Índice

1. Algorithm

#include <algorithm> #include <numeric>

Algo	**Params**	**Funcion**
sort, stable_sort	f, l	ordena el intervalo
nth_element	f, nth, l	*void* ordena el n-esimo, y particiona el resto
fill, fill_n	f, l / n, elem	*void* llena [f, l) o [f, f+n) con elem
lower_bound, upper_bound	f, l, elem	*it* al primer / ultimo donde se puede insertar elem para que quede ordenada
binary_search	f, l, elem	*bool* esta elem en [f, l)
copy	f, l, resul	hace resul+i=f+i $\forall i$
find, find_if, find_first_of	f, l, elem / pred / f2, l2	*it* encuentra i $\in$[f,l) tq. i=elem, pred(i), i$\in$[f2,l2)
count, count_if	f, l, elem/pred	cuenta elem, pred(i)
search	f, l, f2, l2	busca [f2,l2) $\in$ [f,l)
replace, replace_if	f, l, old / pred, new	cambia old / pred(i) por new
reverse	f, l	da vuelta
partition, stable_partition	f, l, pred	pred(i) ad, !pred(i) atras
min_element, max_element	f, l, [comp]	*it* min, max de [f,l]
lexicographical_compare	f1,l1,f2,l2	*bool* con [f1,l1]¡[f2,l2]
next/prev_permutation	f,l	deja en [f,l) la perm sig, ant
set_intersection, set_difference, set_union, set_symmetric_difference,	f1, l1, f2, l2, res	[res, ...) la op. de conj
push_heap, pop_heap, make_heap	f, l, e / e /	mete/saca e en heap [f,l), hace un heap de [f,l)
is_heap	f,l	*bool* es [f,l) un heap
accumulate	f,l,i,[op]	$T = \sum$/oper de [f,l)
inner_product	f1, l1, f2, i	T = i + [f1, l1) . [f2, ...)
partial_sum	f, l, r, [op]	r+i = $\sum$/oper de [f,f+i] $\forall i \in$[f,l)
__builtin_ffs	unsigned int	Pos. del primer 1 desde la derecha
__builtin_clz	unsigned int	Cant. de ceros desde la izquierda.
__builtin_ctz	unsigned int	Cant. de ceros desde la derecha.
__builtin_popcount	unsigned int	Cant. de 1's en x.
__builtin_parity	unsigned int	1 si x es par, 0 si es impar.
__builtin_XXXXXXll	unsigned ll	= pero para long long's.

2. Math

#include <math>

Algo	Params	Funcion
cos, sin, tan	x	regresa en radianes
acos	x	arc cos, rango [0, pi]
asin	x	arc sin, rango[-pi/2,+pi/2]
atan	x	arc tan, rango [-pi/2,+pi/2]
atan2	x	arc tan, rango [-pi,+pi]
exp	x	fpow(e, x)
log	x	logaritmo natural
log10	x	logaritmo base 10
log2	x	logaritmo base 2
sqrt	x	raiz cuadrada, negativos da error
cbrt	x	raiz cubica
ceil	x	redondeo pa′ arriba
floor	x	redondeo pa′ abajo
fabs	x	float/double abs
abs	x	absoluto D:

3. Data Structures

3.1. Disjoint set

```
struct dsu{
  vector<int> pr;

  dsu(int nn = 0){ // O(N)
    pr.resize(nn + 5);
    iota(all(pr), 0);
  }

  int find(int u){ // O(1)
    return pr[u] == u ? u: pr[u] = find(pr[u]);
  }

  bool unite(int u, int v){ // O(1)
    u = find(u), v = find(v);
    if( u != v ){
      pr[v] = u;
      return true;
    }
    return false;
  }
};
```

3.2. Disjoint set with rollback

```
struct dsu{
  vector<int> pr, tot;
  stack<ii> what;

  dsu(int nn = 0){ // O(N)
    pr.resize(nn + 5);
    iota(all(pr), 0);
    tot.resize(nn + 5, 1);
  }

  int find(int u){ // O(logN)
    return pr[u] == u ? u: find(pr[u]);
  }


```

```
  void unite(int u, int v){ // O(logN)
    u = find(u), v = find(v);
    if( u == v ){
      what.push({-1, -1});
    }else{
      if( tot[u] < tot[v] ){
        swap(u, v);
      }
      what.push({u, v});
      tot[u] += tot[v];
      pr[v] = u;
    }
  }

  ii rollback(){ // O(1)
    ii last = what.top();
    what.pop();
    int u = last.f, v = last.s;
    if( u != -1 ){
      tot[u] -= tot[v];
      pr[v] = v;
    }
    return last;
  }
};
```

3.3. Sparce table

```
void process_all_logs(){
  lg[0] = -1;
  For1(i, N - 1){
    lg[i] = lg[i >> 1] + 1;
  }
}

template <class T>
T query(int l, int r){ // O(logN) = op{+, *, ^, |, &}
  T ans = {};
  Rof(k, LogN - 1, 0, -1){
    if( l + (1 << k) - 1 <= r ){
      ans += sp[k][l]);
      l += (1 << k);
    }
  }
  return ans;
}

template <class T>
T query(int l, int r){ // O(1) = op{min, max, gcd}
  int k = lg[r - l + 1];
  return min(sp[k][l], sp[k][r - (1 << k) + 1]);
}

void do_sparce(){ // O(N * logN)
  Forn(i, n){
    sp[0][i] = a[i];
  }
  For1(k, LogN - 1){
    Forn(i, n - (1 << k) + 1){
      sp[k][i] = sp[k - 1][i] + sp[k - 1][i + (1 << (k - 1))];
    }
  }
}
```

3.4. Min-Max queue

```
struct min_queue: deque<ii> { // O(N)
  void add(ii cur){ // add a element to the right {v, pos}
    while( !empty() && back().f >= cur.f ){
      pop_back();
    }
    push_back(cur);
  }

  void rem(int i){// remove all less than i, O(K)
    while( front().s < s ){
      pop_front();
    }
  }

  lli qmin(){ // min[l, r], O(K)
    return front().f;
  }
} mnqu;
```

3.5. Sqrt descomposition

```
template <class T>
T query(int l, int r){ // O(sqrtN)
  T ans = {};
  while( l <= r ){
    if( l % block == 0 && l + block - 1 <= r ){
      ans += sq[who[l]];
      l += block;
    }else{
      ans += a[l++];
    }
  }
  return ans;
}

void do_sqrt_descomposition(){ // O(N)
  block = sqrt(n);
  Forn(i, n){
    who[i] = i / block;
    sq[who[i]] += a[i];
  }
}
```

3.6. Mo's algorithm

```
struct query{
  int l, r, i;
};

vector<query> queries;

void mo(){
  int block = sqrt(n);
  sort(all(queries), [&](query a, query b){
    if( a.l / block == b.l / block ){
      return a.r < b.r;
    }
    return a.l / block < b.l / block;
  });
  int l = queries[0].l, r = l - 1;
  for(query &qq: queries){ // O((N + Q) * sqrt(N))
    while( r < qq.r ){
      add(++r);
    }
    while( r > qq.r ){
      rem(r--);
    }
    while( l < qq.l ){
      rem(l++);
    }
    while( l > qq.l ){
      add(--l);
    }
    ans[qq.i] = solve();
  }
}
```

3.7. Fenwick tree

```
template <class T>
struct fenwick{
  #define lsb(x) (x & -x)
  vector<T> fenw;

  fenwick(int nn = 0){ // O(N)
    fenw.resize(nn + 5, {});
  }

  void update(int sx, T v){ // O(logN)
    For(x, sx, size(fenw) - 1, +lsb(x)){
      fenw[x] += v;
    }
  }

  T query(int sx){ // O(logN)
    T v = {};
    Rof(x, sx, 1, -lsb(x)){
      v += fenw[x];
    }
    return v;
  }

  T at(int x){ // O(1)
    T v = fenw[x];
    int y = x - lsb(x);
```

```
    for(--x; x != y; x -= lsb(x)){
      v -= fenw[x];
    }
    return v;
  }
};
```

