ICPC Reference

Escuela Superior de Cómputo - IPN

Alberto Silva

Contents

section 0. Matematicas

subsection 0.0. Fundamentals

subsubsection 0.0.0. Exponenciación y multiplicación binaria

subsubsection 0.0.0. Mínimo común multiplo y máximo cómun divisor

subsubsection 0.0.0. Euclides extendido

subsection 0.0. Aritmetica modular

subsubsection 0.0.0. Inverso modular

subsubsection 0.0.0. Linear Congruence Equation

subsubsection 0.0.0. Factorial modulo p

subsubsection 0.0.0. Chinese Remainder Theorem

subsection 0.0. Cribas y primos

subsubsection 0.0.0. Criba de eratostenes

```
vector<int> Criba(int n) {
    int raiz = sqrt(n);
    vector<int> criba(n + 1);
    for (int i = 4; i <= n; i +=
    ↪  2)
        criba[i] = 2;
    for (int i = 3; i <= raiz; i
    ↪  += 2)
        if (!criba[i])
            for (int j = i * i; j
            ↪  <= n; j += i)
                if (!criba[j])
                ↪  criba[j] = i;
    return criba;
}
```

subsubsection 0.0.0. Criba de factor primo más pequeño

```
    vector<int> lowestPrime;
void lowestPrimeSieve(int n){
    lowestPrime.resize(n + 1, 1);
    lowestPrime[0] =
    ↪  lowestPrime[1] = 0;
    for(int i = 2; i <= n; ++i)
        lowestPrime[i] = (i & 1 ?
        ↪  i : 2);
    int limit = sqrt(n);
    for(int i = 3; i <= limit; i
    ↪  += 2)
        if(lowestPrime[i] == i)
            for(int j = i * i; j
            ↪  <= n; j += 2 * i)
            if(lowestPrime[j] ==
            ↪  j) lowestPrime[j]
            ↪  = i;
    }
```

subsubsection 0.0.0. Criba de la función φ de Euler

```
vector<int> Phi;
void phiSieve(int n){
    Phi.resize(n + 1);
    for(int i = 1; i <= n; ++i)
        Phi[i] = i;
    for(int i = 2; i <= n; ++i)
        if(Phi[i] == i)
            for(int j = i; j <=
            ↪  n; j += i)
                Phi[j] -= Phi[j]
                ↪  / i;
}
```

subsubsection 0.0.0. Criba de la función μ

```
vector<int> Mu;
void muSieve(int n){
    Mu.resize(n + 1, -1);
    Mu[0] = 0, Mu[1] = 1;
    for(int i = 2; i <= n; ++i)
        if(Mu[i])
            for(int j = 2*i; j <=
            ↪ n; j += i)
                Mu[j] -= Mu[i];
}
```

subsubsection 0.0.0. Triángulo de Pascal

```
vector<vector<lli>> Ncr;
void ncrSieve(lli n){
    Ncr.resize(n + 1);
    Ncr[0] = {1};
    for(lli i = 1; i <= n; ++i){
        Ncr[i].resize(i + 1);
        Ncr[i][0] = Ncr[i][i] =
        ↪ 1;
        for(lli j = 1; j <= i /
        ↪ 2; j++)
            Ncr[i][i - j] =
            ↪ Ncr[i][j] = Ncr[i
            ↪ - 1][j - 1] +
            ↪ Ncr[i +1][j];
        }
    }
```

subsubsection 0.0.0. Criba de primos lineal

```
const int N = 10000000;
int lp[N+1];
vector<int> primes;
void criba(){
    for (int i=2; i<=N; ++i) {
        if (lp[i] == 0) {
            lp[i] = i;
            primes.push_back (i);
        }
        for (int j=0;
        ↪ j<(int)primes.size()
        ↪ && primes[j]<=lp[i]
        ↪ && i*primes[j]<=N;
        ↪ ++j)
            lp[i * primes[j]] =
            ↪ primes[j];
    }
}
```

