

Can Tho University

CTU.NEGATIVEZERO

Contents

1 Contest	2	5.5 Zeckendorf's theorem	11
1.1 C++	2	5.6 Bitwise operation	12
1.2 Debug	2	5.7 Pollard's rho algorithm	12
1.3 Java	2	5.8 Bitset sieve	13
1.4 sublime-build	2	5.9 Block sieve	13
1.5 .bashrc	3	5.10 Combinatorics	13
2 Data structures	3	6 Linear algebra	14
2.1 Sparse table	3	6.1 Gauss elimination	14
2.2 Ordered set	3	7 Geometry	14
2.3 Dsu	3	7.1 Fundamentals	14
2.4 Segment tree	4	7.2 Minimum enclosing circle	17
2.5 Efficient segment tree	4	8 Graph	17
2.6 Persistent lazy segment tree	5	8.1 Bellman-Ford algorithm	17
2.7 Disjoint sparse table	5	8.2 Articulation point and Bridge	17
2.8 Fenwick tree	6	8.3 Strongly connected components	18
2.9 Implicit treap	6	8.4 Topo sort	18
3 Mathematics	7	8.5 K-th smallest shortest path	18
3.1 Trigonometry	7	8.6 Eulerian path	19
3.2 Sums	8	9 Misc.	19
4 String	8	9.1 Ternary search	19
4.1 Prefix function	8	9.2 Dutch flag national problem	20
4.2 Counting occurrences of each prefix	8	9.3 Matrix	20
4.3 Knuth–Morris–Pratt algorithm	8		
4.4 Suffix array	9		
4.5 Manacher's algorithm	9		
4.6 Trie	9		
4.7 Hashing	10		
5 Number Theory	10		
5.1 Euler's totient function	10		
5.2 Mobius function	11		
5.3 Primes	11		
5.4 Wilson's theorem	11		

1 Contest

1.1 C++

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #ifdef LOCAL
5 #include "cp/debug.h"
6 #else
7 #define debug(...)
8 #endif
9
10 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
11
12 const int MOD = (int) 1e9 + 7;
13 const int INF = 0x3f3f3f3f;
14
15 int main() {
16     ios::sync_with_stdio(false); cin.tie(nullptr);
17     // freopen("input.txt", "r", stdin);
18     // freopen("output.txt", "w", stdout);
19
20     return 0;
21 }

```

1.2 Debug

```

1 #define debug(...) { string _s = #__VA_ARGS__; replace(begin(_s), end(_s),
    ',', ' '); stringstream _ss(_s); istream_iterator<string> _it(_ss);
    out_error(_it, __VA_ARGS__); }
2
3 void out_error(istream_iterator<string> it) { cerr << '\n'; }
4
5 template<typename T, typename ...Args>
6 void out_error(istream_iterator<string> it, T a, Args... args) {
7     cerr << " [" << *it << " = " << a << "]" ";
8     out_error(++it, args...);
9 }
10
11 template<typename T, typename G> ostream& operator<<(ostream &os, const
    pair<T, G> &p) {
12     return os << "(" << p.first << ", " << p.second << ")";
13 }
14
15 template<class Con, class = decltype(begin(declval<Con>()))>
16 typename enable_if<!is_same<Con, string>::value, ostream&>::type
17 operator<<(ostream& os, const Con& container) {
18     os << "{";
19     for (auto it = container.begin(); it != container.end(); ++it)
20         os << (it == container.begin() ? "" : ", ") << *it;
21     return os << "}";
22 }

```

1.3 Java

```

1 import java.io.BufferedReader;
2 import java.util.StringTokenizer;
3 import java.io.IOException;
4 import java.io.InputStreamReader;
5 import java.io.PrintWriter;
6 import java.util.ArrayList;
7 import java.util.Arrays;
8 import java.util.Collections;
9 import java.util.Random;
10
11 public class Main {
12     public static void main(String[] args) {
13         FastScanner fs = new FastScanner();
14         PrintWriter out = new PrintWriter(System.out);
15         int n = fs.nextInt();
16         out.println(n);
17         out.close(); // don't forget this line.
18     }
19     static class FastScanner {
20         BufferedReader br;
21         StringTokenizer st;
22         public FastScanner() {
23             br = new BufferedReader(new InputStreamReader(System.in));
24             st = null;
25         }
26         public String next() {
27             while (st == null || st.hasMoreTokens() == false) {
28                 try {
29                     st = new StringTokenizer(br.readLine());
30                 }
31                 catch (IOException e) {
32                     throw new RuntimeException(e);
33                 }
34             }
35             return st.nextToken();
36         }
37
38         public int nextInt() {
39             return Integer.parseInt(next());
40         }
41
42         public long nextLong() {
43             return Long.parseLong(next());
44         }
45
46         public double nextDouble() {
47             return Double.parseDouble(next());
48         }
49     }
50 }

```

1.4 sublime-build

```

1 {

```

```

2   "cmd": ["g++", "-std=c++17", "-fmax-errors=5", "-DLOCAL", "-Wall",
   "-Wextra", "-o", "${file_path}/${file_base_name}.out", "${file}"],
3   "file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)?(?:.*)$",
4   "working_dir": "${file_path}",
5   "selector": "source.cpp, source.c++"
6 }

```

1.5 .bashrc

```

1 alias cpp='g++ -std=c++17 -fmax-errors=5 -DLOCAL -Wall -Wextra'
2
3 #Stress-testing
4 function test {
5     SOL=$1
6     CHECKER=$2
7     for i in {1..100};
8     do
9         ./gen.out > in && ./"$CHECKER.out" < in > ans && ./"$SOL.out" < in >
10        out && diff -Z out ans && echo "Test $i passed!!" || break;
11    done
12 }

```

2 Data structures

2.1 Sparse table

```

1 int st[MAXN][K + 1];
2 for (int i = 0; i < N; i++) {
3     st[i][0] = f(array[i]);
4 }
5 for (int j = 1; j <= K; j++) {
6     for (int i = 0; i + (1 << j) <= N; i++) {
7         st[i][j] = f(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
8     }
9 }
10 // Range Minimum Queries.
11 int lg[MAXN + 1];
12 lg[1] = 0;
13 for (int i = 2; i <= MAXN; i++) {
14     lg[i] = lg[i / 2] + 1;
15 }
16 int j = lg[R - L + 1];
17 int minimum = min(st[L][j], st[R - (1 << j) + 1][j]);
18 // Range Sum Queries.
19 long long sum = 0;
20 for (int j = K; j >= 0; j--) {
21     if ((1 << j) <= R - L + 1) {
22         sum += st[L][j];
23         L += 1 << j;
24     }
25 }

```

2.2 Ordered set

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>

```

```

3 using namespace __gnu_pbds;
4
5 template<typename key_type>
6 using set_t = tree<key_type, null_type, less<key_type>, rb_tree_tag,
7     tree_order_statistics_node_update>;
8
9 void example() {
10     vector<int> nums = {1, 2, 3, 5, 10};
11     set_t<int> st(nums.begin(), nums.end());
12
13     cout << *st.find_by_order(0) << '\n'; // 1
14     assert(st.find_by_order(-INF) == st.end());
15     assert(st.find_by_order(INF) == st.end());
16
17     cout << st.order_of_key(2) << '\n'; // 1
18     cout << st.order_of_key(4) << '\n'; // 3
19     cout << st.order_of_key(9) << '\n'; // 4
20     cout << st.order_of_key(-INF) << '\n'; // 0
21     cout << st.order_of_key(INF) << '\n'; // 5
22 }

```

2.3 Dsu

```

1 struct Dsu {
2     int n;
3     vector<int> par, sz;
4     Dsu(int _n) : n(_n) {
5         sz.resize(n, 1);
6         par.resize(n);
7         iota(par.begin(), par.end(), 0);
8     }
9     int find(int v) {
10         // finding leader/parent of set that contains the element v.
11         // with {path compression optimization}.
12         return (v == par[v] ? v : par[v] = find(par[v]));
13     }
14     bool same(int u, int v) {
15         return find(u) == find(v);
16     }
17     bool unite(int u, int v) {
18         u = find(u); v = find(v);
19         if (u == v) return false;
20         if (sz[u] < sz[v]) swap(u, v);
21         par[v] = u;
22         sz[u] += sz[v];
23         return true;
24     }
25     vector<vector<int>> groups() {
26         // returns the list of the "list of the vertices in a connected
27         // component".
28         vector<int> leader(n);
29         for (int i = 0; i < n; ++i) {
30             leader[i] = find(i);
31         }
32     }
33 }

```

```

31     vector<int> id(n, -1);
32     int count = 0;
33     for (int i = 0; i < n; ++i) {
34         if (id[leader[i]] == -1) {
35             id[leader[i]] = count++;
36         }
37     }
38     vector<vector<int>> result(count);
39     for (int i = 0; i < n; ++i) {
40         result[id[leader[i]]].push_back(i);
41     }
42     return result;
43 }
44 };

