palao - ICPC Library

Contents

1 Data Structures

1.1 Fenwick

```
const int mx = 2e5+5;
ll bit[mx];
int n, q;

ll qry(int i){ // [1,i] 1 indexado
        ll ret = 0;
        for(; i > 0; i -= i & -i)
                ret += bit[i];
        return ret;
}

void increment(ll i, ll v){ // 1 indexado (+= v)
        for(; i <= n; i += i & -i)
                bit[i] += v;
}
```

1.2 Segtree Lazy

```
// Lazy SegTree
const int mx = 2e5+5;
vector<ll> seg(4*mx);
vector<ll> lazy(4*mx,0);
vector<ll> nums(mx);
int n,q;

void build(int l = 0, int r = n-1, int idx = 0){
        if(l == r){
                seg[idx] = nums[l];
                lazy[idx] = 0;
                return;
        }
        int m = (l+r)/2;
        int left = 2*idx+1;
        int right = 2*idx+2;
        build(l,m,left);
        build(m+1,r,right);
        seg[idx] = seg[left] + seg[right];
}

void prop(int l = 0, int r = n-1, int idx = 0){
        seg[idx] += (ll)(r-l+1)*lazy[idx];
        if(l != r){ // nao for folha
                int left = 2*idx+1;
                int right = 2*idx+2;
                lazy[left] += lazy[idx];
                lazy[right] += lazy[idx];
        }
        lazy[idx] = 0;
}

void update(int L, int R, ll val, int l = 0, int r = n-1, int idx = 0){
        if(R < l || L > r) return;
        prop(l,r,idx);
        if(L <= l && r <= R){
                lazy[idx] = val;
                prop(l,r,idx);
        }
        else{
                int m = (l+r)/2;
                int left = 2*idx+1;
                int right = 2*idx+2;
                update(L,R,val,l,m,left);
                update(L,R,val,m+1,r,right);
```

```
                seg[idx] = seg[left] + seg[right];
        }
}

ll query(int L, int R, int l = 0, int r = n-1, int idx = 0){
        prop(l,r,idx);
        if(R < l || L > r) return 0;
        if(L <= l && r <= R){
                return seg[idx];
        }
        int m = (l+r)/2;
        int left = 2*idx+1;
        int right = 2*idx+2;
        return query(L,R,l,m,left) + query(L,R,m+1,r,right);
}
```

1.3 Segtree

```
// SegTree
const int mx = 2e5 + 5;
ll seg[4*mx];
ll nums[mx];
int n,q;
ll merge(ll a, ll b){
        return a+b;
}

void build(int l = 0, int r = n-1, int idx = 0){
        if(l == r){
                seg[idx] = nums[l];
                return;
        }
        int mid = l + (r-l)/2;
        int left = 2*idx + 1;
        int right = 2*idx + 2;
        build(l,mid,left);
        build(mid+1,r,right);
        seg[idx] = merge(seg[left], seg[right]);
}

ll query(int L, int R, int l = 0, int r = n-1, int idx = 0){
        if(R < l || L > r) return 0; // elemento neutro
        if(L <= l && r <= R) return seg[idx];

        int mid = l + (r-l)/2;
        int left = 2*idx + 1;
        int right = 2*idx + 2;
        ll ql = query(L,R,l,mid,left);
        ll qr = query(L,R,mid+1,r,right);
        return merge(ql,qr);
}

void update(int pos, int num, int l = 0, int r = n-1, int idx = 0){
        if(l == r){
                seg[idx] = num;
                return;
        }
        int mid = l + (r-l)/2;
        int left = 2*idx + 1;
        int right = 2*idx + 2;
        if(pos <= mid){
                update(pos,num,l,mid,left);
        }
        else update(pos,num,mid+1,r,right);
        seg[idx] = merge(seg[left],seg[right]);
}
```

1.4 SparseTable

```
vector<vector<ll>> table;
vector<ll> lg2;
void build(int n, vector<ll> v) {
  lg2.resize(n + 1);
  lg2[1] = 0;
  for (int i = 2; i <= n; i++) {
    lg2[i] = lg2[i >> 1] + 1;
  }
  table.resize(lg2[n] + 1);
  for (int i = 0; i < lg2[n] + 1; i++) {
    table[i].resize(n + 1);
  }
  for (int i = 0; i < n; i++) {
    table[0][i] = v[i];
  }
  for (int i = 0; i < lg2[n]; i++) {    for (int j = 0; j < n; j++) {
      if (j + (1 << i) >= n) break;
      table[i + 1][j] = min(table[i][j], table[i][j + (1 << i)]);
    }
  }
}
ll get(int l, int r) { // (l,r) inclusivo
  int k = lg2[r - l + 1];
  return min(table[k][l], table[k][r - (1 << k) + 1]);
}
```

1.5 Dynamic Median

```
const ll inf = 1e18 + 5;
struct DynamicMedian{
  multiset<ll> left, right;
  ll leftsum = 0, rightsum = 0;
  ll get(){
    // if(left.empty()) return -1; // cuidar aqui
    return *left.rbegin();
  }
  ll qry(){ // somatorio de distancia absoluta pra mediana
    ll m = get();
    // if(m == -1) return -1;
    return left.size()*m - leftsum + rightsum - right.size()*m;
  }
  void fix(){
    // (L,R) ou (L+1,R)
    while(right.size() + 1 < left.size()){
      // tirar do l e colocar no r
      auto lst = --left.end();
      rightsum += *lst;
      leftsum -= *lst;
      right.insert(*lst);
      left.erase(lst);
    }
    while(right.size() > left.size()){
      // tirar do r e colocar no l
      leftsum += *right.begin();
      rightsum -= *right.begin();
      left.insert(*right.begin());
      right.erase(right.begin());
    }
  }

  void insert(ll x){
    ll m = (left.empty() ? inf : get());
    if(x <= m){
      left.insert(x);
      leftsum += x;
```

```
    }else{
      right.insert(x);
      rightsum += x;
    }
    fix();
  }
  void erase(ll x){
    auto l = left.find(x);
    if(l != left.end()){
      leftsum -= *l;
      left.erase(l);
    }
    else{
      auto r = right.find(x);
      rightsum -= *r;
      right.erase(r);
    }
    fix();
  }
};
```

1.6 MaxQueue

```
template <class T, class C = less<T>>
struct MaxQueue {
  MaxQueue() { clear(); }
  void clear() {
    id = 0;
    q.clear();
  }
  void push(T x) {
    pair<int, T> nxt(1, x);
    while(q.size() > id && cmp(q.back().second, x)) {
      nxt.first += q.back().first;
      q.pop_back();
    }
    q.push_back(nxt);
  }
  T qry() { return q[id].second;}
  void pop() {
    q[id].first--;
    if(q[id].first == 0) { id++; }
  }
private:
  vector<std::pair<int, T>> q;
  int id;
  C cmp;
};
```