subsubsection 0.0.0. Block sieve

```
int count_primes(int n) {
    const int S = 10000;
    vector<int> primes;
    int nsqrt = sqrt(n);
    vector<char> is_prime(nsqrt +
    ↪ 1, true);
    for (int i = 2; i <= nsqrt;
    ↪ i++) {
        if (is_prime[i]) {
            primes.push_back(i);
            for (int j = i * i; j
            ↪ <= nsqrt; j += i)
                is_prime[j] =
                ↪ false;
        }
    }
    int result = 0;
    vector<char> block(S);
```

```
    for (int k = 0; k * S <= n;
    ↪ k++) {
        fill(block.begin(),
        ↪ block.end(), true);
        int start = k * S;
        for (int p : primes) {
            int start_idx =
            ↪ (start + p - 1) /
            ↪ p;
            int j =
            ↪ max(start_idx, p)
            ↪ * p - start;
            for (; j < S; j += p)
                block[j] = false;
        }
        if (k == 0)
            block[0] = block[1] =
            ↪ false;
        for (int i = 0; i < S &&
        ↪ start + i <= n; i++)
        ↪ {
            if (block[i])
                result++;
        }
    }
    return result;
}
```

subsubsection **0.0.0. Prime factors of $n!$**

if p is prime the highest power p^k of p that divides $n!$ is given by

$$k = \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \left\lfloor \frac{n}{p^3} \right\rfloor + \cdots$$

Cálculo de la depreciación

$$D = \frac{C - R}{N}$$

$$R = C * (1 - \frac{T}{100})^N$$

$$T = \frac{1}{N} * 100$$

D = depreciación anual
C = Costo del activo
R = Valor de rescate
N = Vida útil
T = % de depreciación anual

subsubsection **0.0.0. Primaly test(miller rabin)**

```
lli random(lli a, lli b) {
    lli intervalLength = b - a +
    ↪ 1;
    int neededSteps = 0;
    lli base = RAND_MAX + 1LL;
    while(intervalLength > 0){
      intervalLength /= base;
      neededSteps++;
    }
    intervalLength = b - a + 1;
    lli result = 0;
    for(int stepsDone = 0;
    ↪ stepsDone < neededSteps;
    ↪ stepsDone++){
        result = (result * base +
        ↪ rand());
    }
    result %= intervalLength;
    if(result < 0) result +=
    ↪ intervalLength;
```

```
    return result + a;
}

bool witness(lli a, lli n) {
    lli u = n-1;
    int t = 0;
    while (u % 2 == 0) {
        t++;
        u /= 2;
    }
    lli next = mod_pow(a, u, n);
    if(next == 1)return false;
    lli last;
    for(int i = 0; i < t; i++) {
      last = next;
        next = mod_mult(last,
        ↪  last, n);//(last *
        ↪  last) % n;
        if (next == 1){
          return last != n - 1;
        }
    }
    return next != 1;
}


bool isPrime(lli n, int s) {
    if (n <= 1) return false;
    if (n == 2) return true;
    if (n % 2 == 0) return false;
    for(int i = 0; i < s; i++) {
        lli a = random(1, n-1);
        if (witness(a, n)) return
        ↪  false;
    }
    return true;
}
```

