

Contents

1	Reminder	1
1.1	Bug List	1
1.2	OwO	1
2	Basic	1
2.1	Vimrc	1
2.2	Stress	1
2.3	Run Sample	1
2.4	Runcpp.sh	1
2.5	Others	1
3	Data Structure	2
3.1	BIT	2
3.2	Lazy Propagation Segment Tree	2
3.3	Treap	2
3.4	Persistent Treap	3
3.5	Li Chao Tree	3
3.6	LineContainer	3
3.7	Sparse Table	4
3.8	Time Segment Tree	4
3.9	Dynamic Median	4
3.10	SOS DP	5
4	Flow / Matching	5
4.1	Dinic	5
4.2	MCMF	5
4.3	KM	5
4.4	Hopcroft-Karp	6
4.5	Blossom	6
4.6	Cover / Independent Set	7
4.7	Hungarian Algorithm	7
5	Graph	7
5.1	Heavy-Light Decomposition	7
5.2	Centroid Decomposition	8
5.3	Bellman-Ford + SPFA	8
5.4	BCC - AP	9
5.5	BCC - Bridge	9
5.6	SCC - Tarjan	10
5.7	SCC - Kosaraju	10
5.8	Eulerian Path - Undir	11
5.9	Eulerian Path - Dir	11
5.10	Hamilton Path	11
5.11	Kth Shortest Path	11
5.12	System of Difference Constraints	12

6	String	13
6.1	Aho Corasick	13
6.2	KMP	13
6.3	Z Value	13
6.4	Manacher	13
6.5	Suffix Array	14
6.6	Suffix Automaton	14
6.7	Minimum Rotation	14
6.8	Lyndon Factorization	15
6.9	Rolling Hash	15
6.10	Trie	15
7	Geometry	15
7.1	Basic Operations	15
7.2	Sort by Angle	16
7.3	Intersection	16
7.4	Polygon Area	16
7.5	Convex Hull	16
7.6	Point In Convex	16
7.7	Point Segment Distance	16
7.8	Point in Polygon	16
7.9	Minimum Euclidean Distance	17
7.10	Minkowski Sum	17
7.11	Lower Concave Hull	17
7.12	Pick's Theorem	18
7.13	Rotating SweepLine	17
7.14	Half Plane Intersection	17
7.15	Minimum Enclosing Circle	18
7.16	Union of Circles	18
7.17	Area Of Circle Polygon	18
7.18	3D Point	18
8	Number Theory	19
8.1	FFT	19
8.2	Pollard's rho	19
8.3	Miller Rabin	20
8.4	Fast Power	20
8.5	Extend GCD	20
8.6	Mu + Phi	20
8.7	Discrete Log	20
8.8	sqrt mod	21
8.9	Primitive Root	21
8.10	LinearSieve	21
8.11	Other Formulas	21
8.12	Polynomial	22
9	Linear Algebra	23
9.1	Gaussian-Jordan Elimination	23
9.2	Determinant	23
10	Combinatorics	23
10.1	Catalan Number	23
10.2	Burnside's Lemma	23

2 Basic

2.1 Vimrc

```

set number relativenumber ai t_Co=256 tabstop=4
set mouse=a shiftwidth=4 encoding=utf8
set bs=2 ruler laststatus=2 cmdheight=2
set clipboard=unnamedplus showcmd autoread
set belloff=all
filetype indent on

inoremap ( (<Esc>i
inoremap " "<Esc>i
inoremap [ <Esc>i
inoremap ' '<Esc>i
inoremap { {<CR><Esc>ko

nnoremap <tab> gt
nnoremap <S-tab> gT
inoremap <C-n> <Esc>;tabnew<CR>
nnoremap <C-n> :tabnew<CR>

inoremap <F9> <Esc>;w<CR>:!~/runcpp.sh %:p:t %:p:h<CR>
nnoremap <F9> :w<CR>:!~/runcpp.sh %:p:t %:p:h<CR>

syntax on
colorscheme desert
set filetype=cpp
set background=dark
hi Normal ctermfg=white ctermbg=black

```

2.2 Stress

```

g++ gen.cpp -o gen.out
g++ ac.cpp -o ac.out
g++ wa.cpp -o wa.out
for ((i=0;;i++))
do
    echo "$i"
    ./gen.out > in.txt
    ./ac.out < in.txt > ac.txt
    ./wa.out < in.txt > wa.txt
    diff ac.txt wa.txt || break
done

```

2.3 Run Sample

```

1 prog=$1
2 shift
3 g++ -O2 -std=c++20 -fsanitize=address -Wall -Wextra -
    Wshadow ${prog}.cpp -o ${prog}.out
4 for f in "$@"; do
5     out=${prog}_${(basename "$f")}.out
6     echo "input: $f"
7     cat "$f"
8     echo "output: $out"
9     ./${prog}.out < "$f" | tee "$out"
10    echo
11 done

```

2.4 Runcpp.sh

```

1 #!/bin/bash
2 clear
3 echo "Start compiling $1..."
4 echo
5 g++ -O2 -std=c++20 -fsanitize=address -Wall -Wextra -
    Wshadow $2/$1 -o $2/out
6 if [ "$?" -ne 0 ]
7 then
8     exit 1
9 fi
10 echo
11 echo "Done compiling"
12 echo "===== "
13 echo
14 echo "Input file:"
15 echo
16 cat $2/in.txt
17 echo
18 echo "===== "

```

1 Reminder

1.1 Bug List

- 沒開 long long
- 本地編譯請開-Wall -Wextra -Wshadow -fsanitize=address
- 陣列戳出界 / 開不夠大 / 開太大本地 compile 噴怪 error
- 傳之前先確定選對檔案
- 變數打錯
- 0-base / 1-base
- 忘記初始化
- == 打成 =
- dp[i] 從 dp[i-1] 轉移時忘記特判 i > 0
- std::sort 比較運算子寫成 < 或是讓 = 的情況為 true
- 漏 case / 分 case 要好好想
- 線段樹改值懶標初始值不能設為 0
- 少碰動態開點，能離散化就離散化
- 能不用浮點數運算就不用
- DFS 的時候不小心覆寫到全域變數
- 記得刪 cerr

1.2 OwO

- 可以構造複雜點的測資幫助思考
- 真的卡太久請跳題
- Enjoy The Contest!

```

19 echo
20 declare startTime=`date +%s%N`
21 $2/out < $2/in.txt > $2/out.txt
22 declare endTime=`date +%s%N`
23 delta=`expr $endTime - $startTime`
24 delta=`expr $delta / 1000000`
25 cat $2/out.txt
26 echo
27 echo "time: $delta ms"

```

2.5 Others

```

1 #pragma GCC optimize("Ofast,unroll-loops,no-stack-
  protector,fast-math")
2 #pragma GCC target("see,see2,see3,see4,avx2,bmi,bmi2,
  lzcnt,popcnt,tune=native")
3 #pragma GCC optimize("trapv")
4 mt19937 gen(chrono::steady_clock::now().
  time_since_epoch().count());
5 uniform_int_distribution<int> dis(1, 100);
6 cout << dis(gen) << endl;
7 shuffle(v.begin(), v.end(), gen);
8
9 struct edge {
10     int a, b, w;
11     friend istream& operator>>(istream& in, edge& x) {
12         in >> x.a >> x.b >> x.w; }
13     friend ostream& operator<<(ostream& out, const edge
14     & x) {
15         out << "(" << x.a << "," << x.b << "," << x.w
16         << ")";
17         return out;
18     }
19 };
20 struct cmp {
21     bool operator()(const edge& x, const edge& y) const {
22         return x.w < y.w; }
23 };
24 set<edge, cmp> st; // 遞增
25 map<edge, long long, cmp> mp; // 遞增
26 priority_queue<edge, vector<edge>, cmp> pq; // 遞減
27
28 #include <bits/extc++.h>
29 #include <ext/pb_ds/assoc_container.hpp>
30 #include <ext/pb_ds/tree_policy.hpp>
31 using namespace __gnu_pbds;
32
33 // map
34 tree<int, int, less<>, rb_tree_tag,
35     tree_order_statistics_node_update> tr;
36 tr.order_of_key(element);
37 tr.find_by_order(rank);
38
39 // set
40 tree<int, null_type, less<>, rb_tree_tag,
41     tree_order_statistics_node_update> tr;
42 tr.order_of_key(element);
43 tr.find_by_order(rank);
44
45 gp_hash_table<int, int> ht;
46 ht.find(element);
47 ht.insert({key, value});
48 ht.erase(element);
49
50 // priority queue Big First
51 __gnu_pbds::priority_queue<int, less<int>> big_q;
52 __gnu_pbds::priority_queue<int, greater<int>> small_q;
53 // Small First
54 q1.join(q2); // join

```

3 Data Structure

3.1 BIT

```

1 struct BIT {
2     int n;
3     long long bit[N];
4
5     void init(int x, vector<long long> &a) {
6         n = x;
7         for (int i = 1, j; i <= n; i++) {

```

```

8         bit[i] += a[i - 1], j = i + (i & -i);
9         if (j <= n) bit[j] += bit[i];
10     }
11 }
12
13 void update(int x, long long dif) {
14     while (x <= n) bit[x] += dif, x += x & -x;
15 }
16
17 long long query(int l, int r) {
18     if (l != 1) return query(1, r) - query(1, l -
19     1);
20
21     long long ret = 0;
22     while (l <= r) ret += bit[r], r -= r & -r;
23     return ret;
24 }
25 } bm;

```

3.2 Lazy Propagation Segment Tree

```

1 struct lazy_propagation{
2     // 0-based, [l, r], tg[0]->add, tg[1]->set
3     ll seg[N * 4], tg[2][N*4];
4     void assign (bool op, ll val, int idx){
5         if (op == 0){
6             if (tg[1][idx]) tg[1][idx] += val;
7             else tg[0][idx] += val;
8         }
9         else seg[idx] = 0, tg[0][idx] = 0, tg[1][idx]
10            = val;
11 }
12 ll sum (int idx, int len){
13     if (tg[1][idx]) return tg[1][idx] * len;
14     return tg[0][idx] * len + seg[idx];
15 }
16 void pull (int idx, int len){
17     seg[idx] = sum(2*idx, (len+1)/2) + sum(2*idx+1,
18     len/2);
19 }
20 void push (int idx){
21     if (!tg[0][idx] && !tg[1][idx]) return ;
22     if (tg[0][idx]){
23         assign(0, tg[0][idx], 2*idx);
24         assign(0, tg[0][idx], 2*idx+1);
25         tg[0][idx] = 0;
26     }
27     else{
28         assign(1, tg[1][idx], 2*idx);
29         assign(1, tg[1][idx], 2*idx+1);
30         tg[1][idx] = 0;
31     }
32 }
33 void update (bool op, ll val, int gl, int gr, int l
34 , int r, int idx){
35     if (r < l || gr < l || r < gl) return ;
36     if (gl <= l && r <= gr){
37         assign(op, val, idx);
38         return ;
39     }
40
41     int mid = (l + r) / 2;
42     push(idx);
43     update(op, val, gl, gr, l, mid, 2*idx);
44     update(op, val, gl, gr, mid+1, r, 2*idx+1);
45     pull(idx, r-l+1);
46 }
47
48 ll query (int gl, int gr, int l, int r, int idx){
49     if (r < l || gr < l || r < gl) return 0;
50     if (gl <= l && r <= gr) return sum(idx, r-l+1);
51
52     push(idx), pull(idx, r-l+1);
53     int mid = (l + r) / 2;
54     return query(gl, gr, l, mid, 2*idx) + query(gl,
55     gr, mid+1, r, 2*idx+1);
56 }
57 }bm;

```

3.3 Treap

```

1 mt19937 rng(random_device{}());
2 struct Treap {

```

```

3   Treap *l, *r;
4   int val, sum, real, tag, num, pri, rev;
5   Treap(int k) {
6       l = r = NULL;
7       val = sum = k;
8       num = 1;
9       real = -1;
10      tag = 0;
11      rev = 0;
12      pri = rng();
13  }
14  };
15  int siz(Treap *now) { return now ? now->num : 0; }
16  int sum(Treap *now) {
17      if (!now) return 0;
18      if (now->real != -1) return (now->real + now->tag)
19          * now->num;
20      return now->sum + now->tag * now->num;
21  }
22  void pull(Treap *&now) {
23      now->num = siz(now->l) + siz(now->r) + 1;
24      now->sum = sum(now->l) + sum(now->r) + now->val +
25          now->tag;
26  }
27  void push(Treap *&now) {
28      if (now->rev) {
29          swap(now->l, now->r);
30          now->l->rev ^= 1;
31          now->r->rev ^= 1;
32          now->rev = 0;
33      }
34      if (now->real != -1) {
35          now->real += now->tag;
36          if (now->l) {
37              now->l->tag = 0;
38              now->l->real = now->real;
39              now->l->val = now->real;
40          }
41          if (now->r) {
42              now->r->tag = 0;
43              now->r->real = now->real;
44              now->r->val = now->real;
45          }
46          now->val = now->real;
47          now->sum = now->real * now->num;
48          now->real = -1;
49          now->tag = 0;
50      } else {
51          if (now->l) now->l->tag += now->tag;
52          if (now->r) now->r->tag += now->tag;
53          now->sum += sum(now);
54          now->val += now->tag;
55          now->tag = 0;
56      }
57  }
58  Treap *merge(Treap *a, Treap *b) {
59      if (!a || !b) return a ? a : b;
60      else if (a->pri > b->pri) {
61          push(a);
62          a->r = merge(a->r, b);
63          pull(a);
64          return a;
65      } else {
66          push(b);
67          b->l = merge(a, b->l);
68          pull(b);
69          return b;
70      }
71  }
72  void split_size(Treap *rt, Treap *&a, Treap *&b, int
73      val) {
74      if (!rt) {
75          a = b = NULL;
76          return;
77      }
78      push(rt);
79      if (siz(rt->l) + 1 > val) {
80          b = rt;
81          split_size(rt->l, a, b->l, val);
82          pull(b);
83      } else {
84          a = rt;

```

```

82      split_size(rt->r, a->r, b, val - siz(a->l) - 1)
83      ;
84      pull(a);
85  }
86  void split_val(Treap *rt, Treap *&a, Treap *&b, int val
87      ) {
88      if (!rt) {
89          a = b = NULL;
90          return;
91      }
92      push(rt);
93      if (rt->val <= val) {
94          a = rt;
95          split_val(rt->r, a->r, b, val);
96          pull(a);
97      } else {
98          b = rt;
99          split_val(rt->l, a, b->l, val);
100         pull(b);
101     }

```

3.4 Persistent Treap

```

1  struct node {
2      node *l, *r;
3      char c;
4      int v, sz;
5      node(char x = '$') : c(x), v(mt()), sz(1) {
6          l = r = nullptr;
7      }
8      node(node* p) { *this = *p; }
9      void pull() {
10         sz = 1;
11         for (auto i : {l, r})
12             if (i) sz += i->sz;
13     }
14 } arr[maxn], *ptr = arr;
15 inline int size(node* p) { return p ? p->sz : 0; }
16 node* merge(node* a, node* b) {
17     if (!a || !b) return a ? a : b;
18     if (a->v < b->v) {
19         node* ret = new (ptr++) node(a);
20         ret->r = merge(ret->r, b);
21         ret->pull();
22         return ret;
23     } else {
24         node* ret = new (ptr++) node(b);
25         ret->l = merge(a, ret->l);
26         ret->pull();
27         return ret;
28     }
29 }
30 P<node*> split(node* p, int k) {
31     if (!p) return {nullptr, nullptr};
32     if (k >= size(p->l) + 1) {
33         auto [a, b] = split(p->r, k - size(p->l) - 1);
34         node* ret = new (ptr++) node(p);
35         ret->r = a, ret->pull();
36         return {ret, b};
37     } else {
38         auto [a, b] = split(p->l, k);
39         node* ret = new (ptr++) node(p);
40         ret->l = b, ret->pull();
41         return {a, ret};
42     }

