

Teamnote of 2mic1cup

swishy, lenhanbo, hungcubu

Last upadted on October 10, 2025



Contents

1	Helpers	2		
1.1	Stress Tester	2		
1.2	Random	2		
2	Data Structure	2		
2.1	Iterative Segment Tree	2		
2.2	Lazy Segment Tree	2		
2.3	Sparse Table	3		
2.4	Implicit Treap	3		
2.5	Dynamic Segment Tree	4		
2.6	Persistent Segment Tree	5		
2.7	2D Fenwick Tree	5		
2.8	Disjoint Set Union	5		
2.9	Line Container	6		
2.10	Lichao Tree	6		
2.11	Ordered Set	6		
2.12	Minimum Stack/Deque	7		
2.13	Dynamic Bitset	7		
3	Graph	7		
3.1	Graph	7		
3.2	Tree	8		
3.3	Strongly Connected Components	8		
3.4	Bridges and Articulations	9		
3.5	Two SAT	9		
3.6	MCMF	10		
3.7	Maximum Flow (Dinic)	10		
3.8	Maximum Matching (Hopcroft Karp)	11		
3.9	General Matching (Blossom)	12		
4	Math	12		
4.1	Modular Int	12		
4.2	Modular Square Root	13		
4.3	Discrete Log	13		
4.4	Primate Root	14		
4.5	Euler's Totient Funnction	14		
4.6	Chinese Remainder Theorem	14		
4.7	Extended Euclidean	14		
4.8	Linear Diophantine	15		
4.9	Matrix	15		
4.10	Miller Rabin Primality Test	15		
4.11	Fast Fourier Transform	16		
4.12	OR Convolution	16		
4.13	XOR Convolution	17		
5	String	17		
5.1	Rolling Hash	17		
5.2	Z-Function	17		
5.3	Prefix Function	18		
5.4	Manacher's Algorithm	18		
5.5	Aho-Corasick	18		

1 Helpers

1.1 Stress Tester

Overview

Simple .bat file for stress testing.

Implementation

```
1 @echo off
2 g++ -std=c++20 -o solution test.cpp
3 g++ -std=c++20 -o brute brute.cpp
4 g++ -std=c++20 -o gen gen.cpp
5
6 for /l %%x in (1, 1, 1000) do (
7     gen > input.in
8     solution < input.in > output.out
9     brute < input.in > output2.out
10    fc output.out output2.out > nul
11
12    if ERRORLEVEL 1 (
13        echo INPUT
14        type input.in
15        echo.
16        echo SOLUTION OUTOUT
17        type output.out
18        echo.
19        echo CORRECT OUTPUT
20        type output2.out
21        echo.
22    )
23 )
24 echo all tests passed
```

1.2 Random

Overview

Self explanatory.

Implementation

```
1 #define uid(a, b) uniform_int_distribution<long long>(a,
2   ↪ b)(rng)
3 mt19937 rng(chrono::steady_clock::now().time_since_epoch_
4   ↪ ().count());
```

Usage

- `uid(a, b)` returns random integer between $[a, b]$

2 Data Structure

2.1 Iterative Segment Tree

Overview

For-loop implementation of segment tree, faster than recursive. Note: Operation that depends on ordering is not supported (For example: Minimum prefix sum)

⌚ Time complexity: $\mathcal{O}(n)$ for constructor, $\mathcal{O}(\log n)$ for query

Implementation

```
1 template<typename T>
2 struct SegmentTreeFast{
3     vector<T> a;
4     T defv;
5     int n;
6
7     SegmentTreeFast(int n, T defv) : n(n), defv(defv){
8         a = vector<T>(2 * n, defv);
9     }
10
11     T cmb(T a, T b){ //change if needed
12         return a + b;
13     }
14
15     void build(){ //array is at i + n index
16         for (int i = n - 1; i > 0; --i)
17             a[i] = cmb(a[i << 1], a[i << 1 | 1]);
18     }
19
20     void update(int i, T v){
21         for (a[i += n] = v; i > 1; i >>= 1)
22             a[i >> 1] = cmb(a[i], a[i ^ 1]);
23     }
24
25     T get(int l, int r){
26         r++;
27         T res = defv;
28         for (l += n, r += n; l < r; l >>= 1, r >>= 1){
29             if (l&1) res = cmb(res, a[l++]);
30             if (r&1) res = cmb(res, a[--r]);
31         }
32
33         return res;
34     }
35 };
```

2.2 Lazy Segment Tree

Overview

Segment tree that supports ranged update.

⌚ Time complexity: $\mathcal{O}(n)$ for constructor, $\mathcal{O}(\log n)$ for query

Implementation

```
1 template<typename T>
2 class SegmentTreeLazy{
3 public:
4     vector<T> st, lazy;
5     T defv;
6     int n;
7
8     SegmentTreeLazy(int n, T defv) : n(n), defv(defv){
9         st = vector<T>(n * 4, defv);
10        lazy = vector<T>(n * 4, defv);
11    }
12
13    void update(int l, int r, T v){
14        _update(0, n - 1, 0, l, r, v);
15    }
16
17    T get(int l, int r){
18        return _get(0, n - 1, l, r, 0);
19    }
```

```

19     }
20
21 private:
22     T cmb(T l, T r){
23         return l + r;
24     }
25
26     void push(int i, int l, int r){
27         int mid = (l + r) / 2;
28         lazy[i * 2 + 1] += lazy[i];
29         lazy[i * 2 + 2] += lazy[i];
30
31         st[i * 2 + 1] += (mid - l + 1) * lazy[i];
32         st[i * 2 + 2] += (r - mid) * lazy[i];
33
34         lazy[i] = 0;
35     }
36
37     void _update(int l, int r, int crr, int ql, int qr,
38         ↪ T v){
39         if (qr < l || ql > r)
40             return;
41
42         if (l >= ql && r <= qr){
43             st[crr] += (r - l + 1) * v;
44             lazy[crr] += v;
45             return;
46         }
47
48         push(crr, l, r);
49         int mid = (l + r) / 2;
50         _update(l, mid, crr * 2 + 1, ql, qr, v);
51         _update(mid + 1, r, crr * 2 + 2, ql, qr, v);
52
53         st[crr] = cmb(st[crr * 2 + 1], st[crr * 2 + 2]);
54     }
55
56     T _get(int l, int r, int ql, int qr, int crr){
57         if (qr < l || ql > r)
58             return defv;
59         if (l >= ql && r <= qr)
60             return st[crr];
61
62         push(crr, l, r);
63         int mid = (l + r) / 2;
64         return cmb(_get(l, mid, ql, qr, crr * 2 + 1),
65             ↪ _get(mid + 1, r, ql, qr, crr * 2 + 2));
66     }
67 };

```

2.3 Sparse Table

Overview

Uses binary lifting for efficient queries, offline only.

⌚ Time complexity: $\mathcal{O}(n \log n)$ for constructor, $\mathcal{O}(1)$ for query

Implementation

```

1 template <typename T, class Combine = function<T(const T
2     ↪ &, const T &)>>
3 struct SparseTable{
4     vector<vector<T>> f;
5     vector<int> lg;
6     Combine cmb;
7     int n;
8
9     SparseTable(vector<T> &init, const Combine &cmb) :
10         ↪ n(init.size()), cmb(cmb){

```

```

9         lg = vector<int>(n + 1, 0);
10        for (int i = 2; i <= n; i++){
11            lg[i] = lg[i / 2] + 1;
12        }
13        for (int i = 0; i < n; i++){
14            f.push_back(vector<int>(lg[n] + 1, -1));
15            f[i][0] = init[i];
16        }
17        for (int j = 1; (1 << j) <= n; j++){
18            for (int i = 0; (i + (1 << j) - 1) < n; i++){
19                f[i][j] = cmb(f[i][j - 1], f[i + (1 <<
20                    ↪ (j - 1))] [j - 1]);
21            }
22        }
23
24        T get(int l, int r){
25            int k = lg[r - l + 1];
26            return cmb(f[l][k], f[r - (1 << k) + 1][k]);
27        }
28    };

```

? Usage

- Init minimum range query and uses integer type

```

1 SparseTable<int> rmq(a, [](int a, int b){
2     return min(a, b);
3 });

```

2.4 Implicit Treap

Overview

Implicit treap implementation with range add update and range sum query. push() and upd() functions should be changed accordingly like lazy segment tree.

