

# Competitive Programming Template

ethening

The Chinese University of Hong Kong

September 7, 2023

# Contents

<b>Start</b>	<b>2</b>
Makefile . . . . .	2
Template . . . . .	2
<b>Data Structure</b>	<b>2</b>
Segment Tree . . . . .	2
Lazy Segment Tree . . . . .	4
BIT . . . . .	6
DSU . . . . .	7
KD Tree . . . . .	8
<b>Math</b>	<b>9</b>
Mint . . . . .	9
Combin . . . . .	11
Sieve . . . . .	13
Polynomial . . . . .	13
<b>Graph</b>	<b>18</b>
SCC . . . . .	18
Centroid Decomposition . . . . .	19
Heavy Light Decomposition . . . . .	20
Block Cut Tree . . . . .	22
DSU On Tree . . . . .	23
<b>Flow</b>	<b>24</b>
MaxFlow . . . . .	24
Min Cost Flow . . . . .	26
<b>String</b>	<b>27</b>
KMP . . . . .	27
Trie . . . . .	28
GSAM . . . . .	28
ACA . . . . .	29
Manacher . . . . .	30
SAM . . . . .	31
<b>Geometry</b>	<b>31</b>
Geometry (Jiangly) . . . . .	31
Minkowski Sum . . . . .	36

# Start

## Makefile

```
1  # ICPC
2  % : %.cpp
3      g++ -Wall -Wfatal-errors -Wshadow -g -std=c++2a $< -o $@ -O2
```

## Template

```
1  #include "bits/stdc++.h"
2  using namespace std;
3  using ll = long long;
4  using pii = pair<int, int>;
5  using pil = pair<int, ll>;
6  using pli = pair<ll, int>;
7  using pll = pair<ll, ll>;
8
9  inline void yes() { cout << "YES" << "\n"; return; }
10 inline void no() { cout << "NO" << "\n"; return; }
11
12 template<typename T>
13 bool chmin(T &x, T val) { if (val < x) { x = val; return true; } return false; }
14 template<typename T>
15 bool chmax(T &x, T val) { if (x < val) { x = val; return true; } return false; }
16
17 #define DEBUG 1
18 #define MULTI_TEST 1
19
20 void solve() {
21     int n;
22     cin >> n;
23     vector<int> a(n);
24     for (auto &o: a) {
25         cin >> o;
26     }
27 }
28
29 int main() {
30     cin.tie(0)->sync_with_stdio(0);
31
32     if (MULTI_TEST) {
33         int t;
34         cin >> t;
35         while (t--) {
36             solve();
37         }
38     }
39     else {
40         solve();
41     }
42 }
```

## Data Structure

### Segment Tree

```
1  template<class Info>
2  struct SegmentTree {
3      int n;
4      std::vector<Info> info;
5
6      SegmentTree() : n(0) {}
7      SegmentTree(int n_, Info v_ = Info()) {
8          init(n_, v_);
9      }
10     template<class T>
11     SegmentTree(std::vector<T> init_) {
12         init(init_);
```

```

13     }
14
15     void init(int n_, Info v_ = Info()) {
16         init(std::vector(n_, v_));
17     }
18     template<class T>
19     void init(std::vector<T> init_) {
20         n = init_.size();
21         info.assign(4 << std::lg(n), Info());
22         std::function<void(int, int, int)> build = [&](int p, int l, int r) {
23             if (r - l == 1) {
24                 info[p] = init_[l];
25                 return;
26             }
27             int m = (l + r) / 2;
28             build(2 * p, l, m);
29             build(2 * p + 1, m, r);
30             pull(p);
31         };
32         build(1, 0, n);
33     }
34     void pull(int p) {
35         info[p] = info[2 * p] + info[2 * p + 1];
36     }
37     void modify(int p, int l, int r, int x, const Info &v) {
38         if (r - l == 1) {
39             info[p] = v;
40             return;
41         }
42         int m = (l + r) / 2;
43         if (x < m) {
44             modify(2 * p, l, m, x, v);
45         } else {
46             modify(2 * p + 1, m, r, x, v);
47         }
48         pull(p);
49     }
50     void modify(int p, const Info &v) {
51         modify(1, 0, n, p, v);
52     }
53     Info rangeQuery(int p, int l, int r, int x, int y) {
54         if (l >= y || r <= x) {
55             return Info();
56         }
57         if (l >= x && r <= y) {
58             return info[p];
59         }
60         int m = (l + r) / 2;
61         return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
62     }
63     Info rangeQuery(int l, int r) {
64         return rangeQuery(1, 0, n, l, r);
65     }
66     template<class F>
67     int findFirst(int p, int l, int r, int x, int y, F pred) {
68         if (l >= y || r <= x || !pred(info[p])) {
69             return -1;
70         }
71         if (r - l == 1) {
72             return l;
73         }
74         int m = (l + r) / 2;
75         int res = findFirst(2 * p, l, m, x, y, pred);
76         if (res == -1) {
77             res = findFirst(2 * p + 1, m, r, x, y, pred);
78         }
79         return res;
80     }
81     template<class F>
82     int findFirst(int l, int r, F pred) {
83         return findFirst(1, 0, n, l, r, pred);

```

```

84     }
85     template<class F>
86     int findLast(int p, int l, int r, int x, int y, F pred) {
87         if (l >= y || r <= x || !pred(info[p])) {
88             return -1;
89         }
90         if (r - l == 1) {
91             return l;
92         }
93         int m = (l + r) / 2;
94         int res = findLast(2 * p + 1, m, r, x, y, pred);
95         if (res == -1) {
96             res = findLast(2 * p, l, m, x, y, pred);
97         }
98         return res;
99     }
100     template<class F>
101     int findLast(int l, int r, F pred) {
102         return findLast(1, 0, n, l, r, pred);
103     }
104 };
105
106 constexpr ll inf = 1E18;
107
108 struct Info {
109     ll cnt = 0;
110     ll sum = 0;
111     ll min = inf;
112 };
113
114 Info operator+(Info a, Info b) {
115     Info c;
116     c.cnt = a.cnt + b.cnt;
117     c.sum = a.sum + b.sum;
118     c.min = std::min(a.min, b.min);
119     return c;
120 }

```

## Lazy Segment Tree

```

1  template<class Info, class Tag>
2  struct LazySegmentTree {
3      int n;
4      std::vector<Info> info;
5      std::vector<Tag> tag;
6      LazySegmentTree() : n(0) {}
7      LazySegmentTree(int n_, Info v_ = Info()) {
8          init(n_, v_);
9      }
10     template<class T>
11     LazySegmentTree(std::vector<T> init_) {
12         init(init_);
13     }
14     void init(int n_, Info v_ = Info()) {
15         init(std::vector(n_, v_));
16     }
17     template<class T>
18     void init(std::vector<T> init_) {
19         n = init_.size();
20         info.assign(4 << std::lg(n), Info());
21         tag.assign(4 << std::lg(n), Tag());
22         std::function<void(int, int, int)> build = [&](int p, int l, int r) {
23             if (r - l == 1) {
24                 info[p] = init_[l];
25                 return;
26             }
27             int m = (l + r) / 2;
28             build(2 * p, l, m);
29             build(2 * p + 1, m, r);
30             pull(p);
31         };

```

```

32     build(1, 0, n);
33 }
34 void pull(int p) {
35     info[p] = info[2 * p] + info[2 * p + 1];
36 }
37 void apply(int p, const Tag &v) {
38     info[p].apply(v);
39     tag[p].apply(v);
40 }
41 void push(int p) {
42     apply(2 * p, tag[p]);
43     apply(2 * p + 1, tag[p]);
44     tag[p] = Tag();
45 }
46 void modify(int p, int l, int r, int x, const Info &v) {
47     if (r - l == 1) {
48         info[p] = v;
49         return;
50     }
51     int m = (l + r) / 2;
52     push(p);
53     if (x < m) {
54         modify(2 * p, l, m, x, v);
55     } else {
56         modify(2 * p + 1, m, r, x, v);
57     }
58     pull(p);
59 }
60 void modify(int p, const Info &v) {
61     modify(1, 0, n, p, v);
62 }
63 Info rangeQuery(int p, int l, int r, int x, int y) {
64     if (l >= y || r <= x) {
65         return Info();
66     }
67     if (l >= x && r <= y) {
68         return info[p];
69     }
70     int m = (l + r) / 2;
71     push(p);
72     return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
73 }
74 Info rangeQuery(int l, int r) {
75     return rangeQuery(1, 0, n, l, r);
76 }
77 void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
78     if (l >= y || r <= x) {
79         return;
80     }
81     if (l >= x && r <= y) {
82         apply(p, v);
83         return;
84     }
85     int m = (l + r) / 2;
86     push(p);
87     rangeApply(2 * p, l, m, x, y, v);
88     rangeApply(2 * p + 1, m, r, x, y, v);
89     pull(p);
90 }
91 void rangeApply(int l, int r, const Tag &v) {
92     return rangeApply(1, 0, n, l, r, v);
93 }
94 template<class F>
95 int findFirst(int p, int l, int r, int x, int y, F pred) {
96     if (l >= y || r <= x || !pred(info[p])) {
97         return -1;
98     }
99     if (r - l == 1) {
100         return l;
101     }
102     int m = (l + r) / 2;

