C++ Snippets

Muhammad Samir Assawalhy

July 25, 2024

Contents

1 Competitive programming template

```
1 //
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 #ifdef SAWALHY
6 #include "debug.hpp"
```

```
#else
#define debug(...) 0
#define debug_itr(...) 0
#define debug_bits(...) 0
#endif

#define ll long long
#define int long long
#define all(v) v.begin(), v.end()
#define rall(v) v.rbegin(), v.rend()
#define minit(v, x) v = min(v, x)
#define maxit(v, x) v = max(v, x)

int32_t main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL), cout.tie(NULL);

    ${0}

    return 0;
}
```

2 Competitive programming template with multi-tests

```
//
#include <bits/stdc++.h>
using namespace std;

#ifdef SAWALHY
#include "debug.hpp"
#else
#define debug(...) 0
#define debug_itr(...) 0
#define debug_bits(...) 0
#endif

#define ll long long
#define int long long
#define all(v) v.begin(), v.end()
#define rall(v) v.rbegin(), v.rend()
#define minit(v, x) v = min(v, x)
#define maxit(v, x) v = max(v, x)

void solve() {
    ${0}
}

int32_t main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL), cout.tie(NULL);
```

```
27
28      int t;
29      cin >> t;
30      while (t--)
31          solve();
32
33      return 0;
34  }
```

3 Increase the stack memory limit

```
1   static void run_with_stack_size(void (*func)(void), size_t stsize) {
2       char *stack, *send;
3       stack = (char *)malloc(stsize);
4       send = stack + stsize - 16;
5       send = (char *)((uintptr_t)send / 16 * 16);
6       asm volatile(
7           "mov %%rsp, (%0)\n"
8           "mov %0, %%rsp\n"
9           :
10          : "r"(send));
11      func();
12      asm volatile("mov (%0), %%rsp\n" : : "r"(send));
13      free(stack);
14  }
15
16  int32_t main() {
17      run_with_stack_size(main_, 1024 * 1024 * 1024 / 2); // run with a 512
            MB stack
18      return 0;
19  }
```

4 Read an array of length n from the stdin

```
1   int n;
2   cin >> n;
3   vector<int> a(n);
4   for (int i = 0; i < n; i++) {
5       cin >> a[i];
6   }
```

5 Primality test (brute force)

```
1   bool is_prime(ll n) {
2       if (n < 2) return false;
3       if (n == 2) return true;
4       if (n % 2 == 0) return false;
5       for (ll i = 3; i * i <= n; i += 2)
```

```
        if (n % i == 0) return false;
    return true;
}
```

6 Primality test (miller & rabin probabilistic)

```
bool miller_rabin_ptest(unsigned ll n, int k = 3) {
    if (n < 2) return false;
    if (n == 2) return true;

    while (k--) {
        unsigned ll a = 1LL * rand() * rand() % (n - 2) + 2;// [2 ... n-1]
        unsigned ll r = 1;
        for (unsigned ll p = n - 1; p; p >>= 1) {
            if (p & 1) r = r * a % n;
            a = a * a % n;
        }
        if (r != 1) return false;
    }

    return true;// probably
}
```

7 Prime factorization in o(sqrt(n))

```
map<ll, ll> primefacts(ll n) {
    map<ll, ll> result;
    int r = 0;

    while (n % 2 == 0) {
        r++;
        n = n / 2;
    }

    if (r > 0)
        result[2] = r;

    int sqn = sqrt(n);
    for (int i = 3; i <= sqn; i += 2) {
        r = 0;
        while (n % i == 0) {
            r++;
            n = n / i;
        }
        if (r > 0)
            result[i] = r;
    }

    if (n > 2)
```

```
        result[n] = 1;

    return result;
}
```

8 Euler's totient theorm

```
std::vector<int> phi(${1:n} + 1);
std::iota(phi.begin(), phi.end(), 0);

for (int i = 1; i <= ${2:$1}; i++) {
    for (int j = i << 1; j <= ${2:$1}; j += i)
        phi[j] -= phi[i];
}
```

9 Sieve's algorithm to mark numbers as primes and composites

```
void sieve(vector<bool> &is_prime) {
    is_prime[1] = false;
    is_prime[0] = false;
    int s = is_prime.size();
    for (int i = 4; i < s; i += 2)
        is_prime[i] = false;
    for (int i = 3; i * i < s; i += 2) {
        if (is_prime[i]) {
            for (int j = i * i; j < s; j += i + i)
                is_prime[j] = false;
        }
    }
}
```

10 Fast sieve's algorithm to calc minimum prime

```
std::vector<int> minp, primes;
void sieve(int n) {
    minp.assign(n + 1, 0);
    primes.clear();

    for (int i = 2; i <= n; i++) {
        if (minp[i] == 0) {
            minp[i] = i;
            primes.push_back(i);
        }

        for (auto p : primes) {
            if (i * p > n) {
```

```
                break;
            }
            minp[i * p] = p;
            if (p == minp[i]) {
                break;
            }
        }
    }
}
```

11 Dijkstra's algorithm

```
long long dijkstra(int s, int e, vector<vector<pair<int, int>>> &adj) {
    int n = adj.size();
    vector<int> prev(n + 1);
    vector<ll> dist(n + 1, 1e18);

    typedef pair<ll, int> item;
    priority_queue<item, deque<item>, greater<item>> qu;
    qu.push({0, s});
    dist[s] = 0;

    while (!qu.empty()) {
        auto [d, i] = qu.top();
        qu.pop();

        if (dist[i] < d) continue;
        for (auto [j, D]: adj[i]) {
            if (dist[j] > D + d) {
                prev[j] = i;
                dist[j] = D + d;
                qu.push({dist[j], j});
            }
        }
    }

    // for (int i = e; i != s; i = prev[i]);
    return dist[e];
}
```

12 Mst (kruskal's algorithm)

```
struct Edge {
    int from, to;
    long long weight;
    Edge(int from, int to, long long weight) : from(from), to(to), weight(
       weight) {}
    bool operator<(Edge &e) { return weight < e.weight; }
};
```

```cpp

pair<long long, vector<Edge>> mst_kruskal(vector<Edge> &edges, int n) {
    DSU uf(n + 1);
    double cost = 0;
    vector<Edge> mst_edges;

    sort(edges.rbegin(), edges.rend());

    while (!edges.empty()) {
        auto &e = edges.back();
        edges.pop_back();
        if (uf.uni(e.from, e.to)) {
            cost += e.weight;
            mst_edges.push_back(e);
        }
    };

    if (mst_edges.size() != n - 1)
        return {1e18, {}};

    return {cost, mst_edges};
}
```