```

3.5 Li Chao Tree

```

1  constexpr int maxn = 5e4 + 5;
2  struct line {
3      ld a, b;
4      ld operator()(ld x) { return a * x + b; }
5  } arr[(maxn + 1) << 2];
6  bool operator<(line a, line b) { return a.a < b.a; }
7  #define m ((l + r) >> 1)
8  void insert(line x, int l = 1, int r = 0, int i = maxn)
9      {
10     if (r - l == 1) {
11         if (x(l) > arr[i](l))
12             arr[i] = x;
13         return;

```

```

13     }
14     line a = max(arr[i], x), b = min(arr[i], x);
15     if (a(m) > b(m))
16         arr[i] = a, insert(b, i << 1, 1, m);
17     else
18         arr[i] = b, insert(a, i << 1 | 1, m, r);
19 }
20 ld query(int x, int i = 1, int l = 0, int r = maxn) {
21     if (x < 1 || r <= x) return -numeric_limits<ld>::
22         max();
23     if (r - l == 1) return arr[i](x);
24     return max({arr[i](x), query(x, i << 1, 1, m),
25         query(x, i << 1 | 1, m, r)});
26 }
27 #undef m

```

3.6 LineContainer

```

1 struct Line {
2     mutable ll k, m, p;
3     bool operator<(const Line& o) const { return k < o.
4         k; }
5     bool operator<(ll x) const { return p < x; }
6 };
7
8 struct LineContainer : multiset<Line, less<>> {
9     // (for doubles, use inf = 1/.0, div(a,b) = a/b)
10    static const ll inf = LLONG_MAX;
11    ll div(ll a, ll b) { // floored division
12        return a / b - ((a ^ b) < 0 && a % b); }
13    bool isect(iterator x, iterator y) {
14        if (y == end()) return x->p = inf, 0;
15        if (x->k == y->k) x->p = x->m > y->m ? inf : -
16            inf;
17        else x->p = div(y->m - x->m, x->k - y->k);
18        return x->p >= y->p;
19    }
20    void add(ll k, ll m) {
21        auto z = insert({k, m, 0}), y = z++, x = y;
22        while (isect(y, z)) z = erase(z);
23        if (x != begin() && isect(--x, y)) isect(x, y =
24            erase(y));
25        while ((y = x) != begin() && (--x)->p >= y->p)
26            isect(x, erase(y));
27    }
28    ll query(ll x) {
29        assert(!empty());
30        auto l = *lower_bound(x);
31        return l.k * x + l.m;
32    }
33 };

```

3.7 Sparse Table

```

1 const int lgmx = 19;
2
3 int n, q;
4 int spt[lgmx][maxn];
5
6 void build() {
7     FOR(k, 1, lgmx, 1) {
8         for (int i = 0; i + (1 << k) - 1 < n; i++) {
9             spt[k][i] = min(spt[k - 1][i], spt[k - 1][i
10                 + (1 << (k - 1))]);
11         }
12     }
13 }
14
15 int query(int l, int r) {
16     int ln = len(l, r);
17     int lg = __lg(ln);
18     return min(spt[lg][l], spt[lg][r - (1 << lg) + 1]);
19 }

```

3.8 Time Segment Tree

```

1 constexpr int maxn = 1e5 + 5;
2 V<P<int>> arr[(maxn + 1) << 2];
3 V<int> dsu, sz;
4 V<tuple<int, int, int>> his;

```

```

5 int cnt, q;
6 int find(int x) {
7     return x == dsu[x] ? x : find(dsu[x]);
8 };
9 inline bool merge(int x, int y) {
10    int a = find(x), b = find(y);
11    if (a == b) return false;
12    if (sz[a] > sz[b]) swap(a, b);
13    his.emplace_back(a, b, sz[b]), dsu[a] = b, sz[b] +=
14        sz[a];
15    return true;
16 };
17 inline void undo() {
18     auto [a, b, s] = his.back();
19     his.pop_back();
20     dsu[a] = a, sz[b] = s;
21 }
22 #define m ((l + r) >> 1)
23 void insert(int ql, int qr, P<int> x, int i = 1, int l
24     = 0, int r = q) {
25     // debug(ql, qr, x); return;
26     if (qr <= 1 || r <= ql) return;
27     if (ql <= 1 && r <= qr) {
28         arr[i].push_back(x);
29         return;
30     }
31     if (qr <= m)
32         insert(ql, qr, x, i << 1, 1, m);
33     else if (m <= ql)
34         insert(ql, qr, x, i << 1 | 1, m, r);
35     else {
36         insert(ql, qr, x, i << 1, 1, m);
37         insert(ql, qr, x, i << 1 | 1, m, r);
38     }
39 }
40 void traversal(V<int>& ans, int i = 1, int l = 0, int r
41     = q) {
42     int opcnt = 0;
43     // debug(i, l, r);
44     for (auto [a, b] : arr[i])
45         if (merge(a, b))
46             opcnt++, cnt--;
47     if (r - l == 1)
48         ans[l] = cnt;
49     else {
50         traversal(ans, i << 1, 1, m);
51         traversal(ans, i << 1 | 1, m, r);
52     }
53     while (opcnt--)
54         undo(), cnt++;
55     arr[i].clear();
56 }
57 #undef m
58 inline void solve() {
59     int n, m;
60     cin >> n >> m >> q, q++;
61     dsu.resize(cnt = n), sz.assign(n, 1);
62     iota(dsu.begin(), dsu.end(), 0);
63     // a, b, time, operation
64     unordered_map<ll, V<int>> s;
65     for (int i = 0; i < m; i++) {
66         int a, b;
67         cin >> a >> b;
68         if (a > b) swap(a, b);
69         s[((ll)a << 32) | b].emplace_back(0);
70     }
71     for (int i = 1; i < q; i++) {
72         int op, a, b;
73         cin >> op >> a >> b;
74         if (a > b) swap(a, b);
75         switch (op) {
76             case 1:
77                 s[((ll)a << 32) | b].push_back(i);
78                 break;
79             case 2:
80                 auto tmp = s[((ll)a << 32) | b].back();
81                 s[((ll)a << 32) | b].pop_back();
82                 insert(tmp, i, P<int>{a, b});
83         }
84     }
85     for (auto [p, v] : s) {
86         int a = p >> 32, b = p & -1;
87     }

```

```

84     while (v.size()) {
85         insert(v.back(), q, P<int>{a, b});
86         v.pop_back();
87     }
88 }
89 V<int> ans(q);
90 traversal(ans);
91 for (auto i : ans)
92     cout << i << ' ';
93 cout << endl;
94 }

```

3.9 Dynamic Median

```

1 struct Dynamic_Median {
2     multiset<long long> lo, hi;
3     long long slo = 0, shi = 0;
4     void rebalance() {
5         // keep sz(lo) >= sz(hi) and sz(lo) - sz(hi) <= 1
6         while((int)lo.size() > (int)hi.size() + 1) {
7             auto it = prev(lo.end());
8             long long x = *it;
9             lo.erase(it); slo -= x;
10            hi.insert(x); shi += x;
11        }
12        while((int)lo.size() < (int)hi.size()) {
13            auto it = hi.begin();
14            long long x = *it;
15            hi.erase(it); shi -= x;
16            lo.insert(x); slo += x;
17        }
18    }
19    void add(long long x) {
20        if(lo.empty() || x <= *prev(lo.end())) {
21            lo.insert(x); slo += x;
22        }
23        else {
24            hi.insert(x); shi += x;
25        }
26        rebalance();
27    }
28    void remove_one(long long x) {
29        if(!lo.empty() && x <= *prev(lo.end())) {
30            auto it = lo.find(x);
31            if(it != lo.end()) {
32                lo.erase(it); slo -= x;
33            }
34            else {
35                auto it2 = hi.find(x);
36                hi.erase(it2); shi -= x;
37            }
38        }
39        else {
40            auto it = hi.find(x);
41            if(it != hi.end()) {
42                hi.erase(it); shi -= x;
43            }
44            else {
45                auto it2 = lo.find(x);
46                lo.erase(it2); slo -= x;
47            }
48        }
49        rebalance();
50    }
51 };

```

3.10 SOS DP

```

1 for (int mask = 0; mask < (1 << n); mask++) {
2     for (int submask = mask; submask != 0; submask = (
3         submask - 1) & mask) {
4         int subset = mask ^ submask;
5     }
6 }

```

4 Flow / Matching

4.1 Dinic

```

1 using namespace std;
2 const int N = 2000 + 5;

```

```

3 int n, m, s, t, level[N], iter[N];
4 struct edge {int to, cap, rev;};
5 vector<edge> path[N];
6 void add(int a, int b, int c) {
7     path[a].pb({b, c, sz(path[b])});
8     path[b].pb({a, 0, sz(path[a]) - 1});
9 }
10 void bfs() {
11     memset(level, -1, sizeof(level));
12     level[s] = 0;
13     queue<int> q;
14     q.push(s);
15     while (q.size()) {
16         int now = q.front(); q.pop();
17         for (edge e : path[now]) if (e.cap > 0 && level
18             [e.to] == -1) {
19             level[e.to] = level[now] + 1;
20             q.push(e.to);
21         }
22     }
23     int dfs(int now, int flow) {
24         if (now == t) return flow;
25         for (int &i = iter[now]; i < sz(path[now]); i++) {
26             edge &e = path[now][i];
27             if (e.cap > 0 && level[e.to] == level[now] + 1) {
28                 int res = dfs(e.to, min(flow, e.cap));
29                 if (res > 0) {
30                     e.cap -= res;
31                     path[e.to][e.rev].cap += res;
32                     return res;
33                 }
34             }
35         }
36         return 0;
37     }
38     int dinic() {
39         int res = 0;
40         while (true) {
41             bfs();
42             if (level[t] == -1) break;
43             memset(iter, 0, sizeof(iter));
44             int now = 0;
45             while ((now = dfs(s, INF)) > 0) res += now;
46         }
47         return res;
48     }
49 }

```

4.2 MCMF

```

1 struct MCMF {
2     int n, s, t, par[N + 5], p_i[N + 5], dis[N + 5],
3     vis[N + 5];
4     struct edge {
5         int to, cap, rev, cost;
6     };
7     vector<edge> path[N];
8     void init(int _n, int _s, int _t) {
9         n = _n, s = _s, t = _t;
10        FOR(i, 0, 2 * n + 5)
11            par[i] = p_i[i] = vis[i] = 0;
12    }
13    void add(int a, int b, int c, int d) {
14        path[a].pb({b, c, sz(path[b]), d});
15        path[b].pb({a, 0, sz(path[a]) - 1, -d});
16    }
17    void spfa() {
18        FOR(i, 0, n * 2 + 5)
19            dis[i] = INF,
20            vis[i] = 0;
21        dis[s] = 0;
22        queue<int> q;
23        q.push(s);
24        while (!q.empty()) {
25            int now = q.front();
26            q.pop();
27            vis[now] = 0;
28            for (int i = 0; i < sz(path[now]); i++) {
29                edge e = path[now][i];
30                if (e.cap > 0 && dis[e.to] > dis[now] +
31                    e.cost) {
32                    dis[e.to] = dis[now] + e.cost;
33                    p_i[e.to] = i;
34                    par[e.to] = now;
35                    vis[e.to] = 1;
36                    q.push(e.to);
37                }
38            }
39        }
40    }
41 }

```

```

30         dis[e.to] = dis[now] + e.cost;
31         par[e.to] = now;
32         p_i[e.to] = i;
33         if (vis[e.to] == 0) {
34             vis[e.to] = 1;
35             q.push(e.to);
36         }
37     }
38 }
39 }
40 }
41 pii flow() {
42     int flow = 0, cost = 0;
43     while (true) {
44         spfa();
45         if (dis[t] == INF)
46             break;
47         int mn = INF;
48         for (int i = t; i != s; i = par[i])
49             mn = min(mn, path[par[i]][p_i[i]].cap);
50         flow += mn;
51         cost += dis[t] * mn;
52         for (int i = t; i != s; i = par[i]) {
53             edge &now = path[par[i]][p_i[i]];
54             now.cap -= mn;
55             path[i][now.rev].cap += mn;
56         }
57     }
58     return mp(flow, cost);
59 }
60 };

```

4.3 KM

```

1 struct KM {
2     int n, mx[1005], my[1005], pa[1005];
3     int g[1005][1005], lx[1005], ly[1005], sy[1005];
4     bool vx[1005], vy[1005];
5     void init(int _n) {
6         n = _n;
7         FOR(i, 1, n + 1)
8             fill(g[i], g[i] + 1 + n, 0);
9     }
10    void add(int a, int b, int c) { g[a][b] = c; }
11    void augment(int y) {
12        for (int x, z; y; y = z)
13            x = pa[y], z = mx[x], my[y] = x, mx[x] = y;
14    }
15    void bfs(int st) {
16        FOR(i, 1, n + 1)
17            sy[i] = INF,
18            vx[i] = vy[i] = 0;
19        queue<int> q;
20        q.push(st);
21        for (;;) {
22            while (!q.empty()) {
23                int x = q.front();
24                q.pop();
25                vx[x] = 1;
26                FOR(y, 1, n + 1)
27                    if (!vy[y]) {
28                        int t = lx[x] + ly[y] - g[x][y];
29                        if (t == 0) {
30                            pa[y] = x;
31                            if (!my[y]) {
32                                augment(y);
33                                return;
34                            }
35                            vy[y] = 1, q.push(my[y]);
36                        } else if (sy[y] > t)
37                            pa[y] = x, sy[y] = t;
38                    }
39            }
40            int cut = INF;
41            FOR(y, 1, n + 1)
42                if (!vy[y] && cut > sy[y]) cut = sy[y];
43            FOR(j, 1, n + 1) {
44                if (vx[j]) lx[j] -= cut;
45                if (vy[j])
46                    ly[j] += cut;
47                else
48                    sy[j] -= cut;

```

4.4 Hopcroft-Karp

```

1 struct HopcroftKarp {
2     // id: X = [1, nx], Y = [nx+1, nx+ny]
3     int n, nx, ny, m, MXCNT;
4     vector<vector<int>> g;
5     vector<int> mx, my, dis, vis;
6     void init(int nnx, int nny, int mm) {
7         nx = nnx, ny = nny, m = mm;
8         n = nx + ny + 1;
9         g.clear();
10        g.resize(n);
11    }
12    void add(int x, int y) {
13        g[x].emplace_back(y);
14        g[y].emplace_back(x);
15    }
16    bool dfs(int x) {
17        vis[x] = true;
18        Each(y, g[x]) {
19            int px = my[y];
20            if (px == -1 ||
21                (dis[px] == dis[x] + 1 &&
22                 !vis[px] && dfs(px))) {
23                mx[x] = y;
24                my[y] = x;
25                return true;
26            }
27        }
28        return false;
29    }
30    void get() {
31        mx.clear();
32        mx.resize(n, -1);
33        my.clear();
34        my.resize(n, -1);
35    }
36    while (true) {
37        queue<int> q;
38        dis.clear();
39        dis.resize(n, -1);
40        for (int x = 1; x <= nx; x++) {
41            if (mx[x] == -1) {
42                dis[x] = 0;
43                q.push(x);
44            }
45        }
46        while (!q.empty()) {
47            int x = q.front();
48            q.pop();
49            Each(y, g[x]) {

