Contents

Estructuras de Datos

Segment Tree

```
class SegmentTree<T>(
    val n: Int,
    val operation: (T, T) -> T,
    val neutral: T
) {
    var size: Int
    var st: MutableList<T>

    init {
        size = 1
        while (size <= n) size *= 2
        st = MutableList(2 * size) { neutral }
    }

    // Inicializa el segmento del árbol con los valores dados
    fun init(arr: List<T>) {
        for (i in 0 until n) {
            st[i + size] = arr[i]
        }
        for (i in size - 1 downTo 1) {
            st[i] = operation(st[2 * i], st[2 * i + 1])
        }
    }

    // Actualiza un valor en la posición p
    fun update(p: Int, value: T) {
        var index = p + size
        st[index] = value
        while (index > 1) {
            index /= 2
            st[index] = operation(st[2 * index], st[2 * index + 1])
        }
    }

    // Realiza una consulta sobre el rango [lq, rq]
    fun query(lq: Int, rq: Int): T {
        var left = lq + size
        var right = rq + size
        var lres = neutral
        var rres = neutral

        while (left < right) {
            if (left % 2 == 1) {
                lres = operation(lres, st[left])
                left++
            }
            if (right % 2 == 1) {
                right--
                rres = operation(st[right], rres)
            }
            left /= 2
            right /= 2
        }
        return operation(lres, rres)
    }
}
```

Fenwick Tree

```
class Fenwick<T>(
    val n: Int,
    val operation: (T, T) -> T,
    val neutral: T){
    var ft: MutableList<T>

    init {
        ft = MutableList(n + 1) { neutral }
    }

    fun update(p: Int, value: T) {
        var index = p
        while (index <= n) {
            ft[index] = operation(ft[index], value)
            index += index and -index
        }
    }

    fun query(r: Int): T {
        var index = r
        var res = neutral
        while (index > 0) {
            res = operation(res, ft[index])
            index -= index and -index
        }
        return res
    }
}
```

Grafos

BFS

```
fun bfs(
    graph: List<List<Int>>,
    start: Int
) : List<Int> {
    val n = graph.size
    val min_dist = MutableList(n) { -1 }
    val q = mutableListOf<Int>()
    q.add(start)
    min_dist[start] = 0
    var qi = 0
    while (qi < q.size){
        val u = q[qi]
        qi++
        for (v in graph[u]){
            if (min_dist[v] == -1){
                min_dist[v] = min_dist[u] + 1
                q.add(v)
            }
        }
    }
    return min_dist
}
```

Dijkstra

```
import java.util.PriorityQueue

fun Dijkstra(
    grafo: List<List<Pair<Int,Int>>>,
    inicio: Int
) : Pair<List<Int>, List<Int>> {
    val n = grafo.size
    var dist = MutableList(n) { Int.MAX_VALUE }
    var padre = MutableList(n) { -1 }
    dist[inicio] = 0
    val pq = PriorityQueue<Pair<Int,Int>>(compareBy { -it.second })
    pq.add(Pair(inicio, 0))
    while (pq.isNotEmpty()){
        val (u, d) = pq.poll()
        if (d > dist[u]) continue
        for ((v, w) in grafo[u]){
            if (dist[u] + w < dist[v]){
                dist[v] = dist[u] + w
                padre[v] = u
                pq.add(Pair(v, dist[v]))
            }
        }
    }
    return Pair(dist, padre)
}
```

Bellman-Ford

```
fun BellmanFord(
    grafo: List<List<Pair<Int,Int>>>,
    inicio: Int,
    largo: Int
) : List<List<Int>> {
    val n = grafo.size
    var dist = MutableList(largo+1) { MutableList(n) { Int.MAX_VALUE} }
    dist[0][inicio] = 0
    for (k in 0 until largo){
        for (u in 0 until n){
            for ((v, w) in grafo[u]){
                if( dist[k][u] != Int.MAX_VALUE)
                    dist[k+1][v] = minOf(dist[k+1][v], dist[k][u] + w)
            }
        }
    }
    return dist
}
```

Floyd-Warshall

```
fun FloydWarshall(
    matriz: List<List<Int>>
) : MutableList<MutableList<Int>> {
    val n = matriz.size
    var dist = matriz.map{ it.toMutableList() }.toMutableList()
    for (k in 0 until n){
        for (i in 0 until n){
            for (j in 0 until n){
                dist[i][j] = minOf(dist[i][j], dist[i][k] + dist[k][j])
            }
        }
    }
    return dist
}
```