13 Geometry stuff for competitive prgramming

```cpp
namespace Geometry
{

using T = long double;
const T EPS = 1e-8;
const double PI = acos(-1.0);

template<typename T, typename V>
int cmp(T a, V b) { return (a -= b) < -EPS ? -1 : (a > EPS ? 1 : 0); }
template<typename T, typename V>
bool iseq(T a, V b) { return cmp(a, b) == 0; }
template<typename T>
bool iseq0(T a) { return cmp(a, 0) == 0; }
template<typename T, typename V>
bool islte(T a, V b) { return cmp(a, b) != 1; }
template<typename T, typename V>
bool isgte(T a, V b) { return cmp(a, b) != -1; }
template<typename T, typename V>
bool islt(T a, V b) { return cmp(a, b) == -1; }
template<typename T, typename V>
bool isgt(T a, V b) { return cmp(a, b) == 1; }
template<typename T>
int sign(T val) { return cmp(val, 0); }

enum PointState { OUT,
```

```
26                  IN,
27                  ON };
28
29  typedef struct Point {
30      T x, y;
31
32      Point() {}
33      Point(T _x, T _y) : x(_x), y(_y) {}
34      Point operator+(const Point &p) const { return Point(x + p.x, y + p.y)
           ; }
35      Point operator-(const Point &p) const { return Point(x - p.x, y - p.y)
           ; }
36      Point operator/(T denom) const { return Point(x / denom, y / denom); }
37      Point operator*(T scaler) const { return Point(x * scaler, y * scaler)
           ; }
38
39      T dot(const Point &p) const { return x * p.x + y * p.y; }
40      T cross(const Point &p) const { return x * p.y - y * p.x; }
41      T dot(const Point &a, const Point &b) const { return (a - *this).dot(b
            - *this); }
42      T cross(const Point &a, const Point &b) const { return (a - *this).
           cross(b - *this); }
43      T norm() const { return dot(*this); }
44
45      long double len() const { return sqrtl(dot(*this)); }
46      long double ang(bool pos = true) const {
47          auto a = atan2l(y, x);
48          if (pos && a < 0) a += PI * 2;
49          return a;
50      }
51
52      Point rotate(const Point &p, long double a) { return (*this - p).
           rotate(a) + p; }
53      Point rotate(long double angle) {
54          auto l = len(), a = ang();
55          return Point(l * cos(a + angle), l * sin(a + angle));
56      }
57
58      bool operator==(const Point &p) const { return (*this - p).norm() <=
           EPS; }
59      bool operator!=(const Point &p) const { return !(*this == p); }
60      bool operator<(const Point &p) const { return x < p.x || (x == p.x &&
           y < p.y); }
61      friend ostream &operator<<(ostream &os, const Point &p) { return os <<
            '(' << p.x << ',' << p.y << ')'; }
62      friend istream &operator>>(istream &is, Point &p) { return is >> p.x
           >> p.y; }
63  } pt;
64
65  int ccw(const pt &a, pt b, pt c) {
66      if (a == b) return (a == c ? 0 : +3); // same point or different
```

```
67      b = b - a, c = c - a;
68      if (sign(b.cross(c)) == +1) return +1;          // "COUNTER_CLOCKWISE"
69      if (sign(b.cross(c)) == -1) return -1;          // "CLOCKWISE"
70      if (sign(b.dot(c)) == -1) return +2;            // "ON_RAY_b_a)"
71      if (cmp(b.norm(), c.norm()) == -1) return -2; // "ON_RAY_a_b"
72      return 0;                                       // "ON_SEGMENT"
73  }
74
75  bool colinear(const pt &a, const pt &b, const pt &c) {
76      return abs(ccw(a, b, c)) != 1;
77  }
78
79  pt slope(pt a, pt b, bool change_direction = true) {
80      assert(is_integral_v<T>);
81      long long dx = a.x - b.x;
82      long long dy = a.y - b.y;
83      if (dx == 0 && dy == 0) return pt(0, 0);
84      long long g = gcd(abs(dy), abs(dy));
85      dx /= g, dy /= g;
86      if (change_direction) {
87          if (dx < 0) dy *= -1, dx *= -1;
88          if (dx == 0) dy = abs(dy);
89      }
90      return pt(dx, dy);
91  }
92
93  struct Segment {
94      pt a, b;
95      Segment() {}
96      Segment(pt a, pt b) : a(a), b(b) {}
97      bool operator==(const Segment &s) const { return a == s.a ? b == s.b :
            a == s.b && b == s.a; };
98      friend istream &operator>>(istream &is, Segment &s) { return is >> s.a
            >> s.b; }
99      friend ostream &operator<<(ostream &os, const Segment &s) {
100         return os << "{" << s.a << ", " << s.b << "}";
101     }
102 };
103
104 struct Line : public Segment {
105     Line() {}
106     Line(pt a, pt b) : Segment(a, b) {}
107     bool operator==(const Line &l) const { return iseq0((a - b).cross(l.a
          - l.b)); };
108 };
109
110 struct Ray : public Segment {
111     Ray() {}
112     Ray(pt a, pt b) : Segment(a, b) {}
113     bool operator==(const Ray &r) const { return a == r.a && slope(a, b,
          false) == slope(r.a, r.b, false); };
```

```
};

struct Polygon {
    int n;
    vector<pt> verts;
    Polygon() = default;
    Polygon(int n) : n(n) { verts.resize(n); }
    Polygon(vector<pt> &vert) : verts(vert), n(vert.size()) {}

    T area2() const {
        T a = 0;
        for (int i = 2; i < n; i++)
            a += verts[0].cross(verts[i], verts[i - 1]);
        return abs(a);
    }

    long double area() const { return area2() / 2.0; };

    void no_collinear() {
        vector<pt> v;
        for (int i = 0; i <= n; i++) {
            while (v.size() > 1 && colinear(v.back(), v.end()[-2], verts[i
                % n]))
                v.pop_back();
            v.push_back(verts[i % n]);
        }
        v.pop_back();
        n = v.size();
        verts = v;
        assert(n > 2);
    }

    void ensure_ccw() {
        start_bottom_left();
        if (ccw(verts[0], verts[1], verts.back()) == -1)
            reverse(verts.begin() + 1, verts.end());
    }

    void start_bottom_left() {
        int pos = 0; // most left-bottom point
        for (int i = 1; i < n; i++)
            if (verts[i] < verts[pos])
                pos = i;
        rotate(verts.begin(), verts.begin() + pos, verts.end());
    }
};

bool parallel(const Line &a, const Line &b) { return (a.b - a.a).cross(b.b
    - b.a) == 0; }
bool orthogonal(const Line &a, const Line &b) { return (a.a - a.b).dot(b.a
    - b.b) == 0; }
```

```
162
163 bool intersect(const Line &l, const Line &m) { return !parallel(l, m); }
164
165 bool intersect(const pt &p, const Segment &s) { return ccw(s.a, s.b, p) ==
        0; }
166 bool intersect(const Segment &s, const pt &p) { return intersect(p, s); }
167
168 bool intersect(const pt &p, const Line &l) { return abs(ccw(l.a, l.b, p))
       != 1; }
169 bool intersect(const Line &l, const pt &p) { return intersect(p, l); }
170
171 bool intersect(const Segment &s, const Line &l) { return ccw(l.a, l.b, s.a
       ) * ccw(l.a, l.b, s.b) != 1; }
172 bool intersect(const Line &l, const Segment &s) { return intersect(s, l);
       }
173
174 bool intersect(const Segment &s, const Segment &t) { return ccw(s.a, s.b,
       t.a) * ccw(s.a, s.b, t.b) <= 0 && ccw(t.a, t.b, s.a) * ccw(t.a, t.b, s.
       b) <= 0; }
175
176 bool intersect(const Segment &s, const Ray &r) {
177     auto d1 = (s.a - s.b).cross(r.b - r.a),
178          d2 = (s.a - r.a).cross(r.b - r.a),
179          d3 = (s.a - s.b).cross(s.a - r.a);
180     if (abs(d1) <= EPS)
181         return r.a.cross(r.b, s.a) == 0 &&
182                (r.a.dot(r.b, s.a) >= 0 || r.a.dot(r.b, s.b) >= 0); // NOT
                   BACK
183     return sign(d1) * sign(d2) >= 0 && sign(d1) * sign(d3) >= 0 && abs(d2)
            <= abs(d1);
184 }
185
186 bool intersect(const Ray &r, const Segment &s) { return intersect(s, r); }
187
188 bool intersection(pt a, pt b, pt c, pt d, pt &inter) {
189     assert(is_floating_point_v<T>);
190     long double d1 = (a - b).cross(d - c);
191     long double d2 = (a - c).cross(d - c);
192     if (fabs(d1) <= EPS) return false;
193     long double t1 = d2 / d1;
194     inter = a + (b - a) * t1;
195     return true;
196 }
197
198 template<typename T, typename V>
199 bool intersection(const T &l, const V &m, pt &inter) {
200     if (!intersect(l, m)) return false;
201     return intersection(l.a, l.b, m.a, m.b, inter);
202 }
203
204 // - NOTE: The polygon shouldn't have collinear points.
```

```
205 // - NOTE: First vertex should be the bottom-left, points in ccw order.
206 vector<pt> intersection(const Polygon &poly, const Line &line) {
207     int n = poly.n;
208     vector<pt> inter;
209     const vector<pt> &verts = poly.verts;
210 
211     pt x;
212     for (int i = 1; i <= n; i++) {
213         int I = i % n, J = i - 1, K = (i - 2 + n) % n;
214         if (intersection(line, Segment(verts[I], verts[J]), x)) {
215             if (x == verts[I]) continue;
216             if (x != verts[J]) {
217                 inter.push_back(x);
218                 continue;
219             }
220             int dir1 = ccw(line.a, line.b, verts[I]);
221             int dir2 = ccw(line.a, line.b, verts[K]);
222             if (dir1 * dir2 == -1)
223                 // entering or leaving from a vertex
224                 inter.push_back(verts[J]);
