Repositorio en c++

Universidad de la amazonia

28 de marzo de 2018

Índice

1. Estructuras de datos

1.1. tablas aditivas

Construccion O(n)

```
void build(){
    memset(tab2, 0, sizeof(tab2));
    //en tab2 se guarda tabla aditiva de tab
    tab2[1][1] = tab[0][0];
    for (int i = 2; i < 6; i++) tab2[i][1] = tab2[i-1][1] + tab[i - 1][0];
    for (int j = 2; j < 6; j++) tab2[1][j] = tab2[1][j-1] + tab[0][j - 1];

    for (int i = 2; i < 6; i++)
    for (int j = 2; j < 6; j++)
            tab2[i][j] = tab2[i][j - 1] + tab2[i - 1][j] + tab[i - 1][j - 1] - tab2[i - 1][j - 1];
    //ejemplo en matriz de 5X5 buscar acumulado de 2,2 hasta 5,5
    //tab2[5][5] - tab2[2][5] - tab2[5][2] + tab2[2][2]
}
```

1.2. disjoint set union find

Construccion O(n)

asocia elementos en conjuntos de arboles.

```
struct union_find{
  int padre[100];
  int rango[100];
  vector<int> grupo[100];

  void iniciar(int n){
    for (int i = 0; i < n; i++) {
      padre[i] = i;
      rango[i] = 0;
      grupo[i].clear();
      grupo[i].push_back(i);
    }
  }

  int raiz(int x){
    if(padre[x] == x) return x;
    return raiz(padre[x]);
  }

  void unir(int x, int y){
    x = raiz(x);
    y = raiz(y);
    if(x == y) return;

    if(rango[x] > rango[y]){
      padre[y] = x;
      grupo[x].insert(grupo[x].begin(), grupo[y].begin(), grupo[y].end());
      grupo[y].clear();
      return;
    }

    padre[x] = y;
    grupo[y].insert(grupo[y].begin(), grupo[x].begin(), grupo[x].end());
      grupo[x].clear();
    if(rango[y] == rango[x]) rango[y]++;
  }

  bool MismoGrupo(int x, int y){
    return raiz(x) == raiz(y);
  }

  void grupo_n(int n){
      cout << "elementos en el grupo de " << n << endl;
      n = raiz(n);
      for(int i = 0; i < grupo[n].size(); i++) cout << grupo[n][i] << " ";
      cout << endl;
  }
};
```

1.3. union find con compresion de caminos

asocia elementos de manera simple
metodo mismoGrupo es el mismo del union-find normal.

```
struct union_find{
  int padre[MAX], rango[MAX];

  void iniciar(int n){
    for (int i = 0; i < n; i++) {
      padre[i] = i;
      rango[i] = 0;
    }
  }

    int raiz(int x){
        if(x == padre[x] ) return x;
        else return padre[x] = raiz(padre[x]);
    }

  void unir(int x, int y){
    x = raiz(x);
    y = raiz(y);
    if(x == y) return;

    if(rango[x] > rango[y]){
      padre[y] = x;
      return;
    }

    padre[x] = y;
    if(rango[y] == rango[x]) rango[y]++;
  }
};
```

1.4. segment tree

Ejemplo de RMQ (Range Minium Query)
Contruccion O(n)
Consulta O(log n)
Update O(log n)

```
const int MAX = 4 * 1000;//poner 4 * longitud maxima

struct segment_tree{
    int st[MAX];
    vi A;
    int n, tamst;

    int mov_izq(int index){ return index << 1; }
    int mov_der(int index){ return (index << 1) + 1; }

    void construir(int pos, int izq, int der){
        if(izq == der){
            st[pos] = A[der];
            return;
        }

        construir(mov_izq(pos), izq, (izq + der) >> 1);
        construir(mov_der(pos), ((izq + der) >> 1) + 1, der);
        int aux1 = mov_izq(pos), aux2 = mov_der(pos);
        st[pos] = min(st[aux1], st[aux2]);
    }

    void iniciar(vi arr){//metodo a invocar
        A = arr;
        n = A.size();
        tamst = n << 2;
        construir(1, 0, n - 1);
    }

    int query(int pos, int izq, int der, int i, int j){
        if(i > der || j < izq) return -1;
        if(i <= izq && j >= der) return st[pos];

        int aux1 = query(mov_izq(pos), izq, (izq + der) >> 1, i
            , j);
        int aux2 = query(mov_der(pos), ((izq + der) >> 1) + 1,
            der, i, j);
        if(aux1 == -1) return aux2;
        if(aux2 == -1) return aux1;

        return min(aux1, aux2);
    }
```

```
    int RMQ(int i, int j){//metodo a invocar
        return query(1, 0, n-1, i, j);
    }

    int cambiar(int pos, int izq, int der, int index, int nuevo
        ){
        if(index > der || index < izq) return st[pos];
        if(der == index && izq == index){
            A[index] =  nuevo;
            return st[pos] = nuevo;
        }

        int aux1 = cambiar(mov_izq(pos), izq, (izq + der) >> 1,
             index, nuevo);
        int aux2 = cambiar(mov_der(pos), ((izq + der) >> 1) +
            1, der, index, nuevo);
        return st[pos] = min(aux1, aux2);
    }

    int update(int index, int num){//metodo a invocar
        return cambiar(1, 0, n-1, index, num);
    }
};
```

