

# Dividimos y No Conquistamos (D&!C)

## Contents

<b>1</b>	<b>Template</b>	<b>2</b>
1.1	C++ Template . . . . .	2
1.2	Bash CMD . . . . .	2
1.3	C++ Int128 . . . . .	2
1.4	C++ RNG . . . . .	2
1.5	Python Template . . . . .	2
<b>2</b>	<b>Search</b>	<b>2</b>
2.1	Ternary . . . . .	2
<b>3</b>	<b>Data structures</b>	<b>3</b>
3.1	Fenwick . . . . .	3
3.2	Fenwick - 2D . . . . .	3
3.3	DSU . . . . .	3
3.4	Sparse Table . . . . .	3
3.5	Segment tree . . . . .	4
3.6	Segment Tree Iterativo . . . . .	4
3.7	Segment Tree Persistente . . . . .	4
3.8	Policy Based . . . . .	5
3.9	Chull Trick . . . . .	5
<b>4</b>	<b>Graph</b>	<b>6</b>
4.1	Bellman Ford . . . . .	6
4.2	SCC . . . . .	6
4.3	Bipartite Matching Hopcroft-Karp - With Konig . . . . .	6
4.4	Hungarian . . . . .	7
4.5	Flow - Dinics . . . . .	8
4.6	Flow - MinCostMaxFlow . . . . .	8
4.7	2-Sat . . . . .	10
4.8	Euler Tour . . . . .	10
4.9	Directed MST/Arborescence . . . . .	10
<b>5</b>	<b>Trees</b>	<b>11</b>
5.1	Heavy Light Decomposition . . . . .	11
5.2	Centroid . . . . .	12
5.3	LCA - Const Time . . . . .	12

<b>6</b>	<b>Dynamic Programming</b>	<b>13</b>
6.1	LIS . . . . .	13
<b>7</b>	<b>Strings</b>	<b>13</b>
7.1	Hashing . . . . .	13
7.2	KMP . . . . .	13
7.3	Z-Function . . . . .	13
7.4	Manacher . . . . .	14
7.5	Aho-Corasick . . . . .	14
7.6	Suffix-Array . . . . .	14
7.7	Suffix-Automaton . . . . .	15
<b>8</b>	<b>Math</b>	<b>15</b>
8.1	Euclidean Extended . . . . .	15
8.2	Euler Totient . . . . .	16
8.3	Josephus . . . . .	16
8.4	Mobius . . . . .	16
8.5	NTT . . . . .	17
8.6	FFT . . . . .	17
8.7	Rho . . . . .	18
8.8	Get Divisors . . . . .	19
8.9	Simpson . . . . .	19
8.10	Simplex . . . . .	19
<b>9</b>	<b>Geometry</b>	<b>20</b>
9.1	Point Definition . . . . .	20
9.2	Convex Hull . . . . .	20
9.3	Operations . . . . .	21
9.4	Polygon Area . . . . .	21
9.5	Ray Casting . . . . .	21
<b>10</b>	<b>Other</b>	<b>21</b>
10.1	Mo's algorithm . . . . .	21
<b>11</b>	<b>Ecuations</b>	<b>22</b>
11.1	Combinatorics . . . . .	22
11.2	Discreta . . . . .	22
11.3	Trigonometry . . . . .	24
11.4	Catalan Numbers . . . . .	24
11.5	Geometry . . . . .	25
11.6	Useful math . . . . .	26
11.7	Mobius . . . . .	28
11.8	Burnside . . . . .	28

# 1 Template

## 1.1 C++ Template

```

1 #pragma GCC target ("avx2")
2 #pragma GCC optimize ("O3")
3 #pragma GCC optimize ("unroll-loops")
4 #include <bits/stdc++.h>
5 using namespace std;
6 #define L(i, j, n) for (int i = (j); i < (int)n; i++)
7 #define SZ(x) int((x).size())
8 #define ALL(x) begin(x),end(x)
9 #define vec vector
10 #define pb push_back
11 #define eb emplace_back
12 using ll = long long;
13 using ld = long double;
14 void solve(){}
15 int main(){
16     ios::sync_with_stdio(0);cin.tie(0);
17     int TT = 1;
18     //cin >> TT;
19     while (TT--) {solve();}
20 }
21 // IF NEEDED FOR FILE READ
22 // freopen("in.txt", "r", stdin);
23 // freopen("out.txt", "w", stdout);

```

## 1.2 Bash CMD

```

1 co(){g++ $1/$1.cpp -o $1/$1 --std=c++20 -Wall -Wshadow -Wextra}
2 run(){for f in `ls ./`$1/*.txt`;do echo $f ;./$1/$1 < $f; done}
3 #Build, template.cpp must exist!
4 for x in {A..Z}; do mkdir $x; cp template.cpp $x/$x.cpp; touch $x/in.txt;done

```

## 1.3 C++ Int128

```

1 __int128 read() {
2     __int128 x=0,f=1;
3     char ch=getchar();
4     while (ch<'0' || ch>'9') {
5         if(ch == '-')f=-1;
6         ch=getchar();
7     }

```

```

8     while (ch >= '0' && ch <= '9') {
9         x=x*10+ch-'0';
10        ch=getchar();
11    }
12    return x * f;
13 }
14 void print(__int128 x) {
15     if (x < 0) {
16         putchar('-');
17         x = -x;
18     }
19     if (x > 9) print(x / 10);
20     putchar(x % 10 + '0');
21 }

```

## 1.4 C++ RNG

```

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;

```

## 1.5 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 }

```

### 3 Data structures

#### 3.1 Fenwick

```

1 #define LSO(S) (S & -S) //LeastsignificantOne
2 struct FT { // 1-Index
3     vec<int> ft; int n;
4     FT(vec<int> &v): ft(SZ(v)+1), n(SZ(v)+1) { // 0(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    }
17    int getSum(int l, int r){return sum(r) - sum(l - 1);}
18 };

```

#### 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);}
20 };

```

#### 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 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);

```

```

12     return min(st[bit][l], st[bit][r - (1<<bit) + 1]); // change op
13 }
14 };

```

### 3.5 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){
15         if (L == R){ st[v] = {0, 1}; return; }
16         int m = MD(L, R);
17         build(LC(v), L, m); build(RC(v), m + 1, R);
18         st[v] = merge(st[LC(v)], st[RC(v)]);
19     }
20     void push(int v, int L, int R){
21         if (lz[v]){
22             if (L != R){
23                 st[LC(v)].mx += lz[v]; // Apply to left
24                 st[RC(v)].mx += lz[v]; // And right
25                 lz[LC(v)] += lz[v];
26                 lz[RC(v)] += lz[v];
27             }
28             lz[v] = 0;
29         }
30     }
31     void update(int v, int L, int R, int ql, int qr, ll w){
32         if (ql > R || qr < L) return;
33         push(v, L, R);
34         if (ql == L && qr == R){
35             st[v].mx += w; // Update actual node
36             lz[v] += w; // Add lazy
37             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 };

