

Contents

1 Funciones C++	2
2 Data Structures	3
2.1 SparseTable	3
2.2 Segment Tree	3
2.3 Segment Tree Max Subarray Sum	3
2.4 MergeSortTree	4
2.5 LazySegmentTree	4
2.6 PersistentSegmentTree	6
2.7 Persistent Array	7
2.8 Fenwick	7
2.9 Fenwick 2D	8
2.10 SegmentTree 2D	8
2.11 DynamicSegTree	9
2.12 OrderedSet	10
2.13 Disjoint Set Union	10
2.14 BitTrie	11
2.15 Palindromic Tree	11
2.16 LineContainer	12
2.17 SqrtDecomposition	12
2.18 Treap	13
2.19 Dynamnic Connectivity	14
2.20 Bitset	16
3 Math	16
3.1 Binary Exponentiation And Modular Division	16
3.2 Euler's Totient	16
3.3 Miller Rabin	16
3.4 Diophantine	17
3.5 Discrete Log	18
4 Polynomials	19
4.1 Fast Fourier Transform (FFT)	19
4.2 Number Theoretic Transform (NTT)	19
4.3 Berlekamp-Massey	20
5 Linear Algebra	21
5.1 Simplex	21
6 Graphs	22
6.1 Djikstra	22
6.2 FloydWarshall	22

6.3 BellmanFord	23
6.4 BellmanFordFast(SPFA)	23
6.5 EulerianPath	24
6.6 Kruskal	24
6.7 Strongly Connected Components	24
6.8 2Sat	24
6.9 Finding Bridges	25
6.10 Finding Bridges Online	26
6.11 Articulation Points	27
6.12 Biconnected Components (Block Cut Tree)	27
6.13 Blossom	28
6.14 Marriage	30
7 Trees	30
7.1 Tree Diameter	30
7.2 Heavy Light Decomposition	30
7.3 Lowest Common Ancestor (LCA)	32
7.4 Centroid Decomposition	33
8 Flows	33
8.1 Dinic	33
8.2 Hungarian	34
8.3 Min-cost Max-Flow	35
9 Strings	37
9.1 KMP-Prefix Function	37
9.2 Z-Function	37
9.3 Manacher	37
9.4 String Hashing	37
9.5 MinCyclicRotation	38
9.6 Suffix Array	38
9.7 Trie AhoCorasick	39
9.8 SuffixAutomaton	40
10 DP	42
10.1 Digit DP	42
10.2 Convex Hull Trick Deque	42
10.3 Longest Common Subsequence(LCS)	42
10.4 Edit Distance	43
10.5 Longest Increasing Subsequence(LIS)	43
11 Geometry	43
11.1 Geometry Primitives	43

11.2 Area of simple Polygon	45
11.3 Point Inside Convex Polygon	45
11.4 Nearest Pair Of Points	45
11.5 ConvexHull	46
11.6 Rotating Calipers	46
12 Miscellaneous	47
12.1 Custom Comparators	47
12.2 Random Number Generator	47
12.3 Int Ternary Search	48
12.4 Ternary Search	48
12.5 Parallel Binary Search	48
12.6 Next and Prev Smaller/Greater	48
12.7 2D Prefix Sum	49
12.8 Day of Week	49
12.9 Iterating over all subsets of mask	49
12.10 Int 128	49
12.11 XOR Basis	50
12.12 XOR Convolution	50
12.13 GCD Convolution	50
12.14 LCM Convolution	51
12.15 OR, AND Convolution	51
12.16 Mo's Algorithm	51
12.17 Matrix Exponentiation	52
12.18 Sprague-Grundy Theorem	52
13 Stress Testing Scripts	52
13.1 build.sh	52
13.2 stress.sh	53
13.3 validate.sh	53
14 Useful things	53
14.1 Sums	53
14.2 Catalan numbers	53
14.3 Cayley's formula	54
14.4 Geometric series	54
14.5 Estimates For Divisors	54
14.6 Sum of divisors	54
14.7 Pythagorean Triplets	54
14.8 Derangements	54
15 C++ things	54

1 Funciones C++

```
#include <algorithm> #include <numeric>
```

Algo	Params	Funcion
sort, stable_sort	f, l	ordena el intervalo
nth_element	f, nth, l	void ordena el n-esimo, y particiona el resto
fill, fill_n	f, l / n, elem	void llena [f, l] o [f, f+n) con elem
lower_bound, upper_bound	f, l, elem	it al primer / ultimo donde se puede insertar elem para que quede ordenada
binary_search	f, l, elem	bool esta elem en [f, l)
copy	f, l, resul	hace resul+i=f+i $\forall i$
find, find_if, find_first_of	f, l, elem / pred / f2, l2	it encuentra i $\in [f,l)$ tq. i=elem, pred(i), i $\in [f2,l2)$
count, count_if	f, l, elem/pred	cuenta elem, pred(i)
search	f, l, f2, l2	busca [f2,l2) $\in [f,l)$
replace, replace_if	f, l, old / pred, new	cambia old / pred(i) por new
reverse	f, l	da vuelta
partition, stable_partition	f, l, pred	pred(i) ad, !pred(i) atras
min_element, max_element	f, l, [comp]	it min, max de [f,l]
lexicographical_compare	f1,l1,f2,l2	bool con [f1,l1][f2,l2]
next/prev_permutation	f,l	deja en [f,l) la perm sig, ant
set_intersection, set_difference, set_union, set_symmetric_difference,	f1, l1, f2, l2, res	[res, ...) la op. de conj
push_heap, pop_heap, make_heap	f, l, e / e /	mete/saca e en heap [f,l), hace un heap de [f,l)
is_heap	f,l	bool es [f,l) un heap
accumulate	f,l,i,[op]	$T = \sum$ /oper de [f,l)
inner_product	f1, l1, f2, i	$T = i + [f1, l1] \cdot [f2, \dots)$
partial_sum	f, l, r, [op]	$r+i = \sum$ /oper de [f,f+i] $\forall i \in [f,l)$
_builtin_ffs	unsigned int	Pos. del primer 1 desde la derecha
_builtin_clz	unsigned int	Cant. de ceros desde la izquierda.
_builtin_ctz	unsigned int	Cant. de ceros desde la derecha.
_builtin_popcount	unsigned int	Cant. de 1's en x.
_builtin_parity	unsigned int	1 si x es par, 0 si es impar.
_builtin_XXXXXXll	unsigned ll	= pero para long long's.

2 Data Structures

2.1 SparseTable

```

1 const int MAXN=100005, K=30;
2 int lg[MAXN+1];
3 int st[K + 1][MAXN];
4
5 int mini(int L, int R){
6     int i = lg[R - L + 1];
7     int minimum = min(st[i][L], st[i][R - (1 << i) + 1]);
8     return minimum;
9 }
10
11 int main(){
12     lg[1]=0;
13     for (int i = 2; i <= MAXN; i++)
14         lg[i] = lg[i/2] + 1;
15     std::copy(a.begin(), a.end(), st[0]);
16
17     for (int i = 1; i <= K; i++)
18         for (int j = 0; j + (1 << i) <= n; j++)
19             st[i][j] = min(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
20 }
```

2.2 Segment Tree

```

1 struct SegmentTree {
2     vector<ll> a;
3     int n;
4
5     SegmentTree(int _n) : a(2 * _n, 1e18), n(_n) {}
6
7     void update(int pos, ll val) {
8         for (a[pos += n] = val; pos > 1; pos >>= 1) {
9             a[pos / 2] = min(a[pos], a[pos ^ 1]);
10        }
11    }
12
13    ll get(int l, int r) {
14        ll res = 1e18;
15        for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
16            if (l & 1) {
```

```

17                res = min(res, a[l++]);
18            }
19            if (r & 1) {
20                res = min(res, a[--r]);
21            }
22        }
23        return res;
24    }
25 };
```

2.3 Segment Tree Max Subarray Sum

```

1 const ll inf=1e18;
2
3 struct Node {
4     ll maxi, l_max, r_max, sum;
5
6     Node(ll _maxi, ll _l_max, ll _r_max, ll _sum){
7         maxi=_maxi;
8         l_max=_l_max;
9         r_max=_r_max;
10        sum=_sum;
11    }
12
13    Node operator+(Node b) {
14        return {max(max(maxi, b.maxi), r_max + b.l_max),
15                max(l_max, sum + b.l_max), max(b.r_max, r_max + b.sum),
16                sum + b.sum};
17    }
18
19 };
20
21 struct SegmentTreeMaxSubSum{
22     int n;
23     vector<Node> t;
24
25     SegmentTreeMaxSubSum(int _n) : n(_n), t(2 * _n, Node(-inf, -inf, -inf,
26                                         -inf)) {}
27
28     void update(int pos, ll val) {
29         t[pos += n] = Node(val, val, val, val);
30         for (pos>>=1; pos ; pos >>= 1) {
31             t[pos] = t[2*pos]+t[2*pos+1];
```

```

31     }
32 }
33
34 Node query(int l, int r) {
35     Node node_l = Node(-inf, -inf, -inf, -inf);
36     Node node_r = Node(-inf, -inf, -inf, -inf);
37     for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
38         if (l & 1) {
39             node_l=node_l+t[l++];
40         }
41         if (r & 1) {
42             node_r=t[--r]+node_r;
43         }
44     }
45     return node_l+node_r;
46 }
47 };

```

2.4 MergeSortTree

```

1 vector<int> t[200005];
2 int a[100005];
3 int n;
4
5 void build(){
6     for(int i=0;i<n;i++){
7         t[i+n].push_back(a[i]);
8     }
9     for(int i=n-1;i;i--){
10        auto b=t[2*i], c=t[2*i+1];
11        merge(b.begin(), b.end(), c.begin(), c.end(), back_inserter(t[i]));
12    }
13 }
14
15
16 int q(int l, int r, int mid) {
17     int res = 0;
18     for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
19         if (l&1){
20             res+=upper_bound(all(t[l])), mid)-t[l].begin();
21             l++;
22         }
23         if (r&1){

```

```

24             r--;
25             res+=upper_bound(all(t[r])), mid)-t[r].begin();
26         }
27     }
28     return res;
29 }

```

2.5 LazySegmentTree

```

1 template <typename num_t>
2 struct segtree {
3     int n, depth;
4     vector<num_t> tree, lazy;
5
6     void init(int s, long long* arr) {
7         n = s;
8         tree = vector<num_t>(4 * s, 0);
9         lazy = vector<num_t>(4 * s, 0);
10        init(0, 0, n - 1, arr);
11    }
12
13    num_t init(int i, int l, int r, long long* arr) {
14        if (l == r) return tree[i] = arr[l];
15
16        int mid = (l + r) / 2;
17        num_t a = init(2 * i + 1, l, mid, arr),
18                b = init(2 * i + 2, mid + 1, r, arr);
19        return tree[i] = a.op(b);
20    }
21
22    void update(int l, int r, num_t v) {
23        if (l > r) return;
24        update(0, 0, n - 1, l, r, v);
25    }
26
27    num_t update(int i, int tl, int tr, int ql, int qr, num_t v) {
28        eval_lazy(i, tl, tr);
29
30        if (tr < ql || qr < tl) return tree[i];
31        if (ql <= tl && tr <= qr) {
32            lazy[i] = lazy[i].val + v.val;
33            eval_lazy(i, tl, tr);
34            return tree[i];

```

```

35 }
36
37     int mid = (tl + tr) / 2;
38     num_t a = update(2 * i + 1, tl, mid, ql, qr, v),
39         b = update(2 * i + 2, mid + 1, tr, ql, qr, v);
40     return tree[i] = a.op(b);
41 }
42
43 num_t query(int l, int r) {
44     if (l > r) return num_t::null_v;
45     return query(0, 0, n-1, l, r);
46 }
47
48 num_t query(int i, int tl, int tr, int ql, int qr) {
49     eval_lazy(i, tl, tr);
50
51     if (ql <= tl && tr <= qr) return tree[i];
52     if (tr < ql || qr < tl) return num_t::null_v;
53
54     int mid = (tl + tr) / 2;
55     num_t a = query(2 * i + 1, tl, mid, ql, qr),
56         b = query(2 * i + 2, mid + 1, tr, ql, qr);
57     return a.op(b);
58 }
59
60 void eval_lazy(int i, int l, int r) {
61     tree[i] = tree[i].lazy_op(lazy[i], (r - l + 1));
62     if (l != r) {
63         lazy[i * 2 + 1] = lazy[i].val + lazy[i * 2 + 1].val;
64         lazy[i * 2 + 2] = lazy[i].val + lazy[i * 2 + 2].val;
65     }
66
67     lazy[i] = num_t();
68 }
69 };
70
71
72 int N, Q;
73 int a[maxN];
74
75 struct node {
76     ll val;
77     ll lzAdd;
78
79     ll lzSet;
80     node() {};
81
82 } tree[maxN << 2];
83
84 #define lc p << 1
85 #define rc (p << 1) + 1
86
87 inline void pushup(int p) {
88     tree[p].val = tree[lc].val + tree[rc].val;
89     return;
90 }
91
92 void pushdown(int p, int l, int mid, int r) {
93     // lazy: range set
94     if (tree[p].lzSet != 0) {
95         tree[lc].lzSet = tree[rc].lzSet = tree[p].lzSet;
96         tree[lc].val = (mid - l + 1) * tree[p].lzSet;
97         tree[rc].val = (r - mid) * tree[p].lzSet;
98         tree[lc].lzAdd = tree[rc].lzAdd = 0;
99         tree[p].lzSet = 0;
100    } else if (tree[p].lzAdd != 0) { // lazy: range add
101        if (tree[lc].lzSet == 0) tree[lc].lzAdd += tree[p].lzAdd;
102        else {
103            tree[lc].lzSet += tree[p].lzAdd;
104            tree[lc].lzAdd = 0;
105        }
106        if (tree[rc].lzSet == 0) tree[rc].lzAdd += tree[p].lzAdd;
107        else {
108            tree[rc].lzSet += tree[p].lzAdd;
109            tree[rc].lzAdd = 0;
110        }
111        tree[lc].val += (mid - l + 1) * tree[p].lzAdd;
112        tree[rc].val += (r - mid) * tree[p].lzAdd;
113        tree[p].lzAdd = 0;
114    }
115    return;
116 }
117
118 void build(int p, int l, int r) {
119     tree[p].lzAdd = tree[p].lzSet = 0;
120     if (l == r) {
121         tree[p].val = a[l];
122         return;
123     }
124 }

```

```

51     }
52     int mid = (l + r) >> 1;
53     build(lc, l, mid);
54     build(rc, mid + 1, r);
55     pushup(p);
56     return;
57 }

58 void add(int p, int l, int r, int a, int b, ll val) {
59     if (a > r || b < l) return;
60     if (a <= l && r <= b) {
61         tree[p].val += (r - l + 1) * val;
62         if (tree[p].lzSet == 0) tree[p].lzAdd += val;
63         else tree[p].lzSet += val;
64         return;
65     }
66     int mid = (l + r) >> 1;
67     pushdown(p, l, mid, r);
68     add(lc, l, mid, a, b, val);
69     add(rc, mid + 1, r, a, b, val);
70     pushup(p);
71     return;
72 }
73

74 void set(int p, int l, int r, int a, int b, ll val) {
75     if (a > r || b < l) return;
76     if (a <= l && r <= b) {
77         tree[p].val = (r - l + 1) * val;
78         tree[p].lzAdd = 0;
79         tree[p].lzSet = val;
80         return;
81     }
82     int mid = (l + r) >> 1;
83     pushdown(p, l, mid, r);
84     set(lc, l, mid, a, b, val);
85     set(rc, mid + 1, r, a, b, val);
86     pushup(p);
87     return;
88 }
89

90 ll query(int p, int l, int r, int a, int b) {
91     if (a > r || b < l) return 0;
92     if (a <= l && r <= b) return tree[p].val;
93 }
```

```

94     int mid = (l + r) >> 1;
95     pushdown(p, l, mid, r);
96     return query(lc, l, mid, a, b) + query(rc, mid + 1, r, a, b);
97 }

2.6 PersistentSegmentTree

1 struct Node {
2     ll val;
3     Node *_l, *_r;
4
5     Node(ll x) : val(x), _l(nullptr), _r(nullptr) {}
6     Node(Node *_l, Node *_r) {
7         _l = _l, _r = _r;
8         val = 0;
9         if (_l) val += _l->val;
10        if (_r) val += _r->val;
11    }
12    Node(Node *cp) : val(cp->val), _l(cp->l), _r(cp->r) {}
13};

14 int n, sz = 1;
15 ll a[200001];
16 Node *t[200001];
17
18 Node *build(int l = 1, int r = n) {
19     if (l == r) return new Node(a[l]);
20     int mid = (l + r) / 2;
21     return new Node(build(l, mid), build(mid + 1, r));
22 }
23
24 Node *update(Node *node, int pos, int val, int l = 1, int r = n) {
25     if (l == r) return new Node(val);
26     int mid = (l + r) / 2;
27     if (pos > mid)
28         return new Node(node->l, update(node->r, pos, val, mid + 1, r));
29     else return new Node(update(node->l, pos, val, l, mid), node->r);
30 }
31
32
33 ll query(Node *node, int a, int b, int l = 1, int r = n) {
34     if (l > b || r < a) return 0;
35     if (l >= a && r <= b) return node->val;
36     int mid = (l + r) / 2;
```

```

37     return query(node->l, a, b, l, mid) + query(node->r, a, b, mid + 1, r)
38 }
39
40 int main(){
41     ios_base::sync_with_stdio(false); cin.tie(NULL);
42     int q; cin >> n >> q;
43     for(int i=1;i<=n;i++){
44         cin >> a[i];
45     }
46     t[sz++]=build();
47     while(q--){
48         int ty; cin >> ty;
49         if(ty==1){
50             int k, pos, x; cin >> k >> pos >> x;
51             t[k]=update(t[k], pos, x);
52         }
53         else if(ty==2){
54             int k, l, r; cin >> k >> l >> r;
55             cout << query(t[k], l, r) << endl;
56         }
57         else{
58             int k; cin >> k;
59             t[sz++]=new Node(t[k]);
60         }
61     }
62 }
```

```

14     int mid = (l + r) / 2;
15     return new Node(build(l, mid), build(mid + 1, r));
16 }
17
18 Node *update(Node *node, int val, int pos, int l = 0, int r = n - 1) {
19     if (l == r) return new Node(val);
20     int mid = (l + r) / 2;
21     if (pos > mid)
22         return new Node(node->l, update(node->r, val, pos, mid + 1, r));
23     else return new Node(update(node->l, val, pos, l, mid), node->r);
24 }
25
26 int query(Node *node, int pos, int l = 0, int r = n - 1) {
27     if (l == r) return node->val;
28     int mid = (l + r) / 2;
29     if (pos > mid) return query(node->r, pos, mid + 1, r);
30     return query(node->l, pos, l, mid);
31 }
32
33 int get_item(int index, int time) {
34     // Gets the array item at a given index and time
35     return query(roots[time], index);
36 }
37
38 void update_item(int index, int value, int prev_time, int curr_time) {
39     // Updates the array item at a given index and time
40     roots[curr_time] = update(roots[prev_time], index, value);
41 }
42
43 void init_arr(int nn, int *init) {
44     // Initializes the persistent array, given an input array
45     n = nn;
46     for (int i = 0; i < n; i++) a[i] = init[i];
47     roots[0] = build();
48 }
```