Kruskal

```
fun Kruskal(g: List<Triple<Int,Int,Int>>, n : Int) : Pair<Int, List<Int
    >>{
    var uf = MutableList(n){i -> i}
    fun find(x: Int) : Int{
        if (uf[x] == x) return x
        uf[x] = find(uf[x])
        return uf[x]
    }

    fun union(x: Int, y: Int){
        uf[find(x)] = find(y)
    }

    val aristas = MutableList(n){i -> i}.sortedBy{g[it].third}

    var valor = 0
    var arbol = mutableListOf<Int>()
    for (ar in aristas){
        val (u,v,c) = g[ar]
        if (find(u) != find(v)){
            union(u,v)
            valor += c
            arbol.add(ar)
        }
    }

    return Pair(valor, arbol)
}
```

Ancestro común menor

```
class LCA(
    arbol: List<List<Int>>,
    raiz: Int
){
    var K: Int
    var padre: MutableList<MutableList<Int>>
    var prof: MutableList<Int>
    init {
        val n = arbol.size
        K = 1
        while ((1 shl K) < n) K++
        padre = MutableList(K) { MutableList(n) { -1 } }
        prof = MutableList(n) { -1 }

        fun dfs(u: Int, p: Int){
            padre[0][u] = p
            for (v in arbol[u]){
                if (v == p) continue
                prof[v] = prof[u] + 1
                dfs(v, u)
            }
        }

        dfs(raiz, -1)
        prof[raiz] = 0

        for (k in 1 until K){
            for (u in 0 until n){
                if (padre[k-1][u] != -1){
                    padre[k][u] = padre[k-1][padre[k-1][u]]
                }
            }
        }
    }

    fun lca(uu: Int, vv: Int) : Int {
        var u = uu
        var v = vv

        if (prof[u] < prof[v]) return lca(v, u)
        for (k in K-1 downTo 0){
            if (prof[u] - (1 shl k) >= prof[v]){
                u = padre[k][u]
            }
        }
        if (u == v) return u
        for (k in K-1 downTo 0){
            if (padre[k][u] != padre[k][v]){
                u = padre[k][u]
                v = padre[k][v]
            }
        }
        return padre[0][u]
    }
}
```

Strings

Bordes (KMP)

```
fun bordes(s: String): List<Int>{
    val n = s.length
    val b = MutableList(n+1) { -1 }
    var j = -1;
    for (i in 0 until n){
        while (j >= 0 && s[i] != s[j]){
            j = b[j]
        }
        j++
        b[i+1] = j
    }
    return b
}
```

Función Z

```
fun z(s: String): List<Int>{ // z[i] = max k: s[0,k) == s[i,i+k)
    val n = s.length
    val z = MutableList(n) { 0 }
    var l = 0
    var r = 0
    for (i in 1 until n){
        if (i <= r) z[i] = minOf(r-i+1, z[i-l])
        while (i+z[i] < n && s[z[i]] == s[i+z[i]]) z[i]++
        if (i+z[i]-1 > r){
            l = i
            r = i+z[i]-1
        }
    }
    return z
}
```

Manacher

```
fun Manacher(s: String): Pair<List<Int>, List<Int>>{
    // (d1, d2) = (impares, pares) palindromes
    val n = s.length
    val d1 = MutableList(n) { 0 }
    val d2 = MutableList(n) { 0 }
    var l = 0
    var r = -1
    for (i in 0 until n){
        var k = if (i > r) 1 else minOf(d1[l+r-i], r-i+1)
        while (i-k >= 0 && i+k < n && s[i-k] == s[i+k]) k++
        d1[i] = k--
        if (i+k > r){
            l = i-k
            r = i+k
        }
    }
    l = 0
    r = -1
    for (i in 0 until n){
        var k = if (i > r) 0 else minOf(d2[l+r-i+1], r-i+1)
        while (i-k-1 >= 0 && i+k < n && s[i-k-1] == s[i+k]) k++
        d2[i] = k--
        if (i+k > r){
            l = i-k-1
            r = i+k
        }
    }
    return Pair(d1, d2)
}
```

