Matemática

Brian Morris Esquivel - UNR

Training Camp 2022

Introducción

> Álgebra

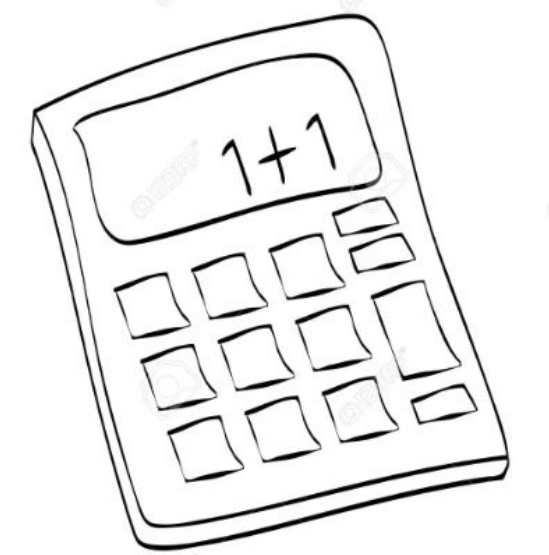
> Geometría

> Combinatoria

> Teoría de Números

Álgebra

Conceptos de Álgebra



- Entero vs. Punto flotante
  - Cálculo de raíz cuadrada
  - Cálculo de potencia
  - Multiplicar en lugar de dividir
- Desglose de fórmulas gigantes

Entero vs. Punto Flotante

Un **número entero** puede representarse de manera exacta por una computadora.

Un número racional también, ya que se puede representar como la razón entre un par de enteros.

Un **número real** en cambio, no siempre se puede representar exactamente.
(¿Cómo representarías pi?)

Variables enteras

int - hasta 2*10^9 como máximo
long long - hasta 4*10^18 como máximo

Variables flotantes

float - 6 dígitos de precisión
double - 15 dígitos de precisión
long double - muchos dígitos

Cálculo de raíz cuadrada

Cálculo normal con flotantes

Se usa la función **sqrt()**.
Alternativa **sqrtl()** si se trabaja con long double.

```
double x;
double raiz = sqrt(x);

long double y;
long double raiz = sqrtl(y);
```

Alternativa con enteros

Búsqueda binaria para encontrar el piso de la raíz.

```
int raiz(int x) {
  int l = 0, r = x+1;
  while (R - L > 1) {
    int m = (l + r) / 2;
    if (m*m <= x) l = m;
    else r = m;
  }
  return l;
}
```

Cálculo de potencia

Cálculo normal con flotantes

Se usa la función **pow()**.
Alternativa **powl()** si se trabaja con long double.

```
double x, exp;
double pot = pow(x, exp);

long double y, exp;
long double pot = powl(y, exp);
```

Alternativa con enteros

Multiplicación secuencial
O exponenciación binaria (lo veremos luego)

```
int exp(int b, int e) {
  int val = 1;
  for (int i = 0; i < e; i++)
    val *= b;
  return val;
}
```

Multiplicar en lugar de dividir

Cálculo normal con flotantes

Solemos consultar igualdad entre puntos flotantes que son en realidad números racionales.

```
int a, b, c, d;
double val1 = a/b;
double val2 = c/d;

if (val1 == val2) {
  // do stuff…
}
```

Alternativa con enteros

Si podemos expresar una fórmula con enteros, lo hacemos.

```
int a, b, c, d;

if (a*d == c*b) {
  // do stuff…
}
```

Desglose de fórmulas gigantes

Alguna vez habrás querido implementar algo así:

```
int a, b, c, d;
scanf(“%d %d %d %d”, &a, &b, &c, &d);
printf(“%d\n”, (a*(b+1) - c*d)/(a*b + c*d));
```

Lo cual no está mal, pero muchas veces es mejor separar en partes:

```
int a, b, c, d;
scanf(“%d %d %d %d”, &a, &b, &c, &d);
int bigVal = a*(b+1), smallVal = a*b, offset = c*d;
int num = bigVal - offset, den = smallVal + offset;
printf(“%d\n”, num/den);
```

Combinatoria

Conceptos de Combinatoria

- Conteo
  - Principio de multiplicación
  - Permutaciones
  - Combinaciones
- Otros temas

Principio de multiplicación

Si Brian tiene 5 remeras y 3 pantalones, entonces tiene 15 formas distintas de elegir una remera y un pantalón.

