

Teamnote of 2mic1cup

swishy, lenhanbo, hungcubu

Last updated on December 11, 2025



Contents

1	Helpers	2	5	String	14
1.1	Stress Tester	2	5.1	Rolling Hash	14
1.2	Stress Tester (Linux)	2	5.2	Z-Function	15
1.3	Random	2	5.3	Prefix Function	15
1.4	Hungbucu's amazing crazy diabolical template	3	5.4	Suffix Array	15
			5.5	Manacher's Algorithm	15
			5.6	Aho-Corasick	15
2	Data Structure	3	6	Tree	16
2.1	Iterative Segment Tree	3	6.1	Tree	16
2.2	Lazy Segment Tree	3	6.2	Lowest Common Ancestor	16
2.3	Sparse Table	4	6.3	Heavy Light Decomposition	16
2.4	Implicit Treap	4	6.4	Centroid Decomposition	17
2.5	Dynamic Segment Tree	5			
2.6	Persistent Segment Tree	5	7	Geometry (Kactl)	17
2.7	2D Fenwick Tree	6	7.1	Kactl template	17
2.8	Disjoint Set Union	6	7.2	Point	17
2.9	Line Container	6	7.3	CCW	17
2.10	Lichao Tree	6	7.4	Circle Intersection	17
2.11	Ordered Set	6	7.5	Circle Line	18
2.12	Minimum Stack/Deque	7	7.6	Circle Polygon	18
2.13	Dynamic Bitset	7	7.7	Circle Tagents	18
			7.8	Circum Circle	18
3	Graph	7	7.9	Closest pair of points	18
3.1	Graph	7	7.10	Convex Hull	18
3.2	Strongly Connected Components	7	7.11	Hull Diameter	19
3.3	Bridges and Articulations	8	7.12	Point inside Hull	19
3.4	Eulerian Path/Cycle	8	7.13	Point on Segment	19
3.5	Two SAT	8	7.14	Segment Distance	19
3.6	MCMF	9	7.15	Segment Intersection	19
3.7	Maximum Flow (Dinic)	9	7.16	Line Distance	19
3.8	Maximum Matching (Hopcroft Karp)	9	7.17	Line Intersection	19
3.9	General Matching (Blossom)	10	7.18	Line Projection	20
			7.19	Line-Hull Intersection	20
4	Math	10	7.20	Polygon Area	20
4.1	Modular Int	10	7.21	Polygon Center	20
4.2	Modular Square Root	11	7.22	Polygon Union	20
4.3	Discrete Log	11	7.23	Polygon Cut	21
4.4	Primitive Root	11	7.24	Point inside Polygon	21
4.5	Euler's Totient Function	12	7.25	Minkowski Sum	21
4.6	Chinese Remainder Theorem	12	7.26	Manhattan MST	21
4.7	Extended Euclidean	12			
4.8	Linear Diophantine	12	8	Notes	21
4.9	Matrix	12	8.1	Finding min cut	22
4.10	Miller Rabin Primality Test	13	8.2	Finding minimum vertex cover on bipartite graph (König's theorem)	22
4.11	Gaussian Elimination	13	8.3	Bitwise	22
4.12	Fast Fourier Transform	13	8.4	CRT	22
4.13	Floor Sum	14	8.5	Cayley Theorem	22
4.14	OR Convolution	14	8.6	Divisor	22
4.15	XOR Convolution	14	8.7	Heap-like permutation	22
			8.8	Little Fermat	22

8.9	Lucas Theorem	22
8.10	Mex	22
8.11	Modulo	22
8.12	Number Theory	22
8.13	Number Formulas	22

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 OUTPUT
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 Stress Tester (Linux)

Overview

Simple .sh file for stress testing.

Implementation

```

1 set -e
2 g++ -std=c++20 code.cpp -o code
3 g++ -std=c++20 gen.cpp -o gen
4 g++ -std=c++20 brute.cpp -o brute
5 for((i = 1; ; ++i)); do
6     ./gen $i > input_file
7     ./code < input_file > myAnswer
8     ./brute < input_file > correctAnswer
9     if diff -Z myAnswer correctAnswer > /dev/null; then
10         echo -e "$Passed test: $i"
11     else
12         echo -e "$Failed test: $i"
13         echo -e "$WA on the following test:$"
14         cat input_file
15         echo -e "$Your answer is:$"
16         cat myAnswer
17         echo -e "$Correct answer is:$"
18         cat correctAnswer
19         break
20     fi
21 done

```

1.3 Random

Overview

Self explanatory.

Implementation

```

1 #define uid(a, b) uniform_int_distribution<long long>(a, b)(rng)
2 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

```

? Usage

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

1.4 Hungbucu's amazing crazy diabolical template

Overview

Bro only contribution

Implementation

```

1 #include <bits/stdc++.h>
2 #include <ext/pb_ds/assoc_container.hpp>
3
4 #define TASK "test"
5 #define ll long long
6 #define ull unsigned ll
7 #define db long double
8 #define pLL pair<ll, ll>
9 #define pLI pair<ll, int>
10 #define pII pair<int, ll>
11 #define pII pair<int, int>
12 #define vec vector
13 #define vL vec<ll>
14 #define vVL vec<vL>
15 #define vI vec<int>
16 #define vVI vec<vI>
17 #define vvVI vec<vvI>
18 #define vvvVI vec<vvvI>
19 #define vD vec<db>
20 #define vuD vec<vD>
21 #define vLL vec<pLL>
22 #define vLI vec<pLI>
23 #define vIL vec<pII>
24 #define vII vec<pII>
25 #define vvII vec<vII>
26 #define vS vec<string>
27 #define vvS vec<vS>
28 #define vB vec<bool>
29 #define vvB vec<vB>
30 #define umap unordered_map
31 #define gphash gp_hash_table
32 #define mset multiset
33 #define pque priority_queue
34 #define all(a) a.begin(), a.end()
35 #define rall(a) a.rbegin(), a.rend()
36 #define stl(a, n) a.begin(), a.begin() + n
37 #define stf(a, n) a.begin() + n, a.end()
38 #define eb emplace_back
39 #define pb push_back
40 #define pf push_front
41 #define popb pop_back
42 #define popf pop_front
43 #define ins insert
44 #define asg assign
45 #define rev reverse
46 #define fi first
47 #define se second
48 #define th third
49 #define ub upper_bound
50 #define lb lower_bound
51 #define ite iterator
52 #define fs(n) fixed << setprecision(n)
53
54 using namespace std;
55 using namespace __gnu_pbds;
56
57 const ll llINF = 1e18;
58 const int intINF = 1e9;
59 const ll MOD = 1e9 + 7;
60
61 template< class T >
62 using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
63   ↳ tree_order_statistics_node_update>;
64 #define oset ordered_set
65
66 template< class A,
67   class B,
68   class C > struct triple {
69     A fi; B se; C th;
70
71     triple() {}
72     triple(A a, B b, C c) : fi(a), se(b), th(c) {}
73 };
74 #define tIII triple<int, int, int>
75 #define tLLL triple<ll, ll, ll>
76 #define vIII vec<tIII>
77 #define vvIII vec<vIII>
78 #define vLLL vec<tLLL>
79
80 mt19937
81   ↳ rd(chrono::high_resolution_clock::now().time_since_epoch().count());
82 ll Rand(ll L, ll R) { return uniform_int_distribution<ll>(L, R)(rd); }
83
84 inline int read() {
85     char c; while (c = getchar(), c < '0' || c > '9'); int n = c - '0';
86     while (c = getchar(), c >= '0' && c <= '9') n = 10 * n + c - '0';
87     return n; }
88
89 vI prime, lpf;
90 void primeSieve(int n) { prime.ans(1, 2); lpf.ans(n + 1, 2); lpf[0] = lpf[1]
91   ↳ = 1;
92     for (int i = 3; i <= n; i += 2) { if (lpf[i] == 2) {
93       ↳ lpf[i] = i; prime.pb(i); }
94     for (int j = 0; j < prime.size() && i * prime[j] <=
95       ↳ n && prime[j] <= lpf[i]; ++ j) lpf[i * prime[j]]
96       ↳ = prime[j];