225         } else if (abs(ccw(line.a, line.b, verts[J])) != 1) {
226             // side (I, J) is on the line
227             bool isWideAngleI = islt(ccw(verts[I], verts[((I + 1) % n)],
                    verts[J]), 0);
228             bool isWideAngleJ = islt(ccw(verts[J], verts[I], verts[K]), 0)
                    ;
229             if (isWideAngleI) inter.push_back(verts[I]);
230             if (isWideAngleJ) inter.push_back(verts[J]);
231             inter.push_back(verts[I]);
232             inter.push_back(verts[J]);
233         }
234     }
235 
236     debug(inter);
237 
238     // sort in one direction, as if you travel on the line
239     // in this direction and see the points one by one
240     // NOTE: points may NOT be eaxctly on the line due to precesion errors
241     sort(all(inter), [&](pt l, pt r) {
242         return sign((line.b - line.a).dot(r - l)) == 1;
243     });
244 
245     assert(inter.size() % 2 == 0);
246     return inter;
247 };
248 
249 struct Circle {
250     pt c;
251     T r;
252 
253     Circle() = default;
```

```
254     Circle(pt c, T r) : c(c), r(r) {}
255     Circle(const vector<pt> &p) {
256         if (p.size() == 1) c = p[0], r = 0;
257         else if (p.size() == 2) {
258             c = (p[0] + p[1]) / 2;
259             r = (p[0] - c).len();
260         } else {
261             assert(p.size() == 3);
262             *this = Circle(p[0], p[1], p[2]);
263         }
264     }
265 
266     Circle(pt a, pt b, pt c) {
267         // if we have a cord in a circle,
268         // the perpendicular from the center will pass from the center
269         // so we simply solve for the interection of two lines
270         auto ABmid = (a + b) / 2.0, BCmid = (b + c) / 2.0;
271         auto ABnorm = pt((a - b).y, -(a - b).x);
272         auto BCnorm = pt((b - c).y, -(b - c).x);
273         bool valid = intersection(
274                 Line(ABmid, ABmid + ABnorm),
275                 Line(BCmid, BCmid + BCnorm), this->c);
276         assert(valid); // unless at least two points are identical
277         r = (a - this->c).len();
278     }
279 
280     friend ostream &operator<<(ostream &os, const Circle &c) {
281         return os << "c{" << c.c << ", " << c.r << "}";
282     }
283 };
284 
285 PointState point_in_triangle(pt a, pt b, pt c, pt point) {
286     int x = ccw(a, b, point), y = ccw(b, c, point), z = ccw(c, a, point);
287     if (sign(x) == sign(y) && sign(y) == sign(z)) return IN;
288     if (x * y * z == 0) return ON;
289     return OUT;
290 }
291 
292 PointState point_in_circle(const pt &p, const vector<pt> &cir) {
293     if (cir.size() == 0) return OUT;
294     auto c = Circle(cir);
295     if (iseq((p - c.c).norm(), c.r * c.r)) return ON;
296     if (islt((p - c.c).norm(), c.r * c.r)) return IN;
297     return OUT;
298 }
299 
300 PointState point_in_polygon(const pt &p, const vector<pt> &polygon) {
301     int wn = 0, n = polygon.size();
302     for (int i = 0, j = 1; i < n; i++, j++, j %= n) {
303         if (ccw(polygon[j], polygon[i], p) == 0) return ON;
304         if ((p.y < polygon[j].y) != (p.y < polygon[i].y)) {
```

```
305             wn += polygon[j].y > polygon[i].y && ccw(p, polygon[i],
                   polygon[j]) == 1;
306             wn -= polygon[j].y < polygon[i].y && ccw(p, polygon[j],
                   polygon[i]) == 1;
307         }
308     }
309     return wn == 0 ? OUT : IN;
310 }
311
312 PointState ray_and_polygon(const Ray &r, const Polygon &polygon) {
313     // NOTE: Should be a good ray (a != b),
314     // and non-degenerate polygon with no duplicated points
315     int n = polygon.n;
316     PointState ans = OUT;
317     for (int i = 0, j = 1, k = 2; i < n; i++, j++, k++, j %= n, k %= n) {
318         if (!intersect(Segment(polygon.verts[i], polygon.verts[j]), r))
               continue;
319         auto x = r.a.cross(r.b, polygon.verts[i]);
320         auto y = r.a.cross(r.b, polygon.verts[j]);
321         auto z = r.a.cross(r.b, polygon.verts[k]);
322         if (x == 0) ans = ON; // Maybe tangent
323         else if (y == 0) {
324             // (the ray splits an internal angle)
325             // Entering from a vertex
326             if (sign(x) * sign(z) == -1) return IN;
327         } else return IN; // Entering from an edge
328     }
329     return ans;
330 }
331
332 vector<pt> &sort_clock(vector<pt> &points, bool cw = false) {
333     int n = points.size();
334
335     // choose the pivot (most bottom-right point)
336     for (int i = 1; i < n; i++) {
337         auto &l = points[0], &r = points[i];
338         int cy = cmp(l.y, r.y), cx = cmp(l.x, r.x);
339         if (cy == 0 ? cx == -1 : cy == +1) swap(l, r);
340     }
341
342     // sorting with points[0] as pivot
343     sort(points.begin() + 1, points.end(),
344          [&](pt l, pt r) {
345              auto c = ccw(points[0], l, r);
346              int cx = cmp(l.x, r.x), cy = cmp(l.y, r.y);
347              // closer to bottom-right comes first
348              if (abs(c) != 1) return cy == 0 ? cx == 1 : cy == -1;
349              return cw ? c == -1 : c == 1;
350          });
351
352     return points;
```

```
353 }
354
355 // sort a convex polygon cw or ccw with the bottom-right as the pivot
356 vector<pt> &sort_convex(vector<pt> &points, bool cw = false) {
357     int n = points.size();
358
359     // choose the pivot (most bottom-right point)
360     for (int i = 1; i < n; i++) {
361         auto &l = points[0], &r = points[i];
362         int cy = cmp(l.y, r.y), cx = cmp(l.x, r.x);
363         if (cy == 0 ? cx == -1 : cy == +1) swap(l, r);
364     }
365
366     // sorting with points[0] as pivot
367     sort(points.begin() + 1, points.end(),
368          [&](pt l, pt r) {
369              auto c = ccw(points[0], l, r);
370              int cx = cmp(l.x, r.x), cy = cmp(l.y, r.y);
371
372              if (abs(c) != 1) { // collinear
373                  if (cw) return cy == 0 ? cx == 1 : cy == 1;
374                  else
375                      return cy == 0 ? cx == -1 : cy == -1;
376              }
377
378              return cw ? c == -1 : c == 1;
379          });
380
381     return points;
382 }
383
384 vector<pt> convexhull(vector<pt> &p, bool strict = false) {
385     int n = p.size(), k = 0, sgn = strict ? 0 : -1;
386     if (n <= 2) return p;
387     vector<pt> ch(2 * n); // CCW
388     auto cmp = [](pt x, pt y) { return (x.x != y.x ? x.x < y.x : x.y < y.y
        ); };
389     sort(begin(p), end(p), cmp);
390     for (int i = 0; i < n; ch[k++] = p[i++]) // lower hull
391         while (k >= 2 && sign((ch[k - 1] - ch[k - 2]).cross(p[i] - ch[k -
            1])) <= sgn) --k;
392     for (int i = n - 2, t = k + 1; i >= 0; ch[k++] = p[i--]) // upper hull
393         while (k >= t && sign((ch[k - 1] - ch[k - 2]).cross(p[i] - ch[k -
            1])) <= sgn) --k;
394     ch.resize(k - 1);
395     return ch;
396 }
397
398 struct PointInConvex {
399     int n;
400     vector<pt> seq;
```

```
401     pt translation;
402
403     PointInConvex(vector<pt> polygon) { prepare_convex_ccw(polygon); }
404
405     void prepare_convex_ccw(vector<pt> &points) {
406         // NOTE: the polygon should be strictly convex
407         n = points.size();
408         int pos = 0; // most left-bottom point
409         for (int i = 1; i < n; i++)
410             if (points[i] < points[pos])
411                 pos = i;
412         rotate(points.begin(), points.begin() + pos, points.end());
413
414         seq.resize(n);
415         for (int i = 0; i < n; i++)
416             seq[i] = points[(i + 1) % n] - points[0];
417         translation = points[0];
418     }
419
420     int check(pt point) {
421         point = point - translation;
422         if (intersect(point, Segment(pt(0, 0), seq[0]))) return 0;
423         if (seq.size() <= 2) return -1;
424
425         int l = 0, r = n - 1;
426         while (r - l > 1) {
427             int mid = (l + r) / 2;
428             if (sign(seq[mid].cross(point)) != -1)
429                 l = mid;
430             else
431                 r = mid;
432         }
433
434         int ok = point_in_triangle(seq[l], seq[l + 1], pt(0, 0), point);
435         if (ok == -1) return -1;
436         if (intersect(point, Segment(seq[l], seq[l + 1]))) return 0;
437         return 1;
438     }
439 };
440
441 struct Welzl {
442     vector<pt> points;
443     Welzl(vector<pt> &_points) : points(_points) {
444         shuffle(all(points), default_random_engine(time(NULL)));
445     }
446
447     Circle get_circle() { return Circle(go()); }
448     vector<pt> go(int i = 0, vector<pt> cir = {}) {
449         if (cir.size() == 3 || i == (int) points.size()) return cir;
450         auto new_cir = go(i + 1, cir);
451         if (point_in_circle(points[i], new_cir) != OUT)
```

```
            return new_cir;
        cir.push_back(points[i]);
        return go(i + 1, cir);
    }
};

}; // namespace Geometry

using namespace Geometry;
```