```

2.4 Segment tree

```

1  /**
2   * Description: A segment tree with range updates and sum queries that
3   * supports three types of operations:
4   * + Increase each value in range [l, r] by x (i.e. a[i] += x).
5   * + Set each value in range [l, r] to x (i.e. a[i] = x).
6   * + Determine the sum of values in range [l, r].
7   */
8  struct SegmentTree {
9      int n;
10     vector<long long> tree, lazy_add, lazy_set;
11     SegmentTree(int _n) : n(_n) {
12         int p = 1;
13         while (p < n) p *= 2;
14         tree.resize(p * 2);
15         lazy_add.resize(p * 2);
16         lazy_set.resize(p * 2);
17     }
18     long long merge(const long long &left, const long long &right) {
19         return left + right;
20     }
21     void build(int id, int l, int r, const vector<int> &arr) {
22         if (l == r) {
23             tree[id] += arr[l];
24             return;
25         }
26         int mid = (l + r) >> 1;
27         build(id * 2, l, mid, arr);
28         build(id * 2 + 1, mid + 1, r, arr);
29         tree[id] = merge(tree[id * 2], tree[id * 2 + 1]);
30     }
31     void push(int id, int l, int r) {
32         if (lazy_set[id] == 0 && lazy_add[id] == 0) return;
33         int mid = (l + r) >> 1;
34         for (int child : {id * 2, id * 2 + 1}) {
35             int range = (child == id * 2 ? mid - l + 1 : r - mid);
36             if (lazy_set[id] != 0) {

```

```

37                 lazy_set[child] = lazy_set[id];
38                 tree[child] = range * lazy_set[id];
39             }
40             lazy_add[child] += lazy_add[id];
41             tree[child] += range * lazy_add[id];
42         }
43         lazy_add[id] = lazy_set[id] = 0;
44     }
45 }
46 void update(int id, int l, int r, int u, int v, int amount, bool
47 set_value = false) {
48     if (r < u || l > v) return;
49     if (u <= l && r <= v) {
50         if (set_value) {
51             tree[id] = 1LL * amount * (r - l + 1);
52             lazy_set[id] = amount;
53             lazy_add[id] = 0; // clear all previous updates.
54         }
55         else {
56             tree[id] += 1LL * amount * (r - l + 1);
57             lazy_add[id] += amount;
58         }
59         return;
60     }
61     push(id, l, r);
62     int mid = (l + r) >> 1;
63     update(id * 2, l, mid, u, v, amount, set_value);
64     update(id * 2 + 1, mid + 1, r, u, v, amount, set_value);
65     tree[id] = merge(tree[id * 2], tree[id * 2 + 1]);
66 }
67 long long get(int id, int l, int r, int u, int v) {
68     if (r < u || l > v) return 0;
69     if (u <= l && r <= v) {
70         return tree[id];
71     }
72     push(id, l, r);
73     int mid = (l + r) >> 1;
74     long long left = get(id * 2, l, mid, u, v);
75     long long right = get(id * 2 + 1, mid + 1, r, u, v);
76     return merge(left, right);
77 };

```

2.5 Efficient segment tree

```

1  template<typename T> struct SegmentTree {
2      int n;
3      vector<T> tree;
4      SegmentTree(int _n) : n(_n), tree(2 * n) {}
5      T merge(const T &left, const T &right) {
6          return left + right;
7      }
8      template<typename G>
9      void build(const vector<G> &initial) {

```

```

10     assert((int) initial.size() == n);
11     for (int i = 0; i < n; ++i) {
12         tree[i + n] = initial[i];
13     }
14     for (int i = n - 1; i > 0; --i) {
15         tree[i] = merge(tree[i * 2], tree[i * 2 + 1]);
16     }
17 }
18 void modify(int i, int v) {
19     tree[i += n] = v;
20     for (i /= 2; i > 0; i /= 2) {
21         tree[i] = merge(tree[i * 2], tree[i * 2 + 1]);
22     }
23 }
24 T get_sum(int l, int r) {
25     // sum of elements from l to r - 1.
26     T ret{};
27     for (l += n, r += n; l < r; l /= 2, r /= 2) {
28         if (l & 1) ret = merge(ret, tree[l++]);
29         if (r & 1) ret = merge(ret, tree[--r]);
30     }
31     return ret;
32 }
33 };

```

2.6 Persistent lazy segment tree

```

1 struct Vertex {
2     int l, r;
3     long long val, lazy;
4     bool has_changed = false;
5     Vertex() {}
6     Vertex(int _l, int _r, long long _val, int _lazy = 0) : l(_l), r(_r),
7         val(_val), lazy(_lazy) {}
8 }
9 struct PerSegmentTree {
10     vector<Vertex> tree;
11     vector<int> root;
12     int build(const vector<int> &arr, int l, int r) {
13         if (l == r) {
14             tree.emplace_back(-1, -1, arr[l]);
15             return tree.size() - 1;
16         }
17         int mid = (l + r) / 2;
18         int left = build(arr, l, mid);
19         int right = build(arr, mid + 1, r);
20         tree.emplace_back(left, right, tree[left].val + tree[right].val);
21         return tree.size() - 1;
22     }
23     int add(int x, int l, int r, int u, int v, int amt) {
24         if (l > v || r < u) return x;
25         if (u <= l && r <= v) {
26             tree.emplace_back(tree[x].l, tree[x].r, tree[x].val + 1LL * amt *
27                 (r - l + 1), tree[x].lazy + amt);

```

```

26         tree.back().has_changed = true;
27         return tree.size() - 1;
28     }
29     int mid = (l + r) >> 1;
30     push(x, l, mid, r);
31     int left = add(tree[x].l, l, mid, u, v, amt);
32     int right = add(tree[x].r, mid + 1, r, u, v, amt);
33     tree.emplace_back(left, right, tree[left].val + tree[right].val, 0);
34     return tree.size() - 1;
35 }
36 long long get_sum(int x, int l, int r, int u, int v) {
37     if (r < u || l > v) return 0;
38     if (u <= l && r <= v) return tree[x].val;
39     int mid = (l + r) / 2;
40     push(x, l, mid, r);
41     return get_sum(tree[x].l, l, mid, u, v) + get_sum(tree[x].r, mid + 1,
42         r, u, v);
43 }
44 void push(int x, int l, int mid, int r) {
45     if (!tree[x].has_changed) return;
46     Vertex left = tree[tree[x].l];
47     Vertex right = tree[tree[x].r];
48     tree.emplace_back(left);
49     tree[x].l = tree.size() - 1;
50     tree.emplace_back(right);
51     tree[x].r = tree.size() - 1;
52
53     tree[tree[x].l].val += tree[x].lazy * (mid - l + 1);
54     tree[tree[x].l].lazy += tree[x].lazy;
55
56     tree[tree[x].r].val += tree[x].lazy * (r - mid);
57     tree[tree[x].r].lazy += tree[x].lazy;
58
59     tree[tree[x].l].has_changed = true;
60     tree[tree[x].r].has_changed = true;
61     tree[x].lazy = 0;
62     tree[x].has_changed = false;
63 };

```

2.7 Disjoint sparse table

```

1 /**
2  * Description: range query on a static array.
3  * Time: O(1) per query.
4  * Tested: stress-test.
5  */
6 const int MOD = (int) 1e9 + 7;
7 struct DisjointSparseTable { // product queries.
8     int n, h;
9     vector<vector<int>> dst;
10    vector<int> lg;
11    DisjointSparseTable(int _n) : n(_n) {
12        h = 1; // in case n = 1: h = 0 !!

```

```

13     int p = 1;
14     while (p < n) p *= 2, h++;
15     lg.resize(p); lg[1] = 0;
16     for (int i = 2; i < p; ++i) {
17         lg[i] = 1 + lg[i / 2];
18     }
19     dst.resize(h, vector<int>(n));
20 }
21 void build(const vector<int> &A) {
22     for (int lv = 0; lv < h; ++lv) {
23         int len = (1 << lv);
24         for (int k = 0; k < n; k += len * 2) {
25             int mid = min(k + len, n);
26             dst[lv][mid - 1] = A[mid - 1] % MOD;
27             for (int i = mid - 2; i >= k; --i) {
28                 dst[lv][i] = 1LL * A[i] * dst[lv][i + 1] % MOD;
29             }
30             if (mid == n) break;
31             dst[lv][mid] = A[mid] % MOD;
32             for (int i = mid + 1; i < min(mid + len, n); ++i) {
33                 dst[lv][i] = 1LL * A[i] * dst[lv][i - 1] % MOD;
34             }
35         }
36     }
37 }
38 int get(int l, int r) {
39     if (l == r) {
40         return dst[0][l];
41     }
42     int i = lg[l ^ r];
43     return 1LL * dst[i][l] * dst[i][r] % MOD;
44 }
45 };

```

2.8 Fenwick tree

```

1 using tree_type = long long;
2 struct FenwickTree {
3     int n;
4     vector<tree_type> fenw_coeff, fenw;
5     FenwickTree() {}
6     FenwickTree(int _n) : n(_n) {
7         fenw_coeff.assign(n, 0); // fenwick tree with coefficient (n - i).
8         fenw.assign(n, 0); // normal fenwick tree.
9     }
10    void build(const vector<int> &A) {
11        assert((int) A.size() == n);
12        vector<int> diff(n);
13        diff[0] = A[0];
14        for (int i = 1; i < n; ++i) {
15            diff[i] = A[i] - A[i - 1];
16        }
17        fenw_coeff[0] = (long long) diff[0] * n;
18        fenw[0] = diff[0];