```

```

91     } }
92     dvs;
93     void dvsSieve(int n) { dvs.ans(n + 1, vI());
94       for (int i = 1; i <= n; ++ i) {
95         for (int j = i; j <= n; j += i)
96           dvs[j].pb(i);
97       } }
98
99     template< class T > bool maximize(T &a, T b) { if (b > a) return a = b, 1;
100   ↳ return 0; }
101     template< class T > bool minimize(T &a, T b) { if (b < a) return a = b, 1;
102   ↳ return 0; }
103
104     ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a; }
105     ll lcm(ll a, ll b) { return a / gcd(a, b) * b; }
106
107     ll fastPow(ll n, ll p,
108       ll m = MOD) { ll r = 1; for (n %= m; p >= 1) { if (p & 1) (r
109   ↳ *= n) %= m; (n *= n) %= m; } return r; }
110     ll invMod(ll n,
111       ll m = MOD) { return fastPow(n, m - 2, m); }
112
113     ll mask(int i) { return i < 0 ? 0 : 1LL << i; }
114     bool bit(ll n, int i) { return n >> i & 1LL; }
115     #define popcount __builtin_popcountll
116     #define clz __builtin_clzll
117     #define ctz __builtin_ctzll

```

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     }
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, T v){
38         if (qr < l || ql > r)
39             return;
40
41         if (l >= ql && r <= qr){
42             st[crr] += (r - l + 1) * v;
43             lazy[crr] += v;
44             return;
45         }
46
47         push(crr, l, r);
48         int mid = (l + r) / 2;
49         _update(l, mid, crr * 2 + 1, ql, qr, v);
50         _update(mid + 1, r, crr * 2 + 2, ql, qr, v);
51
52         st[crr] = cmb(st[crr * 2 + 1], st[crr * 2 + 2]);
53     }
54
55     T _get(int l, int r, int ql, int qr, int crr){
56         if (qr < l || ql > r)
57             return defv;
58         if (l >= ql && r <= qr)
59             return st[crr];
60
61         push(crr, l, r);
62         int mid = (l + r) / 2;
63         return cmb(_get(l, mid, ql, qr, crr * 2 + 1), _get(mid + 1, r, ql, qr,
64             ↵ crr * 2 + 2));
65     };

```

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 &, const T &)>>
2  struct SparseTable{
3      vector<vector<T>> f;
4      vector<int> lg;
5      Combine cmb;
6      int n;
7
8      SparseTable(vector<T> &init, const Combine &cmb) : 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 << (j - 1))][j - 1]);
20             }
21         }
22
23     T get(int l, int r){
24         int k = lg[r - l + 1];

```

```

24         return cmb(f[l][k], f[r - (1 << k) + 1][k]);
25     }
26 };

```

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 &l1, pnode &r){
38         if (!t){
39             l1 = r = nullptr;
40             return;
41         }
42         push(t);
43         int impl_key = add + _cnt(t->l);
44         if (key <= impl_key)
45             split(t->l, key, add, l1, t->l), r = t;
46         else
47             split(t->r, key, add + _cnt(t->l) + 1, t->r, r), l1 = t;
48         upd(t);
49     }
50
51     void merge(pnode &t, pnode l, pnode r){
52         push(l); push(r);
53         if (!l || !r)
54             t = l ? l : r;
55         else if (l->prior > r->prior)
56             merge(r->l, l, r->l), t = r;
57         else
58             merge(l->r, l->r, r), t = l;
59         upd(t);
60     }
61 private:
62     void _erase(pnode &t, int key, int add){
63         push(t);
64         int impl_key = add + _cnt(t->l);
65         if (impl_key == key){
66             pnode it = t;
67             merge(t, t->l, t->r);
68             delete it;

```

```

69     }
70     else if (key < impl_key)
71         _erase(t->l, key, add);
72     else
73         _erase(t->r, key, add + _cnt(t->l) + 1);
74     upd(t);
75 }
76 void push(pnode t){
77     if (!t) return;
78     t->sum += t->add * (ll)_cnt(t);
79     t->val += t->add;
80     if (t->l) t->l->add += t->add;
81     if (t->r) t->r->add += t->add;
82
83     t->add = 0;
84 }
85 int _cnt(pnode t){
86     if (!t) return 0;
87     return t->cnt;
88 }
89 ll _sum(pnode t){
90     if (!t) return 0;
91     push(t);
92     return t->sum;
93 }
94 void upd(pnode t){
95     if (!t) return;
96     t->sum = t->val + _sum(t->l) + _sum(t->r);
97     t->cnt = _cnt(t->l) + _cnt(t->r) + 1;
98 }
99 };

```

2.5 Dynamic Segment Tree

Overview

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

⌚ Time complexity: $\mathcal{O}(\log M)$ 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) + _get(node->r, mid + 1, qr);
55     }
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, tr));
17 }
18
19 int get_sum(Vertex* v, int tl, int tr, int l, int r) {
20     if (l > r)
21         return 0;
22     if (l == tl && tr == r)
23         return v->sum;
24     int tm = (tl + tr) / 2;
25     return get_sum(v->l, tl, tm, l, min(r, tm))
26         + get_sum(v->r, tm+1, tr, max(l, tm+1), r);
27 }
28
29 Vertex* update(Vertex* v, int tl, int tr, int pos, int new_val) {
30     if (tl == tr)
31         return new Vertex(new_val);
32     int tm = (tl + tr) / 2;
33     if (pos <= tm)
34         return new Vertex(update(v->l, tl, tm, pos, new_val), v->r);
35     else
36         return new Vertex(v->l, update(v->r, tm+1, tr, pos, new_val));
37 }

```

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, 1)); //update at the last
5     ↪ moment
6 (...)
7 roots.push_back(update(roots[a], 0, n - 1, x, 1)); //update at some specific
8     ↪ 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) - query(x2, y1 - 1) +
21         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 DisjointSet{
2     vector<int> p;
3     int cnt = 0;
4
5     DisjointSet(){
6         DisjointSet(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 < o.k; }
4     bool operator<(ll x) const { return p < x; }
5 };
6
7 struct LineContainer : multiset<Line, less<>> {
8     // (for doubles, use inf = 1/.0, div(a,b) = a/b)
9     ll div(ll a, ll b) { // floored division
10         return a / b - ((a ^ b) < 0 && a % b); }
11     bool isect(iterator x, iterator y) {
12         if (y == end()) return x->p = inf, 0;
13         if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
14         else x->p = div(y->m - x->m, x->k - y->k);
15         return x->p >= y->p;
16     }
17     //add line y = kx + m
18     void add(ll k, ll m) {
19         auto z = insert({k, m, 0}), y = z++, x = y;
20         while (isect(y, z)) z = erase(z);
21         if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
22         while ((y = x) != begin() && (--x)->p >= 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, int r = -1){
17         if (r == -1) r = n;
18         int m = (l + r) / 2;
19         bool left = line.get(l) > st[indx].get(l);
20         bool mid = line.get(m) > st[indx].get(m);
21
22         if (mid)
23             swap(line, st[indx]);
24         if (r - l == 1) return;
25         else if (left != mid)
26             add_line(line, 2 * indx, l, m);
27         else
28             add_line(line, 2 * indx + 1, m, r);
29     }
30     ll query(ll x, int indx = 1, int l = 0, int r = -1){
31         if (r == -1) r = n;
32         if (r - l == 1) return st[indx].get(x);
33         int mid = (l + r) / 2;
34         if (x < mid)
35             return max(st[indx].get(x), query(x, 2 * indx, l, mid));
36         else
37             return max(st[indx].get(x), query(x, 2 * indx + 1, mid, r));
38     }
39 };
```

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,
6 ↪ 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 (!l.empty()) {l.push(r.top()); r.pop();}
24         while (!t.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    }
20    void remove_dup(){
21        for (int u = 0; u < n; u++){
22            sort(edg[u].begin(), edg[u].end());
23            edg[u].erase(unique(edg[u].begin(), edg[u].end()), edg[u].end());
24        }
25    };
26 };
```

3.2 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 };

```

3.3 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
8     BridgeArt(Graph &G) : G(G), n(G.n){
9         indx = 0;
10        low = num = vector<int>(n, 0);
11        isart = vector<bool>(n, 0);
12
13        for (int i = 0; i < n; i++){
14            if (!num[i])
15                dfs(i, i);
16        }
17        for (int i = 0; i < n; i++){
18            if (isart[i])
19                arts.push_back(i);
20        }
21    }
22
23    void dfs(int u, int pre){
24        low[u] = num[u] = ++indx;
25        int cnt = 0;
26
27        for (int v : G.edg[u]){
28            if (v == pre) continue;
29            if (!num[v]){
30                dfs(v, u);
31                low[u] = min(low[u], low[v]);
32                cnt++;
33                if (u == pre){
34                    if (cnt > 1)
35                        isart[u] = 1;
36                }
37            }
38            else{
39                if (num[u] <= low[v])
40                    isart[u] = 1;
41            }
42            if (num[v] == low[v])
43                bridges.push_back({u, v});
44        }
45        else
46            low[u] = min(low[u], num[v]);
47    }
48 };

```

3.4 Eulerian Path/Cycle

Overview

Find a path/cycle that traverse every edges exactly once.