```

### 3.6 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 i, ll w) {
9         i += K; st[i] += w;
10        while(i>=1)st[i]=st[i*2]+st[i*2+1];}
11    ll query(int l, int r) { // [l, r)
12        ll res = 0;
13        for (l += K, r += K; 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 };

```

### 3.7 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*>
   roots; roots.pb(build(ST_L, ST_R));
4  int p_num = 0; //
5  Vertex * init_leaf(int x) {
6      pool[p_num].sum = x;
7      pool[p_num].l = pool[p_num].r = NULL;
8      return &pool[p_num++];
9  }
10 Vertex * init_node(Vertex * l, Vertex * r) {
11     int sum = 0;
12     if (l) sum += l->sum;
13     if (r) sum += r->sum;
14     pool[p_num].sum = sum; pool[p_num].l = l; pool[p_num].r = r;
15     return &pool[p_num++];
16 Vertex * build(int L, int R){
17     if (L == R){return init_leaf(0);}
18     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);}

```

### 3.8 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,
   tree_order_statistics_node_update>;
5
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,
   Val>;

```

```

10 // como unordered_map (o unordered_set si Val es vacio), pero sin metodo
   count

```

### 3.9 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     }//Take care of overflow!
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 bool spfa(int s, vec<int>& d) {
2     int n = SZ(g); // needs g graph
3     d.assign(n, INF);
4     vec<int> cnt(n, 0), inqueue(n, 0);
5     queue<int> q; q.push(s);
6     d[s] = 0; inqueue[s] = true;
7     while (SZ(q)) {
8         int u = q.front(); q.pop();
9         inqueue[u] = false;
10        for (auto [v, w] : g[u]) {
11            if (d[u] + w < d[v]) {
12                d[v] = d[u] + w;
13                if (!inqueue[v]) {
14                    q.push(v); inqueue[v] = true;
15                    cnt[v]++;
16                    if (cnt[v] > n) return false; // negative cycle
17                }
18            }
19        }
20    }
21    return true;
22 }
```

### 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 );
3 struct hopcroft_karp {
4     int n, m; // n is Left Partition Size, m is Right Partition Size
5     vec<vec<int>> g;
6     vec<int> dist, nxt, ma, mb;
7     hopcroft_karp(int n_, int m_) : n(n_), m(m_), g(n),
8         dist(n), nxt(n), ma(n, -1), mb(m, -1) {}
9     void add(int a, int b) { g[a].pb(b); }
10    bool dfs(int i) {
11        for (int &id = nxt[i]; id < g[i].size(); id++) {
12            int j = g[i][id];
13            if (mb[j] == -1 or (dist[mb[j]] == dist[i]+1 and dfs(mb[j]))) {
14                ma[i] = j, mb[j] = i;
15                return true;
16            }
17        }
18        return false;
19    }
20    bool bfs() {
21        for (int i = 0; i < n; i++) dist[i] = n;
22        queue<int> q;
23        for (int i = 0; i < n; i++) if (ma[i] == -1) {
24            dist[i] = 0;
25            q.push(i);
26        }
27        bool rep = 0;
28        while (q.size()) {
29            int i = q.front(); q.pop();
30            for (int j : g[i]) {
31                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     };
60     L(u,0,n) if (ma[u] < 0) go(go,u);
61     return size;
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             L(k, 0, n) if(k!=j and sn[k]){
39                 auto w = ds[k]-ds[j]; v[k] += w, u[vR[k]] -= w;
40             }
41             u[s] += ds[j];
42             while(dad[j] >= 0){ int d = dad[j]; vR[j] = vR[d]; vL[vR[j]] = j;
43                 j = d; }
44             vR[j] = s; vL[s] = j;
45         }
46         ld value = 0; L(i, 0, n) value += cs[i][vL[i]], ans.pb({i, vL[i]});
47         return value;
48     }
49 };

```

## 4.5 Flow - Dinics

```

1 const int oo = (int)1e9;
2 struct Dinic {
3     bool scaling = false; // com scaling -> O(nm log(MAXCAP)),
4     int lim;               // com constante alta
5     struct edge {
6         int to, cap, rev, flow;
7         bool res;
8         edge(int to_, int cap_, int rev_, bool res_)
9             : to(to_), cap(cap_), rev(rev_), flow(0), res(res_) {}
10    };
11    vec<vec<edge>> g;
12    vec<int> lev, beg;
13    ll F;
14    Dinic(int n) : g(n), F(0) {}
15    void add(int a, int b, int c) {
16        g[a].emplace_back(b, c, g[b].size(), false);
17        g[b].emplace_back(a, 0, g[a].size()-1, true);
18    }
19    bool bfs(int s, int t) {
20        lev = vector<int>(g.size(), -1); lev[s] = 0;
21        beg = vector<int>(g.size(), 0);
22        queue<int> q; q.push(s);
23        while (q.size()) {
24            int u = q.front(); q.pop();
25            for (auto& i : g[u]) {
26                if (lev[i.to] != -1 or (i.flow == i.cap)) continue;
27                if (scaling and i.cap - i.flow < lim) continue;
28                lev[i.to] = lev[u] + 1;
29                q.push(i.to);
30            }
31        }
32        return lev[t] != -1;
33    }
34    int dfs(int v, int s, int f = oo) {
35        if (!f or v == s) return f;
36        for (int& i = beg[v]; i < g[v].size(); i++) {
37            auto& e = g[v][i];
38            if (lev[e.to] != lev[v] + 1) continue;
39            int foi = dfs(e.to, s, min(f, e.cap - e.flow));
40            if (!foi) continue;
41            e.flow += foi, g[e.to][e.rev].flow -= foi;

```

```

42        return foi;
43    }
44    return 0;
45    }
46    ll max_flow(int s, int t) {
47        for (lim = scaling ? (1<<30) : 1; lim; lim /= 2)
48            while (bfs(s, t)) while (int ff = dfs(s, t)) F += ff;
49        return F;
50    }
51    };
52    vec<pair<int, int>> get_cut(Dinic& g, int s, int t) {
53        g.max_flow(s, t);
54        vec<pair<int, int>> cut;
55        vec<int> vis(g.g.size(), 0), st = {s};
56        vis[s] = 1;
57        while (st.size()) {
58            int u = st.back(); st.pop_back();
59            for (auto e : g.g[u]) if (!vis[e.to] and e.flow < e.cap)
60                vis[e.to] = 1, st.push_back(e.to);
61        }
62        for (int i = 0; i < g.g.size(); i++) for (auto e : g.g[i])
63            if (vis[i] and !vis[e.to] and !e.res) cut.emplace_back(i, e.to);
64        return cut;
65    }

```

## 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(
11                 cost_) {}
12    };
13    vec<vec<edge>> g;
14    vec<int> par_idx, par;
15    T inf;
16    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
    custo cost
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                 else q.push_front(to);
42                 is_inside[to] = true;
43             }
44         }
45     }
46     return dist;
47 }
48 bool dijkstra(int s, int t, vec<T>& pot) {
49     priority_queue<pair<T, int>, vec<pair<T, int>>, greater<>> q;
50     dist = vec<T>(g.size(), inf);
51     dist[s] = 0;
52     q.emplace(0, s);
53     while (q.size()) {
54         auto [d, v] = q.top();
55         q.pop();

```

```

56         if (dist[v] < d) continue;
57         for (int i = 0; i < g[v].size(); i++) {
58             auto [to, rev, flow, cap, res, cost] = g[v][i];
59             cost += pot[v] - pot[to];
60             if (flow < cap and dist[v] + cost < dist[to]) {
61                 dist[to] = dist[v] + cost;
62                 q.emplace(dist[to], to);
63                 par_idx[to] = i, par[to] = v;
64             }
65         }
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];