```



```

50         if (my[y] != -1 && dis[my[y]] ==
51             -1) {
52             dis[my[y]] = dis[x] + 1;
53             q.push(my[y]);
54         }
55     }
56
57     bool brk = true;
58     vis.clear();
59     vis.resize(n, 0);
60     for (int x = 1; x <= nx; x++)
61         if (mx[x] == -1 && dfs(x))
62             brk = false;
63
64     if (brk) break;
65 }
66 MXCNT = 0;
67 for (int x = 1; x <= nx; x++)
68     if (mx[x] != -1) MXCNT++;
69 }
70 } hk;

```

4.5 Blossom

```

1  const int N=5e2+10;
2  struct Graph{
3      int to[N],bro[N],head[N],e;
4      int lnk[N],vis[N],stp,n;
5      void init(int _n){
6          stp=0;e=1;n=_n;
7          FOR(i,0,n+1)head[i]=lnk[i]=vis[i]=0;
8      }
9      void add(int u,int v){
10         to[e]=v,bro[e]=head[u],head[u]=e++;
11         to[e]=u,bro[e]=head[v],head[v]=e++;
12     }
13     bool dfs(int x){
14         vis[x]=stp;
15         for(int i=head[x];i;i=bro[i])
16             {
17                 int v=to[i];
18                 if(!lnk[v])
19                     {
20                         lnk[x]=v;lnk[v]=x;
21                         return true;
22                     }
23                 else if(vis[lnk[v]]<stp)
24                     {
25                         int w=lnk[v];
26                         lnk[x]=v,lnk[v]=x,lnk[w]=0;
27                         if(dfs(w))return true;
28                         lnk[w]=v,lnk[v]=w,lnk[x]=0;
29                     }
30             }
31         return false;
32     }
33     int solve(){
34         int ans=0;
35         FOR(i,1,n+1){
36             if(!lnk[i]){
37                 stp++;
38                 ans+=dfs(i);
39             }
40         }
41         return ans;
42     }
43     void print_matching(){
44         FOR(i,1,n+1)
45             if(i<graph.lnk[i])
46                 cout<<i<<" "<<graph.lnk[i]<<endl;
47     }
48 };

```

4.6 Cover / Independent Set

```

1  V(E) Cover: choose some V(E) to cover all E(V)
2  V(E) Independ: set of V(E) not adj to each other
3
4  M = Max Matching
5  Cv = Min V Cover
6  Ce = Min E Cover

```

```

7  Iv = Max V Ind
8  Ie = Max E Ind (equiv to M)
9
10 M = Cv (Konig Theorem)
11 Iv = V \ Cv
12 Ce = V - M
13
14 Construct Cv:
15 1. Run Dinic
16 2. Find s-t min cut
17 3. Cv = {X in T} + {Y in S}

```

4.7 Hungarian Algorithm

```

1  const int N = 2e3;
2  int match[N];
3  bool vis[N];
4  int n;
5  vector<int> ed[N];
6  int match_cnt;
7  bool dfs(int u) {
8      vis[u] = 1;
9      for(int i : ed[u]) {
10         if(match[i] == 0 || !vis[match[i]] && dfs(match
11             [i])) {
12             match[i] = u;
13             return true;
14         }
15     }
16     return false;
17 }
18 void hungary() {
19     memset(match, 0, sizeof(match));
20     match_cnt = 0;
21     for(int i = 1; i <= n; i++) {
22         memset(vis, 0, sizeof(vis));
23         if(dfs(i)) match_cnt++;
24     }
25 }

```

5 Graph

5.1 Heavy-Light Decomposition

```

1  const int N = 2e5 + 5;
2  int n, dfn[N], son[N], top[N], num[N], dep[N], p[N];
3  vector<int> path[N];
4  struct node {
5      int mx, sum;
6  } seg[N << 2];
7  void update(int x, int l, int r, int qx, int val) {
8      if (l == r) {
9          seg[x].mx = seg[x].sum = val;
10         return;
11     }
12     int mid = (l + r) >> 1;
13     if (qx <= mid) update(x << 1, l, mid, qx, val);
14     else update(x << 1 | 1, mid + 1, r, qx, val);
15     seg[x].mx = max(seg[x << 1].mx, seg[x << 1 | 1].mx);
16     seg[x].sum = seg[x << 1].sum + seg[x << 1 | 1].sum;
17 }
18 int big(int x, int l, int r, int ql, int qr) {
19     if (ql <= l && r <= qr) return seg[x].mx;
20     int mid = (l + r) >> 1;
21     int res = -INF;
22     if (ql <= mid) res = max(res, big(x << 1, l, mid,
23         ql, qr));
24     if (mid < qr) res = max(res, big(x << 1 | 1, mid +
25         1, r, ql, qr));
26     return res;
27 }
28 int ask(int x, int l, int r, int ql, int qr) {
29     if (ql <= l && r <= qr) return seg[x].sum;
30     int mid = (l + r) >> 1;
31     int res = 0;
32     if (ql <= mid) res += ask(x << 1, l, mid, ql, qr);
33     if (mid < qr) res += ask(x << 1 | 1, mid + 1, r, ql,
34         qr);
35     return res;
36 }

```

```

34 void dfs1(int now) {
35     son[now] = -1;
36     num[now] = 1;
37     for (auto i : path[now]) {
38         if (!dep[i]) {
39             dep[i] = dep[now] + 1;
40             p[i] = now;
41             dfs1(i);
42             num[now] += num[i];
43             if (son[now] == -1 || num[i] > num[son[now]]) son[now] = i;
44         }
45     }
46 }
47 int cnt;
48 void dfs2(int now, int t) {
49     top[now] = t;
50     cnt++;
51     dfn[now] = cnt;
52     if (son[now] == -1) return;
53     dfs2(son[now], t);
54     for (auto i : path[now])
55         if (i != p[now] && i != son[now]) dfs2(i, i);
56 }
57 int path_big(int x, int y) {
58     int res = -INF;
59     while (top[x] != top[y]) {
60         if (dep[top[x]] < dep[top[y]]) swap(x, y);
61         res = max(res, big(1, 1, n, dfn[top[x]], dfn[x]));
62         x = p[top[x]];
63     }
64     if (dfn[x] > dfn[y]) swap(x, y);
65     res = max(res, big(1, 1, n, dfn[x], dfn[y]));
66     return res;
67 }
68 int path_sum(int x, int y) {
69     int res = 0;
70     while (top[x] != top[y]) {
71         if (dep[top[x]] < dep[top[y]]) swap(x, y);
72         res += ask(1, 1, n, dfn[top[x]], dfn[x]);
73         x = p[top[x]];
74     }
75     if (dfn[x] > dfn[y]) swap(x, y);
76     res += ask(1, 1, n, dfn[x], dfn[y]);
77     return res;
78 }
79 void buildTree() {
80     FOR(i, 0, n - 1) {
81         int a, b;
82         cin >> a >> b;
83         path[a].pb(b);
84         path[b].pb(a);
85     }
86 }
87 void buildHLD(int root) {
88     dep[root] = 1;
89     dfs1(root);
90     dfs2(root, root);
91     FOR(i, 1, n + 1) {
92         int now;
93         cin >> now;
94         update(1, 1, n, dfn[i], now);
95     }
96 }

```

5.2 Centroid Decomposition

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int N = 1e5 + 5;
4 vector<int> a[N];
5 int sz[N], lv[N];
6 bool used[N];
7 int f_sz(int x, int p) {
8     sz[x] = 1;
9     for (int i : a[x])
10         if (i != p && !used[i])
11             sz[x] += f_sz(i, x);
12     return sz[x];
13 }
14 int f_cen(int x, int p, int total) {

```

```

15     for (int i : a[x]) {
16         if (i != p && !used[i] && 2 * sz[i] > total)
17             return f_cen(i, x, total);
18     }
19     return x;
20 }
21 void cd(int x, int p) {
22     int total = f_sz(x, p);
23     int cen = f_cen(x, p, total);
24     lv[cen] = lv[p] + 1;
25     used[cen] = 1;
26     // cout << "cd: " << x << " " << p << " " << cen <<
27     // "\n";
28     for (int i : a[cen]) {
29         if (!used[i])
30             cd(i, cen);
31     }
32 }
33 int main() {
34     ios_base::sync_with_stdio(0);
35     cin.tie(0);
36     int n;
37     cin >> n;
38     for (int i = 0, x, y; i < n - 1; i++) {
39         cin >> x >> y;
40         a[x].push_back(y);
41         a[y].push_back(x);
42     }
43     cd(1, 0);
44     for (int i = 1; i <= n; i++)
45         cout << (char)('A' + lv[i] - 1) << " ";
46     cout << "\n";

```

5.3 Bellman-Ford + SPFA

```

1 int n, m;
2
3 // Graph
4 vector<vector<pair<int, ll> > > g;
5 vector<ll> dis;
6 vector<bool> negCycle;
7
8 // SPFA
9 vector<int> rlx;
10 queue<int> q;
11 vector<bool> inq;
12 vector<int> pa;
13 void SPFA(vector<int>& src) {
14     dis.assign(n + 1, LINF);
15     negCycle.assign(n + 1, false);
16     rlx.assign(n + 1, 0);
17     while (!q.empty()) q.pop();
18     inq.assign(n + 1, false);
19     pa.assign(n + 1, -1);
20
21     for (auto& s : src) {
22         dis[s] = 0;
23         q.push(s);
24         inq[s] = true;
25     }
26
27     while (!q.empty()) {
28         int u = q.front();
29         q.pop();
30         inq[u] = false;
31         if (rlx[u] >= n) {
32             negCycle[u] = true;
33         } else
34             for (auto& e : g[u]) {
35                 int v = e.first;
36                 ll w = e.second;
37                 if (dis[v] > dis[u] + w) {
38                     dis[v] = dis[u] + w;
39                     rlx[v] = rlx[u] + 1;
40                     pa[v] = u;
41                     if (!inq[v]) {
42                         q.push(v);
43                         inq[v] = true;
44                     }
45                 }
46             }

```



```

47     }
48 }
49
50 // Bellman-Ford
51 queue<int> q;
52 vector<int> pa;
53 void BellmanFord(vector<int>& src) {
54     dis.assign(n + 1, LINF);
55     negCycle.assign(n + 1, false);
56     pa.assign(n + 1, -1);
57
58     for (auto& s : src) dis[s] = 0;
59
60     for (int rlx = 1; rlx <= n; rlx++) {
61         for (int u = 1; u <= n; u++) {
62             if (dis[u] == LINF) continue; // Important
63             for (auto& e : g[u]) {
64                 int v = e.first;
65                 ll w = e.second;
66                 if (dis[v] > dis[u] + w) {
67                     dis[v] = dis[u] + w;
68                     pa[v] = u;
69                     if (rlx == n) negCycle[v] = true;
70                 }
71             }
72         }
73     }
74 }
75
76 // Negative Cycle Detection
77 void NegCycleDetect() {
78     /* No Neg Cycle: NO
79     Exist Any Neg Cycle:
80     YES
81     v0 v1 v2 ... vk v0 */
82
83     vector<int> src;
84     for (int i = 1; i <= n; i++)
85         src.emplace_back(i);
86
87     SPFA(src);
88     // BellmanFord(src);
89
90     int ptr = -1;
91     for (int i = 1; i <= n; i++)
92         if (negCycle[i]) {
93             ptr = i;
94             break;
95         }
96
97     if (ptr == -1) {
98         return cout << "NO" << endl, void();
99     }
100
101     cout << "YES\n";
102     vector<int> ans;
103     vector<bool> vis(n + 1, false);
104
105     while (true) {
106         ans.emplace_back(ptr);
107         if (vis[ptr]) break;
108         vis[ptr] = true;
109         ptr = pa[ptr];
110     }
111     reverse(ans.begin(), ans.end());
112
113     vis.assign(n + 1, false);
114     for (auto& x : ans) {
115         cout << x << ' ';
116         if (vis[x]) break;
117         vis[x] = true;
118     }
119     cout << endl;
120 }
121
122 // Distance Calculation
123 void calcDis(int s) {
124     vector<int> src;
125     src.emplace_back(s);
126     SPFA(src);
127     // BellmanFord(src);

```

```

128
129 while (!q.empty()) q.pop();
130 for (int i = 1; i <= n; i++)
131     if (negCycle[i]) q.push(i);
132
133 while (!q.empty()) {
134     int u = q.front();
135     q.pop();
136     for (auto& e : g[u]) {
137         int v = e.first;
138         if (!negCycle[v]) {
139             q.push(v);
140             negCycle[v] = true;
141         }
142     }
143 }
144 }

```

5.4 BCC - AP

```

1 int n, m;
2 int low[maxn], dfn[maxn], instp;
3 vector<int> E, g[maxn];
4 bitset<maxn> isap;
5 bitset<maxm> vis;
6 stack<int> stk;
7 int bccnt;
8 vector<int> bcc[maxn];
9 inline void popout(int u) {
10     bccnt++;
11     bcc[bccnt].emplace_back(u);
12     while (!stk.empty()) {
13         int v = stk.top();
14         if (u == v) break;
15         stk.pop();
16         bcc[bccnt].emplace_back(v);
17     }
18 }
19 void dfs(int u, bool rt = 0) {
20     stk.push(u);
21     low[u] = dfn[u] = ++instp;
22     int kid = 0;
23     Each(e, g[u]) {
24         if (vis[e]) continue;
25         vis[e] = true;
26         int v = E[e] ^ u;
27         if (!dfn[v]) {
28             // tree edge
29             kid++;
30             dfs(v);
31             low[u] = min(low[u], low[v]);
32             if (!rt && low[v] >= dfn[u]) {
33                 // bcc found: u is ap
34                 isap[u] = true;
35                 popout(u);
36             }
37         } else {
38             // back edge
39             low[u] = min(low[u], dfn[v]);
40         }
41     }
42     // special case: root
43     if (rt) {
44         if (kid > 1) isap[u] = true;
45         popout(u);
46     }
47 }
48 void init() {
49     cin >> n >> m;
50     fill(low, low + maxn, INF);
51     REP(i, m) {
52         int u, v;
53         cin >> u >> v;
54         g[u].emplace_back(i);
55         g[v].emplace_back(i);
56         E.emplace_back(u ^ v);
57     }
58 }
59 void solve() {
60     FOR(i, 1, n + 1, 1) {
61         if (!dfn[i]) dfs(i, true);
62     }

```

```

63     vector<int> ans;
64     int cnt = 0;
65     FOR(i, 1, n + 1, 1) {
66         if (isap[i]) cnt++, ans.emplace_back(i);
67     }
68     cout << cnt << endl;
69     Each(i, ans) cout << i << ' ';
70     cout << endl;
71 }

```

5.5 BCC - Bridge