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

Implementation

```

1 vector<pl> edg[def];
2 vector<ll> path;
3 bool b[def];
4
5 void dfs(ll u){
6     while (siz(edg[u])){
7         auto [v, indx] = edg[u].back();
8         edg[u].pop_back();
9
10        if (b[indx])
11            continue;
12        b[indx] = 1;
13        dfs(v);
14    }
15    path.push_back(u);
16 }

```

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
8     TwoSAT(int n) : n(n){
9         edg1 = edg2 = vector<vector<int>>(2 * n);
10        scc = res = vector<int>(2 * n, 0);
11        b = vector<bool>(2 * n, 0);
12    }
13
14    void dfs1(ll u){
15        b[u] = 1;
16        for (ll v : edg1[u]){
17            if (!b[v])
18                dfs1(v);
19        }
20        topo.push(u);
21    }
22
23    void dfs2(ll u, ll root){
24        scc[u] = root;
25        for (ll v : edg2[u]){
26            if (scc[v] == -1)
27                dfs2(v, root);
28        }
29    }
30
31    bool solve(){
32        for (int i = 0; i < 2 * n; i++){
33            scc[i] = -1;
34            if (!b[i])
35                dfs1(i);
36        }
37
38        int j = 0;
39        while (siz(topo)){
40            ll u = topo.top();
41            topo.pop();
42
43            if (scc[u] == -1)
44                dfs2(u, j++);
45        }
46
47        for (int i = 0; i < n; i++){
48            if (scc[i * 2] == scc[i * 2 + 1])
49                return 0;
50            res[i] = scc[i * 2] > scc[i * 2 + 1];
51        }
52
53        return 1;
54    }
55
56    void add(int x, bool a, int y, bool b){
57        int X = x * 2 + (a & 1), Y = y * 2 + (b & 1);
58        int _X = x * 2 + 1 - (a & 1), _Y = y * 2 + 1 - (b & 1);
59
60        edg1[_X].push_back(Y);
61        edg1[_Y].push_back(X);
62        edg2[Y].push_back(_X);
63    }

```



```

64     edg2[X].push_back(_Y);
65 }
66 };

```

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), cost(cost), capacity(capacity){}
6 };
7
8 struct MCMF{
9     vector<vector<edge*>> edg;
10    vector<pair<int, edge*>> par;
11    vector<ll> dis;
12
13    MCMF(int n){
14        edg = vector<vector<edge*>>(n);
15    }
16    void add_edge(int u, int v, ll capacity, ll cost){
17        edge* e = new edge(v, cost, capacity);
18        edge* re = new edge(u, -cost, 0);
19
20        e->rv = re;
21        re->rv = e;
22
23        edg[u].push_back(e);
24        edg[v].push_back(re);
25    }
26    void spfa(int start){
27        int n = edg.size();
28        auto inq = vec(n, 0);
29        dis = vec(n, inf);
30        par = vector<pair<int, edge*>>(n, {-1, nullptr});
31
32        queue<int> q;
33        q.push(start);
34        dis[start] = 0;
35
36        while (q.size()){
37            int u = q.front(); q.pop();
38            inq[u] = 0;
39
40            for (auto e : edg[u]){
41                if (e->capacity > 0 && dis[e->v] > dis[u] + e->cost){
42                    dis[e->v] = dis[u] + e->cost;
43                    par[e->v] = {u, e};
44
45                    if (!inq[e->v]){
46                        inq[e->v] = 1;
47                        q.push(e->v);
48                    }
49                }
50            }
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 && e.cap != e.f) {
24                        dist[e.t] = dist[n] + 1;
25                        q.push(e.t);
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                edge& e = adj[n][ptr[n]];
36                if (dist[e.t] == dist[n] + 1 && e.cap != e.f) {
37                    T flow = augment(e.t, min(amt, e.cap - e.f), t);
38                    if (flow != 0) {
39                        e.f += flow;
40                        adj[e.t][e.rev].f -= flow;
41                        return flow;
42                    }
43                }
44            }
45            return 0;
46        }
47
48        void add(int u, int v, T cap=1, T rcap=0) {
49            adj[u].push_back({ v, (int) adj[v].size(), cap, 0 });
50            adj[v].push_back({ u, (int) adj[u].size() - 1, rcap, 0 });
51        }
52
53        T calc(int s, int t) {
54            T flow = 0;
55            while (bfs(s, t)) {
56                memset(ptr, 0, sizeof ptr);
57                while (T df = augment(s, INF, t))
58                    flow += df;
59            }
60            return flow;
61        }
62
63        void clear() {
64            for (int n = 0; n < V; n++)
65                adj[n].clear();
66        }
67 };

```

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) return gr_buf[x];
10        return gr_buf[x] = group(gr_buf[x]);
11    }
12    void match(int p, int b) {
13        int d = mt[p];
14        mt[p] = b;
15        if (d == -1 || mt[d] != p) return;
16        if (nx[p].second == -1) {
17            mt[d] = nx[p].first;
18            match(nx[p].first, d);
19        } else {
20            match(nx[p].first, nx[p].second);
21            match(nx[p].second, nx[p].first);
22        }
23    }
24    bool arg() {
25        is_ev[st] = st;
26        gr_buf[st] = -1;
27        nx[st] = pi(-1, -1);
28        queue<int> q;
29        q.push(st);
30        while (q.size()) {
31            int a = q.front();
32            q.pop();
33            for (auto b : g[a]) {
34                if (b == st) continue;
35                if (mt[b] == -1) {
36                    mt[b] = a;
37                    match(a, b);
38                    return true;
39                }
40                if (is_ev[b] == st) {
41                    int x = group(a), y = group(b);
42                    if (x == y) continue;
43                    int z = -1;
44                    while (x != -1 || y != -1) {
45                        if (y != -1) swap(x, y);
46                        if (nx[x] == pi(a, b)) {
47                            z = x;
48                            break;
49                        }
50                        nx[x] = pi(a, b);
51                        x = group(nx[mt[x]].first);
52                    }
53                    for (int v : {group(a), group(b)}) {
54                        while (v != z) {
55                            q.push(v);
56                            is_ev[v] = st;
57                            gr_buf[v] = z;
58                            v = group(nx[mt[v]].first);
59                        }
60                    }
61                } else if (is_ev[mt[b]] != st) {
62                    is_ev[mt[b]] = st;
63                    nx[b] = pi(-1, -1);
64                    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) : n(int(g.size())), g(g), mt(n,
73        ↪ -1), is_ev(n, -1), gr_buf(n), nx(n) {
74        for (st = 0; st < n; st++)
75            if (mt[st] == -1) arg();
76    }
77    vector<pi> max_match() {
78        vector<pi> res;
79        for (int i = 0; i < n; i++){
80            if (i < mt[i])
81                res.push_back({i, mt[i]});
82        }
83        return res;
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 + other.val; }
13     modu operator-(modu const& other){ return val - other.val; }
14     modu operator*(modu const& other){ return val * other.val; }
15     modu operator/(modu const& other){ return *this * other.inv(); }
16     modu operator+=(modu const& other) { *this = *this + other; return *this; }
17     modu operator-=(modu const& other) { *this = *this - other; return *this; }
18     modu operator*=(modu const& other) { *this = *this * other; return *this; }
19     modu operator/=(modu const& other) { *this = *this / other; return *this; }
20     modu operator++(int) {modu tmp = *this; *this += 1; return tmp;}
21     modu operator++() {*this += 1; return *this;}
22     modu operator--(int) {modu tmp = *this; *this -= 1; return tmp;}
23     modu operator--() {*this -= 1; return *this;}
24     modu operator-() {return modu(-val);}
25     friend ostream& operator<<(ostream& os, modu const& 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 * cur * cur % MOD) {
14             if (p&1) ans = 111 * ans * cur % MOD;
15         }
16         return ans;
17     };
18     public:
19     EX(int re = 0, int im = 0) : re(re), im(im) {}
20     EX& operator=(EX oth) { return re = oth.re, im = oth.im, *this; }
21     int norm() const {
22         return trim((111 * re * re - 111 * sq * im % MOD * im) % MOD);
23     }
24     EX conj() const {
25         return EX(re, trim(MOD - im));
26     }
27     EX operator*(EX oth) const {
28         return EX((111 * re * oth.re + 111 * sq * im % MOD * oth.im) % MOD,
29             (111 * re * oth.im + 111 * im * oth.re) % MOD);
30     };
31     EX operator/(int n) const {
32         return EX(111 * re * inv(n) % MOD, 111 * im * inv(n) % MOD);
33     }
34     EX operator/(EX oth) const { return *this * oth.conj() / oth.norm(); }
35     EX operator+(EX oth) const { return EX(trim(re + oth.re), trim(im + oth.im)); }
36     EX operator-(EX oth) const { return EX(trim(re - oth.re), trim(im - oth.im)); }
37     };
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 }
34
35 //lenhanboo version
36 ll DiscreteLog(ll a, ll b, ll m) { // a^x = b (mod m)
37     const int B = 35000;
38     ll k = 1 % m, ans = 0, g;
39     while ((g = gcd(a, m)) > 1) {
40         if (b == k) return ans;
41         if (b % g) return -1;
42         b /= g, m /= g, ans++, k = (k * a / g) % m;
43     }
44     if (b == k) return ans;
45     unordered_map<ll, int> m1;
46     ll tot = 1;
47     for (int i = 0; i < B; ++i)
48         m1[tot * b % m] = i, tot = tot * a % m;
49     ll cur = k * tot % m;
50     for (int i = 1; i <= B; ++i, cur = cur * tot % m)
51         if (m1.count(cur))
52             return 111 * i * B - m1[cur] + ans;
53     return -1;
54 }

```

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

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 * 111 * a % p), --b;
6          else
7              a = int (a * 111 * 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
2 // To find any solution for ax + by = c
3 int extended_euclid(int a, int b, int& x, int& y) {
4     if (b == 0) {
5         x = 1;
6         y = 0;
7         return a;
8     }
9     int x1, y1;
10    int d = extended_euclid(b, a % b, x1, y1);
11    x = y1;
12    y = x1 - y1 * (a / b);
13    return d;
14 }