14 Stl policy container (oset, omap)

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template<typename T>
using ordered_set = tree<T, null_type, std::less<T>, rb_tree_tag,
   tree_order_statistics_node_update>;
```

15 Segment tree (simple implementation)

```
// source: https://codeforces.com/blog/entry/18051
template<typename T = long long, T DEFAULT = T(1e18)>
struct Segtree {
    int n = 0;
    vector<T> tree;

    Segtree() = default;
    Segtree(int n) : n(n) { tree.assign(n * 2, DEFAULT); }

    inline T merge(const T &a, const T &b) { return min(a, b); }

    void build() {
        for (int i = n - 1; i > 0; i--)
            tree[i] = merge(tree[i << 1], tree[i << 1 | 1]);
    }

    void update(int i, T val) {
        for (tree[i += n] = val; i > 1; i >>= 1)
            tree[i >> 1] = merge(tree[i], tree[i ^ 1]);
    }

    auto query(int l, int r) {
        T resl = DEFAULT, resr = resl;
        for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
            if (l & 1) resl = merge(resl, tree[l++]);
            if (r & 1) resr = merge(tree[--r], resr);
        }
        return merge(resl, resr);
```

```
    }
};
```

16 Segment tree (sawalhy's implementation)

```
struct Value;
struct Update;
struct Node;

// Replaceable by primitives (using Value = long long)
struct Value {
    long long sum = 0, mn = 1e18, mx = -1e18;
    Value() = default;
    Value(ll value) { sum = mn = mx = value; }

    Value &operator+=(const Value &other) {
        sum += other.sum;
        mn = min(mn, other.mn);
        mx = max(mx, other.mx);
        return *this;
    }

    Value operator+(const Value &other) const {
        return Value(*this) += other;
    }
};

struct Update {
    // NOTE: Sometime you need to split the update, in these cases
    // you should include the range [a, b] of the update in the struct
       Update
    Value value;
    enum State {
        idle,
        relative,
        forced
    } state = idle;

    Update() = default;
    Update(Value value, State state = forced) : value(value), state(state)
       {};

    Update &operator+=(const Update &other) {
        if (state == idle || other.state == forced) {
            *this = other;
        } else {
            assert(other.state == relative);
            value += other.value;
        }
        return *this;
```

```
    }

    void apply_on(Value &other, int cnt) const {
        if (state == forced) other = value;
        else other += value;
        other.sum += value.sum * (cnt - 1);
    }

    Update get(const Node &node) const { return *this; }
};

struct Node {
    int l = -1, r = -1; // [l, r]
    Update up;
    Value value;

    Node() = default;
    Node(int l, int r, const Value &value) : l(l), r(r), value(value){};

    void update(const Update &up) { this->up += up; }

    void apply_update() {
        up.apply_on(value, r - l + 1);
        up.state = Update::idle;
    }
};

struct Segtree {
    int n;
    vector<Node> tree;

    Segtree(int n) {
        if ((n & (n - 1)) != 0)
            n = 1 << (32 - __builtin_clz(n));
        this->n = n;
        tree.assign(n << 1, Node());
        for (int i = n; i < n << 1; i++)
            tree[i].l = tree[i].r = i - n;
        for (int i = n - 1; i > 0; i--)
            tree[i].l = tree[i << 1].l, tree[i].r = tree[i << 1 | 1].r;
    }

    Segtree(const vector<Value> &values) : Segtree(values.size()) {
        for (int i = 0; i < (int) values.size(); i++)
            tree[i + n].value = values[i];
        build();
    }

    void build() {
        for (int i = n - 1; i > 0; --i) pull(i);
    }
```

```
95
96      inline Value query(int i) { return query(1, i, i); }
97      inline Value query(int i, int j) { return query(1, i, j); }
98      inline void update(int i, const Update &val) { update(1, i, i, val); }
99      inline void update(int i, int j, const Update &val) { update(1, i, j,
            val); }
100
101 private:
102     void pull(int i) {
103         tree[i].value = tree[i << 1].value + tree[i << 1 | 1].value;
104     }
105
106     void push(int i) {
107         if (tree[i].up.state != Update::idle) {
108             if (i < n) {
109                 int l = i << 1, r = i << 1 | 1;
110                 tree[l].update(tree[i].up.get(tree[l]));
111                 tree[r].update(tree[i].up.get(tree[r]));
112             }
113             tree[i].apply_update();
114         }
115     }
116
117     Value query(int i, int l, int r) {
118         push(i);
119         if (tree[i].r < l || r < tree[i].l) return Value(); // default
120         if (l <= tree[i].l && tree[i].r <= r) return tree[i].value;
121         return query(i << 1, l, r) + query(i << 1 | 1, l, r);
122     }
123
124     void update(int i, int l, int r, const Update &up) {
125         push(i);
126         if (tree[i].r < l || r < tree[i].l) return;
127         if (l <= tree[i].l && tree[i].r <= r) {
128             tree[i].update(up);
129             push(i); // to apply the update
130             return;
131         }
132         update(i << 1, l, r, up.get(tree[i << 1]));
133         update(i << 1 | 1, l, r, up.get(tree[i << 1 | 1]));
134         pull(i);
135     }
136 };
```

17 Simple segtree with lazy update

```
1  template <class T = long long, T DEFAULT = 0>
2  struct Segtree {
3      int n;
4      vector<T> tree, ups;
```

```

    Segtree(int n) {
        if ((n & (n - 1)) != 0)
            n = 1 << (32 - __builtin_clz(n));
        this->n = n;
        tree.assign(n << 1, DEFAULT);
        ups.assign(n << 1, DEFAULT);
    }

    inline T merge(const T &a, const T &b) { return a + b; }

    inline void push(int i, int L, int R) {
        if (ups[i] != DEFAULT) {
            if (i < n) {
                ups[i << 1] = merge(ups[i << 1], ups[i]);
                ups[i << 1 | 1] = merge(ups[i << 1 | 1], ups[i]);
            }
            tree[i] = merge(tree[i], ups[i] * (R - L + 1));
            ups[i] = DEFAULT;
        }
    }

    T query(int l, int r, int i, int L, int R) {
        push(i, L, R);
        if (R < l || r < L) return DEFAULT;
        if (l <= L && R <= r) return tree[i];
        int mid = (L + R) / 2;
        return merge(
                query(l, r, i << 1, L, mid),
                query(l, r, i << 1 | 1, mid + 1, R));
    }

    void update(int l, int r, const T &up, int i, int L, int R) {
        push(i, L, R);
        if (R < l || r < L) return;
        if (l <= L && R <= r) {
            ups[i] += up, push(i, L, R);
            return;
        }
        int mid = (L + R) / 2;
        update(l, r, up, i << 1, L, mid);
        update(l, r, up, i << 1 | 1, mid + 1, R);
        tree[i] = merge(tree[i << 1], tree[i << 1 | 1]);
    }
};
```

18 Dynamic segtree + persistent segtree

