+1,mid+1,r,max(mid+1,x1),x2,y1,y2);
}
```

## 6.13 Segment tree with Index Compression

```
typedef long long ll;
typedef vector<int> vec;
typedef vector<pair<int,int>> vpii;
const ll mod=1e9+7;
const int MAX=4e5+3;
const int limit=2e5+3;
const int TAM=2e5+1;
ll t[4*MAX];
//ascii https://elcodigoascii.com.ar/


void update(int root,int l,int r,int pos,int val)
{
    if(l==r)
    {
        t[root]+=val;
        return;
    }
    int mid=(l+r)/2;
    if(pos>mid)
    {
        update(2*root+1,mid+1,r,pos,val);
    }
    else{
        update(2*root,l,mid,pos,val);
    }
    t[root]=t[2*root]+t[2*root+1];
}

ll consult(int root,int l,int r, int a,int b)
{
    if(a>b) return 0;
    if(l==a && r==b) return t[root];
    int mid=(l+r)/2;
    return consult(2*root,l,mid,a,min(b,mid))+
    consult(2*root+1,mid+1,r,max(mid+1,a),b);
}

inline void solve()
{
   int n,m,index;
   cin>>n>>m;
   vector<ll> arr(n);
   vector<tuple<char,ll,ll>> queries(m);
   set<ll> salary;
   memset(t,0,sizeof(t));
   FO(i,n){
        ll aux; cin>>aux;
        arr[i]=aux;
        salary.insert(aux);
   }
   FO(i,m)
   {
        char a;
        ll b,c;
        cin>>a>>b>>c;
        queries[i]=make_tuple(a,b,c);
        if(a=='!') salary.insert(c);
   }

   vector<ll> coord(all(salary));
   int tn=coord.size();
   //FO(i,tn) cout<<coord[i]<<" ";
   //cout<<endl;
   FO(i,n)
   {
        index=lower_bound(all(coord),arr[i])-coord.begin();
```

```
        update(1,0,tn-1,index,1);
    }
    FO(i,m)
    {
        char a=get<0>(queries[i]);
        ll b=get<1>(queries[i]);
        ll c=get<2>(queries[i]);
        if(a=='?'){
            b=lower_bound(all(coord),b)-coord.begin();
            c=(upper_bound(all(coord),c)-coord.begin())-1;
            if(b==tn || c==tn ){
                cout<<0<<endl;
            }
            else cout<<consult(1,0,tn-1,b,c)<<endl;
        }
        else{
            index=lower_bound(all(coord),arr[b-1])-coord.begin();
            update(1,0,tn-1,index,-1);
            arr[b-1]=c;
            index=lower_bound(all(coord),arr[b-1])-coord.begin();
            update(1,0,tn-1,index,1);

        }
    }

}
```

## 6.14 Segment Tree Preffix-Suffix-Biggest

```
typedef long long ll;
typedef vector<int> vec;
typedef vector<pair<int,int>> vpii;
const ll mod=1e9+7;
const int MAX=1e5+3;
const int limit=2e5+3;
const int TAM=2e5+1;
ll t[4*TAM];
ll prefix[4*TAM],suffix[4*TAM],biggest[4*TAM];
//ascii https://elcodigoascii.com.ar/
ll cero=0;
void build(int root,int l,int r,vector<ll> &arr)
{
    if(l==r)
    {
        t[root]=arr[l];
        suffix[root]=max(t[root],cero);
        prefix[root]=max(t[root],cero);
        biggest[root]=max(t[root],cero);
        return;
    }
    int mid=(l+r)/2;
    build(2*root,l,mid,arr);
    build(2*root+1,mid+1,r,arr);
    t[root]=t[2*root]+t[2*root+1];
    biggest[root]=max(biggest[2*root],
    max(biggest[2*root+1],suffix[2*root]+prefix[2*root+1]));
    prefix[root]=max(prefix[2*root],t[2*root]+prefix[2*root+1]);
    suffix[root]=max(suffix[2*root+1],t[2*root+1]+suffix[2*root]);

}

void update(int root,int l,int r,int pos,ll val)
{
    if(l==r)
    {
        t[root]=val;
        suffix[root]=max(cero,t[root]);
        prefix[root]=max(cero,t[root]);
        biggest[root]=max(t[root],cero);
        return;
    }
    int mid=(l+r)/2;
    if(pos>mid)
    {
        update(2*root+1,mid+1,r,pos,val);
    }
    else{
        update(2*root,l,mid,pos,val);
    }
    t[root]=t[2*root]+t[2*root+1];
    biggest[root]=max(biggest[2*root],
    max(biggest[2*root+1],suffix[2*root]+prefix[2*root+1]));
    prefix[root]=max(prefix[2*root],t[2*root]+prefix[2*root+1]);
    suffix[root]=max(suffix[2*root+1],t[2*root+1]+suffix[2*root]);
}

ll consult(int root,int l,int r, int a,int b)
{
    if(a>b) return 0;
    if(l==a && r==b) return t[root];
    int mid=(l+r)/2;
    return consult(2*root,l,mid,a,min(b,mid))+
    consult(2*root+1,mid+1,r,max(mid+1,a),b);
}
```

## 6.15 Persistent Array

```
vector<pair<int, int>> arr[100001];  // The persistent array
```

```cpp

int get_item(int index, int time) {
    // Gets the array item at a given index and time
    auto ub =
        upper_bound(arr[index].begin(), arr[index].end(), make_pair(time, INT_MAX));
    return prev(ub)->second;
}

void update_item(int index, int value, int time) {
    // Updates the array item at a given index and time
    // Note that this only works if the time is later than all previous
    // update times
    assert(arr[index].back().first < time);
    arr[index].push_back({time, value});
}

void init_arr(int n, int *init) {
    // Initializes the persistent array, given an input array
    for (int i = 0; i < n; i++) arr[i].push_back({0, init[i]});
}
```

## 6.16 Path Copying - Persistent Array

```cpp
struct Node {
    int val;
    Node *l, *r;

    Node(ll x) : val(x), l(nullptr), r(nullptr) {}
    Node(Node *ll, Node *rr) : val(0), l(ll), r(rr) {}
};

int n, a[100001];      // The initial array and its size
Node *roots[100001];  // The persistent array's roots

Node *build(int l = 0, int r = n - 1) {
    if (l == r) return new Node(a[l]);
    int mid = (l + r) / 2;
    return new Node(build(l, mid), build(mid + 1, r));
}

Node *update(Node *node, int val, int pos, int l = 0, int r = n - 1) {
    if (l == r) return new Node(val);
    int mid = (l + r) / 2;
    if (pos > mid) return new Node(node->l, update(node->r, val, pos, mid + 1, r));
    else return new Node(update(node->l, val, pos, l, mid), node->r);
}

int query(Node *node, int pos, int l = 0, int r = n - 1) {
    if (l == r) return node->val;
    int mid = (l + r) / 2;
    if (pos > mid) return query(node->r, pos, mid + 1, r);
    return query(node->l, pos, l, mid);
}

int get_item(int index, int time) {
    // Gets the array item at a given index and time
    return query(roots[time], index);
}

void update_item(int index, int value, int prev_time, int curr_time) {
    // Updates the array item at a given index and time
    roots[curr_time] = update(roots[prev_time], index, value);
}

void init_arr(int nn, int *init) {
    // Initializes the persistent array, given an input array
    n = nn;
    for (int i = 0; i < n; i++) a[i] = init[i];
    roots[0] = build();
}
```

## 6.17 Persistent Segment Tree

```cpp
using ll = long long;

class PersistentSegtree {
  private:
    struct Node {
        ll sum = 0;
        int l = 0, r = 0;
    };

    const int n;
    vector<Node> tree;
    int timer = 1;

    Node join(int l, int r) { return Node{tree[l].sum + tree[r].sum, l, r}; }

    int build(int tl, int tr, const vector<int> &arr) {
        if (tl == tr) {
```

```cpp
            tree[timer] = {arr[tl], 0, 0};
            return timer++;
        }

        int mid = (tl + tr) / 2;
        tree[timer] = join(build(tl, mid, arr), build(mid + 1, tr,
             arr));

        return timer++;
    }

    int set(int v, int pos, int val, int tl, int tr) {
        if (tl == tr) {
            tree[timer] = {val, 0, 0};
            return timer++;
        }

        int mid = (tl + tr) / 2;
        if (pos <= mid) {
            tree[timer] = join(set(tree[v].l, pos, val, tl, mid),
                tree[v].r);
        } else {
            tree[timer] = join(tree[v].l, set(tree[v].r, pos, val,
                 mid + 1, tr));
        }

        return timer++;
    }

    ll range_sum(int v, int ql, int qr, int tl, int tr) {
        if (qr < tl || tr < ql) { return 0ll; }
        if (ql <= tl && tr <= qr) { return tree[v].sum; }

        int mid = (tl + tr) / 2;
        return range_sum(tree[v].l, ql, qr, tl, mid) +
               range_sum(tree[v].r, ql, qr, mid + 1, tr);
    }

  public:
    PersistentSegtree(int n, int MX_NODES) : n(n), tree(MX_NODES)
        {}

    int build(const vector<int> &arr) { return build(0, n - 1, arr
        ); }

    int set(int root, int pos, int val) { return set(root, pos,
        val, 0, n - 1); }

    ll range_sum(int root, int l, int r) { return range_sum(root,
        l, r, 0, n - 1); }

    int add_copy(int root) {
        tree[timer] = tree[root];
        return timer++;
    }
};
```

## 6.18 Policy Ordered Set

```cpp
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp>
#include <functional> // for less
using namespace __gnu_pbds;

// To allow repetitions
typedef tree<int, null_type, less<int>, rb_tree_tag,
             tree_order_statistics_node_update>
    ordered_set;

// To not allow repetitions
typedef tree<pair<int, int>, null_type,
             less<pair<int, int> >, rb_tree_tag,
             tree_order_statistics_node_update>
    ordered_multiset;

ordered_set pt; // Definition

pt.order_of_key(x); // Number of items strictly smaller than x
pt.find_by_order(k); // Iterator to the kth element
```

## 6.19 Disjoint Set Union

```cpp
// Shout-out to Usaco Guide for DSU implementation: https://usaco.
    guide/gold/dsu?lang=cpp

class DisjointSets{
    private:
        vector<int> parents;
        vector<int> sizes;
        int components;
    public:
        DisjointSets(int size) : parents(size), sizes(size,1),
            components(size){
            for(int i=0; i<size; i++){parents[i] = i;}
        }

        int find(int x) {return parents[x] == x ? x : (parents[x]
            = find(parents[x]));}

```

```cpp
        bool unite(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;
            components--;
            return true;
        }

        vector<int> getAllComponentSizes(){
            map<int, int> component_sizes;
            for (int i = 0; i < parents.size(); ++i){
                int root = find(i);
                if (component_sizes.find(root) == component_sizes.
                    end()){
                    component_sizes[root] = sizes[root];
                }
            }

            vector<int> result;
            for (auto& [root, size] : component_sizes) {
                result.push_back(size);
            }

            return result;
        }


        bool connected(int x, int y) { return find(x) == find(y);}
        int getSize(int x) {return sizes[find(x)];}
        int getComponents() const {return components;}
};
```

## 6.20 DSU to detect cycles

```cpp
class CycleDetectionDSU {
    vector<int> parent;
    vector<int> size;

public:
    CycleDetectionDSU(int n) : parent(n), size(n, 1) {
        iota(parent.begin(), parent.end(), 0);
    }

    int find(int x) {
        return parent[x] == x ? x : parent[x] = find(parent[x]);
    }

    // Returns true if adding edge u-v creates a cycle
    bool add_edge(int u, int v) {
        int u_root = find(u);
        int v_root = find(v);
        if (u_root == v_root) return true;

        if (size[u_root] < size[v_root]) swap(u_root, v_root);
        parent[v_root] = u_root;
        size[u_root] += size[v_root];
        return false;
    }
};
```

## 6.21 DSU to check online bipartitness

```cpp
class BipartiteDSU {
    vector<int> parent;
    vector<int> size;

public:
    BipartiteDSU(int n) : parent(2*n), size(2*n, 1) {
        iota(parent.begin(), parent.end(), 0);
    }

    int find(int x) {
        return parent[x] == x ? x : parent[x] = find(parent[x]);
    }

    // Returns true if graph remains bipartite after adding u-v
    bool add_edge(int u, int v) {
        int u_orig = 2*u;      // Original node
        int u_mirror = 2*u+1;  // Mirror node
        int v_orig = 2*v;
        int v_mirror = 2*v+1;

        // Union u_orig <-> v_mirror and v_orig <-> u_mirror
        for(int i = 0; i < 2; i++) {
            int x = i ? v_orig : u_orig;
            int y = i ? u_mirror : v_mirror;

            int x_root = find(x);
            int y_root = find(y);
            if (x_root != y_root) {
                if (size[x_root] < size[y_root]) swap(x_root,
                    y_root);
```

```
                parent[y_root] = x_root;
                size[x_root] += size[y_root];
            }
        }

        // Check if u is in both partitions
        return find(u_orig) != find(u_mirror);
    }
};


// -- Other implementation --

void make_set(int v) {
    parent[v] = make_pair(v, 0);
    rank[v] = 0;
    bipartite[v] = true;
}

pair<int, int> find_set(int v) {
    if (v != parent[v].first) {
        int parity = parent[v].second;
        parent[v] = find_set(parent[v].first);
        parent[v].second ^= parity;
    }
    return parent[v];
}

void add_edge(int a, int b) {
    pair<int, int> pa = find_set(a);
    a = pa.first;
    int x = pa.second;

    pair<int, int> pb = find_set(b);
    b = pb.first;
    int y = pb.second;

    if (a == b) {
        if (x == y)
            bipartite[a] = false;
    } else {
        if (rank[a] < rank[b])
            swap (a, b);
        parent[b] = make_pair(a, x^y^1);
        bipartite[a] &= bipartite[b];
        if (rank[a] == rank[b])
            ++rank[a];
    }
}

bool is_bipartite(int v) {
    return bipartite[find_set(v).first];
}
```

## 6.22 DSU with rollback

```
class DSU {
  private:
    vector<int> p, sz;
    // stores previous unites
    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); }

        // save this unite operation
        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();
        }
    }
};
```

## 6.23 Dynamic connectivity

```
struct dsu_save {
    int v, rnkv, u, rnku;

    dsu_save() {}

    dsu_save(int _v, int _rnkv, int _u, int _rnku)
```

```cpp
        : v(_v), rnkv(_rnkv), u(_u), rnku(_rnku) {}
};

struct dsu_with_rollbacks {
    vector<int> p, rnk;
    int comps;
    stack<dsu_save> op;

    dsu_with_rollbacks() {}

    dsu_with_rollbacks(int n) {
        p.resize(n);
        rnk.resize(n);
        for (int i = 0; i < n; i++) {
            p[i] = i;
            rnk[i] = 0;
        }
        comps = n;
    }

    int find_set(int v) {
        return (v == p[v]) ? v : find_set(p[v]);
    }

    bool unite(int v, int u) {
        v = find_set(v);
        u = find_set(u);
        if (v == u)
            return false;
        comps--;
        if (rnk[v] > rnk[u])
            swap(v, u);
        op.push(dsu_save(v, rnk[v], u, rnk[u]));
        p[v] = u;
        if (rnk[u] == rnk[v])
            rnk[u]++;
        return true;
    }

    void rollback() {
        if (op.empty())
            return;
        dsu_save x = op.top();
        op.pop();
        comps++;
        p[x.v] = x.v;
        rnk[x.v] = x.rnkv;
        p[x.u] = x.u;
        rnk[x.u] = x.rnku;
    }
};

struct query {
    int v, u;
    bool united;
    query(int _v, int _u) : v(_v), u(_u) {
    }
};

struct QueryTree {
    vector<vector<query>> t;
    dsu_with_rollbacks dsu;
    int T;