```

```

19        for (int i = 1; i < n; ++i) {
20            fenw_coeff[i] = fenw_coeff[i - 1] + (long long) diff[i] * (n - i);
21            fenw[i] = fenw[i - 1] + diff[i];
22        }
23        for (int i = n - 1; i >= 0; --i) {
24            int j = (i & (i + 1)) - 1;
25            if (j >= 0) {
26                fenw_coeff[i] -= fenw_coeff[j];
27                fenw[i] -= fenw[j];
28            }
29        }
30    }
31    void add(vector<tree_type> &fenw, int i, tree_type val) {
32        while (i < n) {
33            fenw[i] += val;
34            i |= (i + 1);
35        }
36    }
37    tree_type __prefix_sum(vector<tree_type> &fenw, int i) {
38        tree_type res{};
39        while (i >= 0) {
40            res += fenw[i];
41            i = (i & (i + 1)) - 1;
42        }
43        return res;
44    }
45    tree_type prefix_sum(int i) {
46        return __prefix_sum(fenw_coeff, i) - __prefix_sum(fenw, i) * (n - i - 1);
47    }
48    void range_add(int l, int r, tree_type val) {
49        add(fenw_coeff, l, (n - l) * val);
50        add(fenw_coeff, r + 1, (n - r - 1) * (-val));
51        add(fenw, l, val);
52        add(fenw, r + 1, -val);
53    }
54    tree_type range_sum(int l, int r) {
55        return prefix_sum(r) - prefix_sum(l - 1);
56    }
57 };

```

2.9 Implicit treap

```

1 struct Node {
2     int val, prior, cnt;
3     bool rev;
4     Node *left, *right;
5     Node() {}
6     Node(int _val) : val(_val), prior(rng()), cnt(1), rev(false),
7         left(nullptr), right(nullptr) {}
8 };
9 // Binary search tree + min-heap.
10 struct Treap {
11     Node *root;

```

```

11 Treap() : root(nullptr) {}
12 int get_cnt(Node *n) { return n ? n->cnt : 0; }
13 void upd_cnt(Node *&n) {
14     if (n) n->cnt = get_cnt(n->left) + get_cnt(n->right) + 1;
15 }
16 void push_rev(Node *treap) {
17     if (!treap || !treap->rev) return;
18     treap->rev = false;
19     swap(treap->left, treap->right);
20     if (treap->left) treap->left->rev ^= true;
21     if (treap->right) treap->right->rev ^= true;
22 }
23 pair<Node*, Node*> split(Node *treap, int x, int smaller = 0) {
24     if (!treap) return {};
25     push_rev(treap);
26     int idx = smaller + get_cnt(treap->left); // implicit val.
27     if (idx <= x) {
28         auto pr = split(treap->right, x, idx + 1);
29         treap->right = pr.first;
30         upd_cnt(treap);
31         return {treap, pr.second};
32     }
33     else {
34         auto pl = split(treap->left, x, smaller);
35         treap->left = pl.second;
36         upd_cnt(treap);
37         return {pl.first, treap};
38     }
39 }
40 Node* merge(Node *l, Node *r) {
41     push_rev(l); push_rev(r);
42     if (!l || !r) return (l ? l : r);
43     if (l->prior < r->prior) {
44         l->right = merge(l->right, r);
45         upd_cnt(l);
46         return l;
47     }
48     else {
49         r->left = merge(l, r->left);
50         upd_cnt(r);
51         return r;
52     }
53 }
54 void insert(int pos, int val) {
55     if (!root) {
56         root = new Node(val);
57         return;
58     }
59     Node *l, *m, *r;
60     m = new Node(val);
61     tie(l, r) = split(root, pos - 1);
62     root = merge(l, merge(m, r));
63 }

```

```

64 void erase(int pos_l, int pos_r) {
65     Node *l, *m, *r;
66     tie(l, r) = split(root, pos_l - 1);
67     tie(m, r) = split(r, pos_r - pos_l);
68     root = merge(l, r);
69 }
70 void reverse(int pos_l, int pos_r) {
71     Node *l, *m, *r;
72     tie(l, r) = split(root, pos_l - 1);
73     tie(m, r) = split(r, pos_r - pos_l);
74     m->rev ^= true;
75     root = merge(l, merge(m, r));
76 }
77 int query(int pos_l, int pos_r);
78     // returns answer for corresponding types of query.
79 void inorder(Node *n) {
80     if (!n) return;
81     push_rev(n);
82     inorder(n->left);
83     cout << n->val << ' ';
84     inorder(n->right);
85 }
86 void print() {
87     inorder(root);
88     cout << '\n';
89 }
90 };

```

3 Mathematics

3.1 Trigonometry

3.1.1 Sum - difference identities

$$\sin(u \pm v) = \sin(u) \cos(v) \pm \cos(u) \sin(v)$$

$$\cos(u \pm v) = \cos(u) \cos(v) \mp \sin(u) \sin(v)$$

$$\tan(u \pm v) = \frac{\tan(u) \pm \tan(v)}{1 \mp \tan(u) \tan(v)}$$

3.1.2 Sum to product identities

$$\cos(u) + \cos(v) = 2 \cos\left(\frac{u+v}{2}\right) \cos\left(\frac{u-v}{2}\right)$$

$$\cos(u) - \cos(v) = -2 \sin\left(\frac{u+v}{2}\right) \sin\left(\frac{u-v}{2}\right)$$

$$\sin(u) + \sin(v) = 2 \sin\left(\frac{u+v}{2}\right) \cos\left(\frac{u-v}{2}\right)$$

$$\sin(u) - \sin(v) = 2 \cos\left(\frac{u+v}{2}\right) \sin\left(\frac{u-v}{2}\right)$$

3.1.3 Product identities

$$\cos(u) \cos(v) = \frac{1}{2} [\cos(u+v) + \cos(u-v)]$$

$$\sin(u) \sin(v) = -\frac{1}{2} [\cos(u+v) - \cos(u-v)]$$

$$\sin(u) \cos(v) = \frac{1}{2} [\sin(u+v) + \sin(u-v)]$$

3.1.4 Double - triple angle identities

$$\sin(2u) = 2 \sin(u) \cos(u)$$

$$\cos(2u) = 2 \cos^2(u) - 1 = 1 - 2 \sin^2(u)$$

$$\tan(2u) = \frac{2 \tan(u)}{1 - \tan^2(u)}$$

$$\sin(3u) = 3 \sin(u) - 4 \sin^3(u)$$

$$\cos(3u) = 4 \cos^3(u) - 3 \cos(u)$$

$$\tan(3u) = \frac{3 \tan(u) - \tan^3(u)}{1 - 3 \tan^2(u)}$$

3.2 Sums

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

$$c + 2c^2 + \dots + nc^n = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, c \neq 1$$

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

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

$$1^3 + 2^3 + 3^3 + \dots + n^3 = \left(\frac{n(n+1)}{2} \right)^2$$

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

$$1^5 + 2^5 + 3^5 + \dots + n^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12}$$

$$1^6 + 2^6 + 3^6 + \dots + n^6 = \frac{n(n+1)(2n+1)(3n^4+6n^3-3n+1)}{42}$$

$$1^7 + 2^7 + 3^7 + \dots + n^7 = \frac{n^2(n+1)^2(3n^4+6n^3-n^2-4n+2)}{24}$$

4 String

4.1 Prefix function

```

1 /**
2  * Description: The prefix function of a string 's' is defined as an array pi
3  *   of length n,
4  *   where pi[i] is the length of the longest proper prefix of the substring
5  *   s[0..i] which is also a suffix of this substring.
6  * Time complexity: O(|S|).
7  */
8 vector<int> prefix_function(const string &s) {
9     int n = (int) s.length();
10    vector<int> pi(n);
11    pi[0] = 0;
12    for (int i = 1; i < n; ++i) {
13        int j = pi[i - 1]; // try length pi[i - 1] + 1.
14        while (j > 0 && s[j] != s[i]) {
15            j = pi[j - 1];
16        }
17        if (s[j] == s[i]) {
18            pi[i] = j + 1;
19        }
20    }
21    return pi;
22 }
```

4.2 Counting occurrences of each prefix

```

1 vector<int> count_occurrences(const string &s) {
2     vector<int> pi = prefix_function(s);
3     int n = (int) s.size();
4     vector<int> ans(n + 1);
5     for (int i = 0; i < n; ++i) {
6         ans[pi[i]]++;
7     }
8     for (int i = n - 1; i > 0; --i) {
9         ans[pi[i - 1]] += ans[i];
10    }
11    for (int i = 0; i <= n; ++i) {
12        ans[i]++;
13    }
14    return ans;
15    // Input: ABACABA
16    // Output: 4 2 2 1 1 1 1
17 }
```

