

Dividimos y No Conquistamos (D&!C)

Contents

1 Template	2
1.1 C++ Template	2
1.2 Python Template	2
2 Search	2
2.1 Ternary	2
2.2 Simulated Annealing	2
3 Data structures	3
3.1 Fenwick	3
3.2 Fenwick - 2D	3
3.3 DSU	3
3.4 Index Compession	3
3.5 Sparse Table	4
3.6 Segment tree	4
3.7 Segment Tree Iterativo	5
3.8 Segment Tree Persistente	5
3.9 Policy Based	5
3.10 SQRT Decomposition	5
3.11 Chull Trick	6
4 Graph	6
4.1 Bellman Ford	6
4.2 SCC	6
4.3 Bipartite Matching Hopcroft-Karp - With Konig	7
4.4 Hungarian	7
4.5 Flow - Dinics	8
4.6 Flow - MinCostMaxFlow	9
4.7 2 Sat	10
4.8 Euler Tour	11
5 Trees	11
5.1 Heavy Light Decomposition	11
5.2 Centroid	11
5.3 LCA - Binary exponentiation	12
5.4 LCA - Const Time	12

6 Dynamic Programming	13
6.1 Knapsack	13
6.2 LIS	13
6.3 Edit Distance	13
6.4 Kadane	13
7 Strings	14
7.1 Hashing	14
7.2 Trie	14
7.3 KMP	14
7.4 LPS	15
7.5 Z-FUNCTION	15
7.6 Manacher	15
7.7 Aho-Corasick	15
7.8 Suffix-Array	16
8 Math	16
8.1 Euclidean Extended	16
8.2 Euler Totient	17
8.3 Josephus	17
8.4 Mobius	18
8.5 NTT	18
8.6 FFT	19
8.7 Rho	20
8.8 Simpson	21
9 Geometry	21
9.1 Convex Hull	21
9.2 Operations	22
9.3 Polygon Area	22
9.4 Ray Casting	22
10 Other	22
10.1 Mo's algorithm	22

1 Template

1.1 C++ Template

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define L(i, j, n) for (int i = (j); i < (int)n; i++)
4 #define SZ(x) int((x).size())
5 #define ALL(x) begin(x),end(x)
6 #define vec vector
7 #define pb push_back
8 #define eb emplace_back
9 using ll = long long;
10 using ld = long double;
11 void solve(){}
12 int main(){
13     ios::sync_with_stdio(0);cin.tie(0);
14     int TT = 1;
15     //cin >> TT;
16     while (TT--) {solve();}
17 }
18 // IF NEEDED FOR FILE READ
19 // freopen("in.txt", "r", stdin);
20 // freopen("out.txt", "w", stdout);

```

1.2 Python Template

```

1 import os, sys, io
2 finput = io.BytesIO(os.read(0, os.fstat(0).st_size)).readline
3 fprint = sys.stdout.write

```

2 Search

2.1 Ternary

```

1 // Minimo de 'f' en '(l,r)'.
2 template<class Fun>ll ternary(Fun f, ll l, ll r) {
3     for (ll d = r-l; d > 2; d = r-l) {
4         ll a = l + d/3, b = r - d/3;
5         if (f(a) > f(b)) l = a; else r = b;
6     }
7     return l + 1;
8 }
9 // para error < EPS, usar iters=log((r-l)/EPS)/log(1.618)

```

```

10 template<class Fun>double golden(Fun f, double l, double r, int iters){
11     double const ratio = (3-sqrt(5))/2;
12     double x1=l+(r-l)*ratio, x2=r-(r-l)*ratio, f1=f(x1), f2=f(x2);
13     while (iters--) {
14         if (f1 > f2) l=x1, x1=x2, f1=f2, x2=r-(r-l)*ratio, f2=f(x2);
15         else r=x2, x2=x1, f2=f1, x1=l+(r-l)*ratio, f1=f(x1);
16     }
17     return (l+r)/2;
18 }

```

2.2 Simulated Annealing

```

1 using my_clock = chrono::steady_clock;
2 struct Random {
3     mt19937_64 engine;
4     Random(): engine(my_clock::now().time_since_epoch().count()) {}
5     template<class Int>Int integer(Int n) {return integer<Int>(0, n);} //
6     ' [0,n)'
7     template<class Int>Int integer(Int l, Int r)
8         {return uniform_int_distribution{l, r-1}(engine);} // '[l,r)'
9     double real() {return uniform_real_distribution{}(engine);} // '[0,1)'
10 } rng;
11 struct Timer {
12     using time = my_clock::time_point;
13     time start = my_clock::now();
14     double elapsed() { // Segundos desde el inicio.
15         time now = my_clock::now();
16         return chrono::duration<double>(now - start).count();
17     }
18 } timer;
19 template<class See, class Upd>struct Annealing {
20     using energy = invoke_result_t<See>;
21     energy curr, low;
22     See see;
23     Upd upd;
24     Annealing(See _see, Upd _upd): see{_see}, upd{_upd}
25     {curr = low = see(), upd();}
26     void simulate(double s, double mult=1) { // Simula por 's' segundos.
27         double t0 = timer.elapsed();
28         for (double t = t0; t-t0 < s; t = timer.elapsed()) {
29             energy near = see();
30             auto delta = double(curr - near);

```

```

31     else {
32         double temp = mult * (1 - (t-t0)/s);
33         if (exp(delta/temp) > rng.real()) upd(), curr = near;
34     }
35 }
36 };
37 auto see = [&] -> double {
38     l = rng.integer(gsz); r = rng.integer(gsz);
39     swap(groups[l], groups[r]);
40     int ans = 0, rem =0;
41     L(i,0,gsz){
42         if (groups[i] > rem) {
43             rem = x;
44             ans++;
45         }
46         rem -= groups[i];
47     }
48     swap(groups[l], groups[r]);
49     return ans;
50 };
51 auto upd = [&] {swap(groups[l], groups[r]);};

```

3 Data structures

3.1 Fenwick

```

1 #define LSO(S) (S & -S) //Least significant One
2 struct FT { // 1-Index
3     vec<int> ft; int n;
4     FT(vec<int> &v): ft(SZ(v)+1), n(SZ(v)+1) { // O(n)
5         L(i, 1, n){
6             ft[i] += v[i-1];
7             if (i + LSO(i) <= n) ft[i + LSO(i)]+=ft[i];
8         }
9     }
10    void update(int pos, int x){ for (int it=pos;it<=n;it+=LSO(it))ft[it]
11        ]+=x; }
12    int sum(int pos){
13        int res = 0;
14        for (int it=pos;it>0;it-=LSO(it))res+=ft[it];
15        return res;
16    }

```

```

16     int getSum(int l, int r){return sum(r) - sum(l - 1);}
17 }

```

3.2 Fenwick - 2D

```

1 #define LSO(S) (S & -S)
2 struct BIT { // 1-Index
3     vec<vec<int>> B;
4     int n; // BUILD: N * N * log(N) * log(N)
5     BIT(int n_ = 1): B(n_+1,vec<int>(n_+1)), sz(n_) {}
6     void add(int i, int j, int delta){ // log(N) * log(N)
7         for (int x = i; x <= n; x += LSO(x))
8             for (int y = j; y <= n; y += LSO(y))
9                 B[x][y] += delta;
10    }
11    int sum(int i, int j){ // log(N) * log(N)
12        int tot = 0;
13        for (int x = i; x > 0; x -= LSO(x))
14            for(int y = j; y > 0; y -= LSO(y))
15                tot += B[x][y];
16        return tot;
17    }
18    int getSum(int x1, int y1, int x2, int y2) {return sum(x2, y2) - sum
19        (x2, y1) - sum(x1, y2) + sum(x1-1,y1-1);}
}

```

3.3 DSU

```

1 struct DSU {
2     vec<int> par, sz; int n;
3     DSU(int n = 1): par(n), sz(n, 1), n(n) { iota(ALL(par), 0); }
4     int find(int a){return a == par[a] ? a : par[a] = find(par[a]);}
5     void join(int a, int b){
6         a=find(a);b=find(b);
7         if (a == b) return;
8         if (sz[b] > sz[a]) swap(a,b);
9         par[b] = a; sz[a] += sz[b];
10    }
11 }

```

3.4 Index Compession

```

1 template<class T>
2 struct Index{ // If only 1 use Don't need to copy T type

```