```

1  int n, m;
2  vector<int> g[maxn], E;
3  int low[maxn], dfn[maxn], instp;
4  int bccnt, bccid[maxn];
5  stack<int> stk;
6  bitset<maxm> vis, isbrg;
7  void init() {
8      cin >> n >> m;
9      REP(i, m) {
10         int u, v;
11         cin >> u >> v;
12         E.emplace_back(u ^ v);
13         g[u].emplace_back(i);
14         g[v].emplace_back(i);
15     }
16     fill(low, low + maxn, INF);
17 }
18 void popout(int u) {
19     bccnt++;
20     while (!stk.empty()) {
21         int v = stk.top();
22         if (v == u) break;
23         stk.pop();
24         bccid[v] = bccnt;
25     }
26 }
27 void dfs(int u) {
28     stk.push(u);
29     low[u] = dfn[u] = ++instp;
30
31     Each(e, g[u]) {
32         if (vis[e]) continue;
33         vis[e] = true;
34
35         int v = E[e] ^ u;
36         if (dfn[v]) {
37             // back edge
38             low[u] = min(low[u], dfn[v]);
39         } else {
40             // tree edge
41             dfs(v);
42             low[u] = min(low[u], low[v]);
43             if (low[v] == dfn[v]) {
44                 isbrg[e] = true;
45                 popout(u);
46             }
47         }
48     }
49 }
50 void solve() {
51     FOR(i, 1, n + 1, 1) {
52         if (!dfn[i]) dfs(i);
53     }
54     vector<pii> ans;
55     vis.reset();
56     FOR(u, 1, n + 1, 1) {
57         Each(e, g[u]) {
58             if (!isbrg[e] || vis[e]) continue;
59             vis[e] = true;
60             int v = E[e] ^ u;
61             ans.emplace_back(mp(u, v));
62         }
63     }
64     cout << (int)ans.size() << endl;
65     Each(e, ans) cout << e.F << ' ' << e.S << endl;
66 }

```

5.6 SCC - Tarjan

```

1  // 2-SAT
2  vector<int> E, g[maxn]; // 1~n, n+1~2n
3  int low[maxn], in[maxn], instp;
4  int sccnt, sccid[maxn];
5  stack<int> stk;
6  bitset<maxn> ins, vis;
7  int n, m;
8  void init() {
9      cin >> m >> n;
10     E.clear();
11     fill(g, g + maxn, vector<int>());
12     fill(low, low + maxn, INF);
13     memset(in, 0, sizeof(in));
14     instp = 1;
15     sccnt = 0;
16     memset(sccid, 0, sizeof(sccid));
17     ins.reset();
18     vis.reset();
19 }
20 inline int no(int u) {
21     return (u > n ? u - n : u + n);
22 }
23 int ecnt = 0;
24 inline void clause(int u, int v) {
25     E.eb(no(u) ^ v);
26     g[no(u)].eb(ecnt++);
27     E.eb(no(v) ^ u);
28     g[no(v)].eb(ecnt++);
29 }
30 void dfs(int u) {
31     in[u] = instp++;
32     low[u] = in[u];
33     stk.push(u);
34     ins[u] = true;
35
36     Each(e, g[u]) {
37         if (vis[e]) continue;
38         vis[e] = true;
39
40         int v = E[e] ^ u;
41         if (ins[v])
42             low[u] = min(low[u], in[v]);
43         else if (!in[v]) {
44             dfs(v);
45             low[u] = min(low[u], low[v]);
46         }
47     }
48     if (low[u] == in[u]) {
49         sccnt++;
50         while (!stk.empty()) {
51             int v = stk.top();
52             stk.pop();
53             ins[v] = false;
54             sccid[v] = sccnt;
55             if (u == v) break;
56         }
57     }
58 }
59 int main() {
60     init();
61     REP(i, m) {
62         char su, sv;
63         int u, v;
64         cin >> su >> u >> sv >> v;
65         if (su == '-' ) u = no(u);
66         if (sv == '-' ) v = no(v);
67         clause(u, v);
68     }
69     FOR(i, 1, 2 * n + 1, 1) {
70         if (!in[i]) dfs(i);
71     }
72     FOR(u, 1, n + 1, 1) {
73         int du = no(u);
74         if (sccid[u] == sccid[du]) {
75             return cout << "IMPOSSIBLE\n", 0;
76         }
77     }
78     FOR(u, 1, n + 1, 1) {
79         int du = no(u);
80         cout << (sccid[u] < sccid[du] ? '+' : '-') << '
';
81     }

```

```
82 |     cout << endl;
83 | }
```

5.7 SCC - Kosaraju

```
1 | const int N = 1e5 + 10;
2 | vector<int> ed[N], ed_b[N]; // 反邊
3 | vector<int> SCC(N); // 最後SCC的分組
4 | bitset<N> vis;
5 | int SCC_cnt;
6 | int n, m;
7 | vector<int> pre; // 後序遍歷
8 |
9 | void dfs(int x) {
10 |     vis[x] = 1;
11 |     for (int i : ed[x]) {
12 |         if (vis[i]) continue;
13 |         dfs(i);
14 |     }
15 |     pre.push_back(x);
16 | }
17 |
18 | void dfs2(int x) {
19 |     vis[x] = 1;
20 |     SCC[x] = SCC_cnt;
21 |     for (int i : ed_b[x]) {
22 |         if (vis[i]) continue;
23 |         dfs2(i);
24 |     }
25 | }
26 |
27 | void kosaraju() {
28 |     for (int i = 1; i <= n; i++) {
29 |         if (!vis[i]) {
30 |             dfs(i);
31 |         }
32 |     }
33 |     SCC_cnt = 0;
34 |     vis = 0;
35 |     for (int i = n - 1; i >= 0; i--) {
36 |         if (!vis[pre[i]]) {
37 |             SCC_cnt++;
38 |             dfs2(pre[i]);
39 |         }
40 |     }
41 | }
```

5.8 Eulerian Path - Undir

```
1 | // from 1 to n
2 | #define gg return cout << "IMPOSSIBLE\n", void();
3 |
4 | int n, m;
5 | vector<int> g[maxn];
6 | bitset<maxn> inodd;
7 |
8 | void init() {
9 |     cin >> n >> m;
10 |    inodd.reset();
11 |    for (int i = 0; i < m; i++) {
12 |        int u, v;
13 |        cin >> u >> v;
14 |        inodd[u] = inodd[u] ^ true;
15 |        inodd[v] = inodd[v] ^ true;
16 |        g[u].emplace_back(v);
17 |        g[v].emplace_back(u);
18 |    }
19 | }
20 | stack<int> stk;
21 | void dfs(int u) {
22 |     while (!g[u].empty()) {
23 |         int v = g[u].back();
24 |         g[u].pop_back();
25 |         dfs(v);
26 |     }
27 |     stk.push(u);
28 | }
```

5.9 Eulerian Path - Dir

```
1 | // from node 1 to node n
```

```
2 | #define gg return cout << "IMPOSSIBLE\n", 0
3 |
4 | int n, m;
5 | vector<int> g[maxn];
6 | stack<int> stk;
7 | int in[maxn], out[maxn];
8 |
9 | void init() {
10 |     cin >> n >> m;
11 |     for (int i = 0; i < m; i++) {
12 |         int u, v;
13 |         cin >> u >> v;
14 |         g[u].emplace_back(v);
15 |         out[u]++, in[v]++;
16 |     }
17 |     for (int i = 1; i <= n; i++) {
18 |         if (i == 1 && out[i] - in[i] != 1) gg;
19 |         if (i == n && in[i] - out[i] != 1) gg;
20 |         if (i != 1 && i != n && in[i] != out[i]) gg;
21 |     }
22 | }
23 | void dfs(int u) {
24 |     while (!g[u].empty()) {
25 |         int v = g[u].back();
26 |         g[u].pop_back();
27 |         dfs(v);
28 |     }
29 |     stk.push(u);
30 | }
31 | void solve() {
32 |     dfs(1) for (int i = 1; i <= n; i++) if ((int)g[i].
33 |         size()) gg;
34 |     while (!stk.empty()) {
35 |         int u = stk.top();
36 |         stk.pop();
37 |         cout << u << ' ';
38 |     }
```

5.10 Hamilton Path

```
1 | // top down DP
2 | // Be Aware Of Multiple Edges
3 | int n, m;
4 | ll dp[maxn][1<<maxn];
5 | int adj[maxn][maxn];
6 |
7 | void init() {
8 |     cin >> n >> m;
9 |     fill(dp[0], dp[maxn-1]+(1<<maxn), -1);
10 | }
11 |
12 | void DP(int i, int msk) {
13 |     if (dp[i][msk] != -1) return;
14 |     dp[i][msk] = 0;
15 |     REP(j, n) if (j != i && (msk & (1<<j)) && adj[j][i]) {
16 |         int sub = msk ^ (1<<i);
17 |         if (dp[j][sub] == -1) DP(j, sub);
18 |         dp[i][msk] += dp[j][sub] * adj[j][i];
19 |         if (dp[i][msk] >= MOD) dp[i][msk] %= MOD;
20 |     }
21 | }
22 |
23 |
24 | int main() {
25 |     WiWiHorz
26 |     init();
27 |
28 |     REP(i, m) {
29 |         int u, v;
30 |         cin >> u >> v;
31 |         if (u == v) continue;
32 |         adj[--u][--v]++;
33 |     }
34 |
35 |     dp[0][1] = 1;
36 |     FOR(i, 1, n, 1) {
37 |         dp[i][1] = 0;
38 |         dp[i][1|(1<<i)] = adj[0][i];
39 |     }
40 |     FOR(msk, 1, (1<<n), 1) {
```

```

41     if (msk == 1) continue;
42     dp[0][msk] = 0;
43 }
44
45 DP(n-1, (1<<n)-1);
46 cout << dp[n-1][(1<<n)-1] << endl;
47
48 return 0;
49 }
50 }

```

5.11 Kth Shortest Path

```

1 // time: O(|E| \lg |E| + |V| \lg |V| + K)
2 // memory: O(|E| \lg |E| + |V|)
3 struct KSP { // 1-base
4     struct nd {
5         int u, v;
6         ll d;
7         nd(int ui = 0, int vi = 0, ll di = INF) {
8             u = ui;
9             v = vi;
10            d = di;
11        }
12    };
13    struct heap {
14        nd* edge;
15        int dep;
16        heap* chd[4];
17    };
18    static int cmp(heap* a, heap* b) { return a->edge->d > b->edge->d; }
19    struct node {
20        int v;
21        ll d;
22        heap* H;
23        nd* E;
24        node() {}
25        node(ll _d, int _v, nd* _E) {
26            d = _d;
27            v = _v;
28            E = _E;
29        }
30        node(heap* _H, ll _d) {
31            H = _H;
32            d = _d;
33        }
34        friend bool operator<(node a, node b) { return a.d > b.d; }
35    };
36    int n, k, s, t, dst[N];
37    nd* nxt[N];
38    vector<nd*> g[N], rg[N];
39    heap *nullNd, *head[N];
40    void init(int _n, int _k, int _s, int _t) {
41        n = _n;
42        k = _k;
43        s = _s;
44        t = _t;
45        for (int i = 1; i <= n; i++) {
46            g[i].clear();
47            rg[i].clear();
48            nxt[i] = NULL;
49            head[i] = NULL;
50            dst[i] = -1;
51        }
52    }
53    void addEdge(int ui, int vi, ll di) {
54        nd* e = new nd(ui, vi, di);
55        g[ui].push_back(e);
56        rg[vi].push_back(e);
57    }
58    queue<int> dfsQ;
59    void dijkstra() {
60        while (dfsQ.size()) dfsQ.pop();
61        priority_queue<node> Q;
62        Q.push(node(0, t, NULL));
63        while (!Q.empty()) {
64            node p = Q.top();
65            Q.pop();
66            if (dst[p.v] != -1) continue;
67            dst[p.v] = p.d;

```

```

68            nxt[p.v] = p.E;
69            dfsQ.push(p.v);
70            for (auto e : rg[p.v]) Q.push(node(p.d + e->d, e->u, e));
71        }
72    }
73    heap* merge(heap* curNd, heap* newNd) {
74        if (curNd == nullNd) return newNd;
75        heap* root = new heap;
76        memcpy(root, curNd, sizeof(heap));
77        if (newNd->edge->d < curNd->edge->d) {
78            root->edge = newNd->edge;
79            root->chd[2] = newNd->chd[2];
80            root->chd[3] = newNd->chd[3];
81            newNd->edge = curNd->edge;
82            newNd->chd[2] = curNd->chd[2];
83            newNd->chd[3] = curNd->chd[3];
84        }
85        if (root->chd[0]->dep < root->chd[1]->dep)
86            root->chd[0] = merge(root->chd[0], newNd);
87        else
88            root->chd[1] = merge(root->chd[1], newNd);
89        root->dep = max(root->chd[0]->dep,
90                      root->chd[1]->dep) +
91                      1;
92        return root;
93    }
94    vector<heap*> V;
95    void build() {
96        nullNd = new heap;
97        nullNd->dep = 0;
98        nullNd->edge = new nd;
99        fill(nullNd->chd, nullNd->chd + 4, nullNd);
100        while (not dfsQ.empty()) {
101            int u = dfsQ.front();
102            dfsQ.pop();
103            if (!nxt[u])
104                head[u] = nullNd;
105            else
106                head[u] = head[nxt[u]->v];
107            V.clear();
108            for (auto& e : g[u]) {
109                int v = e->v;
110                if (dst[v] == -1) continue;
111                e->d += dst[v] - dst[u];
112                if (nxt[u] != e) {
113                    heap* p = new heap;
114                    fill(p->chd, p->chd + 4, nullNd);
115                    p->dep = 1;
116                    p->edge = e;
117                    V.push_back(p);
118                }
119            }
120            if (V.empty()) continue;
121            make_heap(V.begin(), V.end(), cmp);
122            #define L(X) ((X << 1) + 1)
123            #define R(X) ((X << 1) + 2)
124            for (size_t i = 0; i < V.size(); i++) {
125                if (L(i) < V.size())
126                    V[i]->chd[2] = V[L(i)];
127                else
128                    V[i]->chd[2] = nullNd;
129                if (R(i) < V.size())
130                    V[i]->chd[3] = V[R(i)];
131                else
132                    V[i]->chd[3] = nullNd;
133            }
134            head[u] = merge(head[u], V.front());
135        }
136    }
137    vector<ll> ans;
138    void first_K() {
139        ans.clear();
140        priority_queue<node> Q;
141        if (dst[s] == -1) return;
142        ans.push_back(dst[s]);
143        if (head[s] != nullNd)
144            Q.push(node(head[s], dst[s] + head[s]->edge->d));
145        for (int _ = 1; _ < k and not Q.empty(); _++) {
146            node p = Q.top();
147            Q.pop();

```

```

148     ans.push_back(p.d);
149     if (head[p.H->edge->v] != nullNd) {
150         q.H = head[p.H->edge->v];
151         q.d = p.d + q.H->edge->d;
152         Q.push(q);
153     }
154     for (int i = 0; i < 4; i++)
155         if (p.H->chd[i] != nullNd) {
156             q.H = p.H->chd[i];
157             q.d = p.d - p.H->edge->d + p.H->chd
158                 [i]->edge->d;
159             Q.push(q);
160         }
161 }
162 void solve() { // ans[i] stores the i-th shortest
163     path
164     dijkstra();
165     build();
166     first_K(); // ans.size() might less than k
167 } solver;

```

5.12 System of Difference Constraints

```

1 vector<vector<pair<int, ll>>> G;
2 void add(int u, int v, ll w) {
3     G[u].emplace_back(make_pair(v, w));
4 }

```

- $x_u - x_v \leq c \Rightarrow \text{add}(v, u, c)$
- $x_u - x_v \geq c \Rightarrow \text{add}(u, v, -c)$
- $x_u - x_v = c \Rightarrow \text{add}(v, u, c), \text{add}(u, v, -c)$
- $x_u \geq c \Rightarrow \text{add super vertex } x_0 = 0, \text{ then } x_u - x_0 \geq c \Rightarrow \text{add}(u, 0, -c)$
- Don't forget non-negative constraints for every variable if specified implicitly.
- Interval sum \Rightarrow Use prefix sum to transform into differential constraints. Don't forget $S_{i+1} - S_i \geq 0$ if x_i needs to be non-negative.
- $\frac{x_u}{x_v} \leq c \Rightarrow \log x_u - \log x_v \leq \log c$

6 String

6.1 Aho Corasick