    QueryTree() {}

    QueryTree(int _T, int n) : T(_T) {
        dsu = dsu_with_rollbacks(n);
        t.resize(4 * T + 4);
    }

    void add_to_tree(int v, int l, int r, int ul, int ur, query& q
        ) {
        if (ul > ur)
            return;
        if (l == ul && r == ur) {
            t[v].push_back(q);
            return;
        }
        int mid = (l + r) / 2;
        add_to_tree(2 * v, l, mid, ul, min(ur, mid), q);
        add_to_tree(2 * v + 1, mid + 1, r, max(ul, mid + 1), ur, q
            );
    }

    void add_query(query q, int l, int r) {
        add_to_tree(1, 0, T - 1, l, r, q);
    }

    void dfs(int v, int l, int r, vector<int>& ans) {
        for (query& q : t[v]) {
            q.united = dsu.unite(q.v, q.u);
        }
        if (l == r)
            ans[l] = dsu.comps;
        else {
            int mid = (l + r) / 2;
            dfs(2 * v, l, mid, ans);
            dfs(2 * v + 1, mid + 1, r, ans);
        }
        for (query q : t[v]) {
            if (q.united)
```

```
                dsu.rollback();
        }
    }

    vector<int> solve() {
        vector<int> ans(T);
        dfs(1, 0, T - 1, ans);
        return ans;
    }
}
```

## 6.24 Trie

```
class TrieNode
{
  public:
    // Array for children nodes of each node
    TrieNode *children[26];

    // for end of word
    bool isLeaf;

    TrieNode()
    {
        isLeaf = false;
        for (int i = 0; i < 26; i++)
        {
            children[i] = nullptr;
        }
    }

};
  // Method to insert a key into the Trie
void insert(TrieNode *root, const string &key)
{

    // Initialize the curr pointer with the root node
    TrieNode *curr = root;

    // Iterate across the length of the string
    for (char c : key)
    {

        // Check if the node exists for the
        // current character in the Trie
        if (curr->children[c - 'a'] == nullptr)
        {

            // If node for current character does
            // not exist then make a new node
            TrieNode *newNode = new TrieNode();

            // Keep the reference for the newly
            // created node
            curr->children[c - 'a'] = newNode;
        }

        // Move the curr pointer to the
        // newly created node
        curr = curr->children[c - 'a'];
    }

    // Mark the end of the word
    curr->isLeaf = true;
}

// Method to search a key in the Trie
bool search(TrieNode *root, const string &key)
{

    if (root == nullptr)
    {
        return false;
    }

    // Initialize the curr pointer with the root node
    TrieNode *curr = root;

    // Iterate across the length of the string
    for (char c : key)
    {

        // Check if the node exists for the
        // current character in the Trie
        if (curr->children[c - 'a'] == nullptr)
            return false;

        // Move the curr pointer to the
        // already existing node for the
        // current character
        curr = curr->children[c - 'a'];
    }

    // Return true if the word exists
    // and is marked as ending
    return curr->isLeaf;
}

// Method to check if a prefix exists in the Trie
bool isPrefix(TrieNode *root, const string &prefix)
```

```cpp
{
    // Initialize the curr pointer with the root node
    TrieNode *curr = root;

    // Iterate across the length of the prefix string
    for (char c : prefix)
    {
        // Check if the node exists for the current character in the Trie
        if (curr->children[c - 'a'] == nullptr)
            return false;

        // Move the curr pointer to the already existing node
        // for the current character
        curr = curr->children[c - 'a'];
    }

    // If we reach here, the prefix exists in the Trie
    return true;
  }
```

## 6.25 Palindromic Tree

```cpp
const int MAXN = 105000;

struct node {
    int next[26];
    int len;
    int sufflink;
    int num;
};

int len;
char s[MAXN];
node tree[MAXN];
int num;            // node 1 - root with len -1, node 2 - root with len 0
int suff;           // max suffix palindrome
long long ans;

bool addLetter(int pos) {
    int cur = suff, curlen = 0;
    int let = s[pos] - 'a';

    while (true) {
        curlen = tree[cur].len;
        if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])
            break;
        cur = tree[cur].sufflink;
    }
    if (tree[cur].next[let]) {
        suff = tree[cur].next[let];
        return false;
    }

    num++;
    suff = num;
    tree[num].len = tree[cur].len + 2;
    tree[cur].next[let] = num;

    if (tree[num].len == 1) {
        tree[num].sufflink = 2;
        tree[num].num = 1;
        return true;
    }

    while (true) {
        cur = tree[cur].sufflink;
        curlen = tree[cur].len;
        if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
            tree[num].sufflink = tree[cur].next[let];
            break;
        }
    }

    tree[num].num = 1 + tree[tree[num].sufflink].num;

    return true;
}

void initTree() {
    num = 2; suff = 2;
    tree[1].len = -1; tree[1].sufflink = 1;
    tree[2].len = 0; tree[2].sufflink = 1;
}

// -- Other implementation --

const int maxn = 1e5, sigma = 26;

int s[maxn], len[maxn], link[maxn], to[maxn][sigma];

int n, last, sz;

void init()
{
    s[n++] = -1;
    link[0] = 1;
```

```cpp
    len[1] = -1;
    sz = 2;
}

int get_link(int v)
{
    while(s[n - len[v] - 2] != s[n - 1]) v = link[v];
    return v;
}

void add_letter(int c)
{
    s[n++] = c;
    last = get_link(last);
    if(!to[last][c])
    {
        len [sz] = len[last] + 2;
        link[sz] = to[get_link(link[last])][c];
        to[last][c] = sz++;
    }
    last = to[last][c];
}
```

## 6.26 Implicit Treap

```cpp
using namespace std;

#include<random>
#include<chrono>

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

#define ll long long

struct TreapNode{
    ll key, pr, sz;
    TreapNode *l, *r;
};

typedef TreapNode* Treap;

int getSize(Treap &t){
    return t ? t->sz : 0;
}

void updateSize(Treap &t){
    if (t) t->sz = 1 + getSize(t->l) + getSize(t->r);
}

void split(Treap& t, ll k, Treap &l, Treap  &r){
    if(not t) l = r = nullptr;

    else if(k < t->key){
        split(t->l,k,l, t->l);
        r = t;
        updateSize(r);
    }else{
        split(t->r,k,t->r,r);
        l = t;
        updateSize(l);
    }
}

void insert(Treap& t, Treap a){
    if(not t) t=a;
    else if(a->pr > t->pr){
        split(t, a->key, a->l, a->r);
        t = a;
    }else{
        if(a->key < t-> key) insert(t->l,a);
        else insert(t->r,a);
    }
    updateSize(t);
}

void merge(Treap &t, Treap l, Treap r){
    if(not l) t = r;
    else if(not r) t = l;

    else if(l->pr > r->pr){
        merge(l->r, l->r,r);
        t=l;
        updateSize(t);
    }else{
        merge(r->l,l,r->l);
        t=r;
        updateSize(t);
    }
}

void erase(Treap &t, ll k){
    if(not t) return;
    if(t->key == k) merge(t,t->l, t->r);

    else{
        if(k<t->key) erase(t->l,k);
        else erase(t->r, k);
    }
    updateSize(t);
```

```cpp
}

bool find(Treap& t, ll k){
    if (not t) return false;
    if(t->key == k) return true;
    if(k<t->key) return find(t->l,k);
    return find(t->r,k);
}

void insertValue(Treap &t, ll k){
    if(not find(t,k)){
        Treap new_node = new TreapNode {k,rng(), 0,nullptr,
            nullptr};
        insert(t, new_node);
    }
}

ll getKth(Treap &t, int k){
    if(!t || k<=0 || k>getSize(t)) return 0;
    int leftSize = getSize(t->l);
    if(k == leftSize+1) return t->key;
    if(k <= leftSize) return getKth(t->l,k);
    return getKth(t->r, k-leftSize-1);
}
```

## 6.27 Treap

```cpp
typedef struct item * pitem;
struct item {
    int prior, value, cnt;
    bool rev;
    pitem l, r;
};

int cnt (pitem it) {
    return it ? it->cnt : 0;
}

void upd_cnt (pitem it) {
    if (it)
        it->cnt = cnt(it->l) + cnt(it->r) + 1;
}

void push (pitem it) {
    if (it && it->rev) {
        it->rev = false;
        swap (it->l, it->r);
        if (it->l)  it->l->rev ^= true;
        if (it->r)  it->r->rev ^= true;
    }
}

void merge (pitem & t, pitem l, pitem r) {
    push (l);
    push (r);
    if (!l || !r)
        t = l ? l : r;
    else if (l->prior > r->prior)
        merge (l->r, l->r, r),  t = l;
    else
        merge (r->l, l, r->l),  t = r;
    upd_cnt (t);
}

void split (pitem t, pitem & l, pitem & r, int key, int add = 0) {
    if (!t)
        return void( l = r = 0 );
    push (t);
    int cur_key = add + cnt(t->l);
    if (key <= cur_key)
        split (t->l, l, t->l, key, add),  r = t;
    else
        split (t->r, t->r, r, key, add + 1 + cnt(t->l)),  l = t;
    upd_cnt (t);
}

void reverse (pitem t, int l, int r) {
    pitem t1, t2, t3;
    split (t, t1, t2, l);
    split (t2, t2, t3, r-l+1);
    t2->rev ^= true;
    merge (t, t1, t2);
    merge (t, t, t3);
}

void output (pitem t) {
    if (!t)  return;
    push (t);
    output (t->l);
    printf ("%d ", t->value);
    output (t->r);
}
```

# 7 Graph Theory

## 7.1 Bipartite Check BFS

```cpp
bool bfs(int s){
    queue<int> q;
    q.push(s);
    color[s] = 1; // Assign the initial color

    while(!q.empty()){
        int u = q.front();
        q.pop();

        // Check all adjacent vertices of u
        for(auto v : adj[u]){
            // If v is not colored yet
            if(color[v] == 0){
                color[v] = (color[u] == 1) ? 2 : 1;
                q.push(v);
            }
            else if (color[v] == color[u]){
                return false;
            }
        }
    }
    return true;
}
```

## 7.2 Cycle Detection DFS

```cpp
// Thanks CP-Algo for Cycle finding implementation: https://cp-
    algorithms.com/graph/finding-cycle.html

bool dfs(int v, int par) { // passing vertex and its parent vertex
    visited[v] = true;
    for (int u : adj[v]) {
        if(u == par) continue; // skipping edge to parent vertex
        if (visited[u]) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
        parent[u] = v;
        if (dfs(u, parent[u]))
            return true;
    }
    return false;
}

void find_cycle() {
    visited.assign(n+1, false);
    parent.assign(n+1, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (!visited[v] && dfs(v, parent[v]))
            break;
    }

    if (cycle_start == -1) {
        cout << "IMPOSSIBLE" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);

        cout << cycle.size()<<endl;;
        for (int v : cycle)
            cout << v << " ";
        cout << endl;
    }
}
```

## 7.3 Topological Sort

```cpp
vector<int> ans;

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

void topological_sort() {
    visited.assign(n+1, false);
    ans.clear();
    for (int i = 1; i <= n; ++i) {
        if (!visited[i]) {
            dfs(i);
        }
    }
    reverse(ans.begin(), ans.end());
}
```

## 7.4 Kahn's Algorithm

```python
def kahnTopoSort(self,adj: List[List[int]]) -> List[int]:
        #print(adj)
        in_deg = [0] * len(adj)
        for i in range(len(adj)):
            for u in adj[i]:
                in_deg[u]+=1

        q = []
        for i in range(len(in_deg)):
            if in_deg[i] == 0:
                q.append(i)

        arns = []
        while len(q)>0:
            u = q[0]
            q.pop(0)
            arns.append(u)

            for v in adj[u]:
                in_deg[v]-=1
                if in_deg[v] == 0:
                    q.append(v)

        print(str(len(arns))+" "+str(len(adj)))
        if(len(arns) != len(adj)):
            return []

        return arns
```

## 7.5 Lexicographically Min. TopoSort

```cpp
int n;
vector<vector<int>> adj(MAX);
vector<int> in_degree(MAX);
vector<int> group_ids(MAX);
vector<int> ans;

//topological sort implementation: https://cp-algorithms.com/graph
    /topological-sort.html

void topological_sort() {
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater
        <pair<int, int>>> pq;

    for(int i = 1; i <= n; i++) {
        if(in_degree[i] == 0) {
            pq.emplace(group_ids[i], i);
        }
    }

    while(!pq.empty()) {
        int u = pq.top().second;
        pq.pop();
        ans.push_back(u);

        for(int v : adj[u]) {
            in_degree[v]--;
            if(in_degree[v] == 0) {
                pq.emplace(group_ids[v], v);
            }
        }
    }

}
```

## 7.6 BFS Flood Fill

```cpp
bool validate(int x, int y){
    if(vis[x][y]) return false;
    if(maze[x][y] == '#') return false;
    if(x<0 or x>=n or y<0 or y>=m) return false;
    return true;
}

bool solveMaze(int x, int y){
    queue<pii> q;
    q.push(mp(x,y));
    vis[x][y] = true;

    int dx[] = {1, -1, 0, 0};
    int dy[] = {0, 0, 1, -1};
    char move_dir[] = {'D', 'U', 'R', 'L'};

    while(!q.empty()){
        int u =  q.front().fs;
        int v = q.front().sc;
        q.pop();

        if(maze[u][v] == 'B'){
            while(true){
                res.push_back(path[u][v]);

                if(res.back() == 'U' && u + 1 < n) u++;
                if(res.back() == 'D' && u - 1 >= 0) u--;
                if(res.back() == 'L' && v + 1 < m) v++;
                if(res.back() == 'R' && v - 1 >= 0) v--;

                if(u == x and v ==y) break;
```

```
            }
            return true;
        }
        for (int i = 0; i < 4; ++i) {
            int new_u = u + dx[i];
            int new_v = v + dy[i];
            if (validate(new_u, new_v)) {
                path[new_u][new_v] = move_dir[i];
                vis[new_u][new_v] = true;
                q.push(mp(new_u, new_v));
            }
        }
    }
    return false;
}
```

## 7.7 BFS Iterative Flood Fill

```
void floodFill(int x, int y, char color ,int r, int c) {
    if (maze[x][y] == color) return;
    queue<pii> q;
    q.push(pii(x, y));
    while (!q.empty()) {
        pii currentCoor = q.front();
        q.pop();
        x = currentCoor.fi;
        y = currentCoor.sc;
        if (x >= 0 && x < r && y >= 0 && y < c && maze[x][y] !=
            color) {
            maze[x][y] = color;
            q.push(pii(x + 1, y));
            q.push(pii(x - 1, y));
            q.push(pii(x, y + 1));
            q.push(pii(x, y - 1));
        }
    }
}
```

## 7.8 DFS Flood Fill

```
void floodFill(int x, int y, char color,vector<vector<char>>&
    board){
    if(x<0 or y<0 or x>=board.size() or y>=board[x].size() or
        board[x][y] != 'O') return;
    board[x][y] = color;
    floodFill(x+1,y,color,board);
    floodFill(x-1,y,color,board);
    floodFill(x,y+1,color,board);
    floodFill(x,y-1,color,board);
}
```

## 7.9 Lava Flow (Multi-source BFS)

```
struct Cell{
    int x,y,t;
};

const int MAX = 1005;
int n,m;

char maze[MAX][MAX];
int vis[MAX][MAX];
int player[MAX][MAX];
char path[MAX][MAX];
set<pii> isExit;
queue<Cell> q;
string res;

bool isValid(int x, int y){
    if(x < 0 || x >= n || y < 0 || y >= m) return false;
    if(maze[x][y] == '#') return false;
    return true;
}

bool isSafe(int x, int y, int u, int v){
    return player[x][y] == -1 and maze[x][y] != 'M' and (vis[x][y]
         == -1 or player[u][v] + 1 < vis[x][y]);
}


void restorePath(int u, int v, int x, int y){
     while (x != u || y != v) {
        res.push_back(path[u][v]);

        if (res.back() == 'U') u++;
        if (res.back() == 'D') u--;
        if (res.back() == 'L') v++;
        if (res.back() == 'R') v--;
    }
}

bool lavaFlow(int x,int y){
        q.push({x,y,1});
        player[x][y] = 0;

    while(!q.empty()){
        int u =  q.front().x;
```

```
        int v = q.front().y;
            int t = q.front().t;

        q.pop();

            vector<pii> dir = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};

    for(auto it: dir){
                int i = u+it.fs;
                int j = v+it.sc;

                if(isValid(i,j)){
                    if(t == 0){
                        if(vis[i][j] == -1){
                            vis[i][j] = vis[u][v]+1;
                            q.push(Cell{i,j,0});
                        }
                    }else{
                        if(isSafe(i,j,u,v)){
                            path[i][j] = (it.fs == 1) ? 'D' : (it.fs == -1) ? 'U' : (it.sc == 1) ? 'R' : 'L';
                            player[i][j] = player[u][v]+1;
                            q.push(Cell{i,j,1});
                            if (isExit.find({i,j}) != isExit.end()) {
                                if (player[i][j] < vis[i][j] || vis[i][j] == -1) {
                                    restorePath(i, j, x, y);
                                    return true;
                                }
                            }
                        }
                    }
                }