2.7 Persistent Array

```

1 struct Node {
2     int val;
3     Node *l, *r;
4
5     Node(ll x) : val(x), l(nullptr), r(nullptr) {}
6     Node(Node *ll, Node *rr) : val(0), l(ll), r(rr) {}
7 };
8
9 int n, a[100001];      // The initial array and its size
10 Node *roots[100001]; // The persistent array's roots
11
12 Node *build(int l = 0, int r = n - 1) {
13     if (l == r) return new Node(a[l]);
```

2.8 Fenwick

```

1 template <typename T>
2 struct Fenwick {
3     int n;
4     std::vector<T> a;
5 }
```

```

6     Fenwick(int n_ = 0) {
7         init(n_);
8     }
9
10    void init(int n_) {
11        n = n_;
12        a.assign(n, T{});
13    }
14
15    void add(int x, const T &v) {
16        for (int i = x + 1; i <= n; i += i & -i) {
17            a[i - 1] = a[i - 1] + v;
18        }
19    }
20
21    T sum(int x) {
22        T ans{};
23        for (int i = x; i > 0; i -= i & -i) {
24            ans = ans + a[i - 1];
25        }
26        return ans;
27    }
28
29    T rangeSum(int l, int r) {
30        return sum(r) - sum(l);
31    }
32
33    int select(const T &k) {
34        int x = 0;
35        T cur{};
36        for (int i = 1 << std::__lg(n); i; i /= 2) {
37            if (x + i <= n && cur + a[x + i - 1] <= k) {
38                x += i;
39                cur = cur + a[x - 1];
40            }
41        }
42        return x;
43    }
44};

```

2.9 Fenwick 2D

```
1 | struct Fenwick2D{
```

```

2     vector<vector<ll>> b;
3     int n;
4
5     Fenwick2D(int _n) : b(_n+5, vector<ll>(_n+5, 0)), n(_n) {}
6
7     void update(int x, int y, int val){
8         for(; x<=n; x+=(x&-x)){
9             for(int j=y; j<=n; j+=(j&-j)){
10                 b[x][j]+=val;
11             }
12         }
13     }
14
15     ll get(int x, int y){
16         ll ans=0;
17         for(; x; x-=x&-x){
18             for(int j=y; j ;j-=j&-j){
19                 ans+=b[x][j];
20             }
21         }
22         return ans;
23     }
24
25     ll get1(int x1, int y1, int x2, int y2){
26         return get(x2, y2)-get(x1-1, y2)-get(x2, y1-1)+ get(x1-1, y1-1);
27     }
28
29 };

```

2.10 SegmentTree 2D

```

1 | void build_y(int vx, int lx, int rx, int vy, int ly, int ry) {
2 |     if (ly == ry) {
3 |         if (lx == rx)
4 |             t[vx][vy] = a[lx][ly];
5 |         else
6 |             t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
7 |     } else {
8 |         int my = (ly + ry) / 2;
9 |         build_y(vx, lx, rx, vy*2, ly, my);
10 |        build_y(vx, lx, rx, vy*2+1, my+1, ry);
11 |        t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
12 |

```

```

13 }
14
15 void build_x(int vx, int lx, int rx) {
16     if (lx != rx) {
17         int mx = (lx + rx) / 2;
18         build_x(vx*2, lx, mx);
19         build_x(vx*2+1, mx+1, rx);
20     }
21     build_y(vx, lx, rx, 1, 0, m-1);
22 }
23
24 int sum_y(int vx, int vy, int tly, int try_, int ly, int ry) {
25     if (ly > ry)
26         return 0;
27     if (ly == tly && try_ == ry)
28         return t[vx][vy];
29     int tmy = (tly + try_) / 2;
30     return sum_y(vx, vy*2, tly, tmy, ly, min(ry, tmy))
31         + sum_y(vx, vy*2+1, tmy+1, try_, max(ly, tmy+1), ry);
32 }
33
34 int sum_x(int vx, int tlx, int trx, int lx, int rx, int ly, int ry) {
35     if (lx > rx)
36         return 0;
37     if (lx == tlx && trx == rx)
38         return sum_y(vx, 1, 0, m-1, ly, ry);
39     int tmx = (tlx + trx) / 2;
40     return sum_x(vx*2, tlx, tmx, lx, min(rx, tmx), ly, ry)
41         + sum_x(vx*2+1, tmx+1, trx, max(lx, tmx+1), rx, ly, ry);
42 }
43
44
45 void update_y(int vx, int lx, int rx, int vy, int ly, int ry, int x, int
46     y, int new_val) {
47     if (ly == ry) {
48         if (lx == rx)
49             t[vx][vy] = new_val;
50         else
51             t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
52     } else {
53         int my = (ly + ry) / 2;
54         if (y <= my)
55             update_y(vx, lx, rx, vy*2, ly, my, x, y, new_val);
56     }
57 }
58 }
```

```

55     else
56         update_y(vx, lx, rx, vy*2+1, my+1, ry, x, y, new_val);
57     t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
58 }
59 }
60
61 void update_x(int vx, int lx, int rx, int x, int y, int new_val) {
62     if (lx != rx) {
63         int mx = (lx + rx) / 2;
64         if (x <= mx)
65             update_x(vx*2, lx, mx, x, y, new_val);
66         else
67             update_x(vx*2+1, mx+1, rx, x, y, new_val);
68     }
69     update_y(vx, lx, rx, 1, 0, m-1, x, y, new_val);
70 }
```

2.11 DynamicSegTree

```

1 struct Vertex {
2     int left, right;
3     int sum = 0;
4     Vertex *left_child = nullptr, *right_child = nullptr;
5
6     Vertex(int lb, int rb) {
7         left = lb;
8         right = rb;
9     }
10
11    void extend() {
12        if (!left_child && left + 1 < right) {
13            int t = (left + right) / 2;
14            left_child = new Vertex(left, t);
15            right_child = new Vertex(t, right);
16        }
17    }
18
19    void add(int k, int x) {
20        extend();
21        sum += x;
22        if (left_child) {
23            if (k < left_child->right)
24                left_child->add(k, x);
25        }
26    }
27
28    int query(int l, int r) {
29        if (l <= left && r >= right)
30            return sum;
31        if (r <= left || l >= right)
32            return 0;
33        return left_child->query(l, r) + right_child->query(l, r);
34    }
35
36    void update(int k, int v) {
37        if (left == right) {
38            sum = v;
39            return;
40        }
41        if (k < left)
42            left_child->update(k, v);
43        else
44            right_child->update(k, v);
45        sum = left_child->sum + right_child->sum;
46    }
47 }
```

```

25     else
26         right_child->add(k, x);
27     }
28 }
29
30 int get_sum(int lq, int rq) {
31     if (lq <= left && right <= rq)
32         return sum;
33     if (max(left, lq) >= min(right, rq))
34         return 0;
35     extend();
36     return left_child->get_sum(lq, rq) + right_child->get_sum(lq, rq)
37         );
38 };

```

2.12 OrderedSet

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4
5 // to allow repetitions
6 #define ordered_set tree<int, null_type,less_equal<int>, rb_tree_tag,
7     tree_order_statistics_node_update>
8
9 // to not allow repetitions
10 #define ordered_set tree<int, null_type,less<int>, rb_tree_tag,
11     tree_order_statistics_node_update>
12
13 //order_of_key(x): number of items are strictly smaller than x
14
15 //find_by_order(k) iterator to the kth element

```

2.13 Disjoint Set Union

```

1 struct DSU {
2     vector<int> e;
3     vector<pair<int, int>> st;
4
5     DSU(int N) : e(N, -1) {}
6
7     int get(int x) { return e[x] < 0 ? x : e[x] = get(e[x]); }

```

```

9     bool connected(int a, int b) { return get(a) == get(b); }
10
11     int size(int x) { return -e[get(x)]; }
12
13     bool unite(int x, int y) {
14         x = get(x), y = get(y);
15         if (x == y) { return false; }
16         if (e[x] > e[y]) { swap(x, y); }
17         st.push_back({x, e[x]}); e[x] += e[y];
18         st.push_back({y, e[y]}); e[y] = x;
19         e[x] += e[y];
20         e[y] = x;
21         return true;
22     }
23
24     //skip if no rollback
25     int time() {return (int)st.size(); }
26
27     void rollback(int t) {
28         for (int i = time(); i --> t;)
29             e[st[i].first] = st[i].second;
30         st.resize(t);
31     }
32 }
33
34
35 //dsu for checking parity of path length (can be used for checking
36 // bipartiteness)
37 void make_set(int v) {
38     parent[v] = make_pair(v, 0);
39     rank[v] = 0;
40     bipartite[v] = true;
41 }
42
43 pair<int, int> find_set(int v) {
44     if (v != parent[v].first) {
45         int parity = parent[v].second;
46         parent[v] = find_set(parent[v].first);
47         parent[v].second ^= parity;
48     }
49     return parent[v];
50 }

```

```

51 void add_edge(int a, int b) {
52     pair<int, int> pa = find_set(a);
53     a = pa.first;
54     int x = pa.second;
55
56     pair<int, int> pb = find_set(b);
57     b = pb.first;
58     int y = pb.second;
59
60     if (a == b) {
61         if (x == y)
62             bipartite[a] = false;
63     } else {
64         if (rank[a] < rank[b])
65             swap(a, b);
66         parent[b] = make_pair(a, x^y^1);
67         bipartite[a] &= bipartite[b];
68         if (rank[a] == rank[b])
69             ++rank[a];
70     }
71 }
72
73 bool is_bipartite(int v) {
74     return bipartite[find_set(v).first];
75 }
```

2.14 BitTrie

```

1 const int K = 2;
2 struct Vertex {
3     int next[K];
4
5     Vertex() {
6         fill(begin(next), end(next), -1);
7     }
8 };
9
10 //insert
11 for(int j=30;j>=0;j--) {
12     int c = 1&(a[i]>>j);
13     if (trie[v].next[c] == -1) {
14         trie[v].next[c] = trie.size();
```

```

16         trie.emplace_back();
17         d.pb(-1);
18     }
19     v = trie[v].next[c];
20 }
```

2.15 Palindromic Tree

```

1 const int N = 3e5 + 9;
2
3 /*
4 -> cnt contains the number of palindromic suffixes of the node
5 */
6 struct PalindromicTree {
7     struct node {
8         int nxt[26], len, st, en, link, cnt, oc;
9     };
10    string s;
11    vector<node> t;
12    int sz, last;
13    PalindromicTree() {}
14    PalindromicTree(string _s) {
15        s = _s;
16        int n = s.size();
17        t.clear();
18        t.resize(n + 9);
19        sz = 2, last = 2;
20        t[1].len = -1, t[1].link = 1;
21        t[2].len = 0, t[2].link = 1;
22    }
23    int extend(int pos) { // returns 1 if it creates a new palindrome
24        int cur = last, curlen = 0;
25        int ch = s[pos] - 'a';
26        while (1) {
27            curlen = t[cur].len;
28            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) break;
29            cur = t[cur].link;
30        }
31        if (t[cur].nxt[ch]) {
32            last = t[cur].nxt[ch];
33            t[last].oc++;
34            return 0;
35        }
36    }
37 }
```

```

36     sz++;
37     last = sz;
38     t[sz].oc = 1;
39     t[sz].len = t[cur].len + 2;
40     t[cur].nxt[ch] = sz;
41     t[sz].en = pos;
42     t[sz].st = pos - t[sz].len + 1;
43     if (t[sz].len == 1) {
44         t[sz].link = 2;
45         t[sz].cnt = 1;
46         return 1;
47     }
48     while (1) {
49         cur = t[cur].link;
50         curlen = t[cur].len;
51         if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
52             t[sz].link = t[cur].nxt[ch];
53             break;
54         }
55     }
56     t[sz].cnt = 1 + t[t[sz].link].cnt;
57     return 1;
58 }
59 void calc_occurrences() {
60     for (int i = sz; i >= 3; i--) t[t[i].link].oc += t[i].oc;
61 }
62 } t;
63
64 int32_t main() {
65     ios_base::sync_with_stdio(0);
66     cin.tie(0);
67     string s;
68     cin >> s;
69     PalindromicTree t(s);
70     for (int i = 0; i < s.size(); i++) t.extend(i);
71     t.calc_occurrences();
72     long long ans = 0; // number of palindromes
73     for (int i = 3; i <= t.sz; i++) ans += t.t[i].oc;
74     cout << ans << '\n';
75     return 0;
76 }
```

2.16 LineContainer

```

1 //Queries for maximum point x. To change this modify first comparator.
2 struct Line {
3     mutable ll k, m, p;
4     bool operator<(const Line& o) const { return k < o.k; }
5     bool operator<(ll x) const { return p < x; }
6 };
7
8 struct LineContainer : multiset<Line, less<>> {
9     // (for doubles, use inf = 1/.0, div(a,b) = a/b)
10    static const ll inf = LLONG_MAX;
11    ll div(ll a, ll b) { // floored division
12        return a / b - ((a ^ b) < 0 && a % b); }
13    bool isect(iterator x, iterator y) {
14        if (y == end()) return x->p = inf, 0;
15        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
16        else x->p = div(y->m - x->m, x->k - y->k);
17        return x->p >= y->p;
18    }
19    void add(ll k, ll m) {
20        auto z = insert({k, m, 0}), y = z++, x = y;
21        while (isect(y, z)) z = erase(z);
22        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
23        while ((y = x) != begin() && (--x)->p >= y->p)
24            isect(x, erase(y));
25    }
26    ll query(ll x) {
27        assert(!empty());
28        auto l = *lower_bound(x);
29        return l.k * x + l.m;
30    }
31};


```

2.17 SqrtDecomposition

```

1
2 int n, numBlocks;
3 string s;
4
5 struct Block{
6     int l, r;
7     int sz(){
8         return r-l;
9     }

```