Suffix Array

```
fun RB(x : Int, n : Int, r: List<Int>) : Int{
    if(x < n) return r[x]
    else return 0
}

fun csort(sa: MutableList<Int>, r: MutableList<Int>, k : Int){
    val n = sa.size
    var f = MutableList(maxOf(255,n)){0}
    var t = MutableList(n){0}
    for (i in 0 until n) f[RB(i+k,n,r)]++
    var sum = 0
    for (i in 0 until f.size){
        var v = f[i]
        f[i] = sum
        sum += v
    }
    for (i in 0 until n){
        t[f[RB(sa[i]+k,n,r)]++] = sa[i]
    }

    for (i in 0 until n) sa[i] = t[i]
}

fun suffix_array(s0: String): List<Int>{
    val s = s0 + '\u0000'
    val n = s.length
    var rank: Int
    var sa = MutableList(n){it}
    var r = MutableList(n){it -> s[it].code}
    var t = MutableList(n){0}
    var k = 1
    while (k<n){
        csort(sa,r,k)
        csort(sa,r,0)
        t[sa[0]] = 0
        rank = 0
        for (i in 1 until n){
            if(r[sa[i]] != r[sa[i-1]] || RB(sa[i]+k,n,r) != RB(sa[i-1]+
                k,n,r)) rank++
            t[sa[i]] = rank
        }
        for (i in 0 until n) r[i] = t[i]
        if (r[sa[n-1]]==n-1) break
        k *= 2

        println("k = $k")
    }
    return sa
}
```

LCP (Estructura)

```
fun computar_lcp(s0: String, sa: List<Int>): MutableList<Int>{
    val s = s0 + '\u0000'
    val n = s.length
    var L = 0
    var lcp = MutableList(n){0}
    var plcp = MutableList(n){0}
    var phi = MutableList(n){0}

    phi[sa[0]] = -1
    for (i in 1 until n) phi[sa[i]] = sa[i-1]
    for (i in 0 until n){
        if (phi[i] == -1){
            plcp[i] = 0
            continue
        }
        while (s[i+L] == s[phi[i]+L]) L++
        plcp[i] = L
        L = maxOf(L-1,0)
    }
    for (i in 0 until n) lcp[i] = plcp[sa[i]]
    return lcp
}
```

Duval

```
// Dada una string $s$ devuelve la Lyndon decomposition en tiempo
// lineal usando el algoritmo de Duval. Factoriza $s$ como
// $s_1 s_2 \ldots s_k$ con $s_1 \geqq s_2 \geqq \cdots \geqq s_k$
// y tal que $s_i$ es Lyndon, esto es, es su menor rotación.
fun Duval(s: String) : List<String>{
    val n = s.length
    var i = 0
    val ans = mutableListOf<String>()
    while (i < n){
        var j = i + 1
        var k = i
        while (j < n && s[k] <= s[j]){
            if (s[k] < s[j]) k = i
            else k++
            j++
        }
        while (i <= k){
            ans.add(s.substring(i until i+j-k))
            i += j-k
        }
    }
    return ans
}
// Obtener la mínima rotaciónn de $s$: en la descomposición de
// Lyndon de $s^2$ es el último $i<|s|$ con el que empieza una
// Lyndon.
```

Hashing

```
const val P: Long = 1777771
val MOD: List<Long> = listOf(999727999, 1070777777)
val PI: List<Long> = listOf(325255434, 10018302)

class Hashing(c: Char) {
    val h: MutableList<Long>
    val p: MutableList<Long>
    val pi: MutableList<Long>
    init {
        h = MutableList(PI.size) { i -> c.code * P % MOD[i]}
        p = MutableList(PI.size) { P }
        pi = PI.toMutableList()
    }

    // Agrega un prefijo : H(s1) + H(s2) = H(s2s1)
    operator fun plus(h2: Hashing) : Hashing {
        val ans = Hashing('a')
        for (i in 0 until PI.size){
            ans.h[i] = (h[i] * h2.p[i] + h2.h[i]) % MOD[i]
            ans.p[i] = p[i] * h2.p[i] % MOD[i]
            ans.pi[i] = pi[i] * h2.pi[i] % MOD[i]
        }
        return ans
    }
    // Elimina un prefijo
    operator fun minus(h2: Hashing) : Hashing {
        val ans = Hashing('a')
        for (i in 0 until PI.size){
            ans.h[i] = (h[i] - h2.h[i] + MOD[i]) % MOD[i] * h2.pi[i] %
                 MOD[i]
            ans.p[i] = p[i] * h2.pi[i] % MOD[i]
            ans.pi[i] = pi[i] * h2.p[i] % MOD[i]
        }
        return ans
    }
    /// O simplemente comparar h1.h == h2.h en vez de h1==h2
    override fun equals(h2: Any?): Boolean {
        if (h2 !is Hashing) return false
        return h == h2.h
    }
}
fun hash_neutro() : Hashing{
    var ans = Hashing('a')
    for (i in 0 until PI.size){
        ans.h[i] = 0
        ans.p[i] = 1
        ans.pi[i] = 1
    }
    return ans
}
class StringHasher(s: String){
    val h: MutableList<Hashing>
    init {
        h = MutableList(s.length+1) {
            if (it == 0) hash_neutro()
            else Hashing(s[it-1])
        }
        for (i in 1 until s.length+1){
            h[i] = h[i] + h[i-1]
        }
    }
    // Hash de s[l,r)
    fun hash(l: Int, r: Int) : Hashing {
        return h[r] - h[l]
    }
}
```