⌚ Time complexity: $\mathcal{O}(\log n)$ on average for all operations, large constant!!

Implementation

```

1 typedef node* pnode;
2 struct ImplicitTreap{
3 public:
4     pnode root;
5     ImplicitTreap(){
6         root = new node(-1, 0);
7     }
8     void insert(int i, ll val){
9         pnode t1, t2;
10        split(root, i + 1, 0, t1, t2);
11        merge(t1, t1, new node(val));
12        merge(root, t1, t2);
13    }
14    void erase(int i){
15        _erase(root, i + 1, 0);
16    }
17    ll query(int l, int r){
18        pnode t1, t2, t3;
19        split(root, r + 2, 0, t2, t3);
20        split(t2, l + 1, 0, t1, t2);
21
22        ll res = t2->sum;
23        merge(root, t1, t2);
24        merge(root, root, t3);
25    }

```

```

26     return res;
27 }
28 void update(int l, int r, ll val){
29     pnode t1, t2, t3;
30     split(root, r + 2, 0, t2, t3);
31     split(t2, l + 1, 0, t1, t2);
32
33     t2->add += val;
34     merge(root, t1, t2);
35     merge(root, root, t3);
36 }
37 void split(pnode t, int key, int add, pnode &l,
38 ↪ pnode &r){
39     if (!t){
40         l = r = nullptr;
41         return;
42     }
43     push(t);
44     int impl_key = add + _cnt(t->l);
45     if (key <= impl_key)
46         split(t->l, key, add, l, t->l), r = t;
47     else
48         split(t->r, key, add + _cnt(t->l) + 1, t->r,
49 ↪ r), l = t;
50     upd(t);
51 }
52 void merge(pnode &t, pnode l, pnode r){
53     push(l); push(r);
54     if (!l || !r)
55         t = l ? l : r;
56     else if (l->prior > r->prior)
57         merge(r->l, l, r->l), t = r;
58     else
59         merge(l->r, l->r, r), t = l;
60     upd(t);
61 }
62 private:
63 void _erase(pnode &t, int key, int add){
64     push(t);
65     int impl_key = add + _cnt(t->l);
66     if (impl_key == key){
67         pnode it = t;
68         merge(t, t->l, t->r);
69         delete it;
70     }
71     else if (key < impl_key)
72         _erase(t->l, key, add);
73     else
74         _erase(t->r, key, add + _cnt(t->l) + 1);
75     upd(t);
76 }
77 void push(pnode t){
78     if (!t) return;
79     t->sum += t->add * (ll)_cnt(t);
80     t->val += t->add;
81     if (t->l) t->l->add += t->add;
82     if (t->r) t->r->add += t->add;
83
84     t->add = 0;
85 }
86 int _cnt(pnode t){
87     if (!t) return 0;
88     return t->cnt;
89 }
90 ll _sum(pnode t){
91     if (!t) return 0;
92     push(t);
93     return t->sum;
94 }
95 void upd(pnode t){
96     if (!t) return;
97     t->sum = t->val + _sum(t->l) + _sum(t->r);
98     t->cnt = _cnt(t->l) + _cnt(t->r) + 1;
99 }

```

2.5 Dynamic Segment Tree

Overview

Range queries and updates on larger range ($1 \leq l \leq r \leq 10^9$)

Time complexity: $O(\log M)$ for every operations, where M is max range

Implementation

```

1 struct Node{
2     ll sum, tl, tr;
3     Node *l = nullptr, *r = nullptr;
4
5     Node (ll _tl, ll _tr){
6         tl = _tl;
7         tr = _tr;
8         sum = 0;
9     }
10
11     void extend(){
12         if (tl == tr) return;
13         ll mid = (tl + tr) / 2;
14
15         if (!l)
16             l = new Node(tl, mid);
17         if (!r)
18             r = new Node(mid + 1, tr);
19     }
20 };
21
22 class funkysegtree{
23     void _upd(Node *node, ll x, ll val){
24         node->sum += val;
25         if (node->tl > x || node->tr < x)
26             return;
27         if (node->tl == node->tr)
28             return;
29
30         ll mid = (node->tl + node->tr) / 2;
31         node->extend();
32
33         if (x <= mid)
34             _upd(node->l, x, val);
35         else
36             _upd(node->r, x, val);
37     }
38
39     ll _get(Node *node, ll ql, ll qr){
40         if (qr < node->tl || ql > node->tr)
41             return 0;
42
43         else if (ql <= node->tl && qr >= node->tr)
44             return node->sum;
45
46         ll mid = (node->tl + node->tr) / 2;
47         node->extend();
48
49         if (ql > mid)
50             return _get(node->r, ql, qr);
51         else if (qr <= mid)
52             return _get(node->l, ql, qr);
53         else
54             return _get(node->l, ql, mid) +
55 ↪ _get(node->r, mid + 1, qr);
56     }
57 public:
58     Node *root = nullptr;

```

```

59     ll _size;
60
61     funkysegtree(ll __size){
62         root = new Node(0, __size);
63         _size = __size;
64     };
65
66     void upd(ll x, ll val){
67         _upd(root, x, val);
68     }
69
70     ll get(ll l, ll r){
71         return _get(root, l, r);
72     }
73 };

```

2.6 Persistent Segment Tree

Overview

Preserving history for every segment tree updates.

Time complexity: $\mathcal{O}(\log N)$ for every operations

Implementation

```

1  struct Vertex {
2      Vertex *l, *r;
3      int sum;
4
5      Vertex(int val) : l(nullptr), r(nullptr), sum(val) {}
6      Vertex(Vertex *l, Vertex *r) : l(l), r(r), sum(0) {
7          if (l) sum += l->sum;
8          if (r) sum += r->sum;
9      }
10 };
11
12 Vertex* build(ll a[], int tl, int tr) {
13     if (tl == tr)
14         return new Vertex(a[tl]);
15     int tm = (tl + tr) / 2;
16     return new Vertex(build(a, tl, tm), build(a, tm+1,
17         ↪ tr));
18 }
19
20 int get_sum(Vertex* v, int tl, int tr, int l, int r) {
21     if (l > r)
22         return 0;
23     if (l == tl && tr == r)
24         return v->sum;
25     int tm = (tl + tr) / 2;
26     return get_sum(v->l, tl, tm, l, min(r, tm))
27         + get_sum(v->r, tm+1, tr, max(l, tm+1), r);
28 }
29
30 Vertex* update(Vertex* v, int tl, int tr, int pos, int
31     ↪ new_val) {
32     if (tl == tr)
33         return new Vertex(new_val);
34     int tm = (tl + tr) / 2;
35     if (pos <= tm)
36         return new Vertex(update(v->l, tl, tm, pos,
37             ↪ new_val), v->r);
38     else
39         return new Vertex(v->l, update(v->r, tm+1, tr,
40             ↪ pos, new_val));
41 }

```

Usage

- Init and update segment tree with n nodes, each function returns a pointer, save if needed for later.

```

1  vector<Vertex*> roots;
2  roots.push_back(build(a, 0, n - 1)); //init
3  (...)
4  roots.push_back(update(roots.back(), 0, n - 1, x,
5      ↪ 1)); //update at the last moment
6  (...)
7  roots.push_back(update(roots[a], 0, n - 1, x, 1));
8      ↪ //update at some specific moment

```

- Query the segment tree at a specific moment.

```

1  ll res = get_sum(roots[x], 0, n - 1, l, r);

```

2.7 2D Fenwick Tree

Overview

Query and update on a 2D array.

Time complexity: $\mathcal{O}(\log^2 n)$ for every operations

Implementation

```

1  ll bit[1001][1001];
2  ll n, m;
3
4  void update(ll x, ll y, ll val){
5      for (; y <= n; y += (y & (-y))){
6          for (ll i = x; i <= m; i += (i & (-i)))
7              bit[y][i] += val;
8      }
9  }
10
11 ll query(ll x, ll y){
12     ll res = 0;
13     for (ll i = y; i; i -= (i & (-i)))
14         for (ll j = x; j; j -= (j & (-j)))
15             res += bit[i][j];
16     return res;
17 }
18
19 ll query(ll x1, ll y1, ll x2, ll y2){
20     ll res = query(x2, y2) - query(x1 - 1, y2) -
21         ↪ query(x2, y1 - 1) + query(x1 - 1, y1 - 1);
22     return res;
23 }

```

Usage

- query(x, y) returns sum of value from (1, 1) to (x, y).
- query(x1, y1, x2, y2) returns sum of value from (x1, y1) to (x2, y2).