```

1 struct ACautomata {
2     struct Node {
3         int cnt; // 停在此節點的數量
4         Node *go[26], *fail, *dic;
5         // 子節點 fail指標 最近的模式結尾
6         Node() {
7             cnt = 0;
8             fail = 0;
9             dic = 0;
10            memset(go, 0, sizeof(go));
11        }
12    } pool[1048576], *root;
13    int nMem;
14    Node *new_Node() {
15        pool[nMem] = Node();
16        return &pool[nMem++];
17    }
18    void init() {
19        nMem = 0;
20        root = new_Node();
21    }
22    void add(const string &str) { insert(root, str, 0); }
23    void insert(Node *cur, const string &str, int pos) {
24        for (int i = pos; i < str.size(); i++) {

```

```

25         if (!cur->go[str[i] - 'a'])
26             cur->go[str[i] - 'a'] = new_Node();
27         cur = cur->go[str[i] - 'a'];
28     }
29     cur->cnt++;
30 }
31 void make_fail() { // 全部 add 完做
32     queue<Node *> que;
33     que.push(root);
34     while (!que.empty()) {
35         Node *fr = que.front();
36         que.pop();
37         for (int i = 0; i < 26; i++) {
38             if (fr->go[i]) {
39                 Node *ptr = fr->fail;
40                 while (ptr && !ptr->go[i]) ptr = ptr->fail;
41                 fr->go[i]->fail = ptr = (ptr ? ptr->go[i] : root);
42                 fr->go[i]->dic = (ptr->cnt ? ptr : ptr->dic);
43                 que.push(fr->go[i]);
44             }
45         }
46     }
47 }
48 // 出現過不同string的總數
49 int query_unique(const string &text) {
50     Node *p = root;
51     int ans = 0;
52     for (char ch : text) {
53         int i = ch - 'a';
54         while (p && !p->go[i]) p = p->fail;
55         p = p ? p->go[i] : root;
56         if (p->cnt) { ans += p->cnt; p->cnt = 0; }
57         for (Node *t = p->dic; t; t = t->dic) if (t->cnt) {
58             ans += t->cnt; t->cnt = 0;
59         }
60     }
61     return ans;
62 }
63 } AC;

```

6.2 KMP

```

1 vector<int> f;
2 // 沒匹配到可以退回哪裡
3 void buildFailFunction(string &s) {
4     f.resize(s.size(), -1);
5     for (int i = 1; i < s.size(); i++) {
6         int now = f[i - 1];
7         while (now != -1 and s[now + 1] != s[i]) now = f[now];
8         if (s[now + 1] == s[i]) f[i] = now + 1;
9     }
10 }
11 void KMPmatching(string &a, string &b) {
12     for (int i = 0, now = -1; i < a.size(); i++) {
13         while (a[i] != b[now + 1] and now != -1) now = f[now];
14         if (a[i] == b[now + 1]) now++;
15         if (now + 1 == b.size()) {
16             cout << "found a match start at position "
17                 << i - now << endl;
18             now = f[now];
19         }
20     }
21 }

```

6.3 Z Value

```

1 string is, it, s;
2 // is: 被搜尋 it: 要找的
3 int n;
4 vector<int> z;
5 // 計算每個位置 i 開始的字串，和 s 的共前綴長度
6 void init() {
7     cin >> is >> it;
8     s = it + '0' + is;

```

```

9   n = (int)s.size();
10  z.resize(n, 0);
11 }
12 void solve() {
13     int ans = 0;
14     z[0] = n;
15     for (int i = 1, l = 0, r = 0; i < n; i++) {
16         if (i <= r) z[i] = min(z[i - l], r - i + 1);
17         while (i + z[i] < n && s[z[i]] == s[i + z[i]])
18             z[i]++;
19         if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
20         if (z[i] == (int)it.size()) ans++;
21     }
22     cout << ans << endl;
23 }

```

6.4 Manacher

```

1  // 找最長回文
2  int n;
3  string S, s;
4  vector<int> m;
5  void manacher() {
6      s.clear();
7      s.resize(2 * n + 1, '.');
8      for (int i = 0, j = 1; i < n; i++, j += 2) s[j] = S[i];
9      m.clear();
10     m.resize(2 * n + 1, 0);
11     // m[i] := max k such that s[i-k, i+k] is
12     //      palindrome
13     int mx = 0, mxk = 0;
14     for (int i = 1; i < 2 * n + 1; i++) {
15         if (mx - (i - mx) >= 0) m[i] = min(m[mx - (i -
16             mx)], mx + mxk - i);
17         while (0 <= i - m[i] - 1 && i + m[i] + 1 < 2 *
18             n + 1 && s[i - m[i] - 1] == s[i + m[i] + 1]) m[i]
19             ++;
20         if (i + m[i] > mx + mxk) mx = i, mxk = m[i];
21     }
22 }
23 void init() {
24     cin >> S;
25     n = (int)S.size();
26 }
27 void solve() {
28     manacher();
29     int mx = 0, ptr = 0;
30     for (int i = 0; i < 2 * n + 1; i++)
31         if (mx < m[i]) {
32             mx = m[i];
33             ptr = i;
34         }
35     for (int i = ptr - mx; i <= ptr + mx; i++)
36         if (s[i] != '.') cout << s[i];
37     cout << endl;
38 }

```

6.5 Suffix Array

```

1  #define F first
2  #define S second
3  struct SuffixArray { // don't forget s += "$";
4      int n;
5      string s;
6      vector<int> suf, lcp, rk;
7      // 後綴陣列: suf[i] = 第 i 小的後綴起點
8      // LCP 陣列: lcp[i] = suf[i] 與 suf[i-1] 的最長共同
9      //      前綴長度
10     // rank 陣列: rk[i] = 起點在 i 的後綴的名次
11     vector<int> cnt, pos;
12     vector<pair<pair<int, int>, int>> buc[2];
13     void init(string _s) {
14         s = _s;
15         n = (int)s.size();
16         // resize(n): suf, rk, cnt, pos, lcp, buc[0~1]
17         suf.assign(n, 0);
18         rk.assign(n, 0);
19         lcp.assign(n, 0);
20         cnt.assign(n, 0);

```

```

20     pos.assign(n, 0);
21     buc[0].assign(n, {{0,0},0});
22     buc[1].assign(n, {{0,0},0});
23 }
24 void radix_sort() {
25     for (int t : {0, 1}) {
26         fill(cnt.begin(), cnt.end(), 0);
27         for (auto& i : buc[t]) cnt[(t ? i.F.F : i.F
28             .S)++]++;
29         for (int i = 0; i < n; i++)
30             pos[i] = (!i ? 0 : pos[i - 1] + cnt[i -
31                 1]);
32         for (auto& i : buc[t])
33             buc[t ^ 1][pos[(t ? i.F.F : i.F.S)++]
34                 ] = i;
35     }
36 }
37 bool fill_suf() {
38     bool end = true;
39     for (int i = 0; i < n; i++) suf[i] = buc[0][i].
40         S;
41     rk[suf[0]] = 0;
42     for (int i = 1; i < n; i++) {
43         int dif = (buc[0][i].F != buc[0][i - 1].F);
44         end &= dif;
45         rk[suf[i]] = rk[suf[i - 1]] + dif;
46     }
47     return end;
48 }
49 void sa() {
50     for (int i = 0; i < n; i++)
51         buc[0][i] = make_pair(make_pair(s[i], s[i]),
52             i);
53     sort(buc[0].begin(), buc[0].end());
54     if (fill_suf()) return;
55     for (int k = 0; (1 << k) < n; k++) {
56         for (int i = 0; i < n; i++)
57             buc[0][i] = make_pair(make_pair(rk[i],
58                 rk[(i + (1 << k)) % n]), i);
59         radix_sort();
60         if (fill_suf()) return;
61     }
62 }
63 void LCP() {
64     int k = 0;
65     for (int i = 0; i < n - 1; i++) {
66         if (rk[i] == 0) continue;
67         int pi = rk[i];
68         int j = suf[pi - 1];
69         while (i + k < n && j + k < n && s[i + k]
70             == s[j + k]) k++;
71         lcp[pi] = k;
72         k = max(k - 1, 0);
73     }
74 }
75 }
76 SuffixArray suffixarray;

```

6.6 Suffix Automaton

```

1  struct SAM {
2      struct State {
3          int next[26];
4          int link, len;
5          // suffix link, 指向最長真後綴所對應的狀態
6          // 該狀態代表的字串集合中的最長字串長度
7          State() : link(-1), len(0) { memset(next, -1,
8              sizeof next); }
9      };
10     vector<State> st;
11     int last;
12     vector<long long> occ; // 每個狀態的出現次數 (
13     //      endpos 個數)
14     vector<int> first_bkpos; // 出現在哪裡
15     SAM(int maxlen = 0) {
16         st.reserve(2 * maxlen + 5); st.push_back(State
17             ()); last = 0;
18         occ.reserve(2 * maxlen + 5); occ.push_back(0);
19         first_bkpos.push_back(-1);
20     }
21     void extend(int c) {

```



```

19     int cur = (int)st.size();
20     st.push_back(State());
21     occ.push_back(0);
22     first_bkpos.push_back(0);
23     st[cur].len = st[last].len + 1;
24     first_bkpos[cur] = st[cur].len - 1;
25     int p = last;
26     while (p != -1 && st[p].next[c] == -1) {
27         st[p].next[c] = cur;
28         p = st[p].link;
29     }
30     if (p == -1) {
31         st[cur].link = 0;
32     } else {
33         int q = st[p].next[c];
34         if (st[p].len + 1 == st[q].len) {
35             st[cur].link = q;
36         } else {
37             int clone = (int)st.size();
38             st.push_back(st[q]);
39             first_bkpos.push_back(first_bkpos[q]);
40             occ.push_back(0);
41             st[clone].len = st[p].len + 1;
42             while (p != -1 && st[p].next[c] == q) {
43                 st[p].next[c] = clone;
44                 p = st[p].link;
45             }
46             st[q].link = st[cur].link = clone;
47         }
48     }
49     last = cur;
50     occ[cur] += 1;
51 }
52 void finalize_occ() {
53     int m = (int)st.size();
54     vector<int> order(m);
55     iota(order.begin(), order.end(), 0);
56     sort(order.begin(), order.end(), [&](int a, int b){ return st[a].len > st[b].len; });
57     for (int v : order) {
58         int p = st[v].link;
59         if (p != -1) occ[p] += occ[v];
60     }
61 }
62 };

```

6.7 Minimum Rotation

```

1 // rotate(begin(s), begin(s)+minRotation(s), end(s))
2 // 找出字串的最小字典序旋轉
3 int minRotation(string s) {
4     int a = 0, n = s.size();
5     s += s;
6     for (int b = 0; b < n; b++)
7         for (int k = 0; k < n; k++) {
8             if (a + k == b || s[a + k] < s[b + k]) {
9                 b += max(0, k - 1);
10                break;
11            }
12            if (s[a + k] > s[b + k]) {
13                a = b;
14                break;
15            }
16        }
17     return a;
18 }

```

6.8 Lyndon Factorization

```

1 // Duval: 將字串唯一分解為字典序非遞增的 Lyndon 子字串
2 vector<string> duval(string const& s) {
3     int n = s.size();
4     int i = 0;
5     vector<string> factorization;
6     while (i < n) {
7         int j = i + 1, k = i;
8         while (j < n && s[k] <= s[j]) {
9             if (s[k] < s[j])
10                k = i;
11            else
12                k++;

```

```

13        j++;
14    }
15    while (i <= k) {
16        factorization.push_back(s.substr(i, j - k));
17        i += j - k;
18    }
19 }
20 return factorization; // O(n)
21 }

```

6.9 Rolling Hash

```

1 const ll C = 27;
2 inline int id(char c) { return c - 'a' + 1; }
3 struct RollingHash {
4     string s;
5     int n;
6     ll mod;
7     vector<ll> Cexp, hs;
8     RollingHash(string& _s, ll _mod) : s(_s), n((int)_s.size()), mod(_mod) {
9         Cexp.assign(n, 0);
10        hs.assign(n, 0);
11        Cexp[0] = 1;
12        for (int i = 1; i < n; i++) {
13            Cexp[i] = Cexp[i - 1] * C;
14            if (Cexp[i] >= mod) Cexp[i] %= mod;
15        }
16        hs[0] = id(s[0]);
17        for (int i = 1; i < n; i++) {
18            hs[i] = hs[i - 1] * C + id(s[i]);
19            if (hs[i] >= mod) hs[i] %= mod;
20        }
21    }
22    inline ll query(int l, int r) {
23        ll res = hs[r] - (l ? hs[l - 1] * Cexp[r - l + 1] : 0);
24        res = (res % mod + mod) % mod;
25        return res;
26    }
27 };

```

6.10 Trie

```

1 pii a[N][26];
2
3 void build(string &s) {
4     static int idx = 0;
5     int n = s.size();
6     for (int i = 0, v = 0; i < n; i++) {
7         pii &now = a[v][s[i] - 'a'];
8         if (now.first != -1)
9             v = now.first;
10        else
11            v = now.first = ++idx;
12        if (i == n - 1)
13            now.second++;
14    }
15 }

```

7 Geometry

7.1 Basic Operations

```

1 // typedef long long T;
2 typedef long double T;
3 const long double eps = 1e-12;
4
5 short sgn(T x) {
6     if (abs(x) < eps) return 0;
7     return x < 0 ? -1 : 1;
8 }
9
10 struct Pt {
11     T x, y;
12     Pt(T _x = 0, T _y = 0) : x(_x), y(_y) {}
13     Pt operator+(Pt a) { return Pt(x + a.x, y + a.y); }
14     Pt operator-(Pt a) { return Pt(x - a.x, y - a.y); }
15     Pt operator*(T a) { return Pt(x * a, y * a); }
16     Pt operator/(T a) { return Pt(x / a, y / a); }

```