1.5. segment tree con lazy propagation

Permite actualizar rangos del arbol en O(log n).
solo estan los metodos nuevos y los que hay que actualizar, lo demas es lo mismo del segment tree normal.

```
    int lazy[MAX];

    void construir(int pos, int izq, int der){
        lazy[pos] = -1;//reiniciar lazy
        if(izq == der){
            st[pos] = A[der];
            return;
        }

        construir(mov_izq(pos), izq, (izq + der) >> 1);
        construir(mov_der(pos), ((izq + der) >> 1) + 1, der);
        int aux1 = mov_izq(pos), aux2 = mov_der(pos);
        st[pos] = min(st[aux1], st[aux2]);
    }

    int query(int pos, int izq, int der, int i, int j){
        if(i > der || j < izq) return -1;
        solve_lazy(pos, izq, der);//resolver algun lazy
            pendiente
        if(i <= izq && j >= der) return st[pos];

        int aux1 = query(mov_izq(pos), izq, (izq + der) >> 1, i
            , j);
        int aux2 = query(mov_der(pos), ((izq + der) >> 1) + 1,
            der, i, j);
        if(aux1 == -1) return aux2;
        if(aux2 == -1) return aux1;

        return min(aux1, aux2);
    }

    void solve_lazy(int pos, int izq, int der){//resolver lazy
        if(lazy[pos] == -1) return;

        st[pos] = lazy[pos];
        if(izq != der){
            lazy[mov_izq(pos)] = lazy[mov_der(pos)] = lazy[pos
                ];
        }
        lazy[pos] = -1;
    }

    int lazy_propagation(int pos, int izq, int der, int i, int
        j, int nuevo){
        solve_lazy(pos, izq, der);
        if(i > der || j < izq) return st[pos];

        if(i <= izq && j >= der){
            lazy[pos] = nuevo;
            solve_lazy(pos, izq, der);
            return st[pos];
        }
```

```

        int aux1 = lazy_propagation(mov_izq(pos), izq, (izq + der) >> 1, i, j, nuevo);
        int aux2 = lazy_propagation(mov_der(pos), ((izq + der) >> 1) + 1, der, i, j, nuevo);
        return st[pos] = min(aux1, aux2);
    }

    int update(int i, int j, int nuevo){//metodo a invocar
        return lazy_propagation(1, 0, n-1, i, j, nuevo);//propagar lazy
    }
```

1.6. Arbol binario indexado

Arbol de Fenwick, estructura para el RSM(Range Sum Query)
Construccion O(n log n)
Consulta O(log k)
Update O(log n)

```
struct FenwickTree{
    vi ft;

    void construir(int n){//indexamos desde 1
        ft.assign(n + 1, 0);
    }

    void construir(vi &v){
        ft.assign(v.size() + 1, 0);
        for(int i = 1; i <= v.size(); i++)
            actualizar(i, v[i - 1]);
    }

    int lsOne(int n){//bit menos significativo en 1
        return n & (-n);
    }

    int rsq(int i){//suma de 1 hasta i
        int acum = 0;
        for(; i; i -= lsOne(i)) acum+=ft[i];
        return acum;
    }

    int rsq(int i, int j){//suma de i hasta j
        return rsq(j) - ((i==1)? 0: rsq(i - 1));
    }

    void actualizar(int pos, int n){//n = nuevo - anterior
        for(; pos < ft.size(); pos += lsOne(pos))
            ft[pos] += n;
    }
};
```

2. Grafos

2.1. Dijkstra

Ruta minima O((n + m)log n)

```
typedef pair<int, int> ii;//peso, nodo
typedef vector<ii> vii;
typedef vector<vii> vvii;
typedef vector<int> vi;
#define inf 1000000000
vi padre;//opcional, usar cuando se necesite el camino.

vi dijkstra(vvii &grafo, int nodo, int tam){
    padre.assign(tam + 1, -1);
    vi dis(tam + 1, inf);
    priority_queue<ii> cola;
    cola.push(ii(-0, nodo));
    int peso, aux;
    ii par, par2;

    while(cola.size()){
        par = cola.top();
        cola.pop();
        peso = -par.first;
        nodo = par.second;

        if(dis[nodo] <= peso) continue;
        dis[nodo] = peso;

```

```
        for(int i = 0; i < grafo[nodo].size(); i++){
            par2 = grafo[nodo][i];
            aux = dis[nodo] + par2.first;
            if(dis[par2.second] > aux){
                cola.push(ii(-aux, par2.second));
                padre[par2.second] = nodo;
            }
        }
    }

    return dis;
}

void camino(int n){//imprimir el camino
    if(padre[n] == -1) printf("%d", n);
    else{
        camino(padre[n]);
        printf(" %d", n);
    }
}
```

2.2. Bellman-Ford

Ruta minima con pesos negativos O(n^2)

```
typedef pair<int, int> ii;
typedef pair<int, ii> iii;//(peso, nodo padre, nodo hijo)
typedef vector<int> vi;

#define mpdii(a, b, c) iii(a, ii(b, c))
#define inf 1000000000

vector<iii> grafo; //lista de incidencia
vi padre;//opcional

bool BellmanFord(vector<iii> &lista, int nodos, int inicio, vector<int> &dis){

    dis.assign(nodos, inf);
  for(int i = 0; i < nodos; i++) padre[i] = i;
  dis[inicio] = 0;
  int aux;

  for (int i = 0; i < nodos; i++)
    for (int j = 0; j < lista.size(); j++) {
      aux = dis[lista[j].second.first] + lista[j].first;
      if (dis[lista[j].second.second] > aux){
        dis[lista[j].second.second] = aux;
        padre[lista[j].second.second] = lista[j].second.first;
      }
    }

    for(int j = 0; j < nodos; j++){
      aux = dis[lista[j].second.first] + lista[j].first;
      if(dis[lista[j].second.second] > aux)
                return false;//existe ciclo!!!
    }
  return true;
}
```