2.8 Disjoint Set Union

Overview

Union disjoint set lol.

⌚ Time complexity: $\mathcal{O}(\alpha(n))$

🔗 Implementation

```

1 struct DissjointSet{
2     vector<int> p;
3     int cnt = 0;
4
5     DissjointSet(){}
6     DissjointSet(int n){
7         cnt = n;
8         p = vector<int>(n, -1);
9     }
10
11     int find(int n){
12         return p[n] < 0 ? n : p[n] = find(p[n]);
13     }
14
15     void merge(int u, int v){
16         if ((u = find(u)) == (v = find(v)))
17             return;
18
19         cnt--;
20         if (p[v] < p[u])
21             swap(u, v);
22
23         p[u] += p[v];
24         p[v] = u;
25     }
26 };

```

2.9 Line Container

📖 Overview

Add lines of the form $y = kx + m$, and query maximum value at point x .

⌚ Time complexity: $\mathcal{O}(\log n)$

🔗 Implementation

```

1 struct Line {
2     mutable ll k, m, p;
3     bool operator<(const Line& o) const { return k <
4         ↪ 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    ll div(ll a, ll b) { // floored division
11        return a / b - ((a ^ b) < 0 && a % b); }
12    bool isect(iterator x, iterator y) {
13        if (y == end()) return x->p = inf, 0;
14        if (x->k == y->k) x->p = x->m > y->m ?
15            ↪ inf : -inf;
16        else x->p = div(y->m - x->m, x->k -
17            ↪ y->k);
18        return x->p >= y->p;
19    }
20    //add line y = kx + m
21    void add(ll k, ll m) {
22        auto z = insert({k, m, 0}), y = z++, x =
23            ↪ y;
24        while (isect(y, z)) z = erase(z);
25        if (x != begin() && isect(--x, y))
26            ↪ isect(x, y = erase(y));
27        while ((y = x) != begin() && (--x)->p >=
28            ↪ y->p)

```

```

23         isect(x, erase(y));
24     }
25     ll query(ll x) {
26         assert(!empty());
27         auto l = *lower_bound(x);
28         return l.k * x + l.m;
29     }
30 };

```

2.10 Lichao Tree

📖 Overview

Add lines of the form $y = ax + b$, and query maximum value at point x , segment tree implementation.

⌚ Time complexity: $\mathcal{O}(\log n)$

🔗 Implementation

```

1 struct LichaoTree{
2     struct Line{
3         ll a, b;
4         Line() : a(0), b(-inf) {}
5         Line(ll a, ll b): a(a), b(b) {}
6         ll get(ll x){
7             return a * x + b;
8         }
9     };
10    public:
11        vector<Line> st;
12        int n;
13        LichaoTree(int n) : n(n){
14            st.resize(4 * n);
15        }
16        void add_line(Line line, int indx = 1, int l = 0,
17            ↪ int r = -1){
18            if (r == -1) r = n;
19            int m = (l + r) / 2;
20            bool left = line.get(l) > st[indx].get(l);
21            bool mid = line.get(m) > st[indx].get(m);
22
23            if (mid)
24                swap(line, st[indx]);
25            if (r - l == 1) return;
26            else if (left != mid)
27                add_line(line, 2 * indx, l, m);
28            else
29                add_line(line, 2 * indx + 1, m, r);
30        }
31        ll query(ll x, int indx = 1, int l = 0, int r = -1){
32            if (r == -1) r = n;
33            if (r - l == 1) return st[indx].get(x);
34            int mid = (l + r) / 2;
35            if (x < mid)
36                return max(st[indx].get(x), query(x, 2 *
37                    ↪ indx, l, mid));
38            else
39                return max(st[indx].get(x), query(x, 2 *
40                    ↪ indx + 1, mid, r));
41        }
42    };
43 };

```

2.11 Ordered Set

📖 Overview

A set that supports finding k-th maximum value, or getting the order of an element.

🕒 **Time complexity:** $\mathcal{O}(\log n)$, large constant

🔗 Implementation

```
1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3
4 using namespace __gnu_pbds;
5 template<class T> using ordset = tree<T, null_type,
  ↳ less<T>, rb_tree_tag,
  ↳ tree_order_statistics_node_update>;
```

🔍 Usage

- Uses just like a normal set, but with some added functions.

```
1 ordset<int> s;
2 s.insert(1);
3 s.insert(2);
4 s.insert(4);
5 s.find_by_order(0) //Returns 1
6 s.order_of_key(4) //Returns 2
```

2.12 Minimum Stack/Deque

📖 Overview

Maintains minimum value in a stack/deque.

🕒 **Time complexity:** $\mathcal{O}(\alpha(n))$, large constant

🔗 Implementation

```
1 struct minstack {
2     stack<pair<int, int>> st;
3     int getmin() {return st.top().second;}
4     bool empty() {return st.empty();}
5     int size() {return st.size();}
6     void push(int x) {
7         int mn = x;
8         if (!empty()) mn = min(mn, getmin());
9         st.push({x, mn});
10    }
11    void pop() {st.pop();}
12    int top() {return st.top().first;}
13    void swap(minstack &x) {st.swap(x.st);}
14 };
15
16 struct mindeque {
17     minstack l, r, t;
18     void rebalance() {
19         bool f = false;
20         if (r.empty()) {f = true; l.swap(r);}
21         int sz = r.size() / 2;
22         while (sz-- > 0) {t.push(r.top()); r.pop();}
23         while (!r.empty()) {l.push(r.top());
  ↳ r.pop();}
24         while (!l.empty()) {r.push(t.top());
  ↳ t.pop();}
25         if (f) l.swap(r);
26     }
27     int getmin() {
```

```
28         if (l.empty()) return r.getmin();
29         if (r.empty()) return l.getmin();
30         return min(l.getmin(), r.getmin());
31     }
32     bool empty() {return l.empty() && r.empty();}
33     int size() {return l.size() + r.size();}
34     void push_front(int x) {l.push(x);}
35     void push_back(int x) {r.push(x);}
36     void pop_front() {if (l.empty()) rebalance();
  ↳ l.pop();}
37     void pop_back() {if (r.empty()) rebalance();
  ↳ r.pop();}
38     int front() {if (l.empty()) rebalance(); return
  ↳ l.top();}
39     int back() {if (r.empty()) rebalance(); return
  ↳ r.top();}
40     void swap(mindeque &x) {l.swap(x.l);
  ↳ r.swap(x.r);}
41 };
```

2.13 Dynamic Bitset

📖 Overview

Bitset with varied length support. NOTE: This requires relatively new version of GCC, and it might be BUGGED using the shift operator.

🕒 **Time complexity:** $\mathcal{O}(n / 32)$

🔗 Implementation

```
1 #include <tr2/dynamic_bitset>
2 using namespace tr2;
```

🔍 Usage

- Init a dynamic bitset with length n.

```
1 dynamic_bitset<> bs;
2 bs.resize(n);
```

3 Graph

3.1 Graph

📖 Overview

Helper class, some implementations below will use this.

🔗 Implementation

```
1 struct Graph{
2     vector<vector<int>> edg;
3     int n;
4
5     Graph(int n) : n(n){
6         edg = vector<vector<int>>(n, vector<int>());
7     }
8     void add(int u, int v){
```

```

9     edg[u].push_back(v);
10 }
11 void bi_add(int u, int v){
12     edg[u].push_back(v);
13     edg[v].push_back(u);
14 }
15 void clear(){
16     for (int u = 0; u < n; u++){
17         edg[u].clear();
18     }
19 void remove_dup(){
20     for (int u = 0; u < n; u++){
21         sort(edg[u].begin(), edg[u].end());
22         edg[u].erase(unique(edg[u].begin(),
23                               ↪ edg[u].end()), edg[u].end());
24     }
25 };

```

3.2 Tree

Overview

Helper class, some implementations below will use this.