```
const int MIN = -(1 << 30), MAX = (1 << 30) - 1;
const int DEFAULT = 0;
```

```
3
4  struct Node {
5      long long value = DEFAULT;
6      Node *l = nullptr, *r = nullptr;
7      Node() = default;
8      Node(Node *l, Node *r) : l(l), r(r) {
9          if (l) value += l->value;
10         if (r) value += r->value;
11     }
12 };
13
14 struct PersistentSegtree {
15     Node *root = nullptr;
16
17     Node *update(int x, long long value) {
18         return root = update(x, value, root, MIN, MAX);
19     }
20
21     long long query(int L, int R) {
22         return query(L, R, root, MIN, MAX);
23     }
24
25 private:
26     Node *update(int x, long long value, Node *node, int l, int r) {
27         if (l == r) {
28             assert(l == x);
29             Node *ret = new Node();
30             ret->value = value;
31             return ret;
32         }
33
34         int mid = l + (r - l) / 2;
35         Node *left = node ? node->l : nullptr;
36         Node *right = node ? node->r : nullptr;
37         if (x <= mid) return new Node(update(x, value, left, l, mid),
               right);
38         else return new Node(left, update(x, value, right, mid + 1, r));
39     }
40
41     long long query(int L, int R, Node *node, int l, int r) {
42         if (!node || L > r || R < l) return DEFAULT;
43         if (L <= l && r <= R) return node->value;
44         int mid = l + (r - l) / 2;
45         return query(L, R, node->l, l, mid) + query(L, R, node->r, mid +
               1, r);
46     }
47 };
```

19 Bit (1d simple implementation)

```cpp
struct BIT {
    int n;
    vector<int> tree;

    BIT(int size) : n(size) { tree.resize(n + 1, 0); }

    void update(int i, int delta) {
        for (; i <= n; i += i & -i) tree[i] += delta;
    }

    int query(int i) {
        int sum = 0;
        for (; i > 0; i -= i & -i) sum += tree[i];
        return sum;
    }
};
```

20 Bit (multiple dimensions)

```cpp
template<class T, int... Ns>
struct BIT {
    T val = 0;
    void update(T v) { val += v; }
    T query() { return val; }
};

template<class T, int N, int... Ns>
struct BIT<T, N, Ns...> {
    BIT<T, Ns...> bit[N + 1];
    template<typename... Args>
    void update(int pos, Args... args) {
        for (pos++; pos <= N; pos += (pos & -pos)) bit[pos].update(args
            ...);
    }
    template<typename... Args>
    T sum(int r, Args... args) {
        T res = 0;
        for (r++; r; r -= (r & -r)) res += bit[r].query(args...);
        return res;
    }
    template<typename... Args>
    T query(int l, int r, Args... args) {
        return sum(r, args...) - sum(l - 1, args...);
    }
}; // BIT<int,10,10> gives a 2D BIT
```

21 Sparse table

```cpp
template<typename T, class CMP = function<T(const T &, const T &)>>
```

```
class SparseTable {
public:
    int n;
    vector<vector<T>> sp;
    CMP func;

    void build(const vector<T> &a, const CMP &f) {
        func = f;
        n = static_cast<int>(a.size());
        int max_log = 32 - __builtin_clz(n);
        sp.resize(max_log);
        sp[0] = a;
        for (int j = 1; j < max_log; ++j) {
            sp[j].resize(n - (1 << j) + 1);
            for (int i = 0; i <= n - (1 << j); ++i) {
                sp[j][i] = func(sp[j - 1][i], sp[j - 1][i + (1 << (j - 1))
                    ]);
            }
        }
    }

    T query(int l, int r) const {
        int lg = 32 - __builtin_clz(r - l + 1) - 1;
        return func(sp[lg][l], sp[lg][r - (1 << lg) + 1]);
    }
};
```

22 Modular arithmetics stolen from jiangly

```
template<int32_t mod>
struct mint {
    using Z = mint;
    int32_t x;
    mint(int32_t x = 0) : x(norm(x)) {}
    mint(ll x) : x(norm(x % mod)) {}
    inline int32_t norm(int32_t x) const {
        return x >= mod ? x - mod : (x < 0  ? x + mod : x);
    }
    Z power(ll b) const {
        Z res = 1, a = x;
        for (; b; b >>= 1, a *= a)
            if (b & 1) res *= a;
        return res;
    }
    Z inv() const { return assert(x != 0), power(mod - 2); }
    Z operator-() const { return -x; }
    Z &operator*=(const Z &r) { return *this = (ll) x * r.x; }
    Z &operator+=(const Z &r) { return *this = x + r.x; }
    Z &operator-=(const Z &r) { return *this = x - r.x; }
    Z &operator/=(const Z &r) { return *this *= r.inv(); }
```

```
    friend Z operator*(const Z &l, const Z &r) { return Z(l) *= r; }
    friend Z operator+(const Z &l, const Z &r) { return Z(l) += r; }
    friend Z operator-(const Z &l, const Z &r) { return Z(l) -= r; }
    friend Z operator/(const Z &l, const Z &r) { return Z(l) /= r; }
    friend ostream &operator<<(ostream &os, const Z &a) { return os << a.x
       ; }
    friend istream &operator>>(istream &is, Z &a) {
        ll y = 0;
        return is >> y, a = y, is;
    }
};

// constexpr int MOD = 998244353;
constexpr int MOD = 1000000007;
using Z = mint<MOD>;
```

23 Modular combinations

```
vector<Z> fact = {1};
vector<Z> fact_inv = {1};

void build_fact(int n = 1e6) {
    while ((int) fact.size() < n + 1)
        fact.push_back(fact.back() * (int) fact.size());
    fact_inv.resize(fact.size());
    fact_inv.back() = fact.back().inv();
    for (int j = fact_inv.size() - 2; fact_inv[j].x == 0; j--)
        fact_inv[j] = fact_inv[j + 1] * (j + 1);
}

Z ncr(int n, int r) {
    if (r > n || r < 0) return 0;
    if ((int) fact.size() < n + 1) build_fact(n);
    return fact[n] * fact_inv[r] * fact_inv[n - r];
}
```

24 Disjoint set union

```
struct DSU {
    vector<int> size, parent;
    int forests;

    DSU(int n) {
        forests = n;
        size.assign(n, 1);
        parent.resize(n);
        iota(all(parent), 0);
    }

```

```
12      bool connected(int x, int y) { return find(x) == find(y); }
13
14      int find(int x) {
15          if (parent[x] == x) return x;
16          return parent[x] = find(parent[x]);
17      }
18
19      bool uni(int x, int y) {
20          x = find(x), y = find(y);
21          if (x == y) return false;
22          forests--;
23          parent[y] = x;
24          size[x] += size[y];
25          return true;
26      }
27  };
```

25 Matrix exponentiation

```
1   constexpr ll MOD = 1e9 + 7;
2
3   template<typename T = int, int mod = MOD>
4   struct matrix {
5       typedef vector<vector<T>> vv;
6       vv mat;
7       int n, m;
8
9       matrix() { n = 0, m = 0; }
10      matrix(vv mat) : mat(mat) { n = mat.size(), m = mat[0].size(); }
11      matrix(int n, int m, T ini = 0) : n(n), m(m) { mat = vv(n, vector<T>(m
            , ini)); }
12
13      matrix operator*(const matrix &other) const {
14          matrix mat = *this;
15          return mat *= other;
16      }
17
18      matrix operator+(const matrix &other) const {
19          matrix mat = *this;
20          return mat += other;
21      }
22
23      matrix operator-(const matrix &other) const {
24          matrix mat = *this;
25          return mat -= other;
26      }
27
28      matrix &operator*=(const matrix &other) {
29          assert(m == other.n);
30          vector<vector<T>> temp(n, vector<T>(other.m));
```

```
31          for (int i = 0; i < n; i++) {
32              for (int j = 0; j < other.m; j++) {
33                  for (int k = 0; k < m; k++) {
34                      temp[i][j] = (temp[i][j] + 1LL * mat[i][k] * other.mat
                            [k][j]) % mod;
35                  }
36              }
37          }
38          mat = temp;
39          m = other.m;
40          return *this;
41      }
42
43      matrix &operator+=(const matrix &other) {
44          assert(m == other.m && n == other.n);
45          for (int i = 0; i < n; i++) {
46              for (int j = 0; j < m; j++)
47                  mat[i][j] = ((mat[i][j] + other.mat[i][j]) % mod + mod) %
                        mod;
48          }
49          return *this;
50      }
51
52      matrix &operator-=(const matrix &other) {
53          assert(m == other.m && n == other.n);
54          for (int i = 0; i < n; i++) {
55              for (int j = 0; j < m; j++)
56                  mat[i][j] = ((mat[i][j] - other.mat[i][j]) % mod + mod) %
                        mod;
57          }
58          return *this;
59      }
60
61      matrix power(ll p) {
62          assert(p >= 0);
63          matrix m = *this;
64          matrix res = identity(n);
65          for (; p; p >>= 1, m *= m)
66              if (p & 1) res *= m;
67          return res;
68      }
69
70      static matrix identity(int size) {
71          matrix I = vv(size, vector<T>(size));
72          for (int i = 0; i < size; i++)
73              I.mat[i][i] = 1;
74          return I;
75      }
76  };
```

26 Fast input scanner