```

```

36         if (comp[v] == -1) dfs2(v, cc ++);
37     }
38     assign.assign(n, false);
39     for (int i = 0; i < v_n; i += 2) {
40         if (comp[i] == comp[i+1]) return false;
41         assign[i / 2] = comp[i] > comp[i + 1];
42     }
43     return true;
44 }
45 };

```

## 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 }

```

## 4.9 Directed MST/Arborescence

```

1 struct RollbackUF { // Required

```

```

2  vi e; vector<pii> st;
3  RollbackUF(int n) : e(n, -1) {}
4  int size(int x) { return -e[find(x)]; }
5  int find(int x) { return e[x] < 0 ? x : find(e[x]); }
6  int time() { return sz(st); }
7  void rollback(int t) {
8      for (int i = time(); i --> t;)
9          e[st[i].first] = st[i].second;
10     st.resize(t);
11 }
12 bool join(int a, int b) {
13     a = find(a), b = find(b);
14     if (a == b) return false;
15     if (e[a] > e[b]) swap(a, b);
16     st.push_back({a, e[a]});
17     st.push_back({b, e[b]});
18     e[a] += e[b]; e[b] = a;
19     return true;
20 }
21 };
22
23 struct Edge { int a, b; ll w; };
24 struct Node { /// lazy skew heap node
25     Edge key;
26     Node *l, *r;
27     ll delta;
28     void prop() {
29         key.w += delta;
30         if (l) l->delta += delta;
31         if (r) r->delta += delta;
32         delta = 0;
33     }
34     Edge top() { prop(); return key; }
35 };
36 Node *merge(Node *a, Node *b) {
37     if (!a || !b) return a ? b : a;
38     a->prop(), b->prop();
39     if (a->key.w > b->key.w) swap(a, b);
40     swap(a->l, (a->r = merge(b, a->r)));
41     return a;
42 }
43 void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }
44 pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {

```

```

45     RollbackUF uf(n);
46     vector<Node*> heap(n);
47     for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e});
48     ll res = 0;
49     vi seen(n, -1), path(n), par(n);
50     seen[r] = r;
51     vector<Edge> Q(n), in(n, {-1,-1}), comp;
52     deque<tuple<int, int, vector<Edge>>> cycs;
53     rep(s,0,n) {
54         int u = s, qi = 0, w;
55         while (seen[u] < 0) {
56             if (!heap[u]) return {-1,{} };
57             Edge e = heap[u]->top();
58             heap[u]->delta -= e.w, pop(heap[u]);
59             Q[qi] = e, path[qi++] = u, seen[u] = s;
60             res += e.w, u = uf.find(e.a);
61             if (seen[u] == s) { /// found cycle, contract
62                 Node* cyc = 0;
63                 int end = qi, time = uf.time();
64                 do cyc = merge(cyc, heap[w = path[--qi]]);
65                 while (uf.join(u, w));
66                 u = uf.find(u), heap[u] = cyc, seen[u] = -1;
67                 cycs.push_front({u, time, {&Q[qi], &Q[end]} });
68             }
69         }
70         rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
71     }
72     for (auto& [u,t,comp] : cycs) { // restore sol (optional)
73         uf.rollback(t);
74         Edge inEdge = in[u];
75         for (auto& e : comp) in[uf.find(e.b)] = e;
76         in[uf.find(inEdge.b)] = inEdge;
77     }
78     rep(i,0,n) par[i] = in[i].a;
79     return {res, par};
80 }

```

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

```

1 struct LCA {
2     vec<int> depth, in, euler;
3     vec<vec<int>> g, st;
4     int K, n;
5     inline int Min(int i, int j) {return depth[i] <= depth[j] ? i : j;}
6     void dfs(int u, int p) {
7         in[u] = SZ(euler);
8         euler.pb(u);
9         for (int v: g[u]) if (v != p){
10             depth[v] = depth[u] + 1;
11             dfs(v, u);
12             euler.pb(u);
13         }
14     }
15     LCA(int n_): depth(n_), g(vec<vec<int>>(n_)), K(0), n(n_), in(n_) {
16         euler.reserve(2 * n); }
17     void add_edge(int u, int v) {g[u].pb(v);}
18     void build(int root){
19         dfs(root, -1);
20     }
21 }

```

```

19     int ln = SZ(euler);
20     while((1<<K)<=ln)K++;
21     st = vec<vec<int>> (K, vec<int>(ln));
22     L(i,0,ln) st[0][i] = euler[i];
23     for (int i = 1; (1 << i) <= ln; i++) {
24         for (int j = 0; j + (1<<i) <= ln; j++) {
25             st[i][j] = Min(st[i-1][j], st[i-1][j + (1<<(i-1))]);
26         }
27     }
28 }
29 int get(int u, int v) {
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 LIS

```

1 int lis(vec<int> &a) {
2     int n = SZ(a);
3     vector<int> d(n+1, oo);
4     d[0] = -oo;
5     L(i,0,n){
6         int l = upper_bound(d.begin(), d.end(), a[i]) - d.begin();
7         if (d[l-1] < a[i] && a[i] < d[l]) d[l] = a[i];
8     }
9     int ans = 0;
10    for (int l = 0; l <= n; l++) if (d[l] < oo) ans = l;
11    return ans;
12 }

```

## 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 };

```

### 7.2 KMP

```

1 struct KMP {
2     string s; int n; vec<int> p; vec<vec<int>> dfa;
3     KMP(string &s_): s(s_), n(SZ(s_)), p(SZ(s_) + 1), dfa(SZ(s_)+1, vec<
4         int>(26)) {
5         L(i,1,n) p[i + 1] = nxt(p[i], s[i]); // Calculate phi
6     }
7     int nxt(int i, char c) {for (;i=p[i]);if(i<n&&c==s[i])return i+1;
8         return s[0]==c;}
9     void build_dfa(){
10        dfa[0][s[0]-'a'] = 1; // WARN: check lower_case vs upper
11        L(i,1,n+1)L(c,0,26) // If complicated char set use map
12            if (i<n&&s[i]=='a'+c)dfa[i][c]=i+1;
13            else dfa[i][c]=dfa[p[i]][c]; // fallar en i e ir al c
14    }
15    int go(int v, char c){return dfa[v][c-'a'];}
16 };

```

### 7.3 Z-Function

```

1 vec<int> zfun(const string &w){
2     int n = SZ(w), l = 0, r = 0; vec<int> z(n);

```

```

3   z[0] = n;
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 }

```

## 7.4 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].
15 };

```

## 7.5 Aho-Corasick

```

1 struct node {
2     int ch[26], next[26]; // Full DFA Transitions
3     int link = 0, minx = oo; // Suffix Link
4     vec<int> ixs; // Indices of patterns ending here
5     node() { memset(ch, -1, sizeof(ch));}
6 };
7 vec<node> t; vec<int> bfs_order; // easy traverse from short to longer
8 words
9 void init_aho() {t.clear();t.pb(node());bfs_order.clear();}
10 void add_string(const string &s, const int ix) {
11     int v = 0;
12     for (char c_raw : s) {
13         int c = c_raw - 'a';
14         if (t[v].ch[c] == -1) {
15             t[v].ch[c] = SZ(t);