Implementation

```

1 struct Tree{
2     vector<vector<int>> edg;
3     vector<int> par, depth;
4     int n, root;
5
6     Tree(int n, int root) : n(n), root(root){
7         edg = vector<vector<int>>(n, vector<int>());
8     }
9     void add(int u, int v){
10         edg[u].push_back(v);
11         edg[v].push_back(u);
12     }
13     void clear(){
14         for (int u = 0; u < n; u++){
15             edg[u].clear();
16         }
17     void remove_dup(){
18         for (int u = 0; u < n; u++){
19             sort(edg[u].begin(), edg[u].end());
20             edg[u].erase(unique(edg[u].begin(),
21                               ↪ edg[u].end()), edg[u].end());
22         }
23     }
24     void get_info(){
25         par = depth = vector<int>(n, 0);
26         par[root] = -1;
27         dfs(root, -1);
28     }
29     void dfs(int u, int pre){
30         for (int v : edg[u]){
31             if (v == pre) continue;
32             par[v] = u; depth[v] = depth[u] + 1;
33             dfs(v, u);
34         }
35 };

```

3.3 Strongly Connected Components

Overview

Find strongly connected components, compress the graph if needed

🕒 Time complexity: $\mathcal{O}(N)$

Implementation

```

1 struct StronglyConnected{
2     Graph &G;
3     vector<vector<int>> components;
4     vector<int> low, num, new_num;
5     vector<bool> deleted;
6     stack<int> st;
7     int indx, scc, n;
8
9     StronglyConnected(Graph &G) : G(G), n(G.n){
10         low = num = new_num = vector<int>(n, 0);
11         indx = scc = 0;
12         deleted = vector<bool>(n, 0);
13
14         for (int i = 0; i < n; i++){
15             if (!num[i])
16                 dfs(i);
17         }
18     }
19
20     void dfs(int u){
21         low[u] = num[u] = ++indx;
22         st.push(u);
23
24         for (int v : G.edg[u]){
25             if (deleted[v]) continue;
26             if (!num[v]){
27                 dfs(v);
28                 low[u] = min(low[u], low[v]);
29             }
30             else
31                 low[u] = min(low[u], num[v]);
32         }
33
34         if (low[u] == num[u]){
35             int crr = -1;
36             vector<int> cmp;
37
38             while (crr != u){
39                 crr = st.top();
40                 cmp.push_back(crr);
41                 st.pop();
42
43                 new_num[crr] = scc;
44                 deleted[crr] = 1;
45             }
46
47             components.push_back(cmp);
48             scc++;
49         }
50     }
51
52     void compress(){
53         Graph _G(scc);
54         for (int u = 0; u < n; u++){
55             for (int v : G.edg[u]){
56                 int _u = new_num[u], _v = new_num[v];
57                 if (_u != _v)
58                     _G.add(_u, _v);
59             }
60         }
61         G = _G;
62     }
63 };

```


3.4 Bridges and Articulations

Overview

Find bridges and articulations!!

⌚ Time complexity: $\mathcal{O}(N)$

Implementation

```

1 struct BridgeArt{
2     Graph &G;
3     vector<int> low, num, arts;
4     vector<bool> isart;
5     vector<pair<int, int>> bridges;
6     int indx, n;

7     BridgeArt(Graph &G) : G(G), n(G.n){
8         indx = 0;
9         low = num = vector<int>(n, 0);
10        isart = vector<bool>(n, 0);

11        for (int i = 0; i < n; i++){
12            if (!num[i])
13                dfs(i, i);
14        }
15        for (int i = 0; i < n; i++){
16            if (isart[i])
17                arts.push_back(i);
18        }
19    }

20    void dfs(int u, int pre){
21        low[u] = num[u] = ++indx;
22        int cnt = 0;

23        for (int v : G.edg[u]){
24            if (v == pre) continue;
25            if (!num[v]){
26                dfs(v, u);
27                low[u] = min(low[u], low[v]);
28                cnt++;
29                if (u == pre){
30                    if (cnt > 1)
31                        isart[u] = 1;
32                }
33                else{
34                    if (num[u] <= low[v])
35                        isart[u] = 1;
36                }
37                if (num[v] == low[v])
38                    bridges.push_back({u, v});
39            }
40            else
41                low[u] = min(low[u], num[v]);
42        }
43    }
44 }
45 };

```

3.5 Two SAT

Overview

Solve a system of boolean formula, where every clause has exactly two literals.

⌚ Time complexity: $\mathcal{O}(N + M)$, M can be a slowing factor

Implementation

```

1 struct TwoSAT{
2     vector<vector<int>> edg1, edg2;
3     vector<int> scc, res;
4     vector<bool> b;
5     stack<int> topo;
6     int n;

7     TwoSAT(int n) : n(n){
8         edg1 = edg2 = vector<vector<int>>(2 * n);
9         scc = res = vector<int>(2 * n, 0);
10        b = vector<bool>(2 * n, 0);
11    }

12    void dfs1(ll u){
13        b[u] = 1;
14        for (ll v : edg1[u]){
15            if (!b[v])
16                dfs1(v);
17        }
18        topo.push(u);
19    }

20    void dfs2(ll u, ll root){
21        scc[u] = root;
22        for (ll v : edg2[u]){
23            if (scc[v] == -1)
24                dfs2(v, root);
25        }
26    }

27    bool solve(){
28        for (int i = 0; i < 2 * n; i++){
29            scc[i] = -1;
30            if (!b[i])
31                dfs1(i);
32        }

33        int j = 0;
34        while (siz(topo)){
35            ll u = topo.top();
36            topo.pop();

37            if (scc[u] == -1)
38                dfs2(u, j++);
39        }

40        for (int i = 0; i < n; i++){
41            if (scc[i * 2] == scc[i * 2 + 1])
42                return 0;
43            res[i] = scc[i * 2] > scc[i * 2 + 1];
44        }

45        return 1;
46    }

47    void add(int x, bool a, int y, bool b){
48        int X = x * 2 + (a & 1), Y = y * 2 + (b & 1);
49        int _X = x * 2 + 1 - (a & 1), _Y = y * 2 + 1 - (b & 1);

50        edg1[_X].push_back(Y);
51        edg1[_Y].push_back(X);
52        edg2[Y].push_back(_X);
53        edg2[X].push_back(_Y);
54    }
55 }
56 };

```

Usage

- The `add(x, a, y, b)` function add the clause (x OR y), where a, b signify whether x or y is negated or not.
- The `solve()` function returns 1 if there exist a valid assignment, and 0 otherwise. The valid assignment will then be stored in `res`.

3.6 MCMF

Overview

Find a maximum flow with minimum cost, SPFA implementation.

Time complexity: $\mathcal{O}(N^3)$ with a bullshit factor

Implementation

```

1 struct edge{
2     int v;
3     ll cost, capacity;
4     edge* rv;
5     edge(int v, ll cost, ll capacity) : v(v),
6         ↪ cost(cost), capacity(capacity){}
7 };
8
9 struct MCMF{
10     vector<vector<edge*>> edg;
11     vector<pair<int, edge*>> par;
12     vector<ll> dis;
13
14     MCMF(int n){
15         edg = vector<vector<edge*>>(n);
16     }
17     void add_edge(int u, int v, ll capacity, ll cost){
18         edge* e = new edge(v, cost, capacity);
19         edge* re = new edge(u, -cost, 0);
20
21         e->rv = re;
22         re->rv = e;
23
24         edg[u].push_back(e);
25         edg[v].push_back(re);
26     }
27     void spfa(int start){
28         int n = edg.size();
29         auto inq = vec(n, 0);
30         dis = vec(n, inf);
31         par = vector<pair<int, edge*>>(n, {-1, nullptr});
32
33         queue<int> q;
34         q.push(start);
35         dis[start] = 0;
36
37         while (q.size()){
38             int u = q.front(); q.pop();
39             inq[u] = 0;
40
41             for (auto e : edg[u]){
42                 if (e->capacity > 0 && dis[e->v] >
43                     ↪ dis[u] + e->cost){
44                     dis[e->v] = dis[u] + e->cost;
45                     par[e->v] = {u, e};
46
47                     if (!inq[e->v]){
48                         inq[e->v] = 1;
49                         q.push(e->v);
50                     }
51                 }
52             }
53         }
54     }
55 }
```

```

51     }
52 }
53 pl get(int start, int end, ll max_flow = inf){
54     ll flow = 0, cost = 0;
55     while (flow < max_flow){
56         spfa(start);
57         if (dis[end] == inf) break;
58
59         ll f = max_flow - flow;
60         int u = end;
61
62         while (u != start){
63             f = min(f, par[u].y->capacity);
64             u = par[u].x;
65         }
66
67         flow += f;
68         cost += f * dis[end];
69
70         u = end;
71         while (u != start){
72             par[u].y->capacity -= f;
73             par[u].y->rv->capacity += f;
74             u = par[u].x;
75         }
76     }
77     if (flow == max_flow || max_flow == inf)
78         return {flow, cost};
79     else
80         return {-1, -1};
81 }
82 }
```

3.7 Maximum Flow (Dinic)

Overview

Maximum flow using Dinic's algorithm.