```

3   vec<T> d; int sz;
4   Index(const vec<T> &a): d(ALL(a)){
5     sort(ALL(d)); // Sort
6     d.erase(unique(ALL(d)), end(d)); // Erase continuous duplicates
7     sz = SZ(d); }
8   inline int of(T e) const{return lower_bound(ALL(d), e) - begin(d);}
9     // get index
10  inline T at(int i) const{return d[i];} // get value of index
11 };

```

3.5 Sparse Table

```

1 struct SPT {
2   vec<vec<int>> st;
3   SPT(vec<int> &a) {
4     int n = SZ(a), K = 0; while((1<<K)<=n) K++;
5     st = vec<vec<int>>(K, vec<int>(n));
6     L(i,0,n) st[0][i] = a[i];
7     L(i,1,K) for (int j = 0; j + (1 << i) <= n; j++)
8       st[i][j] = min(st[i-1][j], st[i - 1][j + (1 << (i - 1))]);
9         // change op
10    }
11   int get(int l, int r) {
12     int bit = log2(r - l + 1);
13     return min(st[bit][l], st[bit][r - (1<<bit) + 1]); // change op
14   }

```

3.6 Segment tree

```

1 #define LC(v) (v<<1)
2 #define RC(v) ((v<<1)|1)
3 #define MD(L, R) (L+((R-L)>>1))
4 struct node { ll mx;ll cant; };
5 struct ST {
6   vec<node> st; vec<ll> lz; int n;
7   ST(int n = 1): st(4 * n + 10, {oo, oo}), lz(4 * n + 10, 0), n(n) {
8     build(1, 0, n - 1);}
9   node merge(node a, node b){
10     if (a.mx == oo) return b; if (b.mx == oo) return a;
11     if (a.mx == b.mx) return {a.mx, a.cant + b.cant};
12     return {max(a.mx, b.mx), a.mx > b.mx ? a.cant : b.cant};
13   }
14   void build(int v, int L, int R){

```

```

14     if (L == R){ st[v] = {0, 1}; return ;}
15     int m = MD(L, R);
16     build(LC(v), L, m); build(RC(v), m + 1, R);
17     st[v] = merge(st[LC(v)], st[RC(v)]);
18   }
19   void push(int v, int L, int R){
20     if (lz[v]){
21       if (L != R){
22         st[LC(v)].mx += lz[v]; // Apply to left
23         st[RC(v)].mx += lz[v]; // And right
24         lz[LC(v)] += lz[v];
25         lz[RC(v)] += lz[v];
26       }
27       lz[v] = 0;
28     }
29   }
30   void update(int v, int L, int R, int ql, int qr, ll w){
31     if (ql > R || qr < L) return;
32     push(v, L, R);
33     if (ql == L && qr == R){
34       st[v].mx += w; // Update actual node
35       lz[v] += w; // Add lazy
36       push(v, L, R); // Initial spread
37       return;
38     }
39     int m = MD(L, R);
40     update(LC(v), L, m, ql, min(qr, m), w);
41     update(RC(v), m + 1, R, max(m + 1, ql), qr, w);
42     st[v] = merge(st[LC(v)], st[RC(v)]);
43   }
44   node query(int v, int L, int R, int ql, int qr){
45     if (ql > R || qr < L) return {oo, oo};
46     push(v, L, R);
47     if (ql == L && qr == R) return st[v];
48     int m = MD(L, R);
49     return merge(query(LC(v), L, m, ql, min(m, qr)), query(RC(v), m
50       + 1, R, max(m + 1, ql), qr));
51   }
52   node query(int l, int r){return query(1, 0, n - 1, l, r);}
53   void update(int l, int r, ll w){update(1, 0, n - 1, l, r, w);}
54 };

```

3.7 Segment Tree Iterativo

```

1 struct STI {
2     vec<ll> st; int n, K;
3     STI(vec<ll> &a): n(SZ(a)), K(1) {
4         while(K<=n) K<<=1;
5         st.assign(2*K, 0); // 0 default
6         L(i,0,n) st[K+i] = a[i];
7         for (int i = K - 1; i > 0; i --) st[i] = st[i*2] + st[i*2+1];
8     void upd(int pos, ll w) {
9         pos += K; st[pos] += w;
10        while((pos>>1) > 0) st[pos] = st[pos * 2] + st[pos * 2 + 1];
11    query(int l, int r) { // [l, r]
12        ll res = 0;
13        for (l += K, r += K + 1; l < r; l>>=1, r>>=1){
14            if (l & 1) res += st[l++];
15            if (r & 1) res += st[--r];
16        }
17        return res;
18    }
19 };

```

3.8 Segment Tree Persistente

```

1 struct Vertex{Vertex * l, *r;int sum;};
2 const int MVertex = 6000000; // ~= N * logN * 2
3 Vertex pool[MVertex]; // the idea is to keep versions on vec<Vertex*>
4     roots; roots.pb(build(ST_L, ST_R));
5     int p_num = 0;          //
6     Vertex * init_leaf(int x) {
7         pool[p_num].sum = x;
8         pool[p_num].l = pool[p_num].r = NULL;
9         return &pool[p_num++];
10    }
11    Vertex * init_node(Vertex * l, Vertex * r) {
12        int sum = 0;
13        if (l) sum += l->sum;
14        if (r) sum += r->sum;
15        pool[p_num].sum = sum; pool[p_num].l = l; pool[p_num].r = r;
16        return &pool[p_num++];
17    }
18    Vertex * build(int L, int R){
19        if (L == R){return init_leaf(0);}
20        int m = MD(L, R); return init_node(build(L, m), build(m + 1, R));}

```

```

19 Vertex * update(Vertex * v, int L, int R, int pos, int w){
20     if (L == R) return init_leaf(v->sum + w);
21     int m = MD(L, R);
22     if (pos <= m) return init_node(update(v->l, L, m, pos, w), v->r);
23     return init_node(v->l, update(v->r, m + 1, R, pos, w));}
24 int query(Vertex * vl, Vertex * vr, int L, int R, int ql, int qr) {
25     if (!vl || !vr) return 0;
26     if (ql > R || qr < L) return 0;
27     if (ql == L && qr == R) {return vr->sum - vl->sum;}
28     int m = MD(L, R);
29     return query(vl->l, vr->l, L, m, ql, min(m, qr)) +
30             query(vl->r, vr->r, m + 1, R, max(m + 1, ql), qr);}

```

3.9 Policy Based

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 using namespace __gnu_pbds;
3 template<typename Key, typename Val=null_type>
4 using indexed_set = tree<Key, Val, less<Key>, rb_tree_tag,
5     tree_order_statistics_node_update>;
6 // indexed_set<char> s;
7 // char val = *s.find_by_order(0); // acceso por indice
8 // int idx = s.order_of_key('a'); // busca indice del valor
9 template<class Key, class Val=null_type> using htable=gp_hash_table<Key,
10 Val>;
11 // como unordered_map (o unordered_set si Val es vacio), pero sin metodo
12 count

```

3.10 SQRT Decomposition

```

1 struct SQRTDecomp {
2     vec<int> B, Bs, Bid; int n;// DEFINE_BLOCK_SIZE
3     SQRTDecomp(int n_): B(n_), Bid(n_), n_(n_) { // About sqrt N
4         Bs.assign((n + BLOCK_SIZE - 1)/BLOCK_SIZE, 0);
5         L(i,1,n) Bid[i] = Bid[i - 1] + (i % BLOCK_SIZE == 0);
6     } // useful if many updates not many queries, may be better than st
7     void upd(int ix, int w) { B[ix] += w; Bs[Bid[ix]] += w;} // O(1)
8     int query(int l, int r){ // O(BLOCK_SIZE)
9         int ans = 0;
10        for (int i = l; i < r;) {
11            if (i + BLOCK_SIZE > r || (i % BLOCK_SIZE) != 0) ans += B[i
12                ++];
13            else { ans += Bs[Bid[i]]; i += BLOCK_SIZE;}
14        }
15    }

```

```

14     return ans;
15 }
16 };