2 DP

2.1 CHT

```
struct Line {
  ll m, c;
  Line(ll m, ll c) : m(m), c(c) {}
  ll eval(ll x) {
    return m * x + c;
  }
};
struct CHT {
  vector<Line> lines;
  bool bad(Line a, Line b, Line c) {
        // trocar pra < se for max
    return 1.d * (c.c - a.c)*(a.m - b.m) > 1.d * (b.c - a.c)*(a.m - c.m);
  }
  void insert(Line line) { // sortar antes de inserir
    int sz = (int)lines.size();
    for(; sz > 1; --sz) {
      if(bad(lines[sz - 2], lines[sz - 1], line)) {
        lines.pop_back();
        continue;
      }
      break;
    }
    lines.emplace_back(line);
  }
  ll query(ll x) {
    int l = 0, r = (int)lines.size() - 1;
    while(l < r) {
      int m = (l+r)/2;
      // trocar pra < se for max
      if(lines[m].eval(x) > lines[m+1].eval(x)) {
        l = m + 1;
      } else {
        r = m;
      }
    }
    return lines[l].eval(x);
  }
};
```

2.2 Knapsack

```
// Knapsack
const int MXW = 1e5+5;
const int MXN = 105;

int n, max_w;
vector<int> weight(MXN), value(MXN);
vector<vector<ll>> dp(MXN,vector<ll>(MXW,-1));

ll solveDp(int i, int k){ // k -> peso atual
        if(i == n) return 0;
        if(dp[i][k] != -1) return dp[i][k];

        ll ignore = solveDp(i+1,k);
        ll add = -1;
        if(weight[i] + k <= max_w){
                add = value[i] + solveDp(i+1, weight[i] + k);
        }
        return dp[i][k] = max(ignore,add);
}

// iterativo

ll knapsack(){
  vector<ll> dp(dpmx,0);
  for(int i = 0; i < n; i++){
    ll w = weight[i];
    ll v = value[i];
    for(int sz = max_w; sz >= w; sz--){
      dp[sz] = max(dp[sz],dp[sz-w]+v);
    }
  }
  return *max_element(begin(dp),end(dp));
}
```

2.3 LIS

```
// Longest Increasing Sequence
int lis(vector<ll>& nums){
        int n = nums.size();
        vector<ll> s;
```

```
	for(int i = 0; i < n; i++){
		auto it = lower_bound(s.begin(),s.end(),nums[i]);
		if(it == s.end()){
			s.PB(nums[i]);
		}
		else{
			*it = nums[i];
		}
	}
	return (int)s.size();
}
```

3 Geometry

3.1 Point

```
// hypot, atan2, gcd
const double PI = acos(-1);
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<typename T>
struct PT{
  T x, y;
  PT(T x=0, T y=0) : x(x),y(y){}
  bool operator < (PT o) const { return tie(x,y) < tie(o.x,o.y); }
  bool operator == (PT o) const { return tie(x,y) == tie(o.x,o.y); }
  PT operator + (PT o) const { return PT(x+o.x,y+o.y); }
  PT operator - (PT o) const { return PT(x-o.x,y-o.y); }
  PT operator * (T k) const { return PT(x*k,y*k); }
  PT operator / (T k) const { return PT(x/k,y/k); }
  T cross(PT o) const { return x*o.y - y*o.x; }
  T cross(PT a, PT b) const { return (a-*this).cross(b-*this); }
  T dot(PT o) const { return x*o.x + y*o.y; }
  T dist2() const { return x*x + y*y; }
  double len() const { return hypot(x,y); }
  PT perp() const { return PT(-y,x); }
  PT rotate(double a) const { return PT(x*cos(a)-y*sin(a), x*sin(a)+y*cos(a
      )); }
};
ostream &operator<<(ostream &os, const PT<ll> &p) {
  return os << "(" << p.x << "," << p.y << ")";
}
```

3.2 Convex Hull

```
// retorna poligono no sentido anti horario, trocar pra < se quiser horario
template<typename T>
vector<PT<T>> convexHull(vector<PT<T>>& pts, bool sorted = false){
  if(!sorted) sort(begin(pts),end(pts));
  vector<PT<T>> h;
  h.reserve(pts.size() + 1);
  for(int it = 0; it < 2; it++){
    int start = h.size();
    for(PT<T>& c : pts){
      while((int)h.size() >= start + 2){
        PT<T> a = h[h.size()-2], b = h.back();
        // '>=' pra nao descartar pontos colineares
        if((b-a).cross(c-a) > 0) break;
        h.pop_back();
      }
      h.push_back(c);
    }
    reverse(begin(pts),end(pts));
    h.pop_back();
  }
  if(h.size() == 2 && h[0] == h[1]) h.pop_back();
  return h;
}
```

```
// nao funciona se tem pontos colineares!!!!
// considera ponto na aresta como dentro
template<typename T>
bool isInside(vector<PT<T>>& hull, PT<T> p) {
  int n = hull.size();
  PT<T> v0 = p - hull[0], v1 = hull[1] - hull[0], v2 = hull[n-1] - hull[0];
  if(v0.cross(v1) > 0 || v0.cross(v2) < 0){
    return false;
  }
  int l = 1, r = n - 1;
  while(l != r){
    int mid = (l + r + 1) / 2;
    PT<T> v0 = p - hull[0], v1 = hull[mid] - hull[0];
    if(v0.cross(v1) < 0)
      l = mid;
    else
      r = mid - 1;
  }
  v0 = hull[(l+1)%n] - hull[l], v1 = p - hull[l];
  return v0.cross(v1) >= 0;
}

// poligonos
ll polygon_area_db(const vector<Point>& poly){
  ll area = 0;
  for(int i = 0, n = (int)poly.size(); i < n; ++i) {
    int j = i + 1 == n ? 0 : i + 1;
    area += cross(poly[i], poly[j]);
  }
  return abs(area);
}
// Teorema de Pick para lattice points
// Area = insidePts + boundPts/2 - 1
// 2A - b + 2 = 2i
// usar gcd dos lados pra contar bound pts
ll cntInsidePts(ll area_db, ll bound){
  return (area_db + 2LL - bound)/2;
}
```

3.3 Min Enclosing Circle

```
typedef PT<double> P;
double ccRadius(P& A, P& B, P& C) {
  return (B-A).len()*(C-B).len()*(A-C).len()/
      abs((B-A).cross(C-A))/2.0;
}

P ccCenter(P& A, P& B, P& C) {
  P b = C-A, c = B-A;
  return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
// mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
pair<P, double> mec(vector<P>& pts){
	shuffle(begin(pts),end(pts),rng);
	P o = pts[0];
	const double EPSS = 1+1e-8;
	double r = 0;
	for(int i = 0; i < pts.size(); i++) if((o-pts[i]).len() > r * EPSS)
	    {
		o = pts[i], r = 0;
		for(int j = 0; j < i; j++) if((o-pts[j]).len() > r * EPSS){
			o = (pts[i]+pts[j])/2.0;
			r = (o - pts[i]).len();
			for(int k = 0; k < j; k++) if((o-pts[k]).len() > r
			    * EPSS){
				o = ccCenter(pts[i],pts[j],pts[k]);
				r = (o - pts[i]).len();
			}
		}
```

```
        }
        return {o, r};
}
```