```

```

103     push(p);
104     int res = findFirst(2 * p, l, m, x, y, pred);
105     if (res == -1) {
106         res = findFirst(2 * p + 1, m, r, x, y, pred);
107     }
108     return res;
109 }
110 template<class F>
111 int findFirst(int l, int r, F pred) {
112     return findFirst(1, 0, n, l, r, pred);
113 }
114 template<class F>
115 int findLast(int p, int l, int r, int x, int y, F pred) {
116     if (l >= y || r <= x || !pred(info[p])) {
117         return -1;
118     }
119     if (r - l == 1) {
120         return l;
121     }
122     int m = (l + r) / 2;
123     push(p);
124     int res = findLast(2 * p + 1, m, r, x, y, pred);
125     if (res == -1) {
126         res = findLast(2 * p, l, m, x, y, pred);
127     }
128     return res;
129 }
130 template<class F>
131 int findLast(int l, int r, F pred) {
132     return findLast(1, 0, n, l, r, pred);
133 }
134 };
135
136 struct Tag {
137     ll add = 0;
138
139     void apply(Tag t) {
140         add += t.add;
141     }
142 };
143
144 struct Info {
145     ll mn = 1E18;
146
147     void apply(Tag t) {
148         mn += t.add;
149     }
150 };
151
152 Info operator+(Info a, Info b) {
153     return {std::min(a.mn, b.mn)};
154 }

```

## BIT

```

1  template <typename T>
2  struct BIT {
3      int n;
4      vector<T> a;
5
6      BIT(int n = 0) {
7          init(n);
8      }
9
10     void init(int n) {
11         this->n = n;
12         a.assign(n, T());
13     }
14
15     void add(int x, int v) {
16         while (x < n) {

```

```

17         a[x] += v;
18         x += x & -x;
19     }
20 }
21
22 T sum(int x) {
23     auto ret = T();
24     while (x > 0) {
25         ret += a[x];
26         x -= x & -x;
27     }
28     return ret;
29 }
30
31 T rangeSum(int l, int r) {
32     if (l == 0) return sum(r);
33     else return sum(r) - sum(l - 1);
34 }
35
36 };

```

## DSU

```

1  struct DSU {
2      vector<int> f, siz, rnk;
3      int cc;
4
5      DSU() {}
6      DSU(int n) {
7          init(n);
8      }
9
10     void init(int n) {
11         f.resize(n);
12         iota(f.begin(), f.end(), 0);
13         siz.assign(n, 1);
14         rnk.assign(n, 0);
15         cc = n;
16     }
17
18     int find(int x) {
19         while (x != f[x]) {
20             x = f[x] = f[f[x]];
21         }
22         return x;
23     }
24
25     bool same(int x, int y) {
26         return find(x) == find(y);
27     }
28
29     int merge(int x, int y) {
30         x = find(x);
31         y = find(y);
32         if (x == y) return -1;
33         --cc;
34         if (rnk[x] > rnk[y]) swap(x, y);
35         siz[y] += siz[x];
36         f[x] = y;
37         if (rnk[x] == rnk[y]) rnk[y]++;
38         return y;
39     }
40
41     int size(int x) {
42         return siz[find(x)];
43     }
44 };

```



## KD Tree

```
1  template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
2  template<class T>
3  struct Point {
4      typedef Point P;
5      T x, y;
6      explicit Point(T x=0, T y=0) : x(x), y(y) {}
7      bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
8      bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
9      P operator+(P p) const { return P(x+p.x, y+p.y); }
10     P operator-(P p) const { return P(x-p.x, y-p.y); }
11     P operator*(T d) const { return P(x*d, y*d); }
12     P operator/(T d) const { return P(x/d, y/d); }
13     T dot(P p) const { return x*p.x + y*p.y; }
14     T cross(P p) const { return x*p.y - y*p.x; }
15     T cross(P a, P b) const { return (a-*this).cross(b-*this); }
16     T dist2() const { return x*x + y*y; }
17     double dist() const { return sqrt((double)dist2()); }
18     // angle to x-axis in interval [-pi, pi]
19     double angle() const { return atan2(y, x); }
20     P unit() const { return *this/dist(); } // makes dist()==1
21     P perp() const { return P(-y, x); } // rotates +90 degrees
22     P normal() const { return perp().unit(); }
23     // returns point rotated 'a' radians ccw around the origin
24     P rotate(double a) const {
25         return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
26     friend ostream& operator<<(ostream& os, P p) {
27         return os << "(" << p.x << ", " << p.y << ")"; }
28 };
29
30 using T = ll;
31 using P = Point<T>;
32 const T INF = numeric_limits<T>::max();
33
34 bool on_x(const P& a, const P& b) { return a.x < b.x; }
35 bool on_y(const P& a, const P& b) { return a.y < b.y; }
36
37 struct Node {
38     P pt; // if this is a leaf, the single point in it
39     T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
40     Node *first = 0, *second = 0;
41
42     T distance(const P& p) { // min squared distance to a point
43         T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
44         T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
45         return (P(x,y) - p).dist2();
46     }
47
48     Node(vector<P>&& vp) : pt(vp[0]) {
49         for (P p : vp) {
50             x0 = min(x0, p.x); x1 = max(x1, p.x);
51             y0 = min(y0, p.y); y1 = max(y1, p.y);
52         }
53         if (vp.size() > 1) {
54             // split on x if width >= height (not ideal...)
55             sort(begin(vp), end(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
56             // divide by taking half the array for each child (not
57             // best performance with many duplicates in the middle)
58             int half = (int)size(vp)/2;
59             first = new Node({vp.begin(), vp.begin() + half});
60             second = new Node({vp.begin() + half, vp.end()});
61         }
62     }
63 };
64
65 struct KDTree {
66     Node* root;
67     KDTree(const vector<P>& vp) : root(new Node({begin(vp), end(vp)})) {}
68
69     pair<T, P> search(Node *node, const P& p) {
```

```

70     if (!node->first) {
71         // uncomment if we should not find the point itself:
72         // if (p == node->pt) return {INF, P()};
73         return make_pair((p - node->pt).dist2(), node->pt);
74     }
75
76     Node *f = node->first, *s = node->second;
77     T bfirst = f->distance(p), bsec = s->distance(p);
78     if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);
79
80     // search closest side first, other side if needed
81     auto best = search(f, p);
82     if (bsec < best.first)
83         best = min(best, search(s, p));
84     return best;
85 }
86
87 // find nearest point to a point, and its squared distance
88 // (requires an arbitrary operator< for Point)
89 pair<T, P> nearest(const P& p) {
90     return search(root, p);
91 }
92 };

```

## Math

### Mint

```

1  template<class T>
2  constexpr T power(T a, ll b) {
3      T res = 1;
4      for (; b; b /= 2, a *= a) {
5          if (b % 2) res *= a;
6      }
7      return res;
8  }
9
10 constexpr ll mul(ll a, ll b, ll p) {
11     ll res = a * b - ll(1.L * a * b / p) * p;
12     res %= p;
13     if (res < 0) res += p;
14     return res;
15 }
16
17 template<ll P>
18 struct MLong {
19     ll x;
20     constexpr MLong() : x{} {}
21     constexpr MLong(ll x) : x{norm(x % getMod())} {}
22
23     static ll Mod;
24     constexpr static ll getMod() {
25         if (P > 0) {
26             return P;
27         } else {
28             return Mod;
29         }
30     }
31     constexpr static void setMod(ll Mod_) {
32         Mod = Mod_;
33     }
34     constexpr ll norm(ll x) const {
35         if (x < 0) {
36             x += getMod();
37         }
38         if (x >= getMod()) {
39             x -= getMod();
40         }
41         return x;
42     }