4.3 Knuth–Morris–Pratt algorithm

```

1 /**
2  * Searching for a substring in a string.
3  * Time complexity: O(N + M).
4  */
5 vector<int> KMP(const string &text, const string &pattern) {
6     int n = (int) text.length();
7     int m = (int) pattern.length();
8     string s = pattern + '$' + text;
9     vector<int> pi = prefix_function(s);
10    vector<int> indices;
```



```

11     for (int i = 0; i < (int) s.length(); ++i) {
12         if (pi[i] == m) {
13             indices.push_back(i - 2 * m);
14         }
15     }
16     return indices;
17 }

```

4.4 Suffix array

```

1 struct SuffixArray {
2     string s;
3     int n, lim;
4     vector<int> sa, lcp, rank;
5     SuffixArray(const string &s, int _lim = 256) : s(_s), n(s.length() + 1),
6         lim(_lim), sa(n), lcp(n), rank(n) {
7         s += '$';
8         build();
9         kasai();
10        sa.erase(sa.begin());
11        lcp.erase(lcp.begin());
12        s.pop_back();
13    }
14    void build() {
15        vector<int> nrank(n), norder(n), cnt(max(n, lim));
16        for (int i = 0; i < n; ++i) {
17            sa[i] = i; rank[i] = s[i];
18        }
19        for (int k = 0, rank_cnt = 0; rank_cnt < n - 1; k = max(1, k * 2),
20            lim = rank_cnt + 1) {
21            // counting sort.
22            for (int i = 0; i < n; ++i) norder[i] = (sa[i] - k + n) % n;
23            for (int i = 0; i < n; ++i) cnt[rank[i]]++;
24            for (int i = 1; i < lim; ++i) cnt[i] += cnt[i - 1];
25            for (int i = n - 1; i >= 0; --i) sa[--cnt[rank[norder[i]]]] =
26                norder[i];
27            rank[sa[0]] = rank_cnt = 0;
28            for (int i = 1; i < n; ++i) {
29                int u = sa[i], v = sa[i - 1];
30                int nu = u + k, nv = v + k;
31                if (nu >= n) nu -= n;
32                if (nv >= n) nv -= n;
33                if (rank[u] != rank[v] || rank[nu] != rank[nv]) ++rank_cnt;
34                nrank[sa[i]] = rank_cnt;
35            }
36            for (int i = 0; i < rank_cnt + 1; ++i) cnt[i] = 0;
37            rank.swap(nrank);
38        }
39    }
40    void kasai() {
41        for (int i = 0; i < n; ++i) rank[sa[i]] = i;
42        for (int i = 0, k = 0; i < n - 1; ++i, k = max(0, k - 1)) {
43            int j = sa[rank[i] - 1];
44            while (s[i + k] == s[j + k]) k++;

```

```

42        lcp[rank[i]] = k;
43    }
44    // Note: lcp[i] = longest common prefix(sa[i - 1], sa[i]).
45 }
46 };

```

4.5 Manacher's algorithm

```

1 /**
2  * Description: for each position, computes d[0][i] = half length of
3  * longest palindrome centered on i (rounded up), d[1][i] = half length of
4  * longest palindrome centered on i and i - 1.
5  * Time complexity: O(N).
6  * Tested: https://judge.yosupo.jp/problem/enumerate\_palindromes,
7  * stress-tested.
8  */
9 array<vector<int>, 2> manacher(const string &s) {
10     int n = (int) s.size();
11     array<vector<int>, 2> d;
12     for (int z = 0; z < 2; ++z) {
13         d[z].resize(n);
14         int l = 0, r = 0;
15         for (int i = 0; i < n; ++i) {
16             int mirror = l + r - i + z;
17             d[z][i] = (i > r ? 0 : min(d[z][mirror], r - i));
18             int L = i - d[z][i] - z, R = i + d[z][i];
19             while (L >= 0 && R < n && s[L] == s[R]) {
20                 d[z][i]++; L--; R++;
21             }
22             if (R > r) {
23                 l = L; r = R;
24             }
25         }
26     }
27     return d;
28 }

```

4.6 Trie

```

1 struct Trie {
2     const static int ALPHABET = 26;
3     const static char minChar = 'a';
4     struct Vertex {
5         int next[ALPHABET];
6         bool leaf;
7         Vertex() {
8             leaf = false;
9             fill(next, next + ALPHABET, -1);
10        }
11    };
12    vector<Vertex> trie;
13    Trie() { trie.emplace_back(); }
14
15    void insert(const string &s) {
16        int i = 0;

```

```

17     for (const char &ch : s) {
18         int j = ch - minChar;
19         if (trie[i].next[j] == -1) {
20             trie[i].next[j] = trie.size();
21             trie.emplace_back();
22         }
23         i = trie[i].next[j];
24     }
25     trie[i].leaf = true;
26 }
27 bool find(const string &s) {
28     int i = 0;
29     for (const char &ch : s) {
30         int j = ch - minChar;
31         if (trie[i].next[j] == -1) {
32             return false;
33         }
34         i = trie[i].next[j];
35     }
36     return (trie[i].leaf ? true : false);
37 }
38 };

```

4.7 Hashing

```

1 struct Hash61 {
2     static const uint64_t MOD = (1LL << 61) - 1;
3     static uint64_t BASE;
4     static vector<uint64_t> pw;
5     uint64_t addmod(uint64_t a, uint64_t b) const {
6         a += b;
7         if (a >= MOD) a -= MOD;
8         return a;
9     }
10    uint64_t submod(uint64_t a, uint64_t b) const {
11        a += MOD - b;
12        if (a >= MOD) a -= MOD;
13        return a;
14    }
15    uint64_t mulmod(uint64_t a, uint64_t b) const {
16        uint64_t low1 = (uint32_t) a, high1 = (a >> 32);
17        uint64_t low2 = (uint32_t) b, high2 = (b >> 32);
18
19        uint64_t low = low1 * low2;
20        uint64_t mid = low1 * high2 + low2 * high1;
21        uint64_t high = high1 * high2;
22
23        uint64_t ret = (low & MOD) + (low >> 61) + (high << 3) + (mid >> 29)
24        + (mid << 35 >> 3) + 1;
25        // ret %= MOD;
26        ret = (ret >> 61) + (ret & MOD);
27        ret = (ret >> 61) + (ret & MOD);
28        return ret - 1;
29    }
30 }

```

```

29 void ensure_pw(int m) {
30     int n = (int) pw.size();
31     if (n >= m) return;
32     pw.resize(m);
33     for (int i = n; i < m; ++i) {
34         pw[i] = mulmod(pw[i - 1], BASE);
35     }
36 }
37
38 vector<uint64_t> pref;
39 int n;
40 template<typename T> Hash61(const T &s) { // strings or arrays.
41     n = (int) s.size();
42     ensure_pw(n);
43     pref.resize(n + 1);
44     pref[0] = 0;
45     for (int i = 0; i < n; ++i) {
46         pref[i + 1] = addmod(mulmod(pref[i], BASE), s[i]);
47     }
48 }
49 inline uint64_t operator()(const int from, const int to) const {
50     assert(0 <= from <= to && to < n);
51     // pref[to + 1] - pref[from] * pw[to - from + 1]
52     return submod(pref[to + 1], mulmod(pref[from], pw[to - from + 1]));
53 }
54 };
55 mt19937 rng((unsigned int)
56             chrono::steady_clock::now().time_since_epoch().count());
57 uint64_t Hash61::BASE = (MOD >> 2) + rng() % (MOD >> 1);
58 vector<uint64_t> Hash61::pw = vector<uint64_t>(1, 1);

```

5 Number Theory

5.1 Euler's totient function

- Euler's totient function, also known as **phi-function** $\phi(n)$ counts the number of integers between 1 and n inclusive, that are **coprime to** n .

- Properties:

- Divisor sum property: $\sum_{d|n} \phi(d) = n$.
- $\phi(n)$ is a **prime number** when $n = 3, 4, 6$.
- If p is a prime number, then $\phi(p) = p - 1$.
- If p is a prime number and $k \geq 1$, then $\phi(p^k) = p^k - p^{k-1}$.
- If a and b are **coprime**, then $\phi(ab) = \phi(a) \cdot \phi(b)$.
- In general, for **not coprime** a and b , with $d = \gcd(a, b)$ this equation holds:
$$\phi(ab) = \phi(a) \cdot \phi(b) \cdot \frac{d}{\phi(d)}.$$

– With $n = p_1^{k_1} \cdot p_2^{k_2} \cdots p_m^{k_m}$:

$$\begin{aligned}\phi(n) &= \phi(p_1^{k_1}) \cdot \phi(p_2^{k_2}) \cdots \phi(p_m^{k_m}) \\ &= n \cdot \left(1 - \frac{1}{p_1}\right) \cdot \left(1 - \frac{1}{p_2}\right) \cdots \left(1 - \frac{1}{p_m}\right)\end{aligned}$$

• Application in Euler's theorem:

– If $\gcd(a, M) = 1$, then:

$$a^{\phi(M)} \equiv 1 \pmod{M} \Rightarrow a^n \equiv a^{n \bmod \phi(M)} \pmod{M}$$

– In general, for arbitrary a, M and $n \geq \log_2 M$:

$$a^n \equiv a^{\phi(M) + [n \bmod \phi(M)]} \pmod{M}$$

5.2 Mobius function

• For a positive integer $n = p_1^{k_1} \cdot p_2^{k_2} \cdots p_m^{k_m}$:

$$\mu(n) = \begin{cases} 1, & \text{if } n = 1 \\ 0, & \text{if } \exists k_i > 1 \\ (-1)^m & \text{otherwise} \end{cases}$$

• Properties:

– $\sum_{d|n} \mu(d) = [n = 1]$.