3.8. Segment tree

```
struct segtree{
  #define mid (l + r) / 2
  #define left(u) (u + 1)
  #define right(u) (u + ((mid - l + 1) << 1))

  struct node{
    lli sum = 0;
    int mx = -1;
  };

  vector<node> st;
  node z;
  int n;

  friend node operator + (const node a, const node b){
    node c;
    c.sum = a.sum + b.sum;
    c.mx = max(a.mx, b.mx);
    return c;
  }

  void build(int u, int l, int r){ // O(N * logN)
    if( l == r ){
      st[u] = {};
      return;
    }
    build(left(u), l, mid);
    build(right(u), mid + 1, r);
    st[u] = st[left(u)] + st[right(u)];
  }

  segtree(int n1 = 1){
    n = n1;
    st.resize(2 * n); // wow :O!
    build(0, 1, n);
  }

  void update(int u, int l, int r, int kth, int v){ // O(logN)
    if( l == r ){
      st[u] = {};
      return;
    }
    if( kth <= mid ){
      update(left(u), l, mid, kth, v);
    }else{
      update(right(u), mid + 1, r, kth, v);
    }
    st[u] = st[left(u)] + st[right(u)];
  }

  node query(int u, int l, int r, int ll, int rr){ // O(logN)
    if( r < ll || rr < l || r < l ){
      return z;
    }
    if( ll <= l && r <= rr ){
      return st[u];
    }
    return query(left(u), l, mid, ll, rr) +
           query(right(u), mid + 1, r, ll, rr));
  }
  #undef mid
};
```

3.9. Lazy segment tree

```
struct lazy_segtree{
  #define mid (l + r) / 2
  #define left(u) (u + 1)
  #define right(u) (u + ((mid - l + 1) << 1))

  struct node{
    lli sum = 0;
    bool lazy = false;
  };

  vector<node> st;
  node z;
```

```
  int n;

  friend node operator + (const node a, const node b){
    node c;
    c.sum = a.sum + b.sum;
    return c;
  }

  void refresh(int u, int l, int r){ // O(1)
    if( st[u].lazy ){
      st[u].sum = (r - l + 1) - st[u].sum;
      if( l != r ){
        st[left(u)].lazy ^= st[u].lazy;
        st[right(u)].lazy ^= st[u].lazy;
      }
      st[u].lazy = false;
    }
  }

  lazy_segtree(int _n = 1){
    n = _n;
    st.resize(2 * n); // wow :O!
  }

  void update(int u, int l, int r, int ll, int rr){ // O(logN)
    refresh(u, l, r);
    if( rr < l || r < ll || r < l ){
      return;
    }
    if( ll <= l && r <= rr ){
      st[u].lazy ^= 1;
      refresh(u, l, r);
      return;
    }
    update(left(u), l, mid, ll, rr);
    update(right(u), mid + 1, r, ll, rr);
    st[u] = st[left(u)] + st[right(u)];
  }

  node query(int u, int l, int r, int ll, int rr){ // O(logN)
    refresh(u, l, r);
    if( rr < l || r < ll || r < l ){
      return 0;
    }
    if( ll <= l && r <= rr ){
      return st[u].sum;
    }
    return query(left(u), l, mid, ll, rr) +
           query(right(u), mid + 1, r, ll, rr));
  }
  #undef mid
};
```

3.10. Persistent segment tree

```
struct persistent_segtree{
  struct node{
    lli sum = 0;
    node* left = nullptr;
    node* right = nullptr;
  };

  node* root[N];
  int timer = 0;

  persistent_segtree(){
    root[0] = new node();
  }

  void build(node* &cur, int l, int r){ // O(N * logN)
    if( l == r ){
      cur->sum = a[l];
      return;
    }
    int mid = (l + r) / 2;
    if( !cur->left ){
      cur->left = new node();
      cur->right = new node();
    }
    build(cur->left, l, mid);
    build(cur->right, mid + 1, r);
    cur->sum = cur->left->sum + cur->right->sum;
  }

  void build(){
    build(root[0], 1, n);
```

```
  }

  void update(node* last, node* &cur, int l, int r, int pos, int v){ //
      O(logN)
    if( l == r ){
      cur->sum = last->sum + v;
      return;
    }
    int mid = (l + r) / 2;
    if( pos <= mid ){
      cur->left = new node();
      cur->right = last->right;
      update(last->left, cur->left, l, mid, pos, v);
    }else{
      cur->left = last->left;
      cur->right = new node();
      update(last->right, cur->right, mid + 1, r, pos, v);
    }
    cur->sum = cur->left->sum + cur->right->sum;
  }

  void update(int idx, int pos, int v){
    root[++timer] = new node();
    update(root[idx], root[timer], 1, n, pos, v);
  }

  lli qsum(node* &cur, int l, int r, int ll, int rr){ // O(logN)
    if( rr < l || r < ll || r < l || !cur ){
      return 0;
    }
    if( ll <= l && r <= rr ){
      return cur->sum;
    }
    int mid = (l + r) / 2;
    return qsum(cur->left, l, mid, ll, rr) +
           qsum(cur->right, mid + 1, r, ll, rr);
  }

  lli qsum(int idx, int l, int r){
    return qsum(root[idx], 1, n, l, r);
  }
};
```

4. Graphs

4.1. Breadth first search

```
void bfs(int s){ // O(N + M)
  fill_n(dist, n + 1, -1);
  queue<int> qu;
  dist[s] = 0;
  qu.push(s);
  while( !qu.empty() ){
    int u = qu.front();
    qu.pop();
    for(edge &e: g[u]){
      if( dist[e.v] == -1 ){
        dist[e.v] = dist[u] + e.dist;
        qu.push(e.v);
      }
    }
  }
}
```

4.2. Depth first search

```
void dfs(int u, int pr = 0){ // O(N + M)
  for(edge &e: g[u]) if( !vis[e.v] ){
    dfs(e.v, u);
  }
}
```

4.3. Floyd Warshall

```
void floyd_warshall(){ // O(N ^ 3)
  For1(u, n) For1(v, n){
    dist[u][v] = (u == v ? 0: g[u][v]);
  }
  For1(k, n){ // You can set the order
    For1(u, n) For1(v, n){
      umin(dist[u][v], dist[u][k] + dist[k][v]);
    }
  }
}
```

4.4. Bellman Ford

```
void bellmand_ford(int s){ // O(N * M)
  fill_n(dist, n + 1, inf);
  dist[s] = 0;
  for(;;){
    bool any = false;
    for(edge &e: edges) if( dist[e.u] < inf ){
      if( dist[e.u] + e.dist < dist[e.v] ){
        dist[e.v] = dist[e.u] + e.dist;
        any = true;
      }
    }
    if( !any ) break;
  }
}
```

4.5. Dijkstra

```
void dijkstra(int s){ // O((N + M) * logN)
  fill_n(dist, n + 1, inf);
  priority_queue< tuple<int, int> > pq; // {dist, node}
  pq.push({dist[s] = 0, s});
  while( !pq.empty() ){
    int d, u;
    tie(d, u) = pq.top();
    pq.pop();
    if( dist[u] != -d ){
      continue;
    }
    for(edge &e: g[u]){
      if( dist[u] + e.dist < dist[e.v] ){
        dist[e.v] = dist[u] + e.dist;
        pq.push({-dist[e.v], e.v});
      }
    }
  }
}
```

4.6. Tarjan algorithm (SCC)

```
void tarjan(int u, int pr = 0){ // O(N + M)
  static int timer = 0;
  fup[u] = tin[u] = ++timer;
  vis[u] = true;
  slew.push(u);
  for(int v: g[u]){
    if( !tin[v] ){
      tarjan(v, u);
    }
    if( vis[v] ){
      fup[u] = min(fup[u], fup[v]);
    }
  }
  if( fup[u] == tin[u] ){
    vector<int> cc;
    do{
      int v = slew.top();
      slew.pop();
      vis[v] = false;
      cc.pb(v);
    }while( v != u );
    scc.pb(cc);
  }
}
```

4.7. Kosaraju algorithm (SCC)

```
void dfs1(int u){
  vis[u] = 1;
  for(int v: g[u]) if( vis[v] != 1 ){
    dfs1(v);
  }
  order.pb(u);
}

void dfs2(int u){
  vis[u] = 2;
  scc[u] = k;
  for(int v: rg[u]) if( vis[v] != 2 ){
    dfs2(v);
  }
```

```
}

void kosaraju(){ // O(N + M)
  For1(u, n) if( vis[u] != 1 ){
    dfs1(u);
  }
  reverse(all(order));
  for(int u: order) if( vis[u] != 2 ){
    ++k;
    dfs2(u);
  }
}
```

