1 Graficas

1.1 Kruskal

```
//Incluye Union Find bajo Estructuras.
struct edge {
  int u, v, w;
  bool operator< (const edge &o) const{ return w < o.w; }
};
vector<edge> edges;
int kruskal() {
  int res = 0;
  sort(edges.begin(), edges.end());
  for(auto e : edges) if(join(e.u, e.v))
    res += e.w; // uv es arista del MST
  return res;
}
```

1.2 LCA

```
const int MAXN = 1e6, LOG = 20;
vector<int> adj[MAXN];
int up[LOG][MAXN], dep[MAXN];
void dfs(int s, int p = 0) {
  up[0][s] = (p ?: s);
  dep[s] = (p ? dep[p] + 1 : s);
  for(auto v : adj[s]) if(v != s) dfs(v, s);
}
void build() {
  dfs(1);
  for(int l = 1; l < LOG; l++) for(int v = 1; v < MAXN; v++)
    up[l][v] = up[l - 1][up[l - 1][v]];
}
void jmp(int &u, int v, int d) {
  for(int l = LOG - 1; l >= 0; l--) if(d & (1 << l))
      u = up[l][u];
}
int LCA(int u, int v) {
  if(dep[u] < dep[v]) swap(u, v);
  jmp(u, v, dep[u] - dep[v]);
  if(u == v) return u;
  for(int l = LOG - 1; l >= 0; l--)
    if(up[l][u] != up[l][v])
      u = up[l][u], v = up[l][v];
  return up[0][u];
}
```

1.3 Vertices y Aristas de Corte

```
const int MAXN = 1e5;
int low[MAXN], ord[MAXN], tin;
vector<int> adj[MAXN];
int dfs(int s) {
  low[s] = ord[s] = ++tin;
  for(auto v : adj[s]) {
    if(!low[v]) {
      dfs(v);
      if(low[v] > ord[s]) { /* uv es puente */ }
      if(low[v] >= ord[s]) { /* u es punto de articulaci n (o ra z) */}
      low[s] = min(low[s], low[v]);
    }
    else if(ord[v] < ord[s])
      low[s] = min(low[s], ord[v]);
  }
  return low[s];
}
```

1.4 SCC

```
vi val, comp, sta;
int Time, ncomps;
template<class G> int dfs(int s, G &g) {
  int low = val[s] = ++Time, x; sta.push_back(s);
  for(auto v : g[s]) if(comp[v] < 0)
    low = min(low, val[v] ?: dfs(v, g));
  if(low == val[s]) {
    do {
      x = sta.back(); sta.pop_back();
      comp[x] = ncomps;
    } while(x != s);
    ncomps++;
  }
}
template<class G> int scc(G &g) {
  int n = g.size();
  comp.assign(n, -1), val.assign(0, -1);
  Time = ncomps = 0;
  for(int i = 0; i < n; i++) if(comp[i] < 0) dfs(i, g);
}
```

2 Strings

2.1 Z Function

```
vi z_func(string &s) {
  int n = s.length(), l = -1, r = -1;
  vi z(n);
  for(int i = 1; i < n; i++) {
    if(i <= r) z[i] = min(z[i - l], r - i + 1);
    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;
}
```

2.2 Manacher

```
vi manacher(string &s) {
  // Encuentra palindromos de longitud impar.
  int n = s.length(), l = -1, r = -1;
  vi p(n);
  for(int i = 0; i < n; i++) {
    if(i <= r) p[i] = min(p[l + r - i], r - i + 1);
    while(i+p[i]+1 < n && i-p[i]-1 >= 0 && s[i+p[i]+1] == s[i-p[i]-1]) p[i]++;
    if(i + p[i] > r) l = i - p[i], r = i + p[i];
  }
  return p;
}
```

2.3 Suffix Array

```
const int MAXN = 4e5;
string s;
int SA[MAXN], LCP[MAXN], val[MAXN], cnt[MAXN], n;

void csort(int l) {
  int mx = max(300, n), sum = 0, tSA[MAXN];
  fill(cnt, cnt + mx, 0);
  for(int i = 0; i < n; i++)
    cnt[(SA[i] + l < n) ? val[SA[i] + l] : 0]++;
  for(int i = 0; i < mx; i++)
    {int t = cnt[i]; cnt[i] = sum; sum += t;}
  for(int i = 0; i < n; i++)
    tSA[cnt[SA[i] + l < n ? val[SA[i] + l] : 0]++] = SA[i];
  for(int i = 0; i < n; i++)
    SA[i] = tSA[i];
}
void buildSA() {
  int nval[MAXN], rk, l = 1;
  iota(SA, SA + n, 0);
  for(int i = 0; i < n; i++)
    val[SA[i]] = s[i];
  do {
    csort(l);
    csort(0);
    nval[SA[0]] = rk = 0;
    for(int i = 1; i < n; i++) nval[SA[i]] =
      ii{val[SA[i]],val[SA[i]+l]} == ii{val[SA[i-1]],val[SA[i - 1]+l]} ? rk:++rk;
    for(int i = 0; i < n; i++)
      val[i] = nval[i];
    l <<= 1;
  } while(val[SA[n - 1]] != n - 1 && l < n);
}
void buildLCP() {
  int pre[MAXN], PLCP[MAXN], L = 0;
  pre[SA[0]] = -1;
  for(int i = 1; i < n; i++)
    pre[SA[i]] = SA[i - 1];
  for(int i = 0; i < n; i++) {
    if(pre[i] == -1) {
      PLCP[i] = -1;
      continue;
    }
    while(s[i + L] == s[pre[i] + L]) L++;
    PLCP[i] = L;
    L = max(0, L - 1);
  }
  for(int i = 0; i < n; i++)
    LCP[i] = PLCP[SA[i]];
}
```