Time complexity: $\mathcal{O}(V^2E)$ for general graphs, but in practice $\approx \mathcal{O}(E^{1.5})$

Implementation

```

1 template<int V, class T=long long>
2 class max_flow {
3     static const T INF = numeric_limits<T>::max();
4
5     struct edge {
6         int t, rev;
7         T cap, f;
8     };
9
10    public:
11        vector<edge> adj[V];
12        ll dist[V];
13        int ptr[V];
14
15        bool bfs(int s, int t) {
16            memset(dist, -1, sizeof dist);
17            dist[s] = 0;
18            queue<int> q({s});
19            while (!q.empty() && dist[t] == -1) {
20                int n = q.front();
21                q.pop();
22                for (auto& e : adj[n]) {
23                    if (dist[e.t] == -1 &&
24                        ↪ e.cap != e.f) {
25                        dist[e.t] =
26                            ↪ dist[n] + 1;
27                        q.push(e.t);
28                    }
29                }
30            }
31            return dist[t] != -1;
32        }
33
34        T dfs(int u, int t, T f) {
35            if (u == t) return f;
36            for (int& i = ptr[u]; i < adj[u].size(); i++) {
37                edge& e = adj[u][i];
38                if (dist[e.t] == dist[u] + 1 && e.cap > e.f) {
39                    T ret = dfs(e.t, t, min(f, e.cap - e.f));
40                    e.f += ret;
41                    e.rev->f -= ret;
42                    return ret;
43                }
44            }
45            return 0;
46        }
47
48        T max_flow(int s, int t) {
49            T flow = 0;
50            while (bfs(s, t)) {
51                for (int i = 0; i < adj[s].size(); i++) ptr[s][i] = i;
52                T f = dfs(s, t, INF);
53                flow += f;
54            }
55            return flow;
56        }
57 }
```

```

26         }
27     }
28 }
29 return dist[t] != -1;
30 }
31
32 T augment(int n, T amt, int t) {
33     if (n == t) return amt;
34     for (; ptr[n] < adj[n].size(); ptr[n]++)
35         ↪ {
36             edge& e = adj[n][ptr[n]];
37             if (dist[e.t] == dist[n] + 1 &&
38                 ↪ e.cap != e.f) {
39                 T flow = augment(e.t,
40                     ↪ min(amt, e.cap -
41                     ↪ e.f), t);
42                 if (flow != 0) {
43                     e.f += flow;
44                     adj[e.t][e.rev].f -= flow;
45                     ↪ return flow;
46                 }
47             }
48         }
49     return 0;
50 }
51
52 void add(int u, int v, T cap=1, T rcap=0) {
53     adj[u].push_back({ v, (int)
54         ↪ adj[v].size(), cap, 0 });
55     adj[v].push_back({ u, (int)
56         ↪ adj[u].size() - 1, rcap, 0 });
57 }
58
59 T calc(int s, int t) {
60     T flow = 0;
61     while (bfs(s, t)) {
62         memset(ptr, 0, sizeof ptr);
63         while (T df = augment(s, INF, t))
64             flow += df;
65     }
66     return flow;
67 }
68
69 void clear() {
70     for (int n = 0; n < V; n++)
71         adj[n].clear();
72 }
73
74 };

```

3.8 Maximum Matching (Hopcroft Karp)

Overview

Find maximum matching on bipartite graph.

Time complexity: $\mathcal{O}(m\sqrt{n})$ worst case

Implementation

```

1 struct HopcroftKarp{
2     vector<vector<int>> edg;
3     vector<int> U, V;
4     vector<int> pu, pv;
5     vector<int> dist;
6
7     //NOTE: This graph is 1-indexed!!!
8     HopcroftKarp(int n, int m){
9         edg = vector<vector<int>>(n + 1);
10        for (int i = 0; i < n; i++)
11            U.push_back(i + 1);

```

```

12        for (int i = 0; i < m; i++)
13            V.push_back(i + 1);
14
15        pu = vector<int>(n + 1, 0);
16        pv = vector<int>(m + 1, 0);
17        dist = vector<int>(n + 1, inf);
18    }
19
20    void add_edge(int u, int v){
21        edg[u].push_back(v);
22    }
23
24    bool bfs(){
25        queue<int> q;
26        for (int u : U){
27            if (!pu[u]){
28                q.push(u);
29                dist[u] = 0;
30            }
31
32            else
33                dist[u] = inf;
34        }
35
36        dist[0] = inf;
37        while (q.size() > 0){
38            int u = q.front();
39            q.pop();
40
41            if (dist[u] < dist[0]){
42                for (int v : edg[u]){
43                    if (dist[pv[v]] == inf){
44                        q.push(pv[v]);
45                        dist[pv[v]] = dist[u] + 1;
46                    }
47                }
48            }
49        }
50
51        if (dist[0] == inf)
52            return 0;
53        return 1;
54    }
55
56    bool dfs(ll u){
57        if (u == 0) return 1;
58        for (int v : edg[u]){
59            if (dist[pv[v]] == (dist[u] + 1)){
60                if (dfs(pv[v])){
61                    pu[u] = v;
62                    pv[v] = u;
63                    return 1;
64                }
65            }
66        }
67
68        dist[u] = 0;
69        return 0;
70    }
71
72    int solve(){
73        int res = 0;
74        while (bfs()){
75            for (int u : U){
76                if (!pu[u])
77                    if (dfs(u))
78                        res++;
79            }
80        }
81
82        return res;
83    }
84 };

```

3.9 General Matching (Blossom)

Overview

Find maximum matching on general graph.

⌚ Time complexity: $\mathcal{O}(n^3)$ worst case

Implementation

```

1 struct Matching {
2     int n;
3     vector<vector<int>> g;
4     vector<int> mt;
5     vector<int> is_ev, gr_buf;
6     vector<pi> nx;
7     int st;
8     int group(int x) {
9         if(gr_buf[x] == -1 || is_ev[gr_buf[x]] != st)
10             ↪ return gr_buf[x];
11         return gr_buf[x] = group(gr_buf[x]);
12     }
13     void match(int p, int b) {
14         int d = mt[p];
15         mt[p] = b;
16         if(d == -1 || mt[d] != p) return;
17         if(nx[p].second == -1) {
18             mt[d] = nx[p].first;
19             match(nx[p].first, d);
20         } else {
21             match(nx[p].first, nx[p].second);
22             match(nx[p].second, nx[p].first);
23         }
24     }
25     bool arg() {
26         is_ev[st] = st;
27         gr_buf[st] = -1;
28         nx[st] = pi(-1, -1);
29         queue<int> q;
30         q.push(st);
31         while(q.size()) {
32             int a = q.front();
33             q.pop();
34             for(auto b : g[a]) {
35                 if(b == st) continue;
36                 if(mt[b] == -1) {
37                     mt[b] = a;
38                     match(a, b);
39                     return true;
40                 }
41                 if(is_ev[b] == st) {
42                     int x = group(a), y = group(b);
43                     if(x == y) continue;
44                     int z = -1;
45                     while(x != -1 || y != -1) {
46                         if(y != -1) swap(x, y);
47                         if(nx[x] == pi(a, b)) {
48                             z = x;
49                             break;
50                         }
51                     }
52                     nx[x] = pi(a, b);
53                     x = group(nx[mt[x]].first);
54                 }
55                 for(int v : {group(a), group(b)}) {
56                     while(v != z) {
57                         q.push(v);
58                         is_ev[v] = st;
59                         gr_buf[v] = z;
60                         v = group(nx[mt[v]].first);
61                     }
62                 }
63             } else if(is_ev[mt[b]] != st) {
64                 is_ev[mt[b]] = st;
65                 nx[b] = pi(-1, -1);
66                 nx[mt[b]] = pi(a, -1);