        }
    }

    return false;
}
```

## 7.10 Dijkstra

```
typedef pair<ll, ll> pll;

vector<ll> dijkstra(int n, int source, vector<vector<pll>> &adj) {
    vector<ll> dist(n, INF);
    priority_queue<pll, vector<pll>, greater<pll>> pq;
    dist[source] = 0;
    pq.push({0, source});

    while (!pq.empty()) {
        ll d = pq.top().first;
        ll u = pq.top().second;
        pq.pop();

        if (d > dist[u]) continue;

        for (auto &edge : adj[u]) {
            ll v = edge.first;
            ll weight = edge.second;

            if (dist[u] + weight < dist[v]) {
                dist[v] = dist[u] + weight;
                pq.push({dist[v], v});
            }
        }
    }

    return dist;
}
```

## 7.11 Bellman Ford (With path restoring)

```
struct Edge {
    int src, dest, weight;
};

void bellmanFord(int V, int E, vector<Edge>& edges, int start) {
    vector<int> dist(V+1, INT_MAX);
    dist[start] = 0;

    for (int i = 1; i < V; i++) {
        for (int j = 0; j < E; j++) {
            int u = edges[j].src;
            int v = edges[j].dest;
            int weight = edges[j].weight;
            if (dist[u] != INT_MAX && dist[u] + weight < dist[v]) {
                dist[v] = dist[u] + weight;
            }
        }
    }

    for (int j = 0; j < E; j++) {
        int u = edges[j].src;
        int v = edges[j].dest;
```

```
        int weight = edges[j].weight;
        if (dist[u] != INT_MAX && dist[u] + weight < dist[v]) {
            //cout << "Graph contains a negative weight cycle\n";
            return;
        }
    }

    for(int i=1; i<=V; i++){
        if(dist[i]!=INT_MAX){
            cout<<dist[i]<<" ";
        }else{
            cout<<"30000 ";
        }
    }
    cout<<endl;

}

void solve()
{
    vector<int> d(n, INF);
    d[v] = 0;
    vector<int> p(n, -1);

    for (;;) {
        bool any = false;
        for (Edge e : edges)
            if (d[e.a] < INF)
                if (d[e.b] > d[e.a] + e.cost) {
                    d[e.b] = d[e.a] + e.cost;
                    p[e.b] = e.a;
                    any = true;
                }
        if (!any)
            break;
    }

    if (d[t] == INF)
        cout << "No path from " << v << " to " << t << ".";
    else {
        vector<int> path;
        for (int cur = t; cur != -1; cur = p[cur])
            path.push_back(cur);
        reverse(path.begin(), path.end());

        cout << "Path from " << v << " to " << t << ": ";
        for (int u : path)
            cout << u << ' ';
    }
}
```

## 7.12 SPFA Bellman Ford

```
const int INF = 1000000000;
vector<vector<pair<int, int>>> adj;

bool spfa(int s, vector<int>& d) {
    int n = adj.size();
    d.assign(n, INF);
    vector<int> cnt(n, 0);
    vector<bool> inqueue(n, false);
    queue<int> q;

    d[s] = 0;
    q.push(s);
    inqueue[s] = true;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        inqueue[v] = false;

        for (auto edge : adj[v]) {
            int to = edge.first;
            int len = edge.second;

            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                if (!inqueue[to]) {
                    q.push(to);
                    inqueue[to] = true;
                    cnt[to]++;
                    if (cnt[to] > n)
                        return false;  // negative cycle
                }
            }
        }
    }
    return true;
}
```

## 7.13 Floyd-Warshall

```
void floydWarshall(vector<vector<ll>> &d, int n){
    for (int k = 0; k < n; ++k) {
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
            }
        }
    }
```

```
}
```

## 7.14 Prim's Algorithm (MST)

```
ll prim(int V, int E, vector<vector<pll>> &adj) {

    priority_queue<pll, vector<pll>, greater<pll>> pq;

    vector<bool> visited(V, false);

    ll res = 0;

    pq.push({0, 0});

    while(!pq.empty()){
        auto p = pq.top();
        pq.pop();

        int wt = p.first;
        int u = p.second;

        if(visited[u] == true){
            continue;
        }

        res += wt;
        visited[u] = true;

        for(auto v : adj[u]){
            if(visited[v.first] == false){
                pq.push({v.second, v.first});
            }
        }
    }

    for(int i=0; i<V; i++){
        if(!visited[i])
            return -1;
    }

    return res;
}
```

## 7.15 Kruskal's Algorithm (MST)

```
struct Edge { int u, v, weight; };

int kruskal(vector<Edge>& edges, int n) {
    sort(edges.begin(), edges.end(),
        [](Edge& a, Edge& b) { return a.weight < b.weight; });

    DisjointSets dsu(n);
    int total_weight = 0;

    for (Edge& e : edges) {
        if (!dsu.connected(e.u, e.v)) {
            dsu.unite(e.u, e.v);
            total_weight += e.weight;
        }
    }
    return total_weight;
}
```

## 7.16 Another Kruskal

```
struct Edge {
    int u, v, w;
    bool operator<(Edge const& other) {
        return w < other.w;
    }
};

int kruskal(int n, vector<Edge> &edges, DisjointSets &dsu, vector<
    Edge> &ans) {
    int cost = 0;
    sort(edges.begin(), edges.end());
    for (Edge e : edges) {
        if (ans.size() == n - 1) break;
        if(dsu.unite(e.u, e.v)){
            cost += e.w;
            ans.push_back(e);
        }
    }

    if(ans.size()!=n-1) return -1;
    return cost;
}
```

## 7.17 Kosaraju Algorithm (SCC)

```
vector<bool> visited; // keeps track of which vertices are already
     visited

// runs depth first search starting at vertex v.
// each visited vertex is appended to the output vector when dfs
    leaves it.
```

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

// input: adj -- adjacency list of G
// output: components -- the strongy connected components in G
// output: adj_cond -- adjacency list of G^SCC (by root vertices)
void strongly_connected_components(vector<vector<int>> const& adj,
                                  vector<vector<int>> &components,
                                  vector<vector<int>> &adj_cond) {
    int n = adj.size();
    components.clear(), adj_cond.clear();

    vector<int> order; // will be a sorted list of G's vertices by
         exit time

    visited.assign(n, false);

    // first series of depth first searches
    for (int i = 0; i < n; i++)
        if (!visited[i])
            dfs(i, adj, order);

    // create adjacency list of G^T
    vector<vector<int>> adj_rev(n);
    for (int v = 0; v < n; v++)
        for (int u : adj[v])
            adj_rev[u].push_back(v);

    visited.assign(n, false);
    reverse(order.begin(), order.end());

    vector<int> roots(n, 0); // gives the root vertex of a vertex'
        s SCC

    // second series of depth first searches
    for (auto v : order)
        if (!visited[v]) {
            std::vector<int> component;
            dfs(v, adj_rev, component);
            components.push_back(component);
            int root = *min_element(begin(component), end(
                component));
            for (auto u : component)
                roots[u] = root;
        }

    // add edges to condensation graph
    adj_cond.assign(n, {});
    for (int v = 0; v < n; v++)
        for (auto u : adj[v])
            if (roots[v] != roots[u])
                adj_cond[roots[v]].push_back(roots[u]);
}
```

## 7.18 SCC

```cpp
typedef long long ll;
typedef vector<int> vec;
const ll mod=1e9+7;
const int MAX=1e5+3;
vector<vector<int>> g(MAX);
vector<vector<int>> r(MAX);
vector<int> id(MAX);
bool visitados[MAX]={false};
vector<int> l;

void dfs(int s){
    visitados[s]=true;
    for(int c:g[s]){
        if(!visitados[c]) dfs(c);
    }
    l.push_back(s);
}

void rdfs(int s,int d)
{
    visitados[s]=true;
    id[s]=d;
    for(int c:r[s])
    {
        if(!visitados[c]) rdfs(c,d);
    }
}
```

## 7.19 Tarjan algorithm (SCC)

```cpp
/** Takes in an adjacency list and calculates the SCCs of the
    graph. */
class TarjanSolver {
  private:
    vector<vector<int>> rev_adj;
    vector<int> post;
    vector<int> comp;
```

```

    vector<bool> visited;
    int timer = 0;
    int id = 0;

    void fill_post(int at) {
        visited[at] = true;
        for (int n : rev_adj[at]) {
            if (!visited[n]) { fill_post(n); }
        }
        post[at] = timer++;
    }

    void find_comp(int at) {
        visited[at] = true;
        comp[at] = id;
        for (int n : adj[at]) {
            if (!visited[n]) { find_comp(n); }
        }
    }

  public:
    const vector<vector<int>> &adj;

    TarjanSolver(const vector<vector<int>> &adj)
        : adj(adj), rev_adj(adj.size()), post(adj.size()), comp(
          adj.size()),
          visited(adj.size()) {
        vector<int> nodes(adj.size());
        for (int n = 0; n < adj.size(); n++) {
            nodes[n] = n;
            for (int next : adj[n]) { rev_adj[next].push_back(n);
                }
        }

        for (int n = 0; n < adj.size(); n++) {
            if (!visited[n]) { fill_post(n); }
        }
        std::sort(nodes.begin(), nodes.end(),
                  [&](int n1, int n2) { return post[n1] > post[n2
                      ]; });

        visited.assign(adj.size(), false);
        for (int n : nodes) {
            if (!visited[n]) {
                find_comp(n);
                id++;
            }
        }
    }

    int comp_num() const { return id; }

    int get_comp(int n) const { return comp[n]; }
};
```

## 7.20 Finding Articulation Points

```
// adj[u] = adjacent nodes of u
// ap = AP = articulation points
// p = parent
// disc[u] = discovery time of u
// low[u] = 'low' node of u

int dfsAP(int u, int p) {
  int children = 0;
  low[u] = disc[u] = ++Time;
  for (int& v : adj[u]) {
    if (v == p) continue; // we don't want to go back through the
        same path.
                          // if we go back is because we found
                              another way back
    if (!disc[v]) { // if V has not been discovered before
      children++;
      dfsAP(v, u); // recursive DFS call
      if (disc[u] <= low[v]) // condition #1
        ap[u] = 1;
      low[u] = min(low[u], low[v]); // low[v] might be an ancestor
          of u
    } else // if v was already discovered means that we found an
        ancestor
      low[u] = min(low[u], disc[v]); // finds the ancestor with
          the least discovery time
  }
  return children;
}

void AP() {
  ap = low = disc = vector<int>(adj.size());
  Time = 0;
  for (int u = 0; u < adj.size(); u++)
    if (!disc[u])
      ap[u] = dfsAP(u, u) > 1; // condition #2
}
```

## 7.21 Finding bridges

```
// br = bridges, p = parent

```

```
vector<pair<int, int>> br;

int dfsBR(int u, int p) {
  low[u] = disc[u] = ++Time;
  for (int& v : adj[u]) {
    if (v == p) continue; // we don't want to go back through the same path.
                          // if we go back is because we found another way back
    if (!disc[v]) { // if V has not been discovered before
      dfsBR(v, u);  // recursive DFS call
      if (disc[u] < low[v]) // condition to find a bridge
        br.push_back({u, v});
      low[u] = min(low[u], low[v]); // low[v] might be an ancestor of u
    } else // if v was already discovered means that we found an ancestor
      low[u] = min(low[u], disc[v]); // finds the ancestor with the least discovery time
  }
}

void BR() {
  low = disc = vector<int>(adj.size());
  Time = 0;
  for (int u = 0; u < adj.size(); u++)
    if (!disc[u])
      dfsBR(u, u)
}
```

## 7.22 Finding Bridges Online

```
vector<int> par, dsu_2ecc, dsu_cc, dsu_cc_size;
int bridges;
int lca_iteration;
vector<int> last_visit;

void init(int n) {
    par.resize(n);
    dsu_2ecc.resize(n);
    dsu_cc.resize(n);
    dsu_cc_size.resize(n);
    lca_iteration = 0;
    last_visit.assign(n, 0);
    for (int i=0; i<n; ++i) {
        dsu_2ecc[i] = i;
        dsu_cc[i] = i;
        dsu_cc_size[i] = 1;
        par[i] = -1;
    }
    bridges = 0;
}

int find_2ecc(int v) {
    if (v == -1)
        return -1;
    return dsu_2ecc[v] == v ? v : dsu_2ecc[v] = find_2ecc(dsu_2ecc[v]);
}

int find_cc(int v) {
    v = find_2ecc(v);
    return dsu_cc[v] == v ? v : dsu_cc[v] = find_cc(dsu_cc[v]);
}

void make_root(int v) {
    int root = v;
    int child = -1;
    while (v != -1) {
        int p = find_2ecc(par[v]);
        par[v] = child;
        dsu_cc[v] = root;
        child = v;
        v = p;
    }
    dsu_cc_size[root] = dsu_cc_size[child];
}

void merge_path (int a, int b) {
    ++lca_iteration;
    vector<int> path_a, path_b;
    int lca = -1;
    while (lca == -1) {
        if (a != -1) {
            a = find_2ecc(a);
            path_a.push_back(a);
            if (last_visit[a] == lca_iteration){
                lca = a;
                break;
            }
            last_visit[a] = lca_iteration;
            a = par[a];
        }
        if (b != -1) {
            b = find_2ecc(b);
            path_b.push_back(b);
            if (last_visit[b] == lca_iteration){
                lca = b;
                break;
            }
```

```cpp
                last_visit[b] = lca_iteration;
                b = par[b];
            }

        }

        for (int v : path_a) {
            dsu_2ecc[v] = lca;
            if (v == lca)
                break;
            --bridges;
        }
        for (int v : path_b) {
            dsu_2ecc[v] = lca;
            if (v == lca)
                break;
            --bridges;
        }
}

void add_edge(int a, int b) {
    a = find_2ecc(a);
    b = find_2ecc(b);
    if (a == b)
        return;

    int ca = find_cc(a);
    int cb = find_cc(b);

    if (ca != cb) {
        ++bridges;
        if (dsu_cc_size[ca] > dsu_cc_size[cb]) {
            swap(a, b);
            swap(ca, cb);
        }
        make_root(a);
        par[a] = dsu_cc[a] = b;
        dsu_cc_size[cb] += dsu_cc_size[a];
    } else {
        merge_path(a, b);
    }
}
```

## 7.23 Bridge Tree

```cpp
vector<pair<int, int>> g[MAXN];
bool used[MAXN], isBridge[MAXM];
int comp[MAXN], tin[MAXN], minAncestor[MAXN];

vector<int> tree[MAXN]; // Store 2-edge-connected component tree.(
    Bridge tree).

void dfs(int v, int p) {
    tin[v] = minAncestor[v] = ++timer;
    used[v] = 1;
    for(auto &e: g[v]) {
        int to, id;
        tie(to, id) = e;
        if(to == p) continue;
        if(used[to]) {
            minAncestor[v] = min(minAncestor[v], tin[to]);
        } else {
            dfs(to, v);
            minAncestor[v] = min(minAncestor[v], minAncestor[to]);
            if(minAncestor[to] > tin[v]) {
                isBridge[id] = true;
            }
        }
    }
}

void dfs1(int v, int p) {
    used[v] = 1;
    comp[v] = compid;
    for(auto &e: g[v]) {
        int to, id;
        tie(to, id) = e;

        if(isBridge[id]) { // avoid traversing from this edge. so
            we get full component.
            continue;
        }
        if(used[to]) {
            continue;
        }
        dfs1(to, v);
    }
}

vector<pair<int, int>> edges;

void addEdge(int from, int to, int id) {
    g[from].push_back({to, id});
    g[to].push_back({from, id});
    edges[id] = {from, to};
}

void initB() {

    for(int i = 0; i <= compid; ++i)
```

```cpp
        tree[i].clear();
    for(int i = 1; i <= N; ++i)
        used[i] = false;
    for(int i = 1; i <= M; ++i)
        isBridge[i] = false;

    timer = 0;
    compid = 0;
}

void bridge_tree() {

    initB();

    dfs(1, -1); //Assuming graph is connected.

    for(int i = 1; i <= N; ++i)
        used[i] = 0;

    for(int i = 1; i <= N; ++i) {
        if(!used[i]) {
            dfs1(i, -1);
            ++compid;
        }
    }

    for(int i = 1; i <= M; ++i) {
        if(isBridge[i]) {
            int u, v;
            tie(u, v) = edges[i];
            // connect two componets using edge.
            tree[comp[u]].push_back(comp[v]);
            tree[comp[v]].push_back(comp[u]);
        }
    }
}

void init() {
    edges.clear(); edges.resize(M + 1);
    for(int i = 1; i <= N; ++i)
        g[i].clear();
}
```