```

```

43
44 constexpr ll val() const { return x; }
45 explicit constexpr operator ll() const { return x; }
46
47 constexpr MLong operator-() const { MLong res; res.x = norm(getMod() - x); return res; }
48 constexpr MLong inv() const { assert(x != 0); return power(*this, getMod() - 2); }
49 constexpr MLong &operator*=(MLong rhs) & { x = mul(x, rhs.x, getMod()); return *this; }
50 constexpr MLong &operator+=(MLong rhs) & { x = norm(x + rhs.x); return *this; }
51 constexpr MLong &operator-=(MLong rhs) & { x = norm(x - rhs.x); return *this; }
52 constexpr MLong &operator/=(MLong rhs) & { return *this *= rhs.inv(); }
53
54 friend constexpr MLong operator*(MLong lhs, MLong rhs) { MLong res = lhs; res *= rhs; return res; }
55 friend constexpr MLong operator+(MLong lhs, MLong rhs) { MLong res = lhs; res += rhs; return res; }
56 friend constexpr MLong operator-(MLong lhs, MLong rhs) { MLong res = lhs; res -= rhs; return res; }
57 friend constexpr MLong operator/(MLong lhs, MLong rhs) { MLong res = lhs; res /= rhs; return res; }
58
59 friend constexpr std::istream &operator>>(std::istream &is, MLong &a) { ll v{}; is >> v; a = MLong(v); return is;
↪ }
60 friend constexpr std::ostream &operator<<(std::ostream &os, const MLong &a) { return os << a.val(); }
61 friend constexpr bool operator==(MLong lhs, MLong rhs) { return lhs.val() == rhs.val(); }
62 friend constexpr bool operator!=(MLong lhs, MLong rhs) { return lhs.val() != rhs.val(); }
63 };
64
65 template<>
66 ll MLong<@LL>::Mod = 1;
67
68 template<int P>
69 struct MInt {
70     int x;
71     constexpr MInt() : x{} {}
72     constexpr MInt(ll x) : x{norm(x % getMod())} {}
73
74     static int Mod;
75     constexpr static int getMod() {
76         if (P > 0) {
77             return P;
78         } else {
79             return Mod;
80         }
81     }
82     constexpr static void setMod(int Mod_) {
83         Mod = Mod_;
84     }
85     constexpr int norm(int x) const {
86         if (x < 0) {
87             x += getMod();
88         }
89         if (x >= getMod()) {
90             x -= getMod();
91         }
92         return x;
93     }
94
95     constexpr int val() const { return x; }
96     explicit constexpr operator int() const { return x; }
97
98     constexpr MInt operator-() const { MInt res; res.x = norm(getMod() - x); return res; }
99     constexpr MInt inv() const { assert(x != 0); return power(*this, getMod() - 2); }
100    constexpr MInt &operator*=(MInt rhs) & { x = 1LL * x * rhs.x % getMod(); return *this; }
101    constexpr MInt &operator+=(MInt rhs) & { x = norm(x + rhs.x); return *this; }
102    constexpr MInt &operator-=(MInt rhs) & { x = norm(x - rhs.x); return *this; }
103    constexpr MInt &operator/=(MInt rhs) & { return *this *= rhs.inv(); }
104
105    friend constexpr MInt operator*(MInt lhs, MInt rhs) { MInt res = lhs; res *= rhs; return res; }
106    friend constexpr MInt operator+(MInt lhs, MInt rhs) { MInt res = lhs; res += rhs; return res; }
107    friend constexpr MInt operator-(MInt lhs, MInt rhs) { MInt res = lhs; res -= rhs; return res; }
108    friend constexpr MInt operator/(MInt lhs, MInt rhs) { MInt res = lhs; res /= rhs; return res; }
109
110    friend constexpr std::istream &operator>>(std::istream &is, MInt &a) { ll v{}; is >> v; a = MInt(v); return is; }
111    friend constexpr std::ostream &operator<<(std::ostream &os, const MInt &a) { return os << a.val(); }
112    friend constexpr bool operator==(MInt lhs, MInt rhs) { return lhs.val() == rhs.val(); }

```

```

113     friend constexpr bool operator!=(MInt lhs, MInt rhs) { return lhs.val() != rhs.val(); }
114 };
115
116 template<>
117 int MInt<0>::Mod = 1;
118
119 template<int V, int P>
120 constexpr MInt<P> CInv = MInt<P>(V).inv();
121
122 constexpr int P = 1000000007;
123 using Z = MInt<P>;

```

## Combin

```

1  template<class T>
2  constexpr T power(T a, ll b) {
3      T res = 1;
4      for (; b; b /= 2, a *= a) {
5          if (b % 2) res *= a;
6      }
7      return res;
8  }
9
10 constexpr ll mul(ll a, ll b, ll p) {
11     ll res = a * b - ll(1.L * a * b / p) * p;
12     res %= p;
13     if (res < 0) res += p;
14     return res;
15 }
16
17 template<int P>
18 struct MInt {
19     int x;
20     constexpr MInt() : x{} {}
21     constexpr MInt(ll x) : x{norm(x % getMod())} {}
22
23     static int Mod;
24     constexpr static int getMod() {
25         if (P > 0) {
26             return P;
27         } else {
28             return Mod;
29         }
30     }
31     constexpr static void setMod(int Mod_) {
32         Mod = Mod_;
33     }
34     constexpr int norm(int x) const {
35         if (x < 0) {
36             x += getMod();
37         }
38         if (x >= getMod()) {
39             x -= getMod();
40         }
41         return x;
42     }
43
44     constexpr int val() const { return x; }
45     explicit constexpr operator int() const { return x; }
46
47     constexpr MInt operator-() const { MInt res; res.x = norm(getMod() - x); return res; }
48     constexpr MInt inv() const { assert(x != 0); return power(*this, getMod() - 2); }
49     constexpr MInt &operator*=(MInt rhs) & { x = 1LL * x * rhs.x % getMod(); return *this; }
50     constexpr MInt &operator+=(MInt rhs) & { x = norm(x + rhs.x); return *this; }
51     constexpr MInt &operator-=(MInt rhs) & { x = norm(x - rhs.x); return *this; }
52     constexpr MInt &operator/=(MInt rhs) & { return *this *= rhs.inv(); }
53
54     friend constexpr MInt operator*(MInt lhs, MInt rhs) { MInt res = lhs; res *= rhs; return res; }
55     friend constexpr MInt operator+(MInt lhs, MInt rhs) { MInt res = lhs; res += rhs; return res; }
56     friend constexpr MInt operator-(MInt lhs, MInt rhs) { MInt res = lhs; res -= rhs; return res; }
57     friend constexpr MInt operator/(MInt lhs, MInt rhs) { MInt res = lhs; res /= rhs; return res; }

```

```

58
59     friend constexpr std::istream &operator>>(std::istream &is, MInt &a) { ll v{}; is >> v; a = MInt(v); return is; }
60     friend constexpr std::ostream &operator<<(std::ostream &os, const MInt &a) { return os << a.val(); }
61     friend constexpr bool operator==(MInt lhs, MInt rhs) { return lhs.val() == rhs.val(); }
62     friend constexpr bool operator!=(MInt lhs, MInt rhs) { return lhs.val() != rhs.val(); }
63 };
64
65 template<>
66 int MInt<0>::Mod = 1;
67
68 template<int V, int P>
69 constexpr MInt<P> CInv = MInt<P>(V).inv();
70
71 constexpr int P = 1000000007;
72 using Z = MInt<P>;
73
74 struct Comb {
75     int n;
76     std::vector<Z> _fac;
77     std::vector<Z> _invfac;
78     std::vector<Z> _inv;
79
80     Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
81     Comb(int n) : Comb() {
82         init(n);
83     }
84
85     void init(int m) {
86         if (m <= n) return;
87         _fac.resize(m + 1);
88         _invfac.resize(m + 1);
89         _inv.resize(m + 1);
90
91         for (int i = n + 1; i <= m; i++) {
92             _fac[i] = _fac[i - 1] * i;
93         }
94         _invfac[m] = _fac[m].inv();
95         for (int i = m; i > n; i--) {
96             _invfac[i - 1] = _invfac[i] * i;
97             _inv[i] = _invfac[i] * _fac[i - 1];
98         }
99         n = m;
100     }
101
102     Z fac(int m) {
103         if (m > n) init(2 * m);
104         return _fac[m];
105     }
106     Z invfac(int m) {
107         if (m > n) init(2 * m);
108         return _invfac[m];
109     }
110     Z inv(int m) {
111         if (m > n) init(2 * m);
112         return _inv[m];
113     }
114     Z binom(int n, int m) {
115         if (n < m || m < 0) return 0;
116         return fac(n) * invfac(m) * invfac(n - m);
117     }
118     Z catalan(int n) {
119         return binom(2 * n, n) * inv(n + 1);
120     }
121     Z lucas(int n, int m) {
122         if (n % P < m % P) return 0;
123         if (n < P) return binom(n, m);
124         return 1ll * lucas(n / P, m / P) * binom(n % P, m % P);
125     }
126 } comb;

```

## Sieve

```
1  std::vector<int> minp, primes;
2
3  void sieve(int n) {
4      minp.assign(n + 1, 0);
5      primes.clear();
6
7      for (int i = 2; i <= n; i++) {
8          if (minp[i] == 0) {
9              minp[i] = i;
10             primes.push_back(i);
11         }
12
13         for (auto p : primes) {
14             if (i * p > n) {
15                 break;
16             }
17             minp[i * p] = p;
18             if (p == minp[i]) {
19                 break;
20             }
21         }
22     }
23 }
```