Matemáticas

Identidades

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

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

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

$F_{2n+1} = F_n^2 + F_{n+1}^2$

$F_{2n} = F_{n+1}^2 - F_{n-1}^2$

$\sum_{i=1}^n F_i = F_{n+2} - 1$

$F_{n+i}F_{n+j} - F_nF_{n+i+j} = (-1)^n F_iF_j$

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

$\sum_{i=1}^n i^2 = \frac{n\cdot(n+1)\cdot(2n+1)}{6}$

$\sum_{i=1}^n i^3 = \left(\frac{n\cdot(n+1)}{2}\right)^2$

$\sum_{i=1}^n i^4 = \frac{n\cdot(n+1)\cdot(2n+1)\cdot(3n^2+3n-1)}{12}$

$\sum_{i=1}^n i^5 = \left(\frac{n\cdot(n+1)}{2}\right)^2 \cdot \frac{2n^2+2n-1}{3}$

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

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

(Möbius Inv. Formula) Let

$$g(n) = \sum_{d|n} f(d), \text{ then}$$

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

Convolución rápida (FFT y Karatsuba)

```
import kotlin.math.round

data class Complex(val r: Double, val i: Double){
    operator fun plus(x: Complex) = Complex(r + x.r, i + x.i)
    operator fun minus(x: Complex) = Complex(r - x.r, i - x.i)
    operator fun times(x: Complex) = Complex(r*x.r - i*x.i, r*x.i + i*x
         .r)
    operator fun div(x: Double) = Complex(r/x, i/x)
}
class FFT(lg0: Int){
    val lg = lg0+1
    val n = 1 shl lg
    val w = MutableList(n+1){Complex(0.0, 0.0)}

    init {
        val ang0 = 2.0 * Math.PI / n.toDouble()
        for (i in 0 until n+1){
            val ang = ang0 * i
            w[i] = Complex(Math.cos(ang), Math.sin(ang))
        }
    }

    fun fft(a: List<Complex>, inv: Boolean = false) : List<Complex>{
        val p = MutableList(n) { a[Integer.reverse(it) ushr (32 - lg)]
             }

        var len = 2
        while (len <= n) {
            val step = n / len
            for (i in 0 until n step len) {
                for (j in 0 until len / 2) {
                    val u = p[i + j]
                    val v = p[i + j + len / 2] * if (inv) w[n - j *
                         step] else w[j * step]
                    p[i + j] = u + v
                    p[i + j + len / 2] = u - v
                }
            }
            len *= 2
        }

        if (inv) {
            for (i in 0 until n) {
                p[i] = p[i] / n.toDouble()
            }
        }

        return p
    }

    fun multiply(a: List<Long>, b: List<Long>) : List<Long>{
        val a_c = a.map { Complex(it.toDouble(), 0.0) } + MutableList(n
             -a.size){Complex(0.0, 0.0)}
        val b_c = b.map { Complex(it.toDouble(), 0.0) } + MutableList(n
             -b.size){Complex(0.0, 0.0)}
        val fa = fft(a_c)
        val fb = fft(b_c)
        val fc = MutableList(n){fa[it] * fb[it]}
        val c = fft(fc, true)
        return c.map { round(it.r).toLong() }
    }
}


fun Karatsuba(a : List<Long>, b : List<Long>) : List<Long>{
    val m = maxOf(a.size, b.size)
    val n = 1 shl (32 - Integer.numberOfLeadingZeros(m - 1))
    val aa = a + MutableList(n - a.size) { 0.toLong() }
    val bb = b + MutableList(n - b.size) { 0.toLong() }
    return karatsuba(aa, bb)
}

fun karatsuba(a: List<Long>, b: List<Long>): List<Long> {
    if (a.size <= 16) { // Reducir el tamaño de la condición base
        val c = MutableList(2 * a.size - 1) { 0L }
        for (i in a.indices) {
            for (j in b.indices) {
                c[i + j] += a[i] * b[j]
            }
        }
        return c
    }

    val n = a.size
    val k = n / 2
    val a0 = a.subList(0, k)
    val a1 = a.subList(k, n)
    val b0 = b.subList(0, k)
    val b1 = b.subList(k, n)

    val z2 = karatsuba(a1, b1)
    val z0 = karatsuba(a0, b0)
    val a0a1 = List(k) { a0[it] + a1[it] }
    val b0b1 = List(k) { b0[it] + b1[it] }
    val z1 = karatsuba(a0a1, b0b1)

    val result = MutableList(2 * n) { 0L }
    for (i in z0.indices) result[i] += z0[i]
    for (i in z2.indices) result[i + n] += z2[i]
    for (i in z1.indices) result[i + k] += z1[i] - z0.getOrElse(i) { 0L
         } - z2.getOrElse(i) { 0L }

    return result
}
```