subsubsection **0.0.0. Factorización varios metodos**

```
map<lli,lli> fact;
void trial_division4(lli n) {
    for (lli d : primes) {
        if (d * d > n)
            break;
        while (n % d == 0) {
            fact[d]++;
            n /= d;
        }
    }
}
void trial_division2(lli n) {
    while (n % 2 == 0) {
        fact[2]++;
        n /= 2;
    }
    for (long long d = 3; d * d
    ↪  <= n; d += 2) {
        while (n % d == 0) {
            fact[d]++;
            n /= d;
        }
    }
    if (n > 1)
        fact[n]++;
}
/*
    Pollard Method p-1
*/
lli pollard_p_1(lli n){
  int b = 13;
  int q[] = {2, 3, 5, 7, 11, 13};
  lli a = 5% n;
  for (int j = 0; j <10; j ++){
    while (__gcd(a, n)!= 1){
      mod_mult (a, a, n);
      a+= 3;
```

```
      a%= n;
    }
    for (int i = 0; i<6; i ++){
      int qq = q [i];
      int e =  floor(log((double)
      ↪ b) / log((double) qq));
      lli aa = mod_pow(a, mod_pow
      ↪ (qq, e, n), n);
      if (aa == 0)
        continue;
        lli g = __gcd (aa-1, n);
      if (1 <g && g <n)
        return g;
    }

  }
  return 1;
}

/*
    Pollard rho
*/
lli pollard_rho (lli n, unsigned
↪ iterations_count = 100000){
  lli b0 = rand ()% n,b1 = b0,g;
  mod_mult (b1, b1, n);
  if (++b1 == n)
    b1 = 0;
  g = __gcd(abs(b1 - b0), n);
  for (unsigned count = 0; count
  ↪ <iterations_count && (g ==
  ↪ 1 || g == n); count ++){
    mod_mult (b0, b0, n);
    if (++ b0 == n)
      b0 = 0;
    mod_mult (b1, b1, n);
    ++ b1;
    mod_mult (b1, b1, n);
    if (++ b1 == n)
      b1 = 0;
    g = __gcd(abs(b1 - b0), n);
  }
  return g;
}
lli pollard_bent (lli n, unsigned
↪ iterations_count = 19){
  lli b0 = rand ()% n,
    b1 = (b0 * b0 + 2)% n,
    a = b1;
  for (unsigned iteration = 0,
  ↪ series_len = 1; iteration
  ↪ <iterations_count;
  ↪ iteration ++, series_len *=
  ↪ 2){
    lli g = __gcd(b1-b0, n);
    for (unsigned len = 0; len
    ↪ <series_len && (g == 1 &&
    ↪ g == n); len ++){
      b1 = (b1 * b1 + 2)% n;
      g = __gcd(abs (b1-b0), n);
    }
    b0 = a;
    a = b1;
    if (g != 1 && g != n)
      return g;
  }
  return 1;
}
/*
    Pollard monte Carlo
*/
lli pollard_monte_carlo (lli n,
↪ unsigned m = 100){
  lli b = rand ()% (m-2) + 2;
  lli g = 1;
  for (int i = 0; i <100 && g ==
  ↪ 1; i++){
    lli cur = primes[i];
    while (cur <= n)
```

```
      cur *= primes[i];
    cur /= primes[i];
    b = mod_pow (b, cur, n);
    g = __gcd(abs (b-1), n);
    if (g == n)
      g = 1;
  }
  return g;
}
lli prime_div_trivial (lli n){
  if (n == 2 || n == 3)
    return 1;
  if (n <2)
    return 0;
  if (!n&1)
    return 2;
  lli pi;
  for (auto p:primes){
    if (p*p >n)
      break;
    else
      if (n% p == 0)
        return p;
  }
  if (n <1000*10000)
    return 1;
  return 0;
}

lli ferma (lli n){
  lli  x =
  ↪  floor(sqrt((double)n)),y =
  ↪  0,r = x * x - y * y - n;
  for (;;)
    if (r == 0)
      return x != y? x*y: x + y;
    else
      if (r> 0){
        r-= y + y + 1;
        ++y;
      }
      else{
        r+= x + x + 1;
        ++x;
      }
}
lli mult(lli a, lli b, lli mod) {
    return (lli)a * b % mod;
}

lli f(lli x, lli c, lli mod) {
    return (mult(x, x, mod) + c)
    ↪  % mod;
}
lli brent(lli n, lli x0=2, lli
↪   c=1) {
    lli x = x0;
    lli g = 1;
    lli q = 1;
    lli xs, y;

    int m = 128;
    int l = 1;
    while (g == 1) {
        y = x;
        for (int i = 1; i < l;
        ↪   i++)
            x = f(x, c, n);
        int k = 0;
        while (k < l && g == 1) {
            xs = x;
            for (int i = 0; i < m
            ↪   && i < l - k;
            ↪   i++) {
                x = f(x, c, n);
                q = mult(q, abs(y
                ↪   - x), n);
            }
            g = __gcd(q, n);
```

```
            k += m;
        }
        l *= 2;
    }
    if (g == n) {
        do {
            xs = f(xs, c, n);
            g = __gcd(abs(xs -
            ↪ y), n);
        } while (g == 1);
    }
    return g;
}
```