```

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 &y0, int &g) {
2     g = extended_euclid(abs(a), abs(b), x0, y0);
3     if (c % g) {
4         return false;
5     }
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        Matrix operator * (const Matrix &b){
23            Matrix a = *this;
24            assert(a.col() == b.row());
25
26            Matrix c(a.row(), b.col());
27            for (int i = 0; i < a.row(); i++)
28                for (int j = 0; j < b.col(); j++)
29                    for (int k = 0; k < a.col(); k++)
30                        c[i][j] += a[i][k] * b[k][j];
31            return c;
32        }
33
34        Matrix pow(ll x){
35            assert(row() == col());
36            Matrix crr = *this, res = identity(row());
37            while (x > 0){
38                if (x % 2 == 1)
39                    res = res * crr;
40                crr = crr * crr;
41                x /= 2;
42            }
43            return res;
44        }
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, else returns false.
26     if (n < 2)
27         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, 31, 37}) {
37         if (n == a)
38             return true;
39         if (check_composite(n, a, d, r))
40             return false;
41     }
42     return true;
43 }

```

4.11 Guassian Elimination

Overview

Solve system of n equations with m unknowns.

Time complexity: $\mathcal{O}(\min(n, m) \cdot nm)$

Implementation

```

1 //find solution of ax = b (mod prime m)
2 vector<modu> gay(vector<vector<modu>> &a, vector<modu> &b){
3     int n = a.size(), m = a[0].size();
4     for (int i = 0; i < n; i++){
5         a[i].push_back(b[i]);
6         auto pos = vec(m, -1);
7         for (int col = 0, row = 0; col < m && row < n; col++){
8             int epic = -1;
9             for (int i = row; i < n; i++){
10                if (a[i][col])
11                    epic = i;
12            }
13            if (epic == -1)
14                continue;
15            pos[col] = row;
16            for (int i = col; i <= m; i++)
17                swap(a[row][i], a[epic][i]);
18            for (int i = 0; i < n; i++){
19                if (i != row){
20                    modu val = a[i][col] * a[row][col].inv();
21                    for (int j = col; j <= m; j++)
22                        a[i][j] -= a[row][j] * val;
23                }
24            }
25            row++;
26        }
27        vector<modu> res(m, 0);
28        for (int i = 0; i < m; i++){
29            if (pos[i] != -1)
30                res[i] = a[pos[i]][m] * a[pos[i]][i].inv();
31        }
32        for (int i = 0; i < n; i++){
33            modu sum = 0;
34            for (int j = 0; j < m; j++)
35                sum += res[j] * a[i][j];
36            if (sum != a[i][m])
37                return {};
38        }
39        return res;
40    }

```

4.12 Fast Fourier Transform

Overview

$\text{multiplymod}(A, B, M)$ returns C where

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

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 *= 2) {
7         for (int i = len >> 1; i < len; i++) {
8             roots.emplace_back(roots[i]);
9             double angle = 2 * PI * (2 * i + 1 - len) / (len * 2);
10            roots.emplace_back(cos(angle), sin(angle));
11        }
12    }
13 }
14
15 void fft(vector<cpx> &z, bool inverse) {
16     int n = z.size();
17     assert((n & (n - 1)) == 0);
18     ensure_capacity(n);
19     for (unsigned i = 1, j = 0; i < n; i++) {
20         int bit = n >> 1;
21         for (; j >= bit; bit >>= 1)
22             j -= bit;
23         j += bit;
24         if (i < j)
25             swap(z[i], z[j]);
26     }

```

```

27     for (int len = 1; len < n; len <= 1) {
28         for (int i = 0; i < n; i += len * 2) {
29             for (int j = 0; j < len; j++) {
30                 cpx root = inverse ? conj(roots[j + len]) : roots[j + len];
31                 cpx u = z[i + j];
32                 cpx v = z[i + j + len] * root;
33                 z[i + j] = u + v;
34                 z[i + j + len] = u - v;
35             }
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 vector<int> &b, int m) {
43     int need = a.size() + b.size() - 1;
44     int n = 1;
45     while (n < need)
46         n <= 1;
47     vector<cpx> A(n);
48     for (size_t i = 0; i < a.size(); i++) {
49         int x = (a[i] % m + m) % m;
50         A[i] = cpx(x & ((1 << 15) - 1), x >> 15);
51     }
52     fft(A, false);
53
54     vector<cpx> B(n);
55     for (size_t i = 0; i < b.size(); i++) {
56         int x = (b[i] % m + m) % m;
57         B[i] = cpx(x & ((1 << 15) - 1), x >> 15);
58     }
59     fft(B, false);
60
61     vector<cpx> fa(n);
62     vector<cpx> fb(n);
63     for (int i = 0, j = 0; i < n; i++, j = n - i) {
64         cpx a1 = (A[i] + conj(A[j])) * cpx(0.5, 0);
65         cpx a2 = (A[i] - conj(A[j])) * cpx(0, -0.5);
66         cpx b1 = (B[i] + conj(B[j])) * cpx(0.5, 0);
67         cpx b2 = (B[i] - conj(B[j])) * cpx(0, -0.5);
68         fa[i] = a1 * b1 + a2 * b2 * cpx(0, 1);
69         fb[i] = a1 * b2 + a2 * b1;
70     }
71
72     fft(fa, true);
73     fft(fb, true);
74     vector<int> res(need);
75     for (int i = 0; i < need; i++) {
76         long long aa = (long long)(fa[i].real() + 0.5);
77         long long bb = (long long)(fb[i].real() + 0.5);
78         long long cc = (long long)(fa[i].imag() + 0.5);
79         res[i] = (aa % m + (bb % m << 15) + (cc % m << 30)) % m;
80     }
81     return res;
82 }

```

4.13 Floor Sum

Overview

$sum_0^{n-1} \text{floor}((a * i + b) / m)$ (only works for $a, b \geq 0!!!$)

Time complexity: $\mathcal{O}(\log(n + m + a + b))$

Implementation

```

1 ll floor_sum(ll n, ll m, ll a, ll b) {
2     ll ans = 0;
3     if (a >= m) ans += (n - 1) * n * (a / m) / 2, a %= m;
4     if (b >= m) ans += n * (b / m), b %= m;
5     ll y_max = (a * n + b) / m, x_max = (y_max * m - b);
6     if (y_max == 0) return ans;
7     ans += (n - (x_max + a - 1) / a) * y_max;
8     ans += floor_sum(y_max, a, m, (a - x_max % a) % a);
9     return ans;
10 }