2.3. Floyd Warshall

Ruta minima de toda la matriz, recomendable si n $\leq$ 100
O(n^3)

```
#define inf 1000

using namespace std;

vector<vector<int>> matriz(10, vector<int>(10, inf));

void FloydWarshall(vector<vector<int>> &grafo, int nodos){
    int aux;
    for(int i = 0; i < nodos; i++) grafo[i][i] = 0;

    for(int k = 0; k < nodos; k++)
        for(int i = 0; i < nodos; i++)
            for(int j = 0; j < nodos; j++){
                aux = grafo[i][k] + grafo[k][j];
                if(grafo[i][j] > aux) grafo[i][j] = aux;
            }
```

2.4. Kosaraju

Componentes fuertemente conexas grafos si y no dirigidos
O(2(n + m))

```
typedef vector<int> vi;

vector<vi> grafo(5), transpuesto(5), comp;
stack<int> pila;
bool vis[5];

void dfs(int n, vector<vi> lista, bool f, vi &grupo){
  vis[n] = true;
  if(!f) grupo.push_back(n);

  for (int i = 0; i < lista[n].size(); i++)
    if (!vis[lista[n][i]]) dfs(lista[n][i], lista, f, grupo);

    if(f) pila.push(n);
}

void kosaraju(){
  memset(vis, false, sizeof(vis));
  vi no_se_utiliza;

  for (int i = 0; i < 5; i++)
    if(!vis[i]) dfs(i, grafo, true, no_se_utiliza);

  memset(vis, false, sizeof(vis));
  int n;
  while(pila.size()){
    n = pila.top();
    pila.pop();
    if (!vis[n]){
      vi vec;
      dfs(n, transpuesto, false, vec);
      comp.push_back(vec);
    }
  }

  for (int i = 0; i < comp.size(); i++){
    for (int j = 0; j < comp[i].size(); j++) cout << comp[i][j] << " ";
    cout << endl;
  }
```

2.5. Kruskal

Arbol generador minimo, se necesita de un union-find
O(m log n), sin contar el ordenamiento.

```

typedef pair<int, int> ii;
typedef pair<int, ii> piii;//peso, origen y destino
#define mpiii(a, b, c) piii(a, ii(b, c))
//ejemplo de insertar:
//grafo.push_back(mpiii(7, 0, 1))

vector<piii> grafo;//lista de incidencia
union_find arbol;

int kruskal(vector<piii> lista, int nodos, union_find &uf){
  sort(lista.begin(), lista.end());
  uf.iniciar(nodos);
  int acum = 0, ejes = 0, n = nodos - 1;

  for (int i = 0; i < lista.size(); i++)
  {
    if (!uf.MismoGrupo(lista[i].second.first, lista[i].second.second))
    {
      ejes++;
      uf.unir(lista[i].second.first, lista[i].second.second);
      acum += lista[i].first;
      if(ejes == n) return acum;
    }
```

2.6. Topological sort

O(m + n)

```
vector<int> res;//guarda la respuesta.
vector<int> ent;//se debe llenar con la cantidad de
                //aristas entrantes que tiene cada nodo.

```

```cpp
void topological_sort(vvi &lis, int tam){
    res.clear();
    queue<int> s;
    for(int i = 1; i <= tam; i++){
        if(!ent[i]) s.push(i);
    }

    int n, m;
    while(s.size()){
        n = s.front();
        s.pop();
        res.push_back(n);

        for(int i = 0; i < lis[n].size(); i++){
            m = lis[n][i];
            ent[m]--;
            if(!ent[m]) s.push(m);
        }
    }
}
```

3. Programacion dinamica

3.1. Subconjuntos de un conjunto

$O(2^n)$

```cpp
void mask(int n, int ar[]){
    int l = 1 << n;

    for(int i = 0; i < l; i++){
        for(int j = 0; j < n; j++){
            if(i & (1 << j)){
                printf("%d␣", ar[j]);
            }
        }
        printf("\n");
    }
}
```

3.2. Problema de la mochila

```cpp
vi ben;//beneficio
vi cos;//costo
int knapsack(int cap, vi &cos, vi &ben, int n) {
    int dp[n+1][cap+1];

    for(int i = 0; i <= n; i++){
        for(int j = 0; j <= cap; j++){
            if(i == 0 || j == 0) dp[i][j] = 0;
            else if(cos[i - 1] <= j)
                dp[i][j] = max(ben[i - 1] + dp[i - 1][j - cos[i
                    - 1]], dp[i - 1][j]);
            else
                dp[i][j] = dp[i - 1][j];
        }
    }
    return dp[n][cap];
}
```

3.3. Longest Increment Subsecuence

Subsecuencia creciente mas larga
O(n log n)

```cpp
int LIS(){
    int res = 0;
    vector<int> vec(8, 1);

    for(int i = 0; i < 8; i++){
        for(int j = i + 1; j < 8; j++)
            if(arr[i] < arr[j]) vec[j] = max(vec[j], vec[i] +
                1);
        res = max(res, vec[i]);
    }

    return res;
}
```