```

10  };
11
12 Block blocks[2*MAXI];
13 Block newBlocks[2*MAXI];
14
15 void rebuildDecomp(){
16     string newS=s;
17     int k=0;
18     for(int i=0;i<numBlocks;i++){
19         for(int j=blocks[i].l;j<blocks[i].r;j++){
20             newS[k++]=s[j];
21         }
22     }
23     numBlocks=1;
24     blocks[0]={0, n};
25     s=newS;
26 }
27
28 void cut(int a, int b){
29     int pos=0, curBlock=0;
30     for(int i=0;i<numBlocks;i++){
31         Block B=blocks[i];
32         bool containsA = pos < a && pos + B.sz() > a;
33         bool containsB = pos < b && pos + B.sz() > b;
34         int cutA = B.l + a - pos;
35         int cutB = B.l + b - pos;
36         if(containsA && containsB){
37             newBlocks[curBlock++]={B.l, cutA};
38             newBlocks[curBlock++]={cutA, cutB};
39             newBlocks[curBlock++]={cutB, B.r};
40         }
41         else if(containsA){
42             newBlocks[curBlock++]={B.l, cutA};
43             newBlocks[curBlock++]={cutA, B.r};
44         }
45         else if(containsB){
46             newBlocks[curBlock++]={B.l, cutB};
47             newBlocks[curBlock++]={cutB, B.r};
48         }
49         else{
50             newBlocks[curBlock++]=B;
51         }
52     pos += B.sz();
53     }
54     pos=0;
55     numBlocks=0;
56     for(int i=0;i<curBlock;i++){
57         if(pos<a || pos>=b){
58             blocks[numBlocks++]=newBlocks[i];
59         }
60         pos+=newBlocks[i].sz();
61     }
62     pos=0;
63     for(int i=0;i<curBlock;i++){
64         if(pos>=a && pos<=b){
65             blocks[numBlocks++]=newBlocks[i];
66         }
67         pos+=newBlocks[i].sz();
68     }
69 }
70
71 // while doing operations
72 if(numBlocks>MAXI){
73     rebuildDecomp();
74 }
75
76 // rebuild before final ans
77 rebuildDecomp();
78 cout << ans << endl;

```

2.18 Treap

```

1  typedef struct item * pitem;
2  struct item {
3      int prior, value, cnt;
4      bool rev;
5      pitem l, r;
6  };
7
8  int cnt (pitem it) {
9      return it ? it->cnt : 0;
10 }
11
12 void upd_cnt (pitem it) {
13     if (it)
14         it->cnt = cnt(it->l) + cnt(it->r) + 1;

```

```

15 }
16
17 void push (pitem it) {
18     if (it && it->rev) {
19         it->rev = false;
20         swap (it->l, it->r);
21         if (it->l) it->l->rev ^= true;
22         if (it->r) it->r->rev ^= true;
23     }
24 }
25
26 void merge (pitem & t, pitem l, pitem r) {
27     push (l);
28     push (r);
29     if (!l || !r)
30         t = l ? l : r;
31     else if (l->prior > r->prior)
32         merge (l->r, l->r, r), t = l;
33     else
34         merge (r->l, l, r->l), t = r;
35     upd_cnt (t);
36 }
37
38 void split (pitem t, pitem & l, pitem & r, int key, int add = 0) {
39     if (!t)
40         return void( l = r = 0 );
41     push (t);
42     int cur_key = add + cnt(t->l);
43     if (key <= cur_key)
44         split (t->l, l, t->l, key, add), r = t;
45     else
46         split (t->r, t->r, r, key, add + 1 + cnt(t->l)), l = t;
47     upd_cnt (t);
48 }
49
50 void reverse (pitem t, int l, int r) {
51     pitem t1, t2, t3;
52     split (t, t1, t2, l);
53     split (t2, t2, t3, r-l+1);
54     t2->rev ^= true;
55     merge (t, t1, t2);
56     merge (t, t, t3);
57 }

```

```

58
59 void output (pitem t) {
60     if (!t) return;
61     push (t);
62     output (t->l);
63     printf ("%d\u2022", t->value);
64     output (t->r);
65 }

```

2.19 Dynammic Connectivity

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 typedef long long ll;
5
6 struct DSU {
7     vector<int> e;
8     vector<pair<int, int>> st;
9     int cnt;
10
11 DSU(){}
12
13 DSU(int N) : e(N, -1), cnt(N) {}
14
15 int get(int x) { return e[x] < 0 ? x : get(e[x]); }
16
17 bool connected(int a, int b) { return get(a) == get(b); }
18
19 int size(int x) { return -e[get(x)]; }
20
21 bool unite(int x, int y) {
22     x = get(x), y = get(y);
23     if (x == y) { return false; }
24     if (e[x] > e[y]) { swap(x, y); }
25     st.push_back({x, e[x]});
26     st.push_back({y, e[y]});
27     e[x] += e[y];
28     e[y] = x;
29     cnt--;
30     return true;
31 }
32

```

```

33 void rollback(){
34     auto [x, y]=st.back();
35     st.pop_back();
36     e[x] = y;
37     auto [a, b]=st.back();
38     st.pop_back();
39     e[a]=b;
40     cnt++;
41 }
42 };
43
44 struct query {
45     int v, u;
46     bool united;
47     query(int _v, int _u) : v(_v), u(_u) {}
48 };
49
50 struct QueryTree {
51     vector<vector<query>> t;
52     DSU dsu;
53     int T;
54
55     QueryTree(){}
56
57     QueryTree(int _T, int n) : T(_T) {
58         dsu = DSU(n);
59         t.resize(4 * T + 4);
60     }
61
62     void add(int v, int l, int r, int ul, int ur, query& q) {
63         if (ul > ur)
64             return;
65         if (l == ul && r == ur) {
66             t[v].push_back(q);
67             return;
68         }
69         int mid = (l + r) / 2;
70         add(2 * v, l, mid, ul, min(ur, mid), q);
71         add(2 * v + 1, mid + 1, r, max(ul, mid + 1), ur, q);
72     }
73
74     void add_query(query q, int l, int r) {
75         add(1, 0, T - 1, l, r, q);
76     }
77
78     void dfs(int v, int l, int r, vector<int>& ans) {
79         for (query& q : t[v]) {
80             q.united = dsu.unite(q.v, q.u);
81         }
82         if (l == r)
83             ans[l] = dsu.cnt;
84         else {
85             int mid = (l + r) / 2;
86             dfs(2 * v, l, mid, ans);
87             dfs(2 * v + 1, mid + 1, r, ans);
88         }
89         for (query q : t[v]) {
90             if (q.united)
91                 dsu.rollback();
92         }
93     }
94 };
95
96
97 int main(){
98     ios_base::sync_with_stdio(false); cin.tie(NULL);
99     //freopen("connect.in", "r", stdin);
100    //freopen("connect.out", "w", stdout);
101    int n, k; cin >> n >> k;
102    if(k==0) return 0;
103    QueryTree st=QueryTree(k, n);
104    map<pair<int, int>, int> mp;
105    vector<int> ans(k), q;
106    for(int i=0;i<k;i++){
107        char c; cin >> c;
108        if(c=='?'){
109            q.push_back(i);
110            continue;
111        }
112        int u, v; cin >> u >> v;
113        u--; v--;
114        if(u>v) swap(u, v);
115        if(c=='+'){
116            mp[{u, v}]=i;
117        }
118        else{

```

```

119     st.add_query(query(u, v), mp[{u, v}], i);
120     mp[{u, v}]=-1;
121 }
122 }
123 for(auto [x, y]:mp){
124     if(y!=-1){
125         st.add_query(query(x.first, x.second), y, k-1);
126     }
127 }
128 st.dfs(1, 0, k-1, ans);
129 for(int x:q){
130     cout << ans[x] << endl;
131 }
132 }
```

2.20 Bitset

```

1 bitset<3001> b[3001];
2
3 //set() Set the bit value at the given index to 1.
4 //count() Count the number of set bits.
5 //any() Checks if any bit is set
6 //all() Check if all bit is set.
```

3 Math

3.1 Binary Exponentiation And Modular Division

```

1 ll binpow(ll a, ll b){
2     ll r=1;
3     while(b){
4         if(b%2)
5             r=(r*a)%MOD;
6         a=(a*a)%MOD;
7         b/=2;
8     }
9     return r;
10 }
11
12 ll divide(ll a, ll b){
13     return ((a%MOD)*binpow(b, MOD-2))%MOD;
14 }
15 void inverses(long long p) {
```

```

16     inv[MAXN] = exp(fac[MAXN], p - 2, p);
17     for (int i = MAXN; i >= 1; i--) { inv[i - 1] = inv[i] * i % p; }
18 }
```

3.2 Euler's Totient

```

1 //counts coprimes to each number from 1 to n
2 vector<int> phi1(int n) {
3     vector<int> phi(n + 1);
4     for (int i = 0; i <= n; i++)
5         phi[i] = i;
6
7     for (int i = 2; i <= n; i++) {
8         if (phi[i] == i) {
9             for (int j = i; j <= n; j += i)
10                 phi[j] -= phi[j] / i;
11         }
12     }
13     return phi;
14 }
```

3.3 Miller Rabin

```

1 using u64 = uint64_t;
2 using u128 = __uint128_t;
3
4 u64 binpower(u64 base, u64 e, u64 mod) {
5     u64 result = 1;
6     base %= mod;
7     while (e) {
8         if (e & 1)
9             result = (u128)result * base % mod;
10        base = (u128)base * base % mod;
11        e >>= 1;
12    }
13    return result;
14 }
15
16 bool check_composite(u64 n, u64 a, u64 d, int s) {
17     u64 x = binpower(a, d, n);
18     if (x == 1 || x == n - 1)
19         return false;
20     for (int r = 1; r < s; r++) {
21         x = (u128)x * x % n;
```

```

22     if (x == n - 1)
23         return false;
24     }
25     return true;
26 };
27
28
29 bool MillerRabin(ll n) {
30     if (n < 2)
31         return false;
32
33     int r = 0;
34     ll d = n - 1;
35     while ((d & 1) == 0) {
36         d >>= 1;
37         r++;
38     }
39
40     for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
41         if (n == a)
42             return true;
43         if (check_composite(n, a, d, r))
44             return false;
45     }
46     return true;
47 }
```

3.4 Diophantine

If one solution is (x_0, y_0) all solutions can be obtained by $x = x_0 + k * \frac{b}{\gcd(a,b)}$ and $y = y_0 - k * \frac{a}{\gcd(a,b)}$.

```

1 int gcd(int a, int b, int& x, int& y) {
2     if (b == 0) {
3         x = 1;
4         y = 0;
5         return a;
6     }
7     int x1, y1;
8     int d = gcd(b, a % b, x1, y1);
9     x = y1;
10    y = x1 - y1 * (a / b);
11    return d;
12 }
```

```

13
14 bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g) {
15     g = gcd(abs(a), abs(b), x0, y0);
16     if (c % g) {
17         return false;
18     }
19
20     x0 *= c / g;
21     y0 *= c / g;
22     if (a < 0) x0 = -x0;
23     if (b < 0) y0 = -y0;
24     return true;
25 }
26
27
28
29 //n variables
30 vector<ll> find_any_solution(vector<ll> a, ll c) {
31     int n = a.size();
32     vector<ll> x;
33     bool all_zero = true;
34     for (int i = 0; i < n; i++) {
35         all_zero &= a[i] == 0;
36     }
37     if (all_zero) {
38         if (c) return {};
39         x.assign(n, 0);
40         return x;
41     }
42     ll g = 0;
43     for (int i = 0; i < n; i++) {
44         g = __gcd(g, a[i]);
45     }
46     if (c % g != 0) return {};
47     if (n == 1) {
48         return {c / a[0]};
49     }
50     vector<ll> suf_gcd(n);
51     suf_gcd[n - 1] = a[n - 1];
52     for (int i = n - 2; i >= 0; i--) {
53         suf_gcd[i] = __gcd(suf_gcd[i + 1], a[i]);
54     }
55     ll cur = c;
```

```

56     for (int i = 0; i + 1 < n; i++) {
57         ll x0, y0, g;
58         // solve for a[i] * x + suf_gcd[i + 1] * (y / suf_gcd[i + 1]) = cur
59         bool ok = find_any_solution(a[i], suf_gcd[i + 1], cur, x0, y0, g);
60         assert(ok);
61     }
62     // trying to minimize x0 in case x0 becomes big
63     // it is needed for this problem, not needed in general
64     ll shift = abs(suf_gcd[i + 1] / g);
65     x0 = (x0 % shift + shift) % shift;
66 }
67 x.push_back(x0);

68 // now solve for the next suffix
69 cur -= a[i] * x0;
70 }
71 x.push_back(a[n - 1] == 0 ? 0 : cur / a[n - 1]);
72 return x;
73 }
74 }
```

3.5 Discrete Log

Finds discrete logarithm in $O(\sqrt{m})$.

```

1 // Returns minimum x for which a ^ x % m = b % m, a and m are coprime.
2 int solve(int a, int b, int m) {
3     a %= m, b %= m;
4     int n = sqrt(m) + 1;
5
6     int an = 1;
7     for (int i = 0; i < n; ++i)
8         an = (an * 1ll * a) % m;
9
10    unordered_map<int, int> vals;
11    for (int q = 0, cur = b; q <= n; ++q) {
12        vals[cur] = q;
13        cur = (cur * 1ll * a) % m;
14    }
15
16    for (int p = 1, cur = 1; p <= n; ++p) {
17        cur = (cur * 1ll * an) % m;
18        if (vals.count(cur)) {
19            int ans = n * p - vals[cur];
20            return ans;
21        }
22    }
23    return -1;
24 }

25 // Returns minimum x for which a ^ x % m = b % m.
26 int solve(int a, int b, int m) {
27     a %= m, b %= m;
28     int k = 1, add = 0, g;
29     while ((g = gcd(a, m)) > 1) {
30         if (b == k)
31             return add;
32         if (b % g)
33             return -1;
34         b /= g, m /= g, ++add;
35         k = (k * 1ll * a / g) % m;
36     }
37
38     int n = sqrt(m) + 1;
39     int an = 1;
40     for (int i = 0; i < n; ++i)
41         an = (an * 1ll * a) % m;
42
43     unordered_map<int, int> vals;
44     for (int q = 0, cur = b; q <= n; ++q) {
45         vals[cur] = q;
46         cur = (cur * 1ll * a) % m;
47     }
48
49     for (int p = 1, cur = k; p <= n; ++p) {
50         cur = (cur * 1ll * an) % m;
51         if (vals.count(cur)) {
52             int ans = n * p - vals[cur] + add;
53             return ans;
54         }
55     }
56
57     return -1;
58 }
```

4 Polynomials

4.1 Fast Fourier Transform (FFT)

Fast Fourier Transform in $O(n \log n)$.

```

1 using cd = complex<double>;
2 const double PI = acos(-1);
3 //declare size of vectors used like this
4 const int MAXN=2<<19;
5
6 void fft(vector<cd> & a, bool invert) {
7     int n = (int)a.size();
8
9     for (int i = 1, j = 0; i < n; i++) {
10        int bit = n >> 1;
11        for (; j & bit; bit >>= 1)
12            j ^= bit;
13        j ^= bit;
14
15        if (i < j)
16            swap(a[i], a[j]);
17    }
18
19    for (int len = 2; len <= n; len <= 1) {
20        double ang = 2 * PI / len * (invert ? -1 : 1);
21        cd wlen(cos(ang), sin(ang));
22        for (int i = 0; i < n; i += len) {
23            cd w(1);
24            for (int j = 0; j < len / 2; j++) {
25                cd u = a[i+j], v = a[i+j+len/2] * w;
26                a[i+j] = u + v;
27                a[i+j+len/2] = u - v;
28                w *= wlen;
29            }
30        }
31    }
32
33    if (invert) {
34        for (cd & x : a)
35            x /= n;
36    }
37}

```

```

39 vector<int> multiply(vector<int> const& a, vector<int> const& b) {
40     vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
41     int n = 1;
42     while (n < a.size() + b.size())
43         n <= 1;
44     fa.resize(n);
45     fb.resize(n);
46
47     fft(fa, false);
48     fft(fb, false);
49     for (int i = 0; i < n; i++)
50         fa[i] *= fb[i];
51     fft(fa, true);
52
53     vector<int> result(n);
54     for (int i = 0; i < n; i++)
55         result[i] = round(fa[i].real());
56     return result;
57 }
58
59 //normalizing for when mult is between 2 big numbers and not polynomials
60 int carry = 0;
61 for (int i = 0; i < n; i++){
62     result[i] += carry;
63     carry = result[i] / 10;
64     result[i] %= 10;
65 }

```

4.2 Number Theoretic Transform (NTT)

```

1 const int mod = 7340033;
2 const int root = 5;
3 const int root_1 = 4404020;
4 const int root_pw = 1 << 20;
5
6 //Coefficients modulo prime p
7
8 //declare size of vectors used like this
9 const int MAXN=2<<19;
10
11 void fft(vector<int> & a, bool invert) {
12     int n = a.size();

```

```

13
14     for (int i = 1, j = 0; i < n; i++) {
15         int bit = n >> 1;
16         for (; j & bit; bit >>= 1)
17             j ^= bit;
18         j ^= bit;
19
20         if (i < j)
21             swap(a[i], a[j]);
22     }
23
24     for (int len = 2; len <= n; len <=> 1) {
25         int wlen = invert ? root_1 : root;
26         for (int i = len; i < root_pw; i <=> 1)
27             wlen = (int)(1LL * wlen * wlen % mod);
28
29         for (int i = 0; i < n; i += len) {
30             int w = 1;
31             for (int j = 0; j < len / 2; j++) {
32                 int u = a[i+j], v = (int)(1LL * a[i+j+len/2] * w % mod);
33                 a[i+j] = u + v < mod ? u + v : u + v - mod;
34                 a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
35                 w = (int)(1LL * w * wlen % mod);
36             }
37         }
38
39         if (invert) {
40             int n_1 = inverse(n, mod);
41             for (int & x : a)
42                 x = (int)(1LL * x * n_1 % mod);
43         }
44     }
45 }
```