```

3.11 Chull Trick

```

1 typedef ll tc;
2 const tc is_query=-(1LL<<62); // special value for query
3 struct Line {
4     tc m,b;
5     mutable multiset<Line>::iterator it,end;
6     const Line* succ(multiset<Line>::iterator it) const {
7         return (++it==end? NULL : &*it);}
8     bool operator<(const Line& rhs) const {
9         if(rhs.b!=is_query) return m<rhs.m;
10        const Line *s=succ(it);
11        if(!s) return 0;
12        return b-s->b<(s->m-m)*rhs.m;
13    }
14 };
15 struct HullDynamic : public multiset<Line> { // for maximum
16     bool bad(iterator y){
17         iterator z=next(y);
18         if(y==begin()){
19             if(z==end())return false;
20             return y->m==z->m&&y->b<=z->b;
21         }
22         iterator x=prev(y);
23         if(z==end())return y->m==x->m&&y->b<=x->b;
24         return 1.0*(x->b-y->b)*(z->m-y->m)>=1.0*(y->b-z->b)*(y->m-x->m);
25     }
26     iterator next(iterator y){return ++y;}
27     iterator prev(iterator y){return --y;}
28     void add(tc m, tc b){
29         iterator y=insert((Line){m,b});
30         y->it=y;y->end=end();
31         if(bad(y)){erase(y);return;}
32         while(next(y)!=end()&&bad(next(y)))erase(next(y));
33         while(y!=begin()&&bad(prev(y)))erase(prev(y));
34     }
35     tc eval(tc x){
36         Line l=*lower_bound((Line){x,is_query});
37         return l.m*x+l.b;

```

```

38     }
39 };

```

4 Graph

4.1 Bellman Ford

```

1 struct Edge {int a, b, cost;};
2 vector<Edge> edges;
3 int solve(int s) // Source
4 {
5     vector<int> d(n, INF);
6     d[s] = 0;
7     for (int i = 0; i < n - 1; ++i)
8         for (Edge e : edges)
9             if (d[e.a] < INF)
10                 d[e.b] = min(d[e.b], d[e.a] + e.cost);
11 }

```

4.2 SCC

```

1 vec<int> dfs_num(N, -1), dfs_low(N, -1), in_stack(N);
2 int dfs_count = 0;
3 int numSCC = 0;
4 stack<int> st;
5 void dfs(int u){
6     dfs_low[u]=dfs_num[u]=dfs_count++;
7     st.push(u);
8     in_stack[u] = 1;
9     for(int v: G[u]) {
10         if (dfs_num[v] == -1) dfs(v);
11         if (in_stack[v]) dfs_low[u] = min(dfs_low[u], dfs_low[v]);
12     }
13     if (dfs_num[u] == dfs_low[u]){
14         numSCC++;
15         while(1){
16             int v = st.top(); st.pop();
17             in_stack[v] = 0;
18             if (u == v) break;
19         }
20     }
21 }

```

4.3 Bipartite Matching Hopcroft-Karp - With Konig

```

1 mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count())
  );
2 struct hopcroft_karp {
3     int n, m; // n is Left Partition Size, m is Right Partition Size
4     vec<vec<int>> g;
5     vec<int> dist, nxt, ma, mb;
6     hopcroft_karp(int n_, int m_) : n(n_), m(m_), g(n),
7         dist(n), nxt(n), ma(n, -1), mb(m, -1) {}
8     void add(int a, int b) { g[a].pb(b); }
9     bool dfs(int i) {
10         for (int &id = nxt[i]; id < g[i].size(); id++) {
11             int j = g[i][id];
12             if (mb[j] == -1 or (dist[mb[j]] == dist[i]+1 and dfs(mb[j]))) {
13                 ma[i] = j, mb[j] = i;
14                 return true;
15             }
16         }
17         return false;
18     }
19     bool bfs() {
20         for (int i = 0; i < n; i++) dist[i] = n;
21         queue<int> q;
22         for (int i = 0; i < n; i++) if (ma[i] == -1) {
23             dist[i] = 0;
24             q.push(i);
25         }
26         bool rep = 0;
27         while (q.size()) {
28             int i = q.front(); q.pop();
29             for (int j : g[i]) {
30                 if (mb[j] == -1) rep = 1;
31                 else if (dist[mb[j]] > dist[i] + 1) {
32                     dist[mb[j]] = dist[i] + 1;
33                     q.push(mb[j]);
34                 }
35             }
36         }
37         return rep;
38     }
39     int matching() {
40         int ret = 0;

```

```

41         for (auto& i : g) shuffle(ALL(i), rng);
42         while (bfs()) {
43             for (int i = 0; i < n; i++) nxt[i] = 0;
44             for (int i = 0; i < n; i++)
45                 if (ma[i] == -1 and dfs(i)) ret++;
46         }
47         return ret;
48     }
49     vec<int> cover[2]; // if cover[i][j] = 1 -> node i, j is part of cover
50     int konig() {
51         cover[0].assign(n,1); // n left size
52         cover[1].assign(m,0); // m right size
53         auto go = [&](auto&& me, int u) -> void {
54             cover[0][u] = false;
55             for (auto v : g[u]) if (!cover[1][v]) {
56                 cover[1][v] = true;
57                 me(me,mb[v]);
58             }
59         };
59         L(u,0,n) if (ma[u] < 0) go(go,u);
60         return size;
61     }
62 }
63 
```

4.4 Hungarian

```

1 using vi = vec<int>;
2 using vd = vec<ld>;
3 const ld INF = 1e100;    // Para max asignacion, INF = 0, y negar costos
4 bool zero(ld x) {return fabs(x) < 1e-9;} // Para int/ll: return x==0;
5 vec<pii> ans; // Guarda las aristas usadas en el matching: [0..n)x[0..m]
6 struct Hungarian{
7     int n; vec<vd> cs; vi vL, vR;
8     Hungarian(int N, int M) : n(max(N,M)), cs(n,vd(n)), vL(n), vR(n){
9         L(x, 0, N) L(y, 0, M) cs[x][y] = INF;
10    }
11    void set(int x, int y, ld c) { cs[x][y] = c; }
12    ld assign(){
13        int mat = 0; vd ds(n), u(n), v(n); vi dad(n), sn(n);
14        L(i, 0, n) u[i] = *min_element(ALL(cs[i]));
15        L(j, 0, n){
16            v[j] = cs[0][j]-u[0];
17            L(i, 1, n) v[j] = min(v[j], cs[i][j] - u[i]);

```

```

18 }
19 vL = vR = vi(n, -1);
20 L(i, 0, n) L(j, 0, n) if(vR[j] == -1 and zero(cs[i][j] - u[i] - v[j]))
21     ){
22         vL[i] = j; vR[j] = i; mat++; break;
23     }
24 for( mat < n; mat ++){
25     int s = 0, j = 0, i;
26     while(vL[s] != -1) s++;
27     fill(ALL(dad), -1); fill(ALL(sn), 0);
28     L(k, 0, n) ds[k] = cs[s][k]-u[s]-v[k];
29     while(true){
30         j = -1;
31         L(k, 0, n) if(!sn[k] and (j == -1 or ds[k] < ds[j])) j = k;
32         sn[j] = 1; i = vR[j];
33         if(i == -1) break;
34         L(k, 0, n) if(!sn[k]){
35             auto new_ds = ds[j] + cs[i][k] - u[i]-v[k];
36             if(ds[k] > new_ds) ds[k]=new_ds, dad[k]=j;
37         }
38     }
39     L(k, 0, n) if(k!=j and sn[k]){
40         auto w = ds[k]-ds[j]; v[k] += w, u[vR[k]] -= w;
41     }
42     u[s] += ds[j];
43     while(dad[j] >= 0){ int d = dad[j]; vR[j] = vR[d]; vL[vR[j]] = j;
44         j = d; }
45     vR[j] = s; vL[s] = j;
46 }
47 ld value = 0; L(i, 0, n) value += cs[i][vL[i]], ans.pb({i, vL[i]});
48 return value;
}

```

4.5 Flow - Dinics

```

1 struct Dinic {
2     bool scaling = false; // com scaling -> O(nm log(MAXCAP)),
3     int lim;           // com constante alta
4     struct edge {
5         int to, cap, rev, flow;
6         bool res;
7         edge(int to_, int cap_, int rev_, bool res_)

```

```

8         : to(to_), cap(cap_), rev(rev_), flow(0), res(res_) {}
9     };
10    vec<vec<edge>> g;
11    vec<int> lev, beg;
12    ll F;
13    Dinic(int n) : g(n), F(0) {}
14    void add(int a, int b, int c) {
15        g[a].emplace_back(b, c, g[b].size(), false);
16        g[b].emplace_back(a, 0, g[a].size()-1, true);
17    }
18    bool bfs(int s, int t) {
19        lev = vector<int>(g.size(), -1); lev[s] = 0;
20        beg = vector<int>(g.size(), 0);
21        queue<int> q; q.push(s);
22        while (q.size()) {
23            int u = q.front(); q.pop();
24            for (auto& i : g[u]) {
25                if (lev[i.to] != -1 or (i.flow == i.cap)) continue;
26                if (scaling and i.cap - i.flow < lim) continue;
27                lev[i.to] = lev[u] + 1;
28                q.push(i.to);
29            }
30        }
31        return lev[t] != -1;
32    }
33    int dfs(int v, int s, int f = oo) {
34        if (!f or v == s) return f;
35        for (int& i = beg[v]; i < g[v].size(); i++) {
36            auto& e = g[v][i];
37            if (lev[e.to] != lev[v] + 1) continue;
38            int foi = dfs(e.to, s, min(f, e.cap - e.flow));
39            if (!foi) continue;
40            e.flow += foi, g[e.to][e.rev].flow -= foi;
41            return foi;
42        }
43        return 0;
44    }
45    ll max_flow(int s, int t) {
46        for (lim = scaling ? (1<<30) : 1; lim; lim /= 2)
47            while (bfs(s, t)) while (int ff = dfs(s, t)) F += ff;
48        return F;
49    }
50 };