Criba de Eratostenes

```
class Criba(n: Int){
    var criba = MutableList(n+1){-1}
    init {
        for (i in 2..n){
            if (criba[i] == -1){
                if (n/i>=i) for (j in i*i until (n+1) step i){
                    if (criba[j] == -1) criba[j] = i
                }
            }
        }
    }

    fun fact(n: Int) : MutableMap<Int,Int> {
        var res = mutableMapOf<Int,Int>()
        var x = n
        while(criba[x] != -1){
            res[criba[x]] = res.getOrDefault(criba[x], 0) + 1
            x /= criba[x]
        }
        if(x != 1) res[x] = res.getOrDefault(x, 0) + 1
        return res
    }
}
```

Geometría

Punto

```
import kotlin.math.*

class pt(x: Double, y: Double): Comparable<pt>{
    val x = x
    val y = y
    operator fun plus(p: pt) = pt(x + p.x, y + p.y)
    operator fun minus(p: pt) = pt(x - p.x, y - p.y)
    operator fun times(k: Double) = pt(x * k, y * k)
    operator fun div(k: Double) = pt(x / k, y / k)
    operator fun times(p: pt) = x * p.x + y * p.y
    operator fun rem(p: pt) = x * p.y - y * p.x
    fun angle(p: pt) = acos((this * p) / (this.norm() * p.norm()))
    fun norm2() = x * x + y * y
    fun norm() = sqrt(norm2())
    fun unit() = if (norm() > 0) this / norm() else pt(0.0, 0.0)
    fun rot(r: pt) = pt(this % r, this * r)
    fun rot(a: Double) = this.rot(pt(cos(a), sin(a)))
    fun left(p: pt, q: pt) = (q - p).unit() % (this - p).unit() > EPS

    operator override fun compareTo(p: pt): Int = when {
        abs(this.x - p.x) > EPS -> this.x.compareTo(p.x)
        else -> this.y.compareTo(p.y)
    }

    override fun equals(other: Any?) = other is pt && abs(x - other.x)<
         EPS && abs(y - other.y)<EPS
    override fun toString() = "($x, $y)"
}

val ccw90 = pt(1.0,0.0)
val cw90 = pt(-1.0,0.0)
```