### 7.24 2-SAT

```cpp
struct TwoSatSolver {
    int n_vars;
    int n_vertices;
    vector<vector<int>> adj, adj_t;
    vector<bool> used;
    vector<int> order, comp;
    vector<bool> assignment;

    TwoSatSolver(int _n_vars) : n_vars(_n_vars), n_vertices(2 *
        n_vars), adj(n_vertices), adj_t(n_vertices), used(
        n_vertices), order(), comp(n_vertices, -1), assignment(
        n_vars) {
        order.reserve(n_vertices);
    }
    void dfs1(int v) {
        used[v] = true;
        for (int u : adj[v]) {
            if (!used[u])
                dfs1(u);
        }
        order.push_back(v);
    }

    void dfs2(int v, int cl) {
        comp[v] = cl;
        for (int u : adj_t[v]) {
            if (comp[u] == -1)
                dfs2(u, cl);
        }
    }

    bool solve_2SAT() {
        order.clear();
        used.assign(n_vertices, false);
        for (int i = 0; i < n_vertices; ++i) {
            if (!used[i])
                dfs1(i);
        }

        comp.assign(n_vertices, -1);
        for (int i = 0, j = 0; i < n_vertices; ++i) {
            int v = order[n_vertices - i - 1];
            if (comp[v] == -1)
                dfs2(v, j++);
        }

        assignment.assign(n_vars, false);
        for (int i = 0; i < n_vertices; i += 2) {
            if (comp[i] == comp[i + 1])
                return false;
            assignment[i / 2] = comp[i] > comp[i + 1];
        }
        return true;
    }

```

```cpp
    void add_disjunction(int a, bool na, int b, bool nb) {
        // na and nb signify whether a and b are to be negated
        a = 2 * a ^ na;
        b = 2 * b ^ nb;
        int neg_a = a ^ 1;
        int neg_b = b ^ 1;
        adj[neg_a].push_back(b);
        adj[neg_b].push_back(a);
        adj_t[b].push_back(neg_a);
        adj_t[a].push_back(neg_b);
    }

    static void example_usage() {
        TwoSatSolver solver(3); // a, b, c
        solver.add_disjunction(0, false, 1, true);  //     a  v
            not b
        solver.add_disjunction(0, true, 1, true);   // not a  v
            not b
        solver.add_disjunction(1, false, 2, false); //     b  v
              c
        solver.add_disjunction(0, false, 0, false); //     a  v
              a
        assert(solver.solve_2SAT() == true);
        auto expected = vector<bool>(True, False, True);
        assert(solver.assignment == expected);
    }
};
```

## 7.25 Hierholzer's Algorithm (Eulerian Path)

```cpp
int n, m;
vector<vector<int>> g;
vector<int> in, out, path;

// Undirected

int n, m;
vector<vector<pair<int, int>>> g;
vector<int> path;
vector<bool> seen;

void dfs(int node) {
    while (!g[node].empty()) {
        auto [son, idx] = g[node].back();
        g[node].pop_back();
        if (seen[idx]) { continue; }
        seen[idx] = true;
        dfs(son);
    }
    path.push_back(node);
}

// Directed
void dfs(int node) {
    while (!g[node].empty()) {
        int son = g[node].back();
        g[node].pop_back();
        dfs(son);
    }
    path.push_back(node);
}
```

## 7.26 Gale-Shapley Algorithm (Stable marriage)

```cpp
// Checks if woman 'w' prefers 'm1' over 'm'
bool wPrefersM1OverM(vector<vector<int>> &prefer, int w, int m,
    int m1)
{
    int N = prefer[0].size();
    for (int i = 0; i < N; i++)
    {
        // If m1 comes before m, w prefers
        // her current engagement
        if (prefer[w][i] == m1)
            return true;

        // If m comes before m1, w prefers m
        if (prefer[w][i] == m)
            return false;
    }
}

// Implements the stable marriage algorithm
vector<int> stableMarriage(vector<vector<int>> &prefer)
{
    int N = prefer[0].size();

    // Stores women's partners
    vector<int> wPartner(N, -1);

    // Tracks free men
    vector<bool> mFree(N, false);
    int freeCount = N;

    while (freeCount > 0)
    {
        int m;
        for (m = 0; m < N; m++)
```

```cpp
            if (!mFree[m])
                break;

        // Process each woman in m's preference list
        for (int i = 0; i < N && !mFree[m]; i++)
        {
            int w = prefer[m][i];
            if (wPartner[w - N] == -1)
            {
                // Engage m and w if w is free
                wPartner[w - N] = m;
                mFree[m] = true;
                freeCount--;
            }
            else
            {
                int m1 = wPartner[w - N];
                // If w prefers m over her current partner,
                    reassign
                if (!wPrefersM1OverM(prefer, w, m, m1))
                {
                    wPartner[w - N] = m;
                    mFree[m] = true;
                    mFree[m1] = false;
                }
            }
        }
    }
    return wPartner;
}
```

# 8 Trees

## 8.1 Succesor

```cpp
const ll mod=1e9+7;
const ll MAX=1e9+1;
const int limit=2e5+1;
const int m=30;
int succesorM[limit][m];
//ascii https://elcodigoascii.com.ar/

inline void solve()
{
    int n,q; cin>>n>>q;
    int res,aux;
    ll k;
    lFOR(i,n){
        cin>>succesorM[i][0];
    }
    FOR(j,1,m)
    {
        lFOR(i,n)
        {
            succesorM[i][j]=succesorM[succesorM[i][j-1]][j-1];
        }
    }
    FO(i,q)
    {
        cin>>res>>k;
        aux=0;
        while(k)
        {
            if(k%2){
                res=succesorM[res][aux];
            }
            k/=2;
            aux++;
        }
        cout<<res<<endl;
    }
}
```

## 8.2 Euler Tour

```cpp
const int MAXN = 1e5 + 5;

vector<int> adj[MAXN];
int in_time[MAXN], out_time[MAXN];
int timer = 0;

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

    FenwickTree(int n) {
        this->n = n;
        bit.assign(n + 1, 0);
    }

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

    int query(int idx) {
        int sum = 0;
```

```cpp
        for (; idx > 0; idx -= idx & -idx)
            sum += bit[idx];
        return sum;
    }

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

void euler_tour(int root) {
    stack<tuple<int, int, bool>> st;
    st.push({root, -1, false});

    while (!st.empty()) {
        auto [u, parent, visited] = st.top();
        st.pop();

        if (!visited) {
            in_time[u] = ++timer;
            st.push({u, parent, true});

            for (auto it = adj[u].rbegin(); it != adj[u].rend();
                ++it) {
                if (*it != parent) {
                    st.push({*it, u, false});
                }
            }
        } else {
            out_time[u] = ++timer;
        }
    }
}
```

## 8.3 Lowest Common Ancestor

```cpp
struct LCA {
    vector<int> height, euler, first, segtree;
    vector<bool> visited;
    int n;

    LCA(vector<vector<int>> &adj, int root = 0) {
        n = adj.size();
        height.resize(n);
        first.resize(n);
        euler.reserve(n * 2);
        visited.assign(n, false);
        dfs(adj, root);
        int m = euler.size();
        segtree.resize(m * 4);
        build(1, 0, m - 1);
    }

    void dfs(vector<vector<int>> &adj, int node, int h = 0) {
        visited[node] = true;
        height[node] = h;
        first[node] = euler.size();
        euler.push_back(node);
        for (auto to : adj[node]) {
            if (!visited[to]) {
                dfs(adj, to, h + 1);
                euler.push_back(node);
            }
        }
    }

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

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

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

    int lca(int u, int v) {
        int left = first[u], right = first[v];
        if (left > right)
            swap(left, right);
        return query(1, 0, euler.size() - 1, left, right);
    }
};
```

## 8.4 Binary Lifting

```
int n, l;
vector<vector<int>> adj;

int timer;
vector<int> tin, tout;
vector<vector<int>> up;

void dfs(int v, int p)
{
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i-1]][i-1];

    for (int u : adj[v]) {
        if (u != p)
            dfs(u, v);
    }

    tout[v] = ++timer;
}

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

int lca(int u, int v)
{
    if (is_ancestor(u, v))
        return u;
    if (is_ancestor(v, u))
        return v;
    for (int i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}

void preprocess(int root) {
    tin.resize(n);
    tout.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
}
```

## 8.5 Cartesian Tree

```
vector<int> parent(n, -1);
stack<int> s;
for (int i = 0; i < n; i++) {
    int last = -1;
    while (!s.empty() && A[s.top()] >= A[i]) {
        last = s.top();
        s.pop();
    }
    if (!s.empty())
        parent[i] = s.top();
    if (last >= 0)
        parent[last] = i;
    s.push(i);
}
```

## 8.6 Heavy-Light Decomposition

```
vector<int> parent, depth, heavy, head, pos;
int cur_pos;

int dfs(int v, vector<vector<int>> const& adj) {
    int size = 1;
    int max_c_size = 0;
    for (int c : adj[v]) {
        if (c != parent[v]) {
            parent[c] = v, depth[c] = depth[v] + 1;
            int c_size = dfs(c, adj);
            size += c_size;
            if (c_size > max_c_size)
                max_c_size = c_size, heavy[v] = c;
        }
    }
    return size;
}

void decompose(int v, int h, vector<vector<int>> const& adj) {
    head[v] = h, pos[v] = cur_pos++;
    if (heavy[v] != -1)
        decompose(heavy[v], h, adj);
    for (int c : adj[v]) {
        if (c != parent[v] && c != heavy[v])
            decompose(c, c, adj);
    }
```

```
}

void init(vector<vector<int>> const& adj) {
    int n = adj.size();
    parent = vector<int>(n);
    depth = vector<int>(n);
    heavy = vector<int>(n, -1);
    head = vector<int>(n);
    pos = vector<int>(n);
    cur_pos = 0;

    dfs(0, adj);
    decompose(0, 0, adj);
}

int query(int a, int b) {
    int res = 0;
    for (; head[a] != head[b]; b = parent[head[b]]) {
        if (depth[head[a]] > depth[head[b]])
            swap(a, b);
        int cur_heavy_path_max = segment_tree_query(pos[head[b]],
            pos[b]);
        res = max(res, cur_heavy_path_max);
    }
    if (depth[a] > depth[b])
        swap(a, b);
    int last_heavy_path_max = segment_tree_query(pos[a], pos[b]);
    res = max(res, last_heavy_path_max);
    return res;
}
```

## 8.7 Centroid Decomposition

```
vector<vector<int>> adj;
vector<bool> is_removed;
vector<int> subtree_size;

/** DFS to calculate the size of the subtree rooted at `node` */
int get_subtree_size(int node, int parent = -1) {
    subtree_size[node] = 1;
    for (int child : adj[node]) {
        if (child == parent || is_removed[child]) { continue; }
        subtree_size[node] += get_subtree_size(child, node);
    }
    return subtree_size[node];
}

/**
 * Returns a centroid (a tree may have two centroids) of the
   subtree
 * containing node `node` after node removals
 * @param node current node
 * @param tree_size size of current subtree after node removals
 * @param parent parent of u
 * @return first centroid found
 */
int get_centroid(int node, int tree_size, int parent = -1) {
    for (int child : adj[node]) {
        if (child == parent || is_removed[child]) { continue; }
        if (subtree_size[child] * 2 > tree_size) {
            return get_centroid(child, tree_size, node);
        }
    }
    return node;
}

/** Build up the centroid decomposition recursively */
void build_centroid_decomp(int node = 0) {
    int centroid = get_centroid(node, get_subtree_size(node));

    // do something

    is_removed[centroid] = true;

    for (int child : adj[centroid]) {
        if (is_removed[child]) { continue; }
        build_centroid_decomp(child);
    }
}
```

## 8.8 Tree Distances

```
vector<int> graph[200001];
int fir[200001], sec[200001], ans[200001];

void dfs1(int node = 1, int parent = 0) {
    for (int i : graph[node])
        if (i != parent) {
            dfs1(i, node);
            if (fir[i] + 1 > fir[node]) {
                sec[node] = fir[node];
                fir[node] = fir[i] + 1;
            } else if (fir[i] + 1 > sec[node]) {
                sec[node] = fir[i] + 1;
            }
        }
}
```

```cpp

void dfs2(int node = 1, int parent = 0, int to_p = 0) {
    ans[node] = max(to_p, fir[node]);
    for (int i : graph[node])
        if (i != parent) {
            if (fir[i] + 1 == fir[node]) dfs2(i, node, max(to_p,
                sec[node]) + 1);
            else dfs2(i, node, ans[node] + 1);
        }
}
```

# 9 Flows

## 9.1 Ford-Fulkerson Maximum Flow

```cpp
int n;
vector<vector<int>> capacity;
vector<vector<int>> adj;

int bfs(int s, int t, vector<int>& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, INF});

    while (!q.empty()) {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();

        for (int next : adj[cur]) {
            if (parent[next] == -1 && capacity[cur][next]) {
                parent[next] = cur;
                int new_flow = min(flow, capacity[cur][next]);
                if (next == t)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}

int maxflow(int s, int t) {
    int flow = 0;
    vector<int> parent(n);
    int new_flow;

    while (new_flow = bfs(s, t, parent)) {
        flow += new_flow;
        int cur = t;
        while (cur != s) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}
```

## 9.2 Dinic's Max Flow

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

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

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

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

    bool bfs() {
        while (!q.empty()) {
```

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

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

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

## 9.3 Min-cost Flow

```
struct Edge
{
    int from, to, capacity, cost;
};

vector<vector<int>> adj, cost, capacity;

const int INF = 1e9;

void shortest_paths(int n, int v0, vector<int>& d, vector<int>& p)
    {
    d.assign(n, INF);
    d[v0] = 0;
    vector<bool> inq(n, false);
    queue<int> q;
    q.push(v0);
    p.assign(n, -1);

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (int v : adj[u]) {
            if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v]) {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (!inq[v]) {
                    inq[v] = true;
                    q.push(v);
                }
            }
        }
    }
}

int min_cost_flow(int N, vector<Edge> edges, int K, int s, int t)
    {
    adj.assign(N, vector<int>());
    cost.assign(N, vector<int>(N, 0));
    capacity.assign(N, vector<int>(N, 0));
    for (Edge e : edges) {
        adj[e.from].push_back(e.to);
        adj[e.to].push_back(e.from);
        cost[e.from][e.to] = e.cost;
        cost[e.to][e.from] = -e.cost;
        capacity[e.from][e.to] = e.capacity;
```

```
    }

    int flow = 0;
    int cost = 0;
    vector<int> d, p;
    while (flow < K) {
        shortest_paths(N, s, d, p);
        if (d[t] == INF)
            break;

        // find max flow on that path
        int f = K - flow;
        int cur = t;
        while (cur != s) {
            f = min(f, capacity[p[cur]][cur]);
            cur = p[cur];
        }

        // apply flow
        flow += f;
        cost += f * d[t];
        cur = t;
        while (cur != s) {
            capacity[p[cur]][cur] -= f;
            capacity[cur][p[cur]] += f;
            cur = p[cur];
        }
    }

    if (flow < K)
        return -1;
    else
        return cost;
}
```

## 9.4 Hungarian Algorithm

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

vector<int> ans (n+1);
for (int j=1; j<=m; ++j)
    ans[p[j]] = j;

int cost = -v[0];
```