3.4. Max Range Sum

O(n)

```
int main(){
    int n, num, res, aux;

    while(scanf("%d", &n), n){
        res = aux = 0;
        for(int i = 0; i < n; i++){
            scanf("%d", &num);
            aux += num;
            res = max(aux, res);
            if(aux < 0) aux = 0;
        }

        if(res > 0) printf("MRS = %d\n", res);
        else printf("negativo.\n");
    }
    return 0;
}
```

3.5. Subset Sum

```
bool dp[5][50];//fila cantidad de numeros
//columas rango maximo a evaluar

void pre(vi &num){
    memset(dp, false, sizeof(dp));

    for(int i = 0; i < num.size(); i++){
        if(i) for(int j = 1; j < 50; j++)
            if(dp[i - 1][j]) dp[i][j + num[i]] = true;

        dp[i][num[i]] = true;
    }
}
```

4. Otros

4.1. Busqueda binaria

O(log n)

```
int f(int a, int b){
    return ar[a] > b;
}

int busqueda_binaria(int men, int may, int v){
    int epsilon = 1, med = 0;

    while(may-men > epsilon){
        med = (may+men)/2;
        if(f(med,v))
            may = med;
        else
            men = med;
    }
    return men;
}
```

4.2. Raiz babilonica

Encuentra la raiz cuadrada de un numero

```
double raiz(double x) {
    double b = x, h = 0, apro = 1;
    while (apro > 1e-8) {
        b = (h + b) / 2;
        h = x / b;
        apro = abs(h - b);
    }
    return b;
}
```

5. Matematicas

5.1. MCD y MCM

Maximo comun divisor(MCD) y minimo comun multiplo(MCM)

```
int mcd(int a, int b){//algoritmo de euclides
    return  a? mcd(b %a, a): b;
}

int mcm(int a, int b) {
  return a*b/mcd(a,b);
}
```

5.2. Exponenciacion binaria

O(log n)

```
typedef long long int lli;

lli exp_bin (lli a, lli n) {
  lli res = 1;
  while (n) {
    if (n & 1)
      res *= a;
    a *= a;
    n >>= 1;
  }
  return res;
}
```

5.3. Algoritmo extendido de euclides

Encuentra dos numeros x e y tal que: MCD(a, b) = ax + by

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

int main(){
    int n, m, x, y, res;

    while(cin >> n >> m){
        res = gcd_ex(n, m, x, y);
        cout << "gcd = " << res << ", x = " << x << ", y = " <<
             y << endl;
    }
}
```

5.4. Inverso multiplicativo modular

Encuentra un x tal que (a * x) es congruente a 1 con modulo p, entonces:
(a * x) mod p = 1 mod p
necesita del algoritmo extendido de euclides
O(log m)

```
void inverso(int a, int m){
    int x, y;

    int g = gcd_ex (a, m, x, y);
    if (g != 1)
        cout << "no solution";
    else {
        x = (x % m + m) % m;
        cout << x << endl;
    }
}
```

5.5. Phi de euler

Devuelve la cantidad de coprimos de un numero n
O($\sqrt{n}$)

```
int phi (int n) {
  int result = n;
  for (int i=2; i*i<=n; ++i)
    if (n % i == 0) {
      while (n % i == 0)
        n /= i;
      result -= result / i;
    }
  if (n > 1)
    result -= result / n;
  return result;
}
```

5.6. Multiplicacion modular

Encuentra (a*b) mod c, la operacion puede generar overflow si se realiza directamente, el metodo mulmod evita el overflow usando un ciclo, pero se puede usar el tipo de dato int128 de c++11 para poder calcular de manera directa, pero el int128 no se puede leer o imprimir directamente.

```
typedef long long int lli;//metodo normal
lli mulmod (lli a, lli b, lli c) {
  lli x = 0, y = a%c;
  while (b > 0){
    if (b % 2 == 1) x = (x+y) % c;
    y = (y*2) % c;
    b /= 2;
  }
  return x % c;
}

typedef __int128 bi; //metodo con __int128
lli mulmod_2(bi a, bi b, bi c){
    return (lli) ((a*b) % c);
}

int main(){
    lli a, b, c;
    cin >> a >> b >> c;
    cout << mulmod_2((bi) a, (bi) b, (bi) c) << endl;
    return 0;
}
```

5.7. Exponenciacion modular

Encuentra (a^b) mod c, se nesecita implementar previamente multiplicacion modular.

```
ll expmod (ll b, ll e, ll m){//O(log b)
  if(!e) return 1;
  ll q = expmod(b,e/2,m); q = mulmod(q,q,m);
  return e%2? mulmod(b,q,m) : q;
}
```

5.8. Test de Rabin Miller

Devuelve si un numero es primo, requiere de implementar previamente MCD, multiplicacion modular y exponenciacion modular.