Segmento

```
import kotlin.math.*

class Segment(val f: pt, val s: pt) {
    fun length(): Double {
        val dx = f.x - s.x
        val dy = f.y - s.y
        return sqrt(dx * dx + dy * dy)
    }
}

fun pc(a: pt, b: pt, o: pt): Double = (a-o) % (b-o)
fun pe(a: pt, b: pt, o: pt): Double = (a-o) * (b-o)

fun intersect(a: Segment, b: Segment): Boolean{
    val fb = 0.compareTo(pc(a.f, a.s, b.f))
    val sb = 0.compareTo(pc(a.f, a.s, b.s))
    val fa = 0.compareTo(pc(b.f, b.s, a.f))
    val sa = 0.compareTo(pc(b.f, b.s, a.s))
    if ((fb * sb < 0) && (fa * sa<0)) return true
    if ((fb==0 && pe(a.f, a.s, b.f)<=0) || (sb==0 && pe(a.f, a.s,b.s)
         <=0)) return true;
    if ((fa==0 && pe(b.f, b.s, a.f)<=0) || (sa==0 && pe(b.f, b.s, a.s)
         <=0)) return true;
    return false
}

fun dist(p: pt, s: Segment): Double{
    val a = abs(pc(s.f,s.s,p))
    val b = hypot(s.f.x - s.s.x, s.f.y - s.s.y)
    val h = a/b
    val c = hypot(b,h)
    val d1 = (s.f-p).norm()
    val d2 = (s.s-p).norm()
    if(b<EPS || c<= d1 || c<= d2) return minOf(d1,d2)
    return h
}

fun dist(a: Segment, b : Segment) : Double{
    if(intersect(a,b)) return 0.0
    return minOf(
        minOf(dist(a.f,b),dist(a.s,b)),
        min(dist(b.f,a),dist(b.s,a))
    )
}
```

Capsula convexa

```
fun chull(ps: List<pt>) : List<pt>{
    if(ps.size < 3) return ps
    val p = ps.sorted()
    val ch = mutableListOf<pt>()
    for(pi in p){
        while(ch.size > 1 && ch[ch.size - 1].left(ch[ch.size - 2], pi))
             ch.removeAt(ch.size - 1)
        ch.add(pi)
    }
    ch.removeAt(ch.size - 1)
    val t = ch.size
    for(pi in p.reversed()){
        while(ch.size > t+1 && ch[ch.size - 1].left(ch[ch.size - 2], pi
            )) ch.removeAt(ch.size - 1)
        ch.add(pi)
    }
    ch.removeAt(ch.size - 1)
    return ch
}
```

Tablas y Cotas

Primos cercanos a 10^n

9941 9949 9967 9973 10007 10009 10037 10039 10061
10067 10069 10079
99961 99971 99989 99991 100003 100019 100043 100049
100057 100069
999959 999961 999979 999983 1000003 1000033 1000037
1000039

9999943 9999971 9999973 9999991 10000019 10000079 10000103 10000121
99999941 99999959 99999971 99999989 100000007 100000037 100000039 100000049
999999893 999999929 999999937 1000000007 1000000009 1000000021 1000000033

Cantidad de primos menores que 10^n
$\pi(10^1)$ = 4 ; $\pi(10^2)$ = 25 ; $\pi(10^3)$ = 168 ; $\pi(10^4)$ = 1229 ; $\pi(10^5)$ = 9592 ; $\pi(10^6)$ = 78.498 ; $\pi(10^7)$ = 664.579 ; $\pi(10^8)$ = 5.761.455 ; $\pi(10^9)$ = 50.847.534 ; $\pi(10^{10})$ = 455.052,511 ; $\pi(10^{11})$ = 4.118.054.813 ; $\pi(10^{12})$ = 37.607.912.018