Permutaciones

¿Cuántas formas hay de ordenar n elementos?

Supongamos que Juancito tiene 3 pares de medias que quiere ordenar.

Si Juancito tiene 3 pares de medias, el total de patrones distintos que se pueden ver es 6.

La cantidad de formas de ordenar n elementos es:

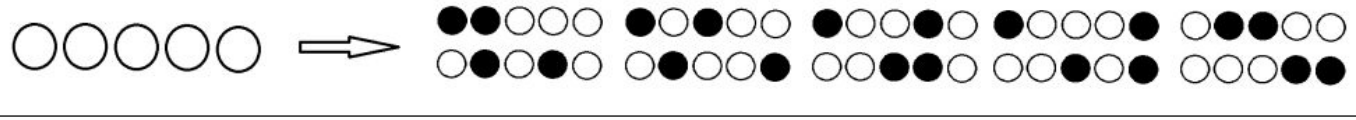
$$Permutaciones(n) = n!$$

Combinaciones

¿Cuántas formas hay de elegir k elementos entre n posibles?

Supongamos que Rogelio apuesta en una quiniela en la que tiene que acertar 2 números entre 5 posibles.

Si fuera que $n = 5$ y $k = 2$, existen exactamente 10 posibles resultados del sorteo.

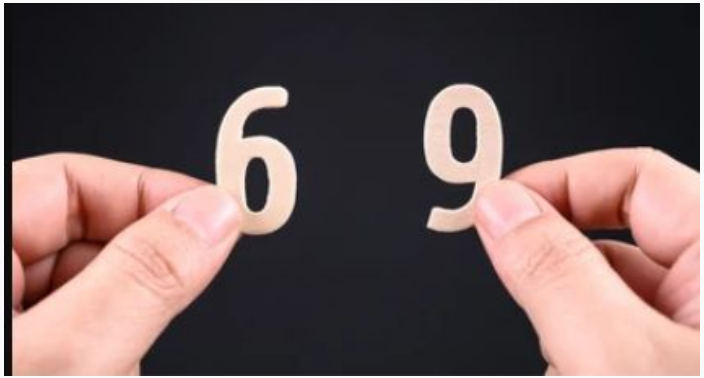


La cantidad de formas de ordenar n elementos es:

$$\boldsymbol{Combinaciones(n,k) = nCk = \binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}}$$

Teoría de Números

Conceptos de Teoría de Números



- Divisibilidad
  - Números primos
  - Criba de eratóstenes
  - MCD y MCM
- Algoritmos útiles
  - Divisores
  - Factorización
- Aritmética Modular

Números primos

Un número primo es aquel que tiene como únicos divisores el 1 y sí mismo.

Existen distintos algoritmos para determinar si un número es primo o no.

```
bool isPrime(int n) {
  for(int p = 2; p*p <= n; p++) {
    if(n%p == 0) return false;
  }
  return true;
}
```

Rabin-Miller (avanzado)

https://cp-algorithms.com/algebra/primality_tests.html#fermat-primality-test

Criba de eratóstenes

La criba de eratóstenes es una tabla que te indica si un número es primo o no.

```
bool isPrime[100500];
void criba() {
  isPrime[0] = isPrime[1] = false;
  for(int i = 2; i < 100500; i++) isPrime[i] = true;

  for(int p = 2; p < 100500; p++) {
    if (isPrime[p]) {
      for(int m = 2*p; m < 100500; m += p) isPrime[p] = false;
    }
  }
}
```