3.4 Clostest Pair

```
pii ClostestPair(vector<PT<ll>>& pts) {
  ll dist = (pts[0]-pts[1]).dist2();
  pii ans(0, 1);
  int n = pts.size();
  vector<int> p(n);
  iota(begin(p),end(p),0);
  sort(p.begin(), p.end(), [&](int a, int b) { return pts[a].x < pts[b].x;
      });
  set<pii> points;
  auto sqr = [](long long x) -> long long { return x * x; };
  for(int l = 0, r = 0; r < n; r++) {
    while(sqr(pts[p[r]].x - pts[p[l]].x) > dist) {
      points.erase(pii(pts[p[l]].y, p[l]));
      l++;
    }
    ll delta = sqrt(dist) + 1;
    auto itl = points.lower_bound(pii(pts[p[r]].y - delta, -1));
    auto itr = points.upper_bound(pii(pts[p[r]].y + delta, n + 1));
    for(auto it = itl; it != itr; it++) {
      ll curDist = (pts[p[r]] - pts[it->second]).dist2();
      if(curDist < dist) {
        dist = curDist;
        ans = pii(p[r], it->second);
      }
    }
    points.insert(pii(pts[p[r]].y, p[r]));
  }
  if(ans.first > ans.second)
    swap(ans.first, ans.second);
  return ans;
}
```

4 ETC

4.1 Ternary Search

```
double f(double t){
        // alguma funcao unimodal -> maximo ou minimo
        //      /\
        //    /  \
        // __/    \__
}

double tern_search(double l, double r){
  for(int it = 0; it < 300; it++){
    double m1 = l + (r-l)/3;
    double m2 = r - (r-l)/3;
    double f1 = f(m1), f2 = f(m2);
    if(f1 < f2) l = m1; //change to > to find maximum
    else r = m2;
  }
  return l;
}
// golden section search
double gss(double a, double b) {
  const double r = (sqrt(5)-1)/2, eps = 1e-7;
  double x1 = b - r*(b-a), x2 = a + r*(b-a);
  double f1 = f(x1), f2 = f(x2);
  for(int it = 0; it < 250 && b-a > eps; it++)
    if (f1 < f2) { //change to > to find maximum
      b = x2; x2 = x1; f2 = f1;
      x1 = b - r*(b-a); f1 = f(x1);
    } else {
      a = x1; x1 = x2; f1 = f2;
      x2 = a + r*(b-a); f2 = f(x2);
    }
  return a;
}
// retorna mais a esquerda no empate
int int_tern_search(int l, int r){
        int lo = l - 1, hi = r;
        while(hi - lo > 1){
                int m = (lo+hi)/2;
                if(f(m) < f(m+1)){ //
                        lo = m;
                }else{
                        hi = m;
                }
        }
        return lo + 1;
}
```

4.2 Mo Algorithm

```
// Mo apelao
// Ordering based on the Hilbert curve
inline int64_t hilbertOrder(int x, int y, int pow, int rotate){
    if(pow == 0) return 0;
    int hpow = 1 << (pow - 1);
    int seg = (x < hpow) ? ( (y < hpow) ? 0 : 3) : ( (y < hpow) ? 1 : 2);
    seg = (seg + rotate) & 3;
    const int rotateDelta[4] = {3, 0, 0, 1};
    int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
    int nrot = (rotate + rotateDelta[seg]) & 3;
    int64_t subSquareSize = int64_t(1) << (2*pow - 2);
    int64_t ans = seg * subSquareSize;
    int64_t add = hilbertOrder(nx, ny, pow - 1, nrot);
    ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add - 1);
    return ans;
}
struct Query{
    int l, r, idx;
    int64_t ord;
    Query(int l, int r, int idx) : l(l), r(r), idx(idx) {
        ord = hilbertOrder(l, r, 21, 0);
    }
    bool operator < (Query &other){
        return ord < other.ord;
    }
};
// hash de cima:
    cc976f44618d4ffc1bce4043eeed0ab2d48e270c90075135727d8e8b83bc8e76
// Mo normal
const int MXN = 2e5;
const int B = sqrt(MXN) + 1;
struct Query {
    int l, r, idx;
    bool operator<(Query o) const{
      return make_pair(l / B, ((l/B) & 1) ? -r : r) < make_pair(o.l / B, ((
          o.l/B) & 1) ? -o.r : o.r);
    }
};

ll a[MXN];
ll resp = 0;
void add(int x);
void remove(int x);
int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
```

```
    int n, q; cin >> n >> q;
    for(int i = 0; i < n; i++)
        cin >> a[i];
    vector<Query> queries;
    for(int i = 0; i < q; i++){
        int l, r; cin >> l >> r;
        queries.push_back(Query(l-1,r-1,i));
    }
    sort(begin(queries),end(queries));
    vector<ll> answers(q);
    int L = 0, R = -1;
    for(Query qr : queries){
      while (L > qr.l) add(--L);
      while (R < qr.r) add(++R);
      while (L < qr.l) remove(L++);
      while (R > qr.r) remove(R--);
      answers[qr.idx] = resp;
    }
    for(int i = 0; i < q; i++)
        cout << answers[i] << "\n";
}
```

4.3 Prime list to 1000

```
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71,
     73
79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157,
     163, 167, 173, 179, 181
191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269,
     271, 277, 281, 283, 293, 307
311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397,
     401, 409, 419, 421, 431, 433
439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523,
     541, 547, 557, 563, 569, 571
577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659,
     661, 673, 677, 683, 691, 701
709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811,
     821, 823, 827, 829, 839, 853
857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953,
     967, 971, 977, 983, 991, 997
Produtorios:
3: 2*3*5 = 30
4: 2*3*5*7 = 210
5: 2*3*5*7*11 = 2.310
6: 2*3*5*7*11*13 = 30.030
7: 2*3*5*7*11*13*17 = 510.510
8: 2*3*5*7*11*13*17*19 = 9.699.690
```