subsubsection 0.0.0. Factorizacion usando todos los metodos

```
void factorize (lli n){
    if (isPrime(n,20))
        fact[n]++;
    else{
        if (n <1000 * 1000){
            lli div =
            ↪ prime_div_trivial(n);
            fact[div]++;
            factorize(n / div);
        }
        else{
            lli div;
            // Pollard's fast
            ↪ algorithms come
            ↪ first
            div =
            ↪ pollard_monte_carlo(n);
            if (div == 1)
                div = brent(n);
            if (div == 1)
                div = pollard_rho
                ↪ (n),cout<<"USE
                ↪ POLLAR_RHO\n";
            if (div == 1)
                div = pollard_p_1
                ↪ (n),cout<<"USE
                ↪ POLLARD_P_1\n";
            if (div == 1)
                div =
                ↪ pollard_bent
                ↪ (n),cout<<"USE
                ↪ POLLARD_BENT\n";
            if (div == 1)
                div = ferma (n);
            // recursively
            ↪ process the found
            ↪ factors
            factorize (div);
            factorize (n / div);
        }
    }
}
```

subsubsection 0.0.0. Numero de divisores hasta 1018

```
bool isSquare(lli val){
  lli lo = 1, hi = val;
  while(lo <= hi){
    lli mid = lo + (hi - lo) / 2;
    lli tmp = (val / mid) / mid;
    ↪ // be careful with
    ↪ overflows!!
    if(tmp == 0)hi = mid - 1;
    else if(mid * mid ==
    ↪ val)return true;
    else if(mid * mid < val)lo =
    ↪ mid + 1;
  }
```

```
  return false;
}
lli countDivisors(lli n) {
    lli ans = 1;
  for(int i = 0; i <
  ↪  primes.size(); i++){
    if(n == 1)break;
    int p = primes[i];
    if(n % p == 0){ // checks
    ↪  whether p is a divisor of
    ↪  n
      int num = 0;
      while(n % p == 0){
        n /= p;
        ++num;
      }
      // p^num divides initial n
      ↪  but p^(num + 1) does
      ↪  not divide initial val
      // => p can be taken 0 to
      ↪  num times => num + 1
      ↪  possibilities!!
      ans *= num + 1;
    }
  }
    if(n == 1)return ans; //
    ↪  first case
  else if(isPrime(n,20))return
  ↪  ans * 2; // second case
  else if(isSquare(n))return ans
  ↪  * 3; // third case but with
  ↪  p == q
  else return ans * 4; // third
  ↪  case with p != q
}
```

subsection 0.0. Funciones multiplicativas

subsubsection 0.0.0. Función φ de Euler

The most famous and important property of Euler's totient function is expressed in **Euler's theorem:**

$$\alpha^{\phi(m)} \equiv 1(mod \quad m)(1)$$

if α and $\boldsymbol{m}$ are relative prime.
In the particular case when m is prime, Euler's theorem turns into **Fermat's little theorem:**

$$\alpha^{m-1} \equiv 1(mod \quad m)(2)$$

$$\alpha^{n} \equiv \alpha^{n \mod \phi(m)} \quad (mod \quad m)(3)$$

This allows computing $x^n mod \quad m$ for very big n, especially if n is the result of another computation, as it allows to compute n under a modulo.