4.3 Berlekamp-Massey

```

1 template<typename T>
2 vector<T> berlekampMassey(const vector<T> &s) {
3     vector<T> c;      // the linear recurrence sequence we are building
4     vector<T> oldC; // the best previous version of c to use (the one
5                   // with the rightmost left endpoint)
6     int f = -1;        // the index at which the best previous version of c
                        failed on
```

```

6     for (int i=0; i<(int)s.size(); i++) {
7         // evaluate c(i)
8         // delta = s_i - \sum_{j=1}^n c_j s_{i-j}
9         // if delta == 0, c(i) is correct
10        T delta = s[i];
11        for (int j=1; j<=(int)c.size(); j++)
12            delta -= c[j-1] * s[i-j]; // c_j is one-indexed, so we
13                           // actually need index j - 1 in the code
14        if (delta == 0)
15            continue; // c(i) is correct, keep going
16        // now at this point, delta != 0, so we need to adjust it
17        if (f == -1) {
18            // this is the first time we're updating c
19            // s_i was the first non-zero element we encountered
20            // we make c of length i + 1 so that s_i is part of the base
21            // case
22            c.resize(i + 1);
23            mt19937 rng(chrono::steady_clock::now().time_since_epoch().
24                         count());
25            for (T &x : c)
26                x = rng(); // just to prove that the initial values don
27                           // t matter in the first step, I will set to random
28                           // values
29            f = i;
30        } else {
31            // we need to use a previous version of c to improve on this
32            // one
33            // apply the 5 steps to build d
34            // 1. set d equal to our chosen sequence
35            vector<T> d = oldC;
36            // 2. multiply the sequence by -1
37            for (T &x : d)
38                x = -x;
39            // 3. insert a 1 on the left
40            d.insert(d.begin(), 1);
41            // 4. multiply the sequence by delta / d(f + 1)
42            T df1 = 0; // d(f + 1)
43            for (int j=1; j<=(int)d.size(); j++)
44                df1 += d[j-1] * s[f+1-j];
45            assert(df1 != 0);
46            T coef = delta / df1; // storing this in outer variable so
47                           // it's O(n^2) instead of O(n^2 log MOD)
48            for (T &x : d)
```

```

42     x *= coef;
43     // 5. insert i - f - 1 zeros on the left
44     vector<T> zeros(i - f - 1);
45     zeros.insert(zeros.end(), d.begin(), d.end());
46     d = zeros;
47     // now we have our new recurrence: c + d
48     vector<T> temp = c; // save the last version of c because it
        might have a better left endpoint
49     c.resize(max(c.size(), d.size()));
50     for (int j=0; j<(int)d.size(); j++)
51         c[j] += d[j];
52     // finally, let's consider updating oldC
53     if (i - (int) temp.size() > f - (int) oldC.size()) {
54         // better left endpoint, let's update!
55         oldC = temp;
56         f = i;
57     }
58 }
59 return c;
}

```

5 Linear Algebra

5.1 Simplex

```

1 /*
2 Parametric Self-Dual Simplex method
3 Solve a canonical LP:
4   min or max. c x
5   s.t. A x <= b
6   x >= 0
7 */
8 #include <bits/stdc++.h>
9 using namespace std;
10 const double eps = 1e-9, oo = numeric_limits<double>::infinity();
11
12 typedef vector<double> vec;
13 typedef vector<vec> mat;
14
15 pair<vec, double> simplexMethodPD(const mat &A, const vec &b, const vec
    &c, bool mini = true){
16     int n = c.size(), m = b.size();

```

```

17     mat T(m + 1, vec(n + m + 1));
18     vector<int> base(n + m), row(m);
19
20     for(int j = 0; j < m; ++j){
21         for(int i = 0; i < n; ++i)
22             T[j][i] = A[j][i];
23         row[j] = n + j;
24         T[j][n + j] = 1;
25         base[n + j] = 1;
26         T[j][n + m] = b[j];
27     }
28
29     for(int i = 0; i < n; ++i)
30         T[m][i] = c[i] * (mini ? 1 : -1);
31
32     while(true){
33         int p = 0, q = 0;
34         for(int i = 0; i < n + m; ++i)
35             if(T[m][i] <= T[m][p])
36                 p = i;
37
38         for(int j = 0; j < m; ++j)
39             if(T[j][n + m] <= T[q][n + m])
40                 q = j;
41
42         double t = min(T[m][p], T[q][n + m]);
43
44         if(t >= -eps){
45             vec x(n);
46             for(int i = 0; i < m; ++i)
47                 if(row[i] < n) x[row[i]] = T[i][n + m];
48             return {x, T[m][n + m] * (mini ? -1 : 1)}; // optimal
49         }
50
51         if(t < T[q][n + m]){
52             // tight on c -> primal update
53             for(int j = 0; j < m; ++j)
54                 if(T[j][p] >= eps)
55                     if(T[j][p] * (T[q][n + m] - t) >= T[q][p] * (T[j][n + m] - t))
56                         q = j;
57
58             if(T[q][p] <= eps)
59                 return {vec(n), oo * (mini ? 1 : -1)}; // primal infeasible

```

```

60 }else{
61     // tight on b -> dual update
62     for(int i = 0; i < n + m + 1; ++i)
63         T[q][i] = -T[q][i];
64
65     for(int i = 0; i < n + m; ++i)
66         if(T[q][i] >= eps)
67             if(T[q][i] * (T[m][p] - t) >= T[q][p] * (T[m][i] - t))
68                 p = i;
69
70     if(T[q][p] <= eps)
71         return {vec(n), oo * (mini ? -1 : 1)}; // dual infeasible
72 }
73
74 for(int i = 0; i < m + n + 1; ++i)
75     if(i != p) T[q][i] /= T[q][p];
76
77 T[q][p] = 1; // pivot(q, p)
78 base[p] = 1;
79 base[row[q]] = 0;
80 row[q] = p;
81
82 for(int j = 0; j < m + 1; ++j){
83     if(j != q){
84         double alpha = T[j][p];
85         for(int i = 0; i < n + m + 1; ++i)
86             T[j][i] -= T[q][i] * alpha;
87     }
88 }
89
90 return {vec(n), oo};
91 }
92
93 int main(){
94     int m, n;
95     bool mini = true;
96     cout << "Número de restricciones:" ;
97     cin >> m;
98     cout << "Número de incognitas:" ;
99     cin >> n;
100    mat A(m, vec(n));
101    vec b(m), c(n);
102 }
```

```

103 for(int i = 0; i < m; ++i){
104     cout << "Restriccion#" << (i + 1) << ":" ;
105     for(int j = 0; j < n; ++j){
106         cin >> A[i][j];
107     }
108     cin >> b[i];
109 }
110 cout << "[0]Max_o [1]Min?: ";
111 cin >> mini;
112 cout << "Coeficientes de" << (mini ? "min" : "max") << "z";
113 for(int i = 0; i < n; ++i){
114     cin >> c[i];
115 }
116 cout.precision(6);
117 auto ans = simplexMethodPD(A, b, c, mini);
118 cout << (mini ? "Min" : "Max") << "uz" << ans.second << ", cuando: "
119 \n";
120 for(int i = 0; i < ans.first.size(); ++i){
121     cout << "x_" << (i + 1) << "uz" << ans.first[i] << "\n";
122 }
123 }
```

6 Graphs

6.1 Djikstra

```

1 priority_queue<ll> q;
2 dist[0]=0;
3 q.push({0, 0});
4 while((int)q.size()){
5     auto [w, v]=q.top(); q.pop();
6     w=-w;
7     if (w>=dist[v]) continue;
8     for (auto [u, p]:adj[v]){
9         if(dist[v]+p<dist[u]){
10             dist[u]=dist[v]+p;
11             q.push({-dist[p], u});
12         }
13     }
14 }
```

6.2 FloydWarshall

```

1 vector<vector<ll>> d(n, vector<ll>(n, 1e18));
2
3 for (int k = 0; k < n; k++) {
4     for (int i = 0; i < n; i++) {
5         for (int j = i + 1; j < n; j++) {
6             long long new_dist = d[i][k] + d[k][j];
7             if (new_dist < d[i][j]) {
8                 d[i][j] = d[j][i] = new_dist;
9             }
10        }
11    }
12}

```

6.3 BellmanFord

```

1 void solve()
2 {
3     vector<int> d(n, INF);
4     d[v] = 0;
5     vector<int> p(n, -1);
6     int x;
7     for (int i = 0; i < n; ++i) {
8         x = -1;
9         for (Edge e : edges)
10            if (d[e.a] < INF)
11                if (d[e.b] > d[e.a] + e.cost) {
12                    d[e.b] = max(-INF, d[e.a] + e.cost);
13                    p[e.b] = e.a;
14                    x = e.b;
15                }
16    }
17
18    if (x == -1)
19        cout << "No negative cycle from " << v;
20    else {
21        int y = x;
22        for (int i = 0; i < n; ++i)
23            y = p[y];
24
25        vector<int> path;
26        for (int cur = y;; cur = p[cur]) {
27            path.push_back(cur);
28            if (cur == y && path.size() > 1)

```

```

29                break;
30            }
31            reverse(path.begin(), path.end());
32
33            cout << "Negative_cycle:" ;
34            for (int u : path)
35                cout << u << ' ';
36        }
37    }

```

6.4 BellmanFordFast(SPFA)

```

1 const int INF = 1000000000;
2 vector<vector<pair<int, int>>> adj;
3
4 bool spfa(int s, vector<int>& d) {
5     int n = adj.size();
6     d.assign(n, INF);
7     vector<int> cnt(n, 0);
8     vector<bool> inqueue(n, false);
9     queue<int> q;
10
11    d[s] = 0;
12    q.push(s);
13    inqueue[s] = true;
14    while (!q.empty()) {
15        int v = q.front();
16        q.pop();
17        inqueue[v] = false;
18
19        for (auto edge : adj[v]) {
20            int to = edge.first;
21            int len = edge.second;
22
23            if (d[v] + len < d[to]) {
24                d[to] = d[v] + len;
25                if (!inqueue[to]) {
26                    q.push(to);
27                    inqueue[to] = true;
28                    cnt[to]++;
29                    if (cnt[to] > n)
30                        return false; // negative cycle
31            }

```

```

32         }
33     }
34     return true;
35 }
36 }
```

6.5 EulerianPath

An Eulerian Path is a path that passes through every edge once. For an undirected graph an eulerian path exists if the degree of every node is even or the degree of exactly two nodes is odd. In the first case, the eulerian path is also an eulerian circuit or cycle. In a directed graph, an eulerian path exists if at most one node has $out_i - in_i = 1$ and at most one node has $in_i - out_i = 1$. A cycle exists if $in_i - out_i = 0$ for all i.

```

1 // check if all edges are visite
2 // directed
3 void dfs(int node) {
4     while (!g[node].empty()) {
5         auto [son, idx] = g[node].back();
6         g[node].pop_back();
7         if (seen[idx]) { continue; }
8         seen[idx] = true;
9         dfs(son);
10    }
11    path.push_back(node);
12 }
13 //undirected
14 void dfs(int node) {
15     while (!g[node].empty()) {
16         int son = g[node].back();
17         g[node].pop_back();
18         dfs(son);
19     }
20     path.push_back(node);
21 }
```

6.6 Kruskal

```

1 template <class T> T kruskal(int N, vector<pair<T, pair<int, int>>>
2     edges) {
3     sort(edges.begin(), edges.end());
4     T ans = 0;
5     DSU D(N + 1); // edges that unite are in MST
```

```

5     for (pair<T, pair<int, int>> &e : edges) {
6         if (D.unite(e.second.first, e.second.second)) { ans += e.first; }
7     }
8     // -1 if the graph is not connected, otherwise the sum of the edge
9     // lengths
10    return (D.size(1) == N ? ans : -1);
11 }
```

6.7 Strongly Connected Components

```

1 const int MAXN=100005;
2 vector<int> adj[MAXN];
3 vector<int> adj1[MAXN];
4 int comp[MAXN];
5 bool vis[MAXN];
6 vector<int> order;
7 int component=0;
8
9 void dfs(int cur){
10    vis[cur]=1;
11    for (int x: adj[cur]){
12        if(!vis[x]){
13            dfs(x);
14        }
15    }
16    order.push_back(cur);
17 }
18
19 void dfs1(int cur){
20    vis[cur]=1;
21    comp[cur]=component;
22    for (int x: adj1[cur]){
23        if(!vis[x]){
24            dfs1(x);
25        }
26    }
27 }
```

6.8 2Sat

```

1 struct TwoSatSolver {
2     int n_vars;
3     int n_vertices;
4     vector<vector<int>> adj, adj_t;
```

```

5     vector<bool> used;
6     vector<int> order, comp;
7     vector<bool> assignment;
8
9     TwoSatSolver(int _n_vars) : n_vars(_n_vars), n_vertices(2 * n_vars),
10      adj(n_vertices), adj_t(n_vertices), used(n_vertices), order(),
11      comp(n_vertices, -1), assignment(n_vars) {
12      order.reserve(n_vertices);
13    }
14    void dfs1(int v) {
15      used[v] = true;
16      for (int u : adj[v]) {
17        if (!used[u])
18          dfs1(u);
19      }
20      order.push_back(v);
21    }
22
23    void dfs2(int v, int cl) {
24      comp[v] = cl;
25      for (int u : adj_t[v]) {
26        if (comp[u] == -1)
27          dfs2(u, cl);
28      }
29
30      bool solve_2SAT() {
31        order.clear();
32        used.assign(n_vertices, false);
33        for (int i = 0; i < n_vertices; ++i) {
34          if (!used[i])
35            dfs1(i);
36
37          comp.assign(n_vertices, -1);
38          for (int i = 0, j = 0; i < n_vertices; ++i) {
39            int v = order[n_vertices - i - 1];
40            if (comp[v] == -1)
41              dfs2(v, j++);
42
43          assignment.assign(n_vars, false);
44          for (int i = 0; i < n_vertices; i += 2) {

```

```

46          if (comp[i] == comp[i + 1])
47            return false;
48          assignment[i / 2] = comp[i] > comp[i + 1];
49        }
50        return true;
51      }
52
53      void add_disjunction(int a, bool na, int b, bool nb) {
54        // na==1 means not a
55        a = 2 * a ^ na;
56        b = 2 * b ^ nb;
57        int neg_a = a ^ 1;
58        int neg_b = b ^ 1;
59        adj[neg_a].push_back(b);
60        adj[neg_b].push_back(a);
61        adj_t[b].push_back(neg_a);
62        adj_t[a].push_back(neg_b);
63      }
64    };

```

6.9 Finding Bridges

```

1 int n; // number of nodes
2 vector<vector<int>> adj; // adjacency list of graph
3
4 vector<bool> visited;
5 vector<int> tin, low;
6 int timer;
7
8 void dfs(int v, int p = -1) {
9   visited[v] = true;
10  tin[v] = low[v] = timer++;
11  for (int to : adj[v]) {
12    if (to == p) continue;
13    if (visited[to]) {
14      low[v] = min(low[v], tin[to]);
15    } else {
16      dfs(to, v);
17      low[v] = min(low[v], low[to]);
18      if (low[to] > tin[v])
19        IS_BRIDGE(v, to);
20    }
21  }

```

```

22 }
23
24 void find_bridges() {
25     timer = 0;
26     visited.assign(n, false);
27     tin.assign(n, -1);
28     low.assign(n, -1);
29     for (int i = 0; i < n; ++i) {
30         if (!visited[i])
31             dfs(i);
32     }
33 }
```