4.4 Highly composite numbers

number	divisors	factorization
1	1	
2	2	2
4	3	2^2
6	4	2*3
12	6	2^2*3
24	8	2^3*3
36	9	2^2*3^2
48	10	2^4*3
60	12	2^2*3*5
120	16	2^3*3*5
180	18	2^2*3^2*5
240	20	2^4*3*5
360	24	2^3*3^2*5
720	30	2^4*3^2*5
840	32	2^3*3*5*7
1.260	36	2^2*3^2*5*7
1.680	40	2^4*3*5*7
2.520	48	2^3*3^2*5*7
5.040	60	2^4*3^2*5*7
7.560	64	2^3*3^3*5*7
10.080	72	2^5*3^2*5*7
15.120	80	2^4*3^3*5*7
20.160	84	2^6*3^2*5*7
25.200	90	2^4*3^2*5^2*7
27.720	96	2^3*3^2*5*7*11
45.360	100	2^4*3^4*5*7
50.400	108	2^5*3^2*5^2*7
55.440	120	2^4*3^2*5*7*11
83.160	128	2^3*3^3*5*7*11
110.880	144	2^5*3^2*5*7*11
166.320	160	2^4*3^3*5*7*11
221.760	168	2^6*3^2*5*7*11
277.200	180	2^4*3^2*5^2*7*11
332.640	192	2^5*3^3*5*7*11
498.960	200	2^4*3^4*5*7*11
554.400	216	2^5*3^2*5^2*7*11
665.280	224	2^6*3^3*5*7*11
720.720	240	2^4*3^2*5*7*11*13
1.081.080	256	2^3*3^3*5*7*11*13
1.441.440	288	2^5*3^2*5*7*11*13
2.162.160	320	2^4*3^3*5*7*11*13
2.882.880	336	2^6*3^2*5*7*11*13
3.603.600	360	2^4*3^2*5^2*7*11*13
4.324.320	384	2^5*3^3*5*7*11*13
6.486.480	400	2^4*3^4*5*7*11*13
7.207.200	432	2^5*3^2*5^2*7*11*13
8.648.640	448	2^6*3^3*5*7*11*13
10.810.800	480	2^4*3^3*5^2*7*11*13
14.414.400	504	2^6*3^2*5^2*7*11*13
17.297.280	512	2^7*3^3*5*7*11*13
21.621.600	576	2^5*3^3*5^2*7*11*13
32.432.400	600	2^4*3^4*5^2*7*11*13
36.756.720	640	2^4*3^3*5*7*11*13*17
43.243.200	672	2^6*3^3*5^2*7*11*13
61.261.200	720	2^4*3^2*5^2*7*11*13*17
73.513.440	768	2^5*3^3*5*7*11*13*17
110.270.160	800	2^4*3^4*5*7*11*13*17
122.522.400	864	2^5*3^2*5^2*7*11*13*17
147.026.880	896	2^6*3^3*5*7*11*13*17
183.783.600	960	2^4*3^3*5^2*7*11*13*17
245.044.800	1008	2^6*3^2*5^2*7*11*13*17
294.053.760	1024	2^7*3^3*5*7*11*13*17
367.567.200	1152	2^5*3^3*5^2*7*11*13*17
551.350.800	1200	2^4*3^4*5^2*7*11*13*17
698.377.680	1280	2^4*3^3*5*7*11*13*17*19
735.134.400	1344	2^6*3^3*5^2*7*11*13*17
1.102.701.600	1440	2^5*3^4*5^2*7*11*13*17
1.396.755.360	1536	2^5*3^3*5*7*11*13*17*19
2.095.133.040	1600	2^4*3^4*5*7*11*13*17*19
2.205.403.200	1680	2^6*3^4*5^2*7*11*13*17
2.327.925.600	1728	2^5*3^2*5^2*7*11*13*17*19
2.793.510.720	1792	2^6*3^3*5*7*11*13*17*19

4.5 Formulas

```
Soma de pg: = a1*(q^n - 1)/(q - 1)
Soma dos impares = n^2
Soma de i^2: = n(n+1)(2n+1)/6

Number theory:
gcd(a+k*b,b) = gcd(a,b)
phi(n) = #coprimos com n <=n
phi(n) >= log2(n)
```

```
phi(phi(n)) <= n/2
a^phi(n) == 1 mod n
a^-1 == a^(m-2) mod m
Conjectura de Goldbach's: todo numero par n > 2 pode ser representado com n
      = a + b onde a e b sao primos
Twin prime: existem infinitos pares p, p + 2 onde ambos sao primos
Legendre's: sempre tem um primo entre n^2 e (n+1)^2
Lagrange's: todo numero inteiro pode ser inscrito como a soma de 4
     quadrados
Wilson's: n eh primo quando (n-1)! mod n = n - 1
Mcnugget: Para dois coprimos x, y  a quantidade de inteiros que nao pode
     ser escrito como ax + by eh (x-1)(y-1)/2,
      o maior inteiro que nao consegue eh x*y-x-y


Geometria:
V+F=A+2
Formula de heron: sqrt(s*(s-a)*(s-b)*(s-c)), s = semiperimetro
Volume de esfera: 4/3pi*r^3
Area da esfera: 4pi*r^2
Volume tetraedro: l^3 * sqrt(2)/12
Projecao u em v = (u . v)/(v . v) * v
```

4.6 Bitset

```cpp
// Comando hash de codigo :w !sha256sum

// Bitset operations
__builtin_popcount(int x);
__builtin_popcountll(ll x);
const int SZ = 1e6;
bitset<SZ> b;
b.reset(); // 00 ... 00
b.set();   // 11 ... 11
b.flip();
b._Find_first(); // retorna SZ se nao tiver
b._Find_next(i);
b.to_ulong();
b.to_string();
b.count();

// rng
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
shuffle(begin(x),end(x),rng);
uniform_int_distribution<int>(0,x)(rng);
```

5 Graph

5.1 Articulation Pts

```cpp
int n, m;
const int mxn = 1e5 + 5;
vector<int> g[mxn];
int tin[mxn], low[mxn];
vector<int> art;
int timer = 1;

void dfs(int u, int p){
  tin[u] = timer++;
  low[u] = tin[u];
  int ch = 0;
  int fw = 0;
  for(int v : g[u]) if(v != p){
    if(tin[v]) // lowlink direta
      low[u] = min(tin[v],low[u]);
    else{
      dfs(v,u);
      fw++;
      low[u] = min(low[v],low[u]);
      ch = max(low[v],ch);
    }
  }
  if(u == p && fw > 1) art.push_back(u);
  else if(u != p && ch && tin[u] <= ch) art.push_back(u);
}
```

5.2 Bridges

```cpp
int n, m;
const int mxn = 1e5 + 5;
vector<int> g[mxn];
int tin[mxn], low[mxn];
vector<pii> bridges;
int timer = 1;

void dfs(int u, int p){
  tin[u] = timer++;
  low[u] = tin[u];
  int ch = 0;
  for(int v : g[u]) if(v != p){
    if(tin[v]) // lowlink direta
      low[u] = min(tin[v],low[u]);
    else{
      dfs(v,u);
      low[u] = min(low[v],low[u]);
      if(tin[u] < low[v]) bridges.push_back({u,v});
    }
  }
}
```

5.3 Dijkstra

```cpp
const int mx = 1e5+5;
using pii = pair<ll,int>;
vector<pii> g[mx];
const ll inf = 8e18;
ll dist[mx]; // setar tudo inf

void dijkstra(ll src){
  dist[src] = 0;
  priority_queue<pii,vector<pii>, greater<pii>> pq;
  pq.push({0,src});
  while(!pq.empty()){
    auto [d, u] = pq.top();
    pq.pop();
    if(d > dist[u]) continue;
    for(auto [w, v] : g[u]){
      ll cur = dist[u] + w;
      if(cur < dist[v]){
        dist[v] = cur;
        pq.push({cur,v});
      }
    }
  }
}
```