## Polynomial

```
1  std::vector<int> rev;
2  template<int P>
3  std::vector<MInt<P>> roots{0, 1};
4
5  template<int P>
6  constexpr MInt<P> findPrimitiveRoot() {
7      MInt<P> i = 2;
8      int k = __builtin_ctz(P - 1);
9      while (true) {
10         if (power(i, (P - 1) / 2) != 1) {
11             break;
12         }
13         i += 1;
14     }
15     return power(i, (P - 1) >> k);
16 }
17
18 template<int P>
19 constexpr MInt<P> primitiveRoot = findPrimitiveRoot<P>();
20
21 template<>
22 constexpr MInt<998244353> primitiveRoot<998244353> {31};
23
24 template<int P>
25 constexpr void dft(std::vector<MInt<P>> &a) {
26     int n = a.size();
27
28     if (int(rev.size()) != n) {
29         int k = __builtin_ctz(n) - 1;
30         rev.resize(n);
31         for (int i = 0; i < n; i++) {
32             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
33         }
34     }
35
36     for (int i = 0; i < n; i++) {
37         if (rev[i] < i) {
38             std::swap(a[i], a[rev[i]]);
39         }
40     }
41     if (roots<P>.size() < n) {
42         int k = __builtin_ctz(roots<P>.size());
43         roots<P>.resize(n);
```

```

44     while ((1 << k) < n) {
45         auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k - 1));
46         for (int i = 1 << (k - 1); i < (1 << k); i++) {
47             roots<P>[2 * i] = roots<P>[i];
48             roots<P>[2 * i + 1] = roots<P>[i] * e;
49         }
50         k++;
51     }
52 }
53 for (int k = 1; k < n; k *= 2) {
54     for (int i = 0; i < n; i += 2 * k) {
55         for (int j = 0; j < k; j++) {
56             MInt<P> u = a[i + j];
57             MInt<P> v = a[i + j + k] * roots<P>[k + j];
58             a[i + j] = u + v;
59             a[i + j + k] = u - v;
60         }
61     }
62 }
63 }
64
65 template<int P>
66 constexpr void idft(std::vector<MInt<P>> &a) {
67     int n = a.size();
68     std::reverse(a.begin() + 1, a.end());
69     dft(a);
70     MInt<P> inv = (1 - P) / n;
71     for (int i = 0; i < n; i++) {
72         a[i] *= inv;
73     }
74 }
75
76 template<int P = 998244353>
77 struct Poly : public std::vector<MInt<P>> {
78     using Value = MInt<P>;
79
80     Poly() : std::vector<Value>() {}
81     explicit constexpr Poly(int n) : std::vector<Value>(n) {}
82
83     explicit constexpr Poly(const std::vector<Value> &a) : std::vector<Value>(a) {}
84     constexpr Poly(const std::initializer_list<Value> &a) : std::vector<Value>(a) {}
85
86     template<class InputIt, class = std::_RequireInputIter<InputIt>>
87     explicit constexpr Poly(InputIt first, InputIt last) : std::vector<Value>(first, last) {}
88
89     template<class F>
90     explicit constexpr Poly(int n, F f) : std::vector<Value>(n) {
91         for (int i = 0; i < n; i++) {
92             (*this)[i] = f(i);
93         }
94     }
95
96     constexpr Poly shift(int k) const {
97         if (k >= 0) {
98             auto b = *this;
99             b.insert(b.begin(), k, 0);
100             return b;
101         } else if (this->size() <= -k) {
102             return Poly();
103         } else {
104             return Poly(this->begin() + (-k), this->end());
105         }
106     }
107
108     constexpr Poly trunc(int k) const {
109         Poly f = *this;
110         f.resize(k);
111         return f;
112     }
113
114     constexpr friend Poly operator+(const Poly &a, const Poly &b) {
115         Poly res(std::max(a.size(), b.size()));
116         for (int i = 0; i < a.size(); i++) {

```

```

115         res[i] += a[i];
116     }
117     for (int i = 0; i < b.size(); i++) {
118         res[i] += b[i];
119     }
120     return res;
121 }
122 constexpr friend Poly operator-(const Poly &a, const Poly &b) {
123     Poly res(std::max(a.size(), b.size()));
124     for (int i = 0; i < a.size(); i++) {
125         res[i] += a[i];
126     }
127     for (int i = 0; i < b.size(); i++) {
128         res[i] -= b[i];
129     }
130     return res;
131 }
132 constexpr friend Poly operator-(const Poly &a) {
133     std::vector<Value> res(a.size());
134     for (int i = 0; i < int(res.size()); i++) {
135         res[i] = -a[i];
136     }
137     return Poly(res);
138 }
139 constexpr friend Poly operator*(Poly a, Poly b) {
140     if (a.size() == 0 || b.size() == 0) {
141         return Poly();
142     }
143     if (a.size() < b.size()) {
144         std::swap(a, b);
145     }
146     int n = 1, tot = a.size() + b.size() - 1;
147     while (n < tot) {
148         n *= 2;
149     }
150     if (((P - 1) & (n - 1)) != 0 || b.size() < 128) {
151         Poly c(a.size() + b.size() - 1);
152         for (int i = 0; i < a.size(); i++) {
153             for (int j = 0; j < b.size(); j++) {
154                 c[i + j] += a[i] * b[j];
155             }
156         }
157         return c;
158     }
159     a.resize(n);
160     b.resize(n);
161     dft(a);
162     dft(b);
163     for (int i = 0; i < n; ++i) {
164         a[i] *= b[i];
165     }
166     idft(a);
167     a.resize(tot);
168     return a;
169 }
170 constexpr friend Poly operator*(Value a, Poly b) {
171     for (int i = 0; i < int(b.size()); i++) {
172         b[i] *= a;
173     }
174     return b;
175 }
176 constexpr friend Poly operator*(Poly a, Value b) {
177     for (int i = 0; i < int(a.size()); i++) {
178         a[i] *= b;
179     }
180     return a;
181 }
182 constexpr friend Poly operator/(Poly a, Value b) {
183     for (int i = 0; i < int(a.size()); i++) {
184         a[i] /= b;
185     }

```



```

186     return a;
187 }
188 constexpr Poly &operator+=(Poly b) {
189     return (*this) = (*this) + b;
190 }
191 constexpr Poly &operator-=(Poly b) {
192     return (*this) = (*this) - b;
193 }
194 constexpr Poly &operator*=(Poly b) {
195     return (*this) = (*this) * b;
196 }
197 constexpr Poly &operator*=(Value b) {
198     return (*this) = (*this) * b;
199 }
200 constexpr Poly &operator/=(Value b) {
201     return (*this) = (*this) / b;
202 }
203 constexpr Poly deriv() const {
204     if (this->empty()) {
205         return Poly();
206     }
207     Poly res(this->size() - 1);
208     for (int i = 0; i < this->size() - 1; ++i) {
209         res[i] = (i + 1) * (*this)[i + 1];
210     }
211     return res;
212 }
213 constexpr Poly integr() const {
214     Poly res(this->size() + 1);
215     for (int i = 0; i < this->size(); ++i) {
216         res[i + 1] = (*this)[i] / (i + 1);
217     }
218     return res;
219 }
220 constexpr Poly inv(int m) const {
221     Poly x((*this)[0].inv());
222     int k = 1;
223     while (k < m) {
224         k *= 2;
225         x = (x * (Poly{2} - trunc(k) * x)).trunc(k);
226     }
227     return x.trunc(m);
228 }
229 constexpr Poly log(int m) const {
230     return (deriv() * inv(m)).integr().trunc(m);
231 }
232 constexpr Poly exp(int m) const {
233     Poly x{1};
234     int k = 1;
235     while (k < m) {
236         k *= 2;
237         x = (x * (Poly{1} - x.log(k) + trunc(k))).trunc(k);
238     }
239     return x.trunc(m);
240 }
241 constexpr Poly pow(int k, int m) const {
242     int i = 0;
243     while (i < this->size() && (*this)[i] == 0) {
244         i++;
245     }
246     if (i == this->size() || 1LL * i * k >= m) {
247         return Poly(m);
248     }
249     Value v = (*this)[i];
250     auto f = shift(-i) * v.inv();
251     return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v, k);
252 }
253 constexpr Poly sqrt(int m) const {
254     Poly x{1};
255     int k = 1;
256     while (k < m) {

```

```

257         k *= 2;
258         x = (x + (trunc(k) * x.inv(k)).trunc(k)) * CInv<2, P>;
259     }
260     return x.trunc(m);
261 }
262 constexpr Poly mult(Poly b) const {
263     if (b.size() == 0) {
264         return Poly();
265     }
266     int n = b.size();
267     std::reverse(b.begin(), b.end());
268     return ((*this) * b).shift(-(n - 1));
269 }
270 std::vector<Value> eval(std::vector<Value> x) const {
271     if (this->size() == 0) {
272         return std::vector<Value>(x.size(), 0);
273     }
274     const int n = std::max(x.size(), this->size());
275     std::vector<Poly> q(4 * n);
276     std::vector<Value> ans(x.size());
277     x.resize(n);
278     std::function<void(int, int, int)> build = [&](int p, int l, int r) {
279         if (r - l == 1) {
280             q[p] = Poly{1, -x[l]};
281         } else {
282             int m = (l + r) / 2;
283             build(2 * p, l, m);
284             build(2 * p + 1, m, r);
285             q[p] = q[2 * p] * q[2 * p + 1];
286         }
287     };
288     build(1, 0, n);
289     std::function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r, const Poly &num) {
290         if (r - l == 1) {
291             if (l < int(ans.size())) {
292                 ans[l] = num[0];
293             }
294         } else {
295             int m = (l + r) / 2;
296             work(2 * p, l, m, num.mult(q[2 * p + 1]).resize(m - l));
297             work(2 * p + 1, m, r, num.mult(q[2 * p]).resize(r - m));
298         }
299     };
300     work(1, 0, n, mult(q[1].inv(n)));
301     return ans;
302 }
303 };
304
305 template<int P = 998244353>
306 Poly<P> berlekampMassey(const Poly<P> &s) {
307     Poly<P> c;
308     Poly<P> oldC;
309     int f = -1;
310     for (int i = 0; i < s.size(); i++) {
311         auto delta = s[i];
312         for (int j = 1; j <= c.size(); j++) {
313             delta -= c[j - 1] * s[i - j];
314         }
315         if (delta == 0) {
316             continue;
317         }
318         if (f == -1) {
319             c.resize(i + 1);
320             f = i;
321         } else {
322             auto d = oldC;
323             d *= -1;
324             d.insert(d.begin(), 1);
325             MInt<P> df1 = 0;
326             for (int j = 1; j <= d.size(); j++) {
327                 df1 += d[j - 1] * s[f + 1 - j];