MCD y MCM

Para calcular el máximo común divisor entre dos números podemos usar el algoritmo de euclides.

```
int gcd(int a, int b) {
  if (b == 0) return a;
  return gcd(b, a%b);
}
// or simply use __gcd(a, b)
```

Para calcular el mínimo común múltiplo entre dos números usamos el MCD

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

Divisores de un número

```
vector<int> Div;
void getDiv(int n) {
  Div.clear();
  for (int d = 1; d * d <= n; d++) {
    if (n%d == 0) {
      Div.push_back(d);
      Div.push_back(n/d);
    }
    if (d*d == n) Div.pop_back();
  }
  sort(D.begin(), D.end());
}
```

Factorización de un número

```
map<int, int> F;
void fact(int n) {
  F.clear();
  for (int p = 2; p * p <= n; p++) {
    while (n%p == 0) {
      F[p]++;
      n /= p;
    }
  }
  if (n > 1) F[n]++;
}
```

¿Qué es?

La aritmética modular es una rama de la matemática que estudia ecuaciones del tipo:

$$a \equiv b(m)$$

Que significa:
"Los enteros a y b tienen el mismo resto en la división por m"

o bien:
"La diferencia $a - b$ es múltiplo de m"

Implementación

Introducción

No es suficiente entender las ecuaciones matemáticas para resolver problemas, también necesitamos saber implementarlos de forma eficiente.

En las soluciones que acabamos de ver encontramos factoriales, los cuales no suelen poder computarse velozmente.

Las respuestas a problemas de conteo suelen tener valores gigantescos.
Se suele pedir que se de la respuesta "módulo cierto primo p". Se necesita saber hacer cálculos en álgebra modular para estos casos.

Aritmética Modular

Para hacer los cálculos es necesario tener claros algunos conceptos básicos de aritmética modular.

No vamos a desarrollar ese tema en esta clase, pero sí voy a enfatizar algunas funciones que son clave para la implementación de soluciones.

Lo primero y principal que tiene que quedar claro es la siguiente relación:

$$\frac{P}{Q} = S \in Z \quad \Rightarrow \quad S(\text{mod } M) = P \cdot Q^{-1}(\text{mod } M)$$

Exponenciación Binaria Modular

Con el método de exponenciación binaria podés calcular potencias enteras de un número rápidamente.

El chiste del método es aprovechar el cálculo de $a^k \pmod M$ para calcular $a^{2k} \pmod M$ utilizando una única operación.

```
const int M = 1e9+7;
typedef long long ll;
ll expmod(ll b, ll e){
    ll ans = 1;
    while(e){
        if(e&1) ans = ans*b %M;
        b = b*b %M; e >>= 1;
    }
    return ans;
}
```

Inverso Modular

Hay muchas formas rápidas (complejidad logarítmica) de calcular el inverso modular de un entero a módulo otro entero M.

Algunos métodos involucran teoremas como: Pequeño Teorema de Fermat, Teorema de Fermat-Euler, Método de Euclides Extendido, etc.

El Pequeño Teorema de Fermat dice que sea p primo y a coprimo a p:

$$a^{p-2} \equiv a^{-1} \pmod{p}$$

```
ll invmod(ll a){ return expmod(a, M-2); }
```

Inverso Modular (cont.)

Además del cálculo del inverso modular en tiempo logarítmico existe otra práctica interesante.

Cuando el módulo M es un número primo se satisface la siguiente fórmula:

$$a^{-1} (\mathrm{mod}\ M) = -\left\lfloor \frac{M}{a} \right\rfloor \cdot \left(M (\mathrm{mod}\ a) \right)^{-1} (\mathrm{mod}\ M)$$

Podemos precalcular los inversos modulares de M para valores de $n \leq 10^6$.

```
typedef long long ll;
const int MAXN = 1e7, M = 1e9+7;
int INV[MAXN];
// ...
INV[1] = 1;
for(ll a = 2; a < MAXN; a++) INV[a] = M - (ll)(M/a)*INV[M%a]%M;
```

Cómputo de factoriales

Calcular factoriales es una tarea que no se puede hacer rápido.