5.4 DSU

```
struct DSU{
        int n;
        vector<int> p,sz;
        DSU(int n) : n(n){
                p.resize(n);
                sz.resize(n,1);
                iota(begin(p), end(p), 0);
        }
        int size(int a){ return sz[root(a)]; }
        int root(int a){ return p[a] = (p[a] == a ? a : root(p[a])); }
        bool unite(int a, int b){
                int ra = root(a), rb = root(b);
                if(ra == rb) return 0;
                if(sz[ra] < sz[rb]) swap(ra,rb);
                p[rb] = ra;
                sz[ra] += sz[rb];
                return 1;
        }
};
```

5.5 Floyd Warshall

```
const int mxn = 505;
const ll inf = 1e18;
ll g[mxn][mxn]; // setar tudo infinito menos (i,i) como 0
int n;
void addEdge(int u, int v, ll w){
  g[u][v] = min(g[u][v],w);
  g[v][u] = min(g[v][u],w); // tirar se for 1 dir
}

void floyd(){
  for(int k = 0; k < n; k++) // << k
    for(int i = 0; i < n; i++)
      for(int j = 0; j < n; j++)
        if(g[i][k] + g[k][j] < g[i][j]) // cuida overflow aqui (inf)
          g[i][j] = g[i][k] + g[k][j];
}
```

5.6 Kosaraju

```
// Kosaraju
const int ms = 1e5 + 5;
vector<int> G[ms], Gt[ms];
vector<int> id, order, root;
vector<bool> vis;
int n;
void dfs1(int u){ // ordem de saida
  vis[u] = true;
  for(int v : G[u])
    if(!vis[v])
      dfs1(v);
  order.push_back(u);
}
void dfs2(int u, int idx){
  id[u] = idx;
  for(int v : Gt[u])
    if(id[v] == -1)
      dfs2(v,idx);
}
// retorna quantidade de componentes
int kosaraju(){
  vis.assign(n,false);
  id.assign(n,-1);
  for(int i = 0; i < n; i++)
    if(!vis[i])
      dfs1(i);
  reverse(begin(order),end(order));
  int idx = 0;
  for(int u : order)
    if(id[u] == -1)
      dfs2(u, idx++), root.push_back(u);
  return idx;
}
```

5.7 Kruskal

```
int n = 1e5;
DSU dsu = DSU(n+5);
using tp = tuple<ll,int,int>
vector<tp> edges(e);
for(auto& [w, u, v] : edges){
        cin >> u >> v >> w;
}
sort(begin(edges),end(edges));
ll cost = 0;
int cnt = 0;
for(auto [w, u, v] : edges){
        if(dsu.unite(u,v)){
                cost += w;
                cnt++;
        }
}
// if(cnt != n-1) cout << "IMPOSSIBLE" << br;
```

5.8 LCA

```
const int mxn = 2e5+5;
const int LOG = 22;
int n, q;
int tin[mxn], tout[mxn];
vector<vector<int>> up; // up[v][k] = 2^k-esimo ancestor de v
vector<int> g[mxn];
int lvl[mxn];
int timer = 0;
void dfs(int u, int p){
    tin[u] = ++timer;
    lvl[u] = lvl[p] + 1;
    up[u][0] = p;
    for(int i = 1; i <= LOG; i++){
        up[u][i] = up[ up[u][i-1] ][i-1];
    }
    for(int v : g[u]){
        if(v != u && !tin[v])
            dfs(v,u);
    }
    tout[u] = ++timer;
}

bool is_ancestor(int u, int v){
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int a, int b){
    if(is_ancestor(a,b)) return a;
    if(is_ancestor(b,a)) return b;
    for(int i = LOG; i >= 0; i--){
        if(!is_ancestor(up[a][i], b)){
            a = up[a][i];
        }
    }
    return up[a][0];
}
```

5.9 Dinic

```
//O(V^2 E), O(E sqrtV) in unit networks
template<typename T>
struct Edge {
  int to;
  T cap, flow;
  Edge(int to, T cap) : to(to), cap(cap), flow(0) {}
  T res() const { return cap - flow; }
};
template<typename T>
struct Dinic {
  using E = Edge<T>;
  int m = 0, n;
  vector<E> ed;
  vector<vector<int>> g;
  vector<int> dist, ptr;
  Dinic(int n) : n(n), g(n), dist(n), ptr(n) {}
  void add_edge(int u, int v, T cap) {
    if(u != v) {
      ed.emplace_back(v, cap);
      edges.emplace_back(u, 0);
      g[u].emplace_back(m++);
      g[v].emplace_back(m++);
    }
  }
  bool bfs(int s, int t) {
    fill(begin(dist), end(dist), n + 1);
    dist[s] = 0;
    queue<int> q({s});
    while(!q.empty()) {
      int u = q.front();
      q.pop();
      if(u == t) break;
      for(int id : g[u]) {
        E& e = edges[id];
        if(e.res() > 0 && dist[e.to] > dist[u] + 1) {
          dist[e.to] = dist[u] + 1;
          q.emplace(e.to);
        }
      }
    }
    return dist[t] != n + 1;
  }
  T max_flow(int s, int t) {
    T total = 0;
    while(bfs(s, t)) {
      fill(begin(ptr), end(ptr), 0);
      while(T flow = dfs(s, t, numeric_limits<T>::max())) {
        total += flow;
      }
    }
    return total;
  }
  bool cut(int u) const { return dist[u] == n + 1; }
};
//hash do de cima:
    c235a4a35cf8a9c14b5a906e6a2885474dc54aca7cd56c1513c803f6a91ead9b
//cut(u) returns where in the min-cut (S,T) the vertex u is
//false: u in S, true: u in T

// T dfs(int u, int t, T flow) {
//   if(u == t || flow == 0) {
//     return flow;
//   }
//   for(int& i = ptr[u]; i < (int)g[u].size(); ++i) {
//     E& e = edges[g[u][i]];
//     E& oe = edges[g[u][i] ^ 1];
//     if(dist[e.to] == dist[oe.to] + 1) {
//       T amt = min(flow, e.res());
//       if(T ret = dfs(e.to, t, amt)) {
//         e.flow += ret;
//         oe.flow -= ret;
//         return ret;
//       }
//     }
//   }
//   return 0;
// }
```