6.10 Finding Bridges Online

```

1 vector<int> par, dsu_2ecc, dsu_cc, dsu_cc_size;
2 int bridges;
3 int lca_iteration;
4 vector<int> last_visit;
5
6 void init(int n) {
7     par.resize(n);
8     dsu_2ecc.resize(n);
9     dsu_cc.resize(n);
10    dsu_cc_size.resize(n);
11    lca_iteration = 0;
12    last_visit.assign(n, 0);
13    for (int i=0; i<n; ++i) {
14        dsu_2ecc[i] = i;
15        dsu_cc[i] = i;
16        dsu_cc_size[i] = 1;
17        par[i] = -1;
18    }
19    bridges = 0;
20 }
21
22 int find_2ecc(int v) {
23     if (v == -1)
24         return -1;
25     return dsu_2ecc[v] == v ? v : dsu_2ecc[v] = find_2ecc(dsu_2ecc[v]);
26 }
27
28 int find_cc(int v) {
```

```

29     v = find_2ecc(v);
30     return dsu_cc[v] == v ? v : dsu_cc[v] = find_cc(dsu_cc[v]);
31 }
32
33 void make_root(int v) {
34     int root = v;
35     int child = -1;
36     while (v != -1) {
37         int p = find_2ecc(par[v]);
38         par[v] = child;
39         dsu_cc[v] = root;
40         child = v;
41         v = p;
42     }
43     dsu_cc_size[root] = dsu_cc_size[child];
44 }
45
46 void merge_path (int a, int b) {
47     ++lca_iteration;
48     vector<int> path_a, path_b;
49     int lca = -1;
50     while (lca == -1) {
51         if (a != -1) {
52             a = find_2ecc(a);
53             path_a.push_back(a);
54             if (last_visit[a] == lca_iteration){
55                 lca = a;
56                 break;
57             }
58             last_visit[a] = lca_iteration;
59             a = par[a];
60         }
61         if (b != -1) {
62             b = find_2ecc(b);
63             path_b.push_back(b);
64             if (last_visit[b] == lca_iteration){
65                 lca = b;
66                 break;
67             }
68             last_visit[b] = lca_iteration;
69             b = par[b];
70         }
71 }
```

```

72 }
73
74     for (int v : path_a)
75         dsu_2ecc[v] = lca
76         if (v == lca)
77             break;
78         --bridges;
79     }
80     for (int v : path_b)
81         dsu_2ecc[v] = lca
82         if (v == lca)
83             break;
84         --bridges;
85     }
86 }
87
88 void add_edge(int a, int b) {
89     a = find_2ecc(a);
90     b = find_2ecc(b);
91     if (a == b)
92         return;
93
94     int ca = find_cc(a);
95     int cb = find_cc(b);
96
97     if (ca != cb) {
98         ++bridges;
99         if (dsu_cc_size[ca] < dsu_cc_size[cb])
100            swap(a, b);
101            swap(ca, cb);
102        }
103        make_root(a);
104        par[a] = dsu_cc[ca];
105        dsu_cc_size[cb] += dsu_cc_size[ca];
106    } else {
107        merge_path(a, b);
108    }
109 }

```

```

3     vector<bool> visited;
4     vector<int> tin, low;
5     int timer;
6
7
8     void dfs(int v, int p = -1) {
9         visited[v] = true;
10        tin[v] = low[v] = timer++;
11        int children=0;
12        for (int to : adj[v]) {
13            if (to == p) continue;
14            if (visited[to]) {
15                low[v] = min(low[v], tin[to]);
16            } else {
17                dfs(to, v);
18                low[v] = min(low[v], low[to]);
19                if (low[to] >= tin[v] && p!=-1)
20                    IS_CUTPOINT(v);
21                ++children;
22            }
23        }
24        if(p == -1 && children > 1)
25            IS_CUTPOINT(v);
26    }
27
28    void find_cutpoints() {
29        timer = 0;
30        visited.assign(n, false);
31        tin.assign(n, -1);
32        low.assign(n, -1);
33        for (int i = 0; i < n; ++i) {
34            if (!visited[i])
35                dfs (i);
36        }
37    }

```

6.11 Articulation Points

```
1 | int n; // number of nodes
2 | vector<vector<int>> adj; // adjacency list of graph
```

6.12 Biconnected Components (Block Cut Tree)

Builds the Block Cut Tree of graph in $O(n)$.

```

4
5     int n = (int)g.size();
6     vector<vector<int>> comps;
7     vector<int> stk;
8     vector<int> num(n);
9     vector<int> low(n);
10    is_cutpoint.resize(n);
11    // Finds the biconnected components
12    function<void(int, int, int &)> dfs = [&](int node, int parent, int &
13        timer) {
14        num[node] = low[node] = ++timer;
15        stk.push_back(node);
16        for (int son : g[node]) {
17            if (son == parent) { continue; }
18            if (num[son]) {
19                low[node] = min(low[node], num[son]);
20            } else {
21                dfs(son, node, timer);
22                low[node] = min(low[node], low[son]);
23                if (low[son] >= num[node]) {
24                    is_cutpoint[node] = (num[node] > 1 || num[son] > 2);
25                    comps.push_back({node});
26                    while (comps.back().back() != son) {
27                        comps.back().push_back(stk.back());
28                        stk.pop_back();
29                    }
30                }
31            }
32        }
33    };
34
35    int timer = 0;
36    dfs(0, -1, timer);
37    id.resize(n);
38    // Build the block-cut tree
39
40    function<vector<vector<int>>()> build_tree = [&]() {
41        vector<vector<int>> t(1);
42        int node_id = 0;
43        for (int node = 0; node < n; node++) {
44            if (is_cutpoint[node]) {
45                id[node] = node_id++;
46                t.push_back({});

```

```

46
47     }
48
49     for (auto &comp : comps) {
50         int node = node_id++;
51         t.push_back({});
52         for (int u : comp)
53             if (!is_cutpoint[u]) {
54                 id[u] = node;
55             } else {
56                 t[node].push_back(id[u]);
57                 t[id[u]].push_back(node);
58             }
59         }
60         return t;
61     };
62     return build_tree();
63
64 }

```

6.13 Blossom

Finds maximum matching in general graph in $O(n^3)$

```
1 const int N = 2e3 + 9;
2
3 mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());
4 struct Blossom {
5     int vis[N], par[N], orig[N], match[N], aux[N], t;
6     int n;
7     bool ad[N];
8     vector<int> g[N];
9     queue<int> Q;
10    Blossom() {}
11    Blossom(int _n) {
12        n = _n;
13        t = 0;
14        for (int i = 0; i <= _n; ++i) {
15            g[i].clear();
16            match[i] = aux[i] = par[i] = vis[i] = aux[i] = ad[i] = orig[i] =
17                0;
18        }
19    }
20}
```

```

19 void add_edge(int u, int v) {
20     g[u].push_back(v);
21     g[v].push_back(u);
22 }
23 void augment(int u, int v) {
24     int pv = v, nv;
25     do {
26         pv = par[v];
27         nv = match[pv];
28         match[v] = pv;
29         match[pv] = v;
30         v = nv;
31     } while (u != pv);
32 }
33 int lca(int v, int w) {
34     ++t;
35     while (true) {
36         if (v) {
37             if (aux[v] == t) return v;
38             aux[v] = t;
39             v = orig[par[match[v]]];
40         }
41         swap(v, w);
42     }
43 }
44 void blossom(int v, int w, int a) {
45     while (orig[v] != a) {
46         par[v] = w;
47         w = match[v];
48         ad[v] = true;
49         if (vis[w] == 1) Q.push(w), vis[w] = 0;
50         orig[v] = orig[w] = a;
51         v = par[w];
52     }
53 }
54 //it finds an augmented path starting from u
55 bool bfs(int u) {
56     fill(vis + 1, vis + n + 1, -1);
57     iota(orig + 1, orig + n + 1, 1);
58     Q = queue<int>();
59     Q.push(u);
60     vis[u] = 0;
61     while (!Q.empty()) {
62         int v = Q.front();
63         Q.pop();
64         ad[v] = true;
65         for (int x : g[v]) {
66             if (vis[x] == -1) {
67                 par[x] = v;
68                 vis[x] = 1;
69                 if (!match[x]) return augment(u, x), true;
70                 Q.push(match[x]);
71                 vis[match[x]] = 0;
72             } else if (vis[x] == 0 && orig[v] != orig[x]) {
73                 int a = lca(orig[v], orig[x]);
74                 blossom(x, v, a);
75                 blossom(v, x, a);
76             }
77         }
78     }
79     return false;
80 }
81 int maximum_matching() {
82     int ans = 0;
83     vector<int> p(n - 1);
84     iota(p.begin(), p.end(), 1);
85     shuffle(p.begin(), p.end(), rnd);
86     for (int i = 1; i <= n; i++) shuffle(g[i].begin(), g[i].end(), rnd);
87     for (auto u : p) {
88         if (!match[u]) {
89             for (auto v : g[u]) {
90                 if (!match[v]) {
91                     match[u] = v, match[v] = u;
92                     ++ans;
93                     break;
94                 }
95             }
96         }
97     }
98     for (int i = 1; i <= n; ++i) if (!match[i] && bfs(i)) ++ans;
99     return ans;
100 }
101 } M;
102 int32_t main() {
103     ios_base::sync_with_stdio(0);
104     cin.tie(0);

```

```

105 int t;
106 cin >> t;
107 while (t--) {
108     int n, m;
109     cin >> n >> m;
110     M = Blossom (n);
111     while (m--) {
112         int u, v;
113         cin >> u >> v;
114         M.add_edge(u, v);
115     }
116     int ans = M.maximum_matching();
117     if (ans * 2 == n) cout << 0 << '\n';
118     else {
119         memset(M.ad, 0, sizeof M.ad);
120         for (int i = 1; i <= n; i++) if (M.match[i] == 0) M.bfs(i);
121         int ans = 0;
122         for (int i = 1; i <= n; i++) ans += M.ad[i]; //nodes which are
123             unmatched in some maximum matching
124         cout << ans << '\n';
125     }
126 }
127 return 0;
}

```

6.14 Marriage

```

1 // MALE OPTIMAL STABLE MARRIAGE PROBLEM O(N^2)
2 // gv[i][j] jth most prefered female for ith male
3 // om[i][j] jth most prefered male for ith female
4 #define MAXN 1000
5 int gv[MAXN][MAXN], om[MAXN][MAXN];
6 int pv[MAXN], pm[MAXN]; // ans
7 int pun[MAXN]; // Auxiliary
8
9 void stableMarriage(int n) {
10     fill_n(pv, n, -1); fill_n(pm, n, -1); fill_n(pun, n, 0);
11     int s = n, i = n-1;
12     #define engage pm[j] = i; pv[i] = j;
13     while (s) {
14         while (pv[i] == -1) {
15             int j = gv[i][pun[i]++;

```

```

16         if (pm[j] == -1) {
17             s--;
18             engage;
19         }
20         else if (om[j][i] < om[j][pm[j]]) {
21             int loser = pm[j];
22             pv[loser] = -1;
23             engage;
24             i = loser;
25         }
26     }
27     i--;
28     if (i < 0) i = n-1;
29 }
30 }

```

7 Trees

7.1 Tree Diameter

```

1 pair<int, int> dfs(const vector<vector<int>> &tree, int node = 1,
2     int previous = 0, int length = 0) {
3     pair<int, int> max_path = {node, length};
4     for (const int &i : tree[node]) {
5         if (i == previous) { continue; }
6         pair<int, int> other = dfs(tree, i, node, length + 1);
7         if (other.second > max_path.second) { max_path = other; }
8     }
9     return max_path;
10 }

```

7.2 Heavy Light Decomposition

```

1 //call dfs1 first
2 struct SegmentTree {
3     vector<ll> a;
4     int n;
5
6     SegmentTree(int _n) : a(2 * _n, 0), n(_n) {}
7
8     void update(int pos, ll val) {
9         for (a[pos += n] = val; pos > 1; pos >>= 1) {
10             a[pos / 2] = (a[pos] ^ a[pos ^ 1]);

```

```

11     }
12 }
13
14 ll get(int l, int r) {
15     ll res = 0;
16     for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
17         if (l & 1) {
18             res^=a[l++];
19         }
20         if (r & 1) {
21             res^=a[--r];
22         }
23     }
24     return res;
25 }
26 };
27
28
29 const int MAXN=500005;
30 vector<int> adj[MAXN];
31 SegmentTree st(MAXN);
32 int a[MAXN], sz[MAXN], to[MAXN], dpth[MAXN], s[MAXN], par[MAXN];
33 int cnt=0;
34
35 void dfs1(int cur, int p){
36     sz[cur]=1;
37     for(int x:adj[cur]){
38         if(x==p) continue;
39         dpth[x]=dpth[cur]+1;
40         par[x]=cur;
41         dfs1(x, cur);
42         sz[cur]+=sz[x];
43     }
44 }
45
46 void dfs(int cur, int p, int l){
47     st.update(cnt, a[cur]);
48     s[cur]=cnt++;
49     to[cur]=l;
50     int g=-1;
51     for(int x:adj[cur]){
52         if(x==p) continue;
53         if(g==-1 || sz[g]<sz[x]){
54             g=x;
55         }
56     }
57     if(g==-1) return;
58     dfs(g, cur, l);
59     for(int x:adj[cur]){
60         if(x==p || x==g) continue;
61         dfs(x, cur, x);
62     }
63 }
64
65 int query(int u, int v){
66     int res=0;
67     while(to[u]!=to[v]){
68         if(dpth[to[u]]<dpth[to[v]]) swap(u, v);
69         res^=st.get(s[to[u]], s[u]+1);
70         u=par[to[u]];
71     }
72     if(dpth[u]>dpth[v]) swap(u, v);
73     res^=st.get(s[u], s[v]+1);
74     return res;
75 }
76
77
78
79 //alternate implementation
80 vector<int> parent, depth, heavy, head, pos;
81 int cur_pos;
82
83
84 int dfs(int v, vector<vector<int>> const& adj) {
85     int size = 1;
86     int max_c_size = 0;
87     for (int c : adj[v]) {
88         if (c != parent[v]) {
89             parent[c] = v, depth[c] = depth[v] + 1;
90             int c_size = dfs(c, adj);
91             size += c_size;
92             if (c_size > max_c_size)
93                 max_c_size = c_size, heavy[v] = c;
94         }
95     }
96     return size;

```

```

97 }
98
99 void decompose(int v, int h, vector<vector<int>> const& adj) {
100    head[v] = h, pos[v] = cur_pos++;
101    if (heavy[v] != -1)
102        decompose(heavy[v], h, adj);
103    for (int c : adj[v]) {
104        if (c != parent[v] && c != heavy[v])
105            decompose(c, c, adj);
106    }
107 }
108
109 void init(vector<vector<int>> const& adj) {
110    int n = adj.size();
111    parent = vector<int>(n);
112    depth = vector<int>(n);
113    heavy = vector<int>(n, -1);
114    head = vector<int>(n);
115    pos = vector<int>(n);
116    cur_pos = 0;
117
118    dfs(0, adj);
119    decompose(0, 0, adj);
120 }
121
122 int query(int a, int b) {
123    int res = 0;
124    for (; head[a] != head[b]; b = parent[head[b]]) {
125        if (depth[head[a]] > depth[head[b]])
126            swap(a, b);
127        int cur_heavy_path_max = segment_tree_query(pos[head[b]], pos[b]
128            ]);
129        res = max(res, cur_heavy_path_max);
130    }
131    if (depth[a] > depth[b])
132        swap(a, b);
133    int last_heavy_path_max = segment_tree_query(pos[a], pos[b]);
134    res = max(res, last_heavy_path_max);
135    return res;
}

```

7.3 Lowest Common Ancestor (LCA)

```

1  const int N=200005;
2  vector<int> adj[N];
3  vector<int> start(N), end1(N), depth(N);
4  vector<vector<int>> t(N, vi(32));
5  int timer=0;
6  int n, l;
// l=(int)ceil(log2(n))
// call dfs(1, 1, 0)
// 1 indexed, dont use 0 indexing
10
11
12 void dfs(int cur, int p, int cnt){
13    depth[cur]=cnt;
14    t[cur][0]=p;
15    start[cur]=timer++;
16    for(int i=1;i<=l;i++){
17        t[cur][i]=t[t[cur][i-1]][i-1];
18    }
19    for(int x:adj[cur]){
20        if(x==p) continue;
21        dfs(x, cur, cnt+1);
22    }
23    end1[cur]=++timer;
}
24
25
26 bool ancestor(int u, int v){
27    return start[u]<=start[v] && end1[u]>=end1[v];
}
28
29
30 int lca(int u, int v){
31    if(ancestor(u, v))
32        return u;
33    if (ancestor(v, u)){
34        return v;
35    }
36    for(int i=l;i>=0;i--){
37        if(!ancestor(t[u][i], v)){
38            u=t[u][i];
39        }
40    }
41    return t[u][0];
}
42

```

7.4 Centroid Decomposition

```

1 // code for xenia and tree
2 const int MAXN=200005;
3
4 vector<int> adj[MAXN];
5 vector<bool> is_removed(MAXN, false);
6 vector<int> subtree_size(MAXN, 0);
7 vector<int> dis(MAXN, 1e9);
8 vector<vector<pair<int, int>>> ancestor(MAXN);
9
10 int get_subtree_size(int node, int parent = -1) {
11     subtree_size[node] = 1;
12     for (int child : adj[node]) {
13         if (child == parent || is_removed[child]) { continue; }
14         subtree_size[node] += get_subtree_size(child, node);
15     }
16     return subtree_size[node];
17 }
18
19 int get_centroid(int node, int tree_size, int parent = -1) {
20     for (int child : adj[node]) {
21         if (child == parent || is_removed[child]) { continue; }
22         if (subtree_size[child] * 2 > tree_size) {
23             return get_centroid(child, tree_size, node);
24         }
25     }
26     return node;
27 }
28
29 void getDist(int cur, int centroid, int p=-1, int dist=1){
30     for (int child:adj[cur]){
31         if(child==p || is_removed[child])
32             continue;
33         dist++;
34         getDist(child, centroid, cur, dist);
35         dist--;
36     }
37     ancestor[cur].push_back(make_pair(centroid, dist));
38 }
39
40 void update(int cur){
41     for (int i=0;i<ancestor[cur].size();i++){