```

17 T operator*(Pt a) { return x * a.x + y * a.y; }
18 T operator^(Pt a) { return x * a.y - y * a.x; }
19 bool operator<(Pt a) { return x < a.x || (x == a.x
    && y < a.y); }
20 // return sgn(x-a.x) < 0 || (sgn(x-a.x) == 0 && sgn
    (y-a.y) < 0); }
21 bool operator==(Pt a) { return sgn(x - a.x) == 0 &&
    sgn(y - a.y) == 0; }
22 };
23
24 Pt mv(Pt a, Pt b) { return b - a; }
25 T len2(Pt a) { return a * a; }
26 T dis2(Pt a, Pt b) { return len2(b - a); }
27 Pt rotate(Pt u) { return {-u.y, u.x}; }
28 Pt unit(Pt x) { return x / sqrtl(x * x); }
29 short ori(Pt a, Pt b) { return ((a ^ b) > 0) - ((a ^ b)
    < 0); }
30 bool onseg(Pt p, Pt l1, Pt l2) {
31     Pt a = mv(p, l1), b = mv(p, l2);
32     return ((a ^ b) == 0) && ((a * b) <= 0);
33 }
34 inline T cross(const Pt &a, const Pt &b, const Pt &c) {
35     return (b.x - a.x) * (c.y - a.y)
36         - (b.y - a.y) * (c.x - a.x);
37 }
38
39 long double polar_angle(Pt ori, Pt pt){
40     return atan2(pt.y - ori.y, pt.x - ori.x);
41 }
42 // slope to degree atan(Slope) * 180.0 / acos(-1.0);
43 bool argcmp(Pt u, Pt v) {
44     auto half = [](const Pt& p) {
45         return p.y > 0 || (p.y == 0 && p.x >= 0);
46     };
47     if (half(u) != half(v)) return half(u) < half(v);
48     return sgn(u ^ v) > 0;
49 }
50 int ori(Pt& o, Pt& a, Pt& b) {
51     return sgn((a - o) ^ (b - o));
52 }
53 struct Line {
54     Pt a, b;
55     Pt dir() { return b - a; }
56 };
57 int PtSide(Pt p, Line L) {
58     return sgn(ori(L.a, L.b, p)); // for int
59     return sgn(ori(L.a, L.b, p) / sqrt(len2(L.a - L.b))
60 );
61 }
62 bool PtOnSeg(Pt p, Line L) {
63     return PtSide(p, L) == 0 and sgn((p - L.a) * (p - L
64 .b)) <= 0;
65 }
66 Pt proj(Pt& p, Line& l) {
67     Pt d = l.b - l.a;
68     T d2 = len2(d);
69     if (sgn(d2) == 0) return l.a;
70     T t = ((p - l.a) * d) / d2;
71     return l.a + d * t;
72 }
73
74 struct Cir {
75     Pt o;
76     T r;
77 };
78 bool disjunct(Cir a, Cir b) {
79     return sgn(sqrtl(len2(a.o - b.o)) - a.r - b.r) >=
80 0;
81 }
82 bool contain(Cir a, Cir b) {
83     return sgn(a.r - b.r - sqrtl(len2(a.o - b.o))) >=
84 0;
85 }
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

7.2 Sort by Angle

```

if (ud(a) != ud(b)) return ud(a) < ud(b);
return (a ^ b) > 0;
});

```

7.3 Intersection

```

bool line_intersect_check(Pt p1, Pt p2, Pt q1, Pt q2) {
    if (onseg(p1, q1, q2) || onseg(p2, q1, q2) || onseg
        (q1, p1, p2) || onseg(q2, p1, p2)) return true;
    Pt p = mv(p1, p2), q = mv(q1, q2);
    return (ori(p, mv(p1, q1)) * ori(p, mv(p1, q2)) <
        0) && (ori(q, mv(q1, p1)) * ori(q, mv(q1, p2))
        < 0);
}
// long double
Pt line_intersect(Pt a1, Pt a2, Pt b1, Pt b2) {
    Pt da = mv(a1, a2), db = mv(b1, b2);
    T det = da ^ db;
    if (sgn(det) == 0) { // parallel
        // return Pt(NAN, NAN);
    }
    T t = ((b1 - a1) ^ db) / det;
    return a1 + da * t;
}
vector<Pt> CircleInter(Cir a, Cir b) {
    double d2 = len2(a.o - b.o), d = sqrt(d2);
    if (d < max(a.r, b.r) - min(a.r, b.r) || d > a.r +
        b.r) return {};
    Pt u = (a.o + b.o) / 2 + (a.o - b.o) * ((b.r * b.r
        - a.r * a.r) / (2 * d2));
    double A = sqrt((a.r + b.r + d) * (a.r - b.r + d) *
        (a.r + b.r - d) * (-a.r + b.r + d));
    Pt v = rotate(b.o - a.o) * A / (2 * d2);
    if (sgn(v.x) == 0 and sgn(v.y) == 0) return {u};
    return {u - v, u + v}; // counter clockwise of a
}
vector<Pt> CircleLineInter(Cir c, Line l) {
    Pt H = proj(c.o, l);
    Pt dir = unit(l.b - l.a);
    T h = sqrtl(len2(H - c.o));
    if (sgn(h - c.r) > 0) return {};
    T d = sqrtl(max((T)0, c.r * c.r - h * h));
    if (sgn(d) == 0) return {H};
    return {H - dir * d, H + dir * d};
}

```

7.4 Polygon Area

```

// 2 * area
T dbPoly_area(vector<Pt>& e) {
    T res = 0;
    int sz = e.size();
    for (int i = 0; i < sz; i++) {
        res += e[i] ^ e[(i + 1) % sz];
    }
    return abs(res);
}

```

7.5 Convex Hull

```

vector<Pt> convexHull(vector<Pt> pts) {
    vector<Pt> hull;
    sort(pts.begin(), pts.end());
    for (int i = 0; i < 2; i++) {
        int b = hull.size();
        for (auto ei : pts) {
            while (hull.size() - b >= 2 && ori(mv(hull[
                hull.size() - 2], hull.back()), mv(hull[
                hull.size() - 2], ei)) == -1) {
                hull.pop_back();
            }
            hull.emplace_back(ei);
        }
        hull.pop_back();
        reverse(pts.begin(), pts.end());
    }
    return hull;
}

```

7.6 Point In Convex

```

1 bool point_in_convex(const vector<Pt> &C, Pt p, bool
  strict = true) {
2   // only works when no three point are collinear
3   int n = C.size();
4   int a = 1, b = n - 1, r = !strict;
5   if (n == 0) return false;
6   if (n < 3) return r && onseg(p, C[0], C.back());
7   if (ori(mv(C[0], C[a]), mv(C[0], C[b])) > 0) swap(a
    , b);
8   if (ori(mv(C[0], C[a]), mv(C[0], p)) >= r || ori(mv
    (C[0], C[b]), mv(C[0], p)) <= -r) return false;
9   while (abs(a - b) > 1) {
10    int c = (a + b) / 2;
11    if (ori(mv(C[0], C[c]), mv(C[0], p)) > 0) b = c
        ;
12    else a = c;
13  }
14  return ori(mv(C[a], C[b]), mv(C[a], p)) < r;
15 }

```

7.7 Point Segment Distance

```

1 double point_segment_dist(Pt q0, Pt q1, Pt p) {
2   if (q0 == q1) {
3     double dx = double(p.x - q0.x);
4     double dy = double(p.y - q0.y);
5     return sqrt(dx * dx + dy * dy);
6   }
7   T d1 = (q1 - q0) * (p - q0);
8   T d2 = (q0 - q1) * (p - q1);
9   if (d1 >= 0 && d2 >= 0) {
10    double area = fabs(double((q1 - q0) ^ (p - q0))
        );
11    double base = sqrt(double(dis2(q0, q1)));
12    return area / base;
13  }
14  double dx0 = double(p.x - q0.x), dy0 = double(p.y -
    q0.y);
15  double dx1 = double(p.x - q1.x), dy1 = double(p.y -
    q1.y);
16  return min(sqrt(dx0 * dx0 + dy0 * dy0), sqrt(dx1 *
    dx1 + dy1 * dy1));
17 }

```

7.8 Point in Polygon

```

1 short inPoly(vector<Pt>& pts, Pt p) {
2   // 0=Bound 1=In -1=Out
3   int n = pts.size();
4   for (int i = 0; i < pts.size(); i++) if (onseg(p,
    pts[i], pts[(i + 1) % n])) return 0;
5   int cnt = 0;
6   for (int i = 0; i < pts.size(); i++) if (
    line_intersect_check(p, Pt(p.x + 1, p.y + 2e9),
    pts[i], pts[(i + 1) % n])) cnt ^= 1;
7   return (cnt ? 1 : -1);
8 }

```

7.9 Minimum Euclidean Distance

```

1 long long Min_Euclidean_Dist(vector<Pt> &pts) {
2   sort(pts.begin(), pts.end());
3   set<pair<long long, long long>> s;
4   s.insert({pts[0].y, pts[0].x});
5   long long l = 0, best = LLONG_MAX;
6   for (int i = 1; i < (int)pts.size(); i++) {
7     Pt now = pts[i];
8     long long lim = (long long)ceil(sqrtl((long
        double)best));
9     while (now.x - pts[l].x > lim) {
10      s.erase({pts[l].y, pts[l].x}); l++;
11    }
12    auto low = s.lower_bound({now.y - lim,
        LLONG_MIN});
13    auto high = s.upper_bound({now.y + lim,
        LLONG_MAX});
14    for (auto it = low; it != high; it++) {
15      long long dy = it->first - now.y;
16      long long dx = it->second - now.x;
17      best = min(best, dx * dx + dy * dy);
18    }

```

```

19    s.insert({now.y, now.x});
20  }
21  return best;
22 }

```

7.10 Minkowski Sum

```

1 void reorder(vector<Pt> &P) {
2   rotate(P.begin(), min_element(P.begin(), P.end(),
    [&](Pt a, Pt b) { return make_pair(a.y, a.x) <
    make_pair(b.y, b.x); }), P.end());
3 }
4 vector<Pt> Minkowski(vector<Pt> P, vector<Pt> Q) {
5   // P, Q: convex polygon
6   reorder(P), reorder(Q);
7   int n = P.size(), m = Q.size();
8   P.push_back(P[0]), P.push_back(P[1]), Q.push_back(Q
    [0]), Q.push_back(Q[1]);
9   vector<Pt> ans;
10  for (int i = 0, j = 0; i < n || j < m; ) {
11    ans.push_back(P[i] + Q[j]);
12    auto val = (P[i + 1] - P[i]) ^ (Q[j + 1] - Q[j]);
13    if (val >= 0) i++;
14    if (val <= 0) j++;
15  }
16  return ans;
17 }

```

7.11 Lower Concave Hull

```

1 struct Line {
2   mutable ll m, b, p;
3   bool operator<(const Line& o) const { return m < o.m;
    }
4   bool operator<(ll x) const { return p < x; }
5 };
6 struct LineContainer : multiset<Line, less<>> {
7   // (for doubles, use inf = 1/.0, div(a,b) = a/b)
8   const ll inf = LLONG_MAX;
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()) { x->p = inf; return false; }
13    if (x->m == y->m) x->p = x->b > y->b ? inf : -inf;
14    else x->p = div(y->b - x->b, x->m - y->m);
15    return x->p >= y->p;
16  }
17  void add(ll m, ll b) {
18    auto z = insert({m, b, 0}), y = z++, x = y;
19    while (isect(y, z)) z = erase(z);
20    if (x != begin() && isect(--x, y)) isect(x, y =
    erase(y));
21    while ((y = x) != begin() && (--x)->p >= y->p)
22      isect(x, erase(y));
23  }
24  ll query(ll x) {
25    assert(!empty());
26    auto l = *lower_bound(x);
27    return l.m * x + l.b;
28  }
29 };
30 };

```

7.12 Pick's Theorem

Consider a polygon which vertices are all lattice points.
 Let i = number of points inside the polygon.
 Let b = number of points on the boundary of the polygon.

Then we have the following formula:

$$Area = i + \frac{b}{2} - 1$$

7.13 Rotating SweepLine

```

1 double cross(const Pt &a, const Pt &b) {
2   return a.x*b.y - a.y*b.x;
3 }
4 int rotatingCalipers(const vector<Pt>& hull) {
5   int m = hull.size();

```

```

6   if (m < 2) return 0;
7   int j = 1;
8   T maxd = 0;
9   for (int i = 0; i < m; ++i) {
10      int ni = (i + 1) % m;
11      while (abs(cross({hull[ni].x - hull[i].x, hull[ni].y - hull[i].y}, {hull[(j+1)%m].x - hull[i].x, hull[(j+1)%m].y - hull[i].y})) > abs(cross({hull[ni].x - hull[i].x, hull[ni].y - hull[i].y}, {hull[j].x - hull[i].x, hull[j].y - hull[i].y}))) {
12          j = (j + 1) % m;
13      }
14      maxd = max(maxd, dis2(hull[i], hull[j]));
15      maxd = max(maxd, dis2(hull[ni], hull[j]));
16  }
17  return maxd; // TODO
18 }

```

7.14 Half Plane Intersection

```

1 bool cover(Line& L, Line& P, Line& Q) {
2     long double u = (Q.a - P.a) ^ Q.dir();
3     long double v = P.dir() ^ Q.dir();
4     long double x = P.dir().x * u + (P.a - L.a).x * v;
5     long double y = P.dir().y * u + (P.a - L.a).y * v;
6     return sgn(x * L.dir().y - y * L.dir().x) * sgn(v) >= 0;
7 }
8 vector<Line> HPI(vector<Line> P) {
9     sort(P.begin(), P.end(), [&](Line& l, Line& m) {
10         if (argcmp(l.dir(), m.dir()) return true;
11         if (argcmp(m.dir(), l.dir()) return false;
12         return ori(m.a, m.b, l.a) > 0;
13     });
14     int l = 0, r = -1;
15     for (size_t i = 0; i < P.size(); ++i) {
16         if (i && !argcmp(P[i - 1].dir(), P[i].dir())) continue;
17         while (l < r && cover(P[i], P[r - 1], P[r])) --r;
18         while (l < r && cover(P[i], P[l], P[l + 1])) ++l;
19         P[++r] = P[i];
20     }
21     while (l < r && cover(P[l], P[r - 1], P[r])) --r;
22     while (l < r && cover(P[r], P[l], P[l + 1])) ++l;
23     if (r - l <= 1 || !argcmp(P[l].dir(), P[r].dir())) return {};
24     if (cover(P[l + 1], P[l], P[r])) return {};
25     return vector<Line>(P.begin() + l, P.begin() + r + 1);
26 }

```

7.15 Minimum Enclosing Circle

```

1 const int INF = 1e9;
2 Pt circumcenter(Pt A, Pt B, Pt C) {
3     // a1(x-A.x) + b1(y-A.y) = c1
4     // a2(x-A.x) + b2(y-A.y) = c2
5     // solve using Cramer's rule
6     T a1 = B.x - A.x, b1 = B.y - A.y, c1 = dis2(A, B) / 2.0;
7     T a2 = C.x - A.x, b2 = C.y - A.y, c2 = dis2(A, C) / 2.0;
8     T D = Pt(a1, b1) ^ Pt(a2, b2);
9     T Dx = Pt(c1, b1) ^ Pt(c2, b2);
10    T Dy = Pt(a1, c1) ^ Pt(a2, c2);
11    if (D == 0) return Pt(-INF, -INF);
12    return A + Pt(Dx / D, Dy / D);
13 }
14 Pt center;
15 T r2;
16 void minEncloseCircle(vector<Pt> pts) {
17     mt19937 gen(chrono::steady_clock::now().time_since_epoch().count());
18     shuffle(pts.begin(), pts.end(), gen);
19     center = pts[0], r2 = 0;
20 }

```

```

21 for (int i = 0; i < pts.size(); i++) {
22     if (dis2(center, pts[i]) <= r2) continue;
23     center = pts[i], r2 = 0;
24     for (int j = 0; j < i; j++) {
25         if (dis2(center, pts[j]) <= r2) continue;
26         center = (pts[i] + pts[j]) / 2.0;
27         r2 = dis2(center, pts[i]);
28         for (int k = 0; k < j; k++) {
29             if (dis2(center, pts[k]) <= r2) continue;
30             center = circumcenter(pts[i], pts[j], pts[k]);
31             r2 = dis2(center, pts[i]);
32         }
33     }
34 }
35 }