```

```

15         t.pb(node());
16     }
17     v = t[v].ch[c];
18 }
19 t[v].ixs.pb(ix);
20 }
21 void build_aho() {
22     bfs_order.pb(0); // Root is first
23     L(c,0,26){
24         if (t[0].ch[c] != -1) {
25             t[0].next[c] = t[0].ch[c];
26             bfs_order.pb(t[0].ch[c]);
27         } else t[0].next[c] = 0;
28     }
29     L(q, 1, SZ(bfs_order)){ // warn: 1 not 0!
30         int u = bfs_order[q];
31         L(c,0,26){
32             if (t[u].ch[c] != -1) {
33                 int v = t[u].ch[c];
34                 t[u].next[c] = v;
35                 t[v].link = t[t[u].link].next[c];
36                 bfs_order.pb(v);
37             } else t[u].next[c] = t[t[u].link].next[c];
38         }
39     }
40 }

```

## 7.6 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 }

```

## 7.7 Suffix-Automaton

```

1 struct state {int len,link;map<char,int> next;}; //clear next!!
2 state st[2 * N]; // Important 2 * n
3 int sz,last;
4 void sa_init(){
5     last=st[0].len=0;sz=1;
6     st[0].link=-1;
7 }
8 void sa_extend(char c){
9     int k=sz++,p;
10    st[k].len=st[last].len+1;
11    for(p=last;p!=-1&&!st[p].next.count(c);p=st[p].link)st[p].next[c]=k;
12    if(p==-1)st[k].link=0;

```

```

13 else {
14     int q=st[p].next[c];
15     if(st[p].len+1==st[q].len)st[k].link=q;
16     else {
17         int w=sz++;
18         st[w].len=st[p].len+1;
19         st[w].next=st[q].next;st[w].link=st[q].link;
20         for(;p!=-1&&st[p].next[c]==q;p=st[p].link)st[p].next[c]=w;
21         st[q].link=st[k].link=w;
22     }
23 }
24 last=k;
25 }

```

## 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;

```

```

25
26     if (x < x_min) {
27         ll k = (x_min - x + b - 1) / b; // Redondeo hacia arriba
28         x += k * b;
29         y -= k * a;
30     } else if (x > x_min) {
31         ll k = (x - x_min) / b;
32         x -= k * b;
33         y += k * a;
34     }
35
36     if (y < y_min) {
37         ll k = (y_min - y + a - 1) / a; // Redondeo hacia arriba
38         x += k * b;
39         y -= k * a;
40     } else if (y > y_min) {
41         ll k = (y - y_min) / a;
42         x -= k * b;
43         y += k * a;
44     }
45
46     return x >= x_min && y >= y_min;
47 }

```

## 8.2 Euler Totient

```

1 vector<ll> compute_totients(ll n) {
2     vector<ll> phi(n + 1);
3     for (ll i = 0; i <= n; i++) phi[i] = i;
4     for (ll i = 2; i <= n; i++) {
5         if (phi[i] != i) continue;
6         for (ll j = i; j <= n; j += i)
7             phi[j] = phi[j] * (i - 1) / i;
8     }
9     return phi;
10 }

```

## 8.3 Josephus

```

1 ll josephus_iterative(ll n, ll k) {
2     ll result = 0;
3     for (ll i = 2; i <= n; ++i)
4         result = (result + k) % i;
5     return result;

```

```

6 }
7 ll josephus_recursive(ll n, ll k) {
8     if (n == 1) return 0;
9     return (josephus_recursive(n - 1, k) + k) % n;
10 }
11 ll josephus_power_of_2(ll n) {
12     ll power = 1;
13     while (power <= n) power <<= 1;
14     power >>= 1;
15     return 2 * (n - power);
16 }

```

## 8.4 Mobius

```

1 vector<ll> compute_mobius(ll n) {
2     vector<ll> mu(n + 1, 1);
3     vector<bool> is_prime(n + 1, true);
4     for (ll i = 2; i <= n; i++) {
5         if (is_prime[i]) { // i es un primo
6             for (ll j = i; j <= n; j += i) {
7                 mu[j] *= -1; // Multiplicamos por -1 para cada primo
8                 is_prime[j] = false;
9             }
10            for (ll j = i * i; j <= n; j += i * i) {
11                mu[j] = 0; // Si tiene un cuadrado de un primo, se pone
12                           // en 0
13            }
14        }
15    }
16    return mu;
17 }
18 ll mobius(ll x) {
19     ll count = 0;
20     for (ll i = 2; i * i <= x; i++) {
21         if (x % (i * i) == 0)
22             return 0;
23         if (x % i == 0) {
24             count++;
25             x /= i;
26         }
27     }
28     if (x > 1) count++;
29     return (count % 2 == 0) ? 1 : -1;

```



## 8.5 NTT

```

29 }

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 }
19
20 void ntt(vector<ll> & a, bool invert) {
21     int n = a.size();
22
23     for (int i = 1, j = 0; i < n; i++) {
24         int bit = n >> 1;
25         for (; j & bit; bit >>= 1)
26             j ^= bit;
27         j ^= bit;
28
29         if (i < j)
30             swap(a[i], a[j]);
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
49 if (invert) {
50     int n_1 = inverse(n, mod);
51     for (auto & x : a)
52         x = (int)(1LL * x * n_1 % mod);
53 }
54 }
55
56 vector<ll> multiply(vector<ll> const &a, vector<ll> const &b) {
57     vector<ll> fa(a.begin(), a.end()), fb(b.begin(), b.end());
58     ll n = 1;
59     while (n < a.size() + b.size())
60         n <<= 1;
61     fa.resize(n);
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    }
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]);
41        // Corregido aqui
42        fft(out);
43
44        vll res(a.size() + b.size() - 1);
45        for (int i = 0; i < res.size(); i++) {
46            res[i] = llround(imag(out[i]) / (4 * n));
47        }

```

```

47    return res;
48 }

```

## 8.7 Rho

```

1 ll mul(ll a, ll b, ll mod) {
2     return (__int128)a * b % mod;
3 }
4
5 ll power(ll a, ll b, ll mod) {
6     ll res = 1;
7     while (b) {
8         if (b & 1) res = mul(res, a, mod);
9         a = mul(a, a, mod);
10        b >>= 1;
11    }
12    return res;
13 }
14
15 bool isPrime(ll n) {
16     if (n < 2) return false;
17     for (ll p : {2, 3, 5, 7, 11, 13, 17, 19, 23}) {
18         if (n % p == 0) return n == p;
19     }
20     ll d = n - 1, s = 0;
21     while ((d & 1) == 0) d >>= 1, ++s;
22     for (ll a : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) {
23         if (a % n == 0) continue;
24         ll x = power(a, d, n);
25         if (x == 1 || x == n - 1) continue;
26         bool ok = false;
27         for (int i = 0; i < s; ++i) {
28             x = mul(x, x, n);
29             if (x == n - 1) { ok = true; break; }
30         }
31         if (!ok) return false;
32     }
33     return true;
34 }
35
36 ll rho(ll n) {
37     if (n % 2 == 0) return 2;
38     while (true) {

```

```

39     ll c = rand() % (n - 1) + 1;
40     ll x = 2, y = 2, d = 1;
41     while (d == 1) {
42         x = (mul(x, x, n) + c) % n;
43         y = (mul(y, y, n) + c) % n;
44         y = (mul(y, y, n) + c) % n;
45         d = std::gcd((x > y ? x - y : y - x), n);
46     }
47     if (d != n) return d;
48 }
49 }
50
51 void fact(ll n, std::map<ll, int>& f) {
52     if (n == 1) return;
53     if (isPrime(n)) { f[n]++; return; }
54     ll d = rho(n);
55     if (d == n) {
56         f[n]++;
57         return;
58     }
59     fact(d, f);
60     fact(n / d, f);
61 }