```

```

42         dis[ancestor[cur][i].first]=min(dis[ancestor[cur][i].first],
43                                         ancestor[cur][i].second);
44     }
45     dis[cur]=0;
46 }
47
48 int query(int cur){
49     int mini=dis[cur];
50     for (int i=0;i<ancestor[cur].size();i++){
51         mini=min(mini, ancestor[cur][i].second+dis[ancestor[cur][i].first
52                                         ]);
53     }
54     return mini;
55 }
56
57 void build_centroid_decomp(int node = 1) {
58     int centroid = get_centroid(node, get_subtree_size(node));
59
60     for (int child : adj[centroid]) {
61         if (is_removed[child]) { continue; }
62         getDist(child, centroid, centroid);
63     }
64
65     is_removed[centroid] = true;
66
67     for (int child : adj[centroid]) {
68         if (is_removed[child]) { continue; }
69         build_centroid_decomp(child);
70     }
71 }
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
625
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1
```

```

9   vector<FlowEdge> edges;
10  vector<vector<int>> adj;
11  int n, m = 0;
12  int s, t;
13  vector<int> level, ptr;
14  queue<int> q;
15
16  Dinic(int n, int s, int t) : n(n), s(s), t(t) {
17      adj.resize(n);
18      level.resize(n);
19      ptr.resize(n);
20  }
21
22  void add_edge(int v, int u, long long cap) {
23      edges.emplace_back(v, u, cap);
24      edges.emplace_back(u, v, 0);
25      adj[v].push_back(m);
26      adj[u].push_back(m + 1);
27      m += 2;
28  }
29
30  bool bfs() {
31      while (!q.empty()) {
32          int v = q.front();
33          q.pop();
34          for (int id : adj[v]) {
35              if (edges[id].cap - edges[id].flow < 1)
36                  continue;
37              if (level[edges[id].u] != -1)
38                  continue;
39              level[edges[id].u] = level[v] + 1;
40              q.push(edges[id].u);
41          }
42      }
43      return level[t] != -1;
44  }
45
46  long long dfs(int v, long long pushed) {
47      if (pushed == 0)
48          return 0;
49      if (v == t)
50          return pushed;
51      for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
52          int id = adj[v][cid];
53          int u = edges[id].u;
54          if (level[v] + 1 != level[u] || edges[id].cap - edges[id].flow < 1)
55              continue;
56          long long tr = dfs(u, min(pushed, edges[id].cap - edges[id].flow));
57          if (tr == 0)
58              continue;
59          edges[id].flow += tr;
60          edges[id ^ 1].flow -= tr;
61          return tr;
62      }
63      return 0;
64  }
65
66  long long flow() {
67      long long f = 0;
68      while (true) {
69          fill(level.begin(), level.end(), -1);
70          level[s] = 0;
71          q.push(s);
72          if (!bfs())
73              break;
74          fill(ptr.begin(), ptr.end(), 0);
75          while (long long pushed = dfs(s, flow_inf)) {
76              f += pushed;
77          }
78      }
79      return f;
80  }
81 }

```

8.2 Hungarian

```

1 #define forn(i,n) for(int i=0;i<int(n);++i)
2 #define forsn(i,s,n) for(int i=s;i<int(n);++i)
3 #define forall(i,c) for(typeof(c.begin()) i=c.begin();i!=c.end();++i)
4 #define DBG(X) cerr << #X << " = " << X << endl;
5 typedef vector<int> vint;
6 typedef vector<vint> vvint;
7
8 void showmt();

```

```

9  /*
10 */
11
12 #define MAXN 256
13 #define INFTO 0x7f7f7f7f
14 int n;
15 int mt[MAXN][MAXN]; // Matriz de costos (X * Y)
16 int xy[MAXN], yx[MAXN]; // Matching resultante (X->Y, Y->X)
17
18 int lx[MAXN], ly[MAXN], slk[MAXN], slkx[MAXN], prv[MAXN];
19 char S[MAXN], T[MAXN];
20
21 void updtree(int x) {
22     forn(y, n) if (lx[x] + ly[y] - mt[x][y] < slk[y]) {
23         slk[y] = lx[x] + ly[y] - mt[x][y];
24         slkx[y] = x;
25     }
26 }
27 int hungar() {
28     forn(i, n) {
29         ly[i] = 0;
30         lx[i] = *max_element(mt[i], mt[i]+n);
31     }
32     memset(xy, -1, sizeof(xy));
33     memset(yx, -1, sizeof(yx));
34
35     forn(m, n) {
36         memset(S, 0, sizeof(S));
37         memset(T, 0, sizeof(T));
38         memset(prv, -1, sizeof(prv));
39         memset(slk, 0x7f, sizeof(slk));
40         queue<int> q;
41         #define bpone(e, p) { q.push(e); prv[e] = p; S[e] = 1; updtree(e); }
42         forn(i, n) if (xy[i] == -1) { bpone(i, -2); break; }
43
44         int x=0, y=-1;
45         while (y== -1) {
46             while (!q.empty() && y== -1) {
47                 x = q.front(); q.pop();
48                 forn(j, n) if (mt[x][j] == lx[x] + ly[j] && !T[j]) {
49                     if (yx[j] == -1) { y = j; break; }
50                     T[j] = 1;
51                     bpone(yx[j], x);
52             }
53         }
54         if (y!= -1) break;
55         int dlt = INFTO;
56         forn(j, n) if (!T[j]) dlt = min(dlt, slk[j]);
57         forn(k, n) {
58             if (S[k]) lx[k] -= dlt;
59             if (T[k]) ly[k] += dlt;
60             if (!T[k]) slk[k] -= dlt;
61         }
62     // q = queue<int>();
63     forn(j, n) if (!T[j] && !slk[j]) {
64         if (yx[j] == -1) {
65             x = slkx[j]; y = j; break;
66         } else {
67             T[j] = 1;
68             if (!S[yx[j]]) bpone(yx[j], slkx[j]);
69         }
70     }
71     if (y!= -1) {
72         for(int p = x; p != -2; p = prv[p]) {
73             yx[y] = p;
74             int ty = xy[p]; xy[p] = y; y = ty;
75         }
76     } else break;
77 }
78 int res = 0;
79 forn(i, n) res += mt[i][xy[i]];
80 return res;
81 }
82 }

1 /**
2 * If costs can be negative, call setpi before maxflow, but note that
3 * negative cost cycles are not supported.
4 * To obtain the actual flow, look at positive values only
5 * Time: $O(F E \log(V))$ where F is max flow. $O(VE)$ for setpi.
6 */
7 #include <bits/stdc++.h>
8 using namespace std;

```

8.3 Min-cost Max-Flow


```

95     if ((v = pi[i] + e.cost) < pi[e.to])
96         pi[e.to] = v, ch = 1;
97     assert(it >= 0); // negative cost cycle
98 }
99 };

```

9 Strings

9.1 KMP-Prefix Function

```

1 // Maximum length of substring that ends at position i and is prefix of
2     string
3 vector<int> KMP(string s){
4     int n=s.length();
5     vector<int> prefixFunction(n, 0);
6     for(int i=1;i<n;i++){
7         int j=prefixFunction[i-1];
8         while(j>0 && s[i]!=s[j]){
9             j=prefixFunction[j-1];
10        }
11        if(s[i]==s[j]){
12            prefixFunction[i]=j+1;
13        }
14    }
15    return prefixFunction;
}

```

9.2 Z-Function

```

1 // Maximum length of substring that begins at position i and is prefix
2     of stringS
3 vector<int> z_function(string s) {
4     int n = s.size();
5     vector<int> z(n);
6     int l = 0, r = 0;
7     for(int i = 1; i < n; i++) {
8         if(i < r) {
9             z[i] = min(r - i, z[i - 1]);
10        }
11        while(i + z[i] < n && s[z[i]] == s[i + z[i]]) {
12            z[i]++;
13        }
14        if(i + z[i] > r) {

```

```

14     l = i;
15     r = i + z[i];
16   }
17 }
18 return z;
}

```

9.3 Manacher

```

1 // finds length of palindromes centered at i
2 vector<int> manacher_odd(string s) {
3     int n = s.size();
4     s = "$" + s + "^";
5     vector<int> p(n + 2);
6     int l = 1, r = 1;
7     for(int i = 1; i <= n; i++) {
8         p[i] = max(0, min(r - i, p[l + (r - i)]));
9         while(s[i - p[i]] == s[i + p[i]]) {
10            p[i]++;
11        }
12        if(i + p[i] > r) {
13            l = i - p[i], r = i + p[i];
14        }
15    }
16    return vector<int>(begin(p) + 1, end(p) - 1);
}

```

9.4 String Hashing

```

1 const ll N = 2e5+1;
2 const ll MOD = 212345678987654321LL;
3 const ll base = 33;
4
5 // double hash or big mod values
6
7 class HashedString {
8 private:
9     // change M and B if you want
10    static const long long M = 1e9 + 9;
11    static const long long B = 9973;
12
13    // pow[i] contains B^i % M
14    static vector<long long> pow;
15

```

```

16 // p_hash[i] is the hash of the first i characters of the given string
17 vector<long long> p_hash;
18
19 public:
20 HashedString(const string &s) : p_hash(s.size() + 1) {
21     while (pow.size() < s.size()) { pow.push_back((pow.back() * B) % M);
22         }
23
24     p_hash[0] = 0;
25     for (int i = 0; i < s.size(); i++) {
26         p_hash[i + 1] = ((p_hash[i] * B) % M + s[i]) % M;
27     }
28
29     long long get_hash(int start, int end) {
30         long long raw_val =
31             (p_hash[end + 1] - (p_hash[start] * pow[end - start + 1]));
32         return (raw_val % M + M) % M;
33     }
34 };
35 vector<long long> HashedString::pow = {1};

```

9.5 MinCyclicRotation

```

1 string least_rotation(string s)
{
2     s += s;
3     vector<int> f(s.size(), -1);
4     int k = 0;
5     for(int j = 1; j < s.size(); j++)
6     {
7         char sj = s[j];
8         int i = f[j - k - 1];
9         while(i != -1 && sj != s[k + i + 1])
10        {
11            if(sj < s[k + i + 1]){
12                k = j - i - 1;
13            }
14            i = f[i];
15        }
16        if(sj != s[k + i + 1])
17        {
18            if(sj < s[k]){

```

```

20             k = j;
21         }
22         f[j - k] = -1;
23     }
24     else
25         f[j - k] = i + 1;
26 }
27 return s.substr(k, s.size() / 2);
28 }

```

9.6 Suffix Array

```

1 //beware of first element of p being $
2 vector<int> suffixArray(string s){
3     int n, k=0;
4     s+="$";
5     n=s.size();
6     vector <int> p(n), c(n);
7     vector <pair<char, int>> a(n);
8     for (int i=0;i<n;i++)a[i]= {s[i], i};
9     sort(a.begin(), a.end());
10    for (int i=0;i<n;i++) p[i]=a[i].second;
11    c[p[0]]=0;
12    for (int i=1;i<n;i++){
13        if(a[i].first!=a[i-1].first) c[p[i]]=c[p[i-1]]+1;
14        else c[p[i]]=c[p[i-1]];
15    }
16    while((1<<k)<n){
17        for(int i=0;i<n;i++){
18            p[i]=(p[i]-(1<<k)+n)%n;
19        }
20        vector<int> cnt(n);
21        for(auto x:c){
22            cnt[x]++;
23        }
24        vector<int> p1(n), pos(n);
25        pos[0]=0;
26        for(int i=1;i<n;i++){
27            pos[i]=pos[i-1]+cnt[i-1];
28        }
29        for(auto x:p){
30            int i=c[x];
31            p1[pos[i]]=x;

```

```

32     pos[i]++;
33 }
34 p=p1;
35 vector<int> c1(n);
36 c1[p[0]]=0;
37 for(int i=1;i<n;i++){
38     pair<int, int> aa={c[p[i]], c[(p[i]+(1<<k))%n]};
39     pair<int, int> bb={c[p[i-1]], c[(p[i-1]+(1<<k))%n]};
40     if(bb==aa){
41         c1[p[i]]=c1[p[i-1]];
42     }
43     else{
44         c1[p[i]]=c1[p[i-1]]+1;
45     }
46 }
47 c=c1;
48 k++;
49 }
50 return p;
51 }

52 vector<int> lcp_construction(string const& s, vector<int> const& p) {
53     int n = s.size();
54     vector<int> rank(n, 0);
55     for (int i = 0; i < n; i++)
56         rank[p[i]] = i;

57     int k = 0;
58     vector<int> lcp(n-1, 0);
59     for (int i = 0; i < n; i++) {
60         if (rank[i] == n - 1) {
61             k = 0;
62             continue;
63         }
64         int j = p[rank[i] + 1];
65         while (i + k < n && j + k < n && s[i+k] == s[j+k])
66             k++;
67         lcp[rank[i]] = k;
68         if (k)
69             k--;
70     }
71     return lcp;
72 }
73 }
```

9.7 Trie AhoCorasick

```

1 const int K = 26;
2
3 struct Vertex {
4     int next[K];
5     bool output = false;
6     int p = -1;
7     char pch;
8     int link = -1;
9     int go[K];
10
11     Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
12         fill(begin(next), end(next), -1);
13         fill(begin(go), end(go), -1);
14     }
15 };
16
17 vector<Vertex> t(1);
18
19 void add_string(string const& s) {
20     int v = 0;
21     for (char ch : s) {
22         int c = ch - 'a';
23         if (t[v].next[c] == -1) {
24             t[v].next[c] = t.size();
25             t.emplace_back(v, ch);
26         }
27         v = t[v].next[c];
28     }
29     t[v].output = true;
30 }
31
32 int go(int v, char ch);
33
34 int get_link(int v) {
35     if (t[v].link == -1) {
36         if (v == 0 || t[v].p == 0)
37             t[v].link = 0;
38         else
39             t[v].link = go(get_link(t[v].p), t[v].pch);
40     }
41     return t[v].link;
42 }
```

```

42 }
43
44 int go(int v, char ch) {
45     int c = ch - 'a';
46     if (t[v].go[c] == -1) {
47         if (t[v].next[c] != -1)
48             t[v].go[c] = t[v].next[c];
49         else
50             t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
51     }
52     return t[v].go[c];
53 }
```

9.8 SuffixAutomaton

```

1 struct state {
2     int len, link, firstposition;
3     map<char, int> next;
4 };
5
6 const int MAXN = 100000;
7 state st[MAXN * 2];
8 ll cnt[MAXN*2], cntPaths[MAXN*2], cntSum[MAXN*2];
9 int sz, last;
10
11 void initSuffixAutomaton() {
12     st[0].len = 0;
13     st[0].link = -1;
14     sz++;
15     last = 0;
16 }
17
18 void insertChar(char c) {
19     int cur = sz++;
20     st[cur].len = st[last].len + 1;
21     st[cur].firstposition=st[last].len;
22     int p = last;
23     while (p != -1 && !st[p].next.count(c)) {
24         st[p].next[c] = cur;
25         p = st[p].link;
26     }
27     if (p == -1) {
28         st[cur].link = 0;
```

```

29     } else {
30         int q = st[p].next[c];
31         if (st[p].len + 1 == st[q].len) {
32             st[cur].link = q;
33         } else {
34             int clone = sz++;
35             st[clone].len = st[p].len + 1;
36             st[clone].next = st[q].next;
37             st[clone].link = st[q].link;
38             st[clone].firstposition=st[q].firstposition;
39             while (p != -1 && st[p].next[c] == q) {
40                 st[p].next[c] = clone;
41                 p = st[p].link;
42             }
43             st[q].link = st[cur].link = clone;
44         }
45     }
46     last = cur;
47     cnt[last]=1;
48 }
49
50 int search(string s){
51     int cur=0, i=0, n=(int)s.length();
52     while(i<n){
53         if(!st[cur].next.count(s[i])) return -1;
54         cur=st[cur].next[s[i]];
55         i++;
56     }
57     //sumar 2 si se quiere 1 indexado
58     return st[cur].firstposition-n+1;
59 }
60
61 void dfs(int cur){
62     cntPaths[cur]=1;
63     for(auto [x, y]:st[cur].next){
64         if(cntPaths[y]==0) dfs(y);
65         cntPaths[cur]+=cntPaths[y];
66     }
67 }
68
69 void countPaths(){
70     dfs(0);
71 }
```

```

72
73 string kthSmallestDistinct(ll k){
74     string s="";
75     int cur=0;
76     while(k>0){
77         for(auto [c, y]:st[cur].next){
78             if(k>cntPaths[y]) k-=cntPaths[y];
79             else{
80                 k--;
81                 s+=c;
82                 cur=y;
83                 break;
84             }
85         }
86     }
87     return s;
88 }

89 void countOccurrences(){
90     vector<pair<int, int>> a;
91     for(int i=sz-1;i>0;i--){
92         a.push_back({st[i].len, i});
93     }
94     sort(a.begin(), a.end());
95     for(int i=sz-2;i>=0;i--){
96         cnt[st[a[i].second].link]+=cnt[a[i].second];
97     }
98 }
99 }

100 void dfs1(int cur){
101     for(auto [x, y]:st[cur].next){
102         if(cntSum[y]==cnt[y]) dfs1(y);
103         cntSum[cur]+=cntSum[y];
104     }
105 }
106 }

107 void countSumOccurrences(){
108     for(int i=0;i<sz;i++){
109         cntSum[i]=cnt[i];
110     }
111     dfs1(0);
112 }
113 }

114

115 string kthSmallest(ll k){
116     string s="";
117     int cur=0;
118     while(k>0){
119         for(auto [c, y]:st[cur].next){
120             if(k>cntSum[y]) k-=cntSum[y];
121             else{
122                 k-=cnt[y];
123                 s+=c;
124                 cur=y;
125                 break;
126             }
127         }
128     }
129     return s;
130 }

131 //longest common substring
132 //build automaton for s first
133 string lcs (string S, string T) {
134     int v = 0, l = 0, best = 0, bestpos = 0;
135     for (int i = 0; i < T.size(); i++) {
136         while (v && !st[v].next.count(T[i])) {
137             v = st[v].link ;
138             l = st[v].len;
139         }
140         if (st[v].next.count(T[i])) {
141             v = st [v].next[T[i]];
142             l++;
143         }
144         if (l > best) {
145             best = l;
146             bestpos = i;
147         }
148     }
149     return T.substr(bestpos - best + 1, best);
150 }

151 int main(){
152     ios_base::sync_with_stdio(false); cin.tie(NULL);
153     string s; cin >> s;
154     initSuffixAutomaton();
155     for(char c:s){
156