4.8. Kahn-s algorithm

```
void topsort(){ // O(N + M), first fill the indeg[]
  queue<int> qu;
  For1(u, n){
    if( !indeg[u] ){
      qu.push(u);
    }
  }
  while( !qu.empty() ){
    int u = qu.front();
    qu.pop();
    order.pb(u);
    for(int v: g[u]){
      if( --indeg[v] == 0 ){
        qu.push(v);
      }
    }
  }
}
```

4.9. Cutpoints

```
void cutpoints(int u, int pr = 0){ // O(N + M)
  static int timer = 0;
  tin[u] = fup[u] = ++timer;
  int children = 0;
  for(int v: g[u]) if( v != pr ){
    if( !tin[v] ){
      ++children;
      cutpoints(v, u);
      fup[u] = min(fup[u], fup[v]);
      if( fup[v] >= tin[u] && pr ){
        // u is a cutpoint
      }
    }
    fup[u] = min(fup[u], tin[v]);
  }
  if( !pr && children > 1 ){
    // u is a cutpoint
  }
}
```

4.10. Bridges

```
void bridges(int u, int pr = 0){ // O(N + M)
  static int timer = 0;
  fup[u] = tin[u] = ++timer;
  for(int v: g[u]) if( v != pr ){
    if( !tin[v] ){
      bridges(v, u);
      fup[u] = min(fup[u], fup[v]);
      if( fup[v] > tin[u] ){
        // bridge u -> v
      }
    }
    fup[u] = min(fup[u], tin[v]);
  }
}
```

4.11. Detect a cycle

```
bool cycle(int u){ // O(N + M)
  vis[u] = 1;
  for(int v: g[u]){
    if( vis[v] == 1 ){
      return true;
    }
    if( !vis[v] && cycle(v) ){
      return true;
    }
  }
  vis[u] = 2;
  return false;
}
```

4.12. Euler tour

Mo's in tree, extended euler tour (tin[u] = ++timer, tout[u] = ++timer)
1. u == lca(u, v), query(tin[u], tin[v])
2. query(tout[u], tin[v]) + query(tin[w], tin[w]) (**w = lca(u, v)**)

4.13. Lowest common ancestor (LCA)

```
void dfs(int u, int pr[]){ // O(N + M)
  tin[u] = ++timer;
  tour[timer] = u;
  for(int v: g[u]) if( v != pr[u] ){
    pr[v] = u;
    dep[v] = dep[u] + 1;
    dfs(v, pr);
  }
  tour[u] = timer;
}
-----------------------------------------------------------
bool anc(int u, int v){ // O(1)
  return tin[u] <= tin[v] && tout[v] <= tout[u];
}

int lca(int u, int v){ // O(logN)
  if( dep[u] > dep[v] ){
    swap(u, v);
  }
  if( anc(u, v) ){
    return u;
  }
  Rof(k, LogN - 1, 0, -1){
    if( pr[k][u] && !anc(pr[k][u], v) ){
      u = pr[k][u];
    }
  }
  return pr[0][u];
}
-----------------------------------------------------------
int lca(int u, int v){ // O(logN)
  if( dep[u] > dep[v] ){
    swap(u, v);
  }
  Rof(k, LogN - 1, 0, -1){
    if( dep[v] - dep[u] >= (1 << k) ){
      v = pr[k][v];
    }
  }
  if( u != v ){
    Rof(k, LogN - 1, 0, -1){
      if( pr[k][v] && pr[k][v] != pr[k][u] ){
        u = pr[k][u];
        v = pr[k][v];
      }
    }
    return pr[0][u];
  }
  return u;
}
```

4.14. Heavy-light decomposition

```
int dfs(int u){ // O(N + M)
  tot[u] = 1, heavy[u] = head[u] = 0;
  for(int v: g[u]) if( !dep[v] ){
    pr[v] = u;
    dep[v] = dep[u] + 1;
    tot[u] += dfs(v);
    if( tot[v] > tot[heavy[u]] ){
      heavy[u] = v;
    }
  }
  return tot[u];
}

void decompose(int u, int h){ // O(N * logN)
  head[u] = h, pos[u] = ++timer, who[timer] = u;
  if( heavy[u] != 0 ){
    decompose(heavy[u], h);
  }
  for(int v: g[u]){
    if( v != pr[u] && v != heavy[u] ){
      decompose(v, v);
    }
  }
}

```

```cpp
void hld(int r = 1){
  pr[r] = timer = 0;
  dep[r] = 1; // Need some of depth
  dfs(r);
  decompose(r, r);
}

template <class todo>
void process_path(int u, int v, todo op){ // O(logN)
  for(; head[u] != head[v]; v = pr[head[v]]){
    if( dep[head[u]] > dep[head[v]] ){
      swap(u, v);
    }
    op(pos[head[v]], pos[v]);
  }
  if( dep[u] > dep[v] ){
    swap(u, v);
  }
  if( u != v ){
    op(pos[heavy[u]], pos[v]);
  }
  op(pos[u], pos[u]); // process lca(u, v) ?
}

template <class T>
void update_path(int u, int v, T x){ // O(logN)
  process_path(u, v, [&](int l, int r){
    st.update(l, r, x);
  });
}

template <class T>
T query_path(int u, int v){ // O(logN)
  T ans = {};
  process_path(u, v, [&](int l, int r){
    ans = unite(ans, st.query(l, r));
  });
  return ans;
}
```

4.15. Centroid decomposition

```cpp
void dfs(int u, int pr = 0){ // O(N + M)
  tot[u] = 1;
  for(int v: g[u]) if( v != pr && !rem[v] ){
    dfs(v, u);
    tot[u] += tot[v];
  }
}

int centroid(int u, int x, int pr = 0){ // O(logN)
  for(int v: g[u]){
    if( v != pr && !rem[v] && 2 * tot[v] > x ){
      return centroid(v, tot, u);
    }
  }
  return u;
}

void decompose(int u, int pr = 0){ // O(N * logN)
  dfs(u);
  cdp[u = centroid(u, tot[u])] = pr;
  rem[u] = true;
  for(int v: g[u]) if( !rem[v] ){
    decompose(v, u);
  }
}
```

5. Flows

5.1. Edmonds-Karp

```cpp
template <class F>
struct edmonds_karp{
  vector< vector<int> > g;
  vector< vector<F> > cap, mem;
  vector<int> path;
  vector<F> flow;
  int s, t;
  int n, m;

  edmonds_karp(int nn, int ss = -1, int tt = -1){
    n = nn + 5;
    m = 0;
    s = ss == -1 ? n + 1: ss;
    t = tt == -1 ? n + 2: tt;
    g.resize(n + 5);
    cap.resize(n + 5, vector<F>(n + 5));
    // mem.resize(n + 5, vector<F>(n + 5)); // Original capacity
    path.resize(n + 5);
    flow.resize(n + 5);
  }

  void add_edge(int u, int v, F ncap){
    g[u].pb(v);
    g[v].pb(u);
    cap[u][v] = cap[v][u] = ncap;
    // mem[u][v] = mem[v][u] = ncap;
  }

  F bfs(){
    fill(all(path), -1);
    fill(all(flow), 0);
    queue<int> qu;
    qu.push(s);
    flow[s] = numeric_limits<F>::max() / 2;
    while( !qu.empty() ){
      int u = qu.front();
      qu.pop();
      for(int v: g[u]){
        if( path[v] == -1 && cap[u][v] > 0 ){
          path[v] = u;
          flow[v] = min(flow[u], cap[u][v]);
          if( v == t ){
            return flow[v];
          }
          qu.push(v);
        }
      }
    }
    return 0;
  }

  F max_flow(){ // O(N * (M ^ 2))
    F flow = 0;
    while( F nflow = bfs() ){
      flow += nflow;
      for(int v = t; v != s; v = path[v]){
        int u = path[v];
        cap[u][v] -= nflow;
        cap[v][u] += nflow;
      }
    }
    return flow;
  }

  void min_cut(){
    For1(u, n - 1) if( flow[u] > 0 ){
      for(int v: g[u]) if( !flow[v] && mem[u][v] ){
        // Cut edge u -> v
      }
    }
  }
};
```