```

```

51 vec<pair<int, int>> get_cut(Dinic& g, int s, int t) {
52     g.max_flow(s, t);
53     vec<pair<int, int>> cut;
54     vec<int> vis(g.g.size(), 0), st = {s};
55     vis[s] = 1;
56     while (st.size()) {
57         int u = st.back(); st.pop_back();
58         for (auto e : g.g[u]) if (!vis[e.to] and e.flow < e.cap)
59             vis[e.to] = 1, st.push_back(e.to);
60     }
61     for (int i = 0; i < g.g.size(); i++) for (auto e : g.g[i])
62         if (vis[i] and !vis[e.to] and !e.res) cut.emplace_back(i, e.to);
63     return cut;
64 }
```

4.6 Flow - MinCostMaxFlow

```

1 // O(nm + f * m log n)
2 // const ll oo = (ll)1e18;
3 template<typename T> struct mcmf {
4     struct edge {
5         int to, rev, flow, cap; // para, id da reversa, fluxo, capacidade
6         bool res; // se eh reversa
7         T cost; // custo da unidade de fluxo
8         edge() : to(0), rev(0), flow(0), cap(0), cost(0), res(false) {}
9         edge(int to_, int rev_, int flow_, int cap_, T cost_, bool res_)
10            : to(to_), rev(rev_), flow(flow_), cap(cap_), res(res_), cost(cost_) {}
11    };
12    vec<vec<edge>> g;
13    vec<int> par_idx, par;
14    T inf;
15    vec<T> dist;
16    mcmf(int n) : g(n), par_idx(n), par(n), inf(numeric_limits<T>::max() / 3) {}
17    void add(int u, int v, int w, T cost) { // de u pra v com cap w e
18        edge a = edge(v, g[v].size(), 0, w, cost, false);
19        edge b = edge(u, g[u].size(), 0, 0, -cost, true);
20        g[u].push_back(a);
21        g[v].push_back(b);
22    }
23    vec<T> spfa(int s) { // nao precisa se nao tiver custo negativo
```

```

24     deque<int> q;
25     vec<bool> is_inside(g.size(), 0);
26     dist = vec<T>(g.size(), inf);
27     dist[s] = 0;
28     q.push_back(s);
29     is_inside[s] = true;
30     while (!q.empty()) {
31         int v = q.front();
32         q.pop_front();
33         is_inside[v] = false;
34         for (int i = 0; i < g[v].size(); i++) {
35             auto [to, rev, flow, cap, res, cost] = g[v][i];
36             if (flow < cap and dist[v] + cost < dist[to]) {
37                 dist[to] = dist[v] + cost;
38
39                 if (is_inside[to]) continue;
40                 if (!q.empty() and dist[to] > dist[q.front()]) q.push_back(to)
41                     ;
42                 else q.push_front(to);
43                 is_inside[to] = true;
44             }
45         }
46     }
47     return dist;
48 }
49 bool dijkstra(int s, int t, vec<T>& pot) {
50     priority_queue<pair<T, int>, vec<pair<T, int>>, greater<>> q;
51     dist = vec<T>(g.size(), inf);
52     dist[s] = 0;
53     q.emplace(0, s);
54     while (q.size()) {
55         auto [d, v] = q.top();
56         q.pop();
57         if (dist[v] < d) continue;
58         for (int i = 0; i < g[v].size(); i++) {
59             auto [to, rev, flow, cap, res, cost] = g[v][i];
60             cost += pot[v] - pot[to];
61             if (flow < cap and dist[v] + cost < dist[to]) {
62                 dist[to] = dist[v] + cost;
63                 q.emplace(dist[to], to);
64                 par_idx[to] = i, par[to] = v;
65             }
66     }
```

```

66     }
67     return dist[t] < inf;
68 }
69 pair<int, T> min_cost_flow(int s, int t, int flow = (int)1e9) {
70     vec<T> pot(g.size(), 0);
71     pot = spfa(s); // mudar algoritmo de caminho minimo aqui
72     int f = 0;
73     T ret = 0;
74     while (f < flow and dijkstra(s, t, pot)) {
75         for (int i = 0; i < g.size(); i++)
76             if (dist[i] < inf) pot[i] += dist[i];
77         int mn_flow = flow - f, u = t;
78         while (u != s){
79             mn_flow = min(mn_flow,
80                           g[par[u]][par_idx[u]].cap - g[par[u]][par_idx[u]].flow);
81             u = par[u];
82         }
83         ret += pot[t] * mn_flow;
84         u = t;
85         while (u != s) {
86             g[par[u]][par_idx[u]].flow += mn_flow;
87             g[u][g[par[u]][par_idx[u]].rev].flow -= mn_flow;
88             u = par[u];
89         }
90         f += mn_flow;
91     }
92     return make_pair(f, ret);
93 }
94 // Opcional: retorna as arestas originais por onde passa flow = cap
95 vec<pair<int,int>> recover() {
96     vec<pair<int,int>> used;
97     for (int i = 0; i < g.size(); i++) for (edge e : g[i])
98         if(e.flow == e.cap && !e.res) used.push_back({i, e.to});
99     return used;
100 }
101 };

```

4.7 2 Sat

```

1 struct TwoSat {
2     int n, v_n;
3     vec<bool> vis, assign;
4     vec<int> order, comp;

```

```

5     vec<vec<int>> g, g_t;
6     TwoSat(int n_): n(n_), v_n(2 * n_), vis(v_n) , assign(n_), comp(v_n
7         , - 1), g(v_n), g_t(v_n) {
8         order.reserve(v_n);
9     }
10    void add_disj(int a, bool na, int b, bool nb) { // negated_a,
11        negated_b
12        a = 2 * a ^ na;
13        b = 2 * b ^ nb;
14        int neg_a = a ^ 1;
15        int neg_b = b ^ 1;
16        g[neg_a].pb(b);
17        g[neg_b].pb(a);
18        g_t[a].pb(neg_b);
19        g_t[b].pb(neg_a);
20    }
21    void dfs1(int u){
22        vis[u] = 1;
23        for (int v: g[u]) if (!vis[v]) dfs1(v);
24        order.pb(u);
25    }
26    void dfs2(int u, int cc) {
27        comp[u] = cc;
28        for (int v: g_t[u]) if (comp[v] == -1) dfs2(v, cc);
29    }
30    bool solve() {
31        order.clear();
32        vis.assign(v_n, 0);
33        L(i, 0, v_n) if (!vis[i]) dfs1(i);
34        comp.assign(v_n, - 1);
35        int cc = 0;
36        L(i, 0, v_n) {
37            int v = order[v_n - 1 - i];
38            if (comp[v] == -1) dfs2(v, cc++);
39        }
40        assign.assign(n, false);
41        for (int i = 0; i < v_n; i += 2) {
42            if (comp[i] == comp[i+1]) return false;
43            assign[i / 2] = comp[i] > comp[i + 1];
44        }
45    }
46 }

```

4.8 Euler Tour

```

1 // Directed version (uncomment commented code for undirected)
2 struct edge {
3     int y;
4     // list<edge>::iterator rev;
5     edge(int y):y(y){}
6 };
7 list<edge> g[N];
8 void add_edge(int a, int b){
9     g[a].push_front(edge(b)); //auto ia=g[a].begin();
10    // g[b].push_front(edge(a));auto ib=g[b].begin();
11    // ia->rev=ib;ib->rev=ia;
12 }
13 vec<int> p;
14 void go(int x){
15     while(g[x].size()){
16         int y=g[x].front().y;
17         //g[y].erase(g[x].front().rev);
18         g[x].pop_front();
19         go(y);
20     }
21     p.push_back(x);
22 }
23 vec<int> get_path(int x){ // get a path that begins in x
24 // check that a path exists from x before calling to get_path!
25     p.clear();go(x);reverse(p.begin(),p.end());
26     return p;
27 }
```