```

```

158     insertChar(c);
159 }
160 }
```

10 DP

10.1 Digit DP

```

1 vector<int> num;
2 ll DP[20][20][2][2];
3
4 ll g(int pos, int last, int f, int z){
5
6     if(pos == num.size()){
7         return 1;
8     }
9
10    if(DP[pos][last][f][z] != -1) return DP[pos][last][f][z];
11    ll res = 0;
12
13    int l=(f ? 9 : num[pos]);
14
15    for(int dgt = 0; dgt<=l; dgt++){
16        if(dgt==last && !(dgt==0 && z==1)) continue;
17        int nf = f;
18        if(f == 0 && dgt < 1) nf = 1;
19        if(z && !dgt) res+=g(pos+1, dgt, nf, 1);
20        else res += g(pos+1, dgt, nf, 0);
21    }
22    DP[pos][last][f][z]=res;
23    return res;
24 }
25
26 ll solve(ll x){
27     num.clear();
28     if(x== -1) return 0;
29     memset(DP, -1, sizeof(DP));
30     while(x>0){
31         num.pb(x%10);
32         x/=10;
33     }
34     reverse(all(num));
35     return g(0, 0, 0, 1);
36 }
```

```

36 }
```

10.2 Convex Hull Trick Deque

```

1 // needs fixing
2
3 struct line {
4     ll a, b;
5     line(ll A, ll B) : a(A), b(B) {}
6     double intersect(const line &line1) const {
7         return 1.0 * (line1.b - b) / (a - line1.a);
8     }
9     ll eval(ll x) {
10         return a * x + b;
11     }
12 };
13
14 // this finds the minimum and slope in increasing
15 deque<line> l[p+1];
16 l[0].push_front({-1, -s[1]});
17 for(int i=1;i<=m;i++){
18     for(int j=p;j>0;j--){
19         if(j>i) continue;
20         while((int)l[j-1].size()>=2 && l[j-1].back().eval(a[i])>=l[j-1][(int)
21             l[j-1].size()-2].eval(a[i])){
22             l[j-1].pop_back();
23         }
24         dp[i][j]=l[j-1].back().eval(a[i])+(a[i]*(i))+s[i];
25         line cur(-i-1, dp[i][j]-s[i+1]);
26         while((int)l[j].size()>=2 && cur.intersect(l[j][1])<=l[j][0].
27             intersect(l[j][1])){
28             l[j].pop_front();
29         }
30     }
31 }
```

10.3 Longest Common Subsequence(LCS)

```

1 string s, t; cin >> s >> t;
2 int n=s.length(), m=t.length();
3 int dp[n+1][m+1];
4 memset(dp, 0, sizeof(dp));
5 for(int i=1;i<=n;i++){
5 }
```

```

6     for(int j=1;j<=m;j++){
7         dp[i][j]=max(dp[i-1][j], dp[i][j-1]);
8         if(s[i-1]==t[j-1]){
9             dp[i][j]=dp[i-1][j-1]+1;
10        }
11    }
12 }
13 int i=n, j=m;
14 string ans="";
15 while(i && j){
16     if(s[i-1]==t[j-1]){
17         ans+=s[i-1];
18         i--; j--;
19     }
20     else if(dp[i][j-1]>=dp[i-1][j]){
21         j--;
22     }
23     else{
24         i--;
25     }
26 }
27 reverse(all(ans));
28 cout << ans << endl;

```

10.4 Edit Distance

```
1 string s, t; cin >> s >> t;
2     int n=s.length(), m=t.length();
3     for (int i=0;i<=n;i++){
4         fill(dp[i], dp[i]+m+1, 1e9);
5     }
6     dp[0][0]=0;
7     for (int i=0;i<=n;i++){
8         for (int j=0;j<=m;j++){
9             if(j){
10                 dp[i][j]=min(dp[i][j], dp[i][j-1]+1);
11             }
12             if(i){
13                 dp[i][j]=min(dp[i][j], dp[i-1][j]+1);
14             }
15             if(i && j){
16                 int a=(s[i-1]!=t[j-1] ? 1:0);
17                 dp[i][j]=min(dp[i][j], dp[i-1][j-1]+a);
18             }
19         }
20     }
21 }
```

18 | }
19 }
20 }

10.5 Longest Increasing Subsequence(LIS)

```
1 vector <int> dp;
2 for (int i=0;i<n;i++){
3     auto it=lower_bound(dp.begin(), dp.end(), v[i]);
4     if(it==dp.end()){
5         dp.push_back(v[i]);
6     }
7     else{
8         int pos=it-dp.begin();
9         dp[pos]=v[i];
10    }
11 }
12 cout << dp.size() << endl;
```

11 Geometry

11.1 Geometry Primitives

```
1 struct point2d {
2     ftype x, y;
3     point2d() {}
4     point2d(ftype x, ftype y): x(x), y(y) {}
5     point2d& operator+=(const point2d &t) {
6         x += t.x;
7         y += t.y;
8         return *this;
9     }
10    point2d& operator-=(const point2d &t) {
11        x -= t.x;
12        y -= t.y;
13        return *this;
14    }
15    point2d& operator*=(ftype t) {
16        x *= t;
17        y *= t;
18        return *this;
19    }
20    point2d& operator/=(ftype t) {
```

```

21     x /= t;
22     y /= t;
23     return *this;
24 }
25 point2d operator+(const point2d &t) const {return point2d(*this) +=
26     t;}
27 point2d operator-(const point2d &t) const {return point2d(*this) ==
28     t;}
29 point2d operator*(ftype t) const {return point2d(*this) *= t;}
30 point2d operator/(ftype t) const {return point2d(*this) /= t;}
31 ftype dot(point2d a, point2d b) {return a.x * b.x + a.y * b.y;}
32 ftype norm(point2d a) {return dot(a, a);}
33 double abs(point2d a) {return sqrt(norm(a));}
34 double proj(point2d a, point2d b) {return dot(a, b) / abs(b);}
35 double angle(point2d a, point2d b) {return acos(dot(a, b) / abs(a) /
36     abs(b));}

//intersect between two lines
37 point2d intersect(point2d a1, point2d d1, point2d a2, point2d d2) {
38     return a1 + cross(a2 - a1, d2) / cross(d1, d2) * d1;
39 }
40 point2d operator*(ftype a, point2d b) { return b * a;}
41
42 struct point3d {
43     ftype x, y, z;
44     point3d() {}
45     point3d(ftype x, ftype y, ftype z): x(x), y(y), z(z) {}
46     point3d& operator+=(const point3d &t) {
47         x += t.x;
48         y += t.y;
49         z += t.z;
50         return *this;
51     }
52     point3d& operator-=(const point3d &t) {
53         x -= t.x;
54         y -= t.y;
55         z -= t.z;
56         return *this;
57     }
58     point3d& operator*=(ftype t) {
59         x *= t;
60         y *= t;

```

```

61         z *= t;
62         return *this;
63     }
64     point3d& operator/=(ftype t) {
65         x /= t;
66         y /= t;
67         z /= t;
68         return *this;
69     }
70     point3d operator+(const point3d &t) const { return point3d(*this) +=
71         t;}
72     point3d operator-(const point3d &t) const {return point3d(*this) ==
73         t;}
74     point3d operator*(ftype t) const {return point3d(*this) *= t;}
75     point3d operator/(ftype t) const {return point3d(*this) /= t;}
76     ftype dot(point3d a, point3d b) {return a.x * b.x + a.y * b.y + a.z *
77         b.z;}
78     ftype norm(point3d a) { return dot(a, a);}
79     double abs(point3d a) {return sqrt(norm(a));}
80     double proj(point3d a, point3d b) {return dot(a, b) / abs(b);}
81     double angle(point3d a, point3d b) {return acos(dot(a, b) / abs(a) /
82         abs(b));}

//cross product
83 point3d cross(point3d a, point3d b) {
84     return point3d(a.y * b.z - a.z * b.y,
85                     a.z * b.x - a.x * b.z,
86                     a.x * b.y - a.y * b.x);
87 }
88 //intersect between three planes
89 point3d intersect(point3d a1, point3d n1, point3d a2, point3d n2,
90     point3d a3, point3d n3) {
91     point3d x(n1.x, n2.x, n3.x);
92     point3d y(n1.y, n2.y, n3.y);
93     point3d z(n1.z, n2.z, n3.z);
94     point3d d(dot(a1, n1), dot(a2, n2), dot(a3, n3));
95     return point3d(triple(d, y, z),
96                     triple(x, d, z),
97                     triple(x, y, d)) / triple(n1, n2, n3);
98 }

```

```
99 | point3d operator*(ftype a, point3d b) {return b * a; }
```

11.2 Area of simple Polygon

```
1 // sum ((px-qx)*(py+qy))/2
2 double area(const vector<point>& fig) {
3     double res = 0;
4     for (unsigned i = 0; i < fig.size(); i++) {
5         point p = i ? fig[i - 1] : fig.back();
6         point q = fig[i];
7         res += (p.x - q.x) * (p.y + q.y);
8     }
9     return fabs(res) / 2;
10 }
```

11.3 Point Inside Convex Polygon

```
1 void normalize(vector<pto> &pt){//delete collinear points first!
2     //this makes it clockwise:
3     if(pt[2].left(pt[0], pt[1])) reverse(pt.begin(), pt.end());
4     int n=sz(pt), pi=0;
5     forn(i, n)
6         if(pt[i].x<pt[pi].x || (pt[i].x==pt[pi].x && pt[i].y<pt[pi].y))
7             pi=i;
8     vector<pto> shift(n); //puts pi as first point
9     forn(i, n) shift[i]=pt[(pi+i)%n];
10    pt.swap(shift);
11 }
12 bool inPolygon(pto p, const vector<pto> &pt){
13     //call normalize first!
14     if(p.left(pt[0], pt[1]) || p.left(pt[sz(pt)-1], pt[0])) return false;
15     int a=1, b=sz(pt)-1;
16     while(b-a>1){
17         int c=(a+b)/2;
18         if(!p.left(pt[0], pt[c])) a=c;
19         else b=c;
20     }
21     return !p.left(pt[a], pt[a+1]);
22 }
```

11.4 Nearest Pair Of Points

```
1 struct pt {
2     int x, y, id;
```

```
3 };
4
5 struct cmp_x {
6     bool operator()(const pt & a, const pt & b) const {
7         return a.x < b.x || (a.x == b.x && a.y < b.y);
8     }
9 };
10
11 struct cmp_y {
12     bool operator()(const pt & a, const pt & b) const {
13         return a.y < b.y;
14     }
15 };
16
17 int n;
18 vector<pt> a;
19
20 double mindist;
21 pair<int, int> best_pair;
22
23 void upd_ans(const pt & a, const pt & b) {
24     double dist = sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y)
25                         );
26     if (dist < mindist) {
27         mindist = dist;
28         best_pair = {a.id, b.id};
29     }
30
31 vector<pt> t;
32
33 void rec(int l, int r) {
34     if (r - l <= 3) {
35         for (int i = l; i < r; ++i) {
36             for (int j = i + 1; j < r; ++j) {
37                 upd_ans(a[i], a[j]);
38             }
39         }
40         sort(a.begin() + l, a.begin() + r, cmp_y());
41         return;
42     }
43
44     int m = (l + r) >> 1;
```

```

45     int midx = a[m].x;
46     rec(l, m);
47     rec(m, r);
48
49     merge(a.begin() + l, a.begin() + m, a.begin() + m, a.begin() + r, t.
50           begin(), cmp_y());
51     copy(t.begin(), t.begin() + r - l, a.begin() + l);
52
53     int tsz = 0;
54     for (int i = l; i < r; ++i) {
55         if (abs(a[i].x - midx) < mindist) {
56             for (int j = tsz - 1; j >= 0 && a[i].y - t[j].y < mindist;
57                 --j)
58                 upd_ans(a[i], t[j]);
59             t[tsz++] = a[i];
60         }
61     }
62
63     t.resize(n);
64     sort(a.begin(), a.end(), cmp_x());
65     mindist = 1E20;
66     rec(0, n);

```

11.5 ConvexHull

```

1 struct pt {
2     double x, y;
3     bool operator == (pt const& t) const {
4         return x == t.x && y == t.y;
5     }
6 };
7
8 int orientation(pt a, pt b, pt c) {
9     double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
10    if (v < 0) return -1; // clockwise
11    if (v > 0) return +1; // counter-clockwise
12    return 0;
13 }
14
15 bool cw(pt a, pt b, pt c, bool include_collinear) {
16     int o = orientation(a, b, c);
17     return o < 0 || (include_collinear && o == 0);

```

```

18 }
19 bool collinear(pt a, pt b, pt c) { return orientation(a, b, c) == 0; }
20
21 void convex_hull(vector<pt>& a, bool include_collinear = false) {
22     pt p0 = *min_element(a.begin(), a.end(), [](pt a, pt b) {
23         return make_pair(a.y, a.x) < make_pair(b.y, b.x);
24     });
25     sort(a.begin(), a.end(), [&p0](const pt& a, const pt& b) {
26         int o = orientation(p0, a, b);
27         if (o == 0)
28             return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
29             < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
30         return o < 0;
31     });
32     if (include_collinear) {
33         int i = (int)a.size()-1;
34         while (i >= 0 && collinear(p0, a[i], a.back())) i--;
35         reverse(a.begin()+i+1, a.end());
36     }
37
38     vector<pt> st;
39     for (int i = 0; i < (int)a.size(); i++) {
40         while (st.size() > 1 && !cw(st[st.size()-2], st.back(), a[i],
41             include_collinear))
42             st.pop_back();
43         st.push_back(a[i]);
44     }
45     if (include_collinear == false && st.size() == 2 && st[0] == st[1])
46         st.pop_back();
47
48     a = st;
49 }

```

11.6 Rotating Calipers

```

1 #define ll long long
2 #define pii array<int,2>
3 #define nx(i) (i+1)%n
4 #define pv(i) (i-1+n)%n
5
6 struct Point {
7     ll x,y;

```

```

8
9     Point operator+(const Point &p) {
10        return {x + p.x, y + p.y};
11    }
12
13    Point operator-(const Point &p) {
14        return {x - p.x, y - p.y};
15    }
16};

17
18 ll cross(Point p1, Point p2) {
19     return p1.x * p2.y - p1.y * p2.x;
20 }

21
22 int sign(ll num) {
23     if (num < 0) return -1;
24     else if (num == 0) return 0;
25     else return 1;
26 }

27
28 vector<pii> all_anti_podal(int n, vector<Point> &p) {
29     int p1 = 0, p2 = 0; // two "pointers"
30     vector<pii> result;
31
32     // parallel edges shouldn't be visited twice
33     vector<bool> vis(n, false);
34
35     for (; p1 < n; p1++) {
36         // the edge that we are going to consider in this iteration
37         // the datatype is Point, but it acts as a vector
38         Point base = p[nx(p1)] - p[p1];
39
40         // the last condition makes sure that the cross products don't
41         // have the same sign
42         while (p2 == p1 || p2 == nx(p1) || sign(cross(base, p[nx(p2)] -
43             p[p2])) == sign(cross(base, p[p2] - p[nx(p2)]))) {
44             p2 = nx(p2);
45         }
46
47         if (vis[p1]) continue;
48         vis[p1] = true;
49
50         result.push_back({p1, p2});
51         result.push_back({nx(p1), p2});
52
53         // if both edges from p1 and p2 are parallel to each other
54         if (cross(base, p[nx(p2)] - p[p2]) == 0) {
55             result.push_back({p1, nx(p2)});
56             result.push_back({nx(p1), nx(p2)});
57             vis[p2] = true;
58         }
59
60     }
61 }

62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
737
738
739
739
740
741
742
743
744
745
745
746
747
747
748
749
749
750
751
752
753
754
755
755
756
757
757
758
759
759
760
761
762
763
764
764
765
766
766
767
767
768
768
769
769
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1
```