5.10 MCMF

```
template<typename Cap, typename Cost>
struct MCMF{
  const Cost INF = numeric_limits<Cost>::max();
  struct Edge {
    int to;
    Cap cap, flow;
    Cost cost;
    Edge(int to, Cap cap, Cost cost) : to(to), cap(cap), flow(0), cost(cost
        ) {}
    Cap res() const { return cap - flow; }
  };
  int m = 0, n;
  vector<Edge> edges;
  vector<vector<int>> g;
  vector<Cap> neck;
  vector<Cost> dist, pot;
  vector<int> from;
  MCMF(int n) : n(n), g(n), neck(n), pot(n) {}
  void add_edge(int u, int v, Cap cap, Cost cost) {
    if(u != v) {
      edges.emplace_back(v, cap, cost);
      edges.emplace_back(u, 0, -cost);
      g[u].emplace_back(m++);
      g[v].emplace_back(m++);
    }
  }
  void spfa(int s) {
    vector<bool> inq(n, false);
    queue<int> q({s});
    while(!q.empty()) {
      auto u = q.front();
      q.pop();
      inq[u] = false;
      for(auto e : g[u]) {
        auto ed = edges[e];
        if(ed.res() == 0) continue;
        Cost w = ed.cost + pot[u] - pot[ed.to];
        if(pot[ed.to] > pot[u] + w) {
          pot[ed.to] = pot[u] + w;
          if(!inq[ed.to]) {
            inq[ed.to] = true;
            q.push(ed.to);
          }
        }
      }
    }
  }
  bool dijkstra(int s, int t) {
    dist.assign(n, INF);
    from.assign(n, -1);
    neck[s] = numeric_limits<Cap>::max();
    using ii = pair<Cost, int>;
    priority_queue<ii, vector<ii>, greater<ii>> pq;
    pq.push({dist[s] = 0, s});
    while(!pq.empty()) {
      auto [d_u, u] = pq.top();
      pq.pop();
      if(dist[u] != d_u) continue;
      for(auto i : g[u]) {
```

```
        auto ed = edges[i];
        Cost w = ed.cost + pot[u] - pot[ed.to];
        if(ed.res() > 0 && dist[ed.to] > dist[u] + w) {
          from[ed.to] = i;
          pq.push({dist[ed.to] = dist[u] + w, ed.to});
          neck[ed.to] = min(neck[u], ed.res());
        }
      }
    }
    return dist[t] < INF;
  }
  pair<Cap, Cost> mcmf(int s, int t, Cap k = numeric_limits<Cap>::max()) {
    Cap flow = 0;
    Cost cost = 0;
    spfa(s);
    while(flow < k && dijkstra(s, t)) {
      Cap amt = min(neck[t], k - flow);
      for(int v = t; v != s; v = edges[from[v] ^ 1].to) {
        cost += edges[from[v]].cost * amt;
        edges[from[v]].flow += amt;
        edges[from[v] ^ 1].flow -= amt;
      }
      flow += amt;
      fix_pot();
    }
    return {flow, cost};
  }
  void fix_pot() {
    for(int u = 0; u < n; ++u) {
      if(dist[u] < INF) {
        pot[u] += dist[u];
      }
    }
  }
};
// hash: 8615758555a5fbae52f7e33dad88b6571dcf9bbb7841fb78589debed2a13d424
```

5.11 Policy Based

```
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including
    tree_order_statistics_node_update
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
ordered_set X;
X.insert(1); X.find_by_order(0); // Acha a key na ordem Y
X.order_of_key(-5); // Acha a ordem da key Y
end(X), begin(X);
```

5.12 2Sat

```
#define PB push_back
// usar ~ para negacao
/*
regras logica
A->B = ~B->~A (contrapositiva)
A->B = ~A | B (lei da implicacao)
~(A|B) = ~A & ~B (de morgan)
A & (B|C) = (A&B) | (A&C) (distributiva)
*/

struct TwoSat{
  int n;
  vector<vector<int>> G, Gt;
  vector<int> id, order, ans;
  vector<bool> vis;
  TwoSat(){}
  TwoSat(int n) : n(n){
    G.resize(2*n);
    Gt.resize(2*n);
    id.assign(2*n,-1);
    ans.resize(n);
  }
  // negativos na esquerda
  void add_edge(int u, int v){
    u = (u < 0 ? -1-u : u + n);
    v = (v < 0 ? -1-v : v + n);
    G[u].PB(v);
    Gt[v].PB(u);
  }
  void add_or(int a, int b){
    add_edge(~a,b);
    add_edge(~b,a);
  }
  // Apenas algum ser 1
  void add_xor(int a, int b){
    add_or(a,b);
    add_or(~a,~b);
  }
  // set(a) = 1, set(~a) = 0
  void set(int a){ // (a|a)
    add_or(a,a);
  }
  // Mesmo valor
  void add_xnor(int a, int b) {
    add_xor(~a,b);
  }
  void dfs1(int u){
    vis[u] = true;
    for(int v : G[u])
      if(!vis[v])
        dfs1(v);

    order.PB(u);
  }
  void dfs2(int u, int idx){
    id[u] = idx;
    for(int v : Gt[u])
      if(id[v] == -1)
        dfs2(v,idx);
  }
  void kosaraju(){
    vis.assign(2*n,false);
    for(int i = 0; i < 2*n; i++)
      if(!vis[i])
        dfs1(i);
    reverse(begin(order),end(order));
    int idx = 0;
    for(int u : order){
      if(id[u] == -1)
        dfs2(u, idx++);
    }
  }
  bool satisfiable(){
    kosaraju();
    for(int i = 0; i < n; i++){
      if(id[i] == id[i + n]) return false;
      ans[i] = (id[i] < id[i + n]);
    }
    return true;
  }
};
```

6 Math

6.1 Extended Euclidean

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

// inverso modular de a
int inv, y;
int g = gcd(a,mod,inv,y);
inv = (inv % m + m) % m;
```

6.2 CRT

```
ll euclid(ll a, ll b, ll&x ,ll&y){
  if(!b) return x = 1, y = 0, a;
  ll d = euclid(b, a % b, y , x);
  return y -= a/b * x, d;
}
ll crt(vector<ll>& rem, vector<ll>& mod){
  int n = rem.size();
  if(n == 0) return 0;
  ll ans = rem[0], m = mod[0];
  for(int i = 1; i < n; i++){
    ll x,y;
    ll g = euclid(mod[i],m,x,y);
    // if((ans - rem[i]) % g != 0) return -5;
    assert((ans - rem[i]) % g == 0);
    ans = ans + 1LL*(rem[i]-ans)*(m/g)*y;
    m = (mod[i]/g)*(m/g)*g;
  }
  return ans;
}
```

6.3 Factorization

```
// Factorization
vector<pii> getFact(int n){
        vector<pii> primes;
        for(int p = 2; p*p <= n; p++){
                if(n % p == 0){
                        int exp = 0;
                        while(n % p == 0){
                                exp++;
                                n /= p;
                        }
                        primes.PB({p,exp});
                }
        }
        if(n > 1) primes.PB({n,1});
        return primes;
}
```

6.4 Division Trick

```
for(int l = 1, r; l <= n; l = r + 1) {
  r = n / (n / l);
  // n / i has the same value for l <= i <= r
  // O(sqrt(n)) different floor(n/i) values
}
```

6.5 Fraction