```
bool es_primo_prob (lli n, int a) {
  if (n == a) return true;
  lli s = 0,d = n-1;
  while (d % 2 == 0) s++,d/=2;

  lli x = expmod(a,d,n);
  if ((x == 1) || (x+1 == n)) return true;

  forn (i, s-1){
    x = mulmod(x, x, n);
    if (x == 1) return false;
    if (x+1 == n) return true;
  }
  return false;
}

bool rabin (lli n){ //devuelve true si n es primo
  if (n == 1) return false;
  const int ar[] = {2,3,5,7,11,13,17,19,23};
  forn (j,9)
    if (!es_primo_prob(n,ar[j]))
      return false;
  return true;
}
```

5.9. Rho de pollard

Factorizacion rapida, usar para $n > 10^{12}$, requiere de implementar previamente el MCD, multiplicacion modular, exponenciacion modular y el test de Rabin Miller.
O($\sqrt[4]{n}$)

```
lli rho(lli n){
    if( (n & 1) == 0 ) return 2;
    lli x = 2 , y = 2 , d = 1;
    lli c = rand() % n + 1;
    while( d == 1 ){
        x = (mulmod( x , x , n ) + c)%n;
        y = (mulmod( y , y , n ) + c)%n;
        y = (mulmod( y , y , n ) + c)%n;
        if( x - y >= 0 ) d = gcd( x - y , n );
        else d = gcd( y - x , n );
    }
    return d==n? rho(n):d;
}
```

```
map<lli, lli> prim;

void factRho (lli n){ //O (lg n)^3. un solo numero
  if (n == 1) return;
  if (rabin(n)){
    prim[n]++;
    return;
  }
  lli factor = rho(n);
  factRho(factor);
  factRho(n/factor);
}

int main(){
    lli n;
    while(scanf("%lld", &n), n > 0){
        prim.clear();
        factRho(n);

        for(map<lli, lli>::iterator it = prim.begin(); it != prim.end(); it++){
            cout << "el " << (it)->first << " aparece " << (it)->second << " veces.\n";
        }
    }
    return 0;
}
```

5.10. factorizacion con criba

Factorizacion usando la criba, mas corto de escribir que rho de pollard, usar para $n \leq 10^{1}2$, guarda los factores en un mapa similar a rho de pollard.

```
int m= 1000010, m2= 1000000, primo[1000020];
vector<lli> p;
map<lli, int> mapa;

void criba(){
    memset(primo, 0, sizeof(primo));

    for(int i = 2; i < m; i++){
        if(primo[i]) continue;
        p.push_back(i);
        primo[i] = i;
        if(i > 1000) continue;

        for(int j = i*i; j < m; j += i)
            primo[j] = i;
    }
}


void factCriba(lli n){
    int l;
    bool s;

    while(n != 1){
        if(n > m2){//n mayor a logintud del array
            l = sqrt(n) + 1;
            s = false;
            for(int i = 0; p[i] <= l; i++){
                if(n % p[i] == 0){
                    mapa[p[i]]++;
                    s = true;
                    n /= p[i];
                    break;
                }
            }
            if(!s){
                mapa[n]++;
                break;
            }
        }else{
            mapa[primo[n]]++;
            n /= primo[n];
        }
    }
}
```

5.11. BigInteger c++

```
typedef unsigned long long int ulli;
```

```
typedef long long int tdato; //no debe ser unsigned para la
    resta!!!
tdato base = 1000000000;

struct biginteger{
    vector<tdato> num;

    void iniciar(int n){
        num.clear();
        while(n){
            num.push_back((n >= base)? n % base: n);
            n /= base;
        }
    }

    void iniciar(string n){
        num.clear();
        for(int i = n.size(); i > 0; i -= 9){
            if(i < 9) num.push_back(atoi(n.substr(0, i).c_str()
                ));
            else num.push_back(atoi(n.substr(i-9, 9).c_str()));
        }
        quitar_zeros_izq();
    }

    void quitar_zeros_izq(){
        while(num.size() && !num.back()) num.pop_back();
    }

    void imprimir(){
        printf("%d", ((num.size())? num.back(): 0));
        for(int i = num.size() - 2; i >= 0; i--)
            printf("%09d", num[i]);
        printf("\n");
    }

    biginteger suma(biginteger b){
        ulli carry = 0, aux;
        int l = max(b.num.size(), num.size());
        biginteger c;

        for(int i = 0; i < l || carry; i++){
            aux = carry;
            if(i < b.num.size()) aux += b.num[i];
            if(i < num.size()) aux += num[i];

            if(aux >= base){
                c.num.push_back(aux % base);
                carry = aux / base;
            }else{
                c.num.push_back(aux);
                carry = 0;
            }
        }
        return c;
    }

    biginteger resta(biginteger b){//asumimos que b es menor
        tdato carry = 0;//no debe ser unsigned
        biginteger c;

        for(int i = 0; i < num.size(); i++){
            c.num.push_back(num[i]);
            c.num[i] -= ((i < b.num.size())? b.num[i]: 0) +
                carry;
            if(c.num[i] < 0){
                c.num[i] += base;
                carry = 1;
            }else carry = 0;
        }
        c.quitar_zeros_izq();
        return c;
    }

    biginteger multiplicar(biginteger b){
        ulli aux = 0, carry;
        biginteger c;
        c.num.assign(num.size() + b.num.size(), 0);

        for(int i = 0; i < num.size(); i++){
            carry = 0;
            for(int j = 0; j < b.num.size() || carry; j++){
```