subsection **0.0. Linear Algebra**

subsubsection **0.0.0. Struct matrix**

subsubsection **0.0.0. Transpuesta**

subsubsection **0.0.0. Traza**

subsubsection **0.0.0. Gauss System of Linear Equationsn**

subsubsection **0.0.0. Gauss Determinant**

subsubsection **0.0.0. Cofactors Matrix**

subsubsection **0.0.0. Matriz inversa**

subsubsection **0.0.0. Adjoint Matrix**

subsubsection **0.0.0. Recurrencias lineales**

subsubsection **0.0.0. Kirchhoff Matrix Tree Theorem**

Count the number of spanning trees in a graph, as the determinant of the Laplacian matrix of the graph. **Laplacian Matrix** :

Given a simple graph G with n vertices, its Laplacian matrix $L_{n\times n}$ is defined as

$$L = D - A$$

The elements of L are given by

$$L_{i,j} = \begin{cases} deg(v_i) & \text{if } i == j \\ -1 & \text{if } i \neq j \text{and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

define $\tau(G)$ as number of spanning trees of a grap G

$$\tau(G) = \det L_{n-1\times n-1}$$

Where $L_{n-1\times n-1}$ is a laplacian matrix deleting any row and any column

$$\det \begin{pmatrix} deg(v_1) & L_{1,2} & \cdots & L_{1,n-1} \\ L_{2,1} & deg(v_2) & \cdots & L_{2,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ L_{n-1,1} & L_{n-1,2} & \cdots & deg(v_{n-1}) \end{pmatrix}$$

Generalization for a multigraph $K_n^m \pm G$
define $\tau(K_n^m \pm G)$ as number of spanning trees of a grap $K_n^m \pm G$

$$\tau(K_n^m \pm G) = n * (nm)^{n-p-2} \det(B)$$

where $B = mnI_p + \alpha * L(G)$is a $p \times p$ matrix, $\alpha = \pm$ according $(K_n^m \pm G)$, and $L(G)$ is the Kirchhoff matrix of G subsection **0.0. Metodos numericos**

subsubsection **0.0.0. FFT**

```
const double PI = acos(-1.0L);
using comp = complex<long
↪ double>;
using lli = long long int;
typedef vector<comp> vec;
#define print(A)  for(auto c : A)
↪ cout << c << " ";
#define isZero(z) abs(z.real()) <
↪ 1e-3;

int lesspow2(int n){
  int ans = 1;
  while(ans<n)ans<<=1;
  return ans;
}

void fft(vec& a, int inv){
  int n = a.size();
  for(int i = 1,j = 0;i<n-1;i++){
    for(int k = n>>1;(j^= k) <k;
    ↪ k>>= 1);
      if(i<j) swap(a[i],a[j]);
  }
  for(int k = 1;k<n;k<<=1){
    comp wk =
    ↪ polar(1.0,PI/k*inv);
    for(int i = 0;i<n;i+= k<<1){
      comp w(1);
      for(int j = 0;j<k;j++,w =
      ↪ w*wk){
        comp t = a[i+j+k]*w;
        a[i+j+k] = a[i+j]-t;
        a[i+j] += t;
      }
    }
  }
  if(inv == -1)
    for(int i = 0;i<n;i++)
      a[i]/=n;

}

void fft(vector<cd> &a,int
↪ invert){
  int n=a.size();
  for(int i=1,j=0;i<n;i++){
    int z=(n>>1);
    for(;(j&z);z=(z>>1)){
      j=(j^z);
    }
    j=(j^z);
    if(i<j)
    swap(a[i],a[j]);
  }
  for(int
  ↪ len=2;len<=n;len=(len<<1)){
    double
    ↪ ang=(2*PI/len)*((invert?-1:1));
    cd wlen(cos(ang),sin(ang));
    for(int i=0;i<n;i+=len){
      cd w(1);
      for(int j=0;j<len/2;j++){
        cd
        ↪ u=a[i+j],v=a[i+j+len/2]*w;
        a[i+j]=u+v;
        a[i+j+len/2]=u-v;
        w*=wlen;
      }
    }
  }
  if(invert){
    for(int i=0;i<n;i++){
      a[i]/=n;
    }
```

```
  }
}

vec multiply(vec &a,vec &b){
  int n0 = a.size()+b.size()-1;
  int n = lesspow2(n0);
  a.resize(n);
  b.resize(n);
  fft(a,1);
  fft(b,1);
  for(int i = 0;i<n;i++)
    a[i]*= b[i];
  fft(a,-1);
  a.resize(n0);
  return a;
}
```