Los problemas que requieren el cálculo de factoriales $n!$ suelen tener cotas en el orden de 10^6 para la variable n.

Lo que se hace para calcular factoriales cuando $n \leq 10^6$ es **precalcular** los factoriales y guardarlos en un arreglo.

```
ll F[MAXN];
// ...
F[0] = 1;
for(ll i = 1; i < MAXN; i++) F[i] = F[i-1]*i %M;
```

Cómputo de factoriales inversos

Le llamamos "factorial inverso" de n módulo M, al inverso modular del *factorial de* n módulo M.

Se pueden calcular los factoriales inversos usando la función *invmod*.

Pero cuando $n \leq 10^6$, es más eficiente tener en cuenta que:

$$(n!)^{-1} \pmod M = (1 \cdot 2 \cdot \cdots \cdot n)^{-1} \pmod M = 1^{-1} \cdot 2^{-1} \cdot \cdots \cdot n^{-1} \pmod M$$

Tras lo cual podemos precalcular los factoriales inversos en tiempo lineal.

```
ll F[MAXN], INV[MAXN], FI[MAXN];
// ...
F[0] = 1; forr(i, 1, MAXN) F[i] = F[i-1]*i %M;
INV[1] = 1; forr(i, 2, MAXN) INV[a] = M - (ll)(M/a)*INV[M%a]%M;
FI[0] = 1; forr(i, 1, MAXN) FI[i] = FI[i-1]*INV[i] %M;
```

Combinaciones – A partir de factoriales

Cuando ya sabés precomputar factoriales y factoriales inversos es sencillo calcular coeficientes binomiales para valores de n y k menores que 10^7.

Por comodidad y prolijidad se suele definir una función que la calcule.

```
ll Comb(ll n, ll k){
    if(n < k) return 0;
    return F[n]*FI[k] %M *FI[n-k] %M;
}
```

Combinaciones - Precálculo

Una propiedad muy útil de coeficientes binomiales es la siguiente:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

(Nótese que los coeficientes binomiales conforman un triángulo de Pascal)

Y por definición: $\binom{n}{0} = \binom{n}{n} = \frac{n!}{0! \cdot n!} = 1$

Precalculamos una tabla de coeficientes binomiales cuando $n \cdot k \leq 10^7$.

```
ll C[MAXN][MAXK];
// ...
forn(i, MAXN){
     C[i][0] = 1; if(i < MAXK) C[i][i] = 1;
    forr(j, 1, min(i, MAXK))
        C[i][j] = (C[i-1][j-1] + C[i-1][j])%M;
}
```

Combinaciones – Precálculo (cont.)

Aunque los valores de n y k no tengan cotas individuales adecuadas, en algunas ocasiones sucede que el valor de $n \cdot k$ está acotado.

Si existe una cota $n \cdot k \leq 10^7$ también podés crear la tabla de coeficientes binomiales para cada entrada n, k.

```
vector<vector<ll>> C;
// ...
scanf("%d %d", &N, &K); C.resize(N, vector<ll>(K));
forn(i, N){
    C[i][0] = 1; if(i < K) C[i][i] = 1;
    forr(j, 1, min(i, K))
        C[i][j] = (C[i-1][j-1] + C[i-1][j])%M;
}
```

Combinaciones – Resumen

Según las cotas que especifique el problema, en términos generales, conviene utilizar un método u otro para calcular coeficientes binomiales.

Cuando $MAXN \cdot MAXK \leq 10^7$:	Tabla precalculada.
Cuando $n \cdot k \leq 10^7$:	Tabla calculada en cada input.
Cuando $n, k \leq 10^7$:	A partir de factoriales.
Cuando $n, k \leq 10^{18}, M \leq 3000$:	Algoritmo de Lucas.

¿Preguntas?

Fin
¡Gracias por venir!