2.4 Hashing

```
struct rhash {
  ll P, Q;  // P ~ cantidad de caracteres del alfabeto, Q ~ 10^9
  vl H, po; // Se pueden usar varios hashes si las colisiones son problema
  rhash(string &s, ll P, ll Q) : P(P), Q(Q) {
    int n = s.length(); H.resize(n); po.resize(n);
    po[0] = 1; H[0] = s[0];
    for(int i = 1; i < n; i++) {
      H[i] = (P*H[i - 1] + s[i])%Q;
      po[i] = (po[i - 1] * P)%Q;
    }
  }
  ll get(ll l, ll r) { // Hash de s[l, r]
    if(l == 0) return H[r];
    ll res = (H[r] - po[r - l + 1]*H[l - 1])%Q;
    return res >= 0 ? res : res + Q;
  }
};
```

3 Geometria

3.1 Punto

```
template<class T>
struct pt {
  T x, y;
  pt(T x = 0, T y = 0) : x(x), y(y) {}
  bool operator< (pt o) const {return (x < o.x || (x == o.x && y < o.y)); }
  bool operator== (pt o) const {return (x == o.x && y == 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 l) const {return pt(l*x, l*y);}
  pt operator/ (T l) const {return pt(x/l, y/l);}
  T dot(pt o) { return x*o.x + y*o.y; }
  T cross(pt o) { return x*o.y - y*o.x; }
  T cross(pt a, pt b) { return (a - *this).cross(b - *this); }
  T normsq(pt o) { return x*x +y*y; }
  double norm(pt o) { return hypot(x, y); }
};
```

3.2 Envolvente

```
vector<pt<ll>> convex_hull(vector<pt<ll>> P) {
  int n = P.size();
  if(n <= 2) return P;
  vector<pt<ll>> L, U;
  sort(P.begin(), P.end());
  for(int i = 0; i < n; i++) {
    while(U.size() >= 2 && U[U.size() - 2].cross(U[U.size() - 1], P[i]) >= 0)
      U.pop_back();
    while(L.size() >= 2 && L[L.size() - 2].cross(L[L.size() - 1], P[n-i-1]) >= 0)
```

```
      L.pop_back();
    U.push_back(P[i]), L.push_back(P[n - i - 1]);
  }
  U.insert(U.end(), L.begin() + 1, L.end() - 1);
  return U;
}
```

4 Flujo

4.1 Dinic

```
const int MAXN = 5000, INF = 1e9;
int dist[MAXN], ptr[MAXN], src, dst;
struct Edge {
  int to, rev, f, cap;
  Edge(int to, int rev, int f, int cap) : to(to), rev(rev), f(f), cap(cap);
};
vector<Edge> G[MAXN];
void addEdge(int u, int v, int cap) {
  G[u].push_back(Edge(v, G[v].size(), 0, cap));
  G[v].push_back(Edge(u, G[u].size() - 1, 0, 0));
}
bool bfs() {
  queue<int> q({src});
  memset(dist, -1, sizeof dist);
  dist[src] = 0;
  while(!q.empty() && dist[dst] == -1) {
    int u = q.front();
    q.pop();
    for(auto e : G[u]) {
      int v = e.to;
      if(dist[v] == -1 && e.f < e.cap) {
        dist[v] = dist[u] + 1;
        q.push(v);
      }
    }
  }
  return dist[dst] != -1;
}
int dfs(int u, int f) {
  if(u == dst || !f) return f;
  for(int &i = ptr[u]; i < G[u].size(); i++) {
    Edge &e = G[u][i];
    int v = e.to;
    if(dist[v] != dist[u] + 1) continue;
    if(int df = dfs(v, min(f, e.cap - e.f))) {
      e.f += df;
      G[v][e.rev].f -= df;
      return df;
    }
  }
  return 0;
}
long long dinic() {
  long long mf = 0;
  while(bfs()) {
    memset(ptr, 0, sizeof ptr);
    while(long long pushed = dfs(src, INF))
      mf += pushed;
  }
  return mf;
}
```