5.2. Dinic

```cpp
template <class F>
struct dinic{
  const static F finf = numeric_limits<F>::max() / 2;

  struct edge{
    int v;
    F cap, flow;
    int inv;
    edge(int nv, F ncap, int ninv):
      v(nv), cap(ncap), flow(0), inv(ninv){}
  };

  vector< vector<edge> > g;
  vector<int> dist, ptr;
  int s, t;
  int n, m;

  dinic(int nn, int ss = -1, int tt = -1){
    n = nn + 5;
    m = 0;
    s = ss == -1 ? n + 1: ss;
    t = tt == -1 ? n + 2: tt;
    g.resize(n + 5);
    dist.resize(n + 5);
    ptr.resize(n + 5);
  }

  void add_edge(int u, int v, F cap){
    edge a(v, cap, sz(g[v]));
    edge b(u, 0, sz(g[u]));
    g[u].pb(a);
    g[v].pb(b);
    m += 2;
  }

  bool bfs(){
    fill(all(dist), -1);
    queue<int> qu;
    dist[s] = 0;
    qu.push(s);
    while( !qu.empty() ){
      int u = qu.front();
      qu.pop();
      for(edge &e: g[u]) if( e.cap - e.flow > 0 ){
        if( dist[e.v] == -1 ){
          dist[e.v] = dist[u] + 1;
          qu.push(e.v);
        }
      }
    }
    return dist[t] != -1;
  }

  F dfs(int u, F flow = finf){
    if( flow <= 0 ){
      return 0;
    }
    if( u == t ){
      return flow;
    }
    for(int &cid = ptr[u]; cid < sz(g[u]); cid++){
      edge &e = g[u][cid];
      if( e.cap - e.flow <= 0 ){
        continue;
      }
      if( dist[u] + 1 == dist[e.v] ){
        F nflow = dfs(e.v, min(flow, e.cap - e.flow));
        if( nflow > 0 ){
          e.flow += nflow;
          g[e.v][e.inv].flow -= nflow;
          return nflow;
        }
      }
    }
    return 0;
  }

  F max_flow(){ // O((N ^ 2) * M)
    F flow = 0;
    while( bfs() ){
      fill(all(ptr), 0);
      while( F nflow = dfs(s) ){
        flow += nflow;
      }
```

```
    }
    return flow;
  }
};
```

5.3. Min cost flow

```
template <class F, class C>
struct mcmf{
  const static C cinf = numeric_limits<C>::max() / 2;
  const static F finf = numeric_limits<F>::max() / 2;

  struct edge{
    int v;
    F cap, flow;
    C cost;
    int inv;
    edge(int nv, C ncost, F ncap, int ninv):
      v(nv), cost(ncost), cap(ncap), flow(0), inv(ninv){}
  };

  vector< vector<edge> > g;
  vector<C> mn;
  vector<int> state, path, from;
  int s, t;
  int n, m;

  mcmf(int nn, int ss = -1, int tt = -1){
    n = nn + 5;
    m = 0;
    s = ss == -1 ? n + 1: ss;
    t = tt == -1 ? n + 2: tt;
    g.resize(n + 5);
    mn.resize(n + 5);
    state.resize(n + 5);
    path.resize(n + 5);
    from.resize(n + 5);
  }

  void add_edge(int u, int v, C cost, F cap){
    edge a(v, cost, cap, sz(g[v]));
    edge b(u, -cost, 0, sz(g[u]));
    g[u].pb(a);
    g[v].pb(b);
    m += 2;
  }

  bool bfs(){
    For1(u, n){
      state[u] = 2;
      mn[u] = cinf;
      path[u] = -1;
    }
    deque<int> qu;
    qu.push_back(s);
    state[s] = 1;
    mn[s] = 0;
    while( !qu.empty() ){
      int u = qu.front();
      qu.pop_front();
      state[u] = 0;
      for(edge &e: g[u]) if( e.cap - e.flow > 0 ){
        if( mn[u] + e.cost < mn[e.v] ){
          mn[e.v] = mn[u] + e.cost;
          path[e.v] = u;
          from[e.v] = g[e.v][e.inv].inv;
          if( state[e.v] == 0 || (!qu.empty() && mn[qu.front()] > mn[e.v
              ]) ){
            qu.push_front(e.v);
          }else if( state[e.v] == 2 ){
            qu.push_back(e.v);
          }
          state[e.v] = 1;
        }
      }
    }
    return mn[t] != cinf;
  }

  pair<C, F> min_cost_flow(){ // O((N ^ 2) * M)
    C cost = 0;
    F flow = 0;
    while( bfs() ){
      F nflow = finf;
      for(int u, v = t; v != s; v = u){
        u = path[v];
```

```
        edge &e = g[u][from[v]];
        nflow = min(nflow, e.cap - e.flow);
      }
      for(int u, v = t; v != s; v = u){
        u = path[v];
        g[u][from[v]].flow += nflow;
        g[v][g[u][from[v]].inv].flow -= nflow;
        cost += g[u][from[v]].cost * nflow;
      }
      flow += nflow;
    }
    return make_pair(cost, flow);
  }
};
```

5.4. Hopcroft-Karp

```
struct hopcroft_karp{
  vector< vector<int> > g;
  vector<int> dist;
  vector<int> match;
  int n, m;

  hopcroft_karp(int nn){
    n = nn, m = 0;
    g.resize(n + 5);
    dist.resize(n + 5);
    match.resize(n + 5, 0);
  }

  void add_edge(int u, int v){
    g[u].pb(v);
    g[v].pb(u);
    m += 2;
  }

  bool bfs(){
    queue<int> qu;
    For1(u, n - 1){
      if( match[u] ){
        dist[u] = inf;
      }else{
        dist[u] = 0;
        qu.push(u);
      }
    }
    dist[0] = inf;
    while( !qu.empty() ){
      int u = qu.front();
      qu.pop();
      if( u != 0 ){
        for(int v: g[u]){
          if( dist[u] + 1 < dist[match[v]]){
            dist[match[v]] = dist[u] + 1;
            qu.push(match[v]);
          }
        }
      }
    }
    return dist[0] != inf;
  }

  bool dfs(int u){
    if( u != 0 ){
      for(int v: g[u]){
        if( dist[u] + 1 == dist[match[v]] && dfs(match[v]) ){
          match[u] = v, match[v] = u;
          return true;
        }
      }
      dist[u] = inf;
      return false;
    }
    return true;
  }

  int max_matching(){ // O(N + M)
    int matching = 0;
    while( bfs() ){
      For1(u, n){
        if( !match[u] && dfs(u) ){
          matching++;
        }
      }
    }
    return matching;
```

```
  }
};
```

6. Number Theory

6.1. Inverse

```
void do_inverse(){ // mod < 1e7
  inv[0] = inv[1] = 1ll;
  For(i, 2, mod - 1, +1){
    inv[i] = (inv[mod % i] * (mod - (mod / i)) % mod;
  }
}
----------------------------------------------------------------
void factorials(){ // n! y inv(n!)
  fac[0] = 1ll;
  For1(i, N - 1){
    fac[i] = 1ll * fac[i - 1] * i % mod;
  }
  ifac[N - 1] = fpow(fac[N - 1], mod - 2);
  Rof(i, N - 2, 0, -1){
    ifac[i] = 1ll * ifac[i + 1] * (i + 1) % mod;
  }
}
----------------------------------------------------------------
void magic(){ // p ^ k y inv(p ^ k)
  pw[0] = 1LL;
  For1(i, N - 1){
    pw[i] = 1LL * pw[i - 1] * P % mod;
  }
  ipw[N - 1] = fpow(pw[N - 1], mod - 2);
  Rof(i, N - 2, 0, -1){
    ipw[i] = 1LL * ipw[i + 1] * P % mod;
  }
}
```