```
                aux = c.num[i + j] + carry + (num[i] * ((j < b.num.size())? b.num[j] : 0));
                carry = aux / base;
                c.num[i + j] = aux % base;
            }
        }
        c.quitar_zeros_izq();
        return c;
    }

    int comparar(biginteger b){//este es: 1 mayor, 0 igual, -1 menor
        if(num.size() > b.num.size()){
            return 1;
        }else if(num.size() < b.num.size()){
            return -1;
        }else{
            for(int i = num.size() - 1; i >= 0; i--){
                if(num[i] > b.num[i]) return 1;
                else if(num[i] < b.num[i]) return -1;
            }
            return 0;
        }
    }
};
```

5.12. Fraccion

```
struct fraccion {
    int num, den;

    void iniciar(int x, int y) {
        num = x;
        den = y;
        fraccion c = simplificar();
        num = c.num;
        den = c.den;
    }

    fraccion sumar(fraccion b) {
        fraccion c;
        c.num = num * b.den + b.num * den;
        c.den = den * b.den;
        return c.simplificar();
    }

    fraccion restar(fraccion b) {
        fraccion c;
        c.num = num * b.den - b.num * den;
        c.den = den * b.den;
        return c.simplificar();
    }

    fraccion multiplicar(fraccion b) {
        fraccion c;
        c.num = num * b.num;
        c.den = den * b.den;
        return c.simplificar();
    }

    fraccion inversa() {
        fraccion c;
        c.iniciar(den, num);
        return c;
    }

    fraccion dividir(fraccion b) {
        return multiplicar(b.inversa()).simplificar();
    }

    int mcd(int a, int b){
        return  a? mcd(b %a, a): b;
    }

    fraccion simplificar() {
        fraccion c;
        c.num = num;
        c.den = den;
        if (c.den < 0) {
            c.num *= -1;
            c.den *= -1;
        }
        if (c.num == 0) {
```

```
            c.den = 1;
        } else {
            int dividir = mcd(c.num, c.den);
            c.num /= dividir;
            c.den /= dividir;
        }
        return c;
    }

    string toString() {
        stringstream ss;
        ss << num;
        if (den == 1) {
            return ss.str();
        }
        ss << "/";
        ss << den;
        return ss.str();
    }
};
```

6. Cadenas

6.1. Algoritmo de bordes

Encuentra la longitud del mayor borde de un string n.

```
int bordes[1000];

void algoritmoBordes(string subcad){
    int i = 1, j = -1;
    bordes[0] = -1;

    while(i < subcad.size()) {
        while(j >= 0 && subcad[i] != subcad[j])
            j = bordes[j];
        i++; j++;
        bordes[i] = j;
    }
}
```

6.2. KMP

Encuentra si una cadena n es subcadena de otra cadena m, requiere de implementar y ejecutar previamente el algoritmo de bordes
O(n+m)