```

```

65         gr_buf[mt[b]] = b;
66         q.push(mt[b]);
67     }
68 }
69 }
70 return false;
71 }
72 Matching(const vector<vector<int>> &g) :
73     ↪ n(int(_g.size())), g(_g), mt(n, -1), is_ev(n,
74     ↪ -1), gr_buf(n), nx(n) {
75         for(st = 0; st < n; st++)
76             if(mt[st] == -1) arg();
77     }
78     vector<pi> max_match() {
79         vector<pi> res;
80         for (int i = 0; i < n; i++){
81             if(i < mt[i])
82                 res.push_back({i, mt[i]});
83         }
84     };

```

4 Math

4.1 Modular Int

Overview

Helper class, some implementations below will use this.

Implementation

```

1 template<ll mod = 1000000007>
2 struct modu{
3     ll val;
4     modu(ll x){
5         val = x;
6         val %= mod;
7         if (val < 0) val += mod;
8     }
9     modu(){ val = 0; }
10
11     operator ll() const { return val; }
12     modu operator+(modu const& other){ return val +
13     ↪ other.val; }
14     modu operator-(modu const& other){ return val -
15     ↪ other.val; }
16     modu operator*(modu const& other){ return val *
17     ↪ other.val; }
18     modu operator/(modu const& other){ return *this *
19     ↪ other.inv(); }
20     modu operator+=(modu const& other) { *this = *this +
21     ↪ other; return *this; }
22     modu operator-=(modu const& other) { *this = *this -
23     ↪ other; return *this; }
24     modu operator*=(modu const& other) { *this = *this *
25     ↪ other; return *this; }
26     modu operator/=(modu const& other) { *this = *this /
27     ↪ other; return *this; }
28     modu operator++(int) {modu tmp = *this; *this += 1;
29     ↪ return tmp;}
30     modu operator++() {*this += 1; return *this;}
31     modu operator--(int) {modu tmp = *this; *this -= 1;
32     ↪ return tmp;}
33     modu operator--() {*this -= 1; return *this;}
34     modu operator~() {return modu(-val);}
35     friend ostream& operator<<(ostream& os, modu const&
36     ↪ m) { return os << m.val; }

```

```

26 friend istream& operator>>(istream& is, modu & m) {
    ↪ return is >> m.val; }
27
28 modu pow(ll x) const{
29     if (x == 0)
30         return 1;
31     if (x % 2 == 0){
32         modu tmp = pow(x / 2);
33         return tmp * tmp;
34     }
35     else
36         return pow(x - 1) * *this;
37 }
38
39 modu inv() const{ return pow(mod - 2); }
40 };

```

4.2 Modular Square Root

Overview

Operations on field

$$\langle u, v \rangle = u + v\sqrt{k} \pmod{p}$$

Implementation

```

1 ll MOD = 999999893;
2 ll sq = 2;
3
4 class EX {
5     int re, im;
6     static int trim(int a) {
7         if (a >= MOD) a -= MOD;
8         if (a < 0) a += MOD;
9         return a;
10    }
11    static int inv(const int a) {
12        int ans = 1;
13        for (int cur = a, p = MOD - 2; p >= 1, cur = 111
14            ↪ * cur * cur % MOD) {
15            if (p&1) ans = 111 * ans * cur % MOD;
16        }
17        return ans;
18    };
19 public:
20    EX(int re = 0, int im = 0) : re(re), im(im) {}
21    EX& operator=(EX oth) { return re = oth.re, im =
22        ↪ oth.im, *this; }
23    int norm() const {
24        return trim((111 * re * re - 111 * sq * im % MOD *
25            ↪ im) % MOD);
26    }
27    EX conj() const {
28        return EX(re, trim(MOD - im));
29    }
30    EX operator*(EX oth) const {
31        return EX((111 * re * oth.re + 111 * sq * im % MOD *
32            ↪ oth.im) % MOD,
33            (111 * re * oth.im + 111 * im * oth.re) %
34            ↪ MOD);
35    };
36    EX operator/(int n) const {
37        return EX(111 * re * inv(n) % MOD, 111 * im * inv(n)
38            ↪ % MOD);
39    }
40    EX operator/(EX oth) const { return *this * oth.conj()
41        ↪ / oth.norm(); }

```

```

35 EX operator+(EX oth) const { return EX(trim(re +
    ↪ oth.re), trim(im + oth.im)); }
36 EX operator-(EX oth) const {
37     return EX(trim(re - oth.re), trim(im - oth.im));
38 }
39 EX pow(long long n) const {
40     EX ans(1);
41     for (EX a = *this; n >= 1, a = a * a) {
42         if (n&1) ans = a * ans;
43     }
44     return ans;
45 }
46 bool operator==(EX oth) const { return re == oth.re
    ↪ and im == oth.im; }
47 bool operator!=(EX oth) const { return not (*this ==
    ↪ oth); }
48 int real() const& { return re; }
49 int imag() const& { return im; }
50 };

```

4.3 Discrete Log

Overview

Given a, b, m , find any x that satisfy

$$a^x = b \pmod{m}$$

● Time complexity: $\mathcal{O}(N \log \log N)$

Implementation

```

1 // Returns minimum x for which a ^ x % m = b % m.
2 int solve(int a, int b, int m) {
3     a %= m, b %= m;
4     int k = 1, add = 0, g;
5     while ((g = gcd(a, m)) > 1) {
6         if (b == k)
7             return add;
8         if (b % g)
9             return -1;
10        b /= g, m /= g, ++add;
11        k = (k * 111 * a / g) % m;
12    }
13
14    int n = sqrt(m) + 1;
15    int an = 1;
16    for (int i = 0; i < n; ++i)
17        an = (an * 111 * a) % m;
18
19    unordered_map<int, int> vals;
20    for (int q = 0, cur = b; q <= n; ++q) {
21        vals[cur] = q;
22        cur = (cur * 111 * a) % m;
23    }
24
25    for (int p = 1, cur = k; p <= n; ++p) {
26        cur = (cur * 111 * an) % m;
27        if (vals.count(cur)) {
28            int ans = n * p - vals[cur] + add;
29            return ans;
30        }
31    }
32    return -1;
33 }

```

4.4 Primate Root

Overview

Given a, n , find g so that for any a such that $\gcd(a, n) = 1$, there exists k such that

$$g^k = a \pmod n$$

Time complexity: $\mathcal{O}(\text{Ans} \cdot \log \phi(n) \cdot \log n)$

Implementation

```

1 int powmod (int a, int b, int p) {
2     int res = 1;
3     while (b)
4         if (b & 1)
5             res = int (res * 1ll * a % p), --b;
6         else
7             a = int (a * 1ll * a % p), b >>= 1;
8     return res;
9 }
10
11 int generator (int p) {
12     vector<int> fact;
13     int phi = p-1, n = phi;
14     for (int i=2; i*i<=n; ++i)
15         if (n % i == 0) {
16             fact.push_back (i);
17             while (n % i == 0)
18                 n /= i;
19         }
20     if (n > 1)
21         fact.push_back (n);
22
23     for (int res=2; res<=p; ++res) {
24         bool ok = true;
25         for (size_t i=0; i<fact.size() && ok; ++i)
26             ok &= powmod (res, phi / fact[i], p) != 1;
27         if (ok) return res;
28     }
29     return -1;
30 }

```

4.5 Euler's Totient Function

Overview

Find $\phi(i)$ for i from 1 to N .