## 9.5 Kuhn's Algorithm

```
int n, k;
vector<vector<int>> g;
vector<int> mt;
vector<bool> used;

bool try_kuhn(int v) {
    if (used[v])
        return false;
    used[v] = true;
    for (int to : g[v]) {
        if (mt[to] == -1 || try_kuhn(mt[to])) {
            mt[to] = v;
            return true;
        }
    }
    return false;
}

int main() {
    //... reading the graph ...

    mt.assign(k, -1);
```

```
    for (int v = 0; v < n; ++v) {
        used.assign(n, false);
        try_kuhn(v);
    }

    for (int i = 0; i < k; ++i)
        if (mt[i] != -1)
            printf("%d %d\n", mt[i] + 1, i + 1);
}
```

# 10 Dynamic Programming

## 10.1 Coin Problem (Count ways)

```
vector<ll> coins(n);
for(int i=0; i<n; i++){
    cin>>coins[i];
}

vector<ll> dp(x+1,0);
dp[0] = 1;
for(int i=0; i<=x; i++){
    for(int j=0; j<n; j++){
        if(i-coins[j]>=0){
            dp[i] = (dp[i] + dp[i-coins[j]]);
            dp[i]%=MOD;
        }
    }
}


cout<<dp[x]<<endl;
```

## 10.2 Coin Problem (Count sorted ways)

```
vector<ll> coins(n);
for(int i=0; i<n; i++){
    cin>>coins[i];
}

int dp[102][1000005];
dp[0][0] = 1;
for(int i=1; i<=n; i++){
    for(int j=0; j<=x; j++){
        dp[i][j] = dp[i-1][j];
        int l = j-coins[i-1];
        if(l>=0){
            dp[i][j] += (dp[i][l])%MOD;
            dp[i][j]%=MOD;
        }
    }
}


cout<<dp[n][x]%MOD<<endl;
```

## 10.3 Coin Problem (Minimum)

```
vector<ll> coins(n);
for(int i=0; i<n; i++){
    cin>>coins[i];
}

vector<ll> dp(x+1,INT_MAX);
dp[0] = 0;
for(int i=0; i<=x; i++){
    for(int j=0; j<n; j++){
        if(i-coins[j]>=0){
            dp[i] = min(dp[i], dp[i-coins[j]]+1);
        }
    }
}

if(dp[x] != INT_MAX){
    cout<<dp[x]<<endl;
}else{
    cout<<"-1"<<endl;
}
```

## 10.4 Counting paths

```
int n; cin>>n;
char grid[n][n];
int dp[n][n];

for(int i=0; i<n; i++){
    for(int j=0; j<n; j++){
        cin>>grid[i][j];
        dp[i][j] = 0;
    }
}
if(grid[0][0] != '*')dp[0][0] = 1;
else dp[0][0] = 0;
for(int i=0; i<n; i++){
    for(int j=0; j<n; j++){
```

```
        if(grid[i+1][j] == '.' and i+1 < n){
            dp[i+1][j] += dp[i][j]%MOD;
        }
        if(grid[i][j+1] == '.' and j+1 < n){
            dp[i][j+1] += dp[i][j]%MOD;
        }

        if(grid[i][j] == '*'){
            dp[i][j] = 0;
        }
    }
}
cout<<dp[n-1][n-1]%MOD<<endl;
```

## 10.5 Longest Increasing Subsequence

```
vector<int> lis(vector<int> const& a) {
    int n = a.size();
    vector<int> d(n, 1), p(n, -1);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            if (a[j] < a[i] && d[i] < d[j] + 1) {
                d[i] = d[j] + 1;
                p[i] = j;
            }
        }
    }

    int ans = d[0], pos = 0;
    for (int i = 1; i < n; i++) {
        if (d[i] > ans) {
            ans = d[i];
            pos = i;
        }
    }

    vector<int> subseq;
    while (pos != -1) {
        subseq.push_back(a[pos]);
        pos = p[pos];
    }
    reverse(subseq.begin(), subseq.end());
    return subseq;
}
```

## 10.6 Length of LIS

```
int lis(vector<ll> const& a) {
    int n = a.size();
    const int INF = 1e9;
    vector<int> d(n+1, INF);
    d[0] = -INF;

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

    int ans = 0;
    for (int l = 0; l <= n; l++) {
        if (d[l] < INF)
            ans = l;
    }
    return ans;
}
```

## 10.7 Longest Common Subsequence

```
// Returns length of LCS for s1[0..m-1], s2[0..n-1]
int lcs(string &s1, string &s2) {
    int m = s1.size();
    int n = s2.size();

    // Initializing a matrix of size (m+1)*(n+1)
    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));

    // Building dp[m+1][n+1] in bottom-up fashion
    for (int i = 1; i <= m; ++i) {
        for (int j = 1; j <= n; ++j) {
            if (s1[i - 1] == s2[j - 1])
                dp[i][j] = dp[i - 1][j - 1] + 1;
            else
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
        }
    }

    // dp[m][n] contains length of LCS for s1[0..m-1]
    // and s2[0..n-1]
    return dp[m][n];
}
```

## 10.8 Edit Distance

```
int editDistance(string &s1, string &s2) {

```

```cpp
    int m = s1.length();
    int n = s2.length();

    // Create a table to store results of subproblems
    vector<vector<int>> dp(m + 1, vector<int>(n + 1));

    // Fill the known entries in dp[][]
    // If one string is empty, then answer
    // is length of the other string
    for (int i = 0; i <= m; i++)
        dp[i][0] = i;
    for (int j = 0; j <= n; j++)
        dp[0][j] = j;

    // Fill the rest of dp[][]
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (s1[i - 1] == s2[j - 1])
                dp[i][j] = dp[i - 1][j - 1];
            else
                dp[i][j] = 1 + min({dp[i][j - 1],
                                    dp[i - 1][j],
                                    dp[i - 1][j - 1]});
        }
    }

    return dp[m][n];
}
```

## 10.9 Bitmask DP

```cpp
typedef long long ll;
typedef vector<int> vec;
const ll mod=1e9+7;
const int limit=20;
vector<pair<ll,ll>> dp((1<<limit));
//ascii https://elcodigoascii.com.ar/

inline void solve()
{
    int n; cin>>n;
    ll x; cin>>x;
    vector<ll> weight(n);
    dp[0]={1,0};
    FO(i,n) cin>>weight[i];
    for(ll i=1;i<(1<<n);i++)
    {
        dp[i]={n+1,0};
        for(int j=0;j<n;j++)
        {
            if(i&(1<<j))
            {
                pair<ll,ll> aux=dp[i^(1<<j)];
                if(aux.second+weight[j]<=x){
                    aux.second+=weight[j];
                }
                else{
                    aux.first++;
                    aux.second=weight[j];
                }
                dp[i]=min(dp[i],aux);
            }
        }
    }
    cout<<dp[(1<<n)-1].first<<endl;
}
```

## 10.10 Digit DP

```cpp
typedef long long ll;
typedef vector<int> vec;
const ll mod=1e9+7;
ll dp[20][10][2][2];
//ascii https://elcodigoascii.com.ar/

ll mem(int idx,int tight,int prev,int ld,string s)
{
    if(idx==0)
    {
        return 1;
    }
    if(dp[idx][prev][ld][tight]!=-1){
        return dp[idx][prev][ld][tight];
    }
    int k=9;
    if(tight) k=s[s.size()-idx]-'0';
    ll sum=0;
    for(int i=0;i<=k;i++)
    {
        if(ld || prev!=i)
        {
            int new_ld,new_tight;
            if(i==0 && ld) new_ld=1;
            else new_ld=0;
            if(tight && k==i) new_tight=1;
            else new_tight=0;
            sum+=mem(idx-1,new_tight,i,new_ld,s);
        }
```

```
    }
    dp[idx][prev][ld][tight]=sum;
    return sum;
}
```

## 10.11 Double DP

```
typedef long long ll;
typedef vector<int> vec;
const ll mod=1e9+7;
const ll MAX=1e6+3;
ll dp[MAX][2];
//ascii https://elcodigoascii.com.ar/

inline void solve()
{
    int n; cin>>n;
    dp[n][0]=1;
    dp[n][1]=1;
    for(int i=n-1;i>0;i--)
    {
        dp[i][1]=4*dp[i+1][1]+dp[i+1][0];
        dp[i][0]=2*dp[i+1][0]+dp[i+1][1];
        dp[i][1]%=mod;
        dp[i][0]%=mod;
    }
    cout<<(dp[1][1]+dp[1][0])%mod<<endl;
}
```

# 11 Math

## 11.1 Prime

```
bool prime(int t){
    if(t%2 == 0){
        return false;
    } else {
        for(int i=3; i*i <=t; i+=2){
            if((t%i)==0){
                return false;
            }
        }
    }
    return true;
}
```

## 11.2 Miller Rabin

```
bool MillerRabin(u64 n) { // returns true if n is prime, else
    returns false.
    if (n < 2)
        return false;

    int r = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }

    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```

## 11.3 Sieve of Erathostenes

```
int n;
vector<bool> is_prime(n+1, true);
is_prime[0] = is_prime[1] = false;
for (int i = 2; i * i <= n; i++) {
    if (is_prime[i]) {
        for (int j = i * i; j <= n; j += i)
            is_prime[j] = false;
    }
}
```

## 11.4 Sieve of Eratosthenes (count primes)

```
int count_primes(int n) {
    const int S = 10000;

    vector<int> primes;
    int nsqrt = sqrt(n);
    vector<char> is_prime(nsqrt + 2, true);
    for (int i = 2; i <= nsqrt; i++) {
        if (is_prime[i]) {
            primes.push_back(i);
            for (int j = i * i; j <= nsqrt; j += i)
                is_prime[j] = false;
```

```
        }
    }

    int result = 0;
    vector<char> block(S);
    for (int k = 0; k * S <= n; k++) {
        fill(block.begin(), block.end(), true);
        int start = k * S;
        for (int p : primes) {
            int start_idx = (start + p - 1) / p;
            int j = max(start_idx, p) * p - start;
            for (; j < S; j += p)
                block[j] = false;
        }
        if (k == 0)
            block[0] = block[1] = false;
        for (int i = 0; i < S && start + i <= n; i++) {
            if (block[i])
                result++;
        }
    }
    return result;
}
```

## 11.5 Segmented Sieve

```
vector<char> segmentedSieve(long long L, long long R) {
    // generate all primes up to sqrt(R)
    long long lim = sqrt(R);
    vector<char> mark(lim + 1, false);
    vector<long long> primes;
    for (long long i = 2; i <= lim; ++i) {
        if (!mark[i]) {
            primes.emplace_back(i);
            for (long long j = i * i; j <= lim; j += i)
                mark[j] = true;
        }
    }

    vector<char> isPrime(R - L + 1, true);
    for (long long i : primes)
        for (long long j = max(i * i, (L + i - 1) / i * i); j <= R
            ; j += i)
            isPrime[j - L] = false;
    if (L == 1)
        isPrime[0] = false;
    return isPrime;
}
```

## 11.6 Linear sieve

```
const int N = 10000000;
vector<int> lp(N+1);
vector<int> pr;

for (int i=2; i <= N; ++i) {
    if (lp[i] == 0) {
        lp[i] = i;
        pr.push_back(i);
    }
    for (int j = 0; i * pr[j] <= N; ++j) {
        lp[i * pr[j]] = pr[j];
        if (pr[j] == lp[i]) {
            break;
        }
    }
}
```

## 11.7 Sum of divisors

```
long long SumOfDivisors(long long num) {
    long long total = 1;

    for (int i = 2; (long long)i * i <= num; i++) {
        if (num % i == 0) {
            int e = 0;
            do {
                e++;
                num /= i;
            } while (num % i == 0);

            long long sum = 0, pow = 1;
            do {
                sum += pow;
                pow *= i;
            } while (e-- > 0);
            total *= sum;
        }
    }
    if (num > 1) {
        total *= (1 + num);
    }
    return total;
}
```

## 11.8 Finding the divisors of a number (Trial Division)

```
vector<long long> trial_division2(long long n) {
    vector<long long> factorization;
    while (n % 2 == 0) {
        factorization.push_back(2);
        n /= 2;
    }
    for (long long d = 3; d * d <= n; d += 2) {
        while (n % d == 0) {
            factorization.push_back(d);
            n /= d;
        }
    }
    if (n > 1)
        factorization.push_back(n);
    return factorization;
}
```

## 11.9 Factorials

```
// Precompute factorials and inverse factorials
void precompute(ll n = MAXN - 1) {
    factorial[0] = factorial[1] = 1;

    // Compute factorials
    for (ll i = 2; i <= n; i++) {
        factorial[i] = (factorial[i - 1] * i) % MOD;
    }

    // Compute inverse factorials efficiently
    inv_factorial[n] = modInv(factorial[n]);
    for (ll i = n - 1; i >= 0; i--) {
        inv_factorial[i] =
            (inv_factorial[i + 1] * (i + 1)) % MOD;
    }
}
```

## 11.10 Binpow

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

## 11.11 Modulo Inverse

```
int modInverse(int A, int M) {
    int m0 = M;
    int y = 0, x = 1;

    if (M == 1)
        return 0;

    while (A > 1) {
        // q is quotient
        int q = A / M;
        int t = M;

        // m is remainder now, process same as
        // Euclid's algo
        M = A % M, A = t;
        t = y;

        // Update y and x
        y = x - q * y;
        x = t;
    }

    // Make x positive
    if (x < 0)
        x += m0;

    return x;
}
```

## 11.12 BinPow Modulo Inv

```
ll modInv(ll a, ll mod = MOD) {
    return power(a, mod - 2, mod);
}
```

## 11.13 Binomial Coefficients

```
long long binomial_coefficient(int n, int k) {
    return factorial[n] * inverse_factorial[k] % m *
        inverse_factorial[n - k] % m;
}
```

## 11.14 Newton Method (Sqrt and iSqrt)

```
double sqrt_newton(double n) {
    const double eps = 1E-15;
    double x = 1;
    for (;;) {
        double nx = (x + n / x) / 2;
        if (abs(x - nx) < eps)
            break;
        x = nx;
    }
    return x;
}

int isqrt_newton(int n) {
    int x = 1;
    bool decreased = false;
    for (;;) {
        int nx = (x + n / x) >> 1;
        if (x == nx || nx > x && decreased)
            break;
        decreased = nx < x;
        x = nx;
    }
    return x;
}
```

## 11.15 Integration with Simpson Method

```
const int N = 1000 * 1000; // number of steps (already multiplied
    by 2)

double simpson_integration(double a, double b){
    double h = (b - a) / N;
    double s = f(a) + f(b); // a = x_0 and b = x_2n
    for (int i = 1; i <= N - 1; ++i) { // Refer to final Simpson's
         formula
        double x = a + h * i;
        s += f(x) * ((i & 1) ? 4 : 2);
    }
    s *= h / 3;
    return s;
}
```

## 11.16 Ternary Search

```
double ternary_search(double l, double r) {
    double eps = 1e-9;              //set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);      //evaluates the function at m1
        double f2 = f(m2);      //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);                    //return the maximum of f(x)
        in [l, r]
}
```

## 11.17 DP Pascal triangle 1D

```
int binomialCoeff(int n, int k) {
    vector<int> dp(k + 1);

      // nC0 is 1
    dp[0] = 1;

    for (int i = 1; i <= n; i++) {

        // Compute next row of pascal triangle using
        // the previous row
        for (int j = min(i, k); j > 0; j--)
            dp[j] = dp[j] + dp[j - 1];
    }
    return dp[k];
}
```

## 11.18 DP Pascal triangle 2D

```
// Returns value of Binomial Coefficient C(n, k)
int binomialCoeff(int n, int k) {
      vector<vector<int>> dp(n + 1, vector<int> (k + 1));

    // Calculate value of Binomial Coefficient
    // in bottom up manner
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= min(i, k); j++) {

            // Base Cases
            if (j == 0 || j == i)
                dp[i][j] = 1;

            // Calculate value using previously
```

```
            // stored values
            else
                dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j];
        }
    }

    return dp[n][k];
}
```

## 11.19 Euler's Totient

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

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

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

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

## 11.20 Diophantine equations

```
void shift_solution(int & x, int & y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}

int find_all_solutions(int a, int b, int c, int minx, int maxx,
    int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;

    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

## 11.21 Discrete Log

```
// Returns minimum x for which a ^ x % m = b % m.
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int k = 1, add = 0, g;
    while ((g = gcd(a, m)) > 1) {
        if (b == k)
            return add;
```

```cpp
        if (b % g)
            return -1;
        b /= g, m /= g, ++add;
        k = (k * 1ll * a / g) % m;
    }

    int n = sqrt(m) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;

    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }

    for (int p = 1, cur = k; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur] + add;
            return ans;
        }
    }
    return -1;
}
```