```

7.16 Union of Circles

```

1 // Area[i] : area covered by at least i circle
2 vector<T> CircleUnion(const vector<Cir> &C) {
3     const int n = C.size();
4     vector<T> Area(n + 1);
5     auto check = [&](int i, int j) {
6         if (!contain(C[i], C[j])) return false;
7         return sgn(C[i].r - C[j].r) > 0 or (sgn(C[i].r - C[j].r) == 0 and i < j);
8     };
9     struct Teve {
10         double ang; int add; Pt p;
11         bool operator<(const Teve &b) { return ang < b.ang; }
12     };
13     auto ang = [&](Pt p) { return atan2(p.y, p.x); };
14     for (int i = 0; i < n; i++) {
15         int cov = 1;
16         vector<Teve> event;
17         for (int j = 0; j < n; j++) if (i != j) {
18             if (check(j, i)) cov++;
19             else if (!check(i, j) and !disjunct(C[i], C[j])) {
20                 auto I = CircleInter(C[i], C[j]);
21                 assert(I.size() == 2);
22                 double a1 = ang(I[0] - C[i].o), a2 = ang(I[1] - C[i].o);
23                 event.push_back({a1, 1, I[0]});
24                 event.push_back({a2, -1, I[1]});
25                 if (a1 > a2) cov++;
26             }
27         }
28         if (event.empty()) {
29             Area[cov] += acos(-1) * C[i].r * C[i].r;
30             continue;
31         }
32         sort(event.begin(), event.end());
33         event.push_back(event[0]);
34         for (int j = 0; j + 1 < event.size(); j++) {
35             cov += event[j].add;
36             Area[cov] += (event[j].p ^ event[j + 1].p) / 2.;
37             double theta = event[j + 1].ang - event[j].ang;
38             if (theta < 0) theta += 2 * acos(-1);
39             Area[cov] += (theta - sin(theta)) * C[i].r * C[i].r / 2.;
40         }
41     }
42     return Area;
43 }

```

7.17 Area Of Circle Polygon

```

1 double AreaOfCirclePoly(Cir C, vector<Pt> &P) {
2     auto arg = [&](Pt p, Pt q) { return atan2(p ^ q, p * q); };
3     double r2 = (double)(C.r * C.r / 2);
4     auto tri = [&](Pt p, Pt q) {
5         Pt d = q - p;
6         T a = (d * p) / (d * d);
7         T b = ((p * p) - C.r * C.r) / (d * d);

```

```

8   T det = a * a - b;
9   if (det <= 0) return (double)(arg(p, q) * r2);
10  T s = max((T)0.0L, -a - sqrtl(det));
11  T t = min((T)1.0L, -a + sqrtl(det));
12  if (t < 0 || 1 <= s) return (double)(arg(p, q)
13      * r2);
14  Pt u = p + d * s, v = p + d * t;
15  return (double)(arg(p, u) * r2 + (u ^ v) / 2 +
16      arg(v, q) * r2);
17  };
18  long double sum = 0.0L;
19  for (int i = 0; i < (int)P.size(); i++)
20      sum += tri(P[i] - C.o, P[(i + 1) % P.size()] -
21          C.o);
22  return (double)fabsl(sum);

```

7.18 3D Point

```

1  struct Pt {
2      double x, y, z;
3      Pt(double _x = 0, double _y = 0, double _z = 0): x(_x
4          ), y(_y), z(_z){}
5      Pt operator + (const Pt &o) const
6      { return Pt(x + o.x, y + o.y, z + o.z); }
7      Pt operator - (const Pt &o) const
8      { return Pt(x - o.x, y - o.y, z - o.z); }
9      Pt operator * (const double &k) const
10     { return Pt(x * k, y * k, z * k); }
11     Pt operator / (const double &k) const
12     { return Pt(x / k, y / k, z / k); }
13     double operator * (const Pt &o) const
14     { return x * o.x + y * o.y + z * o.z; }
15     Pt operator ^ (const Pt &o) const
16     { return {Pt(y * o.z - z * o.y, z * o.x - x * o.z, x
17         * o.y - y * o.x)}; }
18 };
19 double abs2(Pt o) { return o * o; }
20 double abs(Pt o) { return sqrt(abs2(o)); }
21 Pt cross3(Pt a, Pt b, Pt c)
22 { return (b - a) ^ (c - a); }
23 double area(Pt a, Pt b, Pt c)
24 { return abs(cross3(a, b, c)); }
25 double volume(Pt a, Pt b, Pt c, Pt d)
26 { return cross3(a, b, c) * (d - a); }
27 bool coplaner(Pt a, Pt b, Pt c, Pt d)
28 { return sign(volume(a, b, c, d)) == 0; }
29 Pt proj(Pt o, Pt a, Pt b, Pt c) // o proj to plane abc
30 { Pt n = cross3(a, b, c);
31   return o - n * ((o - a) * (n / abs2(n))); }
32 Pt line_plane_intersect(Pt u, Pt v, Pt a, Pt b, Pt c) {
33     // intersection of line uv and plane abc
34     Pt n = cross3(a, b, c);
35     double s = n * (u - v);
36     if (sign(s) == 0) return {-1, -1, -1}; // not found
37     return v + (u - v) * ((n * (a - v)) / s); }
38 Pt rotateAroundAxis(Pt v, Pt axis, double theta) {
39     axis = axis / abs(axis); // axis must be unit
40     // vector
41     double cosT = cos(theta);
42     double sinT = sin(theta);
43     Pt term1 = v * cosT;
44     Pt term2 = (axis ^ v) * sinT;
45     Pt term3 = axis * ((axis * v) * (1 - cosT));
46     return term1 + term2 + term3;
47 }

```

8 Number Theory

8.1 FFT

```

1  typedef complex<double> cp;
2
3  const double pi = acos(-1);
4  const int NN = 131072;
5
6  struct FastFourierTransform {
7      /*
8          Iterative Fast Fourier Transform
9          How this works? Look at this

```

```

0th recursion 0(000) 1(001) 2(010)
3(011) 4(100) 5(101) 6(110)
7(111)
1th recursion 0(000) 2(010) 4(100)
6(110) | 1(011) 3(011) 5(101)
7(111)
2th recursion 0(000) 4(100) | 2(010)
6(110) | 1(011) 5(101) | 3(011)
7(111)
3th recursion 0(000) | 4(100) | 2(010) |
6(110) | 1(011) | 5(101) | 3(011) |
7(111)
All the bits are reversed => We can save
the reverse of the numbers in an array!
*/
int n, rev[NN];
cp omega[NN], iomega[NN];
void init(int n_) {
    n = n_;
    for (int i = 0; i < n_; i++) {
        // Calculate the nth roots of unity
        omega[i] = cp(cos(2 * pi * i / n_), sin(2 *
            pi * i / n_));
        iomega[i] = conj(omega[i]);
    }
    int k = __lg(n_);
    for (int i = 0; i < n_; i++) {
        int t = 0;
        for (int j = 0; j < k; j++) {
            if (i & (1 << j)) t |= (1 << (k - j -
                1));
        }
        rev[i] = t;
    }
}

void transform(vector<cp> &a, cp *xomega) {
    for (int i = 0; i < n; i++)
        if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int len = 2; len <= n; len <= 1) {
        int mid = len >> 1;
        int r = n / len;
        for (int j = 0; j < n; j += len)
            for (int i = 0; i < mid; i++) {
                cp tmp = xomega[r * i] * a[j + mid
                    + i];
                a[j + mid + i] = a[j + i] - tmp;
                a[j + i] = a[j + i] + tmp;
            }
    }
}

void fft(vector<cp> &a) { transform(a, omega); }
void ifft(vector<cp> &a) {
    transform(a, iomega);
    for (int i = 0; i < n; i++) a[i] /= n;
}

} FFT;

const int MAXN = 262144;
// (must be 2^k)
// 262144, 524288, 1048576, 2097152, 4194304
// before any usage, run pre_fft() first
typedef long double ld;
typedef complex<ld> cplx; // real() ,imag()
const ld PI = acosl(-1);
const cplx I(0, 1);
cplx omega[MAXN + 1];
void pre_fft() {
    for (int i = 0; i <= MAXN; i++) {
        omega[i] = exp(i * 2 * PI / MAXN * I);
    }
}

// n must be 2^k
void fft(int n, cplx a[], bool inv = false) {
    int basic = MAXN / n;
    int theta = basic;
    for (int m = n; m >= 2; m >>= 1) {
        int mh = m >> 1;
        for (int i = 0; i < mh; i++) {
            cplx w = omega[inv ? MAXN - (i * theta %
                MAXN) : i * theta % MAXN];

```



```

79     for (int j = i; j < n; j += m) {
80         int k = j + mh;
81         cplx x = a[j] - a[k];
82         a[j] += a[k];
83         a[k] = w * x;
84     }
85 }
86 theta = (theta * 2) % MAXN;
87 }
88 int i = 0;
89 for (int j = 1; j < n - 1; j++) {
90     for (int k = n >> 1; k > (i ^ k); k >>= 1);
91     if (j < i) swap(a[i], a[j]);
92 }
93 if (inv) {
94     for (i = 0; i < n; i++) a[i] /= n;
95 }
96 }
97 cplx arr[MAXN + 1];
98 inline void mul(int _n, long long a[], int _m, long
99     long b[], long long ans[]) {
100     int n = 1, sum = _n + _m - 1;
101     while (n < sum) n <= 1;
102     for (int i = 0; i < n; i++) {
103         double x = (i < _n ? a[i] : 0), y = (i < _m ? b
104             [i] : 0);
105         arr[i] = complex<double>(x + y, x - y);
106     }
107     fft(n, arr);
108     for (int i = 0; i < n; i++) arr[i] = arr[i] * arr[i
109         ];
110     fft(n, arr, true);
111     for (int i = 0; i < sum; i++) ans[i] = (long long
112         int)(arr[i].real() / 4 + 0.5);
113 }
114 long long a[MAXN];
115 long long b[MAXN];
116 long long ans[MAXN];
117 int a_length;
118 int b_length;

```

8.2 Pollard's rho

```

1 ll add(ll x, ll y, ll p) {
2     return (x + y) % p;
3 }
4 ll qMul(ll x, ll y, ll mod) {
5     ll ret = x * y - (ll)((long double)x / mod * y) *
6     mod;
7     return ret < 0 ? ret + mod : ret;
8 }
9 ll f(ll x, ll mod) { return add(qMul(x, x, mod), 1, mod
10 ); }
11 pollard_rho(ll n) {
12     if (!(n & 1)) return 2;
13     while (true) {
14         ll y = 2, x = rand() % (n - 1) + 1, res = 1;
15         for (int sz = 2; res == 1; sz *= 2) {
16             for (int i = 0; i < sz && res <= 1; i++) {
17                 x = f(x, n);
18                 res = __gcd(llabs(x - y), n);
19             }
20             y = x;
21         }
22         if (res != 0 && res != n) return res;
23     }
24 }
25 vector<ll> ret;
26 void fact(ll x) {
27     if (miller_rabin(x)) {
28         ret.push_back(x);
29         return;
30     }
31     ll f = pollard_rho(x);
32     fact(f);
33     fact(x / f);
34 }

```

8.3 Miller Rabin

```

1 // n < 4,759,123,141      3 : 2, 7, 61
2 // n < 1,122,004,669,633 4 : 2, 13, 23, 1662803
3 // n < 3,474,749,660,383 6 : pirms <= 13
4 // n < 2^64              7 :
5 // 2, 325, 9375, 28178, 450775, 9780504, 1795265022
6 bool witness(ll a, ll n, ll u, int t) {
7     if (!(a % n)) return 0;
8     ll x = mypow(a, u, n);
9     for (int i = 0; i < t; i++) {
10         ll nx = mul(x, x, n);
11         if (nx == 1 && x != 1 && x != n - 1) return 1;
12         x = nx;
13     }
14     return x != 1;
15 }
16 bool miller_rabin(ll n, int s = 100) {
17     // iterate s times of witness on n
18     // return 1 if prime, 0 otherwise
19     if (n < 2) return 0;
20     if (!(n & 1)) return n == 2;
21     ll u = n - 1;
22     int t = 0;
23     while (!(u & 1)) u >>= 1, t++;
24     while (s--) {
25         ll a = randll() % (n - 1) + 1;
26         if (witness(a, n, u, t)) return 0;
27     }
28     return 1;
29 }

```

8.4 Fast Power

Note: $a^n \equiv a^{(n \bmod (p-1))} \pmod{p}$

8.5 Extend GCD

```

1 ll GCD;
2 pll extgcd(ll a, ll b) {
3     if (b == 0) {
4         GCD = a;
5         return pll{1, 0};
6     }
7     pll ans = extgcd(b, a % b);
8     return pll{ans.S, ans.F - a / b * ans.S};
9 }
10 pll bezout(ll a, ll b, ll c) {
11     bool negx = (a < 0), negy = (b < 0);
12     pll ans = extgcd(abs(a), abs(b));
13     if (c % GCD != 0) return pll{-LLINF, -LLINF};
14     return pll{ans.F * c / GCD * (negx ? -1 : 1),
15         ans.S * c / GCD * (negy ? -1 : 1)};
16 }
17 ll inv(ll a, ll p) {
18     if (p == 1) return -1;
19     pll ans = bezout(a % p, -p, 1);
20     if (ans == pll{-LLINF, -LLINF}) return -1;
21     return (ans.F % p + p) % p;
22 }

```

8.6 Mu + Phi

```

1 const int maxn = 1e6 + 5;
2 ll f[maxn];
3 vector<int> lpf, prime;
4 void build() {
5     lpf.clear();
6     lpf.resize(maxn, 1);
7     prime.clear();
8     f[1] = ...; /* mu[1] = 1, phi[1] = 1 */
9     for (int i = 2; i < maxn; i++) {
10         if (lpf[i] == 1) {
11             lpf[i] = i;
12             prime.emplace_back(i);
13             f[i] = ...; /* mu[i] = 1, phi[i] = i-1 */
14         }
15         for (auto& j : prime) {
16             if (i * j >= maxn) break;
17             lpf[i * j] = j;
18             if (i % j == 0)
19                 f[i * j] = ...; /* 0, phi[i]*j */
20             else
21                 f[i * j] = ...; /* -mu[i], phi[i]*phi[j] */
22         }
23     }
24 }

```



```

22         if (j >= lpf[i]) break;
23     }
24 }
25 }

```

8.7 Discrete Log

```

1 long long mod_pow(long long a, long long e, long long p)
2 {
3     long long r = 1 % p;
4     while(e){
5         if(e & 1) r = (__int128)r * a % p;
6         a = (__int128)a * a % p;
7         e >>= 1;
8     }
9     return r;
10 }
11 long long mod_inv(long long a, long long p){
12     return mod_pow((a%p+p)%p, p-2, p);
13 }
14 // BSGS: solve a^x = y (mod p), gcd(a,p)=1, p prime,
15 // return minimal x>=0, or -1 if no solution
16 long long bsgs(long long a, long long y, long long p){
17     a%=p; y%=p;
18     if(y==1%p) return 0; // x=0
19     long long m = (long long)ceil(sqrt((long double)p));
20     ;
21     // baby steps: a^j
22     unordered_map<long long, long long> table;
23     table.reserve(m*2);
24     long long cur = 1%p;
25     for(long long j=0; j<m; ++j){
26         if(!table.count(cur)) table[cur]=j;
27         cur = (__int128)cur * a % p;
28     }
29     long long am = mod_pow(a, m, p);
30     long long am_inv = mod_inv(am, p);
31     long long gamma = y % p;
32     for(long long i=0; i<m; ++i){
33         auto it = table.find(gamma);
34         if(it != table.end()){
35             long long x = i*m + it->second;
36             return x;
37         }
38         gamma = (__int128)gamma * am_inv % p;
39     }
40     return -1;
41 }