Time complexity: $\mathcal{O}(N \log \log N)$

Implementation

```

1 int phi[def];
2 void phi(int n) {
3     phi[0] = 0;
4     phi[1] = 1;
5     for (int i = 2; i <= n; i++)
6         phi[i] = i - 1;
7
8     for (int i = 2; i <= n; i++)
9         for (int j = 2 * i; j <= n; j += i)
10             phi[j] -= phi[i];
11 }

```

4.6 Chinese Remainder Theorem

Overview

Given a system of congruences

$$a = a_1 \pmod{M_1}, a = a_2 \pmod{M_2}, \dots$$

where M_i might not be pairwise coprime, find any a that satisfy it.

Time complexity: $\mathcal{O}(N \log \max(M_i))$

Implementation

```

1 typedef __int128_t i128;
2 i128 execlid(i128 a, i128 b, i128& x, i128& y){
3     if (b == 0) {
4         x = 1;
5         y = 0;
6         return a;
7     }
8     i128 x1, y1;
9     i128 d = execlid(b, a % b, x1, y1);
10    x = y1;
11    y = x1 - y1 * (a / b);
12    return d;
13 }
14
15 struct CBT{
16     i128 A = 0, M = 0;
17     void add(i128 a, i128 m){
18         a = ((a % m) + m) % m;
19         i128 _M = M;
20         if (M == 0){
21             A = a, M = m;
22             return;
23         }
24         if (A == -1) return;
25         i128 p, q;
26         i128 g = execlid(M, m, p, q);
27         if ((a - A) % g != 0){
28             A = -1, M = -1;
29             return;
30         }
31         i128 mul = (a - A) / g;
32         M = m * M / g;
33         A = ((_M * p * mul + A) % M) + M % M;
34     }
35 };

```

Usage

- The `add(x,y)` function add the condition $a = x \pmod y$.
- If $a \neq -1$, the solution a will satisfy $a = A \pmod M$.

4.7 Extended Euclidean

Overview

Given a, b , find any x, y that satisfy

$$ax + by = \gcd(a, b)$$

Note that the function pass x, y by reference and returns $\gcd(a, b)$.

⌚ Time complexity: $\mathcal{O}(\log n)$

🔗 Implementation

```

1 int extended_euclid(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 = extended_euclid(b, a % b, x1, y1);
9     x = y1;
10    y = x1 - y1 * (a / b);
11    return d;
12 }

```

4.8 Linear Diophantine

📖 Overview

Given a, b, c , find any x, y that satisfy

$$ax + by = c$$

⌚ Time complexity: $\mathcal{O}(\log n)$

🔗 Implementation

```

1 bool find_any_solution(int a, int b, int c, int &x0, int
2     &y0, int &g) {
3     g = extended_euclid(abs(a), abs(b), x0, y0);
4     if (c % g) {
5         return false;
6     }
7     x0 *= c / g;
8     y0 *= c / g;
9     if (a < 0) x0 = -x0;
10    if (b < 0) y0 = -y0;
11    return true;
12 }

```

4.9 Matrix

📖 Overview

Matrix helper class.

🔗 Implementation

```

1 template <typename T>
2 struct Matrix{
3     vector<vector<T>> m;
4     Matrix (vector<vector<T>> &m) : T(m){}
5     Matrix (int r, int c) {
6         m = vector<vector<T>>(r, vector<T>(c));
7     }
8
9     int row() const {return m.size();}
10    int col() const {return m[0].size();}
11
12    static Matrix identity(int n){
13        Matrix res = Matrix(n, n);
14        for (int i = 0; i < n; i++)

```

```

15        res[i][i] = 1;
16        return res;
17    }
18
19    auto & operator [] (int i) { return m[i]; }
20    const auto & operator [] (int i) const { return m[i];
21    ↪ }
22
23    Matrix operator * (const Matrix &b){
24        Matrix a = *this;
25        assert(a.col() == b.row());
26
27        Matrix c(a.row(), b.col());
28        for (int i = 0; i < a.row(); i++)
29            for (int j = 0; j < b.col(); j++)
30                for (int k = 0; k < a.col(); k++)
31                    c[i][j] += a[i][k] * b[k][j];
32        return c;
33    }
34
35    Matrix pow(ll x){
36        assert(row() == col());
37        Matrix crr = *this, res = identity(row());
38        while (x > 0){
39            if (x % 2 == 1)
40                res = res * crr;
41            crr = crr * crr;
42            x /= 2;
43        }
44        return res;
45    }

```

4.10 Miller Rabin Primality Test

📖 Overview

Deterministic implementation of Miller Rabin.

⌚ Time complexity: Should be fast

🔗 Implementation

```

1 ll binpower(ll base, ll e, ll mod) {
2     ll result = 1;
3     base %= mod;
4     while (e) {
5         if (e & 1)
6             result = (__int128_t)result * base % mod;
7         base = (__int128_t)base * base % mod;
8         e >>= 1;
9     }
10    return result;
11 }
12
13 bool check_composite(ll n, ll a, ll d, int s) {
14     ll x = binpower(a, d, n);
15     if (x == 1 || x == n - 1)
16         return false;
17     for (int r = 1; r < s; r++) {
18         x = (__int128_t)x * x % n;
19         if (x == n - 1)
20             return false;
21     }
22     return true;
23 };
24
25 bool MillerRabin(ll n) { // returns true if n is prime,
26     ↪ else returns false.
27     if (n < 2)
28         return false;

```

```

28
29     int r = 0;
30     ll d = n - 1;
31     while ((d & 1) == 0) {
32         d >>= 1;
33         r++;
34     }
35
36     for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
37 ↪ 31, 37}) {
38         if (n == a)
39             return true;
40         if (check_composite(n, a, d, r))
41             return false;
42     }
43     return true;

```

4.11 Fast Fourier Transform

Overview

multiplymod(*A*, *B*, *M*) returns *C* where

$$C[u] = \sum_{i=1}^{|A|} \sum_{j=1}^{|B|} A_i \cdot B_j \mod M \quad (i+j=u)$$

⌚ Time complexity: $\mathcal{O}(n \log n)$

Implementation

```

1  using cpx = complex<double>;
2  const double PI = acos(-1);
3  vector<cpx> roots = {{0, 0}, {1, 0}};
4
5  void ensure_capacity(int min_capacity) {
6      for (int len = roots.size(); len < min_capacity; len
7 ↪ *= 2) {
8          for (int i = len >> 1; i < len; i++) {
9              roots.emplace_back(roots[i]);
10             double angle = 2 * PI * (2 * i + 1 - len) /
11 ↪ (len * 2);
12             roots.emplace_back(cos(angle), sin(angle));
13         }
14     }
15
16     void fft(vector<cpx> &z, bool inverse) {
17         int n = z.size();
18         assert((n & (n - 1)) == 0);
19         ensure_capacity(n);
20         for (unsigned i = 1, j = 0; i < n; i++) {
21             int bit = n >> 1;
22             for (; j >= bit; bit >>= 1)
23                 j -= bit;
24             j += bit;
25             if (i < j)
26                 swap(z[i], z[j]);
27         }
28         for (int len = 1; len < n; len <= 1) {
29             for (int i = 0; i < n; i += len * 2) {
30                 for (int j = 0; j < len; j++) {
31                     cpx root = inverse ? conj(roots[j +
32 ↪ len]) : roots[j + len];
33                     cpx u = z[i + j];
34                     cpx v = z[i + j + len] * root;
35                     z[i + j] = u + v;
36                     z[i + j + len] = u - v;
37                 }
38             }
39         }