# 12 Polynomials

## 12.1 FFT

```cpp
using cd = complex<double>;
const double PI = acos(-1);

int reverse(int num, int lg_n) {
    int res = 0;
    for (int i = 0; i < lg_n; i++) {
        if (num & (1 << i))
            res |= 1 << (lg_n - 1 - i);
    }
    return res;
}

void fft(vector<cd> & a, bool invert) {
    int n = a.size();
    int lg_n = 0;
    while ((1 << lg_n) < n)
        lg_n++;

    for (int i = 0; i < n; i++) {
        if (i < reverse(i, lg_n))
            swap(a[i], a[reverse(i, lg_n)]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        double ang = 2 * PI / len * (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2; j++) {
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }

    if (invert) {
        for (cd & x : a)
            x /= n;
    }
}

vector<int> multiply(vector<int> const& a, vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}

// Normalization

int carry = 0;
for (int i = 0; i < n; i++){
    result[i] += carry;
```

```
    carry = result[i] / 10;
    result[i] %= 10;
}
```

## 12.2 NTT

```
const int mod = 7340033;
const int root = 5;
const int root_1 = 4404020;
const int root_pw = 1 << 20;

void fft(vector<int> & a, bool invert) {
    int n = a.size();

    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        int wlen = invert ? root_1 : root;
        for (int i = len; i < root_pw; i <<= 1)
            wlen = (int)(1LL * wlen * wlen % mod);

        for (int i = 0; i < n; i += len) {
            int w = 1;
            for (int j = 0; j < len / 2; j++) {
                int u = a[i+j], v = (int)(1LL * a[i+j+len/2] * w %
                    mod);
                a[i+j] = u + v < mod ? u + v : u + v - mod;
                a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
                w = (int)(1LL * w * wlen % mod);
            }
        }
    }

    if (invert) {
        int n_1 = inverse(n, mod);
        for (int & x : a)
            x = (int)(1LL * x * n_1 % mod);
    }
}
```

## 12.3 Berlekamp Messey

```
vector<T> berlekampMassey(const vector<T> &s) {
    vector<T> c;    // the linear recurrence sequence we are
        building
    vector<T> oldC; // the best previous version of c to use (the
        one with the rightmost left endpoint)
    int f = -1;     // the index at which the best previous
        version of c failed on
    for (int i=0; i<(int)s.size(); i++) {
        // evaluate c(i)
        // delta = s_i - \sum_{j=1}^n c_j s_{i-j}
        // if delta == 0, c(i) is correct
        T delta = s[i];
        for (int j=1; j<=(int)c.size(); j++)
            delta -= c[j-1] * s[i-j];   // c_j is one-indexed, so
                we actually need index j - 1 in the code
        if (delta == 0)
            continue;   // c(i) is correct, keep going
        // now at this point, delta != 0, so we need to adjust it
        if (f == -1) {
            // this is the first time we're updating c
            // s_i was the first non-zero element we encountered
            // we make c of length i + 1 so that s_i is part of
                the base case
            c.resize(i + 1);
            mt19937 rng(chrono::steady_clock::now().
                time_since_epoch().count());
            for (T &x : c)
                x = rng();  // just to prove that the initial
                    values don't matter in the first step, I will
                    set to random values
            f = i;
        } else {
            // we need to use a previous version of c to improve
                on this one
            // apply the 5 steps to build d
            // 1. set d equal to our chosen sequence
            vector<T> d = oldC;
            // 2. multiply the sequence by -1
            for (T &x : d)
                x = -x;
            // 3. insert a 1 on the left
            d.insert(d.begin(), 1);
            // 4. multiply the sequence by delta / d(f + 1)
            T df1 = 0;  // d(f + 1)
            for (int j=1; j<=(int)d.size(); j++)
                df1 += d[j-1] * s[f+1-j];
            assert(df1 != 0);
            T coef = delta / df1;   // storing this in outer
                variable so it's O(n^2) instead of O(n^2 log MOD)
```

```
            for (T &x : d)
                x *= coef;
            // 5. insert i - f - 1 zeros on the left
            vector<T> zeros(i - f - 1);
            zeros.insert(zeros.end(), d.begin(), d.end());
            d = zeros;
            // now we have our new recurrence: c + d
            vector<T> temp = c; // save the last version of c
                because it might have a better left endpoint
            c.resize(max(c.size(), d.size()));
            for (int j=0; j<(int)d.size(); j++)
                c[j] += d[j];
            // finally, let's consider updating oldC
            if (i - (int) temp.size() > f - (int) oldC.size()) {
                // better left endpoint, let's update!
                oldC = temp;
                f = i;
            }
        }
    }
    return c;
}
```

# 13 Linear Algebra

## 13.1 Determinant of a Matrix

```
const double EPS = 1E-9;
int n;
vector < vector<double> > a (n, vector<double> (n));

double det = 1;
for (int i=0; i<n; ++i) {
    int k = i;
    for (int j=i+1; j<n; ++j)
        if (abs (a[j][i]) > abs (a[k][i]))
            k = j;
    if (abs (a[k][i]) < EPS) {
        det = 0;
        break;
    }
    swap (a[i], a[k]);
    if (i != k)
        det = -det;
    det *= a[i][i];
    for (int j=i+1; j<n; ++j)
        a[i][j] /= a[i][i];
    for (int j=0; j<n; ++j)
        if (j != i && abs (a[j][i]) > EPS)
            for (int k=i+1; k<n; ++k)
                a[j][k] -= a[i][k] * a[j][i];
}

cout << det;
```

## 13.2 Rank of a Matrix

```
const double EPS = 1E-9;

int compute_rank(vector<vector<double>> A) {
    int n = A.size();
    int m = A[0].size();

    int rank = 0;
    vector<bool> row_selected(n, false);
    for (int i = 0; i < m; ++i) {
        int j;
        for (j = 0; j < n; ++j) {
            if (!row_selected[j] && abs(A[j][i]) > EPS)
                break;
        }

        if (j != n) {
            ++rank;
            row_selected[j] = true;
            for (int p = i + 1; p < m; ++p)
                A[j][p] /= A[j][i];
            for (int k = 0; k < n; ++k) {
                if (k != j && abs(A[k][i]) > EPS) {
                    for (int p = i + 1; p < m; ++p)
                        A[k][p] -= A[j][p] * A[k][i];
                }
            }
        }
    }
    return rank;
}
```

## 13.3 Gauss-Jordan

```
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be infinity or a
     big number

int gauss (vector < vector<double> > a, vector<double> & ans) {
    int n = (int) a.size();
```

```
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
```

## 13.4 Matrix Exponentiation

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

using ll = long long;

const ll MOD = 1e9 + 7;

using Matrix = array<array<ll, 2>, 2>;

Matrix mul(Matrix a, Matrix b) {
    Matrix res = {{{0, 0}, {0, 0}}};
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            for (int k = 0; k < 2; k++) {
                res[i][j] += a[i][k] * b[k][j];
                res[i][j] %= MOD;
            }
        }
    }

    return res;
}

int main() {
    ll n;
    cin >> n;

    Matrix base = {{{1, 0}, {0, 1}}};
    Matrix m = {{{1, 1}, {1, 0}}};

    for (; n > 0; n /= 2, m = mul(m, m)) {
        if (n & 1) base = mul(base, m);
    }

    cout << base[0][1];
}
```

# 14 Geometry

## 14.1 Line Segment Intersection

```
// BeginCodeSnip{Point Class}
struct Point {
    int x, y;
    Point(int a = 0, int b = 0) : x(a), y(b) {}

    friend istream &operator>>(istream &in, Point &p) {
        int x, y;
        in >> p.x >> p.y;
        return in;
    }
};
// EndCodeSnip

```

```
int sign(long long num) {
    if (num < 0) {
        return -1;
    } else if (num == 0) {
        return 0;
    } else {
        return 1;
    }
}

long long trigonometric_sense(Point p, Point p1, Point p2) {
    return sign(1LL * (p1.x - p.x) * (p2.y - p.y) -
                1LL * (p2.x - p.x) * (p1.y - p.y));
}

// Check if the rectangles with [P1, P2] and [P3, P4] as diagonals
     intersect
bool quick_check(Point p1, Point p2, Point p3, Point p4) {
    int x1, x2, x3, x4, y1, y2, y3, y4;
    x1 = min(p1.x, p2.x), x2 = max(p1.x, p2.x);
    y1 = min(p1.y, p2.y), y2 = max(p1.y, p2.y);
    x3 = min(p3.x, p4.x), x4 = max(p3.x, p4.x);
    y3 = min(p3.y, p4.y), y4 = max(p3.y, p4.y);
    return x2 < x3 || x4 < x1 || y2 < y3 || y4 < y1;
}

bool check(Point p1, Point p2, Point p3, Point p4) {
    if (trigonometric_sense(p1, p2, p3) * trigonometric_sense(p1,
        p2, p4) > 0) {
        return false;
    }
    if (trigonometric_sense(p3, p4, p1) * trigonometric_sense(p3,
        p4, p2) > 0) {
        return false;
    }
    return true;
}

int main() {
    int test_num;
    cin >> test_num;
    for (int t = 0; t < test_num; t++) {
        Point p1, p2, p3, p4;
        cin >> p1 >> p2 >> p3 >> p4;

        if (quick_check(p1, p2, p3, p4)) {
            cout<<"NO"<<endl;
        } else if (check(p1, p2, p3, p4)) {
            cout<<"YES"<<endl;
        } else {
            cout<<"NO"<<endl;
        }
    }
}
```

## 14.2 Minimum Euclidian Distance

```
const ll mod=1e9+7;
const ll MAX=8e18;
const ll limit=1e9+1;
//ascii https://elcodigoascii.com.ar/

ll distance(point a,point b){
    return (a.X-b.X)*(a.X-b.X)+(a.Y-b.Y)*(a.Y-b.Y);
}

inline void solve()
{
    int n; cin>>n;
    vector<point> sortedX(n);
    set<point> sortedY;
    FO(i,n)
    {
        ll x,y; cin>>x>>y;
        sortedX[i]=make_pair(x,y);
    }
    sort(all(sortedX));
    sortedY.insert(make_pair(sortedX[0].Y,sortedX[0].X));
    ll d,minSquare=MAX;
    int j=0;
    FOR(i,1,n)
    {
        d=ceil(sqrt(minSquare));
        while(sortedX[i].X-sortedX[j].X>d)
        {
            sortedY.erase(make_pair(sortedX[j].Y,sortedX[j].X));
            j++;
        }
        auto lower=sortedY.lower_bound(make_pair(sortedX[i].Y-d,0)
            );
        auto upper=sortedY.upper_bound(make_pair(sortedX[i].Y+d,0)
            );
        for(auto pointer=lower;pointer!=upper;pointer++)
        {
            minSquare=min(minSquare,distance(*pointer,make_pair(
                sortedX[i].Y,sortedX[i].X)));
        }
        sortedY.insert(make_pair(sortedX[i].Y,sortedX[i].X));
    }
    cout<<minSquare<<endl;
```

```
}
```

## 14.3 Point in polygon

```
struct point{
    ll x,y;
    void show(){
        cout<<x<<" "<<y<<endl;
    }
};

int sign(ll a){
    if(a<0) return -1;
    if(a==0) return 0;
    if(a>0) return 1;
}

int signCP(point p,point p1,point p2)
{
    return sign(1LL*((p1.x-p.x)*(p2.y-p.y)-(p1.y-p.y)*(p2.x-p.x)));
}

bool intersect(point n, point m,point a,point b)
{
    if(signCP(n,a,b)*signCP(m,a,b)>0) return false;
    if(signCP(a,n,m)*signCP(b,n,m)>0) return false;
    return true;
}

bool inside(point a,point b,point c){
    return a.x>=min(b.x,c.x) && a.x<=max(b.x,c.x) && a.y>=min(b.y,c.y)
    && a.y<=max(b.y,c.y);
}

inline void solve()
{
    int n,m; cin>>n>>m;
    vector<point> vertices(n);
    FO(i,n)
    {
        cin>>vertices[i].x>>vertices[i].y;
    }
    point query,par,init,first,second;
    int counter;
    int resta=0;
    FO(i,m)
    {
        resta=0;
        counter=0;
        cin>>query.x>>query.y;
        par.x=query.x;
        par.y=-MAX-1;
        init.x=vertices[0].x;
        init.y=vertices[0].y;
        first.x=init.x;
        first.y=init.y;
        bool ver=false;
        for(int j=1;j<=n;j++)
        {
            second.x=vertices[j%n].x;
            second.y=vertices[j%n].y;
            point AB,u;
            AB.x=second.x-first.x;
            AB.y=second.y-first.y;
            u.x=second.x-query.x;
            u.y=second.y-query.y;
            if((AB.x*u.y-AB.y*u.x)==0 && inside(query,first,second)){
                cout<<"BOUNDARY"<<endl;
                ver=true;
                break;
            }
            if(intersect(query,par,first,second) && first.x<=query.x && query.x<second.x)
            {
                counter++;
            }
            if(intersect(query,par,first,second) && second.x<=query.x && query.x<first.x){
                counter++;
            }
            first.x=second.x;
            first.y=second.y;
        }
        point AB,u;
        AB.x=init.x-first.x;
        AB.y=init.y-first.y;
        u.x=init.x-query.x;
        u.y=init.y-query.y;
        if(!ver){
            //if(intersect(query,par,first,init)) counter++;
            if((counter)&1) cout<<"INSIDE";
            else cout<<"OUTSIDE";
            cout<<endl;
        }
    }
}
```

## 14.4 Point Location Test

```
struct point{
    double x,y;
};

struct Vector{
    double a=0,b=0;
    void getVector(point p1,point p2){
        a=p2.x-p1.x;
        b=p2.y-p1.y;
    }

    double getModulo(){
        return pow(a*a+b*b,0.5);
    }

    Vector getUnitarian(){
        Vector x;
        x.a=a/getModulo();
        x.b=b/getModulo();
        //cout<<x.a<<" "<<x.b<<endl;
        return x;
    }

};

double dotProduct(Vector x,Vector y)
{
    return x.a*y.a+x.b*y.b;
}

double CrossProduct(Vector x,Vector y)
{
    return x.a*y.b-x.b*y.a;
}

inline void solve()
{

    point p1,p2,p3,p4;
    cin>>p1.x>>p1.y>>p2.x>>p2.y>>p3.x>>p3.y;
    Vector u,v,t;
    u.getVector(p1,p3);
    //cout<<u.a<<" "<<u.b<<endl;
    v.getVector(p2,p3);
    if(CrossProduct(u,v)>0) cout<<"LEFT"<<endl;
    else if(CrossProduct(u,v)<0) cout<<"RIGHT"<<endl;
    else cout<<"TOUCH"<<endl;

}
```

## 14.5 Polygon Area

```
struct point{
    ll x,y;
};

ll CrossP(point a,point b){
    return a.x*b.y-a.y*b.x;
}

inline void solve()
{
    int n; cin>>n;
    ll res=0;
    point p1,p2,p3;
    cin>>p3.x>>p3.y;
    p1.x=p3.x;
    p1.y=p3.y;
    FO(i,n-1)
    {
        cin>>p2.x>>p2.y;
        res+=CrossP(p1,p2);
        p1.x=p2.x;
        p1.y=p2.y;
    }
    res+=CrossP(p1,p3);
    cout<<abs(res)<<endl;
}
```

## 14.6 Convex Hull

```
const ll mod=1e9+7;
const ll limit=4e9;
//ascii https://elcodigoascii.com.ar/

int orientation(point a,point b,point c){
    ll ori=(b.y-c.y)*(b.x-a.x)-(b.y-a.y)*(b.x-c.x);
    if(ori==0) return 0;
    if(ori>0) return 1;
    return 2;
}

void getLastTwo(point &a,point &b,stack<point> &s)
{
    a=s.top();
    s.pop();
```