```

## 8.8 Get Divisors

```

1 vector<ll> getDivisors(const map<ll, int>& f) {
2     vector<ll> divisors = { 1 };
3     for (auto [p, e] : f) {
4         vector<ll> next;
5         ll pe = 1;
6         for (int i = 0; i <= e; i++) {
7             for (ll d : divisors) next.pb(d * pe);
8             pe *= p;
9         }
10        divisors.swap(next);
11    }
12    sort(ALL(divisors));
13    return divisors;
14 }

```

## 8.9 Simpson

```

1 ld simpsonRule(function<ld(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);

```

## 8.10 Simplex

```

1 pair<ld, vec<ld>> simplex(vec<vec<ld>> A, vec<ld> b, vec<ld> c) {
2     const ld EPS = (ld)1e-9;
3     int n = SZ(b), m = SZ(c);
4
5     vec<int> X(m), Y(n);
6     L(j, 0, m) X[j] = j;
7     L(i, 0, n) Y[i] = m + i;
8
9     ld z = 0;
10
11    auto pivot = [&](int x, int y) {
12        swap(X[y], Y[x]);
13    }
14
15    ld inv = (ld)1 / A[x][y];
16    b[x] *= inv;
17    L(j, 0, m) if (j != y) A[x][j] *= inv;
18    A[x][y] = inv;
19
20    L(i, 0, n) if (i != x && fabs1(A[i][y]) > EPS) {
21        ld coef = A[i][y];

```

```

21     b[i] -= coef * b[x];
22     L(j, 0, m) if (j != y) A[i][j] -= coef * A[x][j];
23     A[i][y] = -coef * A[x][y];
24 }
25
26 z += c[y] * b[x];
27 L(j, 0, m) if (j != y) c[j] -= c[y] * A[x][j];
28 c[y] = -c[y] * A[x][y];
29 };
30
31 while (true) {
32     int x = -1, y = -1;
33     ld mn = -EPS;
34     L(i, 0, n) if (b[i] < mn) { mn = b[i]; x = i; }
35     if (x < 0) break;
36     L(j, 0, m) if (A[x][j] < -EPS) { y = j; break; }
37     if (y < 0) {
38         return { numeric_limits<ld>::quiet_NaN(), {} };
39     }
40     pivot(x, y);
41 }
42
43 while (true) {
44     int y = -1, x = -1;
45     ld mx = EPS;
46     L(j, 0, m) if (c[j] > mx) { mx = c[j]; y = j; }
47     if (y < 0) break;
48
49     ld best = numeric_limits<ld>::infinity();
50     L(i, 0, n) if (A[i][y] > EPS) {
51         ld val = b[i] / A[i][y];
52         if (val < best) { best = val; x = i; }
53     }
54     if (x < 0) {
55         return { numeric_limits<ld>::infinity(), {} };
56     }
57     pivot(x, y);
58 }
59
60 vec<ld> sol(m, 0);
61 L(i, 0, n) if (Y[i] < m) sol[Y[i]] = b[i];
62 return { z, sol };
63 }

```

## 9 Geometry

### 9.1 Point Definition

```

1 const double EPS = 1e-7;
2 struct pt { // for 3D add z coordinate, define EPS
3     double x,y;
4     pt(double x, double y):x(x),y(y){}
5     pt(){}
6     double norm2(){return *this**this;}
7     double norm(){return sqrt(norm2());}
8     bool operator==(pt p){return abs(x-p.x)<=EPS&&abs(y-p.y)<=EPS;}
9     pt operator+(pt p){return pt(x+p.x,y+p.y);}
10    pt operator-(pt p){return pt(x-p.x,y-p.y);}
11    pt operator*(double t){return pt(x*t,y*t);}
12    pt operator/(double t){return pt(x/t,y/t);}
13    double operator*(pt p){return x*p.x+y*p.y;} // dot prod
14    // pt operator^(pt p){ // only for 3D
15    //     return pt(y*p.z-z*p.y,z*p.x-x*p.z,x*p.y-y*p.x);}
16    double angle(pt p){ // redefine acos for values out of range
17        return acos(*this*p/(norm()*p.norm()));}
18    pt unit(){return *this/norm();}
19    double operator%(pt p){return x*p.y-y*p.x;} // cross prod
20    // 2D from now on
21    bool operator<(pt p)const{ // for convex hull
22        return x<p.x-EPS|| (abs(x-p.x)<=EPS&&y<p.y-EPS);}
23    bool left(pt p, pt q){ // is it to the left of directed line pq?
24        return (q-p)%(*this-p)>EPS;}
25    pt rot(pt r){return pt(*this*r,*this*r);}
26    pt rot(double a){return pt(sin(a),cos(a));}
27 };
28 pt ccw90(1,0);
29 pt cw90(-1,0);

```

### 9.2 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());

```

```

7   points.erase(unique(points.begin(), points.end()), points.end());
8   vector<Point> hull;
9   // Parte inferior
10  for (const auto& p : points) {
11      while (hull.size() >= 2 && cross_product(hull[hull.size() - 2],
12          hull[hull.size() - 1], p) < 0)
13          hull.pop_back();
14      if (hull.empty() || hull.back() != p) {
15          hull.push_back(p);
16      }
17  }
18  // Parte superior
19  int t = hull.size() + 1;
20  for (int i = points.size() - 1; i >= 0; --i) {
21      while (hull.size() >= t && cross_product(hull[hull.size() - 2],
22          hull[hull.size() - 1], points[i]) < 0)
23          hull.pop_back();
24      if (hull.empty() || hull.back() != points[i]) {
25          hull.push_back(points[i]);
26      }
27  }
28  hull.pop_back();
29  return hull;

```

### 9.3 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  }

```

### 9.4 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].
8              second * polygon[j].first);
9      }
10  return abs(area) / 2.0;

```

### 9.5 Ray Casting

```

1  int inPolygon(const vector<pt>& p, pt a) { // 0: Outside, 1: Inside, 2:
2      Boundary
3      int ans = 0; int n = SZ(p);
4      L(i,0,n) {
5          pt p1 = p[i], p2 = p[(i + 1) % n];
6          if ((p2 - p1) % (a - p1) == 0 &&
7              min(p1.x, p2.x) <= a.x && a.x <= max(p1.x, p2.x) &&
8              min(p1.y, p2.y) <= a.y && a.y <= max(p1.y, p2.y)) return 2;
9          if ((p1.y > a.y) != (p2.y > a.y)) {
10             ll cp = (p2 - p1) % (a - p1);
11             if (p1.y < p2.y ? cp > 0 : cp < 0) ans = 1 - ans;
12         }
13     }
14  return ans;

```

## 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  }
15  return res;
16 } // sort by this order
17 auto add = [&](int ix) { /* Add A[ix] to state*/};
18 auto rem = [&](int ix) { /* Remove A[ix] from state*/};
19 int c_l = 0, c_r = -1; // Cursors [0,-1] so r add 0 on first q
20 for(const auto &q: qs){
21     while(c_l > qr.l) add(--c_l);
22     while(c_r < qr.r) add(++c_r);
23     while(c_l < qr.l) rem(c_l++);
24     while (c_r > qr.r) rem(c_r--);
25     ans[qr.id] = /*State.Answer()*/;
26 }