```

```

36     }
37 }
38 if (inverse)
39     for (int i = 0; i < n; i++)
40         z[i] /= n;
41 }
42 vector<int> multiply_mod(const vector<int> &a, const
43 ↪ vector<int> &b, int m) {
44     int need = a.size() + b.size() - 1;
45     int n = 1;
46     while (n < need)
47         n <= 1;
48     vector<cpx> A(n);
49     for (size_t i = 0; i < a.size(); i++) {
50         int x = (a[i] % m + m) % m;
51         A[i] = cpx(x & ((1 << 15) - 1), x >> 15);
52     }
53     fft(A, false);
54
55     vector<cpx> B(n);
56     for (size_t i = 0; i < b.size(); i++) {
57         int x = (b[i] % m + m) % m;
58         B[i] = cpx(x & ((1 << 15) - 1), x >> 15);
59     }
60     fft(B, false);
61
62     vector<cpx> fa(n);
63     vector<cpx> fb(n);
64     for (int i = 0, j = 0; i < n; i++, j = n - i) {
65         cpx a1 = (A[i] + conj(A[j])) * cpx(0.5, 0);
66         cpx a2 = (A[i] - conj(A[j])) * cpx(0, -0.5);
67         cpx b1 = (B[i] + conj(B[j])) * cpx(0.5, 0);
68         cpx b2 = (B[i] - conj(B[j])) * cpx(0, -0.5);
69         fa[i] = a1 * b1 + a2 * b2 * cpx(0, 1);
70         fb[i] = a1 * b2 + a2 * b1;
71     }
72
73     fft(fa, true);
74     fft(fb, true);
75     vector<int> res(need);
76     for (int i = 0; i < need; i++) {
77         long long aa = (long long)(fa[i].real() + 0.5);
78         long long bb = (long long)(fb[i].real() + 0.5);
79         long long cc = (long long)(fa[i].imag() + 0.5);
80         res[i] = (aa % m + (bb % m << 15) + (cc % m <<
81 ↪ 30)) % m;
82     }
83     return res;
84 }

```

4.12 OR Convolution

Overview

convoluteor(*A*, *B*) returns *C* where

$$C[u] = \sum_{i=1}^{|A|} \sum_{j=1}^{|B|} A_i \cdot B_j \mod M \quad (i|j=u)$$

⌚ Time complexity: $\mathcal{O}(2^N \cdot N)$

Implementation

```

1  vector<int> convolute_or(vector<int> &a, vector<int> &b){
2      int n = a.size();
3      for (int i = 0; i < n; i++) for (int j = 0; j < (1
4 ↪ << n); j++){
5          if ((j >> i) & 1){
6              a[j] += a[j - (1 << i)];

```



```

6         b[j] += b[j - (1 << i)];
7     }
8 }
9 for (int i = n - 1; i >= 0; i--){
10     for (int j = (1 << n) - 1; j >= 0; j--){
11         if ((j >> i) & 1)
12             a[j] -= a[j - (1 << i)];
13     }
14 }
15 auto c = vector<int>(n, 0);
16 for (int i = n - 1; i < (1 << n); i++){
17     c[i] = a[i] * b[i];
18     for (int i = n - 1; i >= 0; i--){
19         for (int j = (1 << n) - 1; j >= 0; j--){
20             if ((j >> i) & 1)
21                 c[j] -= c[j - (1 << i)];
22         }
23     }
24 }

```

4.13 XOR Convolution

Overview

idk lol.

Implementation

```

1 void xorconv(vector<int> &a,int modul){ // chuyen tu dang
   ↳ bình thường sang dạng đặc biệt, xong cu lay a[i] =
   ↳ b[i] * c[i] ...
2     int n = a.size();
3     for(int m = n/2; m; m/=2){
4         for(int i = 0 ; i < n; i+= 2 * m){
5             for(int j = 0; j <m; ++ j){
6                 int x = a[i + j];
7                 int y = a[i + j + m];
8                 a[i + j] = (x + y)%modul;
9                 a[i + j + m] = (x-y+modul) % modul;
10            }
11        }
12    }
13 }
14 void xorconv2(vector<int> &a,int modul){ // chuyen tu
   ↳ dạng đặc biệt về dạng bình thường => đáp án sau khi
   ↳ fft
15     int n = a.size();
16     for(int m = 1; m<n; m*=2){
17         for(int i = 0 ; i < n; i+= 2 * m){
18             for(int j = 0; j <m; ++ j){
19                 int x = a[i + j];
20                 int y = a[i + j + m];
21                 a[i + j] = (x + y)%modul;
22                 a[i + j + m] = (x-y+modul) % modul;
23            }
24        }
25    }
26     for(int i = 0; i<n; ++i){
27         a[i] = 1LL * (1LL)a[i] * binpow(n,modul - 2,
   ↳ modul) %modul;
28     }
29 }

```

5 String

5.1 Rolling Hash

Overview

Rolling hash implementation, use multiple mod if necessary.

Time complexity: $\mathcal{O}(N)$

Implementation

```

1 struct hashu{
2     ll n;
3     vector<dd> p, h;
4
5     hashu(string s){
6         n = s.size();
7         p = vector<dd>(n + 1);
8         h = vector<dd>(n + 1);
9
10        p[0] = {1, 1};
11        for (int i = 1; i <= n; i++)
12            p[i] = p[i - 1] * base;
13        for (int i = 1; i <= n; i++)
14            h[i] = (h[i - 1] * base + (s[i]
   ↳ - 1] - '0'));
15    }
16
17    dd get(ll l, ll r){
18        return h[r + 1] - (h[l] * p[r - l + 1]);
19    }
20 };

```

5.2 Z-Function

Overview

Return an array where the i -th element corresponds to the longest sub-string starting from i that matches the prefix of s .

Time complexity: $\mathcal{O}(N)$

Implementation

```

1 vector<int> z_func(string s){
2     int n = s.size();
3     vector<int> v(n);
4
5     int l = 0, r = 0;
6     for (int i = 1; i < n; i++){
7         if (i < r)
8             v[i] = min(r - i, v[i - l]);
9         while ((v[i] + i) < n && s[v[i]] ==
   ↳ s[v[i] + i])
10             v[i]++;
11         if ((v[i] + i) > r)
12             l = i, r = v[i] + i;
13    }
14
15    return v;
16 }

```

5.3 Prefix Function

Overview

Return an array where the i -th element corresponds to the longest sub-string ending at i that matches the prefix of s .

Time complexity: $\mathcal{O}(N)$

Implementation

```
1 vector<int> pref_func(string s){
2     int n = siz(s);
3     vector<int> v(n);
4
5     for (int i = 1; i < n; i++){
6         ll j = v[i - 1];
7         while (j > 0 && s[j] != s[i])
8             j = v[j - 1];
9         if (s[j] == s[i])
10            j++;
11        v[i] = j;
12    }
13
14    return v;
15 }
```

5.4 Manacher's Algorithm

Overview

Return an array where the i -th element corresponds to the longest palindrome that has i as the center, note that the algorithm only works for odd length palindrome, even can also be easily handled by inserting a dummy character in every even indicies.

Time complexity: $\mathcal{O}(N)$

Implementation

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

5.5 Aho-Corasick

Overview

Construct an automaton of Trie nodes, where $dp[i][c]$ is the next state of i when adding character c . If no state exists, we repeatedly go through the next longest available suffix j of i , and try to get $dp[j][c]$.

Time complexity: $\mathcal{O}(M * K)$, where M is the number of nodes in the Trie, and K is the alphabet size

Implementation

```
1 struct node{
2     int p[26];
3     int link;
4
5     node(){
6         for (int i = 0; i < 26; i++){
7             p[i] = -1;
8         }
9     };
10
11 struct Trie{
12     int indx = 1;
13     int dp[def][26];
14     vector<node> p;
15
16     Trie(){
17         p.push_back(node());
18     }
19
20     int add(string s){
21         ll crr = 0;
22         for (int i = 0; i < s.size(); i++){
23             int c = s[i] - 'a';
24             if (p[crr].p[c] == -1){
25                 p[crr].p[c] = indx++;
26                 p.push_back(node());
27             }
28
29             crr = p[crr].p[c];
30         }
31
32         return crr;
33     }
34
35     void buildsuffix(){
36         int n = p.size();
37
38         queue<int> q;
39         q.push(0);
40
41         p[0].link = 0;
42         for (int i = 0; i < n; i++) for (int j = 0; j <
43             ↳ 26; j++){
44             dp[i][j] = 0;
45
46             while (q.size()){
47                 int u = q.front();
48                 q.pop();
49
50                 for (int i = 0; i < 26; i++){
51                     int v = p[u].p[i];
52                     if (v != -1){
53                         dp[u][i] = v;
54                         p[v].link = (u == 0)? 0 :
55                             ↳ dp[p[u].link][i];
56                         q.push(v);
57                     }
58                     else
59                         dp[u][i] = dp[p[u].link][i];
60                 }
61             }
62         }
63     }
64 }
```

62 };