5 Trees

5.1 Heavy Light Decomposition

```

1 int ans[N], par[N], depth[N], head[N], pos[N];
2 vec<int> heavy(N, -1);
3 int t = 0;
4 vec<int> g[N];
5 int dfs(int u) {
6     int size = 1;
7     int max_size = 0;
8     for (int v: g[u]) if (v != par[u]) {
9         par[v] = u;
10        size += dfs(v);
11        if (size > max_size) {
12            max_size = size;
13            head[u] = v;
14        }
15    }
16    return size;
17 }
18 void decompose(int u, int h){
19     head[u] = h;
20     pos[u] = t++;
21     if (heavy[u] != -1) decompose(heavy[u], h);
22     for (int v: G[u]) if (v != par[u] && v != heavy[u]) {
23         decompose(v, v);
24     }
25 }
26 int query(int a, int b) {
27     int resp = -1;
28     for (; head[a] != head[b]; b = par[head[b]]){ // Subi todo el heavy
29         path y a su padre // Next
30         if (depth[head[a]] > depth[head[b]]) swap(a, b);
31         resp = max(resp, st.query(pos[head[b]], pos[b])); // pos[head[b]
32             ] < pos[b]
33     }
34     if (depth[a] > depth[b]) swap(a, b); // Una vez misma path(head)
35         entonces es una query [a,b]
36     resp = max(resp, st.query(pos[a], pos[b]));
37     return resp;
38 }
39 dfs(root);
40 decompose(root, root);
```

5.2 Centroid

```

1 int sz[N];
2 bool removed[N];
3 int getSize(int u, int p){
4     sz[u] = 1;
5     for(int v: G[u]) if (v != p && !removed[v]){
6         sz[u] += getSize(v, u);
7     }
8 }
```

```
8     return sz[u];
9 }
10 int centroid(int u, int p, int tz){
11     for (int v: g[u])
12         if (v != p && !removed[v] && sz[v] * 2 > tz) return centroid(v,
13             u, tz);
14     return u;
15 }
16 int build(int u){
17     int c = centroid(u, -1, getSize(u, -1));
18     removed[c] = 1;
19     for (int v: G[c]) if (!removed[v]) { build(v); }
20     return c;
21 }
```

5.3 LCA - Binary exponentiation

```

1 vec<int> g[N];
2 int K; // K should be  $(1 \ll K) > n$ 
3 int jump[20][N];
4 int depth[N];
5
6 void dfs(int u, int p){
7     for (int v: g[u]) if (v != p) {
8         jump[0][v] = u;
9         L(i, 1, K + 1) {
10            jump[i][v] = -1;
11            if (jump[i - 1][v] != -1) {
12                jump[i][v] = jump[i - 1][jump[i - 1][v]];
13            }
14        }
15        depth[v] = depth[u] + 1;
16        dfs(v, u);
17    }
18 }
19
20 int LCA(int u, int v){
21     if (depth[u] < depth[v]) swap(u, v); // Make u the deepest
22     for (int i = K; i >= 0; i --){ // make them same depth
23         if (jump[i][u] != -1 && depth[jump[i][u]] >= depth[v]){
24             u = jump[i][u];
25         }
26     }

```

```
27     if (u == v) return u; // u is parent of v
28     for (int i = K; i >= 0; i --){
29         if (jump[i][u] != jump[i][v] && jump[i][u] != -1 && jump[i][v]
30             != -1){
31             u = jump[i][u];
32             v = jump[i][v];
33         }
34     }
35 }
```

5.4 LCA - Const Time

```

30     int su = in[u];
31     int sv = in[v];
32     if (sv < su) swap(sv, su);
33     int bit = log2(sv - su + 1);
34     return Min(st[bit][su], st[bit][sv - (1<<bit) + 1]);
35   }
36 }

```

6 Dynamic Programming

6.1 Knapsack

```

1 int knapsack(vector<int>& values, vector<int>& weights, int W) {
2     int n = values.size();
3     vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));
4
5     for(int i = 1; i <= n; i++) {
6         for(int w = 0; w <= W; w++) {
7             if(weights[i-1] <= w) {
8                 dp[i][w] = max(dp[i-1][w],
9                                 dp[i-1][w-weights[i-1]] + values[i-1]);
10            } else {
11                dp[i][w] = dp[i-1][w];
12            }
13        }
14    }
15    return dp[n][W];
16 }

```

6.2 LIS

```

1 vector<int> getLIS(vector<int>& arr) {
2     int n = arr.size();
3     vector<int> dp(n + 1, INT_MAX); // dp[i] = smallest value that ends
4                                     // an LIS of length i
5     vector<int> len(n);           // Length of LIS ending at each
6                                     // position
7     dp[0] = INT_MIN;
8     for(int i = 0; i < n; i++) {
9         int j = upper_bound(dp.begin(), dp.end(), arr[i]) - dp.begin();
10        dp[j] = arr[i];
11        len[i] = j;
12    }
13 }

```

```

11 // Find maxLen and reconstruct sequence
12 int maxLen = 0;
13 for(int i = n-1; i >= 0; i--) maxLen = max(maxLen, len[i]);
14 vector<int> lis;
15 for(int i = n-1, currLen = maxLen; i >= 0; i--) {
16     if(len[i] == currLen) {
17         lis.push_back(arr[i]);
18         currLen--;
19     }
20 }
21 reverse(lis.begin(), lis.end());
22 return lis;
23 }

```

6.3 Edit Distance

```

1 int editDistance(string& s1, string& s2) {
2     int n = s1.length(), m = s2.length();
3     vector<vector<int>> dp(n + 1, vector<int>(m + 1));
4     for(int i = 0; i <= n; i++) dp[i][0] = i;
5     for(int j = 0; j <= m; j++) dp[0][j] = j;
6     for(int i = 1; i <= n; i++) {
7         for(int j = 1; j <= m; j++) {
8             if(s1[i-1] == s2[j-1]) {
9                 dp[i][j] = dp[i-1][j-1];
10            } else {
11                dp[i][j] = 1 + min({dp[i-1][j], // deletion
12                                dp[i][j-1], // insertion
13                                dp[i-1][j-1]}); // replacement
14            }
15        }
16    }
17    return dp[n][m];
18 }

```

6.4 Kadane

```

1 pair<int, pair<int,int>> kadane(vector<int>& arr) {
2     int maxSoFar = arr[0], maxEndingHere = arr[0];
3     int start = 0, end = 0, s = 0;
4
5     for(int i = 1; i < arr.size(); i++) {
6         if(maxEndingHere + arr[i] < arr[i]) {
7             maxEndingHere = arr[i];

```

```

8     s = i;
9 } else {
10    maxEndingHere += arr[i];
11 }
12
13 if(maxEndingHere > maxSoFar) {
14    maxSoFar = maxEndingHere;
15    start = s;
16    end = i;
17 }
18
19 return {maxSoFar, {start, end}}; // max, l, r
20 }

```

7 Strings

7.1 Hashing

```

1 static constexpr ll ms[] = {1'000'000'007, 1'000'000'403};
2 static constexpr ll b = 500'000'000;
3 struct StrHash { // Hash polinomial con exponentes decrecientes.
4     vec<ll> hs[2], bs[2];
5     StrHash(string const& s) {
6         int n = SZ(s);
7         L(k, 0, 2) {
8             hs[k].resize(n+1), bs[k].resize(n+1, 1);
9             L(i, 0, n) {
10                 hs[k][i+1] = (hs[k][i] * b + s[i]) % ms[k];
11                 bs[k][i+1] = bs[k][i] * b % ms[k];
12             }
13         }
14     }
15     ll get(int idx, int len) const { // Hashes en 's[idx, idx+len)'.
16         ll h[2];
17         L(k, 0, 2) {
18             h[k] = hs[k][idx+len] - hs[k][idx] * bs[k][len] % ms[k];
19             if (h[k] < 0) h[k] += ms[k];
20         }
21         return (h[0] << 32) | h[1];
22     }
23 };
24
25 pll union_hash(vec<pll> hs, vec<ll> lens){ //use arrays makes it slower

```

```

26     ll len = 0;
27     for(int i = hs.size()-1; i > 0; i--){
28         len += lens[i];
29         pll& [l1, l2] = hs[i];
30         pll& [r1, r2] = hs[i-1];
31         l1 = ((l1 * binpow(b, len, ms[0])) % ms[0] + r1) % ms[0];
32         l2 = ((l2 * binpow(b, len, ms[1])) % ms[1] + r2) % ms[1];
33     }
34
35     return hs[0];
36 }