6.2. Sieve of Eratosthenes

```
int p[N >> 6];
#define isp(x) !(p[x >> 6] & (1 << ((x >> 1) % 32)))
#define add(x) p[x >> 6] |= (1 << ((x >> 1) % 32))

bool prime(lli x){
  return x == 2 || (x % 2 && isp(x) && x > 1);
}

void sieve(){
  for(lli i = 3; 1LL * i * i < N; i += 2) if( isp(i) ){
    for(lli j = 1LL * i * i; j < N; j += (i << 1)){
      add(j);
    }
  }
}
----------------------------------------------------------------
void factorize_sieve(){ // O(N)
  iota(p, p + N, 0);
  For(i, 2, N - 1, +1) if( p[i] == i ){
    For(j, i, N - 1, +i) if( p[j] == j ){
      p[j] = i;
    }
  }
}
```

6.3. Phi of euler

```
void phi_and_primes(){
  Forn(i, N){
    isp[i] = true;
    phi[i] = i;
  }
  isp[0] = isp[1] = false;
  For1(i, N - 1) if( i > 1 ){
    if( isp[i] ){
      For(j, i, N - 1, +i){
        isp[j] = (i == j);
        phi[j] /= i;
        phi[j] *= i - 1;
      }
    }
```

```
  }
}
```

6.4. GCD / LCM

```
lli gcd(lli a, lli b){
  return b ? gcd(b, a % b) : a;
}

lli lcm(lli a, lli b){
  return a / gcd(a, b) * b;
}

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

6.5. Linear diophantine equations

Let $g = gcd(a, b)$ and let x, y be integers which satisfy the following: $a * x + b * y = c$ then $x' = x + k * \frac{b}{g}$, $y' = y_0 - k * \frac{a}{g}$ are solutions of the given Diophantine equation.

```
bool lde_any_solution(lli a, lli b, lli c, lli &x0, lli &y0, lli &g) {
  g = gcd(abs(a), abs(b), x0, y0);
  if (c % g) return false;
  x0 *= c / g;
  y0 *= c / g;
  if (a < 0) x0 = -x0;
  if (b < 0) y0 = -y0;
  return true;
}

void shift_solution(lli & x, lli & y, lli a, lli b, lli cnt) {
  x += cnt * b;
  y -= cnt * a;
}

lli lde_all_solutions(lli a, lli b, lli c, lli minx, lli maxx, lli miny,
    lli maxy) {
  lli x, y, g;
  if (!lde_any_solution(a, b, c, x, y, g)) return 0;
  a /= g;
  b /= g;
  lli sign_a = a > 0 ? +1 : -1;
  lli sign_b = b > 0 ? +1 : -1;
  shift_solution(x, y, a, b, (minx - x) / b);
  if (x < minx) shift_solution(x, y, a, b, sign_b);
  if (x > maxx) return 0;
  lli lx1 = x;
  shift_solution(x, y, a, b, (maxx - x) / b);
  if (x > maxx) shift_solution(x, y, a, b, -sign_b);
  lli rx1 = x;
  shift_solution(x, y, a, b, -(miny - y) / a);
  if (y < miny) shift_solution(x, y, a, b, -sign_a);
  if (y > maxy) return 0;
  lli lx2 = x;
  shift_solution(x, y, a, b, -(maxy - y) / a);
  if (y > maxy) shift_solution(x, y, a, b, sign_a);
  lli rx2 = x;
  if (lx2 > rx2) swap(lx2, rx2);
  lli lx = max(lx1, lx2);
  lli rx = min(rx1, rx2);
  if (lx > rx) return 0;
  return (rx - lx) / abs(b) + 1;
}
```

6.6. Modular power

```
lli fpow(lli x, lli y){ // O(logY)
  lli r = 1;
  while( y > 0 ){
    if( y & 1 ) r = r * x;
    x = x * x;
    y >>= 1;
  }
  return r;
}
```

6.7. Fibonacci matrix

```
void mul(lli a[2][2], lli b[2][2]){
  lli w = a[0][0] * b[0][0] % mod + a[0][1] * b[1][0] % mod;
  lli x = a[0][0] * b[0][1] % mod + a[0][1] * b[1][1] % mod;
  lli y = a[1][0] * b[0][0] % mod + a[1][1] * b[1][0] % mod;
  lli z = a[1][0] * b[0][1] % mod + a[1][1] * b[1][1] % mod;
  a[0][0] = w % mod, a[0][1] = x % mod;
  a[1][0] = y % mod, a[1][1] = z % mod;
}

lli fib(lli n){ // O(logN)
  if( n == 0 ) return 0;
  n--;
  n %= mod;
  lli mat[2][2] = {{0, 1},{1, 1}};
  lli ans[2][2] = {{0, 1},{1, 1}};
  while( n > 0 ){
    if( n & 1) mul(ans, mat);
    mul(mat, mat);
    n >>= 1;
  }
  return ans[0][1];
}
```

6.8. Miller Test

```
bool check_composite(ulli n, ulli p, ulli d, int s){
  ulli x = fpow(p, d, n);
  if( x == 1 || x == n - 1 ){
    return false;
  }
  For1(it, s - 1){
    x = (__uint128_t) x * x % n;
    if( x == n - 1 ){
      return false;
    }
  }
  return true;
}