```
char in[1 << 24];
struct Scanner {
    char const *o;
    Scanner() : o(in) { load(); }
    void load() { in[fread(in, 1, sizeof(in) - 4, stdin)] = 0; }
    unsigned readInt() {
        unsigned u = 0;
        while (*o && *o <= 32)
            ++o;
        while (*o >= '0' && *o <= '9')
            u = u * 10 + (*o++ - '0');
        return u;
    }
} sc;
```

27 Pascal triagle, useful for combinations

```
vector<vector<Z>> pascal;
void build_pascal(int d) {
    pascal = {{1}};
    while (d--) {
        vector<Z> &lastrow = pascal.back();
        int s = lastrow.size();
        vector<Z> newrow(s + 1);
        newrow.front() = 1;
        newrow.back() = 1;
        for (int i = 1; i < s; i++)
            newrow[i] = lastrow[i] + lastrow[i - 1];
        pascal.push_back(newrow);
    }
}
```

28 Big integer

```
template<int base = 10>
class bigint {
public:
    vector<int> digits;

    bigint(unsigned ll value = 0) { set_value(value); }

    bigint(string s) {
        digits.resize(s.size());
        for (int i = (int) s.size() - 1; i >= 0; i--) {
            digits[i] = s[(int) s.size() - 1 - i] - '0';
        }
```

```
13      }
14
15      template<typename RandomIt>
16      bigint(RandomIt begin, RandomIt end) {
17          digits.assign(begin, end);
18      }
19
20      void set_value(ll value) {
21          digits.clear();
22          while (value) {
23              digits.push_back(value % base);
24              value /= base;
25          }
26      }
27
28      int size() const { return digits.size(); }
29
30      void trim() {
31          while (digits.back() == 0 && digits.size() > 1)
32              digits.pop_back();
33      }
34
35      int &operator[](int i) { return digits[i]; }
36
37      int operator[](int i) const { return digits[i]; }
38
39      void operator*=(const bigint &rhs) {
40          vector<int> res(size() + rhs.size() + 1);
41          for (int i = 0; i < size(); i++) {
42              for (int j = 0; j < rhs.size(); j++) {
43                  res[i + j] += digits[i] * rhs[j];
44              }
45          }
46          for (int i = 0; i < (int) res.size() - 1; i++) {
47              res[i + 1] += res[i] / base;
48              res[i] %= base;
49          }
50          digits = res;
51          trim();
52      }
53
54      void operator+=(const bigint &rhs) {
55          digits.resize(max(size(), rhs.size()) + 1);
56          int i;
57          for (i = 0; i < rhs.size(); i++) {
58              digits[i] += rhs[i];
59              if (digits[i] >= base) {
60                  digits[i + 1] += digits[i] / base;
61                  digits[i] %= base;
62              }
63          }
```

```
64          while (i < (int) digits.size() - 1 && digits[i] >= base) {
65              digits[i + 1] = digits[i] / base;
66              digits[i] %= base;
67          }
68          trim();
69      }
70
71      void operator%=(ll mod) {
72          ll p = 1;
73          ll res = 0;
74          for (int i = 0; i < size(); i++) {
75              res = (res + p * digits[i] % mod) % mod;
76              p = p * base % mod;
77          }
78          *this = res;
79      }
80
81      friend bool operator==(bigint &lhs, bigint &rhs) {
82          return lhs.digits == rhs.digits;
83      }
84
85      friend bool operator!=(bigint &lhs, bigint &rhs) {
86          return lhs.digits != rhs.digits;
87      }
88
89      friend bool operator<(bigint &lhs, bigint &rhs) {
90          if (lhs.size() != rhs.size())
91              return lhs.size() < rhs.size();
92          for (int i = lhs.size() - 1; i >= 0; i--) {
93              if (lhs[i] < rhs[i]) return true;
94              if (lhs[i] > rhs[i]) return false;
95          }
96          return false;// equal
97      }
98
99      friend ostream &operator<<(ostream &os, const bigint &bi) {
100         for (int i = bi.size() - 1; i >= 0; i--) os << bi[i];
101         return os;
102     }
103 };
```

29 Extended euclidian algorithm

```
1  // a * x + b * y = gcd(a, b)
2  pair<ll, ll> exgcd(ll a, ll b) {
3      if (!b) return {1, 0};
4      pair<int, int> p = exgcd(b, a % b);
5      return {p.second, p.first - (a / b) * p.second};
6  }
```

30 Modular inverse for coprimes not only prime mod

```
// source: https://codeforces.com/blog/entry/23365
// a and b must be co-prime. returns (1 / a) mod b.
ll mod_inv(ll a, ll b) {
    return 1 < a ? b - mod_inv(b % a, a) * b / a : 1;
}
```

31 Trie data structure

```
struct Trie {
    vector<vector<int>> child;
    vector<int> count;

    Trie() {
        add_node();
    }

    int add_node() {
        count.push_back(0);
        child.push_back(vector<int>(26));
        return count.size() - 1;
    }

    void insert(const string &s) {
        int cur = 0; // root
        for (auto c: s) {
            if (child[cur][c - 'a'] == 0)
                child[cur][c - 'a'] = add_node();
            cur = child[cur][c - 'a'];
            count[cur]++;
        }
    }
};
```

32 String hashing implementation (polynomial hashing)

```
typedef valarray<ll> val;
constexpr int hashes = 2;
vector<val> B;
const val M = {
        1000000007,
        1444444447,
        // 998244353,
};

```

```
void setB(int n) {
    if (B.size() == 0) {
        mt19937 rng(random_device{}());
        B.assign(2, val(1, hashes));
        for (int i = 0; i < hashes; i++)
            B.back()[i] = uniform_int_distribution<ll>(1, M[i] - 1)(rng);
    }
    while ((int) B.size() <= n) B.push_back(B.back() * B[1] % M);
}

struct Hash {
    vector<val> h;

    Hash(const string &s) : h(s.size() + 1) {
        setB(s.size()), h[0] = val(hashes);
        for (int i = 0; i < (int) s.size(); i++)
            h[i + 1] = (h[i] * B[1] + s[i]) % M;
    }

    auto get(int l, int r) {
        array<ll, hashes> arr;
        val ans = (h[r + 1] - h[l] * B[r - l + 1] % M + M) % M;
        for (int i = 0; i < hashes; i++) arr[i] = ans[i];
        return arr;
    }
};
```

33 Eulerian path/circuit in directed graphs

```
template<typename Edge>
class DirectedEulerian {
public:
    int n, m;
    vector<vector<pair<int, Edge>>> adj;

    DirectedEulerian(int n, int m) : n(n), m(m) {
        adj.assign(n, vector<pair<int, Edge>>());
    }

    void add_edge(int u, int v, Edge edge) {
        adj[u].emplace_back(v, edge);
    }

    vector<Edge> path(bool circuit = false) {
        vector<Edge> path;
        int in = 0, out = 0;

        calc_deg();
        int start = -1, end = -1;
        for (int i = 0; i < n; i++) {
```

```
22              if (indeg[i] > outdeg[i])
23                  in += indeg[i] - outdeg[i], end = i;
24              else if (indeg[i] < outdeg[i])
25                  out += outdeg[i] - indeg[i], start = i;
26          }
27
28          if (m == 0 || !((in == 0 && out == 0) || (in == 1 && out == 1 && !
               circuit))) {
29              return {};
30          }
31
32          if (start == -1) {
33              assert(end == -1);
34              for (int i = 0; i < n; i++) {
35                  if (outdeg[i] > 0) {
36                      start = end = i;
37                      break;
38                  }
39              }
40          }
41
42          dfs(start, {}, path);
43
44          path.pop_back();
45          reverse(all(path));
46
47          return path;
48      }
49
50  private:
51      vector<int> indeg, outdeg;
52
53      void calc_deg() {
54          indeg.assign(n, 0);
55          outdeg.assign(n, 0);
56          for (int i = 0; i < n; i++) {
57              outdeg[i] = adj[i].size();
58              for (auto &j: adj[i]) indeg[j.first]++;
59          }
60      }
61
62      void dfs(int i, Edge e, vector<Edge> &path) {
63          while (outdeg[i] > 0)
64              outdeg[i]--, dfs(adj[i][outdeg[i]].first, adj[i][outdeg[i]].
                   second, path);
65          path.push_back(e);
66      }
67  };
```

34 Eulerian path/circuit in undirected graphs