```

4.14 OR Convolution

Overview

$\text{convoluteor}(A, B)$ 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 << n); j++){
4         if ((j >> i) & 1){
5             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 }
25 // Don gian la dung dp sos de tinh cho A[i] va B[i]
26 // Sau do C[i] = a[i] * b[i]
27 // Luc nay dao nguoc dp sos de co C[i] voi moi i thay vi la toan bo subset cua i

```

4.15 XOR Convolution

Overview

- We want to transform $a[x] \rightarrow b[x]$ to perform pure multiple operation.
- $a[x]$ with contribute $a[x]$ to $b[y]$ if $\text{bitcount}(x \& y) \mod 2 = 0$
- $a[x]$ with contribute $-a[x]$ to $b[y]$ if $\text{bitcount}(x \& y) \mod 2 = 1$
- So we can use dp sos to compute $\text{dp}[\text{mask}][j] = \text{number of } y \text{ that } \text{bitcount}(\text{mask} \& y) == j$ with principle of inclusive and exclusive.

Implementation

```

1 void xorconv(vector<int> &a, int modul) { // chuyen tu dang binh thuong sang dang
2     // dac biet, zong cu lay a[i] = b[i] * c[i] ...
3     int n = a.size();
4     for (int m = n/2; m; m/=2){
5         for (int i = 0; i < n; i += 2 * m){
6             for (int j = 0; j < m; ++j){
7                 int x = a[i + j];
8                 int y = a[i + j + m];
9                 a[i + j] = (x + y) % modul;
10                a[i + j + m] = (x - y + modul) % modul;
11            }
12        }
13    }
14 }
15 void xorconv2(vector<int> &a, int modul) { // chuyen tu dang dac biet ve dang binh
16     // thuong => dap an sau khi fft
17     int n = a.size();
18     for (int m = 1; m < n; m *= 2){
19         for (int i = 0; i < n; i += 2 * m){
20             for (int j = 0; j < m; ++j){
21                 int x = a[i + j];
22                 int y = a[i + j + m];
23                 a[i + j] = (x + y) % modul;
24                 a[i + j + m] = (x - y + modul) % modul;
25             }
26        }
27    }
28    for (int i = 0; i < n; ++i){
29        a[i] = 1LL * (1LL) a[i] * binpow(n, modul - 2, modul) % modul;
30    }
31 }

```

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 substring 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 substring 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 Suffix Array

📖 Overview

Return the lexicographic order of all suffixes of s .

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

🔗 Implementation

```
1 int sa[N], tmp[2][N], c[N], rk[N], lcp[N];
2 void buildSA(string s) { // sort in ascending order O(nlog)
3     int *x = tmp[0], *y = tmp[1], m = 256, n = s.size();
4     for (int i = 0; i < m; ++i) c[i] = 0;
5     for (int i = 0; i < n; ++i) c[x[i]] = s[i]++;
6     for (int i = 1; i < m; ++i) c[i] += c[i - 1];
7     for (int i = n - 1; ~i; --i) sa[--c[x[i]]] = i;
8     for (int k = 1; k < n; k <= 1) {
9         for (int i = 0; i < m; ++i) c[i] = 0;
10        for (int i = 0; i < n; ++i) c[x[i]]++;
11        for (int i = 1; i < m; ++i) c[i] += c[i - 1];
12        int p = 0;
13        for (int i = n - k; i < n; ++i) y[p++] = i;
14        for (int i = 0; i < n; ++i) if (sa[i] >= k)
15            y[p++] = sa[i] - k;
16        for (int i = n - 1; ~i; --i)
17            sa[--c[x[y[i]]]] = y[i];
18        y[sa[0]] = p = 0;
19        for (int i = 1; i < n; ++i) {
20            int a = sa[i], b = sa[i - 1];
21            if (! (x[a] == x[b] && a + k < n && b + k < n && x[a + k] == x[b + k])) p++;
22            y[sa[i]] = p;
23        }
24        if (n == p + 1) break;
25        swap(x, y), m = p + 1;
26    }
27 }
28 void buildLCP(string s) { O(n)
29     // lcp[i] = LCP(sa[i - 1], sa[i])
30     // lcp(i, j) = query_lcp_min [rk[i] + 1, rk[j] + 1)
31     int n = s.length(), val = 0;
32     for (int i = 0; i < n; ++i) rk[sa[i]] = i;
33     for (int i = 0; i < n; ++i) {
34         if (!rk[i]) lcp[rk[i]] = 0;
35         else {
36             if (val) val--;
37             int p = sa[rk[i] - 1];
38             while (val + i < n && val + p < n && s[val + i] == s[val + p]) val++;
39             lcp[rk[i]] = val;
40         }
41     }
42 }
```

5.5 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 indices.

⌚ 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 }
```

5.6 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 < 26; j++){
43             dp[i][j] = 0;
44
45             while (q.size()){
46                 int u = q.front();
47                 q.pop();
48
49                 for (int i = 0; i < 26; i++){
50                     int v = p[u].p[i];
51                     if (v != -1){
52                         dp[u][i] = v;
53                         p[v].link = (u == 0)? 0 : dp[p[u].link][i];
54                         q.push(v);
55                     }
56
57                     else
58                         dp[u][i] = dp[p[u].link][i];
59                 }
60             }
61         }
62 };

```

6 Tree

6.1 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(), edg[u].end()), edg[u].end());
21         }
22     }
23     void get_info(){
24         par = depth = vector<int>(n, 0);
25         par[root] = -1;
26         dfs(root, -1);

```

```

27     }
28     void dfs(int u, int pre){
29         for (int v : edg[u]){
30             if (v == pre) continue;
31             par[v] = u; depth[v] = depth[u] + 1;
32             dfs(v, u);
33         }
34     }
35 };

```

6.2 Lowest Common Ancestor

Overview

Uses binary lifting to find the k-th parent of a node.

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

Implementation

```

1 struct LCA{
2     vector<vector<int>> f;
3     Tree T;
4     int n, k;
5
6     LCA(Tree &T) : T(_T){
7         n = T.n; k = log2(n) + 2;
8         for (int i = 0; i < n; i++){
9             f.push_back(vector<int>(k, -1));
10            T.get_info();
11
12            for (int i = 0; i < n; i++){
13                f[i][0] = T.par[i];
14                for (int j = 1; j < k; j++) for (int i = 0; i < n; i++){
15                    int p = f[i][j - 1];
16                    if (p != -1)
17                        f[i][j] = f[p][j - 1];
18                }
19            }
20
21            int get(int u, int v){
22                if (T.depth[u] < T.depth[v])
23                    swap(u, v);
24                for (int i = k - 1; i >= 0; i--){
25                    if (f[u][i] != -1 && T.depth[f[u][i]] >= T.depth[v])
26                        u = f[u][i];
27                }
28                if (u == v) return u;
29                for (int i = k - 1; i >= 0; i--){
30                    if (f[u][i] != -1 && f[u][i] != f[v][i])
31                        u = f[u][i], v = f[v][i];
32                }
33                return T.par[u];
34            }
35 };

```

6.3 Heavy Light Decomposition

Overview

Clean implementation of HLD, only uses 1 segment, $pos[u]$ is the position of u on the segment. Change the query function if needed, for now it's just max query using a segment tree

Time complexity: $\mathcal{O}(n \log n)$ for build, $\mathcal{O}(\log^2 n)$ for query

Implementation

```

1 struct HLD{
2     vector<int> head, par, h, pos, big;
3     int n, indx = 0;
4     Tree T;
5
6     HLD(Tree &T) : T(_T){
7         n = T.n;
8         head = par = h = pos = big = vector<int>(n, 0);
9         dfs(0, -1);
10        decompose(0, 0, -1);
11    }
12    int dfs(int u, int pre){
13        ll res = 1;
14        big[u] = -1;
15
16        int crr_size = 0;
17        for (int v : T.edg[u]){
18            if (v == pre)
19                continue;
20
21            par[v] = u; h[v] = h[u] + 1;
22            int child_size = dfs(v, u);
23

```



```

24         if (child_size > crr_size)
25             big[u] = v, crr_size = child_size;
26         res += child_size;
27     }
28
29     return res;
30 }
31 void decompose(int u, int root, int pre){
32     head[u] = root, pos[u] = indx++;
33     if (big[u] != -1)
34         decompose(big[u], root, u);
35     for (int v : T.edg[u]){
36         if (v == pre || v == big[u])
37             continue;
38         decompose(v, v, u);
39     }
40 }
41 ll query(int u, int v){
42     ll res = -inf;
43     while (head[u] != head[v]){
44         if (h[head[u]] < h[head[v]])
45             swap(u, v);
46         maxi(res, st.get(pos[head[u]], pos[u]));
47         u = par[head[u]];
48     }
49
50     if (h[u] < h[v])
51         swap(u, v);
52     maxi(res, st.get(pos[v], pos[u]));
53
54     return res;
55 }
56 };

```

6.4 Centroid Decomposition

Overview

Uses the centroid of a tree to decompose into smaller subtrees, each node will be recursively decomposed in $\mathcal{O}(\log)$ times.