Divisores

Cantidad de divisores (σ_0) para algunos $n/\neg\exists n' < n, \sigma_0(n') \geqslant \sigma_0(n)$
$\sigma_0(60)$ = 12 ; $\sigma_0(120)$ = 16 ; $\sigma_0(180)$ = 18 ; $\sigma_0(240)$ = 20 ; $\sigma_0(360)$ = 24 ; $\sigma_0(720)$ = 30 ; $\sigma_0(840)$ = 32 ; $\sigma_0(1260)$ = 36 ; $\sigma_0(1680)$ = 40 ; $\sigma_0(10080)$ = 72 ; $\sigma_0(15120)$ = 80 ; $\sigma_0(50400)$ = 108 ; $\sigma_0(83160)$ = 128 ; $\sigma_0(110880)$ = 144 ; $\sigma_0(498960)$ = 200 ; $\sigma_0(554400)$ = 216 ; $\sigma_0(1081080)$ = 256 ; $\sigma_0(1441440)$ = 288 $\sigma_0(4324320)$ = 384 ; $\sigma_0(8648640)$ = 448
Suma de divisores (σ_1) para algunos $n/\neg\exists n' < n, \sigma_1(n') \geqslant \sigma_1(n)$; $\sigma_1(96)$ = 252 ; $\sigma_1(108)$ = 280 ; $\sigma_1(120)$ = 360 ; $\sigma_1(144)$ = 403 ; $\sigma_1(168)$ = 480 ; $\sigma_1(960)$ = 3048 ; $\sigma_1(1008)$ = 3224 ; $\sigma_1(1080)$ = 3600 ; $\sigma_1(1200)$ = 3844 ; $\sigma_1(4620)$ = 16128 ; $\sigma_1(4680)$ = 16380 ; $\sigma_1(5040)$ = 19344 ; $\sigma_1(5760)$ = 19890 ; $\sigma_1(8820)$ = 31122 ; $\sigma_1(9240)$ = 34560 ; $\sigma_1(10080)$ = 39312 ; $\sigma_1(10920)$ = 40320 ; $\sigma_1(32760)$ = 131040 ; $\sigma_1(35280)$ = 137826 ; $\sigma_1(36960)$ = 145152 ; $\sigma_1(37800)$ = 148800 ; $\sigma_1(60480)$ = 243840 ; $\sigma_1(64680)$ = 246240 ; $\sigma_1(65520)$ = 270816 ; $\sigma_1(70560)$ = 280098 ; $\sigma_1(95760)$ = 386880 ; $\sigma_1(98280)$ = 403200 ; $\sigma_1(100800)$ = 409448 ; $\sigma_1(491400)$ = 2083200 ; $\sigma_1(498960)$ = 2160576 ; $\sigma_1(514080)$ = 2177280 ; $\sigma_1(982800)$ = 4305280 ; $\sigma_1(997920)$ = 4390848 ; $\sigma_1(1048320)$ = 4464096 ; $\sigma_1(4979520)$ = 22189440 ; $\sigma_1(4989600)$ = 22686048 ; $\sigma_1(5045040)$ = 23154768 ; $\sigma_1(9896040)$ = 44323200 ; $\sigma_1(9959040)$ = 44553600 ; $\sigma_1(9979200)$ = 45732192

Factoriales

0! = 1	11! = 39.916.800
1! = 1	12! = 479.001.600 ($\in$ int)
2! = 2	13! = 6.227.020.800
3! = 6	14! = 87.178.291.200
4! = 24	15! = 1.307.674.368.000
5! = 120	16! = 20.922.789.888.000
6! = 720	17! = 355.687.428.096.000
7! = 5.040	18! = 6.402.373.705.728.000
8! = 40.320	19! = 121.645.100.408.832.000
9! = 362.880	20! = 2.432.902.008.176.640.000 $\in$ ll
10! = 3.628.800	21! = 51.090.942.171.709.400.000

max signed tint = 9.223.372.036.854.775.807
max unsigned tint = 18.446.744.073.709.551.615

Consejos

Debugging

- ¿Si n = 0 anda? (similar casos borde tipo n=1, n=2, etc)
- ¿Si hay puntos alineados anda?
- ¿Si es vacío anda?
- ¿Si hay multiejes anda?
- ¿Si no tiene aristas anda?
- ¿Si tiene ciclos anda?
- ¿Si tiene un triángulo anda?
- ¿Los arrays son suficientemente grandes? (siempre denle bastante de más por las dudas, pero tampoco se ceben como para que ya no entre en memoria XD)
- ¿Puede dar integer overflow? (SIEMPRE mirar el integer overflow con MUCHO cuidado)
- ¿Podés dividir por cero en algún caso?
- ¿Estás memorizando la recursión bien?
- ¿El caso base está bien hecho y se llega siempre?
- ¿Están bien puestas las cotas iniciales de la binary / inicialización del acumulador máximo/mínimo?
- ¿Estás inicializando bien antes de cada caso?
- ¿Le copiaste el input dos veces en el archivo de entrada (para ver que de igual y bien las dos veces)? [No aplica cuando viene solo una instancia de input]
- ¿Pasa los ejemplos? [No es joda, Leo se quedo afuera de la mundial por esto]

Hitos de prueba

- **45min** todas las columnas de la tabla llena
- **2h** todos conocen todo
- **3h** reunión estratégica
- **4h** reunión estratégica