```

7.2 Trie

```

1 struct Trie {
2     map<char, int> ch;
3     bool eee;
4     Trie(): eee(0) {}
5 };
6 vec<Trie> t;
7 void initTrie(){t.clear(); t.pb(Trie());}
8 void insert(string &word) {
9     int v = 0;
10    for(char c : word) {
11        if(!t[v].ch[c]) {
12            t[v].ch[c] = SZ(t);
13            t.pb(Trie());
14        }
15        v = t[v].ch[c];
16    }
17    t[v].eee = 1;
18 }

```

7.3 KMP

```

1 vec<int> kmp(string pat, string sec){ //geeks4geeks implementation with
2     some changes
3     int m = pat.length();
4     int n = sec.length();
5     cout << m << " " << n << endl;
6
7     vec<int> lps = getLps(pat);
8     vec<int> res;

```

```

9  int i = 0;
10 int j = 0;
11
12 while((n - i) >= (m - j)){
13     if(pat[j] == sec[i]){
14         i++;
15         j++;
16     }
17     if(j == m){
18         res.push_back(i - j);
19         j = lps[j - 1];
20     }
21     else{
22         if(i < n && pat[j] != sec[i]){
23             if(j != 0) j = lps[j - 1];
24             else i = i + 1;
25         }
26     }
27 }
28
29 return res;
30 }
```

7.4 LPS

```

1 vec<int> getLps(string pat){ //geek4geeks implementatio with some
2     changes
3     vec<int> lps(pat.length(), 0);
4     int len = 0;
5     int i = 1;
6     lps[0] = 0;
7     while(i < pat.length()){
8         if(pat[i] == pat[len]){
9             len++;
10            lps[i] = len;
11            i++;
12        }
13        else //pat[i] != pat[len]
14        {
15            lps[i] = 0;
16            i++;
17        }
18    }
19 }
```

```

18     return lps;
19 }
```

7.5 Z-FUNCTION

```

1 template<class Char=char>vec<int> zfun(const basic_string<Char>& w) {
2     int n = SZ(w), l = 0, r = 0; vec<int> z(n);
3     z[0] = w.length();
4     L(i, 1, n) {
5         if (i <= r) {z[i] = min(r - i + 1, z[i - 1]);}
6         while (i + z[i] < n && w[z[i]] == w[i + z[i]]) {++z[i];}
7         if (i + z[i] - 1 > r) {l = i, r = i + z[i] - 1;}
8     }
9     return z;
10 }
```

7.6 Manacher

```

1 struct Manacher {
2     vec<int> p;
3     Manacher(string const& s) {
4         int n = SZ(s), m = 2*n+1, l = -1, r = 1;
5         vec<char> t(m); L(i, 0, n) t[2*i+1] = s[i];
6         p.resize(m); L(i, 1, m) {
7             if (i < r) p[i] = min(r-i, p[l+r-i]);
8             while (p[i] <= i && i < m-p[i] && t[i-p[i]] == t[i+p[i]]) ++p[i];
9             if (i+p[i] > r) l = i-p[i], r = i+p[i];
10        }
11    } // Retorna palindromos de la forma {comienzo, largo}.
12    pii at(int i) const {int k = p[i]-1; return pair{i/2-k/2, k};}
13    pii odd(int i) const {return at(2*i+1);} // Mayor centrado en s[i].
14    pii even(int i) const {return at(2*i);} // Mayor centrado en s[i-1,i].
```

7.7 Aho-Corasick

```

1 bool vis[N], r[N];
2 struct ACVertex {
3     map<char,int> next,go;
4     int p,link;
5     char pch;
6     vec<int> leaf;
7     ACACvertex(int p=-1, char pch=-1):p(p),pch(pch),link(-1){}
8 };
```

```

9  vec<ACvertex> t;
10 void aho_init(){ //do not forget!!
11   t.clear();t.pb(ACvertex());
12 }
13 void add_string(string &s, int id){
14   int v=0;
15   for(char c:s){
16     if(!t[v].next.count(c)){
17       t[v].next[c]=t.size();
18       t.pb(ACvertex(v,c));
19     }
20     v=t[v].next[c];
21   }
22   t[v].leaf.pb(id);
23 }
24 int go(int v, char c);
25 int get_link(int v){ // Failure link
26   if(t[v].link<0)
27     if(!v||!t[v].p)t[v].link=0;
28   else t[v].link=go(get_link(t[v].p),t[v].pch);
29   return t[v].link;
30 }
31 int go(int v, char c){ // state = go(state, ch) this state is ACvertex
32   id
33   if(!t[v].go.count(c))
34     if(t[v].next.count(c))t[v].go[c]=t[v].next[c];
35   else t[v].go[c]=v==0?0:go(get_link(v),c);
36   return t[v].go[c];
37 }
38 void proc(int x){
39   if (x == - 1 || vis[x]) return;
40   vis[x] = 1;
41   L(i,0,SZ(t[x].leaf)) r[t[x].leaf[i]] = 1;
42   proc(get_link(x));
}

```

7.8 Suffix-Array

```

1 #define RB(x) ((x) < n ? r[x] : 0)
2 void csort(vec<int>& sa, vec<int>& r, int k) {
3   int n = SZ(sa);
4   vec<int> f(max(255, n)), t(n);
5   L(i,0, n) ++f[RB(i+k)];

```

```

6   int sum = 0;
7   L(i,0, max(255, n)) f[i] = (sum += f[i]) - f[i];
8   L(i,0, n) t[f[RB(sa[i]+k)]++] = sa[i];
9   sa = t;
10 }
11 vec<int> compute_sa(string& s){ // O(n*log2(n))
12   int n = SZ(s) + 1, rank;
13   vec<int> sa(n), r(n), t(n);
14   iota(all(sa), 0);
15   L(i,0, n) r[i] = s[i];
16   for (int k = 1; k < n; k *= 2) {
17     csort(sa, r, k), csort(sa, r, 0);
18     t[sa[0]] = rank = 0;
19     L(i, 1, n) {
20       if(r[sa[i]] != r[sa[i-1]] || RB(sa[i]+k) != RB(sa[i-1]+k)) ++rank;
21       t[sa[i]] = rank;
22     }
23     r = t;
24     if (r[sa[n-1]] == n-1) break;
25   }
26   return sa; // sa[i] = i-th suffix of s in lexicographical order
27 }
28 vec<int> compute_lcp(string& s, vec<int>& sa){
29   int n = SZ(s) + 1, K = 0;
30   vec<int> lcp(n), plcp(n), phi(n);
31   phi[sa[0]] = -1;
32   L(i, 1, n) phi[sa[i]] = sa[i-1];
33   L(i,0,n) {
34     if (phi[i] < 0) { plcp[i] = 0; continue; }
35     while(s[i+K] == s[phi[i]+K]) ++K;
36     plcp[i] = K;
37     K = max(K - 1, 0);
38   }
39   L(i,0, n) lcp[i] = plcp[sa[i]];
40   return lcp; // lcp[i] = longest common prefix between sa[i-1] and sa[i]
41 }

```

8 Math

8.1 Euclidean Extended

```

1 ll extendedGCD(ll a, ll b, ll &x, ll &y) {

```

```

2     if (b == 0) {
3         x = 1;
4         y = 0;
5         return a;
6     }
7     ll x1, y1;
8     ll gcd = extendedGCD(b, a % b, x1, y1);
9     x = y1;
10    y = x1 - (a / b) * y1;
11    return gcd;
12 }

13
14 bool findSolutionWithConstraints(ll a, ll b, ll c, ll x_min, ll y_min,
15     ll &x, ll &y) {
16     ll g = extendedGCD(a, b, x, y);
17
18     if (c % g != 0) return false;
19
20     x *= c / g;
21     y *= c / g;
22
23     // Ajustamos las variables a/g y b/g para mover las soluciones
24     a /= g;
25     b /= g;
26
27     if (x < x_min) {
28         ll k = (x_min - x + b - 1) / b; // Redondeo hacia arriba
29         x += k * b;
30         y -= k * a;
31     } else if (x > x_min) {
32         ll k = (x - x_min) / b;
33         x -= k * b;
34         y += k * a;
35     }
36
37     if (y < y_min) {
38         ll k = (y_min - y + a - 1) / a; // Redondeo hacia arriba
39         x += k * b;
40         y -= k * a;
41     } else if (y > y_min) {
42         ll k = (y - y_min) / a;
43         x -= k * b;
44         y += k * a;
45     }
46 }