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

Implementation

```

1 vector<ll> edg[def];
2 bool dead[def];
3 ll cnt[def];
4
5 void dfs(ll u, ll pre){
6     cnt[u] = 1;
7     for (ll v : edg[u]){
8         if (v == pre || dead[v])
9             continue;
10        dfs(v, u);
11        cnt[u] += cnt[v];
12    }
13 }
14
15 ll centroid(ll u, ll pre, ll n){
16     for (ll v : edg[u]){
17         if (v == pre || dead[v])
18             continue;
19         if (cnt[v] > (n / 2))
20             return centroid(v, u, n);
21     }
22     return u;
23 }
24 long long get(ll u){
25     dfs(u, -1);
26     ll root = centroid(u, -1, cnt[u]);
27     dead[root] = 1;
28
29     for (ll v : edg[root]){
30         if (!dead[v])
31             get(v);
32     }
33     return res;
34 }

```

7 Geometry (Kactl)

7.1 Kactl template

Overview

Kactl implementation sometimes use their own template, reference this for clarity.

Implementation

```

1 #define rep(i, a, b) for(int i = a; i < (b); ++i)
2 #define all(x) begin(x), end(x)
3 #define sz(x) (int)(x).size()
4 typedef long long ll;
5 typedef pair<int, int> pii;
6 typedef vector<int> vi;

```

7.2 Point

Overview

Helper class, some implementations below will use this.

Implementation

```

1 template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
2 template<class T>
3 struct Point {
4     typedef Point P;
5     T x, y;
6     explicit Point(T x=0, T y=0) : x(x), y(y) {}
7     bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
8     bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
9     P operator+(P p) const { return P(x+p.x, y+p.y); }
10    P operator-(P p) const { return P(x-p.x, y-p.y); }
11    P operator*(T d) const { return P(x*d, y*d); }
12    P operator/(T d) const { return P(x/d, y/d); }
13    T dot(P p) const { return x*p.x + y*p.y; }
14    T cross(P p) const { return x*p.y - y*p.x; }
15    T cross(P a, P b) const { return (a-*this).cross(b-*this); }
16    T dist2() const { return x*x + y*y; }
17    double dist() const { return sqrt((double)dist2()); }
18    // angle to x-axis in interval [-pi, pi]
19    double angle() const { return atan2(y, x); }
20    P unit() const { return *this/dist(); } // makes dist()=1
21    P perp() const { return P(-y, x); } // rotates +90 degrees
22    P normal() const { return perp().unit(); }
23    // returns point rotated 'a' radians ccw around the origin
24    P rotate(double a) const {
25        return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
26    friend ostream& operator<<(ostream& os, P p) {
27        return os << "(" << p.x << ", " << p.y << ")"; }
28 };

```

7.3 CCW

Overview

- Returns where p is as seen from s towards e . $1/0/-1 \Leftrightarrow$ left/on line/right.
- If the optional argument eps is given 0 is returned if p is within distance eps from the line.

Implementation

```

1 template<class P>
2 int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }
3
4 template<class P>
5 int sideOf(const P& s, const P& e, const P& p, double eps) {
6     auto a = (e-s).cross(p-s);
7     double l = (e-s).dist()*eps;
8     return (a > l) - (a < -l);
9 }

```

7.4 Circle Intersection

Overview

Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

Time complexity: $\mathcal{O}(1)$

Implementation

```

1 typedef Point<double> P;
2 bool circleInter(P a, P b, double r1, double r2, pair<P, P>* out) {
3     if (a == b) { assert(r1 != r2); return false; }
4     P vec = b - a;
5     double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
6     p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
7     if (sum*sum < d2 || dif*dif > d2) return false;
8     P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
9     *out = {mid + per, mid - per};
10    return true;
11 }

```

7.5 Circle Line

Overview

Finds the intersection between a circle and a line. Returns a vector of either 0, 1, or 2 intersection points.

Time complexity: $\mathcal{O}(1)$

Implementation

```

1 template<class P>
2 vector<P> circleLine(P c, double r, P a, P b) {
3     P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
4     double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
5     if (h2 < 0) return {};
6     if (h2 == 0) return {p};
7     P h = ab.unit() * sqrt(h2);
8     return {p - h, p + h};
9 }

```

7.6 Circle Polygon

Overview

Returns the area of the intersection of a circle with a ccw polygon.

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

Implementation

```

1 #define arg(p, q) atan2(p.cross(q), p.dot(q))
2 double circlePoly(P c, double r, vector<P> ps) {
3     auto tri = [&](P p, P q) {
4         auto r2 = r * r / 2;
5         P d = q - p;
6         auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
7         auto det = a * a - b;
8         if (det <= 0) return arg(p, q) * r2;
9         auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
10        if (t < 0 || 1 <= s) return arg(p, q) * r2;
11        P u = p + d * s, v = q + d * (t-1);
12        return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
13    };
14    auto sum = 0.0;
15    rep(i, 0, sz(ps))
16        sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
17    return sum;
18 }

```

7.7 Circle Tangents

Overview

- Finds the external tangents of two circles, or internal if r2 is negated.
- Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers).
- .first and .second give the tangency points at circle 1 and 2 respectively.
- To find the tangents of a circle with a point set r2 to 0.

Time complexity: $\mathcal{O}(1)$

Implementation

```

1 template<class P>
2 vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2) {
3     P d = c2 - c1;
4     double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
5     if (d2 == 0 || h2 < 0) return {};
6     vector<pair<P, P>> out;
7     for (double sign : {-1, 1}) {
8         P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
9         out.push_back({c1 + v * r1, c2 + v * r2});
10    }
11    if (h2 == 0) out.pop_back();
12    return out;
13 }

```

7.8 Circum Circle

Overview

The circumcircle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.

Time complexity: $\mathcal{O}(1)$

Implementation

```

1 typedef Point<double> P;
2 double ccRadius(const P& A, const P& B, const P& C) {
3     return (B-A).dist()*(C-B).dist()*(A-C).dist()/
4     abs((B-A).cross(C-A))/2;
5 }
6 P ccCenter(const P& A, const P& B, const P& C) {
7     P b = C-A, c = B-A;
8     return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
9 }

```

7.9 Closest pair of points

Overview

Finds the closest pair of points.

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

Implementation

```

1 typedef Point<ll> P;
2 pair<P, P> closest(vector<P> v) {
3     assert(sz(v) > 1);
4     set<P> S;
5     sort(all(v), [](P a, P b) { return a.y < b.y; });
6     pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
7     int j = 0;
8     for (P p : v) {
9         P d{1 + (ll)sqrt(ret.first), 0};
10        while (v[j].y <= p.y - d.x) S.erase(v[j++]);
11        auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
12        for (; lo != hi; ++lo)
13            ret = min(ret, {(ll)p.dist2(), {*lo, p}});
14        S.insert(p);
15    }
16    return ret.second;
17 }

```

7.10 Convex Hull

Overview

Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.

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

Implementation

```

1 typedef Point<ll> P;
2 vector<P> convexHull(vector<P> pts) {
3     if (sz(pts) <= 1) return pts;
4     sort(all(pts));
5     vector<P> h(sz(pts)+1);
6     int s = 0, t = 0;
7     for (int it = 2; it--; s = --t, reverse(all(pts)))
8         for (P p : pts) {
9             while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
10            h[t++] = p;
11        }
12    return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
13 }

```

7.11 Hull Diameter

Overview

Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).

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

Implementation

```

1 typedef Point<ll> P;
2 array<P, 2> hullDiameter(vector<P> S) {
3     int n = sz(S), j = n < 2 ? 0 : 1;
4     pair<ll, array<P, 2>> res{{0, {S[0], S[0]}}};
5     rep(i, 0, j)
6         for (; j = (j + 1) % n) {
7             res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
8             if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
9                 break;
10        }
11    return res.second;
12 }

```

7.12 Point inside Hull

Overview

- Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.

- NOTE:** Requires **7.12** and **7.2**.

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

Implementation

```

1 bool inHull(const vector<P>& l, P p, bool strict = true) {
2     int a = 1, b = sz(l) - 1, r = !strict;
3     if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
4     if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
5     if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p) <= -r)
6         return false;
7     while (abs(a - b) > 1) {
8         int c = (a + b) / 2;
9         (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
10    }
11    return sgn(l[a].cross(l[b], p)) < r;
12 }

```

7.13 Point on Segment

Overview

Returns true iff p lies on the line segment from s to e . Use $segDist(s, e, p) \leq \epsilon$ instead when using `Point < double >`.