12.3 Int Ternary Search

```

1 int lo = -1, hi = n;
2 while (hi - lo > 1){
3     int mid = (hi + lo)>>1;
4     if (f(mid) > f(mid + 1))
5         hi = mid;
6     else
7         lo = mid;
8 }
9 //lo + 1 is the answer

```

12.4 Ternary Search

```

1 double ternary_search(double l, double r) {
2     double eps = 1e-9;           //set the error limit here
3     while (r - l > eps) {
4         double m1 = l + (r - l) / 3;
5         double m2 = r - (r - l) / 3;
6         double f1 = f(m1);        //evaluates the function at m1
7         double f2 = f(m2);        //evaluates the function at m2
8         if (f1 < f2)
9             l = m1;
10        else
11            r = m2;
12    }
13    return f(l);                //return the maximum of f(x) in [l,
14                                r]
}

```

12.5 Parallel Binary Search

```

1 int lo[maxn], hi[maxn];
2 vector<int> tocheck[maxn];
3
4 bool c=true;
5 while(c){
6     c=false;
7     //initialize changes of structure to 0
8
9     for(int i=0;i<k;i++){
10        if(low[i]!=high[i]){
11            check[(low[i]+high[i])/2].pb(i);
12        }
}

```

```

13 }
14
15 for(int i=0;i<m;i++){
16     // apply change for ith query
17
18     while(check[i].size()){
19         c=true;
20         int x=check[i].back();
21         check[i].pop_back();
22
23         if(operationToCheck){
24             high[x]=i;
25         }
26         else{
27             low[x]=i+1;
28         }
29     }
30 }
31

```

12.6 Next and Prev Smaller/Greater

```

1 vector<int> nextSmaller(vector<int> a, int n){
2     stack<int> s;
3     vector<int> res(n, n);
4     for(int i=0;i<n;i++){
5         while(s.size() && a[s.top()]>a[i]){
6             res[s.top()]=i;
7             s.pop();
8         }
9         s.push(i);
10    }
11    return res;
12 }
13
14 vector<int> prevSmaller(vector<int> a, int n){
15     stack<int> s;
16     vector<int> res(n, -1);
17     for(int i=n-1;i>=0;i--){
18         while(s.size() && a[s.top()]>a[i]){
19             res[s.top()]=i;
20             s.pop();
21         }
}

```

```

22     s.push(i);
23 }
24 return res;
25 }
```

12.7 2D Prefix Sum

```

1 int b[MAXN][MAXN];
2 int a[MAXN][MAXN];
3
4 for (int i = 1; i <= N; i++) {
5     for (int j = 1; j <= N; j++) {
6         b[i][j] = a[i][j] + b[i - 1][j] +
7                     b[i][j - 1] - b[i - 1][j - 1];
8     }
9 }
10
11 for (int q = 0; q < Q; q++) {
12     int from_row, to_row, from_col, to_col;
13     cin >> from_row >> from_col >> to_row >> to_col;
14     cout << b[to_row][to_col] - b[from_row - 1][to_col] -
15         b[to_row][from_col - 1] +
16         b[from_row - 1][from_col - 1]
17         << '\n';
18 }
```

12.8 Day of Week

```

1 int dayOfWeek(int d, int m, lli y){
2     if(m == 1 || m == 2){
3         m += 12;
4         --y;
5     }
6     int k = y % 100;
7     lli j = y / 100;
8     return (d + 13*(m+1)/5 + k + k/4 + j/4 + 5*j) % 7;
9 }
```

12.9 Iterating over all subsets of mask

```

1 for (int mk = 0; mk < (1 << k); mk++) {
2     Ap[mk] = 0;
3     for (int s = mk;; s = (s - 1) & mk) {
4         Ap[mk] += A[s];
```

```

5         if (!s)
6             break;
7     }
8 }
```

12.10 Int 128

```

1 //cout for __int128
2 ostream &operator<<(ostream &os, const __int128 & value){
3     char buffer[64];
4     char *pos = end(buffer) - 1;
5     *pos = '\0';
6     __int128 tmp = value < 0 ? -value : value;
7     do{
8         --pos;
9         *pos = tmp % 10 + '0';
10        tmp /= 10;
11    }while(tmp != 0);
12    if(value < 0){
13        --pos;
14        *pos = '-';
15    }
16    return os << pos;
17 }
18
19 //cin for __int128
20 istream &operator>>(istream &is, __int128 & value){
21     char buffer[64];
22     is >> buffer;
23     char *pos = begin(buffer);
24     int sgn = 1;
25     value = 0;
26     if(*pos == '-'){
27         sgn = -1;
28         ++pos;
29     }else if(*pos == '+'){
30         ++pos;
31     }
32     while(*pos != '\0'){
33         value = (value << 3) + (value << 1) + (*pos - '0');
34         ++pos;
35     }
36     value *= sgn;
```

```

37     return is;
38 }
39
40
41 ll mult(__int128 a, __int128 b){ return ((a*1LL*b)%MOD + MOD)%MOD; }
```

12.11 XOR Basis

```

1 int basis[d]; // basis[i] keeps the mask of the vector whose f value is
2   i
3
4 int sz; // Current size of the basis
5
6 void insertVector(int mask) {
7   //turn for around if u want max xor
8   for (int i = 0; i < d; i++) {
9     if ((mask & 1 << i) == 0) continue; // continue if i != f(mask)
10
11   if (!basis[i]) { // If there is no basis vector with the i'th bit
12     set, then insert this vector into the basis
13     basis[i] = mask;
14     ++sz;
15
16     return;
17   }
18
19   mask ^= basis[i]; // Otherwise subtract the basis vector from this
20   // vector
21 }
```

12.12 XOR Convolution

```

1 void FWHT (int A[], int k, int inv) {
2   for (int j = 0; j < k; j++)
3     for (int i = 0; i < (1 << k); i++)
4       if (~i & (1 << j)) {
5         int p0 = A[i];
6         int p1 = A[i | (1 << j)];
7
8         A[i] = p0 + p1;
9         A[i | (1 << j)] = p0 - p1;
```

```

11           if (inv) {
12             A[i] /= 2;
13             A[i | (1 << j)] /= 2;
14           }
15         }
16     }
17
18 void XOR_conv (int A[], int B[], int C[], int k) {
19   FWHT(A, k, false);
20   FWHT(B, k, false);
21
22   for (int i = 0; i < (1 << k); i++)
23     C[i] = A[i] * B[i];
24
25   FWHT(A, k, true);
26   FWHT(B, k, true);
27   FWHT(C, k, true);
28 }
```

12.13 GCD Convolution

```

1 vector<int> PrimeEnumerate(int n) {
2   vector<int> P; vector<bool> B(n + 1, 1);
3   for (int i = 2; i <= n; i++) {
4     if (B[i]) P.push_back(i);
5     for (int j : P) { if (i * j > n) break; B[i * j] = 0; if (i % j ==
6       0) break; }
7   }
8   return P;
9 }
10
11 template<typename T>
12 void MultipleZetaTransform(vector<T>& v) {
13   const int n = (int)v.size() - 1;
14   for (int p : PrimeEnumerate(n)) {
15     for (int i = n / p; i; i--)
16       v[i] += v[i * p];
17   }
18 }
19
20 template<typename T>
21 void MultipleMobiusTransform(vector<T>& v) {
```

```

22 const int n = (int)v.size() - 1;
23 for (int p : PrimeEnumerate(n)) {
24     for (int i = 1; i * p <= n; i++)
25         v[i] -= v[i * p];
26 }
27 }

28 template<typename T>
29 vector<T> GCDConvolution(vector<T> A, vector<T> B) {
30     MultipleZetaTransform(A);
31     MultipleZetaTransform(B);
32     for (int i = 0; i < A.size(); i++) A[i] *= B[i];
33     MultipleMobiusTransform(A);
34     return A;
35 }
36

```

12.14 LCM Convolution

```

1 /* Linear Sieve, O(n) */
2 vector<int> PrimeEnumerate(int n) {
3     vector<int> P; vector<bool> B(n + 1, 1);
4     for (int i = 2; i <= n; i++) {
5         if (B[i]) P.push_back(i);
6         for (int j : P) { if (i * j > n) break; B[i * j] = 0; if (i % j ==
7             0) break; }
8     }
9     return P;
10 }

11 template<typename T>
12 void DivisorZetaTransform(vector<T>& v) {
13     const int n = (int)v.size() - 1;
14     for (int p : PrimeEnumerate(n)) {
15         for (int i = 1; i * p <= n; i++)
16             v[i * p] += v[i];
17     }
18 }

19 template<typename T>
20 void DivisorMobiusTransform(vector<T>& v) {
21     const int n = (int)v.size() - 1;
22     for (int p : PrimeEnumerate(n)) {
23         for (int i = n / p; i; i--)
24

```

```

25         v[i * p] -= v[i];
26     }
27 }

28

29 template<typename T>
30 vector<T> LCMConvolution(vector<T> A, vector<T> B) {
31     DivisorZetaTransform(A);
32     DivisorZetaTransform(B);
33     for (int i = 0; i < A.size(); i++) A[i] *= B[i];
34     DivisorMobiusTransform(A);
35     return A;
36 }
37

```

12.15 OR, AND Convolution

For OR convolution run subset SOS DP for arrays, multiply then inverse SOS DP. For AND convolution same idea but superset.

12.16 Mo's Algorithm

```

1 ll n, q;
2 ll cur=0;
3 ll cnt[1000005];
4 ll answers[200500];
5 ll BLOCK_SIZE;
6 ll arr[200500];

7 pair< pair<ll, ll>, ll> queries[200500];
8
9
10 inline bool cmp(const pair< pair<ll, ll>, ll> &x, const pair< pair<ll,
11     ll>, ll> &y) {
12     ll block_x = x.first.first / BLOCK_SIZE;
13     ll block_y = y.first.first / BLOCK_SIZE;
14     if(block_x != block_y)
15         return block_x < block_y;
16     return x.first.second < y.first.second;
17 }
18
19 int main(){
20     cin >> n >> q;
21     BLOCK_SIZE =(ll)(sqrt(n));
22     for(int i = 0; i < n; i++)
23

```

```

22     cin >> arr[i];
23
24     for(int i = 0; i < q; i++) {
25         cin >> queries[i].first.first >> queries[i].first.second;
26         queries[i].second = i;
27     }
28
29     sort(queries, queries + q, cmp);
30
31     ll l = 0, r = -1;
32
33     for(int i = 0; i < q; i++) {
34         ll left = queries[i].first.first;
35         left--;
36         ll right = queries[i].first.second;
37         right--;
38
39         while(r < right) {
40             //operations
41             r++;
42         }
43         while(r > right) {
44             //operations
45             r--;
46         }
47
48         while(l < left) {
49             //operations
50             l++;
51         }
52         while(l > left) {
53             //operations
54             l--;
55         }
56         answers[queries[i].second] = cur;
57     }
58 }
```

12.17 Matrix Exponentiation

```

1 | vector<vector<ll>> mul(vector<vector<ll>> a, vector<vector<ll>> b, int n
2 |     ) {
3 |         vector<vector<ll>> res(n, vector<ll>(n, 0));
4 | }
```

```

3 |     for (int i = 0; i < n; i++) {
4 |         for (int j = 0; j < n; j++) {
5 |             for (int k = 0; k < n; k++) {
6 |                 res[i][j] += a[i][k] * b[k][j];
7 |                 res[i][j] %= MOD;
8 |             }
9 |         }
10 |     }
11 |
12 |     return res;
13 | }
14
15 //for fibonacci
16 vector<vll> a(2, vll(2));
17 vector<vll> iden(2, vll(2));
18 a[0][0]=1;
19 a[0][1]=1;
20 a[1][0]=1;
21 a[1][1]=0;
22 iden[0][0]=1;
23 iden[1][1]=1;
24 iden[0][1]=0;
25 iden[1][0]=0;
26 for (;k>0;k/=2){
27     if(k&1) iden=mul(iden, a);
28     a=mul(a, a);
29 }
```

12.18 Sprague-Grundy Theorem

- 1) Get all transitions from state.
- 2) Calculate Grundy value for each independent game and xor them.
- 3) Grundy value of a state is mex of the Grundy values of transitions.
- 4) Zero means losing, anything else means winning state.

13 Stress Testing Scripts

13.1 build.sh

This file should be called before stress.sh or validate.sh. build.sh name.cpp

```

1 | g++ -static -DLOCAL -lm -s -x c++ -Wall -Wextra -O2 -std=c++17 -o $1 $1.
2 |     cpp
```

13.2 stress.sh

Format is stress.sh Awrong Agen Numtests

```

1 #!/usr/bin/env bash
2
3 for ((testNum=0;testNum<$4;testNum++))
4 do
5     ./$3 > input
6     ./$2 < input > outSlow
7     ./$1 < input > outWrong
8     H1='md5sum outWrong'
9     H2='md5sum outSlow'
10    if !(cmp -s "outWrong" "outSlow")
11    then
12        echo "Error_found!"
13        echo "Input:"
14        cat input
15        echo "Wrong_Output:"
16        cat outWrong
17        echo "Slow_Output:"
18        cat outSlow
19        exit
20    fi
21 done
22 echo Passed $4 tests

```

13.3 validate.sh

Format is validate.sh Awrong Validator Agen NumTests

```

1 #!/usr/bin/env bash
2
3 for ((testNum=0;testNum<$4;testNum++))
4 do
5     ./$3 > input
6     ./$1 < input > out
7     cat input out > data
8     ./$2 < data > res
9     result=$(cat res)
10    if [ "${result:0:2}" != "OK" ];
11    then
12        echo "Error_found!"
13        echo "Input:"

```

```

14         cat input
15         echo "Output:"
16         cat out
17         echo "Validator_Result:"
18         cat res
19         exit
20     fi
21 done
22 echo Passed $4 tests

```

14 Useful things

14.1 Sums

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

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

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

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

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

14.2 Catalan numbers

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

$$C_0 = 1, C_{n+1} = \frac{2(2n+1)}{n+2} C_n, C_{n+1} = \sum C_i C_{n-i}$$

$$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \dots$$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested. If prefix is given, number of ways is $\binom{n}{remaining_closed} - \binom{n}{remaining_closed+1}$.
- binary trees with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.

- ways a convex polygon with $n + 2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

14.3 Cayley's formula

Number of labeled trees of n vertices is n^{n-2} . Number of rooted forest of n vertices is $(n+1)^{n-1}$.

14.4 Geometric series

Infinite

$$a + ar + ar^2 + ar^3 + \dots + \sum_{k=0}^{\infty} ar^k$$

$$\text{Sum} = \frac{a}{1-r}$$

Finite

$$a + ar + ar^2 + ar^3 + \dots + \sum_{k=0}^n ar^k$$

$$\text{Sum} = \frac{a(1-r^{n+1})}{1-r}$$

14.5 Estimates For Divisors

$$\sum_{d|n} d = O(n \log \log n).$$

The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200 000 for $n < 1e19$.

14.6 Sum of divisors

$$\sum d|n = \frac{p_1^{e_1+1}-1}{p_1-1} + \frac{p_2^{e_2+1}-1}{p_2-1} + \dots + \frac{p_n^{e_n+1}-1}{p_n-1}$$

14.7 Pythagorean Triplets

The Pythagorean triples are uniquely generated by

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

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

14.8 Derangements

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

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

15 C++ things

- Compiling

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

- Bits library

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

- Fast IO

```
ios_base::sync_with_stdio(false); cin.tie(NULL);
```

- Compiler Optimizations

```
#pragma GCC target("popcnt"): makes bit operations faster
```

```
#pragma GCC optimize("Ofast"): auto-vectorize for loops and optimizes floating points better (assumes associativity and turns off denormals).
```

```
#pragma GCC target("avx2"): can double performance of vectorized code, but causes crashes on old machines.
```

```
#pragma GCC optimize("O3,unroll-loops")
```

```
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
```

- Decimal Printing

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

- Bit Tricks

$x \& -x$ is least bit in x .

$c = x \& -x$, $r = x+c$; $((r^x) >> 2)/c$ | r , next number bigger than x same number of bits set.