```

8.8 sqrt mod

```

1 // the Jacobi symbol is a generalization of the
2 // Legendre symbol,
3 // such that the bottom doesn't need to be prime.
4 // (n/p) -> same as legendre
5 // (n/ab) = (n/a)(n/b)
6 // work with long long
7 int Jacobi(int a, int m) {
8     int s = 1;
9     for (; m > 1; ) {
10         a %= m;
11         if (a == 0) return 0;
12         const int r = __builtin_ctz(a);
13         if ((r & 1) && ((m + 2) & 4)) s = -s;
14         a >>= r;
15         if (a & m & 2) s = -s;
16         swap(a, m);
17     }
18     return s;
19 }
20 // solve x^2 = a (mod p)
21 // 0: a == 0
22 // -1: a isn't a quad res of p
23 // else: return X with X^2 % p == a
24 // doesn't work with long long
25 int QuadraticResidue(int a, int p) {
26     if (p == 2) return a & 1;
27     if (int jc = Jacobi(a, p); jc <= 0) return jc;
28     int b, d;
29     for (; ; ) {
30         b = rand() % p;
31         d = (1LL * b * b + p - a) % p;

```

```

31         if (Jacobi(d, p) == -1) break;
32     }
33     int f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
34     for (int e = (1LL + p) >> 1; e >>= 1) {
35         if (e & 1) {
36             tmp = (1LL * g0 * f0 + 1LL * d * (1LL * g1
37                 * f1 % p)) % p;
38             g1 = (1LL * g0 * f1 + 1LL * g1 * f0) % p;
39             g0 = tmp;
40         }
41         tmp = (1LL * f0 * f0 + 1LL * d * (1LL * f1 * f1
42             % p)) % p;
43         f1 = (2LL * f0 * f1) % p;
44         f0 = tmp;
45     }
46     return g0;
47 }

```

8.9 Primitive Root

```

1 unsigned long long primitiveRoot(ull p) {
2     auto fac = factor(p - 1);
3     sort(all(fac));
4     fac.erase(unique(all(fac)), fac.end());
5     auto test = [p, fac](ull x) {
6         for(ull d : fac)
7             if (modpow(x, (p - 1) / d, p) == 1)
8                 return false;
9         return true;
10    };
11    uniform_int_distribution<unsigned long long> unif
12        (1, p - 1);
13    unsigned long long root;
14    while(!test(root = unif(rng)));
15    return root;
16 }

```

8.10 LinearSieve

```

1 const int C = 1e7 + 2;
2 int mo[C], lp[C], phi[C], isp[C];
3 vector<int> prime;
4 void sieve() {
5     mo[1] = phi[1] = 1;
6     for(int i = 1; i < C; i++) lp[i] = 1;
7     for(int i = 2; i < C; i++) {
8         if(lp[i] == 1) {
9             lp[i] = i;
10            prime.push_back(i);
11            isp[i] = 1;
12            mo[i] = -1;
13            phi[i] = i - 1;
14        }
15        for(int p : prime) {
16            if(i * p >= C) break;
17            lp[i * p] = p;
18            if(i % p == 0) {
19                phi[p * i] = phi[i] * p;
20                break;
21            }
22            phi[i * p] = phi[i] * (p - 1);
23            mo[i * p] = mo[i] * mo[p];
24        }
25    }
26 }

```

8.11 Other Formulas

- Inversion:
 $aa^{-1} \equiv 1 \pmod{m}$. a^{-1} exists iff $\gcd(a, m) = 1$.
- Linear inversion:
 $a^{-1} \equiv (m - \lfloor \frac{m}{a} \rfloor) \times (m \bmod a)^{-1} \pmod{m}$
- Fermat's little theorem:
 $a^p \equiv a \pmod{p}$ if p is prime.
- Euler function:
 $\phi(n) = n \prod_{p|n} \frac{p-1}{p}$

- Euler theorem:
 $a^{\phi(n)} \equiv 1 \pmod{n}$ if $\gcd(a, n) = 1$.

- Extended Euclidean algorithm:

$$ax + by = \gcd(a, b) = \gcd(b, a \bmod b) = \gcd(b, a - \lfloor \frac{a}{b} \rfloor b) = bx_1 + (a - \lfloor \frac{a}{b} \rfloor b)y_1 = ay_1 + b(x_1 - \lfloor \frac{a}{b} \rfloor y_1)$$

- Divisor function:

$$\sigma_x(n) = \sum_{d|n} d^x \cdot n = \prod_{i=1}^r p_i^{a_i}.$$

$$\sigma_x(n) = \prod_{i=1}^r \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1} \text{ if } x \neq 0. \sigma_0(n) = \prod_{i=1}^r (a_i + 1).$$

- Chinese remainder theorem (Coprime Moduli):

$$x \equiv a_i \pmod{m_i}.$$

$$M = \prod m_i. M_i = M/m_i. t_i = M_i^{-1}.$$

$$x = kM + \sum a_i t_i M_i, k \in \mathbb{Z}.$$

- Chinese remainder theorem:

$$x \equiv a_1 \pmod{m_1}, x \equiv a_2 \pmod{m_2} \Rightarrow x = m_1 p + a_1 = m_2 q + a_2 \Rightarrow m_1 p - m_2 q = a_2 - a_1$$

$$\text{Solve for } (p, q) \text{ using ExtGCD.}$$

$$x \equiv m_1 p + a_1 \equiv m_2 q + a_2 \pmod{\text{lcm}(m_1, m_2)}$$

- Avoiding Overflow: $ca \bmod cb = c(a \bmod b)$

- Dirichlet Convolution: $(f * g)(n) = \sum_{d|n} f(d)g(n/d)$

- Important Multiplicative Functions + Properties:

- $\epsilon(n) = [n = 1]$
- $1(n) = 1$
- $id(n) = n$
- $\mu(n) = 0$ if n has squared prime factor
- $\mu(n) = (-1)^k$ if $n = p_1 p_2 \cdots p_k$
- $\epsilon = \mu * 1$
- $\phi = \mu * id$
- $[n = 1] = \sum_{d|n} \mu(d)$
- $[gcd = 1] = \sum_{d|gcd} \mu(d)$

- Möbius inversion: $f = g * 1 \Leftrightarrow g = f * \mu$

8.12 Polynomial

```
1 const int maxk = 20;
2 const int maxn = 1<<maxk;
3 const ll LINF = 1e18;
```

```
4 /* P = r*2^k + 1
```

```
5 P          r    k    g
6 998244353    119  23    3
7 1004535809    479  21    3
```

```
8
9 P          r    k    g
10 3          1    1    2
11 5          1    2    2
12 17         1    4    3
13 97         3    5    5
14 193        3    6    5
15 257        1    8    3
16 7681       15    9   17
17 12289      3   12   11
18 40961      5   13    3
19 65537      1   16    3
20 786433     3   18   10
21 5767169    11  19    3
22 7340033    7   20    3
23 23068673   11  21    3
24 104857601  25  22    3
25 167772161  5   25    3
26 469762049  7   26    3
27 1004535809 479  21    3
28 2013265921 15  27   31
29 2281701377 17  27    3
30 3221225473 3   30    5
31 75161927681 35  31    3
```

```
33 77309411329    9   33    7
34 206158430209    3   36   22
35 2061584302081   15   37    7
36 2748779069441    5   39    3
37 6597069766657    3   41    5
38 39582418599937    9   42    5
39 79164837199873    9   43    5
40 263882790666241   15   44    7
41 1231453023109121  35   45    3
42 1337006139375617  19   46    3
43 3799912185593857  27   47    5
44 4222124650659841  15   48   19
45 7881299347898369    7   50    6
46 31525197391593473  7   52    3
47 180143985094819841  5   55    6
48 194555039024054273 27   56    5
49 4179340454199820289 29   57    3
50 9097271247288401921 505  54    6 */
```

```
51
52 const int g = 3;
53 const ll MOD = 998244353;
54
55 ll pw(ll a, ll n) { /* fast pow */ }
56
57 #define siz(x) (int)x.size()
58
59 template<typename T>
60 vector<T>& operator+=(vector<T>& a, const vector<T>& b)
61 {
62     if (siz(a) < siz(b)) a.resize(siz(b));
63     for (int i = 0; i < min(siz(a), siz(b)); i++) {
64         a[i] += b[i];
65         a[i] -= a[i] >= MOD ? MOD : 0;
66     }
67     return a;
68 }
69
70 template<typename T>
71 vector<T>& operator--(vector<T>& a, const vector<T>& b)
72 {
73     if (siz(a) < siz(b)) a.resize(siz(b));
74     for (int i = 0; i < min(siz(a), siz(b)); i++) {
75         a[i] -= b[i];
76         a[i] += a[i] < 0 ? MOD : 0;
77     }
78     return a;
79 }
80
81 template<typename T>
82 vector<T> operator-(const vector<T>& a) {
83     vector<T> ret(siz(a));
84     for (int i = 0; i < siz(a); i++) {
85         ret[i] = -a[i] < 0 ? -a[i] + MOD : -a[i];
86     }
87     return ret;
88 }
89
90 vector<ll> X, iX;
91 vector<int> rev;
92
93 void init_ntt() {
94     X.clear(); X.resize(maxn, 1); // x1 = g^((p-1)/n)
95     iX.clear(); iX.resize(maxn, 1);
96
97     ll u = pw(g, (MOD-1)/maxn);
98     ll iu = pw(u, MOD-2);
99
100     for (int i = 1; i < maxn; i++) {
101         X[i] = X[i-1] * u;
102         iX[i] = iX[i-1] * iu;
103         if (X[i] >= MOD) X[i] %= MOD;
104         if (iX[i] >= MOD) iX[i] %= MOD;
105     }
106
107     rev.clear(); rev.resize(maxn, 0);
108     for (int i = 1, hb = -1; i < maxn; i++) {
109         if (!(i & (i-1))) hb++;
110         rev[i] = rev[i ^ (1<<hb)] | (1<<(maxk-hb-1));
111     }
112
113 template<typename T>
114 void NTT(vector<T>& a, bool inv=false) {
```

```

113 int _n = (int)a.size();
114 int k = __lg(_n) + ((1<<__lg(_n)) != _n);
115 int n = 1<<k;
116 a.resize(n, 0);

```

```

117 short shift = maxk-k;
118 for (int i = 0; i < n; i++)
119     if (i > (rev[i]>>shift))
120         swap(a[i], a[rev[i]>>shift]);

```

```

121 for (int len = 2, half = 1, div = maxn>>1; len <= n; len<=1, half<=1, div>>=1) {
122     for (int i = 0; i < n; i += len) {
123         for (int j = 0; j < half; j++) {
124             T u = a[i+j];
125             T v = a[i+j+half] * (inv ? ix[j*div] :
126                 X[j*div]) % MOD;
127             a[i+j] = (u+v >= MOD ? u+v-MOD : u+v);
128             a[i+j+half] = (u-v < 0 ? u-v+MOD : u-v);
129         }
130     }

```

```

131 } } }
132
133 if (inv) {
134     T dn = pw(n, MOD-2);
135     for (auto& x : a) {
136         x *= dn;
137         if (x >= MOD) x %= MOD;
138     }
}

```

```

139
140 template<typename T>
141 inline void resize(vector<T>& a) {
142     int cnt = (int)a.size();
143     for (; cnt > 0; cnt--) if (a[cnt-1]) break;
144     a.resize(max(cnt, 1));
145 }

```

```

146
147 template<typename T>
148 vector<T>& operator*=(vector<T>& a, vector<T> b) {
149     int na = (int)a.size();
150     int nb = (int)b.size();
151     a.resize(na + nb - 1, 0);
152     b.resize(na + nb - 1, 0);

```

```

153
154 NTT(a); NTT(b);
155 for (int i = 0; i < (int)a.size(); i++) {
156     a[i] *= b[i];
157     if (a[i] >= MOD) a[i] %= MOD;
158 }
159 NTT(a, true);

```

```

160
161 resize(a);
162 return a;

```

```

163 }
164
165 template<typename T>
166 void inv(vector<T>& ia, int N) {
167     vector<T> _a(move(ia));
168     ia.resize(1, pw(_a[0], MOD-2));
169     vector<T> a(1, -_a[0] + (-_a[0] < 0 ? MOD : 0));
170
171     for (int n = 1; n < N; n<=1) {
172         // n -> 2*n
173         // ia' = ia(2-a*ia);

```

```

174
175     for (int i = n; i < min(siz(_a), (n<<1)); i++)
176         a.emplace_back(-_a[i] + (-_a[i] < 0 ? MOD : 0));

```

```

177
178     vector<T> tmp = ia;
179     ia *= a;
180     ia.resize(n<<1);
181     ia[0] = ia[0] + 2 >= MOD ? ia[0] + 2 - MOD : ia[0] + 2;
182     ia *= tmp;
183     ia.resize(n<<1);
184 }
185 ia.resize(N);
186 }

```

```

187
188 template<typename T>
189 void mod(vector<T>& a, vector<T>& b) {

```

```

190 int n = (int)a.size()-1, m = (int)b.size()-1;
191 if (n < m) return;
192
193 vector<T> ra = a, rb = b;
194 reverse(ra.begin(), ra.end()); ra.resize(min(n+1, n-m+1));
195 reverse(rb.begin(), rb.end()); rb.resize(min(m+1, n-m+1));

```

```

196
197 inv(rb, n-m+1);

```

```

198
199 vector<T> q = move(ra);
200 q *= rb;
201 q.resize(n-m+1);
202 reverse(q.begin(), q.end());

```

```

203
204 q *= b;
205 a -= q;
206 resize(a);

```

```

207 }
208
209 /* Kitamasa Method (Fast Linear Recurrence):
210 Find a[K] (Given a[j] = c[0]a[j-N] + ... + c[N-1]a[j-1])
211 Let B(x) = x^N - c[N-1]x^(N-1) - ... - c[1]x^1 - c[0]
212 Let R(x) = x^K mod B(x) (get x^K using fast pow and use poly mod to get R(x))
213 Let r[i] = the coefficient of x^i in R(x)
214 => a[K] = a[0]r[0] + a[1]r[1] + ... + a[N-1]r[N-1] */

```

9 Linear Algebra

9.1 Gaussian-Jordan Elimination

```

1 int n;
2 vector<vector<ll>> v;
3 void gauss(vector<vector<ll>>& v) {
4     int r = 0;
5     for (int i = 0; i < n; i++) {
6         bool ok = false;
7         for (int j = r; j < n; j++) {
8             if (v[j][i] != 0) continue;
9             swap(v[j], v[r]);
10            ok = true;
11            break;
12        }
13        if (!ok) continue;
14        ll div = inv(v[r][i]);
15        for (int j = 0; j < n + 1; j++) {
16            v[r][j] *= div;
17            if (v[r][j] >= MOD) v[r][j] %= MOD;
18        }
19        for (int j = 0; j < n; j++) {
20            if (j == r) continue;
21            ll t = v[j][i];
22            for (int k = 0; k < n + 1; k++) {
23                v[j][k] -= v[r][k] * t % MOD;
24                if (v[j][k] < 0) v[j][k] += MOD;
25            }
26        }
27        r++;
28    }
29 }

```

9.2 Determinant

1. Use GJ Elimination, if there's any row consists of only 0, then $\det = 0$, otherwise $\det = \text{product of diagonal elements}$.

2. Properties of \det :

- Transpose: Unchanged
- Row Operation 1 - Swap 2 rows: $-\det$
- Row Operation 2 - $k\vec{r}_i$: $k \times \det$
- Row Operation 3 - $k\vec{r}_i$ add to \vec{r}_j : Unchanged

10 Combinatorics

10.1 Catalan Number

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

0	1	1	2	5
4	14	42	132	429
8	1430	4862	16796	58786
12	208012	742900	2674440	9694845

10.2 Burnside's Lemma

- Let X be the original set.
- Let G be the group of operations acting on X .
- Let X^g be the set of x not affected by g .
- Let X/G be the set of orbits.
- Then the following equation holds:

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