bool miller(ulli n){
  if( n < 2 ){
    return false;
  }
  int r = 0;
  ulli d = n - 1;
  while( (d & 1) == 0 ){
    d >>= 1;
    r++;
  }
  for(int p: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}){
    if( n == p )
      return true;
    if( check_composite(n, p, d, r) )
      return false;
  }
  return true;
}
```

6.9. Teoremas

1. Teorema de Fermat de la suma de 2 cuadrados: Un primo puede ser expresado como la suma de 2 cuadrados
$$p = x^2 + y^2 \tag{1}$$
si: $p \equiv 1(mod 4)$
2. Teorema de la suma de 2 cuadrados: Un numero n puede ser expresado como la suma de 2 cuadrados si y solo si su factorización en primos no contiene ningun $p \equiv 3(mod 4)$ elevado a una potencia impar.
3. Teorema de Wilson:
$$p|(p-1)! + 1 \tag{2}$$
4. Prime number theorem
$$\phi(n)/n = n/log(n) \tag{3}$$
Probabilidad de que un numero menor que n sea primo.
5. El producto de N numeros enteros consecutivos es divisible entre el producto de los primeros N enteros positivos.

7. Numeros

1. Lucas Numbers:
$$\sum L_1 = 1 \tag{4}$$
$$\sum L_n = F_{n-1} + F_{n+1} \tag{5}$$

2. Twin primes:

$$T(p_i, p_j) si |p_i - p_j| = 2 \tag{6}$$

$$a^{\phi(m)} \equiv 1(mod m) \tag{7}$$

a y m son coprimos

3.

$$\phi(ab) = \phi(a) * \phi(b) * \frac{d}{\phi(d)} \tag{8}$$

donde d es gcd(a,b)

7.1. Formulas

1. Si a,b impar:

$$a^2 - b^2 | 8 \tag{9}$$

2.

$$n/(n-1)/(n-2).../1 = \frac{n^2}{n!} \tag{10}$$

3.

$$n(n^2-1)(3n+2)|24 \tag{11}$$

4.

$$\sum i^2 = \frac{n(n+1)(2n+1)}{6} \tag{12}$$

5.

$$\sum x^i = \frac{x^n - 1}{x - 1} \tag{13}$$

6.

$$\sum x^3 = (\sum x)^2 \tag{14}$$

7.

$$1*2 + 2*3 + 3*4 + 4*5... = \frac{n(n+1)(n+2)}{3} \tag{15}$$

8.

$$\sum F_i = F_(n+2) - 1 \tag{16}$$

Donde F don los numeros de Fibonnaci

7.2. Ternas pitagoricas

1. Sea gcd(m,n)=1

$$a = n^2 - m^2 \tag{17}$$

$$b = 2mn \tag{18}$$

$$c = m^2 + n^2 \tag{19}$$

2. En la terna pitagorica:

   a) Exactamente uno entre a y b, es multiplo de 2.
   b) Exactamente uno entre a y b, es multiplo de 3.
   c) Exactamente uno entre a y b, es multiplo de 4.
   d) Exactamente uno entre a,b,c es multiplo de 5.

7.3. LCM Y GCD

a) Suma de GCD

$$\sum gcd(i,n) = \sum_{d|n} d\phi(n/d) \tag{20}$$

b) Suma de LCM

$$\sum lcm(i,n) = n\frac{(1 + \sum_{d|n} \phi(d)d)}{2} \tag{21}$$

7.4. Phi de Euler

a)

$$\phi(p^k) = p^k - ^{k-1} \tag{22}$$

b)

$$\phi(ab) = \phi(a) * \phi(b) * \frac{d}{\phi(d)} \tag{23}$$

c)

$$a^{\phi m} \equiv 1 mod(m) \tag{24}$$

d)

$$a^{\phi m} \equiv 1 mod(m) \tag{25}$$

Con m y a coprimos.

e)

$$a^n \equiv a^{nmod\phi(m)}(modm) \tag{26}$$

f)

$$x^n \equiv x^{\phi(m)+(nmod\phi(m))}(modm) \tag{27}$$

Si x y m no son coprimos. y n mayor o igual log m

7.5. Diphantine Equations

$ax + by = c$ Tiene solucion si g=gcd(a,b) divide a c.

Todas las soluciones de la ecuacion se pueden escribir de la forma:

$$x = x_0 + k(\frac{b}{g}) \tag{28}$$

$$y = y_0 - k(\frac{a}{g}) \tag{29}$$

8. Math

8.1. Identidades

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

$\sum_{i=0}^{n} i\binom{n}{i} = n * 2^{n-1}$

$\sum_{i=m}^{n} i = \frac{n(n+1)}{2} - \frac{m(m-1)}{2} = \frac{(n+1-m)(n+m)}{2}$

$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$

$\sum_{i=0}^{n} i(i-1) = \frac{8}{6}(\frac{n}{2})(\frac{n}{2}+1)(n+1)$ (doubles) $\rightarrow$ Sino ver caso impar y par

$\sum_{i=0}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4} = [\sum_{i=1}^{n} i]^2$

$\sum_{i=0}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$

$\sum_{i=0}^{n} i^p = \frac{(n+1)^{p+1}}{p+1} + \sum_{k=1}^{p} \frac{B_k}{p-k+1}\binom{p}{k}(n+1)^{p-k+1}$

$r = e - v + k + 1$

8.2. Ec. Caracteristica

$a_0T(n) + a_1T(n-1) + ... + a_kT(n-k) = 0$

$p(x) = a_0x^k + a_1x^{k-1} + ... + a_k$

Sean $r_1, r_2, ..., r_q$ las raíces distintas, de mult. $m_1, m_2, ..., m_q$

$T(n) = \sum_{i=1}^{q} \sum_{j=0}^{m_i-1} c_{ij}n^j r_i^n$

Las constantes c_{ij} se determinan por los casos base.

8.3. Teorema Chino del Resto

$$y = \sum_{j=1}^{n}(x_j * (\prod_{i=1,i\neq j}^{n} m_i)_{m_j}^{-1} * \prod_{i=1,i\neq j}^{n} m_i)$$

8.4. Tablas y cotas (Primos, Divisores, Factoriales, etc)

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 ($\in$ `tint`)
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

Primos

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
1009 1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087 1091 1093 1097
1103 1109 1117 1123 1129 1151 1153 1163 1171 1181 1187 1193 1201 1213 1217 1223
1229 1231 1237 1249 1259 1277 1279 1283 1289 1291 1297 1301 1303 1307 1319 1321
1327 1361 1367 1373 1381 1399 1409 1423 1427 1429 1433 1439 1447 1451 1453 1459
1471 1481 1483 1487 1489 1493 1499 1511 1523 1531 1543 1549 1553 1559 1567 1571
1579 1583 1597 1601 1607 1609 1613 1619 1621 1627 1637 1657 1663 1667 1669 1693
1697 1699 1709 1721 1723 1733 1741 1747 1753 1759 1777 1783 1787 1789 1801 1811
1823 1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1931 1933 1949
1951 1973 1979 1987 1993 1997 1999 2003 2011 2017 2027 2029 2039 2053 2063 2069
2081 2083 2087 2089 2099 2111 2113 2129 2131 2137 2141 2143 2153 2161 2179 2203
2207 2213 2221 2237 2239 2243 2251 2267 2269 2273 2281 2287 2293 2297 2309 2311

2333 2339 2341 2347 2351 2357 2371 2377 2381 2383 2389 2393 2399 2411 2417 2423
2437 2441 2447 2459 2467 2473 2477 2503 2521 2531 2539 2543 2549 2551 2557 2579
2591 2593 2609 2617 2621 2633 2647 2657 2659 2663 2671 2677 2683 2687 2689 2693
2699 2707 2711 2713 2719 2729 2731 2741 2749 2753 2767 2777 2789 2791 2797 2801
2803 2819 2833 2837 2843 2851 2857 2861 2879 2887 2897 2903 2909 2917 2927 2939
2953 2957 2963 2969 2971 2999 3001 3011 3019 3023 3037 3041 3049 3061 3067 3079
3083 3089 3109 3119 3121 3137 3163 3167 3169 3181 3187 3191 3203 3209 3217 3221
3229 3251 3253 3257 3259 3271 3299 3301 3307 3313 3319 3323 3329 3331 3343 3347
3359 3361 3371 3373 3389 3391 3407 3413 3433 3449 3457 3461 3463 3467 3469 3491
3499 3511 3517 3527 3529 3533 3539 3541 3547 3557 3559 3571

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

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

Logaritmo base 2

$1 = 0$	$10^{10} = 33.2192809489$
$10 = 3.3219280949$	$10^{11} = 36.5412090438$
$10^2 = 6.6438561898$	$10^{12} = 39.8631371386$
$10^3 = 9.9657842847$	$10^{13} = 43.1850652335$
$10^4 = 13.2877123795$	$10^{14} = 46.5069933284$
$10^5 = 16.6096404744$	$10^{15} = 49.8289214233$
$10^6 = 19.9315685693$	$10^{16} = 53.1508495182$
$10^7 = 23.2534966642$	$10^{17} = 56.4727776131$
$10^8 = 26.5754247591$	$10^{18} = 59.7947057080$
$10^9 = 29.8973528540$	$10^{19} = 63.1166338029$

Numeros a binario

0: 0	8: 1000	16: 10000	24: 11000
1: 1	9: 1001	17: 10001	25: 11001
2: 10	10: 1010	18: 10010	26: 11010
3: 11	11: 1011	19: 10011	27: 11011
4: 100	12: 1100	20: 10100	28: 11100
5: 101	13: 1101	21: 10101	29: 11101
6: 110	14: 1110	22: 10110	30: 11110
7: 111	15: 1111	23: 10111	31: 11111

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* $n/\neg\exists n' < n, \sigma_1(n') \geqslant \sigma_1(n)$
$\sigma_1(96) = 252$; $\sigma_1(108) = 280$; $\sigma_1(120) = 360$; $\sigma_1(144) = 403$; $\sigma_1(168) = 480$
$\sigma_1(960) = 3048$; $\sigma_1(1008) = 3224$; $\sigma_1(1080) = 3600$; $\sigma_1(1200) = 3844$
$\sigma_1(4620) = 16128$; $\sigma_1(4680) = 16380$; $\sigma_1(5040) = 19344$; $\sigma_1(5760) = 19890$
$\sigma_1(8820) = 31122$; $\sigma_1(9240) = 34560$; $\sigma_1(10080) = 39312$; $\sigma_1(10920) = 40320$
$\sigma_1(32760) = 131040$; $\sigma_1(35280) = 137826$; $\sigma_1(36960) = 145152$; $\sigma_1(37800) = 148800$
$\sigma_1(60480) = 243840$; $\sigma_1(64680) = 246240$; $\sigma_1(65520) = 270816$; $\sigma_1(70560) = 280098$
$\sigma_1(95760) = 386880$; $\sigma_1(98280) = 403200$; $\sigma_1(100800) = 409448$
$\sigma_1(491400) = 2083200$; $\sigma_1(498960) = 2160576$; $\sigma_1(514080) = 2177280$
$\sigma_1(982800) = 4305280$; $\sigma_1(997920) = 4390848$; $\sigma_1(1048320) = 4464096$
$\sigma_1(4979520) = 22189440$; $\sigma_1(4989600) = 22686048$; $\sigma_1(5045040) = 23154768$
$\sigma_1(9896040) = 44323200$; $\sigma_1(9959040) = 44553600$; $\sigma_1(9979200) = 45732192$

8.5. Geometria

8.6. Geo geo

1. Ley de senos
$$\frac{a}{sinA} = \frac{b}{sinB} = \frac{c}{sinC} \tag{30}$$

2. Ley de cosenos
$$a^2 = b^2 + c^2 - 2bccos(A) \tag{31}$$

Teorema de Pick: (Area, puntos interiores y puntos en el borde) $A = I + \frac{B}{2} - 1$

3. Calcular puntos enteros sobre una recta A(x, y) -¿B(x, y): gcd(A.x - B.x, A.y - B.y)

9. Strings

9.1. Hash (usa al menos dos hashes)