Time complexity: $\mathcal{O}(1)$

Implementation

```

1 template<class P> bool onSegment(P s, P e, P p) {
2     return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
3 }

```

7.14 Segment Distance

Overview

Returns the shortest distance between point p and the line segment from point s to e .

Time complexity: $\mathcal{O}(1)$

Implementation

```

1 template<class P> bool onSegment(P s, P e, P p) {
2     return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
3 }

```

7.15 Segment Intersection

Overview

- If a unique intersection point between the line segments going from s_1 to e_1 and from s_2 to e_2 exists then it is returned.

- If no intersection point exists an empty vector is returned.

- If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment.

- NOTE:** Requires **7.12**.

Time complexity: $\mathcal{O}(1)$

Implementation

```

1 template<class P> vector<P> segInter(P a, P b, P c, P d) {
2     auto oa = c.cross(d, a), ob = c.cross(d, b),
3         oc = a.cross(b, c), od = a.cross(b, d);
4     // Checks if intersection is single non-endpoint point.
5     if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
6         return {(a * ob - b * oa) / (ob - oa)};
7     set<P> s;
8     if (onSegment(c, d, a)) s.insert(a);
9     if (onSegment(c, d, b)) s.insert(b);
10    if (onSegment(a, b, c)) s.insert(c);
11    if (onSegment(a, b, d)) s.insert(d);
12    return {all(s)};
13 }

```

7.16 Line Distance

Overview

- Returns the signed distance between point p and the line containing points a and b .

- Positive value on left side and negative on right as seen from a towards b . $a=b$ gives nan.

Time complexity: $\mathcal{O}(1)$

Implementation

```

1 template<class P>
2 double lineDist(const P& a, const P& b, const P& p) {
3     return (double)(b-a).cross(p-a)/(b-a).dist();
4 }

```

7.17 Line Intersection

Overview

- If a unique intersection point of the lines going through s_1, e_1 and s_2, e_2 exists $\{1, \text{point}\}$ is returned.

- If no intersection point exists $\{0, (0,0)\}$ is returned and if infinitely many exists $\{-1, (0,0)\}$ is returned.

🕒 Time complexity: $\mathcal{O}(1)$

🔗 Implementation

```
1 template<class P>
2 pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
3     auto d = (e1 - s1).cross(e2 - s2);
4     if (d == 0) // if parallel
5         return {(s1.cross(e1, s2) == 0), P(0, 0)};
6     auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
7     return {1, (s1 * p + e1 * q) / d};
8 }
```

7.18 Line Projection

📖 Overview

- Projects point p onto line ab . Set `refl=true` to get reflection of point p across line ab instead.
- The wrong point will be returned if P is an integer point and the desired point doesn't have integer coordinates.

🕒 Time complexity: $\mathcal{O}(1)$

🔗 Implementation

```
1 template<class P>
2 P lineProj(P a, P b, P p, bool refl=false) {
3     P v = b - a;
4     return p - v.perp() * (1 + refl) * v.cross(p - a) / v.dist2();
5 }
```

7.19 Line-Hull Intersection

📖 Overview

Line-convex polygon intersection. The polygon must be ccw and have no collinear points. `lineHull(line, poly)` returns a pair describing the intersection of a line with the polygon:

- $(-1, -1)$ if no collision,
- $(i, -1)$ if touching the corner i ,
- (i, i) if along side $(i, i + 1)$,
- (i, j) if crossing sides $(i, i + 1)$ and $(j, j + 1)$.

In the last case, if a corner i is crossed, this is treated as happening on side $(i, i + 1)$. The points are returned in the same order as the line hits the polygon. `extrVertex` returns the point of a hull with the max projection onto a line.

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

🔗 Implementation

```
1 #define cmp(i, j) sgn(dir.perp().cross(poly[(i)%n] - poly[(j)%n]))
2 #define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
3 template <class P> int extrVertex(vector<P>& poly, P dir) {
4     int n = sz(poly), lo = 0, hi = n;
5     if (extr(0)) return 0;
6     while (lo + 1 < hi) {
7         int m = (lo + hi) / 2;
8         if (extr(m)) return m;
9         int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
10        (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
11    }
12    return lo;
13 }
14
15 #define cmpl(i) sgn(a.cross(poly[i], b))
16 template <class P>
17 array<int, 2> lineHull(P a, P b, vector<P>& poly) {
18     int endA = extrVertex(poly, (a - b).perp());
19     int endB = extrVertex(poly, (b - a).perp());
20     if (cmpl(endA) < 0 || cmpl(endB) > 0)
21         return {-1, -1};
22     array<int, 2> res;
23     rep(i, 0, 2) {
24         int lo = endB, hi = endA, n = sz(poly);
25         while ((lo + 1) % n != hi) {
26             int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
27             (cmpl(m) == cmpl(endB) ? lo : hi) = m;
```

```
28         }
29         res[i] = (lo + !cmpl(hi)) % n;
30         swap(endA, endB);
31     }
32     if (res[0] == res[1]) return {res[0], -1};
33     if (!cmpl(res[0]) && !cmpl(res[1]))
34         switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
35             case 0: return {res[0], res[0]};
36             case 2: return {res[1], res[1]};
37         }
38     return res;
39 }
```

7.20 Polygon Area

📖 Overview

Returns twice the signed area of a polygon.

🕒 Time complexity: $\mathcal{O}(n)$

🔗 Implementation

```
1 template<class T>
2 T polygonArea2(vector<Point<T>>& v) {
3     T a = v.back().cross(v[0]);
4     rep(i, 0, sz(v)-1) a += v[i].cross(v[i+1]);
5     return a;
6 }
```

7.21 Polygon Center

📖 Overview

Returns the center of mass for a polygon.

🕒 Time complexity: $\mathcal{O}(n)$

🔗 Implementation

```
1 typedef Point<double> P;
2 P polygonCenter(const vector<P>& v) {
3     P res(0, 0); double A = 0;
4     for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
5         res = res + (v[i] + v[j]) * v[j].cross(v[i]);
6         A += v[j].cross(v[i]);
7     }
8     return res / A / 3;
9 }
```

7.22 Polygon Union

📖 Overview

Calculates the area of the union of n polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to `sideOf/sgn`, but shouldn't be needed)

🕒 Time complexity: $\mathcal{O}(n^2)$, where n is the total number of points

🔗 Implementation

```
1 typedef Point<double> P;
2 double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
3 double polyUnion(vector<vector<P>>& poly) {
4     double ret = 0;
5     rep(i, 0, sz(poly)) rep(v, 0, sz(poly[i])) {
6         P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
7         vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
8         rep(j, 0, sz(poly)) if (i != j) {
9             rep(u, 0, sz(poly[j])) {
10                P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
11                int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
12                if (sc != sd) {
13                    double sa = C.cross(D, A), sb =
14                        C.cross(D, B);
15                    if (min(sc, sd) < 0)
16                        segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
17                } else if (!sc && !sd && j < i &&
18                    sgn((B-A).dot(D-C)) > 0) {
```

```

17         segs.emplace_back(rat(C - A, B - A), 1);
18         segs.emplace_back(rat(D - A, B - A), -1);
19     }
20 }
21 }
22 sort(all(segs));
23 for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
24 double sum = 0;
25 int cnt = segs[0].second;
26 rep(j, 1, sz(segs)) {
27     if (!cnt) sum += segs[j].first - segs[j - 1].first;
28     cnt += segs[j].second;
29 }
30 ret += A.cross(B) * sum;
31 }
32 return ret / 2;
33 }

```

7.23 Polygon Cut

Overview

Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.

⌚ Time complexity: $\mathcal{O}(n)$

Implementation

```

1 typedef Point<double> P;
2 vector<P> polygonCut(const vector<P>& poly, P s, P e) {
3     vector<P> res;
4     rep(i, 0, sz(poly)) {
5         P cur = poly[i], prev = i ? poly[i-1] : poly.back();
6         auto a = s.cross(e, cur), b = s.cross(e, prev);
7         if ((a < 0) != (b < 0))
8             res.push_back(cur + (prev - cur) * (a / (a - b)));
9         if (a < 0)
10            res.push_back(cur);
11     }
12     return res;
13 }

```

7.24 Point inside Polygon

Overview

- Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary.
- NOTE: Requires 7.12 and 7.2.