```
template<typename Edge>
class UndirectedEulerian {
public:
    int n, m;
    vector<vector<pair<int, Edge>>> adj; // NOTE: dont't add a self-edge twice

    UndirectedEulerian(int n, int m) : n(n), m(m) {
        adj.assign(n, vector<pair<int, Edge>>());
    }

    void add_edge(int u, int v, Edge edge) {
        adj[u].emplace_back(v, edge);
        adj[v].emplace_back(u, edge);
    }


    vector<Edge> path(bool circuit = false) {
        vector<Edge> path;

        cnt.clear();
        calc_deg();
        int start = -1, end = -1, odds = 0;
        for (int i = 0; i < n; i++) {
            if (deg[i] & 1) {
                odds++;
                if (~start)
                    end = i;
                else
                    start = i;
            }
        }

        if (m == 0 || !(odds == 0 || (odds == 2 && !circuit))) {
            return {};
        }

        if (start == -1) {
            assert(end == -1);
            for (int i = 0; i < n; i++) {
                if (deg[i] > 0) {
                    start = end = i;
                    break;
                }
            }
        }

        dfs(start, -1, {}, path);
```

```

        path.pop_back();
        reverse(all(path));

        return path;
    }

private:
    vector<int> deg;
    map<pair<int, int>, int> cnt;

    void calc_deg() {
        deg.assign(n, 0);
        for (int i = 0; i < n; i++) {
            for (auto &j: adj[i]) {
                deg[j.first]++;
                if (i == j.first)
                    deg[j.first]++;
                if (i <= j.first)
                    cnt[{i, j.first}]++;
            }
        }
    }

    void dfs(int i, int p, Edge e, vector<Edge> &path) {
        cnt[{min(i, p), max(i, p)}]--;
        while (adj[i].size()) {
            auto [j, E] = adj[i].back();
            adj[i].pop_back();
            if (cnt[{min(i, j), max(i, j)}] == 0) continue;
            dfs(j, i, E, path);
        }
        path.push_back(e);
    }
};
```

35 Mo's algorithm

```
int block_size;
struct MO {
    struct Query {
        int l, r, idx;
        bool operator<(const Query &q) const {
            if (l / block_size != q.l / block_size)
                return pair(l, r) < pair(q.l, q.r);
            return (l / block_size & 1) ? (r < q.r) : (r > q.r);
        }
    };

    vector<int> arr;
```

```
    vector<Query> queries;
    int l = 0, r = -1;

    MO() {}
    MO(const vector<int> &arr) : arr(arr) {}

    void add_query(const Query &q) {
        queries.push_back(q);
    }

    vector<int> get_ans() {
        block_size = arr.size() / sqrt(queries.size()) + 1;
        vector<int> ans(queries.size());
        sort(all(queries));

        l = queries.front().l, r = queries.front().l - 1;
        for (auto &q: queries) {
            set_range(q);
            ans[q.idx] = ans_query(q);
        }

        return ans;
    }

    void set_range(Query &q) {
        // [l, r] inclusive
        while (l > q.l) add(arr[--l]);
        while (r < q.r) add(arr[++r]);
        while (l < q.l) remove(arr[l++]);
        while (r > q.r) remove(arr[r--]);
    }

    void add(int x) {
    }

    void remove(int x) {
    }

    int ans_query(Query &q) {
    }
};
```

36 Mo algorithm on a tree using euler tour

```
struct MOTree {
    vector<vector<int>> adj;
    vector<int> arr, vals, st, en;
    int n;
    MO mo;
    LCA lca;
```

```

    MOTree(const vector<vector<int>> &adj, const vector<int> &vals) {
        this->vals = vals, this->adj = adj, n = adj.size();
        st.resize(n), en.resize(n);
        lca = LCA(adj);
        _dfs(0, 0, vals);
        mo = MO(arr);
    }

    void _dfs(int i, int p, const vector<int> vals) {
        st[i] = arr.size();
        arr.push_back(vals[i]);
        for (auto j: adj[i]) {
            if (p == j) continue;
            _dfs(j, i, vals);
        }
        en[i] = arr.size();
        arr.push_back(vals[i]);
    }

    void add_query(int u, int v, MO::Query q) {
        if (st[u] > st[v]) swap(u, v);
        int lc = lca.query(u, v);
        if (lc == u) q.l = st[u], q.r = st[v];
        else q.l = en[u], q.r = st[v], q.lc_value = vals[lc];
        mo.add_query(q);
    }

    auto get_ans() {
        return mo.get_ans();
    }
};
```

37 Torjan's algorithm, strongly connected components

```
struct SCC {
    int N, ID = 0, COMP = 0;
    vector<vector<int>> adj;
    vector<int> id, comp, st;

    SCC(const vector<vector<int>> &adj) : adj(adj), N(adj.size()) {
        id.resize(N), comp = vector<int>(N, -1);
        go();
    }

    void go() {
        for (int i = 0; i < N; i++)
            if (!id[i]) dfs(i);
    }

```

```
    int dfs(int i) {
        int low = id[i] = ++ID;
        st.push_back(i);
        for (int j: adj[i])
            if (comp[j] == -1)
                // id[j] != 0 -> in stack, don't dfs
                low = min(low, id[j] ?: dfs(j));
        if (low == id[i]) {
            COMP++;
            for (int j = -1; j != i;)
                comp[j = st.back()] = COMP, st.pop_back();
        }
        return low;
    }
};
```

38 Kmp string algorithm

```
vector<int> KMP(const string &a, const string &b) {
    // search for b in a
    vector<int> ans;
    int n = a.length(), m = b.length();
    int b_table[n];
    b_table[0] = 0;

    for (int i = 1, k = 0; i < m; i++) {
        while (k > 0 && b[k] != b[i])
            k = b_table[k - 1];
        k += b[i] == b[k];
        b_table[i] = k;
    }

    for (int i = 0, k = 0; i < n; i++) {
        while (k > 0 && b[k] != a[i])
            k = b_table[k - 1];
        k += b[k] == a[i];
        if (k == m) {
            k = b_table[k - 1];
            ans.push_back(i - m + 1);
        }
    }

    return ans;
}
```

39 Z algorithm for strings

```
vector<int> zfunction(string s) {
    int n = s.size();
```

```
3
4       vector<int> z(n);
5       for (int i = 1, l = 1, r = 1; i < n; i++) {
6           if (i < r) z[i] = min(z[i - l], r - i);
7           while (i + z[i] < n && s[i + z[i]] == s[z[i]]) z[i]++;
8           if (i + z[i] > r) r = i + z[i], l = i;
9       }
10
11      return z;
12  }
```

40 Aho corasick

```
1   struct AhoCorasick {
2       static constexpr int K = 100;
3       vector<vector<int>> child, id;
4       vector<int> fail;
5
6       AhoCorasick() { add_node(); }
7
8       int add_node() {
9           id.push_back({}), fail.push_back(0);
10          child.push_back(vector<int>(K));
11          return child.size() - 1;
12      }
13
14      int insert(const string &s, int ind) {
15          int cur = 0; // root
16          for (auto c: s) {
17              if (child[cur][c - 'A'] == 0)
18                  child[cur][c - 'A'] = add_node();
19              cur = child[cur][c - 'A'];
20          }
21          id[cur].push_back(ind);
22          return cur;
23      }
24
25      int go(int cur, int nxt) {
26          while (cur && !child[cur][nxt]) cur = fail[cur];
27          return child[cur][nxt];
28      }
29
30      void build() {
31          queue<int> q;
32          for (int i = 0; i < K; i++)
33              if (child[0][i]) q.push(child[0][i]);
34          while (!q.empty()) {
35              int cur = q.front();
36              q.pop();
37              for (int nxt = 0; nxt < K; nxt++) {
```

```
                if (!child[cur][nxt]) continue;
                fail[child[cur][nxt]] = go(fail[cur], nxt);
                q.push(child[cur][nxt]);
            }
        }
    }
};
```

41 Random utils

```
mt19937 rng = mt19937(random_device()());

void seed(int s) { rng = mt19937(s); }

int rand_int(int x, int y) {
    return uniform_int_distribution<int>(x, y)(rng);
}
```

42 Least common ancestor using binary lifting