```

```

44     }
45
46     return x >= x_min && y >= y_min;
47 }

```

8.2 Euler Totient

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4
5
6 vector<ll> compute_totients(ll n) {
7     vector<ll> phi(n + 1);
8     for (ll i = 0; i <= n; i++) {
9         phi[i] = i;
10    }
11
12    for (ll i = 2; i <= n; i++) {
13        if (phi[i] == i) { // i es primo
14            for (ll j = i; j <= n; j += i) {
15                phi[j] = phi[j] * (i - 1) / i;
16            }
17        }
18    }
19
20    return phi;
21 }

```

8.3 Josephus

```

1 #include <iostream>
2 using namespace std;
3
4 typedef long long ll;
5
6 ll josephus_iterative(ll n, ll k) {
7     ll result = 0;
8     for (ll i = 2; i <= n; ++i) {
9         result = (result + k) % i;
10    }
11
12    return result;
13 }

```

```

14     }
15     ll josephus_recursive(ll n, ll k) {
16
17         if (n == 1)
18             return 0;
19
20         return (josephus_recursive(n - 1, k) + k) % n;
21     }
22
23
24     ll josephus_power_of_2(ll n) {
25
26         ll power = 1;
27         while (power <= n) {
28             power <= 1;
29         }
30         power >= 1;
31
32         return 2 * (n - power);
33     }
34 }
```

```

19         }
20     }
21
22     return mu;
23 }
24
25
26     ll mobius(ll x) {
27         ll count = 0;
28         for (ll i = 2; i * i <= x; i++) {
29             if (x % (i * i) == 0)
30                 return 0;
31             if (x % i == 0) {
32                 count++;
33                 x /= i;
34             }
35         }
36
37         if (x > 1) count++;
38
39         return (count % 2 == 0) ? 1 : -1;
40 }
```

8.4 Mobius

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4
5
6 vector<ll> compute_mobius(ll n) {
7     vector<ll> mu(n + 1, 1);
8     vector<bool> is_prime(n + 1, true);
9
10    for (ll i = 2; i <= n; i++) {
11        if (is_prime[i]) { // i es un primo
12            for (ll j = i; j <= n; j += i) {
13                mu[j] *= -1; // Multiplicamos por -1 para cada primo
14                is_prime[j] = false;
15            }
16            for (ll j = i * i; j <= n; j += i * i) {
17                mu[j] = 0; // Si tiene un cuadrado de un primo, se pone
18                en 0
19            }
20        }
21    }
22 }
```

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using cd = complex<double>;
4 typedef long long ll;
5 const ll mod = 998244353;
6 const ll root = 31;
7 const ll root_1 = inverse(root, mod);
8 const ll root_pw = 1 << 23;
9
10 ll inverse(ll a, ll m) {
11     ll res = 1, exp = m - 2;
12     while (exp) {
13         if (exp % 2 == 1) res = (1LL * res * a) % m;
14         a = (1LL * a * a) % m;
15         exp /= 2;
16     }
17     return res;
18 }
```

8.5 NTT

```

19 void ntt(vector<ll> & a, bool invert) {
20     int n = a.size();
21
22     for (int i = 1, j = 0; i < n; i++) {
23         int bit = n >> 1;
24         for (; j & bit; bit >>= 1)
25             j ^= bit;
26         j ^= bit;
27
28         if (i < j)
29             swap(a[i], a[j]);
30     }
31
32
33     for (int len = 2; len <= n; len <= 1) {
34         int wlen = invert ? root_1 : root;
35         for (int i = len; i < root_pw; i <= 1)
36             wlen = (int)(1LL * wlen * wlen % mod);
37
38         for (int i = 0; i < n; i += len) {
39             int w = 1;
40             for (int j = 0; j < len / 2; j++) {
41                 int u = a[i+j], v = (int)(1LL * a[i+j+len/2] * w % mod);
42                 a[i+j] = u + v < mod ? u + v : u + v - mod;
43                 a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
44                 w = (int)(1LL * w * wlen % mod);
45             }
46         }
47
48         if (invert) {
49             int n_1 = inverse(n, mod);
50             for (auto & x : a)
51                 x = (int)(1LL * x * n_1 % mod);
52         }
53     }
54
55     vector<ll> multiply(vector<ll> const &a, vector<ll> const &b) {
56         vector<ll> fa(a.begin(), a.end()), fb(b.begin(), b.end());
57         ll n = 1;
58         while (n < a.size() + b.size())
59             n <= 1;
60         fa.resize(n);
61
62     }

```

```

62     fb.resize(n);
63
64     ntt(fa, false);
65     ntt(fb, false);
66     for (ll i = 0; i < n; i++)
67         fa[i] = (fa[i] * fb[i]) % mod;
68     ntt(fa, true);
69
70     vector<ll> result(n);
71     for (ll i = 0; i < n; i++)
72         result[i] = fa[i];
73     return result;
74 }

```

8.6 FFT

```

1  typedef long long ll;
2  typedef complex<double> C;
3  typedef vector<double> vd;
4  typedef vector<ll> vll;
5  const double PI = acos(-1);
6
7  void fft(vector<C>& a) {
8      int n = a.size(), L = 31 - __builtin_clz(n);
9      static vector<C> R(2, 1);
10     static vector<C> rt(2, 1);
11     for (static int k = 2; k < n; k *= 2) {
12         R.resize(n); rt.resize(n);
13         auto x = polar(1.0, PI / k);
14         for (int i = k; i < 2 * k; i++)
15             rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
16     }
17     vector<int> rev(n);
18     for (int i = 0; i < n; i++) rev[i] = (rev[i / 2] | (i & 1) << L) /
19         2;
20     for (int i = 0; i < n; i++) if (i < rev[i]) swap(a[i], a[rev[i]]);
21     for (int k = 1; k < n; k *= 2)
22         for (int i = 0; i < n; i += 2 * k) for (int j = 0; j < k; j++) {
23             auto x = (double*)&rt[j + k], y = (double*)&a[i + j + k];
24             C z(x[0] * y[0] - x[1] * y[1], x[0] * y[1] + x[1] * y[0]);
25             a[i + j + k] = a[i + j] - z;
26             a[i + j] += z;
27         }

```

```

27 }
28
29 vll multiply(const vll& a, const vll& b) {
30     if (a.empty() || b.empty()) return {};
31     vd fa(a.begin(), a.end()), fb(b.begin(), b.end());
32     int L = 32 - __builtin_clz(fa.size() + fb.size() - 1), n = 1 << L;
33     vector<C> in(n), out(n);
34
35     for (int i = 0; i < a.size(); i++) in[i] = C(fa[i], 0);
36     for (int i = 0; i < b.size(); i++) in[i].imag(fb[i]);
37
38     fft(in);
39     for (C& x : in) x *= x;
40     for (int i = 0; i < n; i++) out[i] = in[-i & (n - 1)] - conj(in[i]);
        // Corregido aqui
41     fft(out);
42
43     vll res(a.size() + b.size() - 1);
44     for (int i = 0; i < res.size(); i++) {
45         res[i] = llround(imag(out[i]) / (4 * n));
46     }
47     return res;
48 }
```

8.7 Rho

```

1 //RECOMENDADO USAR UNSIGNED LONG LONG
2 static inline ll mulmod(ll a, ll b, ll m) {
3     return (ll)((__int128)a * b % m);
4 }
5
6 static inline ll powmod(ll b, ll e, ll m) {
7     ll r = 1;
8     while (e) {
9         if (e & 1) r = mulmod(r, b, m);
10        b = mulmod(b, b, m);
11        e >>= 1;
12    }
13    return r;
14 }
15
16 // RNG rapido
17 static inline ll splitmix64(ll x) {
```