```

```

328         }
329         assert(df1 != 0);
330         auto coef = delta / df1;
331         d *= coef;
332         Poly<P> zeros(i - f - 1);
333         zeros.insert(zeros.end(), d.begin(), d.end());
334         d = zeros;
335         auto temp = c;
336         c += d;
337         if (i - temp.size() > f - oldC.size()) {
338             oldC = temp;
339             f = i;
340         }
341     }
342 }
343 c *= -1;
344 c.insert(c.begin(), 1);
345 return c;
346 }
347
348
349 template<int P = 998244353>
350 MInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
351     int m = q.size() - 1;
352     while (n > 0) {
353         auto newq = q;
354         for (int i = 1; i <= m; i += 2) {
355             newq[i] *= -1;
356         }
357         auto newp = p * newq;
358         newq = q * newq;
359         for (int i = 0; i < m; i++) {
360             p[i] = newp[i * 2 + n % 2];
361         }
362         for (int i = 0; i <= m; i++) {
363             q[i] = newq[i * 2];
364         }
365         n /= 2;
366     }
367     return p[0] / q[0];
368 }

```

## Graph

### SCC

```

1 struct SCC {
2     int n;
3     std::vector<std::vector<int>>> adj;
4     std::vector<int> stk;
5     std::vector<int> dfn, low, bel;
6     int cur, cnt;
7
8     SCC() {}
9     SCC(int n) {
10         init(n);
11     }
12
13     void init(int n) {
14         this->n = n;
15         adj.assign(n, {});
16         dfn.assign(n, -1);
17         low.resize(n);
18         bel.assign(n, -1);
19         stk.clear();
20         cur = cnt = 0;
21     }
22
23     void addEdge(int u, int v) {
24         adj[u].push_back(v);

```

```

25     }
26
27     void dfs(int x) {
28         dfn[x] = low[x] = cur++;
29         stk.push_back(x);
30
31         for (auto y : adj[x]) {
32             if (dfn[y] == -1) {
33                 dfs(y);
34                 low[x] = std::min(low[x], low[y]);
35             } else if (bel[y] == -1) {
36                 low[x] = std::min(low[x], dfn[y]);
37             }
38         }
39
40         if (dfn[x] == low[x]) {
41             int y;
42             do {
43                 y = stk.back();
44                 bel[y] = cnt;
45                 stk.pop_back();
46             } while (y != x);
47             cnt++;
48         }
49     }
50
51     std::vector<int> work() {
52         for (int i = 0; i < n; i++) {
53             if (dfn[i] == -1) {
54                 dfs(i);
55             }
56         }
57         return bel;
58     }
59 };

```

## Centroid Decomposition

```

1  void solve() {
2      int n, T;
3      cin >> n >> T;
4
5      vector<vector<array<int, 2>>> g(n);
6
7      for (int i = 1; i < n; i++) {
8          int u, v, w;
9          cin >> u >> v >> w;
10         u--, v--;
11         g[u].push_back({v, w});
12         g[v].push_back({u, w});
13     }
14
15     int q;
16     cin >> q;
17
18     vector<vector<array<int, 3>>> qry(n);
19     for (int i = 0; i < q; i++) {
20         int a, b;
21         cin >> a >> b;
22         a--, b--;
23         qry[0].push_back({a, b, i});
24     }
25     vector<ll> ans(q, inf);
26
27     vector<bool> vis(n);
28     vector<int> siz(n), bel(n);
29
30     auto dfs = [&](auto self, int x, int p) -> void {
31         siz[x] = 1;
32         for (auto [y, w] : g[x]) {
33             if (y == p) {

```

```

34         continue;
35     }
36     self(self, y, x);
37     siz[x] += siz[y];
38 }
39 };
40 dfs(dfs, 0, -1);
41
42 vector<ll> dep(n);
43 auto solve = [&](auto &&self, int r) -> void {
44     auto Q = std::move(qry[r]);
45
46     auto find = [&](auto self, int x, int p, int s) -> int {
47         for (auto [y, _] : g[x]) {
48             if (y == p || vis[y] || 2 * siz[y] <= s) {
49                 continue;
50             }
51             return self(self, y, x, s);
52         }
53         return x;
54     };
55     r = find(find, r, -1, siz[r]);
56     vis[r] = true;
57
58     auto dfs = [&](auto self, int x, int p) -> void {
59         siz[x] = 1;
60         for (auto [y, w] : g[x]) {
61             if (y == p || vis[y]) {
62                 continue;
63             }
64             dep[y] = dep[x] + w;
65             bel[y] = x == r ? y : bel[x];
66             self(self, y, x);
67             siz[x] += siz[y];
68         }
69     };
70     dfs(dfs, r, -1);
71
72     for (auto [a, b, i] : Q) {
73         ans[i] = min(ans[i], dep[a] + dep[b]);
74         if (bel[a] == bel[b]) {
75             qry[bel[a]].push_back({a, b, i});
76         }
77     }
78     for (auto [y, _] : g[r]) {
79         if (!vis[y]) {
80             self(self, y);
81         }
82     }
83 };
84 solve(solve, 0);
85
86 for (int i = 0; i < q; i++) {
87     cout << ans[i] << "\n";
88 }
89 }

```

## Heavy Light Decomposition

```

1 struct HLD {
2     int n;
3     std::vector<int> siz, top, dep, parent, in, out, seq;
4     std::vector<std::vector<int>>> adj;
5     int cur;
6
7     HLD() {}
8     HLD(int n) {
9         init(n);
10    }
11    void init(int n) {
12        this->n = n;

```

```

13     siz.resize(n);
14     top.resize(n);
15     dep.resize(n);
16     parent.resize(n);
17     in.resize(n);
18     out.resize(n);
19     seq.resize(n);
20     cur = 0;
21     adj.assign(n, {});
22 }
23 void addEdge(int u, int v) {
24     adj[u].push_back(v);
25     adj[v].push_back(u);
26 }
27 void work(int root = 0) {
28     top[root] = root;
29     dep[root] = 0;
30     parent[root] = -1;
31     dfs1(root);
32     dfs2(root);
33 }
34 void dfs1(int u) {
35     if (parent[u] != -1) {
36         adj[u].erase(std::find(adj[u].begin(), adj[u].end(), parent[u]));
37     }
38
39     siz[u] = 1;
40     for (auto &v : adj[u]) {
41         parent[v] = u;
42         dep[v] = dep[u] + 1;
43         dfs1(v);
44         siz[u] += siz[v];
45         if (siz[v] > siz[adj[u][0]]) {
46             std::swap(v, adj[u][0]);
47         }
48     }
49 }
50 void dfs2(int u) {
51     in[u] = cur++;
52     seq[in[u]] = u;
53     for (auto v : adj[u]) {
54         top[v] = v == adj[u][0] ? top[u] : v;
55         dfs2(v);
56     }
57     out[u] = cur;
58 }
59 int lca(int u, int v) {
60     while (top[u] != top[v]) {
61         if (dep[top[u]] > dep[top[v]]) {
62             u = parent[top[u]];
63         } else {
64             v = parent[top[v]];
65         }
66     }
67     return dep[u] < dep[v] ? u : v;
68 }
69
70 int dist(int u, int v) {
71     return dep[u] + dep[v] - 2 * dep[lca(u, v)];
72 }
73
74 int jump(int u, int k) {
75     if (dep[u] < k) {
76         return -1;
77     }
78
79     int d = dep[u] - k;
80
81     while (dep[top[u]] > d) {
82         u = parent[top[u]];
83     }

```

```

84
85     return seq[in[u] - dep[u] + d];
86 }
87
88 bool isAncestor(int u, int v) {
89     return in[u] <= in[v] && in[v] < out[u];
90 }
91
92 int rootedParent(int u, int v) {
93     std::swap(u, v);
94     if (u == v) {
95         return u;
96     }
97     if (!isAncestor(u, v)) {
98         return parent[u];
99     }
100     auto it = std::upper_bound(adj[u].begin(), adj[u].end(), v, [&](int x, int y) {
101         return in[x] < in[y];
102     }) - 1;
103     return *it;
104 }
105
106 int rootedSize(int u, int v) {
107     if (u == v) {
108         return n;
109     }
110     if (!isAncestor(v, u)) {
111         return siz[v];
112     }
113     return n - siz[rootedParent(u, v)];
114 }
115
116 int rootedLca(int a, int b, int c) {
117     return lca(a, b) ^ lca(b, c) ^ lca(c, a);
118 }
119 };

```

## Block Cut Tree

```

1  struct BlockCutTree {
2      int n;
3      std::vector<std::vector<int>> adj;
4      std::vector<int> dfn, low, stk;
5      int cnt, cur;
6      std::vector<std::pair<int, int>> edges;
7
8      BlockCutTree() {}
9      BlockCutTree(int n) {
10         init(n);
11     }
12
13     void init(int n) {
14         this->n = n;
15         adj.assign(n, {});
16         dfn.assign(n, -1);
17         low.resize(n);
18         stk.clear();
19         cnt = cur = 0;
20         edges.clear();
21     }
22
23     void addEdge(int u, int v) {
24         adj[u].push_back(v);
25         adj[v].push_back(u);
26     }
27
28     void dfs(int x) {
29         stk.push_back(x);
30         dfn[x] = low[x] = cur++;
31
32         for (auto y : adj[x]) {