– If a and b are **coprime**, then $\mu(ab) = \mu(a) \cdot \mu(b)$.

– Mobius inversion: let f and g be arithmetic functions:

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu\left(\frac{n}{d}\right) g(d)$$

5.3 Primes

Approximating the number of primes up to n :

n	$\pi(n)$	$\frac{n}{\ln n - 1}$
100 ($1e^2$)	25	28
500 ($5e^2$)	95	96
1000 ($1e^3$)	168	169
5000 ($5e^3$)	669	665
10000 ($1e^4$)	1229	1218
50000 ($5e^4$)	5133	5092
100000 ($1e^5$)	9592	9512
500000 ($5e^5$)	41538	41246
1000000 ($1e^6$)	78498	78030
5000000 ($5e^6$)	348513	346622

($\pi(n)$ = the number of primes less than or equal to n , $\frac{n}{\ln n - 1}$ is used to approximate $\pi(n)$).

5.4 Wilson's theorem

A positive integer n is a prime if and only if:

$$(n-1)! \equiv n-1 \pmod{n}$$

5.5 Zeckendorf's theorem

The Zeckendorf's theorem states that every positive integer n can be represented uniquely as a sum of one or more distinct non-consecutive Fibonacci numbers. For example:

$$64 = 55 + 8 + 1$$

$$85 = 55 + 21 + 8 + 1$$

```
1 vector<int> zeckendofth_theorem(int n) {
2     vector<int> fibs = {1, 1};
3     int sz = 2;
4     while (fibs.back() <= n) {
5         fibs.push_back(fibs[sz - 1] + fibs[sz - 2]);
6         sz++;
7     }
8     fibs.pop_back();
9     vector<int> nums;
10    int p = sz - 1;
11    while (n > 0) {
12        if (n >= fibs[p]) {
13            nums.push_back(fibs[p]);
14            n -= fibs[p];
15        }
16        p--;
17    }
18    return nums;
19 }
```

5.6 Bitwise operation

- $a + b = (a \oplus b) + 2(a \& b)$
- $a | b = (a \oplus b) + (a \& b)$
- $a \& (b \oplus c) = (a \& b) \oplus (a \& c)$
- $a | (b \& c) = (a | b) \& (a | c)$
- $a \& (b | c) = (a \& b) | (a \& c)$
- $a | (a \& b) = a$
- $a \& (a | b) = a$
- $n = 2^k \Leftrightarrow !(n \& (n - 1)) = 1$
- $-a = \sim a + 1$
- $(4i) \oplus (4i+1) \oplus (4i+2) \oplus (4i+3) = 0$
- Iterating over all subsets of a set and iterating over all submasks of a mask:

```

1 for (int mask = 0; mask < (1 << n); ++mask) {
2     for (int i = 0; i < n; ++i) {
3         if (mask & (1 << i)) {
4             // do something...
5         }
6     }
7     // Time complexity: O(n * 2^n).
8 }
9 for (int mask = 0; mask < (1 << n); ++mask) {
10    for (int submask = mask; ; submask = (submask - 1) & mask) {
11        // do something...
12        if (submask == 0) break;
13    }
14    // Time complexity: O(3^n).
15 }
```

5.7 Pollard's rho algorithm

```

1 using num_t = long long;
2 const int PRIME_MAX = (int) 4e4; // for handle numbers <= 1e9.
3 const int LIMIT = (int) 1e9;
4 vector<int> primes;
5 int small_primes[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 73, 113,
6     193, 311, 313, 407521, 299210837};
7 void linear_sieve(int n);
8 num_t mulmod(num_t a, num_t b, num_t mod);
9 num_t powmod(num_t a, num_t n, num_t mod);
10 bool miller_rabin(num_t a, num_t d, int s, num_t mod) {
11     num_t x = powmod(a, d, mod);
12     if (x == mod - 1 || x == 1) {
13         return true;
14     }
15     for (int i = 0; i < s - 1; ++i) {
16         x = mulmod(x, x, mod);
17         if (x == mod - 1) return true;
18     }
19     return false;
20 }
21 bool is_prime(num_t n, int tests = 10) {
22     if (n < 4) return (n > 1);
23     num_t d = n - 1;
24     int s = 0;
25     while (d % 2 == 0) { d >>= 1; s++; }
```

```

25     for (int i = 0; i < tests; ++i) {
26         int a = small_primes[i];
27         if (n == a) return true;
28         if (n % a == 0 || !miller_rabin(a, d, s, n)) return false;
29     }
30     return true;
31 }
32 num_t f(num_t x, int c, num_t mod) { // f(x) = (x^2 + c) % mod.
33     x = mulmod(x, x, mod);
34     x += c;
35     if (x >= mod) x -= mod;
36     return x;
37 }
38 num_t pollard_rho(num_t n, int c) {
39     // algorithm to find a random divisor of 'n'.
40     // using random function: f(x) = (x^2 + c) % n.
41     num_t x = 2, y = x, d;
42     long long p = 1;
43     int dist = 0;
44     while (true) {
45         y = f(y, c, n);
46         dist++;
47         d = __gcd(llabs(x - y), n);
48         if (d > 1) break;
49         if (dist == p) { dist = 0; p *= 2; x = y; }
50     }
51     return d;
52 }
53 void factorize(int n, vector<num_t> &factors);
54 void llfactorize(num_t n, vector<num_t> &factors) {
55     if (n < 2) return;
56     if (is_prime(n)) {
57         factors.emplace_back(n);
58         return;
59     }
60     if (n < LIMIT) {
61         factorize(n, factors);
62         return;
63     }
64     num_t d = n;
65     for (int c = 2; d == n; c++) {
66         d = pollard_rho(n, c);
67     }
68     llfactorize(d, factors);
69     llfactorize(n / d, factors);
70 }
71 vector<num_t> gen_divisors(vector<pair<num_t, int>> &factors) {
72     vector<num_t> divisors = {1};
73     for (auto &x : factors) {
74         int sz = (int) divisors.size();
75         for (int i = 0; i < sz; ++i) {
76             num_t cur = divisors[i];
77             for (int j = 0; j < x.second; ++j) {
```

```

78         cur *= x.first;
79         divisors.push_back(cur);
80     }
81 }
82 }
83 return divisors; // this array is NOT sorted yet.
84 }

```

5.8 Bitset sieve

```

1 /**
2  * Description: sieve of eratosthenes for large n (up to 1e9).
3  * Time and space (tested on codeforces):
4  *   + For n = 1e8: ~200 ms, 6 MB.
5  *   + For n = 1e9: ~4000 ms, 60 MB.
6  */
7 const int N = (int) 1e8;
8 bitset<N / 2 + 1> isPrime;
9 void sieve(int n = N) {
10     isPrime.flip();
11     isPrime[0] = false;
12     for (int i = 3; i <= (int) sqrt(n); i += 2) {
13         if (isPrime[i >> 1]) {
14             for (int j = i * i; j <= n; j += 2 * i) {
15                 isPrime[j >> 1] = false;
16             }
17         }
18     }
19 }
20 void example(int n) {
21     sieve(n);
22     int primeCnt = (n >= 2);
23     for (int i = 3; i <= n; i += 2) {
24         if (isPrime[i >> 1]) {
25             primeCnt++;
26         }
27     }
28     cout << primeCnt << '\n';
29 }

```

5.9 Block sieve

```

1 /**
2  * Description: very fast sieve of eratosthenes for large n (up to 1e9).
3  * Source: kactl.
4  * Time and space (tested on codeforces):
5  *   + For n = 1e8: ~160 ms, 60 MB.
6  *   + For n = 1e9: ~1600 ms, 505 MB.
7  * Need to check memory limit.
8  */
9 const int N = (int) 1e8;
10 bitset<N + 1> is_prime;
11 vector<int> fast_sieve() {
12     const int S = (int) sqrt(N), R = N / 2;
13     vector<int> primes = {2};

```

```

14     vector<bool> sieve(S + 1, true);
15     vector<array<int, 2>> cp;
16     for (int i = 3; i <= S; i += 2) {
17         if (sieve[i]) {
18             cp.push_back({i, i * i / 2});
19             for (int j = i * i; j <= S; j += 2 * i) {
20                 sieve[j] = false;
21             }
22         }
23     }
24     for (int L = 1; L <= R; L += S) {
25         array<bool, S> block{};
26         for (auto &[p, idx] : cp) {
27             for (; idx < S + L; idx += p) block[idx - L] = true;
28         }
29         for (int i = 0; i < min(S, R - L); ++i) {
30             if (!block[i]) primes.push_back((L + i) * 2 + 1);
31         }
32     }
33     for (int p : primes) is_prime[p] = true;
34     return primes;
35 }