```
    b=s.top();
    s.pop();
}

void show(point a){
    cout<<a.x<<" "<<a.y<<endl;
}

//Graham scan

void solve(){
    int n; cin>>n;
    vector<point> puntos(n);
    FO(i,n){
        ll a,b; cin>>a>>b;
        puntos[i]=make_pair(a,b);
    }
    sort(all(puntos));
    //Lower Part
    stack<point> lower;
    FO(i,n)
    {
        if(lower.size()<2){
            lower.push(puntos[i]);
            continue;
        }
        point a,b;
        getLastTwo(a,b,lower);
        if(orientation(a,b,puntos[i])<2)
        {
            lower.push(b);
            lower.push(a);
            lower.push(puntos[i]);
        }
        else{
            lower.push(b);
            i--;
        }
    }
    stack<point> upper;
    for(int i=n-1;i>=0;i--)
    {
        if(upper.size()<2){
            upper.push(puntos[i]);
            continue;
        }
        point a,b;
        getLastTwo(a,b,upper);
        if(orientation(a,b,puntos[i])<2)
        {
            upper.push(b);
            upper.push(a);
            upper.push(puntos[i]);
        }
        else{
            upper.push(b);
            i++;
        }
    }

    set<point> res;

    while(!lower.empty()){
        res.insert(lower.top());
        lower.pop();
    }
    while(!upper.empty()){
        res.insert(upper.top());
        upper.pop();
    }
    cout<<res.size()<<endl;
    for(auto c:res) show(c);
}
```

## 14.7 Complex point

```
typedef double T;
typedef complex<T> pt;
#define x real()
#define y imag()

typedef long long ll;
typedef vector<int> vec;
const ll mod=1e9+7;
const int MAX=2e5+3;

//ascii https://elcodigoascii.com.ar/

T norma(pt a){return a.x*a.x+a.y*a.y;}


int sgn(T X){
    return (T(0)<X)-(T(0)>X);
}


pt translate(pt a,pt v){return a+v;}
pt scale(pt p,pt c,T factor){return c+(p-c)*factor;}
pt rot(pt p,T a){return p*polar(1.0,a);}
pt perp(pt p){return pt({-p.y,p.x});}
```

```
pt linearFunc(pt p,pt q,pt r,pt fp,pt fq){
    return fp+(r-p)*(fq-fp)/(q-p);
}
T dot(pt v,pt w){ return v.x*w.x+v.y*w.y;}
T cross(pt v,pt w){ return v.x*w.y-v.y*w.x;}

bool isperp(pt a,pt b){return dot(a,b)==0;}

double angle(pt v,pt w){
    return acos(clamp(dot(v,w)/abs(v)/abs(w),-1.0,-1.0));
}

T orientation(pt a,pt b,pt c){return cross(b-a,c-a);}

bool inAngle(pt a,pt b,pt c,pt p){
    if(orientation(a,b,c)<0) swap(b,c);
    return sgn(orientation(a,b,p))*sgn(orientation(a,c,p))<=0;
}

bool isconvex(vector<pt> p){
    bool hasPos=false,hasNeg=false;
    for(int i=0,n=p.size();i<n;i++){
        int o=orientation(p[i],p[(i+1)%n],p[(i+2)%n]);
        if(o>0) hasPos=true;
        if(o<0) hasNeg=true;
    }
    return !(hasPos && hasNeg);
}

inline void solve()
{
    pt p{3,-4};
    p+=pt({1,2});
    cout<<p<<endl;
    cout<<norma(p)<<endl;
}
```

## 14.8 Polar sort

```
#define x real()
#define y imag()

typedef long long ll;
typedef double T;
typedef complex<T> pt;
typedef vector<int> vec;
const ll mod=1e9+7;
const int MAX=2e5+3;

T cross(pt v,pt w){ return v.x*w.y-v.y*w.x;}
T norma(pt a){return a.x*a.x+a.y*a.y;}
//ascii https://elcodigoascii.com.ar/

bool half(pt p){
    assert(p.x!=0 || p.y!=0);
    return p.y>0 || (p.y==0 && p.x<0);
}

void polarSort(vector<pt> &v){
    sort(all(v),[](pt v,pt w){
        return make_tuple(half(v),0)<make_tuple(half(w),cross(v,w)
            );
    });
}

void polarSortNorm(vector<pt> &v){
    sort(all(v),[](pt v,pt w){
        return make_tuple(half(v),0,norma(v))<make_tuple(half(w),
            cross(v,w),norma(w));
    });
}
inline void solve()
{

}
```

# 15 Strings

## 15.1 Marranadas de Quique

```
//To Upper and Lower
transform(s.begin(), s.end(), s.begin(), ::toupper);
transform(s.begin(), s.end(), s.begin(), ::tolower);

// From i to the end
string a = s.substr(i);
// From i to j
string a = s.substr(i,j);

int a;
int b;
int c;
char comma;
char colon;

// Createa a stringstream object
stringstream ss(fullString);
```

```
// Extract the strings
ss >> a >> colon >> b >> comma >> c;

// String constructor with a char
string result(n, c);
```

## 15.2 KMP Algorithm

```
// LPS for s, lps[i] could also be defined as the longest prefix
     which is also a proper suffix
vi computeLPS(string s){
    size_t len = 0;
    size_t M = s.size();
    vi lps(M, 0);

    size_t i = 1;
    while(i < M) {
        if( s[i] == s[len]){
            len++;
            lps[i] = len;
            i++;
        } else {
            if(len != 0){
                len = lps[len-1];
            } else {
                lps[i] = 0;
                i++;
            }
        }
    }

    return lps;
}

// Get number of occurrences of a pattern in a text using KMP
// O(N+M)
size_t KMPOccurrences(string pattern, string text){
    vi lps = computeLPS(pattern); // LPS array

    size_t M = pattern.size();
    size_t N = text.size();

    size_t i = 0; // Index for text
    size_t j = 0; // Index for pattern

    size_t cnt = 0; // Counter

    while ((N - i) >= (M - j)) {
        // Watch for the pattern
        if (pattern[j] == text[i]) {
            j++;
            i++;
        }

        // If the full match found
        if (j == M) {
            cnt++;
            j = lps[j - 1];
        }

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

    return cnt;
}
```

## 15.3 Rolling Hash

```
// Rolling hash
struct Hash {
    // Prime number and modulo
    long long p = 31, m = 1e9 + 7;
    long long hash_value;
    Hash(const string& s)
    {
        long long hash_so_far = 0;
        long long p_pow = 1;
        const long long n = s.length();
        for (long long i = 0; i < n; ++i) {
            hash_so_far
                = (hash_so_far + (s[i] - 'a' + 1) * p_pow)
                  % m;
            p_pow = (p_pow * p) % m;
        }
        hash_value = hash_so_far;
    }
    bool operator==(const Hash& other)
    {
        return (hash_value == other.hash_value);
    }
```

```
};

// Usage
int main(){
    string s = "hello";

    return 0;
}
```

## 15.4 Hash marrano

```
vector<vector<int>> group_identical_strings(vector<string> const&
    s) {
    int n = s.size();
    vector<pair<long long, int>> hashes(n);
    for (int i = 0; i < n; i++)
        hashes[i] = {compute_hash(s[i]), i};

    sort(hashes.begin(), hashes.end());

    vector<vector<int>> groups;
    for (int i = 0; i < n; i++) {
        if (i == 0 || hashes[i].first != hashes[i-1].first)
            groups.emplace_back();
        groups.back().push_back(hashes[i].second);
    }
    return groups;
}
```

## 15.5 Suffix Array

```
// Structure to store information of a suffix
struct suffix
{
    int index;
    char *suff;
};

// A comparison function used by sort() to compare two suffixes
int cmp(struct suffix a, struct suffix b)
{
    return strcmp(a.suff, b.suff) < 0? 1 : 0;
}

// This is the main function that takes a string 'txt' of size n
    as an
// argument, builds and return the suffix array for the given
    string
int *buildSuffixArray(char *txt, int n)
{
    // A structure to store suffixes and their indexes
    struct suffix suffixes[n];

    // Store suffixes and their indexes in an array of structures.
    // The structure is needed to sort the suffixes alphabetically
    // and maintain their old indexes while sorting
    for (int i = 0; i < n; i++)
    {
        suffixes[i].index = i;
        suffixes[i].suff = (txt+i);
    }

    // Sort the suffixes using the comparison function
    // defined above.
    sort(suffixes, suffixes+n, cmp);

    // Store indexes of all sorted suffixes in the suffix array
    int *suffixArr = new int[n];
    for (int i = 0; i < n; i++)
        suffixArr[i] = suffixes[i].index;

    // Return the suffix array
    return  suffixArr;
}

// A utility function to print an array of given size
void printArr(int arr[], int n)
{
    for(int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}
```

## 15.6 LCP

```
// Structure to store information of a suffix
struct suffix
{
    int index;  // To store original index
    int rank[2]; // To store ranks and next rank pair
};

// A comparison function used by sort() to compare two suffixes
// Compares two pairs, returns 1 if first pair is smaller
int cmp(struct suffix a, struct suffix b)
{
```

```
    return (a.rank[0] == b.rank[0])? (a.rank[1] < b.rank[1] ?1: 0) :
           (a.rank[0] < b.rank[0] ?1: 0);
}

// This is the main function that takes a string 'txt' of size n as an
// argument, builds and return the suffix array for the given string
vector<int> buildSuffixArray(string txt, int n)
{
    // A structure to store suffixes and their indexes
    struct suffix suffixes[n];

    // Store suffixes and their indexes in an array of structures.
    // The structure is needed to sort the suffixes alphabetically
    // and maintain their old indexes while sorting
    for (int i = 0; i < n; i++)
    {
        suffixes[i].index = i;
        suffixes[i].rank[0] = txt[i] - 'a';
        suffixes[i].rank[1] = ((i+1) < n)? (txt[i + 1] - 'a'): -1;
    }

    // Sort the suffixes using the comparison function
    // defined above.
    sort(suffixes, suffixes+n, cmp);

    // At his point, all suffixes are sorted according to first
    // 2 characters.  Let us sort suffixes according to first 4
    // characters, then first 8 and so on
    int ind[n];  // This array is needed to get the index in suffixes[]
    // from original index.  This mapping is needed to get
    // next suffix.
    for (int k = 4; k < 2*n; k = k*2)
    {
        // Assigning rank and index values to first suffix
        int rank = 0;
        int prev_rank = suffixes[0].rank[0];
        suffixes[0].rank[0] = rank;
        ind[suffixes[0].index] = 0;

        // Assigning rank to suffixes
        for (int i = 1; i < n; i++)
        {
            // If first rank and next ranks are same as that of previous
            // suffix in array, assign the same new rank to this suffix
            if (suffixes[i].rank[0] == prev_rank &&
                    suffixes[i].rank[1] == suffixes[i-1].rank[1])
            {
                prev_rank = suffixes[i].rank[0];
                suffixes[i].rank[0] = rank;
            }
            else // Otherwise increment rank and assign
            {
                prev_rank = suffixes[i].rank[0];
                suffixes[i].rank[0] = ++rank;
            }
            ind[suffixes[i].index] = i;
        }

        // Assign next rank to every suffix
        for (int i = 0; i < n; i++)
        {
            int nextindex = suffixes[i].index + k/2;
            suffixes[i].rank[1] = (nextindex < n)?
                                  suffixes[ind[nextindex]].rank[0]: -1;
        }

        // Sort the suffixes according to first k characters
        sort(suffixes, suffixes+n, cmp);
    }

    // Store indexes of all sorted suffixes in the suffix array
    vector<int>suffixArr;
    for (int i = 0; i < n; i++)
        suffixArr.push_back(suffixes[i].index);

    // Return the suffix array
    return  suffixArr;
}

/* To construct and return LCP */
vector<int> kasai(string txt, vector<int> suffixArr)
{
    int n = suffixArr.size();

    // To store LCP array
    vector<int> lcp(n, 0);

    // An auxiliary array to store inverse of suffix array
    // elements. For example if suffixArr[0] is 5, the
    // invSuff[5] would store 0.  This is used to get next
    // suffix string from suffix array.
    vector<int> invSuff(n, 0);

    // Fill values in invSuff[]
    for (int i=0; i < n; i++)
```

```
        invSuff[suffixArr[i]] = i;

    // Initialize length of previous LCP
    int k = 0;

    // Process all suffixes one by one starting from
    // first suffix in txt[]
    for (int i=0; i<n; i++)
    {
        /* If the current suffix is at n-1, then we dont
           have next substring to consider. So lcp is not
           defined for this substring, we put zero. */
        if (invSuff[i] == n-1)
        {
            k = 0;
            continue;
        }

        /* j contains index of the next substring to
           be considered  to compare with the present
           substring, i.e., next string in suffix array */
        int j = suffixArr[invSuff[i]+1];

        // Directly start matching from k'th index as
        // at-least k-1 characters will match
        while (i+k<n && j+k<n && txt[i+k]==txt[j+k])
            k++;

        lcp[invSuff[i]] = k; // lcp for the present suffix.

        // Deleting the starting character from the string.
        if (k>0)
            k--;
    }

    // return the constructed lcp array
    return lcp;
}

// Utility function to print an array
void printArr(vector<int>arr, int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}
```

## 15.7 Z Function

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

## 15.8 Longest Palindrome

```
typedef long long ll;
typedef vector<int> vec;
const ll mod=1e9+7;
const int MAX=1e6+3;
vector<int> lps(2*MAX);
int n;
string s;

//ascii https://elcodigoascii.com.ar/

void show(int idx)
{
    int start=(idx-lps[idx])/2;
    int end=start+lps[idx];
    for(int i=start;i<end;i++){
        cout<<s[i];
    }

}

inline void solve()
{
    cin>>s;
    n=s.size();
    lps[0]=0;
    lps[1]=1;
    int rightCenter,leftCenter,center,curRightCenter,curLeftCenter
        ;
```

```
    center=1;
    rightCenter=center+lps[center];
    leftCenter=center-lps[center];
    int maxLPScenter=1;
    int diff=-1;
    bool exp;
    for(curRightCenter=2;curRightCenter<2*n+1;curRightCenter++)
    {
        //Condicion de cambio de centro
        curLeftCenter=2*center-curRightCenter;
        diff=rightCenter-curRightCenter;
        exp=false;
        if(diff>=0){
            if(lps[curLeftCenter]<diff){
                lps[curRightCenter]=lps[curLeftCenter];
            }
            else if(lps[curLeftCenter]==diff && rightCenter==2*n)
            {
                lps[curRightCenter]=lps[curLeftCenter];
            }
            else if(lps[curLeftCenter]==diff && rightCenter<2*n){
                lps[curRightCenter]=lps[curLeftCenter];
                exp=true;
            }
            else if(lps[curLeftCenter]>diff){
                lps[curRightCenter]=diff;
                exp=true;
            }
        }
        else{
            lps[curRightCenter]=0;
            exp=true;
        }
        if(exp)
        {
            while(((curRightCenter+lps[curRightCenter])<2*n &&
                curRightCenter-lps[curRightCenter]>0)
            && ((curRightCenter+lps[curRightCenter]+1)%2==0 || s[(
                curRightCenter+lps[curRightCenter]+1)/2]==s[(
                curRightCenter-lps[curRightCenter]-1)/2])){
                lps[curRightCenter]++;
            }
        }
        if(lps[curRightCenter]>lps[maxLPScenter])
        {
            maxLPScenter=curRightCenter;
        }
        if(curRightCenter+lps[curRightCenter]>rightCenter){
            center=curRightCenter;
            rightCenter=curRightCenter+lps[curRightCenter];
        }
    }
    show(maxLPScenter);

}
```

## 15.9 String Hashing

```
typedef long long ll;
typedef vector<int> vec;
const ll mod=1e9+7;
const int MAX=1e6+3;
const ll A=911382323;
const ll B=972663749;
ll str[MAX];
ll pk[MAX];
bool prefix[MAX]={false};



ll subs(int i,int j)
{
    if(i)
        return ((str[j]-pk[j-i+1]*str[i-1])%B+B)%B;
    else
        return str[j];
}

//ascii https://elcodigoascii.com.ar/

inline void solve()
{
    string s; cin>>s;
    memset(prefix,true,sizeof(prefix));
    str[0]=s[0];
    pk[0]=1;
    int n=s.size();
    for(int i=1;i<n;i++)
    {
        str[i]=A*str[i-1]+s[i];
        pk[i]=pk[i-1]*A;
        pk[i]%=B;
        str[i]%=B;
    }
    ll aux;
    bool ver;
    for(int i=1;i<=n;i++)
    {
        aux=subs(0,i-1);
        for(int j=0;j+i<=n;j+=i)
```