```

```

33         if (dfn[y] == -1) {
34             dfs(y);
35             low[x] = std::min(low[x], low[y]);
36             if (low[y] == dfn[x]) {
37                 int v;
38                 do {
39                     v = stk.back();
40                     stk.pop_back();
41                     edges.emplace_back(n + cnt, v);
42                 } while (v != y);
43                 edges.emplace_back(x, n + cnt);
44                 cnt++;
45             }
46         } else {
47             low[x] = std::min(low[x], dfn[y]);
48         }
49     }
50 }
51
52 std::pair<int, std::vector<std::pair<int, int>>> work() {
53     for (int i = 0; i < n; i++) {
54         if (dfn[i] == -1) {
55             stk.clear();
56             dfs(i);
57         }
58     }
59     return {cnt, edges};
60 }
61 };

```

## DSU On Tree

```

1  struct FreqBuckets {
2      vector<int> occ;
3      vector<int> freq;
4      FreqBuckets(int n, int maxC) : occ(maxC + 1, 0), freq(n + 1) { }
5
6      void add(int x, int mul) {
7          if (mul == +1) {
8              occ[x]++;
9              freq[occ[x]]++;
10         }
11         else if (mul == -1) {
12             freq[occ[x]]--;
13             occ[x]--;
14         }
15         else assert(false);
16     }
17 };
18
19 int main() {
20     ios::sync_with_stdio(false);
21     cin.tie(0);
22
23     int n,m; cin >> n >> m;
24
25     vector<int> c(n);
26     for (int i = 0; i < n; i++)
27         cin >> c[i];
28
29     vector<vector<int>> g(n);
30     for (int i = 0; i + 1 < n; i++) {
31         int u,v; cin >> u >> v; u--; v--;
32         g[u].emplace_back(v);
33         g[v].emplace_back(u);
34     }
35
36     vector<int> sz(n, 1);
37     function<void(int, int)> dfs_hld = [&](int u, int p) {
38         if (p != -1) {
39             auto it = find(g[u].begin(), g[u].end(), p);

```



```

40         assert(it != g[u].end());
41         g[u].erase(it);
42     }
43
44     for (auto& v : g[u]) {
45         dfs_hld(v, u);
46         sz[u] += sz[v];
47         if (sz[v] > sz[g[u][0]])
48             swap(v, g[u][0]);
49     }
50 };
51 dfs_hld(0, -1);
52
53 vector<vector<pair<int, int>>> qry(n);
54 for (int i = 0; i < m; i++) {
55     int v, k; cin >> v >> k; v--;
56     qry[v].emplace_back(k, i);
57 }
58
59 const int maxC = 100000;
60 FreqBuckets cnt(n, maxC);
61 vector<int> ans(m, -1);
62
63 function<void(int, int)> dfs_addonly = [&](int u, int mul) {
64     cnt.add(c[u], mul);
65     for (auto& v : g[u])
66         dfs_addonly(v, mul);
67 };
68
69 function<void(int)> dfs_solve = [&](int u) {
70     for (auto& v : g[u]) {
71         if (v == g[u][0]) continue;
72         dfs_solve(v);
73         dfs_addonly(v, -1);
74     }
75
76     if (!g[u].empty())
77         dfs_solve(g[u][0]);
78     cnt.add(c[u], +1);
79     for (auto& v : g[u]) {
80         if (v == g[u][0]) continue;
81         dfs_addonly(v, +1);
82     }
83
84     for (auto& [k, i] : qry[u])
85         ans[i] = (k <= n ? cnt.freq[k] : 0);
86 };
87 dfs_solve(0);
88
89 for (int i = 0; i < m; i++)
90     cout << ans[i] << "\n";
91
92 return 0;
93 }

```

## Flow

### MaxFlow

```

1  constexpr int inf = 1E9;
2
3  template<class T>
4  struct MaxFlow {
5      struct _Edge {
6          int to;
7          T cap;
8          _Edge(int to, T cap) : to(to), cap(cap) {}
9      };
10
11     int n;

```

```

12     std::vector<_Edge> e;
13     std::vector<std::vector<int>>> g;
14     std::vector<int> cur, h;
15
16     MaxFlow() {}
17     MaxFlow(int n) {
18         init(n);
19     }
20
21     void init(int n) {
22         this->n = n;
23         e.clear();
24         g.assign(n, {});
25         cur.resize(n);
26         h.resize(n);
27     }
28
29     bool bfs(int s, int t) {
30         h.assign(n, -1);
31         std::queue<int> que;
32         h[s] = 0;
33         que.push(s);
34         while (!que.empty()) {
35             const int u = que.front();
36             que.pop();
37             for (int i : g[u]) {
38                 auto [v, c] = e[i];
39                 if (c > 0 && h[v] == -1) {
40                     h[v] = h[u] + 1;
41                     if (v == t) {
42                         return true;
43                     }
44                     que.push(v);
45                 }
46             }
47         }
48         return false;
49     }
50
51     T dfs(int u, int t, T f) {
52         if (u == t) {
53             return f;
54         }
55         auto r = f;
56         for (int &i = cur[u]; i < int(g[u].size()); ++i) {
57             const int j = g[u][i];
58             auto [v, c] = e[j];
59             if (c > 0 && h[v] == h[u] + 1) {
60                 auto a = dfs(v, t, std::min(r, c));
61                 e[j].cap -= a;
62                 e[j ^ 1].cap += a;
63                 r -= a;
64                 if (r == 0) {
65                     return f;
66                 }
67             }
68         }
69         return f - r;
70     }
71
72     void addEdge(int u, int v, T c) {
73         g[u].push_back(e.size());
74         e.emplace_back(v, c);
75         g[v].push_back(e.size());
76         e.emplace_back(u, 0);
77     }
78
79     T flow(int s, int t) {
80         T ans = 0;
81         while (bfs(s, t)) {
82             cur.assign(n, 0);
83             ans += dfs(s, t, std::numeric_limits<T>::max());
84         }

```

```

83     return ans;
84 }
85
86 std::vector<bool> minCut() {
87     std::vector<bool> c(n);
88     for (int i = 0; i < n; i++) {
89         c[i] = (h[i] != -1);
90     }
91     return c;
92 }
93
94 struct Edge {
95     int from;
96     int to;
97     T cap;
98     T flow;
99 };
100 std::vector<Edge> edges() {
101     std::vector<Edge> a;
102     for (int i = 0; i < e.size(); i += 2) {
103         Edge x;
104         x.from = e[i + 1].to;
105         x.to = e[i].to;
106         x.cap = e[i].cap + e[i + 1].cap;
107         x.flow = e[i + 1].cap;
108         a.push_back(x);
109     }
110     return a;
111 }
112 };

```

## Min Cost Flow

```

1  constexpr int inf = 1E9;
2
3  template<class T>
4  struct MinCostFlow {
5      struct _Edge {
6          int to;
7          T cap;
8          T cost;
9          _Edge(int to_, T cap_, T cost_) : to(to_), cap(cap_), cost(cost_) {}
10     };
11     int n;
12     std::vector<_Edge> e;
13     std::vector<std::vector<int>> g;
14     std::vector<T> h, dis;
15     std::vector<int> pre;
16     bool dijkstra(int s, int t) {
17         dis.assign(n, std::numeric_limits<T>::max());
18         pre.assign(n, -1);
19         std::priority_queue<std::pair<T, int>, std::vector<std::pair<T, int>>, std::greater<std::pair<T, int>>> que;
20         dis[s] = 0;
21         que.emplace(0, s);
22         while (!que.empty()) {
23             T d = que.top().first;
24             int u = que.top().second;
25             que.pop();
26             if (dis[u] != d) {
27                 continue;
28             }
29             for (int i : g[u]) {
30                 int v = e[i].to;
31                 T cap = e[i].cap;
32                 T cost = e[i].cost;
33                 if (cap > 0 && dis[v] > d + h[u] - h[v] + cost) {
34                     dis[v] = d + h[u] - h[v] + cost;
35                     pre[v] = i;
36                     que.emplace(dis[v], v);
37                 }
38             }

```

```

39     }
40     return dis[t] != std::numeric_limits<T>::max();
41 }
42 MinCostFlow() {}
43 MinCostFlow(int n_) {
44     init(n_);
45 }
46 void init(int n_) {
47     n = n_;
48     e.clear();
49     g.assign(n, {});
50 }
51 void addEdge(int u, int v, T cap, T cost) {
52     g[u].push_back(e.size());
53     e.emplace_back(v, cap, cost);
54     g[v].push_back(e.size());
55     e.emplace_back(u, 0, -cost);
56 }
57 std::pair<T, T> flow(int s, int t) {
58     T flow = 0;
59     T cost = 0;
60     h.assign(n, 0);
61     while (dijkstra(s, t)) {
62         for (int i = 0; i < n; ++i) {
63             h[i] += dis[i];
64         }
65         T aug = std::numeric_limits<int>::max();
66         for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
67             aug = std::min(aug, e[pre[i]].cap);
68         }
69         for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
70             e[pre[i]].cap -= aug;
71             e[pre[i] ^ 1].cap += aug;
72         }
73         flow += aug;
74         cost += aug * h[t];
75     }
76     return std::make_pair(flow, cost);
77 }
78 struct Edge {
79     int from;
80     int to;
81     T cap;
82     T cost;
83     T flow;
84 };
85 std::vector<Edge> edges() {
86     std::vector<Edge> a;
87     for (int i = 0; i < e.size(); i += 2) {
88         Edge x;
89         x.from = e[i + 1].to;
90         x.to = e[i].to;
91         x.cap = e[i].cap + e[i + 1].cap;
92         x.cost = e[i].cost;
93         x.flow = e[i + 1].cap;
94         a.push_back(x);
95     }
96     return a;
97 }
98 };