```

5.10 Combinatorics

5.10.1 Fibonacci numbers

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- The first 20 Fibonacci numbers ($n = 0, 1, 2, \dots, 19$):

$$F_n = 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181$$

- Properties:

$$\left. \begin{aligned} F_{2n+1} &= F_n^2 + F_{n+1}^2 \\ F_{2n} &= F_{n-1} \cdot F_n + F_n \cdot F_{n+1} \\ F_{n+1} \cdot F_{n-1} - F_n^2 &= (-1)^n \end{aligned} \right| \begin{aligned} n \mid m &\iff F_n \mid F_m \\ (F_n, F_m) &= F_{(n,m)} \end{aligned}$$

5.10.2 Catalan numbers

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

$$C_{n+1} = \sum_{i=0}^n C_i C_{n-i}, \quad C_0 = 1, \quad C_n = \frac{4n-2}{n+1} C_{n-1}$$

- The first 12 Catalan numbers ($n = 0, 1, 2, \dots, 11$):

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

- Applications of Catalan numbers:

- difference binary search trees with n vertices from 1 to n .
- rooted binary trees with $n + 1$ leaves (vertices are not numbered).
- correct bracket sequence of length $2 * n$.
- permutation $[n]$ with no 3-term increasing subsequence (i.e. doesn't exist $i < j < k$ for which $a[i] < a[j] < a[k]$).
- ways a convex polygon of $n + 2$ sides can split into triangles by connecting vertices.

5.10.3 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k non-empty groups.

$$S(n, k) = S(n - 1, k - 1) + kS(n - 1, k)$$

$$S(n, 1) = S(n, n) = 1$$

$$S(n, k) = \frac{1}{k!} \sum_{i=0}^k (-1)^{k-i} \binom{k}{i} i^n$$

5.10.4 Derangements

Permutation of the elements of a set, such that no element appears in its original position (no fixed point). Recursive formulas:

$$D(n) = (n - 1)[D(n - 1) + D(n - 2)] = nD(n - 1) + (-1)^n$$

6 Linear algebra

6.1 Gauss elimination

```

1 const double EPS = 1e-9;
2 const int INF = 2; // it doesn't actually have to be infinity or a big number
3 int gauss (vector < vector<double> > a, vector<double> & ans) {
4     int n = (int) a.size();
5     int m = (int) a[0].size() - 1;
6     vector<int> where (m, -1);
7     for (int col=0, row=0; col<m && row<n; ++col) {
8         int sel = row;
9         for (int i=row; i<n; ++i)
10             if (abs (a[i][col]) > abs (a[sel][col]))
11                 sel = i;
12         if (abs (a[sel][col]) < EPS)
13             continue;

```

```

14         for (int i=col; i<=m; ++i)
15             swap (a[sel][i], a[row][i]);
16         where[col] = row;
17
18         for (int i=0; i<n; ++i)
19             if (i != row) {
20                 double c = a[i][col] / a[row][col];
21                 for (int j=col; j<=m; ++j)
22                     a[i][j] -= a[row][j] * c;
23             }
24         ++row;
25     }
26     ans.assign (m, 0);
27     for (int i=0; i<m; ++i)
28         if (where[i] != -1)
29             ans[i] = a[where[i]][m] / a[where[i]][i];
30     for (int i=0; i<n; ++i) {
31         double sum = 0;
32         for (int j=0; j<m; ++j)
33             sum += ans[j] * a[i][j];
34         if (abs (sum - a[i][m]) > EPS)
35             return 0;
36     }
37     for (int i=0; i<m; ++i)
38         if (where[i] == -1)
39             return INF;
40     return 1;
41 }

```

7 Geometry

7.1 Fundamentals

7.1.1 Point

```

1 const double PI = acos(-1);
2 const double EPS = 1e-9;
3 typedef double ftype;
4 struct point {
5     ftype x, y;
6     point(ftype _x = 0, ftype _y = 0): x(_x), y(_y) {}
7     point& operator+=(const point& other) {
8         x += other.x; y += other.y; return *this;
9     }
10    point& operator-=(const point& other) {
11        x -= other.x; y -= other.y; return *this;
12    }
13    point& operator*=(ftype t) {
14        x *= t; y *= t; return *this;
15    }
16    point& operator/=(ftype t) {
17        x /= t; y /= t; return *this;
18    }
19    point operator+(const point& other) const {
20        return point(*this) += other;

```

```

21     }
22     point operator-(const point& other) const {
23         return point(*this) -= other;
24     }
25     point operator*(ftype t) const {
26         return point(*this) *= t;
27     }
28     point operator/(ftype t) const {
29         return point(*this) /= t;
30     }
31     point rotate(double angle) const {
32         return point(x * cos(angle) - y * sin(angle), x * sin(angle) + y *
33             cos(angle));
34     }
35     friend istream& operator>>(istream &in, point &t);
36     friend ostream& operator<<(ostream &out, const point& t);
37     bool operator<(const point& other) const {
38         if (fabs(x - other.x) < EPS)
39             return y < other.y;
40         return x < other.x;
41     };
42
43     istream& operator>>(istream &in, point &t) {
44         in >> t.x >> t.y;
45         return in;
46     }
47     ostream& operator<<(ostream &out, const point& t) {
48         out << t.x << ' ' << t.y;
49         return out;
50     }
51
52     ftype dot(point a, point b) {return a.x * b.x + a.y * b.y;}
53     ftype norm(point a) {return dot(a, a);}
54     ftype abs(point a) {return sqrt(norm(a));}
55     ftype angle(point a, point b) {return acos(dot(a, b) / (abs(a) * abs(b)));}
56     ftype proj(point a, point b) {return dot(a, b) / abs(b);}
57     ftype cross(point a, point b) {return a.x * b.y - a.y * b.x;}
58     bool ccw(point a, point b, point c) {return cross(b - a, c - a) > EPS;}
59     bool collinear(point a, point b, point c) {return fabs(cross(b - a, c - a)) <
60         EPS;}
61     point intersect(point a1, point d1, point a2, point d2) {
62         double t = cross(a2 - a1, d2) / cross(d1, d2);
63         return a1 + d1 * t;
64     }

```

7.1.2 Line

```

1 struct line {
2     double a, b, c;
3     line (double _a = 0, double _b = 0, double _c = 0): a(_a), b(_b), c(_c) {}
4     friend ostream & operator<<(ostream& out, const line& l);
5 };
6 ostream & operator<<(ostream& out, const line& l) {
7     out << l.a << ' ' << l.b << ' ' << l.c;

```

```

8     return out;
9 }
10 void pointsToLine(const point& p1, const point& p2, line& l) {
11     if (fabs(p1.x - p2.x) < EPS)
12         l = {1.0, 0.0, -p1.x};
13     else {
14         l.a = - (double)(p1.y - p2.y) / (p1.x - p2.x);
15         l.b = 1.0;
16         l.c = - l.a * p1.x - l.b * p1.y;
17     }
18 }
19 void pointsSlopeToLine(const point& p, double m, line& l) {
20     l.a = -m;
21     l.b = 1;
22     l.c = -l.a * p.x - l.b * p.y;
23 }
24 bool areParallel(const line& l1, const line& l2) {
25     return fabs(l1.a - l2.a) < EPS && fabs(l1.b - l2.b) < EPS;
26 }
27 bool areSame(const line& l1, const line& l2) {
28     return areParallel(l1, l2) && fabs(l1.c - l2.c) < EPS;
29 }
30 bool areIntersect(line l1, line l2, point& p) {
31     if (areParallel(l1, l2)) return false;
32     p.x = - (l1.c * l2.b - l1.b * l2.c) / (l1.a * l2.b - l1.b * l2.a);
33     if (fabs(l1.b) > EPS) p.y = - (l1.c + l1.a * p.x);
34     else p.y = - (l2.c + l2.a * p.x);
35     return 1;
36 }
37 double distToLine(point p, point a, point b, point& c) {
38     double t = dot(p - a, b - a) / norm(b - a);
39     c = a + (b - a) * t;
40     return abs(c - p);
41 }
42 double distToSegment(point p, point a, point b, point& c) {
43     double t = dot(p - a, b - a) / norm(b - a);
44     if (t > 1.0)
45         c = point(b.x, b.y);
46     else if (t < 0.0)
47         c = point(a.x, a.y);
48     else
49         c = a + (b - a) * t;
50     return abs(c - p);
51 }
52 bool intersectTwoSegment(point a, point b, point c, point d) {
53     ftype ABxAC = cross(b - a, c - a);
54     ftype ABxAD = cross(b - a, d - a);
55     ftype CDxCA = cross(d - c, a - c);
56     ftype CDxCB = cross(d - c, b - c);
57     if (ABxAC == 0 || ABxAD == 0 || CDxCA == 0 || CDxCB == 0) {
58         if (ABxAC == 0 && dot(a - c, b - c) <= 0) return true;
59         if (ABxAD == 0 && dot(a - d, b - d) <= 0) return true;
60         if (CDxCA == 0 && dot(c - a, d - a) <= 0) return true;