```

## 11 Ecuations

### 11.1 Combinatorics

$$\binom{n}{k} = \binom{n}{n-k}$$

$$\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}, \quad (1 \leq k \leq n)$$

$$\sum_{k=0}^n \binom{n}{k} = 2^n$$

$$\sum_{m=0}^n \binom{m}{k} = \binom{n+1}{k+1}, \quad (n \geq k \geq 0)$$

$$\sum_{k=0}^m \binom{n+k}{k} = \binom{n+m+1}{m}$$

$$\binom{n}{0}^2 + \binom{n}{1}^2 + \cdots + \binom{n}{n}^2 = \binom{2n}{n}$$

$$\sum_{k=1}^n k \binom{n}{k} = n 2^{n-1}$$

$$F_n = \sum_{k=0}^{\lfloor \frac{n-1}{2} \rfloor} \binom{n-k-1}{k}$$

$$F_{n+1} = \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n-k}{k}$$

### 11.2 Discreta

**Vandermonde convolution:**

$$\sum_{k=0}^n \binom{r}{k} \binom{s}{n-k} = \binom{r+s}{n}$$

**Multinomial theorem:**

$$(x_1 + \cdots + x_m)^n = \sum_{\substack{a_1 + \cdots + a_m = n \\ a_i \geq 0}} \frac{n!}{a_1! \cdots a_m!} x_1^{a_1} \cdots x_m^{a_m}$$

**Binomial inversion (sequence form):**

$$g(n) = \sum_{k=0}^n \binom{n}{k} f(k) \iff f(n) = \sum_{k=0}^n (-1)^{n-k} \binom{n}{k} g(k)$$

**Stars and bars (nonnegative):**

$$x_1 + \cdots + x_k = n, \ x_i \geq 0 \Rightarrow \# = \binom{n+k-1}{k-1}$$

**Positive parts:**

$$x_1 + \cdots + x_k = n, \ x_i \geq 1 \Rightarrow \# = \binom{n-1}{k-1}$$

**Compositions of n:**

$$\#\{\text{ordered positive sum of } n \text{ into } k \text{ parts}\} = \binom{n-1}{k-1}, \quad \#\{\text{all compositions}\} = 2^{n-1}$$

**Upper bounds via inclusion-exclusion:**

$$x_1 + \cdots + x_k = n, \ 0 \leq x_i \leq u_i \Rightarrow \# = \sum_{S \subseteq \{1, \dots, k\}} (-1)^{|S|} \binom{n - \sum_{i \in S} (u_i + 1) + k - 1}{k-1}$$

$$(\text{toma } \binom{t}{k-1} = 0 \text{ si } t < k-1)$$

**Multiset combinations:**

$$\#\{k\text{-multicombinations from } n \text{ types}\} = \binom{n+k-1}{k}$$

**Multiset permutations:**

$$\#\{\text{perm of multiset with counts } m_1, \dots, m_r\} = \frac{(m_1 + \cdots + m_r)!}{m_1! \cdots m_r!}$$

**Circular permutations:**

$$\#\{\text{distinct cyclic orders of } n \text{ items}\} = (n-1)!$$

**Surjections count (onto functions):**

$$\#\{f : [m] \rightarrow [n] \text{ onto}\} = \sum_{j=0}^n (-1)^j \binom{n}{j} (n-j)^m = n! S(m, n)$$

**Derangements:**

$$!n = n! \sum_{i=0}^n \frac{(-1)^i}{i!} \quad \text{and} \quad !n \approx \frac{n!}{e}$$

**Stirling numbers (second kind):**

$$S(n, k) = k S(n-1, k) + S(n-1, k-1), \quad S(0, 0) = 1$$

**Stirling numbers (first kind, unsigned):**

$$c(n, k) = c(n-1, k-1) + (n-1) c(n-1, k), \quad c(0, 0) = 1$$

**Expansions with falling powers:**

$$x^{\underline{n}} = \sum_{k=0}^n s(n, k) x^k, \quad x^n = \sum_{k=0}^n S(n, k) x^{\underline{k}}$$

$$(\text{here } x^{\underline{k}} = x(x-1) \cdots (x-k+1), \ s(n, k) = (-1)^{n-k} c(n, k))$$

**Bell numbers:**

$$B_n = \sum_{k=0}^n S(n, k), \quad \sum_{n \geq 0} B_n \frac{x^n}{n!} = \exp(e^x - 1)$$

**Cayley trees:**

$$\#\{\text{labeled trees on } n \text{ vertices}\} = n^{n-2}$$

**Perfect matchings in complete graph:**

$$\#\{\text{perfect matchings in } K_{2n}\} = (2n-1)!! = \frac{(2n)!}{2^n n!}$$

**Grid shortest paths:**

$$\#\{\text{monotone paths from } (0,0) \text{ to } (a,b)\} = \binom{a+b}{a}$$

**Ballot (Bertrand special case):**

$$p > q \Rightarrow \#\{\text{prefix-wise leading sequences}\} = \frac{p-q}{p+q} \binom{p+q}{q}$$

**Alternating binomial sums:**

$$\sum_{k=0}^n (-1)^k \binom{n}{k} = 0 \quad (n \geq 1), \quad \sum_{k=0}^n (-1)^k \binom{n}{k} k^m = 0 \quad (0 \leq m < n)$$

**Lucas theorem (mod prime p):**

$$n = \sum n_i p^i, \quad k = \sum k_i p^i \Rightarrow \binom{n}{k} \equiv \prod_i \binom{n_i}{k_i} \pmod{p}$$

### 11.3 Trigonometry

$$\begin{aligned} \sin(-x) &= -\sin x, & \cos(-x) &= \cos x, & \tan(-x) &= -\tan x \\ \sin^2 x + \cos^2 x &= 1, & 1 + \tan^2 x &= \sec^2 x, & 1 + \cot^2 x &= \csc^2 x \end{aligned}$$

$$\sin(\alpha \pm \beta) = \sin \alpha \cos \beta \pm \cos \alpha \sin \beta$$

$$\cos(\alpha \pm \beta) = \cos \alpha \cos \beta \mp \sin \alpha \sin \beta$$

$$\tan(\alpha \pm \beta) = \frac{\tan \alpha \pm \tan \beta}{1 \mp \tan \alpha \tan \beta}$$

$$\sin(2x) = 2 \sin x \cos x$$

$$\cos(2x) = \cos^2 x - \sin^2 x = 1 - 2 \sin^2 x = 2 \cos^2 x - 1$$

$$\tan(2x) = \frac{2 \tan x}{1 - \tan^2 x}$$

$$\begin{aligned} \sin(3x) &= 3 \sin x - 4 \sin^3 x, & \cos(3x) &= 4 \cos^3 x - 3 \cos x \\ \sin^2 \frac{x}{2} &= \frac{1 - \cos x}{2}, & \cos^2 \frac{x}{2} &= \frac{1 + \cos x}{2}, & \tan \frac{x}{2} &= \frac{\sin x}{1 + \cos x} = \frac{1 - \cos x}{\sin x} \end{aligned}$$

$$\sin \alpha \sin \beta = \frac{1}{2} [\cos(\alpha - \beta) - \cos(\alpha + \beta)]$$