5 Estructuras

5.1 CHT

```
// Tomado de KACTL. O(nlog n). Para O(n) se hace un deque de lineas y se saca del
// lado correspondiente cuando salen de la envolvente o no son relevantes.
bool Q;
struct Line {
  mutable ll m, b, p;
  bool operator<(const Line& o) const {
    return Q ? p < o.p : m < o.m;
  }
};
struct LineContainer : multiset<Line> {
  const ll inf = LLONG_MAX;
  ll div(ll a, ll b) {
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) { x->p = inf; return false; }
    if (x->m == y->m) x->p = x->b > y->b ? inf : -inf;
    else x->p = div(y->b - x->b, x->m - y->m);
    return x->p >= y->p;
  }
  void add(ll m, ll b) {
    auto z = insert({m, b, 0}), y = z++, x = y;
    while (isect(y, z)) z  = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)-> p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    Q = 1; auto l = *lower_bound({0, 0, x}); Q = 0;
    return l.m * x + l.b;
  }
};
```

5.2 Union Find

```
const int MAXN = 1e6;
int rep[MAXN], sz[MAXN];
void init() {
  fill(rep, rep + MAXN, -1), fill(sz, sz + MAXN, 1);
}
int find(int u) {
  return (rep[u] == -1) ? u : rep[u] = find(rep[u]);
}
bool join(int u, int v) {
  u = find(u), v = find(v);
  if(u == v) return false;
  if(sz[u] < sz[v]) swap(u, v);
  return sz[u] += sz[v], rep[v] = u, true;
}
```

5.3 Wavelet Tree

```
struct wnode {
  wnode *left, *right;
  int lo, hi;
  vector<int> c;
  wnode(int lo, int hi, int* st, int* en) : lo(lo), hi(hi) {
    if(hi == lo || st == en)
      return;
    int mi = (lo + hi)/2;
    auto f = [mi](int x) { return x <= mi; };
    c.push_back(0);
    for(auto it = st; it != en; ++it)
      c.push_back(c.back() + f(*it));
    auto it = stable_partition(st, en, f);
    left = new wnode(lo, mi, st, it);
    right = new wnode(mi + 1, hi, it, en);
  }
  int kth(int L, int R, int k) {
    if(lo == hi)
      return lo;
    int der = c[R], izq = c[L - 1], tol = der - izq;
    if(tol >= k)
      return left->kth(izq + 1, der, k);
    return right->kth(L - izq, R - der, k - tol);
  }
};
```

5.4 Segment Tree

```
const int NE = 1e9;
struct node {
  int mn, l, r;
  node *left, *right;
  node(int l, int r, int* A) : l(l), r(r) {
    if(l == r)
      mn = A[l];
    else {
      int mi = (l + r)/2;
      left = new node(l, mi, A);
      right = new node(r + 1, mi, A);
      mn = min(left->mn, right->mn);
    }
  }
  void upd(int p, int v) {
    if(r < p || p < l)
      return;
    if(l == r) {
      mn = v;
      return;
    }
    left->upd(p, v), right->upd(p, v);
    mn = min(left->mn, right->mn);
  }
  int qry(int rl, int rr) {
    if(rr < l || r < rl)
      return NE;
    if(rl <= l && r <= rr)
      return mn;
    return min(left->qry(rl, rr), right->qry(rl, rr));
  }
};
```

6 Mate

6.1 Miller-Rabin

```
bool isprime(ll p) {
  if(p == 1) return false;
  if(p % 2 == 0) return p == 2 ? true : false;
  ll d = p - 1;
  while(d % 2 == 0) d >>= 1ll;
  for(int its = 0; its < 15; its++) {
    ll a = (rand() % (p - 1)) + 1, x = d;
    a = mpow(a, d, p);
    while(x != p - 1 && a != p - 1 && a != 1) {
      a = mmul(a, a, p);
      x *= 2ll;
    }
    if(a != p - 1 && x % 2 == 0) return false;
  }
  return true;
}
```

6.2 CRT

```
template<typename T> void euclid(T a, T b, T &x, T &y) {
  if(!b) {x = 1, y = 0; return;}
  euclid(b, a % b, y, x);
  y -= a/b * x;
}
template<typename T> T crt(T x1, T m1, T x2, T m2) {
  T d = __gcd(m1, m2), r, s;
  if(x1 % d != x2 % d)
    return -1;
  euclid(m1, m2, r, s);
  m1 /= d, m2 /= d;
  T mod = d*m1*m2, a1 = ((m1*r)%mod*x2)%mod, a2 = ((m2*s)%mod*x1)%mod, y = (a1 +
      a2)%mod;
  return (y >= 0 ? y : y + mod);
}
```

7 Varios

7.1 LIS

```
vl lis(vl a) {
  int n = a.size(), sz = 0; map<pl, pl> pre;
  vl dp(n + 1, LLONG_MAX); dp[0] = LLONG_MIN;
  for(int i = 0; i < n; i++) {
    auto it = upper_bound(dp.begin(), dp.end(), a[i]); //a[i]-1 para estricta.
    if(*it == LLONG_MAX) sz++; *it = min(*it, a[i]);
    int pos = distance(dp.begin(), it); pre[{pos, a[i]}] = {pos - 1, dp[pos - 1]};
  }
  vl ans; pl cu = {sz, dp[sz]};
  do {
    ans.push_back(cu.second);
    cu = pre[cu];
  } while(cu.first);
  return reverse(ans.begin(), ans.end()), ans;
}
```