```

```

61         if (CDxCB == 0 && dot(c - b, d - b) <= 0) return true;
62         return false;
63     }
64     return (ABxAC * ABxAD < 0 && CDxCA * CDxCB < 0);
65 }
66 void perpendicular(line l1, point p, line& l2) {
67     if (fabs(l1.a) < EPS)
68         l2 = {1.0, 0.0, -p.x};
69     else {
70         l2.a = -l1.b / l1.a;
71         l2.b = 1.0;
72         l2.c = -l2.a * p.x - l2.b * p.y;
73     }
74 }

```

7.1.3 Circle

```

1 int insideCircle(const point& p, const point& center, ftype r) {
2     ftype d = norm(p - center);
3     ftype rSq = r * r;
4     return fabs(d - rSq) < EPS ? 0 : (d - rSq >= EPS ? 1 : -1);
5 }
6 bool circle2PointsR(const point& p1, const point& p2, ftype r, point& c) {
7     double h = r * r - norm(p1 - p2) / 4.0;
8     if (fabs(h) < 0) return false;
9     h = sqrt(h);
10    point perp = (p2 - p1).rotate(PI / 2.0);
11    point m = (p1 + p2) / 2.0;
12    c = m + perp * (h / abs(perp));
13    return true;
14 }

```

7.1.4 Triangle

```

1 double areaTriangle(double ab, double bc, double ca) {
2     double p = (ab + bc + ca) / 2;
3     return sqrt(p) * sqrt(p - ab) * sqrt(p - bc) * sqrt(p - ca);
4 }
5 double rInCircle(double ab, double bc, double ca) {
6     double p = (ab + bc + ca) / 2;
7     return areaTriangle(ab, bc, ca) / p;
8 }
9 double rInCircle(point a, point b, point c) {
10    return rInCircle(abs(a - b), abs(b - c), abs(c - a));
11 }
12 bool inCircle(point p1, point p2, point p3, point &ctr, double &r) {
13     r = rInCircle(p1, p2, p3);
14     if (fabs(r) < EPS) return false;
15     line l1, l2;
16     double ratio = abs(p2 - p1) / abs(p3 - p1);
17     point p = p2 + (p3 - p2) * (ratio / (1 + ratio));
18     pointsToLine(p1, p, l1);
19     ratio = abs(p1 - p2) / abs(p2 - p3);
20     p = p1 + (p3 - p1) * (ratio / (1 + ratio));
21     pointsToLine(p2, p, l2);
22     areIntersect(l1, l2, ctr);

```

```

23     return true;
24 }
25 double rCircumCircle(double ab, double bc, double ca) {
26     return ab * bc * ca / (4.0 * areaTriangle(ab, bc, ca));
27 }
28 double rCircumCircle(point a, point b, point c) {
29     return rCircumCircle(abs(b - a), abs(c - b), abs(a - c));
30 }

```

7.1.5 Convex hull

```

1 vector<point> CH_Andrew(vector<point> &Pts) { // overall O(n log n)
2     int n = Pts.size(), k = 0;
3     vector<point> H(2 * n);
4     sort(Pts.begin(), Pts.end());
5     for (int i = 0; i < n; ++i) {
6         while ((k >= 2) && !ccw(H[k - 2], H[k - 1], Pts[i])) --k;
7         H[k++] = Pts[i];
8     }
9     for (int i = n - 2, t = k + 1; i >= 0; --i) {
10        while ((k >= t) && !ccw(H[k - 2], H[k - 1], Pts[i])) --k;
11        H[k++] = Pts[i];
12    }
13    H.resize(k);
14    return H;
15 }

```

7.1.6 Polygon

```

1 double perimeter(const vector<point> &P) {
2     double ans = 0.0;
3     for (int i = 0; i < (int)P.size() - 1; ++i)
4         ans += abs(P[i] - P[i + 1]);
5     return ans;
6 }
7 double area(const vector<point> &P) {
8     double ans = 0.0;
9     for (int i = 0; i < (int)P.size() - 1; ++i)
10        ans += (P[i].x * P[i + 1].y - P[i + 1].x * P[i].y);
11    return fabs(ans) / 2.0;
12 }
13 bool isConvex(const vector<point> &P) {
14    int n = (int)P.size();
15    if (n <= 3) return false;
16    bool firstTurn = ccw(P[0], P[1], P[2]);
17    for (int i = 1; i < n - 1; ++i)
18        if (ccw(P[i], P[i + 1], P[(i + 2) == n ? 1 : i + 2]) != firstTurn)
19            return false;
20    return true;
21 }
22 int insidePolygon(point pt, const vector<point> &P) {
23    int n = (int)P.size();
24    if (n <= 3) return -1;
25    bool on_polygon = false;
26    for (int i = 0; i < n - 1; ++i)
27        if (fabs(abs(P[i] - pt) + abs(pt - P[i + 1]) - abs(P[i] - P[i + 1]))

```



```

    < EPS)
28     on_polygon = true;
29     if (on_polygon) return 0;
30     double sum = 0.0;
31     for (int i = 0; i < n - 1; ++i) {
32         if (ccw(pt, P[i], P[i + 1]))
33             sum += angle(P[i] - pt, P[i + 1] - pt);
34         else
35             sum -= angle(P[i] - pt, P[i + 1] - pt);
36     }
37     return fabs(sum) > PI ? 1 : -1;
38 }

```

7.2 Minimum enclosing circle

```

1 /**
2  * Description: computes the minimum circle that encloses all the given
3  * points.
4  */
5
6 double abs(point a) { return sqrt(a.X * a.X + a.Y * a.Y); }
7
8 point center_from(double bx, double by, double cx, double cy) {
9     double B = bx * bx + by * by, C = cx * cx + cy * cy, D = bx * cy - by *
10     cx;
11     return point((cy * B - by * C) / (2 * D), (bx * C - cx * B) / (2 * D));
12 }
13
14 circle circle_from(point A, point B, point C) {
15     point I = center_from(B.X - A.X, B.Y - A.Y, C.X - A.X, C.Y - A.Y);
16     return circle(I + A, abs(I));
17 }
18
19 const int N = 100005;
20 int n, x[N], y[N];
21 point a[N];
22
23 circle emo_welzl(int n, vector<point> T) {
24     if (T.size() == 3 || n == 0) {
25         if (T.size() == 0) return circle(point(0, 0), -1);
26         if (T.size() == 1) return circle(T[0], 0);
27         if (T.size() == 2) return circle((T[0] + T[1]) / 2, abs(T[0] - T[1])
28         / 2);
29     }
30     return circle_from(T[0], T[1], T[2]);
31 }
32
33 random_shuffle(a + 1, a + n + 1);
34 circle Result = emo_welzl(0, T);
35 for (int i = 1; i <= n; i++)
36     if (abs(Result.X - a[i]) > Result.Y + 1e-9) {
37         T.push_back(a[i]);
38         Result = emo_welzl(i - 1, T);
39         T.pop_back();
40     }
41 return Result;
42 }

```

8 Graph

8.1 Bellman-Ford algorithm

```

1 /**
2  * Description: single source shortest path in a weighted (negative or
3  * positive) directed graph.
4  * Time: O(N * M).
5  * Tested: https://open.kattis.com/problems/shortestpath3
6  */
7 const int64_t INF = (int64_t) 2e18;
8 struct Edge {
9     int u, v; // u -> v
10     int64_t w;
11     Edge() {}
12     Edge(int _u, int _v, int64_t _w) : u(_u), v(_v), w(_w) {}
13 };
14 vector<int64_t> bellmanFord(int s) {
15     // dist[stating] = 0.
16     // dist[u] = +INF, if u is unreachable.
17     // dist[u] = -INF, if there is a negative cycle on the path from s to u.
18     // -INF < dist[u] < +INF, otherwise.
19     vector<int64_t> dist(n, INF);
20     dist[s] = 0;
21     for (int i = 0; i < n - 1; ++i) {
22         bool any = false;
23         for (auto [u, v, w] : edges) {
24             if (dist[u] != INF && dist[v] > w + dist[u]) {
25                 dist[v] = w + dist[u];
26                 any = true;
27             }
28         }
29         if (!any) break;
30     }
31     // handle negative cycles
32     for (int i = 0; i < n - 1; ++i) {
33         for (auto [u, v, w] : edges) {
34             if (dist[u] != INF && dist[v] > w + dist[u]) {
35                 dist[v] = -INF;
36             }
37         }
38     }
39     return dist;
40 }

```

8.2 Articulation point and Bridge

```

1 /**
2  * Description: finding articulation points and bridges in a simple
3  * undirected graph.
4  * Tested: https://oj.vnoi.info/problem/graph_
5  */
6 const int N = (int) 1e5;
7 vector<int> g[N];
8 int num[N], low[N], dfs_timer;