⌚ Time complexity: $\mathcal{O}(n)$

Implementation

```

1 template<class P>
2 bool inPolygon(vector<P> &p, P a, bool strict = true) {
3     int cnt = 0, n = sz(p);
4     rep(i, 0, n) {
5         P q = p[(i + 1) % n];
6         if (onSegment(p[i], q, a)) return !strict;
7         //or: if (segDist(p[i], q, a) <= eps) return !strict;
8         cnt ^= ((a.y < p[i].y) - (a.y < q.y)) * a.cross(p[i], q) > 0;
9     }
10    return cnt;
11 }

```

7.25 Minkowski Sum

Overview

Consider two sets A and B of points on a plane. Minkowski sum $A + B$ is defined as $\{a + b | a \in A, b \in B\}$.

⌚ Time complexity: $\mathcal{O}(|P| + |Q|)$

Implementation

```

1 typedef Point<ll> P;
2 void reorder_polygon(vector<P> &p){
3     size_t pos = 0;
4     for(size_t i = 1; i < p.size(); i++){
5         if(p[i].y < p[pos].y || (p[i].y == p[pos].y && p[i].x < p[pos].x))
6             pos = i;
7     }
8     rotate(p.begin(), p.begin() + pos, p.end());
9 }
10
11 vector<P> minkowski(vector<P> p, vector<P> Q){
12     // the first vertex must be the lowest
13     reorder_polygon(p);
14     reorder_polygon(Q);
15     // we must ensure cyclic indexing
16     p.push_back(p[0]);
17     p.push_back(p[1]);
18     Q.push_back(Q[0]);
19     Q.push_back(Q[1]);
20     // main part
21     vector<P> result;
22     size_t i = 0, j = 0;
23     while(i < p.size() - 2 || j < Q.size() - 2){
24         result.push_back(p[i] + Q[j]);
25         auto cross = (p[i + 1] - p[i]).cross(Q[j + 1] - Q[j]);
26         if(cross >= 0 && i < p.size() - 2)
27             ++i;
28         if(cross <= 0 && j < Q.size() - 2)
29             ++j;
30     }
31     return result;
32 }

```

7.26 Manhattan MST

Overview

Given N points, returns up to $4 * N$ edges, which are guaranteed to contain a minimum spanning tree for the graph with edge weights $w(p, q) = |p.x - q.x| + |p.y - q.y|$. Edges are in the form (distance, src, dst). Use a standard MST algorithm on the result to find the final MST.

⌚ Time complexity: $\mathcal{O}(n)$

Implementation

```

1 typedef Point<int> P;
2 vector<array<int, 3>> manhattanMST(vector<P> ps) {
3     vi id(sz(ps));
4     iota(all(id), 0);
5     vector<array<int, 3>> edges;
6     rep(k, 0, 4) {
7         sort(all(id), [&](int i, int j) {
8             return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
9         map<int, int> sweep;
10        for (int i : id) {
11            for (auto it = sweep.lower_bound(-ps[i].y);
12                 it != sweep.end(); sweep.erase(it++)) {
13                int j = it->second;
14                P d = ps[i] - ps[j];
15                if (d.y > d.x) break;
16                edges.push_back({d.y + d.x, i, j});
17            }
18            sweep[-ps[i].y] = i;
19        }
20        for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x, p.y);
21    }
22    return edges;
23 }

```

8 Notes



8.1 Finding min cut

To build a min cut, once you have finished finding the max flow, bfs from source one more time. Edges that connect reached vertex and unreached vertex is considered a cut.

8.2 Finding minimum vertex cover on bipartite graph (König's theorem)

- Size of maximum matching = Size of minimum vertex cover.
- To build, use flow to find the maximum matching, and bfs from source one more time. The minimum vertex cover is the set of all vertices in the left partition that were not visited, combined with all vertices in the right partition that were visited.
- The weighted version is the same, except the capacity of the edge from source/sink to a vertex is that vertex weight.

8.3 Bitwise

- When work with both or and and: if there exist a subset of which sum ($|$, $\&$) is X , exists a subset with size $\leq \log_2(A_i)$. So when work with these type of problems, we just need to maintain a small number of candidates.
- There are no data structures for OR and AND problems (excepts for or, and convolution), when encountered, just look for essential observations.

8.4 CRT

When calculate with modulo m is not prime. Factorize m to prime numbers and calculate solution on these prime numbers. Then use CRT to find the final answer

8.5 Cayley Theorem

Number of labeled vertex n nodes tree is n^{n-2}

8.6 Divisor

- $X = \sum_D \phi(D)$ for all Divisor D of X . In some cases we can use this property to calculate $\gcd(a, b)$.
- There is at most 1 divisor of x that exceeds \sqrt{x}

8.7 Heap-like permutation

- A permutation is called heaplike if value of parent nodes is smaller/larger than its children.
- If there are no constraints, probability of a node u to belongs to a heaplike tree is independent with all other nodes. $P(u) = 1/(\text{size of subtree } u)$.
- The number of heaplike permutation is $N!/P(u)$

8.8 Little Fermat

For all prime numbers p , $a^p \equiv a \pmod p$

8.9 Lucas Theorem

$n! \pmod m = n_1! n_2! n_3! \dots \pmod m$ with n_i are m -base demonstration of n . k is the same

8.10 Mex

- Mex can not be calculate by binary search or whatsoever.
- In a permutation, $\text{mex}(1, r) = \min(\text{mex}(1, r), \text{mex}(1, n))$

8.11 Modulo

- $A[i] \equiv -A[i] \pmod M$ so we can abuse this property which such problems related to $(a[i] + b[i])$
- Wilson theorem : n is a prime number iff $(n-1)! \equiv n-1 \pmod n$.
- $x \equiv a \pmod n$ and $x \equiv a \pmod m$ means $x \equiv a \pmod{\text{lcm}(m, n)}$
- Find $a^x \equiv b \pmod m$ with baby step - giant step algo (discrete log)
- $A[x \pmod a] = B[x \pmod b] = i$ pattern of size $\gcd(a, b)$.
- Consider primitive root with problems with product modulo

8.12 Number Theory

- And, or, gcd as $f : F(i) = j$ that $f[j..i] = X$, $F(i) = F(i+1)$ so we can do some magical things like binary search on this function.
- pythagore $a = m^2 - n^2$, $b = 2mn$, $c = m^2 + n^2$.

8.13 Number Formulas

- Bernoulli numbers
 $B_0 = 1, B_1^\pm = \pm \frac{1}{2}, B_2 = \frac{1}{6}, B_3 = 0$
 $\sum_{j=0}^m \binom{m+1}{j} B_j = 0$, EGF is $B(x) = \frac{x}{e^x - 1} = \sum_{n=0}^{\infty} B_n \frac{x^n}{n!}$.
 $S_m(n) = \sum_{k=1}^n k^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k^+ n^{m+1-k}$
- Stirling numbers of the second kind Partitions of n distinct elements into exactly k groups.
 $S(n, k) = S(n-1, k-1) + kS(n-1, k)$, $S(n, 1) = S(n, n) = 1$
 $S(n, k) = \frac{1}{k!} \sum_{i=0}^k (-1)^{k-i} \binom{k}{i} i^n$
 $x^n = \sum_{i=0}^n S(n, i) (x)_i$
- Pentagonal number theorem
 $\prod_{n=1}^{\infty} (1 - x^n) = 1 + \sum_{k=1}^{\infty} (-1)^k (x^{k(3k+1)/2} + x^{k(3k-1)/2})$
- Catalan numbers
 $C_n^{(k)} = \frac{1}{(k-1)n+1} \binom{kn}{n}$
 $C^{(k)}(x) = 1 + x[C^{(k)}(x)]^k$
 $\frac{(2n)!}{(n+1)!n!}$
- Eulerian numbers
Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k :s s.t. $\pi(j) > \pi(j+1)$, $k+1$:s s.t. $\pi(j) \geq j$, k :s s.t. $\pi(j) > j$.
 $E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k)$
 $E(n, 0) = E(n, n-1) = 1$
 $E(n, k) = \sum_{j=0}^k (-1)^j \binom{n+1}{j} (k+1-j)^n$
- Burnside's Lemma
Let X be a set and G be a group that acts on X . For $g \in G$, denote by X^g the elements fixed by g :

$$X^g = \{x \in X \mid gx = x\}$$

Then

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|.$$

- Pick's theorem

For simple polygon, when points are all integer, we have $A = \frac{\#\{\text{lattice points in the interior}\} + \#\{\text{lattice points on the boundary}\}}{2} - 1$

- Apollonius Circle

The set $z : |z-a| = k|z-b|$ is the circle of radius $\frac{k|b-a|}{|k^2-1|}$ centered at $\frac{k^2b-a}{k^2-1}$.