```
        {
            if(aux!=subs(j,j+i-1))
            {
                //cout<<aux<<" "<<subs(j,j+i-1)<<" "<<i<<" "<<j<<endl;
                prefix[i]=false;
                break;
            }
        }
        if(!prefix[i]) continue;
        if(n%i && (subs(n-n%i,n-1)!=subs(0,n%i-1)))
        {
            continue;
        }
        cout<<i<<" ";
    }
}
```

## 15.10 Manacher Algorithm

```
vector<int> manacher(string s) {
    string t;
    for(auto c: s) {
        t += string("#") + c;
    }
    auto res = manacher_odd(t + "#");
    return vector<int>(begin(res) + 1, end(res) - 1);
}
```

## 15.11 Suffix Automaton

```
struct state {
    int len, link;
    map<char, int> next;
};

const int MAXLEN = 100000;
state st[MAXLEN * 2];
int sz, last;

void sa_init() {
    st[0].len = 0;
    st[0].link = -1;
    sz++;
    last = 0;
}

void sa_extend(char c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    int p = last;
    while (p != -1 && !st[p].next.count(c)) {
        st[p].next[c] = cur;
        p = st[p].link;
    }
    if (p == -1) {
        st[cur].link = 0;
    } else {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len) {
            st[cur].link = q;
        } else {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            while (p != -1 && st[p].next[c] == q) {
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}

long long get_diff_strings(){
    long long tot = 0;
    for(int i = 1; i < sz; i++) {
        tot += st[i].len - st[st[i].link].len;
    }
    return tot;
}

long long get_tot_len_diff_substings() {
    long long tot = 0;
    for(int i = 1; i < sz; i++) {
        long long shortest = st[st[i].link].len + 1;
        long long longest = st[i].len;

        long long num_strings = longest - shortest + 1;
        long long cur = num_strings * (longest + shortest) / 2;
        tot += cur;
    }
    return tot;
}

string lcs (string S, string T) {
    sa_init();
```

```
    for (int i = 0; i < S.size(); i++)
        sa_extend(S[i]);

    int v = 0, l = 0, best = 0, bestpos = 0;
    for (int i = 0; i < T.size(); i++) {
        while (v && !st[v].next.count(T[i])) {
            v = st[v].link ;
            l = st[v].len;
        }
        if (st[v].next.count(T[i])) {
            v = st [v].next[T[i]];
            l++;
        }
        if (l > best) {
            best = l;
            bestpos = i;
        }
    }
    return T.substr(bestpos - best + 1, best);
}
```

# 16 Formulas

## 16.1 Sums

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1} - c^a}{c-1}, c \neq 1$$

**Gauss**

$$1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}$$

**Gauss squares**

$$1^2 + 2^2 + 3^2 + ... + n^2 = \frac{n(2n+1)(n+1)}{6}$$

**Cubes**

$$1^3 + 2^3 + 3^3 + ... + n^3 = \frac{n^2(n+1)^2}{4}$$

**Powers of 4**

$$1^4 + 2^4 + 3^4 + ... + n^4 = \frac{n(2n+1)(n+1)(3n^2+3n-1)}{30}$$

## 16.2 Catalan numbers

$$C_0 = 1, \quad C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} \quad \text{(Recursive)}$$

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!} \quad \text{(Closed-form)}$$

- **Valid Parentheses**: Count of balanced parentheses expressions with $n$ pairs.
- **Full Binary Trees**: Structurally unique full binary trees with $n+1$ leaves.
- **Polygon Triangulation**: Ways to triangulate a convex $(n+2)$-gon.
- **Dyck Paths**: Paths from $(0,0)$ to $(2n,0)$ that never dip below the x-axis.
- **Non-Crossing Partitions**: Ways to connect $2n$ points on a circle without crossing chords.
- **Stack Permutations**: Valid stack-sortable permutations of length $n$.
- **Mountain Ranges**: Sequences of $2n$ up/down steps forming valid mountain ranges.
- **Unique BSTs**: Number of distinct binary search trees with $n$ keys.
- **Diagonal-Avoiding Paths**: Paths in a grid from $(0,0)$ to $(n,n)$ without crossing the diagonal.

## 16.3 Cayley's Formula

Number of labeled trees of n vertices: $n^{n-2}$.

Number of rooted forest of n vertices is: $(n+1)^{n-1}$

## 16.4 Geometric series

**Finite:**

$$\sum_{k=0}^{n} ar^k = \begin{cases} a\dfrac{1-r^{n+1}}{1-r} & \text{if } r \neq 1, \\ a(n+1) & \text{if } r = 1. \end{cases}$$

**Infinite:**

$$\sum_{k=0}^{\infty} ar^k = \frac{a}{1-r} \quad \text{(converges iff } |r| < 1)$$

### 16.5 Divisors

The number of divisors of any number n is:

$$\begin{cases} \approx 100 & n < 5 \times 10^4 \\ \approx 500 & n < 1 \times 10^7 \\ \approx 2000 & n < 1 \times 10^10 \\ \approx 200000 & n < 1 \times 10^19 \end{cases}$$

### 16.6 Number of primes between 1 and n

$$\frac{n}{ln(n)}$$

### 16.7 Pythagorean triplets

$$a = k \cdot (m^2 - n^2), \quad b = k \cdot (2mn), \quad c = k \cdot (m^2 + n^2)$$

With $m > n > 0$, $k = 0$, $m \perp n$, and either m or n even.

### 16.8 Derangments

Permutations of a set sush that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1) + D(n-2)) = nD(n-1) + (-1)^n = \lfloor \frac{n!}{e} \rfloor$$

## 17 Miscellaneous

### 17.1 Implementation tricks

```
// Read full line
string s;
getline(cin, s);

// Read while input is provided
while(getline(cin, s))

// Print n leading zeros
cout << setw(n) << setfill('0') << x << endl;
```

### 17.2 Get Least Significant Bit

```
int getLestSignificantBit(int i) {
    return i & -i;
}
```

### 17.3 Is power of two?

```
bool isPowerOfTwo(int n) {
    return (n > 0) && ((n & (n - 1)) == 0);
}
```

### 17.4 Random number generator

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

uniform_int_distribution<int>(0,n)
normal_distribution<> normal_dist(mean,2)
exponential_distribution
```

### 17.5 Custom comparators

```
bool cmp(const Edge &x, const Edge &y) {return x.w < y.w}
```

### 17.6 Kadane's Algorithm

```
inline void solve()
{
    int n; cin>>n;
    vector<int> normal(n);
    vector<int> rever(n);
    FO(i,n){
        cin>>normal[i];
        rever[i]=-normal[i];
    }
    ll sum = 0, max_sum = -1e9;
    ll sumr=0;
    for (int i = 0; i < n; i++) {
        sum += normal[i];
        max_sum = max(max_sum, sum);
        sumr+= rever[i];
        max_sum=max(max_sum,sumr);
        if(i%2==1){
            sum=max(sum,sumr);
```

```cpp
            sumr=max(sum,sumr);
        }
        if (sum < 0) sum = 0;
        if (sumr<0) sumr=0;
    }
    cout<<max_sum<<endl;
    //Geeks for geeks
    //https://www.geeksforgeeks.org/cses-solutions-maximum-subarray-sum/
}
```

## 17.7 Moore's Voting Algorithm

```cpp
int majorityElement(vector<int>& nums) {
    int vote = 0, r = 0;
    for(int i=0; i<nums.size();i++){
        if(nums[i] == nums[r])
            vote++;
        else
            vote--;
        if(vote == 0){
            r = i;
            vote = 1;
        }
    }

    int cnt = 0;
    int goal = (nums.size())/2;
    for(int i=0; i<nums.size(); i++){
        if(nums[i] == nums[r]){
            cnt++;
            if(cnt > goal){
                break;
            }
        }
    }

    return nums[r];
}
```

## 17.8 ASCII table

**ASCII TABLE**

| Decimal | Hexadecimal | Binary | Octal | Char |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | [NULL] |
| 1 | 1 | 1 | 1 | [START OF HEADING] |
| 2 | 2 | 10 | 2 | [START OF TEXT] |
| 3 | 3 | 11 | 3 | [END OF TEXT] |
| 4 | 4 | 100 | 4 | [END OF TRANSMISSION] |
| 5 | 5 | 101 | 5 | [ENQUIRY] |
| 6 | 6 | 110 | 6 | [ACKNOWLEDGE] |
| 7 | 7 | 111 | 7 | [BELL] |
| 8 | 8 | 1000 | 10 | [BACKSPACE] |
| 9 | 9 | 1001 | 11 | [HORIZONTAL TAB] |
| 10 | A | 1010 | 12 | [LINE FEED] |
| 11 | B | 1011 | 13 | [VERTICAL TAB] |
| 12 | C | 1100 | 14 | [FORM FEED] |
| 13 | D | 1101 | 15 | [CARRIAGE RETURN] |
| 14 | E | 1110 | 16 | [SHIFT OUT] |
| 15 | F | 1111 | 17 | [SHIFT IN] |
| 16 | 10 | 10000 | 20 | [DATA LINK ESCAPE] |
| 17 | 11 | 10001 | 21 | [DEVICE CONTROL 1] |
| 18 | 12 | 10010 | 22 | [DEVICE CONTROL 2] |
| 19 | 13 | 10011 | 23 | [DEVICE CONTROL 3] |
| 20 | 14 | 10100 | 24 | [DEVICE CONTROL 4] |
| 21 | 15 | 10101 | 25 | [NEGATIVE ACKNOWLEDGE] |
| 22 | 16 | 10110 | 26 | [SYNCHRONOUS IDLE] |
| 23 | 17 | 10111 | 27 | [END OF TRANS. BLOCK] |
| 24 | 18 | 11000 | 30 | [CANCEL] |
| 25 | 19 | 11001 | 31 | [END OF MEDIUM] |
| 26 | 1A | 11010 | 32 | [SUBSTITUTE] |
| 27 | 1B | 11011 | 33 | [ESCAPE] |
| 28 | 1C | 11100 | 34 | [FILE SEPARATOR] |
| 29 | 1D | 11101 | 35 | [GROUP SEPARATOR] |
| 30 | 1E | 11110 | 36 | [RECORD SEPARATOR] |
| 31 | 1F | 11111 | 37 | [UNIT SEPARATOR] |
| 32 | 20 | 100000 | 40 | [SPACE] |
| 33 | 21 | 100001 | 41 | ! |
| 34 | 22 | 100010 | 42 | " |
| 35 | 23 | 100011 | 43 | # |
| 36 | 24 | 100100 | 44 | $ |
| 37 | 25 | 100101 | 45 | % |
| 38 | 26 | 100110 | 46 | & |
| 39 | 27 | 100111 | 47 | ' |
| 40 | 28 | 101000 | 50 | ( |
| 41 | 29 | 101001 | 51 | ) |
| 42 | 2A | 101010 | 52 | * |
| 43 | 2B | 101011 | 53 | + |
| 44 | 2C | 101100 | 54 | , |
| 45 | 2D | 101101 | 55 | - |
| 46 | 2E | 101110 | 56 | . |
| 47 | 2F | 101111 | 57 | / |

| Decimal | Hexadecimal | Binary | Octal | Char |
|---|---|---|---|---|
| 48 | 30 | 110000 | 60 | 0 |
| 49 | 31 | 110001 | 61 | 1 |
| 50 | 32 | 110010 | 62 | 2 |
| 51 | 33 | 110011 | 63 | 3 |
| 52 | 34 | 110100 | 64 | 4 |
| 53 | 35 | 110101 | 65 | 5 |
| 54 | 36 | 110110 | 66 | 6 |
| 55 | 37 | 110111 | 67 | 7 |
| 56 | 38 | 111000 | 70 | 8 |
| 57 | 39 | 111001 | 71 | 9 |
| 58 | 3A | 111010 | 72 | : |
| 59 | 3B | 111011 | 73 | ; |
| 60 | 3C | 111100 | 74 | < |
| 61 | 3D | 111101 | 75 | = |
| 62 | 3E | 111110 | 76 | > |
| 63 | 3F | 111111 | 77 | ? |
| 64 | 40 | 1000000 | 100 | @ |
| 65 | 41 | 1000001 | 101 | A |
| 66 | 42 | 1000010 | 102 | B |
| 67 | 43 | 1000011 | 103 | C |
| 68 | 44 | 1000100 | 104 | D |
| 69 | 45 | 1000101 | 105 | E |
| 70 | 46 | 1000110 | 106 | F |
| 71 | 47 | 1000111 | 107 | G |
| 72 | 48 | 1001000 | 110 | H |
| 73 | 49 | 1001001 | 111 | I |
| 74 | 4A | 1001010 | 112 | J |
| 75 | 4B | 1001011 | 113 | K |
| 76 | 4C | 1001100 | 114 | L |
| 77 | 4D | 1001101 | 115 | M |
| 78 | 4E | 1001110 | 116 | N |
| 79 | 4F | 1001111 | 117 | O |
| 80 | 50 | 1010000 | 120 | P |
| 81 | 51 | 1010001 | 121 | Q |
| 82 | 52 | 1010010 | 122 | R |
| 83 | 53 | 1010011 | 123 | S |
| 84 | 54 | 1010100 | 124 | T |
| 85 | 55 | 1010101 | 125 | U |
| 86 | 56 | 1010110 | 126 | V |
| 87 | 57 | 1010111 | 127 | W |
| 88 | 58 | 1011000 | 130 | X |
| 89 | 59 | 1011001 | 131 | Y |
| 90 | 5A | 1011010 | 132 | Z |
| 91 | 5B | 1011011 | 133 | [ |
| 92 | 5C | 1011100 | 134 | \ |
| 93 | 5D | 1011101 | 135 | ] |
| 94 | 5E | 1011110 | 136 | ^ |
| 95 | 5F | 1011111 | 137 | _ |

| Decimal | Hexadecimal | Binary | Octal | Char |
|---|---|---|---|---|
| 96 | 60 | 1100000 | 140 | ` |
| 97 | 61 | 1100001 | 141 | a |
| 98 | 62 | 1100010 | 142 | b |
| 99 | 63 | 1100011 | 143 | c |
| 100 | 64 | 1100100 | 144 | d |
| 101 | 65 | 1100101 | 145 | e |
| 102 | 66 | 1100110 | 146 | f |
| 103 | 67 | 1100111 | 147 | g |
| 104 | 68 | 1101000 | 150 | h |
| 105 | 69 | 1101001 | 151 | i |
| 106 | 6A | 1101010 | 152 | j |
| 107 | 6B | 1101011 | 153 | k |
| 108 | 6C | 1101100 | 154 | l |
| 109 | 6D | 1101101 | 155 | m |
| 110 | 6E | 1101110 | 156 | n |
| 111 | 6F | 1101111 | 157 | o |
| 112 | 70 | 1110000 | 160 | p |
| 113 | 71 | 1110001 | 161 | q |
| 114 | 72 | 1110010 | 162 | r |
| 115 | 73 | 1110011 | 163 | s |
| 116 | 74 | 1110100 | 164 | t |
| 117 | 75 | 1110101 | 165 | u |
| 118 | 76 | 1110110 | 166 | v |
| 119 | 77 | 1110111 | 167 | w |
| 120 | 78 | 1111000 | 170 | x |
| 121 | 79 | 1111001 | 171 | y |
| 122 | 7A | 1111010 | 172 | z |
| 123 | 7B | 1111011 | 173 | { |
| 124 | 7C | 1111100 | 174 | \| |
| 125 | 7D | 1111101 | 175 | } |
| 126 | 7E | 1111110 | 176 | ~ |
| 127 | 7F | 1111111 | 177 | [DEL] |

# 18 C++ stuff

## 18.1 Compilation

```
g++-13 -std=c++20 name.cpp
```

## 18.2 Compiler optimizations

```cpp
// Makes bit operations faster
#pragma GCC target("popcnt")

//Auto vectorize for-loops and optimizes floating points (assumes associativity and turns off denormals)
#pragma GCC optimize("Ofast")

// Doubles performance of vectorized code, crashes in old computers
#pragma GCC target("avx2")

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

## 18.3 Decimal printing

Friendly reminder to use `printf()` with decimals

```
cout<< fixed << setprecision(n)<<endl;
```

## 18.4 Bit tricks

x & `-x` is the least bit in x

c = x & -x, r=x+c, `(((bin_pow(r,x)) >> 2)/c) OR r` next number bigger than x same number of bits set.