```
// de tfg

template<class T>
T gcd(T a, T b) { return b == 0 ? a : gcd(b, a % b); }

template<class T>
struct Frac {
        T p, q;
        Frac() {
                p = 0, q = 1;
        }
        Frac(T x) {
                p = x;
                q = 1;
        }
        Frac(T a, T b) {
                if(b == 0) {
                        a = 0;
                        b = 1;
                }
                p = a;
                q = b;
                fix();
        }
        Frac<T> operator + (Frac<T> o) const { return Frac(p * o.q + o.p *
            q, q * o.q); }
        Frac<T> operator - (Frac<T> o) const {return Frac(p * o.q - o.p * q
            , q * o.q); }
        Frac<T> operator * (Frac<T> o) const { return Frac(p * o.p, q * o.q
            ); }
        Frac<T> operator / (Frac<T> o) const { return Frac(p * o.q, q * o.p
            ); }

        void fix() {
                if(q < 0) {
                        q = -q;
                        p = -p;
                }
                auto g = gcd(std::max(p, -p), q);
                p /= g;
                q /= g;
        }

        bool operator < (Frac<T> o) const { return ((*this) - o).p < 0; }
        bool operator > (Frac<T> o) const { return ((*this) - o).p > 0; }

        friend ostream& operator << (ostream &os, const Frac<T> &f) {
                return os << f.p << '/' << f.q;
        }
        friend istream& operator >> (istream &is, Frac<T> &f) {
                char trash;
                return is >> f.p >> trash >> f.q;
        }
};
```

6.6 Gaussian Elimination

```
template<typename T>
struct GaussianElimination {
  // may change if using doubles
  static bool cmp(const T& a, const T& b) { return a == b; }
  vector<vector<T>> a, inv;
  vector<int> pivot;
  GaussianElimination(const vector<vector<T>> a = {}) : a(a) {}
  void add_equation(const vector<T>& equation) {
    a.emplace_back(equation);
  }
  /*
    pair(0, ans) impossible
    pair(1, ans) one solution
    pair(2, ans) infinite solutions
  */
  pair<int, vector<T>> solve_system(bool findInverse = false) {
    int n = (int)a.size();
    int m = (int)a[0].size() - 1;
    pivot.assign(m, -1);
    if(findInverse) {
      inv.assign(n, vector<T>(n));
      for(int i = 0; i < n; ++i) inv[i][i] = T(1);
    }
    for(int col = 0, row = 0; col < m && row < n; ++col) {
      int sel = -1;
      for(int i = row; i < n; ++i) {
        if(!cmp(a[i][col], 0)) {
          sel = i;
          break;
        }
      }
      if(sel == -1) continue;
      for(int j = col; j <= m; ++j) {
        swap(a[row][j], a[sel][j]);
      }
      if(findInverse) swap(inv[row], inv[sel]);
      for(int i = 0; i < n; ++i) {
        if(i == row) continue;
        T c = a[i][col] / a[row][col];
        for(int j = col; j <= m; ++j) {
          a[i][j] -= c * a[row][j];
        }
        if(!findInverse) continue;
        for(int j = 0; j < n; ++j) {
          inv[i][j] -= c * inv[row][j];
        }
      }
      pivot[col] = row++;
    }
    vector<T> ans(m);
    for(int j = 0; j < m; ++j) {
      if(pivot[j] == -1) continue;
      //normalize pivots
      int i = pivot[j];
      for(int k = j + 1; k <= m; ++k) {
        a[i][k] /= a[i][j];
      }
      if(findInverse) {
        for(int k = 0; k < n; ++k) {
          inv[i][k] /= a[i][j];
        }
      }
      a[i][j] = T(1);
      ans[j] = a[i][m];
    }
    for(int i = 0; i < n; ++i) {
      T value(0);
      for(int j = 0; j < m; ++j) {
        value += ans[j] * a[i][j];
      }
      if(!cmp(value, a[i][m])) return make_pair(0, ans);
    }
    for(int j = 0; j < m; ++j) {
      if(pivot[j] == -1) return make_pair(2, ans);
    }
    return make_pair(1, ans);
  }
};
```

6.7 Fastexp

```
// Fast Exp
const ll mod = 1e9+7;


ll fexpll(ll a, ll n){
        ll ans = 1;
        while(n){
                if(n & 1) ans = (ans * a) % mod;
                a = (a * a) % mod;
                n >>= 1;
        }
        return ans;
}
// matriz quadrada
class Matrix{
        public:
        vector<vector<ll>> mat;
        int m;
        Matrix(int m): m(m){
                mat.resize(m);
                for(int i = 0; i < m; i++) mat[i].resize(m,0);
        }
        Matrix operator * (const Matrix& rhs){
                Matrix ans = Matrix(m);
                for(int i = 0; i < m; i++)
                        for(int j = 0; j < m; j++)
                                for(int k = 0; k < m; k++)
                                        ans.mat[i][j] = (ans.mat[i][j] + (
                                            mat[i][k] * rhs.mat[k][j]) %
                                            mod) % mod;
                return ans;
        }
};

Matrix fexp(Matrix a, ll n){
        int m = a.m;
        Matrix ans = Matrix(m);
        for(int i = 0; i < m; i++) ans.mat[i][i] = 1;
        while(n){
                if(n & 1) ans = ans * a;
                a = a * a;
                n >>= 1;
        }
        return ans;
}
```

6.8 Pollard Rho

```
// from: https://github.com/kth-competitive-programming/kactl
typedef unsigned long long ull;
ull modmul(ull a, ull b, ull M) {
        ll ret = a * b - M * ull(1.L / M * a * b);
        return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
        ull ans = 1;
        for (; e; b = modmul(b, b, mod), e /= 2)
                if (e & 1) ans = modmul(ans, b, mod);
```

```
		return ans;
}
bool isPrime(ull n) {
	if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
	ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
	    s = __builtin_ctzll(n-1), d = n >> s;
	for (ull a : A) {
		ull p = modpow(a%n, d, n), i = s;
		while (p != 1 && p != n - 1 && a % n && i--)
			p = modmul(p, p, n);
		if (p != n-1 && i != s) return 0;
	}
	return 1;
}
ull pollard(ull n) {
	ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
	auto f = [&](ull x) { return modmul(x, x, n) + i; };
	while (t++ % 40 || __gcd(prd, n) == 1) {
		if (x == y) x = ++i, y = f(x);
		if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
		x = f(x), y = f(f(y));
	}
	return __gcd(prd, n);
}
vector<ull> factor(ull n) {
	if (n == 1) return {};
	if (isPrime(n)) return {n};
	ull x = pollard(n);
	auto l = factor(x), r = factor(n / x);
	l.insert(l.end(), begin(r),end(r));
	return l;
}
// hash: 3782d14edf6f6c81aa19f1dbbf0b31d3fa0b82704f5aeb2f2554fd3bc8404702
```

6.9 Phi