```

## String

### KMP

```

1  int main(){
2      string s;
3      cin >> s;
4      int n = (int)s.size();
5      s = " " + s;

```

```

6
7     vector<int> nxt(n + 1);
8     for (int i = 2, p = 0; i <= n; i++) {
9         while (p && s[p + 1] != s[i]) p = nxt[p];
10        p = nxt[i] = p + (s[p + 1] == s[i]);
11    }
12
13
14    string t;
15    cin >> t;
16    int m = (int)t.size();
17    t = " " + t;
18    for (int i = 1, p = 0; i <= m; i++) {
19        while (p && s[p + 1] != t[i]) p = nxt[p];
20        p += (s[p + 1] == t[i]);
21        if (p == n) {
22            cout << i - n + 1 << "\n";
23            p = nxt[p];
24        }
25    }
26
27    for (int i = 1; i <= n; i++) {
28        cout << nxt[i] << "\n";
29    }
30
31    return 0;
32 }

```

## Trie

```

1 struct Trie {
2     const int S = 26;
3     int N;
4
5     int cnt = 0; // root = 0
6
7     vector<vector<int>> tr;
8     vector<int> pos;
9
10    Trie() : N(0) {}
11    Trie(int n) : N(2 * n + 5), tr(N, vector<int>(S)), pos(N) {}
12
13    void insert(string s, int id) {
14        int p = 0;
15        for (char it: s) {
16            if (!tr[p][it - 'a']) tr[p][it - 'a'] = ++cnt;
17            p = tr[p][it - 'a'];
18        }
19    }
20 };

```

## GSAM

```

1 struct GSAM {
2     const int S = 26;
3     int N;
4
5     vector<vector<int>> son;
6     vector<int> len, fa;
7
8     int cnt = 1; // root = 1
9
10    GSAM() : N(0) {}
11    GSAM(int n) : N(2 * n + 5), son(N, vector<int>(S)), len(N), fa(N) {}
12
13    int insert(int p, int it) {
14        int cur = ++cnt;
15        len[cur] = len[p] + 1;
16        while (!son[p][it]) son[p][it] = cur, p = fa[p];
17        if (!p) return fa[cur] = 1, cur;

```

```

18     int q = son[p][it];
19     if (len[p] + 1 == len[q]) return fa[cur] = q, cur;
20     int cl = ++cnt;
21     son[cl] = son[q];
22     len[cl] = len[p] + 1, fa[cl] = fa[q], fa[q] = fa[cur] = cl;
23     while (son[p][it] == q) son[p][it] = cl, p = fa[p];
24     return cur;
25 }
26
27 void build(Trie &T) {
28     queue<int> Q;
29     Q.push(0), T.pos[0] = 1;
30     while (!Q.empty()) {
31         int cur = Q.front();
32         Q.pop();
33         for (int i = 0; i < S; i++) {
34             int p = T.tr[cur][i];
35             if (!p) continue;
36             T.pos[p] = insert(T.pos[cur], i), Q.push(p);
37         }
38     }
39 }
40 };

```

## ACA

```

1  #include <bits/stdc++.h>
2
3  using i64 = long long;
4
5  struct AhoCorasick {
6      static constexpr int ALPHABET = 26;
7      struct Node {
8          int len;
9          int link;
10         std::array<int, ALPHABET> next;
11         Node() : link{}, next{} {}
12     };
13
14     std::vector<Node> t;
15
16     AhoCorasick() {
17         init();
18     }
19
20     void init() {
21         t.assign(2, Node());
22         t[0].next.fill(1);
23         t[0].len = -1;
24     }
25
26     int newNode() {
27         t.emplace_back();
28         return t.size() - 1;
29     }
30
31     int add(const std::vector<int> &a) {
32         int p = 1;
33         for (auto x : a) {
34             if (t[p].next[x] == 0) {
35                 t[p].next[x] = newNode();
36                 t[t[p].next[x]].len = t[p].len + 1;
37             }
38             p = t[p].next[x];
39         }
40         return p;
41     }
42
43     int add(const std::string &a, char offset = 'a') {
44         std::vector<int> b(a.size());
45         for (int i = 0; i < a.size(); i++) {

```

```

46         b[i] = a[i] - offset;
47     }
48     return add(b);
49 }
50
51 void work() {
52     std::queue<int> q;
53     q.push(1);
54
55     while (!q.empty()) {
56         int x = q.front();
57         q.pop();
58
59         for (int i = 0; i < ALPHABET; i++) {
60             if (t[x].next[i] == 0) {
61                 t[x].next[i] = t[t[x].link].next[i];
62             } else {
63                 t[t[x].next[i]].link = t[t[x].link].next[i];
64                 q.push(t[x].next[i]);
65             }
66         }
67     }
68 }
69
70 int next(int p, int x) {
71     return t[p].next[x];
72 }
73
74 int next(int p, char c, char offset = 'a') {
75     return next(p, c - 'a');
76 }
77
78 int link(int p) {
79     return t[p].link;
80 }
81
82 int len(int p) {
83     return t[p].len;
84 }
85
86 int size() {
87     return t.size();
88 }
89 };

```

## Manacher

```

1  using i64 = long long;
2  std::vector<int> manacher(std::string s) {
3      std::string t = "#";
4      for (auto c : s) {
5          t += c;
6          t += '#';
7      }
8      int n = t.size();
9      std::vector<int> r(n);
10     for (int i = 0, j = 0; i < n; i++) {
11         if (2 * j - i >= 0 && j + r[j] > i) {
12             r[i] = std::min(r[2 * j - i], j + r[j] - i);
13         }
14         while (i - r[i] >= 0 && i + r[i] < n && t[i - r[i]] == t[i + r[i]]) {
15             r[i] += 1;
16         }
17         if (i + r[i] > j + r[j]) {
18             j = i;
19         }
20     }
21     return r;
22 }

```

## SAM

```
1 struct SuffixAutomaton {
2     static constexpr int ALPHABET_SIZE = 26, N = 1e6;
3     struct Node {
4         int len;
5         int link;
6         int next[ALPHABET_SIZE];
7         Node() : len(0), link(0), next{} {}
8     } t[2 * N];
9     int cntNodes;
10    SuffixAutomaton() {
11        cntNodes = 1;
12        std::fill(t[0].next, t[0].next + ALPHABET_SIZE, 1);
13        t[0].len = -1;
14    }
15    int extend(int p, int c) {
16        if (t[p].next[c]) {
17            int q = t[p].next[c];
18            if (t[q].len == t[p].len + 1)
19                return q;
20            int r = ++cntNodes;
21            t[r].len = t[p].len + 1;
22            t[r].link = t[q].link;
23            std::copy(t[q].next, t[q].next + ALPHABET_SIZE, t[r].next);
24            t[q].link = r;
25            while (t[p].next[c] == q) {
26                t[p].next[c] = r;
27                p = t[p].link;
28            }
29            return r;
30        }
31        int cur = ++cntNodes;
32        t[cur].len = t[p].len + 1;
33        while (!t[p].next[c]) {
34            t[p].next[c] = cur;
35            p = t[p].link;
36        }
37        t[cur].link = extend(p, c);
38        return cur;
39    }
40 };
```

## Geometry

### Geometry (Jiangly)

```
1 using i64 = long long;
2 template<class T>
3 struct Point {
4     T x;
5     T y;
6     Point(T x_ = 0, T y_ = 0) : x(x_), y(y_) {}
7
8     template<class U>
9     operator Point<U>() {
10         return Point<U>(U(x), U(y));
11     }
12     Point &operator+=(Point p) & {
13         x += p.x;
14         y += p.y;
15         return *this;
16     }
17     Point &operator-=(Point p) & {
18         x -= p.x;
19         y -= p.y;
20         return *this;
21     }
22     Point &operator*=(T v) & {
23         x *= v;
```



```

24         y *= v;
25         return *this;
26     }
27     Point &operator/=(T v) & {
28         x /= v;
29         y /= v;
30         return *this;
31     }
32     Point operator-() const {
33         return Point(-x, -y);
34     }
35     friend Point operator+(Point a, Point b) {
36         return a += b;
37     }
38     friend Point operator-(Point a, Point b) {
39         return a -= b;
40     }
41     friend Point operator*(Point a, T b) {
42         return a *= b;
43     }
44     friend Point operator/(Point a, T b) {
45         return a /= b;
46     }
47     friend Point operator*(T a, Point b) {
48         return b *= a;
49     }
50     friend bool operator==(Point a, Point b) {
51         return a.x == b.x && a.y == b.y;
52     }
53     friend std::istream &operator>>(std::istream &is, Point &p) {
54         return is >> p.x >> p.y;
55     }
56     friend std::ostream &operator<<(std::ostream &os, Point p) {
57         return os << "(" << p.x << ", " << p.y << ")";
58     }
59 };
60
61 template<class T>
62 T dot(Point<T> a, Point<T> b) {
63     return a.x * b.x + a.y * b.y;
64 }
65
66 template<class T>
67 T cross(Point<T> a, Point<T> b) {
68     return a.x * b.y - a.y * b.x;
69 }
70
71 template<class T>
72 T square(Point<T> p) {
73     return dot(p, p);
74 }
75
76 template<class T>
77 double length(Point<T> p) {
78     return std::sqrt(double(square(p)));
79 }
80
81 long double length(Point<long double> p) {
82     return std::sqrt(square(p));
83 }
84
85 template<class T>
86 Point<T> normalize(Point<T> p) {
87     return p / length(p);
88 }
89
90 template<class T>
91 struct Line {
92     Point<T> a;
93     Point<T> b;
94     Line(Point<T> a_ = Point<T>(), Point<T> b_ = Point<T>()) : a(a_), b(b_) {}