$$\cos \alpha \cos \beta = \frac{1}{2} [\cos(\alpha - \beta) + \cos(\alpha + \beta)]$$

$$\sin \alpha \cos \beta = \frac{1}{2} [\sin(\alpha + \beta) + \sin(\alpha - \beta)]$$

$$\sin \alpha \pm \sin \beta = 2 \sin \frac{\alpha \pm \beta}{2} \cos \frac{\alpha \mp \beta}{2}$$

$$\cos \alpha + \cos \beta = 2 \cos \frac{\alpha + \beta}{2} \cos \frac{\alpha - \beta}{2}$$

$$\cos \alpha - \cos \beta = -2 \sin \frac{\alpha + \beta}{2} \sin \frac{\alpha - \beta}{2}$$

$$\sum_{k=0}^{n-1} \cos(a + kd) = \frac{\sin(\frac{nd}{2})}{\sin(\frac{d}{2})} \cos\left(a + \frac{(n-1)d}{2}\right)$$

$$\sum_{k=0}^{n-1} \sin(a + kd) = \frac{\sin(\frac{nd}{2})}{\sin(\frac{d}{2})} \sin\left(a + \frac{(n-1)d}{2}\right)$$

$$\text{Law of sines: } \frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2R$$

$$\text{Law of cosines: } c^2 = a^2 + b^2 - 2ab \cos C \quad (\text{and cyclic})$$

$$\text{Area: } \Delta = \frac{1}{2} ab \sin C = \frac{1}{2} bc \sin A = \frac{1}{2} ca \sin B$$

$$(x', y') = (x \cos \theta - y \sin \theta, x \sin \theta + y \cos \theta)$$

$$e^{ix} = \cos x + i \sin x, \quad \cos x = \frac{e^{ix} + e^{-ix}}{2}, \quad \sin x = \frac{e^{ix} - e^{-ix}}{2i}$$

$$\pi \text{ rad} = 180^\circ, \quad 1^\circ = \frac{\pi}{180} \text{ rad}$$

### 11.4 Catalan Numbers

**Recursive definition:**

$$C_0 = C_1 = 1$$

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}, \quad n \geq 2$$



**Closed form:**

$$C_n = \frac{1}{n+1} \binom{2n}{n}$$

**Combinatorial equivalent:**

$$C_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{1}{n+1} \binom{2n}{n}, \quad n \geq 0$$

**Combinatorial meaning:**

Number of ways to: (i) arrange  $n$  balanced parenthesis pairs; (ii) full binary trees with  $n+1$  leaves; (iii) Dyck paths of length  $2n$  that never cross the diagonal.

**Generalized form  $(k)$ :**

$$C_n^{(k)} = \frac{k+1}{n+k+1} \binom{2n+k}{n}$$

**Extended recurrence:**

$$C_n^{(k)} = \sum_{a_1 + \dots + a_k = n} C_{a_1} C_{a_2} \dots C_{a_k}, \quad C_0 = 1$$

**Efficient recurrence (for computation):**

$$C_n = \frac{2(2n-1)}{n+1} C_{n-1}, \quad n \geq 1$$

**Generating function:**

$$C(x) = \sum_{n=0}^{\infty} C_n x^n = \frac{1 - \sqrt{1-4x}}{2x}$$

**Asymptotic behavior:**

$$C_n \sim \frac{4^n}{n^{3/2} \sqrt{\pi}}$$

**Examples:**

$$C_0 = 1, \quad C_1 = 1, \quad C_2 = 2, \quad C_3 = 5, \quad C_4 = 14, \quad C_5 = 42$$

## 11.5 Geometry

**Rectangle:**

$$A = b h$$

Area with base  $b$  and height  $h$ .

**Triangle:**

$$A = \frac{1}{2} b h$$

$$A = \sqrt{s(s-a)(s-b)(s-c)}, \quad s = \frac{a+b+c}{2} \quad (\text{Heron})$$

Base-height or Heron using side lengths  $a, b, c$ .

**Parallelogram & rhombus:**

$$A_{\text{parallelogram}} = b h, \quad A_{\text{rhombus}} = \frac{D d}{2}$$

$D, d$  are diagonals of a rhombus.

**Trapezoid:**

$$A = \frac{(B+b)}{2} h$$

$B$  and  $b$  are the parallel sides (bases).

**Regular  $n$ -gon:**

$$A = \frac{1}{2} P a = \frac{n l a}{2}$$

$P$  perimeter,  $l$  side,  $a$  apothem.

**Circle:**

$$A = \pi r^2, \quad C = 2\pi r$$

**Circular sector (angle in radians):**

$$A = \frac{1}{2} r^2 \theta, \quad \text{arc length } L = r \theta$$

**Circular segment (height  $h$ ):**

$$A = r^2 \arccos\left(\frac{r-h}{r}\right) - (r-h) \sqrt{2rh - h^2}$$

Region cut by a chord;  $0 < h < 2r$ .

**Annulus (circular crown):**

$$A = \pi(R^2 - r^2)$$

Difference of two concentric disks ( $R > r$ ).

**Ellipse:**

$$A = \pi ab$$

$a, b$  are semi-axes.

**Polygon by coordinates (shoelace):**

$$A = \frac{1}{2} \left| \sum_{i=1}^n (x_i y_{i+1} - x_{i+1} y_i) \right|, \quad (x_{n+1}, y_{n+1}) = (x_1, y_1)$$

Works for any simple polygon in the plane.

**Triangle by coordinates:**

$$A = \frac{1}{2} |x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)|$$

**Triangle from sides and circumradius/inradius:**

$$A = \frac{abc}{4R}, \quad A = r s$$

$R$  circumradius,  $r$  inradius,  $s$  semiperimeter.

**Lune (difference of two circular sectors):**

$$A_{\text{lune}} = \frac{1}{2} r_1^2 \theta_1 - \frac{1}{2} r_2^2 \theta_2$$

Two sectors overlapping with angles  $\theta_1, \theta_2$  matching the same chord.

**Lens (two equal circles radius  $r$ , center distance  $d$ ):**

$$A = 2r^2 \arccos\left(\frac{d}{2r}\right) - \frac{d}{2} \sqrt{4r^2 - d^2}, \quad 0 < d < 2r$$

Intersection of two equal disks.

**Spherical cap (radius  $R$ , height  $h$ ):**

$$A_{\text{cap}} = 2\pi R h$$

Surface area of the cap on a sphere. (Volume:  $V = \frac{\pi h^2}{3} (3R - h)$ )

## 11.6 Useful math

**Arithmetic series:**

$$\sum_{k=1}^n k = \frac{n(n+1)}{2}$$

**Squares & cubes:**

$$\sum_{k=1}^n k^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{k=1}^n k^3 = \left(\frac{n(n+1)}{2}\right)^2$$

**Geometric series ( $r \neq 1$ ):**

$$\sum_{i=0}^{n-1} r^i = \frac{r^n - 1}{r - 1}$$

For mod prime  $p$ : multiply by  $(r-1)^{-1} \equiv (r-1)^{p-2} \pmod{p}$ .

**Power sum of base  $a$ :**

$$1 + a + \cdots + a^n = \frac{a^{n+1} - 1}{a - 1} \quad (a \neq 1), \quad = n + 1 \quad (a = 1)$$

**Harmonic numbers:**

$$H_n = \sum_{k=1}^n \frac{1}{k} \approx \ln n + \gamma + \frac{1}{2n}$$

$\gamma \approx 0.57721$  (Euler–Mascheroni). Useful for estimates.