```
const int LIM = 1e6+5;
int phi[LIM];
void sieve(){
	iota(phi, phi + LIM, 0);
	for(int i = 2; i < LIM; i++){
		if(phi == i){
			for(int j = i; j < LIM; j += i){
				phi[j] -= phi[j] / i;
			}
		}
	}
}

template<typename T>
T phi(T n) {
  T ans = n;
  for(T p = 2; p * p <= n; p++) {
    if(n % p == 0) {
      ans -= ans / p;
      while(n % p == 0) {
        n /= p;
      }
    }
  }
  if(n > 1) {
    ans -= ans / n;
  }
  return ans;
}
```

7 String

7.1 RabinKarp

```
// Rabin Karp
/*
Some Big Prime Numbers:
37'139'213
const ll MOD1 = 131'807'699; -> Big Prime Number for hash 1
const ll MOD1 = 127'065'427; -> Big Prime Number for hash 2
const ll base = 127;         -> Random number larger than the Alphabet
*/
const ll base = 997;
const ll mod[] ={1000000007, 1000000009};
const int MXSZ = 1e6+2;
ll pot[2][MXSZ];
// Lembrar de chamar BUILDPOTS!!!!!
// getkey eh INCLUSIVO
void buildPots(){
  pot[0][0] = 1;
  pot[1][0] = 1;
  for(int j = 0; j < 2; j++)
    for(int i = 1; i < MXSZ; i++)
      pot[j][i] = (pot[j][i-1]*base) % mod[j];
}
class RabinKarp{
public:
  string s;
  int sz;
  vector<ll> h[2];
  RabinKarp(){}
  RabinKarp(const string& str): s(str){
    sz = str.size();
    h[0].resize(sz+1);
    h[1].resize(sz+1);
    h[0][0] = s[0], h[1][0] = s[0];
    for(int j = 0; j < 2; j++)
      for(int i = 1; i < sz; i++)
        h[j][i] = ((h[j][i-1]*base)+s[i])%mod[j];
  }
  ll getKey(int l, int r){
    ll x = h[0][r], y = h[1][r];
    if(l > 0){
      x = (((x - pot[0][r-l+1]*h[0][l-1])%mod[0] + mod[0])%mod[0]);
      y = (((y - pot[1][r-l+1]*h[1][l-1])%mod[1] + mod[1])%mod[1]);
    }
    return (x<<32LL)|y;
  }
};
// hash de buildpots pra baixo: 97523
    f3c3aa5a2f0ae00021355eb26036f231e731b032edfdb6bd96153886ca7
```

7.2 Trie

```
int trie[ms][sigma], terminal[ms], z = 1;

void insert(string &p) {
  int cur = 0;
  for(int i = 0; i < p.size(); i++) {
    int id = p[i]-'a';
    if(!trie[cur][id]) {
      trie[cur][id] = z++;
    }
    cur = trie[cur][id];
  }
  terminal[cur]++;
```

```
}

int count(string &p) {
  int cur = 0;
  for(int i = 0; i < p.size(); i++) {
    int id = p[i]-'a';
    if(!trie[cur][id]) {
      return false;
    }
    cur = trie[cur][id];
  }
  return terminal[cur];
}
```

7.3 KMP

```
vector<int> getBorder(string str) {
  int n = str.size();
  vector<int> border(n, -1);
  for(int i = 1, j = -1; i < n; i++) {
    while(j >= 0 && str[i] != str[j + 1]) {
      j = border[j];
    }
    if(str[i] == str[j + 1]) {
      j++;
    }
    border[i] = j;
  }
  return border;
}
int matchPattern(const string &txt, const string &pat, const vector<int> &
   border) {
  int freq = 0;
  for(int i = 0, j = -1; i < txt.size(); i++) {
    while(j >= 0 && txt[i] != pat[j + 1]) {
      j = border[j];
    }
    if(pat[j + 1] == txt[i]) {
      j++;
    }
    if(j + 1 == (int) pat.size()) {
      //found occurence
      freq++;
      j = border[j];
    }
  }
  return freq;
}
```

7.4 Z Function

```
vector<int> Zfunction(string &s){
        int n = s.size();
        vector<int> z (n, 0);
        for(int i=1, l=0, r=0; i<n;  i++)       {
                if(i <= r) z[i] = min(z[i-l], r-i+1);
                while(z[i] + i < n && s[z[i]] == s[i+z[i]]) z[i]++;
                if(r < i+z[i]-1) l = i, r = i+z[i]-1;
        }
        return z;
}
```

7.5 Aho-Corasick

```
struct AhoType {
  static const int ALPHA = 26;
  static int f(char c) { return c - 'A'; } // ver se ta maiusculo ou
     minusculo aqui
};
template<typename AhoType>
struct AhoCorasick {
  struct Node {
    int nxt[AhoType::ALPHA] {};
    int p = 0, ch = 0, len = 0;
    int link = 0;
    int occ_link = 0;
    Node(int p = 0, int ch = 0, int len = 0) : p(p), ch(ch), len(len) {}
  };
  vector<Node> tr;
  AhoCorasick() : tr(1) {}
  template<typename Iterator>
  void add_word(Iterator first, Iterator last) {
    int cur = 0, len = 1;
    for(; first != last; ++first) {
      auto ch = AhoType::f(*first);
      if(tr[cur].nxt[ch] == 0) {
        tr[cur].nxt[ch] = int(tr.size());
        tr.emplace_back(cur, ch, len);
      }
      cur = tr[cur].nxt[ch];
      ++len;
    }
    tr[cur].occ_link = cur;
  }
  void build() {
    vector<int> bfs(int(tr.size()));
    int s = 0, t = 1;
    while(s < t) {
      int v = bfs[s++], u = tr[v].link;
      if(tr[v].occ_link == 0) {
        tr[v].occ_link = tr[u].occ_link;
      }
      for(int ch = 0; ch < AhoType::ALPHA; ++ch) {
        auto& nxt = tr[v].nxt[ch];
        if(nxt == 0) {
          nxt = tr[u].nxt[ch];
        } else {
          tr[nxt].link = v > 0 ? tr[u].nxt[ch] : 0;
          bfs[t++] = nxt;
        }
      }
    }
  }
  template<typename Iterator>

  vector<pair<int,int>> get_all_matches(Iterator first, Iterator last)
      const {
        vector<pair<int,int>> occs;
    for(int cur = 0, i = 0; first != last; ++i, ++first) {
      auto ch = AhoType::f(*first);
      cur = tr[cur].nxt[ch];
      for(int v = tr[cur].occ_link; v > 0; v = tr[tr[v].link].occ_link) {
              // i = pos text, v = state
              occs.push_back({1+i-tr[v].len, i});
      }
    }
        return occs;
  }
  template<typename T>
  int get_next(int cur, T ch) const { return tr[cur].nxt[AhoType::f(ch)]; }
};
```