```
void kmp(string cad, string subcad){
    int i = 0, j = 0;
    while(i < cad.size()){
        while(j >= 0 && cad[i] != subcad[j]) j = bordes[j];
        i++; j++;
        if(j == subcad.size()){
            printf("%s esta en el indice %d de la cadena: %s\n",
                subcad.c_str(), i - j, cad.c_str());
            j = bordes[j];
        }
    }
}
```

7. Tips and formulas(ufps)

7.1. ASCII Table

Caracteres ASCII con sus respectivos valores numéricos.

No.	ASCII	No.	ASCII
0	NUL	16	DLE
1	SOH	17	DC1
2	STX	18	DC2
3	ETX	19	DC3
4	EOT	20	DC4
5	ENQ	21	NAK
6	ACK	22	SYN
7	BEL	23	ETB
8	BS	24	CAN
9	TAB	25	EM
10	LF	26	SUB
11	VT	27	ESC
12	FF	28	FS
13	CR	29	GS
14	SO	30	RS
15	SI	31	US

No.	ASCII	No.	ASCII
32	(space)	48	0
33	!	49	1
34	”	50	2
35	#	51	3
36	$	52	4
37	%	53	5
38	&	54	6
39	’	55	7
40	(	56	8
41	)	57	9
42	*	58	:
43	+	59	;
44	,	60	¡
45	-	61	=
46	.	62	¿
47	/	63	?

No.	ASCII	No.	ASCII
64	@	80	P
65	A	81	Q
66	B	82	R
67	C	83	S
68	D	84	T
69	E	85	U
70	F	86	V
71	G	87	W
72	H	88	X
73	I	89	Y
74	J	90	Z
75	K	91	[
76	L	92	\
77	M	93	]
78	N	94	ˆ
79	O	95	_

No.	ASCII	No.	ASCII
96	‘	112	p
97	a	113	q
98	b	114	r
99	c	115	s
100	d	116	t
101	e	117	u
102	f	118	v
103	g	119	w
104	h	120	x
105	i	121	y
106	j	122	z
107	k	123	{
108	l	124	|
109	m	125	}
110	n	126	˜
111	o	127	

7.2. Formulas

PERMUTACIÓN Y COMBINACIÓN	
Combinación (Coeficiente Binomial)	Número de subconjuntos de k elementos escogidos de un conjunto con n elementos. $\binom{n}{k} = \binom{n}{n-k} = \frac{n!}{k!(n-k)!}$
Combinación con repetición	Número de grupos formados por n elementos, partiendo de m tipos de elementos. $CR_m^n = \binom{m+n-1}{n} = \frac{(m+n-1)!}{n!(m-1)!}$
Permutación	Número de formas de agrupar n elementos, donde importa el orden y sin repetir elementos $P_n = n!$
Permutación múltiple	Elegir r elementos de n posibles con repetición n^r
Permutación con repetición	Se tienen n elementos donde el primer elemento se repite a veces , el segundo b veces , el tercero c veces, ... $PR_n^{a,b,c\ldots} = \frac{P_n}{a!b!c!\ldots}$
Permutaciones sin repetición	Núumero de formas de agrupar r elementos de n disponibles, sin repetir elementos $\frac{n!}{(n-r)!}$
DISTANCIAS	

Continúa en la siguiente columna

Distancia Euclideana	$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
Distancia Manhattan	$d_M(P_1, P_2) = \|x_2 - x_1\| + \|y_2 - y_1\|$
CIRCUNFERENCIA Y CÍRCULO	
Considerando r como el radio, α como el ángulo del arco o sector, y (R, r) como radio mayor y menor respectivamente.	
Área	$A = \pi * r^2$
Longitud	$L = 2 * \pi * r$
Longitud de un arco	$L = \frac{2 * \pi * r * \alpha}{360}$
Área sector circular	$A = \frac{\pi * r^2 * \alpha}{360}$
Área corona circular	$A = \pi(R^2 - r^2)$
TRIÁNGULO	
Considerando b como la longitud de la base, h como la altura, letras minúsculas como la longitud de los lados, letras mayúsculas como los ángulos, y r como el radio de círcunferencias asociadas.	
Área conociendo base y altura	$A = \frac{1}{2} b * h$

Continúa en la siguiente columna

Área conociendo 2 lados y el ángulo que forman	$A = \frac{1}{2} b * a * sin(C)$
Área conociendo los 3 lados	$A = \sqrt{p(p-a)(p-b)(p-c)}$ con $p = \frac{a+b+c}{2}$
Área de un triángulo circunscrito a una circunferencia	$A = \frac{abc}{4r}$
Área de un triángulo inscrito a una circunferencia	$A = r(\frac{a+b+c}{2})$
Área de un triangulo equilátero	$A = \frac{\sqrt{3}}{4} a^2$

RAZONES TRIGONOMÉTRICAS

Considerando un triangulo rectángulo de lados a, b y c, con vértices A, B y C (cada vértice opuesto al lado cuya letra minuscula coincide con el) y un ángulo α con centro en el vertice A. a y b son catetos, c es la hipotenusa:

$sin(\alpha) = \frac{cateto\ opuesto}{hipotenusa} = \frac{a}{c}$

Continúa en la siguiente columna

$cos(\alpha) = \frac{cateto\ adyacente}{hipotenusa} = \frac{b}{c}$

$tan(\alpha) = \frac{cateto\ opuesto}{cateto\ adyacente} = \frac{a}{b}$

$sec(\alpha) = \frac{1}{cos(\alpha)} = \frac{c}{b}$

$csc(\alpha) = \frac{1}{sin(\alpha)} = \frac{c}{a}$

$cot(\alpha) = \frac{1}{tan(\alpha)} = \frac{b}{a}$

PROPIEDADES DEL MÓDULO (RESIDUO)

Propiedad neutro	(a % b) % b = a % b
Propiedad asociativa en multiplicación	(ab) % c = ((a % c)(b % c)) % c
Propiedad asociativa en suma	(a + b) % c = ((a % c) + (b % c)) % c

CONSTANTES

Pi	$\pi = acos(-1) \approx 3{,}14159$

Continúa en la siguiente columna

e	$e \approx 2{,}71828$
Número áureo	$\phi = \dfrac{1+\sqrt{5}}{2} \approx 1{,}61803$

7.3. Sequences

Listado de secuencias mas comunes y como hallarlas.

Estrellas octangulares	0, 1, 14, 51, 124, 245, 426, 679, 1016, 1449, 1990, 2651, ...
	$f(n) = n * (2 * n^2 - 1)$.
Euler totient	1, 1, 2, 2, 4, 2, 6, 4, 6, 4, 10, 4, 12, 6,...
	$f(n)$ = Cantidad de números naturales $\leq n$ coprimos con n.
Números de Bell	1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, ...
	Se inicia una matriz triangular con f[0][0] = f[1][0] = 1. La suma de estos dos se guarda en f[1][1] y se traslada a f[2][0]. Ahora se suman f[1][0] con f[2][0] y se guarda en f[2][1]. Luego se suman f[1][1] con f[2][1] y se guarda en f[2][2] trasladandose a f[3][0] y así sucesivamente. Los valores de la primera columna contienen la respuesta.