```
lli prefix_query(int l, int r){
  return (h[prefix][r] - h[prefix][l - 1] + mod) % mod * ipw[l - 1] %
      mod;
}

lli suffix_query(int l, int r){
  return (h[suffix][l] - h[suffix][r + 1] + mod) % mod * ipw[n - r] %
      mod;
}

bool palindrome(int l, int r){
  l++, r++;
  return prefix_query(l, r) == suffix_query(l, r);
}

void pre(string &s){
  n = sz(s);
  s = "$" + s;
  h[prefix][0] = 0;
  For1(i, n){
    h[prefix][i] = (h[prefix][i - 1] + 1LL * (s[i] - 'a' + 1) * pw[i -
        1]) % mod;
  }
  h[suffix][n + 1] = 0;
  Rof(i, n, 1, -1){
    h[suffix][i] = (h[suffix][i + 1] + 1LL * (s[i] - 'a' + 1) * pw[n - i
        ]) % mod;
  }
}
```

9.2. KMP

period = n - lps[n - 1], period(abcabc) = 2, $n \% period \equiv 0$

```
void pre(string &s){ // O(N)
  lps[0] = 0;
  For1(i, sz(s) - 1){
    int j = lps[i - 1];
    while( j > 0 && s[i] != s[j] ){
      j = lps[j - 1];
    }
    lps[i] = j + (s[i] == s[j]);
  }
}

int kmp(string &s, string &t){ // O(N), how many times t occurs in s
  pre(t);
  int j = 0, tot = 0;
  Forn(i, sz(a)){
    while( j > 0 and s[i] != t[j] ){
      j = lps[j - 1];
    }
    if( s[i] == t[i] ){
      j++;
    }
    if( j == sz(t) ){
      tot++; // pos: i - sz(t);
    }
  }
  return tot;
}
```

9.3. Manacher algorithm

```
int L[MAXN * 2 + 5];
string manacher(string s) {
  int N = s.len() * 2 + 1;
  L[0] = 0;
  L[1] = 1;
  int C = 1;
  int R = 2;
  int iMirror;
  int maxLPSLength = 0;
```

```
  int maxLPSCenterPosition = 0;
  int diff = -1;
  FOR(i, 2, N) {
    iMirror  = 2*C-i;
    L[i] = 0;
    diff = R - i;
    if(diff > 0) L[i] = min(L[iMirror], diff);
    while ( ((i + L[i]) < N && (i - L[i]) > 0) &&
      ( ((i + L[i] + 1) % 2 == 0) ||
      (s[(i + L[i] + 1)/2] == s[(i - L[i] - 1)/2] )))
      L[i]++;

    if(L[i] > maxLPSLength) {
      maxLPSLength = L[i];
      maxLPSCenterPosition = i;
    }
    if (i + L[i] > R)  {
      C = i;
      R = i + L[i];
    }
  }
  int ini = (maxLPSCenterPosition - maxLPSLength)/2;
  int fin = ini + maxLPSLength;
  return s.substr(ini, fin-ini);
}
```

9.4. Trie

```
struct prefix_tree{
  vector< map<char, int> > trie
  vector<bool> isw;

  prefix_tree(){
    new_node();
  }

  int new_node(){
    trie.pb(map<char, int>());
    isw.pb(false);
    return sz(trie) - 1;
  }

  void insert(string &s, int u = 0){
    for(char c: s){
      if( !trie[u][c] ){
        trie[u][c] = new_node();
      }
      u = trie[u][c];
    }
    isw[u] = true;
  }

  bool find(string &s, int u = 0){
    for(char c: s){
      if( !trie[u][c] ){
        return false;
      }
      u = trie[u][c];
    }
    return isw[u];
  }
};
```

9.5. Suffix automaton

```
struct suffix_automaton{
  vector< map<char, int> > trie;
  vector<int> link;
  vector<int> len;
  int last;

  suffix_automaton(){
    last = new_node();
  }

  int new_node(){
    trie.pb({});
    link.pb(-1);
    len.pb(0);
    return sz(trie) - 1;
  }

  void extend(char c){
    int u = new_node();
    len[u] = len[last] + 1;
    int p = last;
```

```
    while( p != -1 && !trie[p].count(c) ){
      trie[p][c] = u;
      p = link[p];
    }
    if( p == -1 ){
      link[u] = 0;
    }else{
      int q = trie[p][c];
      if( len[p] + 1 == len[q] ){
        link[u] = q;
      }else{
        int clone = new_node();
        len[clone] = len[p] + 1;
        trie[clone] = trie[q];
        link[clone] = link[q];
        while( p != -1 && trie[p][c] == q ){
          trie[p][c] = clone;
          p = link[p];
        }
        link[q] = link[u] = clone;
      }
    }
    last = u;
  }
-------------------------------------------------------------
  bool find(string &s, int u){
    for(char c: s){
      if( !trie[u].count(c) ){
        return false;
      }
      u = trie[u][c];
    }
    return true;
  }
-------------------------------------------------------------
  int longest_common_substring(string &s, int u = 0){
    int mx = 0, cur_len = 0;
    for(char c: s){
      while( u && !trie[u].count(c) ){
        u = link[u];
        cur_len = len[u];
      }
      if( trie[u][c] ){
        u = trie[u][c];
        cur_len++;
      }
      mx = max(mx, cur_len);
    }
    return mx;
  }
-------------------------------------------------------------
  // different substrings
  // dp[u] = trie[u].empty() ? 1: 1 + all(trie[u])
  string lexicographically_kth_substring(lli kth, int u = 0){
    string s = "";
    while( kth > 0 ){
      for(auto &[c, v]: trie[u]){
        if( kth <= dp[v] ){
          s.pb(c);
          kth--;
          u = v;
          break;
        }
        kth -= dp[v];
      }
    }
    return s;
  }
-------------------------------------------------------------
  // occurs[u] = 1, occurs[clone] = 0
  void all_occurrences_in(string &s){
    vector<int> lens;
    Forn(u, sz(trie)){
      lens.pb(u);
    }
    sort(all(lens), [=](int u, int v){
      return len[u] > len[v];
    });
    for(int u: lens){
      occurs[link[u]] += occurs[u];
    }
  }

  int how_many_times(string &s, int u = 0){
    for(char c: s){
      if( !trie[u][c] ){
```

```
        return 0;
      }
      u = trie[u][c];
    }
    return occurs[u];
  }
};
```

9.6. Aho corasick

```
struct aho_corasick{
  vector< map<char, int> > trie;
  vector<int> cnt;
  vector<int> link;

  aho_corasick(){
    new_node();
  }

  int new_node(){
    trie.pb(map<char, int>());
    cnt.pb(0);
    link.pb(0);
    return sz(trie) - 1;
  }

  void insert(string &s, int u = 0){
    for(char c: s){
      if( !trie[u][c] ){
        trie[u][c] = new_node();
      }
      u = trie[u][c];
    }
    cnt[u]++;
  }

  int suffix(int u, char c){
    while( u && !trie[u][c] ){
      u = link[u];
    }
    return trie[u][c];
  }

  void push_links(){
    queue<int> qu;
    qu.push(0);
    while( !qu.empty() ){
      int u = qu.front();
      qu.pop();
      for(char c: alpha){
        int &v = trie[u][c];
        if( v == 0 ){
          v = trie[link[u]][c];
          continue;
        }
        link[v] = u ? suffix(link[u], c): 0;
        cnt[v] += cnt[link[v]];
        qu.push(v);
      }
    }
  }

  int match(string &s, int u = 0){
    int ans = 0;
    for(char c: s){
      u = suffix(u, c);
      ans += cnt[u];
    }
    return ans;
  }
};
```

10. Game Theory

10.1. Grundy Numbers