```

18     x += 0x9e3779b97f4a7c15ULL;
19     x = (x ^ (x >> 30)) * 0xbff58476d1ce4e5b9ULL;
20     x = (x ^ (x >> 27)) * 0x94d049bb133111ebULL;
21     return x ^ (x >> 31);
22 }
23 static ll rng_state = 0x1234567890abcdefULL ^ chrono::
24     high_resolution_clock::now().time_since_epoch().count();
25 static inline ll rnd() { return splitmix64(rng_state += 0
26                                         x9e3779b97f4a7c15ULL); }
27
28 // trial division pequena para acelerar
29 static const int SMALL_P_MAX = 1000;
30 static vector<int> small_primes;
31
32 static void sieve_small() {
33     vector<bool> is(SMALL_P_MAX + 1, true);
34     is[0] = is[1] = false;
35     for (int i = 2; i * i <= SMALL_P_MAX; ++i) if (is[i])
36         for (int j = i * i; j <= SMALL_P_MAX; j += i) is[j] = false;
37     for (int i = 2; i <= SMALL_P_MAX; ++i) if (is[i]) small_primes.
38         push_back(i);
39
40     bool isPrime(ll n) {
41         if (n < 2) return false;
42         // divide por primos pequenos
43         for (int p : small_primes) {
44             if ((ll)p * (ll)p > n) break;
45             if (n % p == (ll)0) return n == (ll)p;
46         }
47         if (n < 4) return true; // 2,3
48         // Miller-Rabin deterministico para 64-bit
49         ll d = n - 1, s = 0;
50         while ((d & 1) == 0) d >>= 1, ++s;
51         auto witness = [&](ll a) -> bool {
52             if (a % n == 0) return false;
53             ll x = powmod(a % n, d, n);
54             if (x == 1 || x == n - 1) return false;
55             for (int i = 1; i < s; ++i) {
56                 x = mulmod(x, x, n);
57                 if (x == n - 1) return false;
58             }
59             return true; // es testigo: n compuesto
60 }
```

```

58    };
59    // Bases correctas para 64-bit
60    for (ll a : {2ULL, 3ULL, 5ULL, 7ULL, 11ULL, 13ULL, 17ULL, 19ULL, 23
61        ULL,
62        325ULL, 9375ULL, 28178ULL, 450775ULL, 9780504ULL,
63        1795265022ULL}) {
64        if (a == 0) continue;
65        if (a % n == 0) continue;
66        if (witness(a)) return false;
67    }
68    return true;
69 }

ll pollard_rho(ll n) {
70    if ((n & 1ULL) == 0ULL) return 2ULL;
71    while (true) {
72        ll c = (rnd() % (n - 1)) + 1; // [1..n-1]
73        ll x = (rnd() % (n - 2)) + 2; // [2..n-1]
74        ll y = x;
75        ll d = 1;
76        // limite de iteraciones para evitar lazos raros
77        for (int it = 0; it < 1'000'000 && d == 1; ++it) {
78            x = (mulmod(x, x, n) + c) % n;
79            y = (mulmod(y, y, n) + c) % n;
80            y = (mulmod(y, y, n) + c) % n;
81            ll diff = x > y ? x - y : y - x;
82            d = std::gcd(diff, n);
83        }
84        if (d == 1 || d == n) continue;
85        return d;
86    }
87 }

void fact(ll n, map<ll,int> &F) {
88    if (n == 1) return;
89    if (isPrime(n)) { F[n]++; return; }
90    for (int p : small_primes) {
91        if ((ll)p * (ll)p > n) break;
92        while (n % p == 0) { F[p]++; n /= p; }
93    }
94    if (n == 1) return;
95    if (isPrime(n)) { F[n]++; return; }
96    ll d = pollard_rho(n);
97}

```

```

99    fact(d, F);
100   fact(n / d, F);
101 }

```

8.8 Simpson

```

1 ld simpsonRule(function<ld> f, ld a, ld b, int n) {
2     // Asegurarse de que n sea par
3     if (n % 2 != 0) {
4         n++;
5     }
6     ld h = (b - a) / n;
7     ld s = f(a) + f(b);

8
9     // Suma de terminos interiores con los factores apropiados
10    for (int i = 1; i < n; i++) {
11        ld x = a + i * h;
12        s += (i % 2 == 1 ? 4.0L : 2.0L) * f(x);
13    }
14    // Multiplica por h/3
15    return (h / 3.0L) * s;
16 }
17 // Ejemplo: integrar la funcion x^2 entre 0 y 3
18 auto f = [&](ld x){ return x * x; };
19 ld a = 0.0L, b = 3.0L;
20 int n = 1000; // numero de subintervalos
21 ld resultado = simpsonRule(f, a, b, n);

```

9 Geometry

9.1 Convex Hull

```

1 typedef pair<ll, ll> Point;
2 ll cross_product(Point O, Point A, Point B) {
3     return (A.first - O.first) * (B.second - O.second) - (A.second - O.
4         second) * (B.first - O.first);
5 }
6 vector<Point> convex_hull(vector<Point>& points) {
7     sort(points.begin(), points.end());
8     points.erase(unique(points.begin(), points.end()), points.end());
9     vector<Point> hull;
10    // Parte inferior
11    for (const auto& p : points) {

```

```

1   while (hull.size() >= 2 && cross_product(hull[hull.size() - 2],
2       hull[hull.size() - 1], p) < 0)
3       hull.pop_back();
4   if (hull.empty() || hull.back() != p) {
5       hull.push_back(p);
6   }
7 // Parte superior
8 int t = hull.size() + 1;
9 for (int i = points.size() - 1; i >= 0; --i) {
10    while (hull.size() >= t && cross_product(hull[hull.size() - 2],
11        hull[hull.size() - 1], points[i]) < 0)
12        hull.pop_back();
13    if (hull.empty() || hull.back() != points[i]) {
14        hull.push_back(points[i]);
15    }
16}
17 hull.pop_back();
18 return hull;
19 }
```

9.2 Operations

```

1  ll cross_product(pair<ll, ll> P1, pair<ll, ll> P2, pair<ll, ll> P3) {
2      ll x1 = P2.first - P1.first;
3      ll y1 = P2.second - P1.second;
4      ll x2 = P3.first - P1.first;
5      ll y2 = P3.second - P1.second;
6      return x1 * y2 - y1 * x2;
7  }
8 double distancia(pair<ll, ll> P1, pair<ll, ll> P2) {
9     return sqrt((P2.first - P1.first) * (P2.first - P1.first) +
10                 (P2.second - P1.second) * (P2.second - P1.second));
11 }
12 ll dot_product(pair<ll, ll> P1, pair<ll, ll> P2, pair<ll, ll> P3) {
13     ll x1 = P2.first - P1.first;
14     ll y1 = P2.second - P1.second;
15     ll x2 = P3.first - P1.first;
16     ll y2 = P3.second - P1.second;
17     return x1 * x2 + y1 * y2;
18 }
```

9.3 Polygon Area

```

1  typedef pair<ll, ll> Point;
2  double polygon_area(const vector<Point>& polygon) {
3      ll area = 0;
4      int n = polygon.size();
5      for (int i = 0; i < n; ++i) {
6          ll j = (i + 1) % n;
7          area += (polygon[i].first * polygon[j].second - polygon[i].second * polygon[j].first);
8      }
9      return abs(area) / 2.0;
10 }
```

9.4 Ray Casting

```

1  typedef pair<ll, ll> Point;
2  bool is_point_in_polygon(const vector<Point>& polygon, Point p) {
3      bool inside = false;
4      int n = polygon.size();
5      for (int i = 0, j = n - 1; i < n; j = i++) {
6          if ((polygon[i].second > p.second) != (polygon[j].second > p.
7              second) &&
8              p.first < (polygon[j].first - polygon[i].first) * (p.second -
9                  polygon[i].second) /
10                 (polygon[j].second - polygon[i].second) + polygon[11
12                 i].first) {
13                     inside = !inside;
14                 }
15             }
16         }
17     return inside;
18 }
```

10 Other

10.1 Mo's algorithm

```

1  const int BLOCK_SIZE = 450; using U64 = uint64_t;
2  struct query {int l, r, id; U64 order;};
3  U64 hilbertorder(U64 x, U64 y) {
4      const U64 logn = __lg(max(x, y) * 2 + 1) | 1;
5      const U64 maxn = (1ull << logn) - 1;
6      U64 res = 0;
7      for (U64 s = 1ull << (logn - 1); s; s >>= 1) {
8          bool rx = x & s, ry = y & s;
```

```
9     res = (res << 2) | (rx ? ry ? 2 : 1 : ry ? 3 : 0);
10    if (!rx) {
11        if (ry) x ^= maxn, y ^= maxn;
12        swap(x, y);
13    }
14    return res;
15 } // sort by this order
16 auto add = [&](int ix) { /* Add A[ix] to state*/};
17 auto rem = [&](int ix) { /* Remove A[ix] from state*/};
18 int c_l = 0, c_r = -1; // Cursors [0,-1] so r add 0 on first q
19 for(const auto &qr: qs){
20     while(c_l > qr.l) add(--c_l);
21     while(c_r < qr.r) add(++c_r);
22     while(c_l < qr.l) rem(c_l++);
23     while (c_r > qr.r) rem(c_r--);
24     ans[qr.id] = /*State.Answer()*/;
25 }
26 }
```