Números de Catalán	1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, ...
	$f(n) = \dfrac{(2n)!}{(n+1)!n!}$
Números de Fermat	3, 5, 17, 257, 65537, 4294967297, 18446744073709551617, ...
	$f(n) = 2^{(2^n)} + 1$
Números de Fibonacci	0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...
	$f(0) = 0$; $f(1) = 1$; $f(n) = f(n-1) + f(n-2)$ para $n > 1$

Continúa en la siguiente columna

Números de Lucas	2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, ...
	$f(0) = 2$; $f(1) = 1$; $f(n) = f(n-1) + f(n-2)$ para $n > 1$
Números de Pell	0, 1, 2, 5, 12, 29, 70, 169, 408, 985, 2378, 5741, 13860, ...
	$f(0) = 0; f(1) = 1; f(n) = 2f(n-1) + f(n-2)$ para $n > 1$
Números de Tribonacci	0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, 274, 504, ...
	$f(0) = f(1) = 0; f(2) = 1; f(n) = f(n-1) + f(n-2) + f(n-3)$ para $n > 2$
Números factoriales	1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, ...
	$f(0) = 1; f(n) = \prod_{k=1}^{n} k$ para $n > 0$.
Números piramidales cuadrados	0, 1, 5, 14, 30, 55, 91, 140, 204, 285, 385, 506, 650, ...
	$f(n) = \dfrac{n*(n+1)*(2*n+1)}{6}$
Números primos de Mersenne	3, 7, 31, 127, 8191, 131071, 524287, 2147483647, ...
	$f(n) = 2^{p(n)} - 1$ donde p representa valores primos iniciando en $p(0) = 2$.
Números tetraedrales	0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, ...
	$f(n) = \dfrac{n*(n+1)*(n+2)}{6}$
Números triangulares	0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, ...
	$f(n) = \dfrac{n(n+1)}{2}$

Continúa en la siguiente columna

OEIS A000127	1, 2, 4, 8, 16, 31, 57, 99, 163, 256, 386, 562, ...	
	$f(n) = \dfrac{(n^4 - 6n^3 + 23n^2 - 18n + 24)}{24}$.	
Secuencia de Narayana	1, 1, 1, 2, 3, 4, 6, 9, 13, 19, 28, 41, 60, 88, 129, ...	
	$f(0) = f(1) = f(2) = 1; f(n) = f(n-1) + f(n-3)$ para todo $n > 2$.	
Secuencia de Silvestre	2, 3, 7, 43, 1807, 3263443, 10650056950807, ...	
	$f(0) = 2; f(n+1) = f(n)^2 - f(n) + 1$	
Secuencia de vendedor perezoso	1, 2, 4, 7, 11, 16, 22, 29, 37, 46, 56, 67, 79, 92, 106, ...	
	Equivale al triangular(n) + 1. Máxima número de piezas que se pueden formar al hacer n cortes a un disco. $f(n) = \dfrac{n(n+1)}{2} + 1$	
Suma de los divisores de un número	1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28, 14, 24, ...	
	Para todo $n > 1$ cuya descomposición en factores primos es $n = p_1^{a_1} p_2^{a_2} ... p_k^{a_k}$ se tiene que: $f(n) = \dfrac{p_1^{a_1+1} - 1}{p_1 - 1} * \dfrac{p_2^{a_2+1} - 1}{p_2 - 1} * ... * \dfrac{p_k^{a_k+1} - 1}{p_k - 1}$	

7.4. Time Complexities

Aproximación del mayor número n de datos que pueden procesarse para cada una de las complejidades algoritmicas. Tomar esta tabla solo como referencia.

Complexity	**n**
$O(n!)$	11
$O(n^5)$	50
$O(2^n * n^2)$	18
$O(2^n * n)$	22
$O(n^4)$	100
$O(n^3)$	500
$O(n^2 \log_2 n)$	1.000
$O(n^2)$	10.000
$O(n \log_2 n)$	10^6
$O(n)$	10^8
$O(\sqrt{n})$	10^{16}
$O(\log_2 n)$	-
$O(1)$	-

8. Extras

8.1. Formulas extra

formula de triangulos degenerados:

$$\frac{(a+b-c)*(a+c-b)*(b+c-a)}{a*b*c}$$

Si el resultado es mayor que 0.5, es posible formar el triangulo.

Ecuacion de la recta que pasa por dos puntos:

$$\frac{x-x_1}{x_2-x_1} = \frac{y-y_1}{y_2-y_1}$$

Distancia de un punto a una recta:
Teniendo una recta con formula de la forma: $ax+by+c$ la distancia minima a un punto p de la forma (x, y) la distancia minima esta dada por la formula:

$$d = \frac{ax+by+c}{\sqrt{a^2+b^2}}$$

Formula de numeros fibonacci:

$$f(n) = \frac{1}{\sqrt{5}} * [(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n]$$

Determinante de Gauss:
Encontrar el area de un poligono en el plano cartesiano a partir de sus vertices

$$A = \frac{1}{2}\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_1 & y_1 \end{bmatrix}$$

$$S = x_1y_2 + x_2y_3 + ... + x_ny_1$$

$$D = x_2y_1 + x_3y_2 + ... + x_1y_n$$

$$A = \frac{1}{2}|S - D|$$

8.2. Secuencias

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

Fibonacci:

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418 317811 514229 832040 1346269 2178309 3524578 5702887 9227465 14930352 24157817 39088169 63245986 102334155 165580141 267914296 433494437 701408733 1134903170 1836311903

Factoriales:

1 2 6 24 120 720 5040 40320 362880 3628800 39916800 479001600 6227020800 87178291200 1307674368000 20922789888000 355687428096000 6402373705728000 121645100408832000

Potencias de dos: de 1 hasta 63

1 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768 65536 131072 262144 524288 1048576 2097152 4194304 8388608 16777216 33554432 67108864 134217728 268435456 536870912 1073741824 2147483648 4294967296 8589934592 17179869184 34359738368 68719476736 137438953472 274877906944 549755813888 1099511627776 2199023255552 4398046511104 8796093022208 17592186044416 35184372088832 70368744177664 140737488355328 281474976710656 562949953421312 1125899906842624 2251799813685248 4503599627370496 9007199254740992 18014398509481984 36028797018963968 72057594037927936 144115188075855872 288230376151711744 576460752303423488 1152921504606846976 2305843009213693952 4611686018427387904 9223372036854775808