```

```

95 };
96
97 template<class T>
98 Point<T> rotate(Point<T> a) {
99     return Point(-a.y, a.x);
100 }
101
102 template<class T>
103 int sgn(Point<T> a) {
104     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
105 }
106
107 template<class T>
108 bool pointOnLineLeft(Point<T> p, Line<T> l) {
109     return cross(l.b - l.a, p - l.a) > 0;
110 }
111
112 template<class T>
113 Point<T> lineIntersection(Line<T> l1, Line<T> l2) {
114     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1.a - l1.b));
115 }
116
117 template<class T>
118 bool pointOnSegment(Point<T> p, Line<T> l) {
119     return cross(p - l.a, l.b - l.a) == 0 && std::min(l.a.x, l.b.x) <= p.x && p.x <= std::max(l.a.x, l.b.x)
120         && std::min(l.a.y, l.b.y) <= p.y && p.y <= std::max(l.a.y, l.b.y);
121 }
122
123 template<class T>
124 bool pointInPolygon(Point<T> a, std::vector<Point<T>> p) {
125     int n = p.size();
126     for (int i = 0; i < n; i++) {
127         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
128             return true;
129         }
130     }
131
132     int t = 0;
133     for (int i = 0; i < n; i++) {
134         auto u = p[i];
135         auto v = p[(i + 1) % n];
136         if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
137             t ^= 1;
138         }
139         if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
140             t ^= 1;
141         }
142     }
143
144     return t == 1;
145 }
146
147 // 0 : not intersect
148 // 1 : strictly intersect
149 // 2 : overlap
150 // 3 : intersect at endpoint
151 template<class T>
152 std::tuple<int, Point<T>, Point<T>> segmentIntersection(Line<T> l1, Line<T> l2) {
153     if (std::max(l1.a.x, l1.b.x) < std::min(l2.a.x, l2.b.x)) {
154         return {0, Point<T>(), Point<T>()};
155     }
156     if (std::min(l1.a.x, l1.b.x) > std::max(l2.a.x, l2.b.x)) {
157         return {0, Point<T>(), Point<T>()};
158     }
159     if (std::max(l1.a.y, l1.b.y) < std::min(l2.a.y, l2.b.y)) {
160         return {0, Point<T>(), Point<T>()};
161     }
162     if (std::min(l1.a.y, l1.b.y) > std::max(l2.a.y, l2.b.y)) {
163         return {0, Point<T>(), Point<T>()};
164     }
165     if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {

```

```

166     if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
167         return {0, Point<T>(), Point<T>()};
168     } else {
169         auto maxx1 = std::max(l1.a.x, l1.b.x);
170         auto minx1 = std::min(l1.a.x, l1.b.x);
171         auto maxy1 = std::max(l1.a.y, l1.b.y);
172         auto miny1 = std::min(l1.a.y, l1.b.y);
173         auto maxx2 = std::max(l2.a.x, l2.b.x);
174         auto minx2 = std::min(l2.a.x, l2.b.x);
175         auto maxy2 = std::max(l2.a.y, l2.b.y);
176         auto miny2 = std::min(l2.a.y, l2.b.y);
177         Point<T> p1(std::max(minx1, minx2), std::max(miny1, miny2));
178         Point<T> p2(std::min(maxx1, maxx2), std::min(maxy1, maxy2));
179         if (!pointOnSegment(p1, l1)) {
180             std::swap(p1.y, p2.y);
181         }
182         if (p1 == p2) {
183             return {3, p1, p2};
184         } else {
185             return {2, p1, p2};
186         }
187     }
188 }
189 auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
190 auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
191 auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
192 auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
193
194 if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) || (cp3 < 0 && cp4 < 0)) {
195     return {0, Point<T>(), Point<T>()};
196 }
197
198 Point p = lineIntersection(l1, l2);
199 if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
200     return {1, p, p};
201 } else {
202     return {3, p, p};
203 }
204 }
205
206 template<class T>
207 bool segmentInPolygon(Line<T> l, std::vector<Point<T>> p) {
208     int n = p.size();
209     if (!pointInPolygon(l.a, p)) {
210         return false;
211     }
212     if (!pointInPolygon(l.b, p)) {
213         return false;
214     }
215     for (int i = 0; i < n; i++) {
216         auto u = p[i];
217         auto v = p[(i + 1) % n];
218         auto w = p[(i + 2) % n];
219         auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
220
221         if (t == 1) {
222             return false;
223         }
224         if (t == 0) {
225             continue;
226         }
227         if (t == 2) {
228             if (pointOnSegment(v, l) && v != l.a && v != l.b) {
229                 if (cross(v - u, w - v) > 0) {
230                     return false;
231                 }
232             }
233         } else {
234             if (p1 != u && p1 != v) {
235                 if (pointOnLineLeft(l.a, Line(v, u))
236                     || pointOnLineLeft(l.b, Line(v, u))) {

```

```

237         return false;
238     }
239     } else if (p1 == v) {
240         if (l.a == v) {
241             if (pointOnLineLeft(u, l)) {
242                 if (pointOnLineLeft(w, l)
243                     && pointOnLineLeft(w, Line(u, v))) {
244                     return false;
245                 }
246             } else {
247                 if (pointOnLineLeft(w, l)
248                     || pointOnLineLeft(w, Line(u, v))) {
249                     return false;
250                 }
251             }
252         } else if (l.b == v) {
253             if (pointOnLineLeft(u, Line(l.b, l.a))) {
254                 if (pointOnLineLeft(w, Line(l.b, l.a))
255                     && pointOnLineLeft(w, Line(u, v))) {
256                     return false;
257                 }
258             } else {
259                 if (pointOnLineLeft(w, Line(l.b, l.a))
260                     || pointOnLineLeft(w, Line(u, v))) {
261                     return false;
262                 }
263             }
264         } else {
265             if (pointOnLineLeft(u, l)) {
266                 if (pointOnLineLeft(w, Line(l.b, l.a))
267                     || pointOnLineLeft(w, Line(u, v))) {
268                     return false;
269                 }
270             } else {
271                 if (pointOnLineLeft(w, l)
272                     || pointOnLineLeft(w, Line(u, v))) {
273                     return false;
274                 }
275             }
276         }
277     }
278 }
279 }
280 return true;
281 }
282
283 template<class T>
284 std::vector<Point<T>> hp(std::vector<Line<T>> lines) {
285     std::sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
286         auto d1 = l1.b - l1.a;
287         auto d2 = l2.b - l2.a;
288
289         if (sgn(d1) != sgn(d2)) {
290             return sgn(d1) == 1;
291         }
292
293         return cross(d1, d2) > 0;
294     });
295
296     std::deque<Line<T>> ls;
297     std::deque<Point<T>> ps;
298     for (auto l : lines) {
299         if (ls.empty()) {
300             ls.push_back(l);
301             continue;
302         }
303
304         while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
305             ps.pop_back();
306             ls.pop_back();
307         }

```

```

308
309     while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
310         ps.pop_front();
311         ls.pop_front();
312     }
313
314     if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
315         if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
316
317             if (!pointOnLineLeft(ls.back().a, l)) {
318                 assert(ls.size() == 1);
319                 ls[0] = l;
320             }
321             continue;
322         }
323         return {};
324     }
325
326     ps.push_back(lineIntersection(ls.back(), l));
327     ls.push_back(l);
328 }
329
330 while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
331     ps.pop_back();
332     ls.pop_back();
333 }
334 if (ls.size() <= 2) {
335     return {};
336 }
337 ps.push_back(lineIntersection(ls[0], ls.back()));
338
339 return std::vector(ps.begin(), ps.end());
340 }
341
342 using i128 = __int128;
343 using P = Point<i128>;

```

## Minkowski Sum

```

1  template<class P>
2  vector<P> minkowski(vector<P> C1, vector<P> C2){
3      auto reorder_polygon = [](vector<P> &pts){
4          size_t pos = 0;
5          for (size_t i = 1; i < pts.size(); i++){
6              if(pts[i].y < pts[pos].y || (pts[i].y == pts[pos].y && pts[i].x < pts[pos].x))
7                  pos = i;
8          }
9          rotate(pts.begin(), pts.begin() + pos, pts.end());
10     };
11     reorder_polygon(C1);
12     reorder_polygon(C2);
13     C1.push_back(C1[0]);
14     C1.push_back(C1[1]);
15     C2.push_back(C2[0]);
16     C2.push_back(C2[1]);
17     vector<P> ret;
18     size_t i = 0, j = 0;
19     while (i < C1.size() - 2 || j < C2.size() - 2){
20         ret.push_back(C1[i] + C2[j]);
21         auto cross = (C1[i + 1] - C1[i]).cross(C2[j + 1] - C2[j]);
22         if (cross >= 0 && i < C1.size() - 2)
23             ++i;
24         if (cross <= 0 && j < C2.size() - 2)
25             ++j;
26     }
27     return ret;
28 }

```