```

```

8  bool joint[N];
9  vector<pair<int, int>> bridges;
10 void dfs(int u, int prev) {
11     low[u] = num[u] = ++dfs_timer;
12     int child = 0;
13     for (int v : g[u]) {
14         if (v == prev) continue;
15         if (num[v]) low[u] = min(low[u], num[v]);
16         else {
17             dfs(v, u);
18             low[u] = min(low[u], low[v]);
19             child++;
20             if (low[v] >= num[v]) {
21                 bridges.emplace_back(u, v);
22             }
23             if (u != prev && low[v] >= num[u]) joint[u] = true;
24         }
25     }
26     if (u == prev && child > 1) joint[u] = true;
27 }
28
29 int main() {
30     int n, m;
31     cin >> n >> m;
32     for (int i = 0; i < m; ++i) {
33         int u, v;
34         cin >> u >> v;
35         u--; v--;
36         g[u].push_back(v);
37         g[v].push_back(u);
38     }
39     for (int i = 0; i < n; ++i) {
40         if (!num[i]) dfs(i, i);
41     }
42     return 0;
43 }

```

8.3 Strongly connected components

```

1  /**
2   * Description: Tarjan's algorithm finds strongly connected components
3   * in a directed graph. If vertices u and v belong to the same component,
4   * then scc_id[u] == scc_id[v].
5   * Tested: https://judge.yosupo.jp/problem/scc
6   */
7  const int N = (int) 5e5;
8  vector<int> g[N], st;
9  int low[N], num[N], dfs_timer, scc_id[N], scc;
10 bool used[N];
11 void Tarjan(int u) {
12     low[u] = num[u] = ++dfs_timer;
13     st.push_back(u);
14     for (int v : g[u]) {
15         if (used[v]) continue;

```

```

16         if (num[v] == 0) {
17             Tarjan(v);
18             low[u] = min(low[u], low[v]);
19         }
20         else {
21             low[u] = min(low[u], num[v]);
22         }
23     }
24     if (low[u] == num[u]) {
25         int v;
26         do {
27             v = st.back(); st.pop_back();
28             debug(u, v)
29             used[v] = true;
30             scc_id[v] = scc;
31         } while (v != u);
32         scc++;
33     }
34 }

```

8.4 Topo sort

```

1  /**
2   * Description: A topological sort of a directed acyclic graph
3   * is a linear ordering of its vertices such that for every directed edge
4   * from vertex u to vertex v, u comes before v in the ordering.
5   * Note: If there are cycles, the returned list will have size smaller than n
6   * (i.e, topo.size() < n).
7   * Tested: https://judge.yosupo.jp/problem/scc
8   */
9  vector<int> topo_sort(const vector<vector<int>> &g) {
10     int n = (int) g.size();
11     vector<int> indeg(n);
12     for (int u = 0; u < n; ++u) {
13         for (int v : g[u]) indeg[v]++;
14     }
15     queue<int> q; // Note: use min-heap to get the smallest lexicographical
16     // order.
17     for (int u = 0; u < n; ++u) {
18         if (indeg[u] == 0) q.emplace(u);
19     }
20     vector<int> topo;
21     while (!q.empty()) {
22         int u = q.front(); q.pop();
23         topo.emplace_back(u);
24         for (int v : g[u]) {
25             if (--indeg[v] == 0) q.emplace(v);
26         }
27     }
28     return topo;
29 }

```

8.5 K-th smallest shortest path

```

1  /** Finding the k-th smallest shortest path from vertex s to vertex t,

```

```

2  * each vertex can be visited more than once.
3  */
4  using adj_list = vector<vector<pair<int, int>>>;
5  vector<int> k_smallest(const adj_list &g, int k, int s, int t) {
6      int n = (int) g.size();
7      vector<long long> ans;
8      vector<int> cnt(n);
9      using pli = pair<long long, int>;
10     priority_queue<pli, vector<pli>, greater<pli>> pq;
11     pq.emplace(0, s);
12     while (!pq.empty() && cnt[t] < k) {
13         int u = pq.top().second;
14         long long d = pq.top().first;
15         pq.pop();
16         if (cnt[u] == k) continue;
17         cnt[u]++;
18         if (u == t) {
19             ans.push_back(d);
20         }
21         for (auto [v, cost] : g[u]) {
22             pq.emplace(d + cost, v);
23         }
24     }
25     assert(ans.size() == k);
26     return ans;
27 }

```

8.6 Eulerian path

8.6.1 Directed graph

```

1  /**
2   * Hierholzer's algorithm.
3   * Description: An Eulerian path in a directed graph is a path that visits
4   * all edges exactly once.
5   * An Eulerian cycle is a Eulerian path that is a cycle.
6   * Time complexity: O(|E|).
7   */
8  vector<int> find_path_directed(const vector<vector<int>> &g, int s) {
9      int n = (int) g.size();
10     vector<int> stack, cur_edge(n), vertices;
11     stack.push_back(s);
12     while (!stack.empty()) {
13         int u = stack.back();
14         stack.pop_back();
15         while (cur_edge[u] < (int) g[u].size()) {
16             stack.push_back(u);
17             u = g[u][cur_edge[u]++];
18         }
19         vertices.push_back(u);
20     }
21     reverse(vertices.begin(), vertices.end());
22     return vertices;
23 }

```

8.6.2 Undirected graph

```

1  /**
2   * Hierholzer's algorithm.
3   * Description: An Eulerian path in a undirected graph is a path that visits
4   * all edges exactly once.
5   * An Eulerian cycle is a Eulerian path that is a cycle.
6   * Time complexity: O(|E|).
7   */
8  struct Edge {
9      int to;
10     list<Edge>::iterator reverse_edge;
11     Edge(int _to) : to(_to) {}
12 };
13 vector<int> vertices;
14 void find_path(vector<list<Edge>> &g, int u) {
15     while (!g[u].empty()) {
16         int v = g[u].front().to;
17         g[v].erase(g[u].front().reverse_edge);
18         g[u].pop_front();
19         find_path(g, v);
20     }
21     vertices.emplace_back(u); // reversion list.
22 }
23 void add_edge(int u, int v) {
24     g[u].emplace_front(v);
25     g[v].emplace_front(u);
26     g[u].front().reverse_edge = g[v].begin();
27     g[v].front().reverse_edge = g[u].begin();
28 }

```

9 Misc.

9.1 Ternary search

```

1  const double eps = 1e-9;
2  double ternary_search_max(double l, double r) {
3      // find x0 such that: f(x0) > f(x), \all x: l <= x <= r.
4      while (r - l > eps) {
5          double mid1 = l + (r - l) / 3;
6          double mid2 = r - (r - l) / 3;
7          if (f(mid1) < f(mid2)) l = mid1;
8          else r = mid2;
9      }
10     return l;
11 }
12 double ternary_search_min(double l, double r) {
13     // find x0 such that: f(x0) < f(x), \all x: l <= x <= r.
14     while (r - l > eps) {
15         double mid1 = l + (r - l) / 3;
16         double mid2 = r - (r - l) / 3;
17         if (f(mid1) > f(mid2)) l = mid1;
18         else r = mid2;
19     }
20     return l;
21 }

```

21 }

9.2 Dutch flag national problem

```

1 void dutch_flag_national(vector<int> &arr) {
2     // All elements that are LESS than pivot are moved to the LEFT.
3     // All elements that are GREATER than pivot are moved to the RIGHT.
4     // E.g. [1, 2, 0, 0, 2, 2, 1], pivot = 1 -> [0, 0, 1, 1, 2, 2, 2].
5     int n = (int) arr.size();
6     int i = 0, j = 0, k = n - 1;
7     int pivot = 1;
8     // 0....i....j....k....n
9     while (j <= k) {
10         if (arr[j] < pivot) {
11             swap(arr[i], arr[j]);
12             i++;
13             j++;
14         }
15         else if (arr[j] > pivot) {
16             swap(arr[j], arr[k]);
17             k--;
18         }
19         else {
20             j++;
21         }
22     }
23     // 0 <= index <= i - 1: arr[index] < mid.
24     // i <= index <= k: arr[index] = mid.
25     // k + 1 <= index < sz: arr[index] > mid.
26 }

```

9.3 Matrix

```

1 struct Matrix {
2     static const matrix_type INF = numeric_limits<matrix_type>::max();
3     int N, M;
4     vector<vector<matrix_type>> mat;
5
6     Matrix(int _N, int _M, matrix_type v = 0) : N(_N), M(_M) {
7         mat.assign(N, vector<matrix_type>(M, v));
8     }
9     static Matrix identity(int n) { // return identity matrix.
10         Matrix I(n, n);
11         for (int i = 0; i < n; ++i) {
12             I[i][i] = 1;
13         }
14         return I;
15     }
16
17     vector<matrix_type>& operator[](int r) { return mat[r]; }
18     const vector<matrix_type>& operator[](int r) const { return mat[r]; }
19
20     Matrix& operator*=(const Matrix &other) {
21         assert(M == other.N); // [N x M] [other.N x other.M]
22         Matrix res(N, other.M);

```

```

23     for (int r = 0; r < N; ++r) {
24         for (int c = 0; c < other.M; ++c) {
25             long long square_mod = (long long) MOD * MOD;
26             long long sum = 0;
27             for (int g = 0; g < M; ++g) {
28                 sum += (long long) mat[r][g] * other[g][c];
29                 if (sum >= square_mod) sum -= square_mod;
30             }
31             res[r][c] = sum % MOD;
32         }
33     }
34     mat.swap(res.mat); return *this;
35 }
36 };

```