```
int mex(unordered_set<int> &st){
  int x = 0;
  while( st.find(x) != st.end() ) x++;
  return x;
}

int grundy(int n){
  if( n == 0 ) return 0;
  unordered_set<int> st;
  For1(i, 3){
```

```
    st.insert(grundy(n - i));
  }
  return mex(st);
}
```

11. Dynamic Programming

11.1. Matrix Chain Multiplication

```
lli calc(int l, int r){
  if( l >= r ){
    return 0ll;
  }
  lli &ans = dp[l][r];
  if( ans == -1 ){
    ans = inf;
    For(k, l, r, +1){
      ans = min(ans, calc(l, k) + calc(k + 1, r) + inc());
    }
  }
  return ans;
}
```

11.2. Knapsack 0/1

```
void knapsack01(){
  Forn(i, n){ // cant elementos
    // Rof(j, size(dp) - 1, v[i].snd, -1){ // Se tiene un unico elemento
    For(j, v[i].snd, size(dp) - 1, 1){ // Infinita cantidad de un mismo
        elemento
      dp[j] = min(dp[j], dp[j - v[i].snd] + v[i].fst); // min, max
    }
  }
}
```

11.3. Convex Hull Trick

```
struct line{
  // with eq. y = mx + c
  mutable lli m, c, p;
  bool operator < (const line &l) const{ return m < l.m; }
  bool operator < (const lli &x) const{ return p < x; }

  lli eval(lli x){
    return m * x + c;
  }
};

struct convex_hull: multiset<line, less<>> {
  // for doubles, use inf = 1/.0, div(a,b) = a/b
  lli div(lli a, lli 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->c > y->c ? inf: -inf);
    }else{
      x->p = div(y->c - x->c, x->m - y->m);
    }
    return x->p >= y->p;
  }

  void add(lli m, lli c){
    auto z = insert({m, c, 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));
    }
  }

  // solve dp[i] = max{j < i; dp[j] + b[j] * a[i]}
  // (b[j] >= b[j + 1]) or (a[i] <= a[i + 1])
  lli query(lli x){
    auto l = *lower(x);
    return l.eval(x);
  }
};
```

11.4. Kadane

```
template <class T>
void kadane(){
  T mx = a[0], sum = a[0];
  For1(i, n - 1){
    sum = max(a[i], sum + a[i]);
    mx = max(mx, sum);
  }
  return mx;
}
```

12. Combinatorics

12.1. n! mod p

```
lli factmod(lli n, lli p){
  lli r = 1;
  for(; n > 1; n /= p){
    r = (r * ((n / p) % 2 ? p - 1: 1)) % p;
    For(i, 2, n % p, +1) r = r * i % p;
  }
  return r % p;
}
```

12.2. Pascal's triangle

```
void pascal(int N){
  Forn(n, N - 1){
    choose[n][0] = choose[n][n] = 1;
    For1(k, n - 1){
      choose[n][k] = choose[n - 1][k - 1] + choose[n - 1][k];
    }
  }
}
```

12.3. Combi

```
lli choose(lli n, lli k){
  return 1ll * fac[n] * ifac[k] % mod * ifac[n - k] % mod;
}

lli choose(lli n, lli k){
  double r = 1;
  For1(i, k) r = r * (n - k + i) / i;
  return lli(r + 0.01);
}
```

12.4. Catalan numbers

```
// Python Solution
catalan = [0 for i in range(150 + 5)]
def fcatalan(n):
    catalan[0] = 1
    catalan[1] = 1
    for i in range(2, n + 1):
        catalan[i] = 0
        for j in range(i):
            catalan[i] = catalan[i] + catalan[j] * catalan[i - j - 1]

fcatalan(151)
```

12.5. Identities

1. $$\binom{n}{n} = \binom{n}{0} = 1 \tag{32}$$

2. $$\binom{n}{1} = n \tag{33}$$

3. $$\binom{n}{k} = \binom{n}{n-k} \tag{34}$$

4. $$\binom{n-1}{k-1} = \binom{n-1}{k} = \binom{n}{k} \tag{35}$$

5. $$2^n = \sum_{i=0}^{n} \binom{n}{i} \tag{36}$$

6. Hockey Stick Rule - Pascal triangle

$$\sum_{i=0}^{r} C_i^{n+1} = \sum_{i=0}^{r} C_n^{n+i} = C_r^{n+r+1} = C_{n+1}^{n+r+1} \tag{37}$$

7. Burnside's lemma

   (useful in taking account of symmetry when counting mathematical objects)

$$|X/G| = \frac{1}{|G|}\sum_{g\in G}|X^g| \tag{38}$$

13. Ayudamemoria

Cant. decimales

```
#include <iomanip>
cout << setprecision(2) << fixed;
```

Rellenar con espacios(para justificar)

```
#include <iomanip>
cout << setfill('␣') << setw(3) << 2 << endl;
```

Leer hasta fin de linea

```
#include <sstream>
//hacer cin.ignore() antes de getline()
while(getline(cin, line)){
     istringstream is(line);
     while(is >> X)
       cout << X << "␣";
     cout << endl;
}
```

Aleatorios

```
#define RAND(a, b) (rand()%(b-a+1)+a)
srand(time(NULL));
```

Doubles Comp.

```
const double EPS = 1e-9;
x == y  <=> fabs(x-y) < EPS
x >  y  <=> x > y + EPS
x >= y  <=> x > y - EPS
```

Expandir pila

```
#include <sys/resource.h>
rlimit rl;
getrlimit(RLIMIT_STACK, &rl);
rl.rlim_cur=1024L*1024L*256L;//256mb
setrlimit(RLIMIT_STACK, &rl);
```

C++11

```
g++ --std=c++11 a.cpp && ./a.out < in.txt > out.txt
```

Leer del teclado

```
freopen("/dev/tty", "a", stdin);
```

Iterar subconjunto

```
for(int sbm=bm; sbm; sbm=(sbm-1)&bm)
```

File setup

```
//tambien se pueden usar comas: {a, x, m, l}
touch {a..l}.in; tee {a..l}.cpp < tem.cpp
```

14. Template

```
#include <bits/stdc++.h>
#define For(i, a, b, c) for(int i = (a); i <= (b); i += (c))
#define Rof(i, a, b, c) for(int i = (a); i >= (b); i += (c))
#define Forn(i, n) For(i, 0, (n) - 1, +1)
#define For1(i, n) For(i, 1, (n), +1)
#define sz(x) (int) (x).size()
#define all(x) (x).begin(), (x).end()
#define _ ios_base::sync_with_stdio(0), cin.tie(0), cout.tie(0), cout.
    precision(15);

#define f first
#define s second
#define pb push_back
#define lower lower_bound
#define upper upper_bound
#define ENDL '\n'
using namespace std;

typedef long long lli;
```

```
19 typedef pair<int, int> ii;
20 typedef long double ld;
21
22 #define deb(x) cout << #x": " << (x) << endl;
23
24 int main(){ _
25   return 0;
26 }
```

---

15. Errores comunes, no metas penalty hoy, porfa

¿Ya consideraste todos los casos?

- Sin nadie lo lleva, es porque es dificil, espera a que alguien lo suba :p
- Falta de un long long (WA) :c
- const int N es incorrecta (WA, RTE)
- Casos de n pequeña, especiales (RTE)
- Division entre 0 (WA, RTE)
- Acceso a una posicion no existente (RTE)
- setprecision no correcto (WA)
- Ciclo infinito, revisa esos while(true) (TLE)
- Reiniciar las variables, arreglos, estructuras sin son MUCHOS casos (WA)
- mx/mn incorrectos (WA)
- Enviar debugs (WA)
- Uso de una variable que no es (WA)
- Sigue intentando, el penalty es hasta que sea AC :p
- long double, el double no sirve :'c
- eps = 1e-9 es importante, sino NO va a jalar
- Haz casos ptm!