```
struct LCA {
    int n, LOG;
    vector<int> depth;
    vector<vector<int>> up, adj;

    LCA(int n, int root = 0) : n(n), LOG(log2(n) + 1) {
        adj.resize(n), depth.resize(n);
        up.assign(n, vector<int>(LOG, root));
    }

    void add_edge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void dfs(int u, int p) {
        for (auto v: adj[u]) {
            if (v == p) continue;
            up[v][0] = u;
            depth[v] = depth[u] + 1;
            dfs(v, u);
        }
    }

    void build(int root = 0) {
        dfs(root, root);
        for (int k = 1; k < LOG; k++)
            for (int u = 0; u < n; u++)
                up[u][k] = up[up[u][k - 1]][k - 1];
```

```
    }

    int query(int u, int v) const {
        if (depth[u] < depth[v]) swap(u, v);
        for (int k = LOG - 1; k >= 0; k--) {
            if (depth[up[u][k]] >= depth[v]) {
                u = up[u][k];
            }
        }
        if (u == v) return u;
        for (int k = LOG - 1; k >= 0; k--) {
            if (up[u][k] != up[v][k]) {
                u = up[u][k];
                v = up[v][k];
            }
        }
        return up[u][0];
    }
};
```

43 Least common ancestor using sparse table

```
struct LCA {
    int n, LOG, _time;
    vector<int> first, depth;
    vector<vector<int>> table;

    LCA() {}
    LCA(const vector<vector<int>> &adj) { build(adj); }

    void _dfs(int u, int p, const vector<vector<int>> &adj) {
        first[u] = _time;
        table[0][_time++] = u;
        for (auto v: adj[u]) {
            if (v == p) continue;
            depth[v] = depth[u] + 1;
            _dfs(v, u, adj);
            table[0][_time++] = u;
        }
    }

    void build(const vector<vector<int>> &adj, int root = 0) {
        n = sz(adj), LOG = log2(n) + 3;
        depth.resize(n), first.resize(n);
        table.assign(LOG, vector<int>(2 * n));
        _time = 0;
        _dfs(root, root, adj);
        assert(_time < 2 * n);
        for (int i = 1; i < LOG; i++) {
            for (int j = 0; j + (1 << i) <= 2 * n; j++) {
```

```
                if (depth[table[i - 1][j]] < depth[table[i - 1][j + (1 << (i - 1))]]) {
                    table[i][j] = table[i - 1][j];
                } else {
                    table[i][j] = table[i - 1][j + (1 << (i - 1))];
                }
            }
        }
    }

    int query(int u, int v) const {
        u = first[u], v = first[v];
        if (u == v) return table[0][u];
        if (u > v) swap(u, v);
        int lg = 31 - __builtin_clz(v - u + 1);
        if (depth[table[lg][u]] < depth[table[lg][v - (1 << lg) + 1]]) {
            return table[lg][u];
        } else {
            return table[lg][v - (1 << lg) + 1];
        }
    }
};
```

44 Centroid decomposition of a tree

```
struct Centroids {
    vector<vector<int>> edges;
    vector<bool> removed;
    vector<int> par;
    vector<int> sz;
    int n;

    Centroids(int n) : n(n) {
        edges.resize(n), removed.resize(n);
        sz.resize(n), par.assign(n, -1);
    }

    void add_edge(int a, int b) {
        edges[a].push_back(b);
        edges[b].push_back(a);
    }

    void find_size(int v, int p = -1) {
        sz[v] = 1;
        for (int x: edges[v]) {
            if (x == p || removed[x]) continue;
            find_size(x, v), sz[v] += sz[x];
        }
    }

```

```
    int find_centroid(int v, int p, int n) {
        for (int x: edges[v]) {
            if (x == p || removed[x]) continue;
            if (sz[x] > n / 2) return find_centroid(x, v, n);
        }
        return v;
    }

    void build(int v = 0, int p = -1) {
        find_size(v);
        int c = find_centroid(v, -1, sz[v]);
        removed[c] = true, par[c] = p;
        for (int x: edges[c])
            if (!removed[x]) build(x, c);
    }
};
```

45 Generator utils for the stress tester

```
struct Gen {
    static vector<int> perm(int n) {
        vector<int> a(n);
        for (int i = 0; i < n; i++)
            a[i] = i;
        random_shuffle(a.begin(), a.end());
        return a;
    }

    static vector<int> tree_parents(int n) {
        vector<int> p(n - 1);
        auto a = perm(n);
        // a.begin(), node 1, is the root
        random_shuffle(a.begin() + 1, a.end());
        for (int i = 1; i < n; i++)
            p[i - 1] = a[rand_int(0, i - 1)] + 1;
        return p;
    }

    static vector<pair<int, int>> tree_edges(int n) {
        auto a = perm(n);
        vector<pair<int, int>> edges;
        for (int i = 1; i < n; i++)
            edges.push_back({a[i] + 1, a[rand_int(0, i - 1)] + 1});
        return edges;
    }


    static vector<vector<int>> tree_adj(int n) {
        vector<vector<int>> adj(n);
        for (auto [u, v]: tree_edges(n)) {
```

```
            u--, v--;
            adj[u].push_back(v);
            adj[v].push_back(u);
        }
        return adj;
    }

    static string str(int n) {
        string s;
        for (int i = 0; i < n; i++)
            s += rand_int('a', 'z');
        return s;
    }
};
```

46 Stress testing mechanism

```
namespace stress {

void brute() {

}

mt19937 rng = mt19937(random_device()());

void seed(int s) { rng = mt19937(s); }

int rand_int(int x, int y) {
    return uniform_int_distribution<int>(x, y)(rng);
}

void generate() {
    int n = rand_int(2, 20), q = rand_int(1, 5);

    cout << n << ' ' << q << endl;

    while (q--) {
        int l = rand_int(1ll, n - 1);
        int r = rand_int(l + 1, n);
        cout << l << ' ' << r << endl;
    }
}

string readAllTheFile(const string &filename) {
    ifstream file(filename);
    string content;
    string line;

    if (file.is_open()) {
        while (getline(file, line)) content += line + "\n";
```

```
34          file.close();
35      }
36      return content;
37  }
38
39  void stress(int argc, char **argv) {
40  #ifndef STRESS
41      return;
42  #endif
43      if (argc > 1) seed(stoi(argv[1]));
44      for (int iter = 0; iter < 100000; iter++) {
45          debug(iter);
46          FILE *input, *output;
47
48          input = freopen("/tmp/stress-input", "w", stdout);
49          generate();
50          fclose(input);
51
52          input = freopen("/tmp/stress-input", "r", stdin);
53          output = freopen("/tmp/stress-main", "w", stdout);
54          solve();
55          fclose(input);
56          fclose(output);
57
58          input = freopen("/tmp/stress-input", "r", stdin);
59          output = freopen("/tmp/stress-brute", "w", stdout);
60          brute();
61          fclose(input);
62          fclose(output);
63
64          string m = readAllTheFile("/tmp/stress-main");
65          string b = readAllTheFile("/tmp/stress-brute");
66
67          if (m != b) {
68              string i = readAllTheFile("/tmp/stress-input");
69              cerr << "input  ---------------------------------" << endl;
70              cerr << i << endl;
71              cerr << "main   ---------------------------------" << endl;
72              cerr << m << endl;
73              cerr << "brute  ---------------------------------" << endl;
74              cerr << b << endl;
75              exit(1);
76          }
77      }
78      exit(0);
79  }
80
81  }
```

47 Xor basis

```cpp
template<int bits>
struct XORBasis {
    array<int, bits> basis;

    void reset() { basis = array<int, bits>(); }

    void insert(const vector<int> &v) {
        for (auto x: v) insert(x);
    }

    bool insert(int x) {
        for (int b = 0; x && b < bits; b++) {
            if (x >> b & 1 ^ 1) continue;
            if (!basis[b]) return basis[b] = x, true;
            x ^= basis[b];
        }
        return false;
    }

    int size() {
        int sz = 0;
        for (auto b: basis) sz += !!b;
        return sz;
    }

    int mask(int x) {
        int m = 0;
        for (int b = 0; b < bits; b++) {
            if (x >> b & 1 ^ 1) continue;
            if (!basis[b]) return -1;
            x ^= basis[b], m |= 1 << b;
        }
        if (x != 0) return -1;
        return m;
    }

    int reduce(int x) {
        for (int b = 0; x && b < bits; b++)
            if (x >> b & 1) x ^= basis[b];
        return x;
    }
};
```