**Basic mod rules:**

$$(a \pm b) \pmod{m} = ((a \pmod{m}) \pm (b \pmod{m})) \pmod{m}$$

$$(a \cdot b) \pmod{m} = ((a \pmod{m}) \cdot (b \pmod{m})) \pmod{m}$$

**Fermat little theorem (prime  $p$ ):**

$$a^{p-1} \equiv 1 \pmod{p} \quad \text{if } p \nmid a, \quad a^{-1} \equiv a^{p-2} \pmod{p}$$

**Euler theorem:**

$$a^{\varphi(m)} \equiv 1 \pmod{m} \text{ if } \gcd(a, m) = 1$$

**Chinese remainder (pairwise coprime):**

$$x \equiv a_i \pmod{m_i} \Rightarrow x \equiv \sum_i a_i M_i y_i \pmod{M}$$

$$M = \prod m_i, M_i = M/m_i, y_i \equiv M_i^{-1} \pmod{m_i}.$$

**gcd/lcm relation:**

$$\text{lcm}(a, b) = \frac{|ab|}{\gcd(a, b)}$$

**Binomial theorem:**

$$(x + y)^n = \sum_{k=0}^n \binom{n}{k} x^{n-k} y^k$$

**Stars and bars (non-neg.):**

$$x_1 + \dots + x_k = n, x_i \geq 0 \Rightarrow \# = \binom{n+k-1}{k-1}$$

**Permutations & combinations:**

$$P(n, k) = \frac{n!}{(n-k)!}, \quad \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

**Derangements (approx):**

$$!n = \left\lfloor \frac{n!}{e} + \frac{1}{2} \right\rfloor, \quad !n = n! \sum_{i=0}^n \frac{(-1)^i}{i!}$$

**Stirling approximation:**

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

**Inclusion-Exclusion (finite):**

$$\left| \bigcup_{i=1}^m A_i \right| = \sum_i |A_i| - \sum_{i < j} |A_i \cap A_j| + \dots + (-1)^{m+1} |A_1 \cap \dots \cap A_m|$$

**Dot and cross (2D):**

$$u \cdot v = u_x v_x + u_y v_y = |u||v| \cos \theta, \quad u \times v = u_x v_y - u_y v_x$$

$$|u \times v| = 2 \times \text{triangle area}(u, v). \text{ Orientation by sign of } u \times v.$$

**Distance point to line AB:**

$$\text{dist}(P, AB) = \frac{|(B - A) \times (P - A)|}{|B - A|}$$

**Projection length on AB:**

$$\text{proj}_{AB}(P) = \frac{(P - A) \cdot (B - A)}{|B - A|}$$

**AM-GM (non-neg.):**

$$\frac{x_1 + \dots + x_n}{n} \geq (x_1 \dots x_n)^{1/n}$$

**Cauchy-Schwarz:**

$$\left( \sum_i a_i b_i \right)^2 \leq \left( \sum_i a_i^2 \right) \left( \sum_i b_i^2 \right)$$

**Log rules:**

$$\log_a b = \frac{\ln b}{\ln a}, \quad \log(ab) = \log a + \log b$$

**Fast exponent splits:**

$$a^{x+y} = a^x a^y, \quad a^{2^k} = \underbrace{(\dots (a^2)^2 \dots)}_{k \text{ times}}$$

**Divisor count/sum (multiplicative):**

$$n = \prod p_i^{e_i} \Rightarrow \tau(n) = \prod (e_i + 1), \quad \sigma(n) = \prod \frac{p_i^{e_i+1} - 1}{p_i - 1}$$

$\tau(n)$  = number of divisors,  $\sigma(n)$  = sum of divisors.

**Linearity of expectation:**

$$\mathbb{E}\left[\sum_i X_i\right] = \sum_i \mathbb{E}[X_i] \quad (\text{no independence needed})$$

**Binomial distribution:**

$$X \sim \text{Bin}(n, p) \Rightarrow \mathbb{E}[X] = np, \quad \text{Var}(X) = np(1 - p)$$

## 11.7 Mobius

**Mobius function mu:**

$$\mu(1) = 1$$

$$\mu(n) = 0 \text{ if } \exists p^2 \mid n, \quad \mu(n) = (-1)^k \text{ if } n \text{ is square-free with } k \text{ distinct primes.}$$

**Basic convolutions:**

$$\sum_{d \mid n} \mu(d) = \begin{cases} 1, & n = 1, \\ 0, & n > 1, \end{cases} \quad (\mu * \mathbf{1})(n) = \varepsilon(n).$$

**Mobius inversion (divisor-sum):**

If

$$g(n) = \sum_{d \mid n} f(d),$$

then

$$f(n) = \sum_{d \mid n} \mu(d) g\left(\frac{n}{d}\right) = \sum_{d \mid n} \mu\left(\frac{n}{d}\right) g(d).$$

**Inversion over multiples:**

If

$$G(n) = \sum_{k \geq 1} f(kn),$$

then

$$f(n) = \sum_{k \geq 1} \mu(k) G(kn).$$

**Euler totient via mu:**

$$\varphi(n) = \sum_{d \mid n} \mu(d) \frac{n}{d}.$$

**Counting coprimes up to x:**

$$\#\{1 \leq m \leq x : \gcd(m, n) = 1\} = \sum_{d \mid n} \mu(d) \left\lfloor \frac{x}{d} \right\rfloor.$$

**Square-free count up to x:**

$$Q(x) = \#\{n \leq x : n \text{ square-free}\} = \sum_{k \leq \sqrt{x}} \mu(k) \left\lfloor \frac{x}{k^2} \right\rfloor.$$

**GCD=1 k-tuples:**

$$\#\{1 \leq x_1, \dots, x_k \leq n : \gcd(x_1, \dots, x_k) = 1\} = \sum_{d=1}^n \mu(d) \left\lfloor \frac{n}{d} \right\rfloor^k.$$

**Dirichlet series:**

$$\sum_{n=1}^{\infty} \frac{\mu(n)}{n^s} = \frac{1}{\zeta(s)}, \quad \Re(s) > 1.$$

**Mertens function:**

$$M(x) = \sum_{n \leq x} \mu(n).$$

## 11.8 Burnside

**Necklaces under cyclic  $C_n$  (gcd form):**

$$N_{\text{rot}} = \frac{1}{n} \sum_{k=0}^{n-1} m^{\gcd(n, k)}.$$

**Equivalent divisor form (for  $C_n$ ):**

$$N_{\text{rot}} = \frac{1}{n} \sum_{d|n} \varphi(d) m^{n/d}.$$

**Necklaces/bracelets under dihedral  $D_n$  (gcd form):**

$$N = \begin{cases} \frac{1}{2n} \left( \sum_{k=0}^{n-1} m^{\gcd(n,k)} + n m^{(n+1)/2} \right), & n \text{ odd,} \\ \frac{1}{2n} \left( \sum_{k=0}^{n-1} m^{\gcd(n,k)} + \frac{n}{2} m^{n/2} + \frac{n}{2} m^{n/2+1} \right), & n \text{ even.} \end{cases}$$

Here gcd is the greatest common divisor. A rotation by  $k$  positions has  $\gcd(n, k)$  cycles.