subsection **0.0. Combinatorics**

subsubsection **0.0.0. Binomial coefficents**

```
  int i,j;
  long bc[MAXN][MAXN];
  for (i=0; i<=n; i++) bc[i][0] =
  ↪ 1;
  for (j=0; j<=n; j++) bc[j][j] =
  ↪ 1;
  for (i=1; i<=n; i++)
      for (j=1; j<i; j++)
          bc[i][j] = bc[i-1][j-1]
          ↪ + bc[i-1][j];
  return bc[n][m];
}

/*
    O(k) solution
    Only calc C(n,k)
*/
int binomial_Coeff_2(int n, int
↪ k)  {
    int res = 1;
    if ( k > n - k )
        k = n - k;
    for (int i = 0; i < k; ++i){
        res *= (n - i);
        res /= (i + 1);
    }
    return res;
}
/*
    O(k) solution
    Only calc C(n,k)
*/
int binomial_Coeff_3(int n, int
↪ k){
    vector<int> C(k+1,0);
    C[0] = 1;  // nC0 is 1
    for (int i = 1; i <= n; i++)
    ↪ {
        for (int j = min(i, k); j
        ↪ > 0; j--)
            C[j] = C[j] + C[j-1];
    }
    return C[k];
}
/****************************************************************
 *   Factorial modulo P
    If only need one factorial
    O(P logp n)
 *  Tested [?]
****************************************************************/
int factmod(int n, int p) {
    int res = 1;
    while (n > 1) {
        res = (res * ((n/p) % 2 ?
        ↪ p-1 : 1)) % p;
```

```
        for (int i = 2; i <= n%p;
        ↪ ++i)
            res = (res * i) % p;
        n /= p;
    }
    return res % p;
}


/*****************************************************************************
    Lucas Theorem
 *  Computes C(N,R)%p in
↪   O(log(n)) if P is prime
 *  Tested [Codeforces D - Sasha
↪   and Interesting Fact from
↪   Graph Theory]

*****************************************************************************
/*
    Precalc
        -Inverse modular to n
        -Factorial modulo p
        -Inverse modular of
↪   factorial
*/
const int mod = 1000000007;
const int MAXN =1000007;
lli inv[MAXN];
lli fact[MAXN];
lli invfact[MAXN];
void calc(int m){
    inv[0] = inv[1] = 1;
    fact[0] = fact[1] = 1;
    invfact[0] = invfact[1] = 1;
    for(int i = 2; i <= m; ++i) {
        inv[i] = (inv[mod % i]*
        ↪ (mod - (mod/i)))%mod;
        fact[i] = (fact[i - 1]*
        ↪ i)%mod;
        invfact[i] = (invfact[i -
        ↪ 1]* inv[i])%mod;
    }
}
/*
    Lucas Theorem
*/
lli Lucas(lli N,lli R){
  if(R<0||R>N)
    return 0;
  if(R==0||R==N)
    return 1ll;
  if(N>=mod)
    return
    ↪ (1ll*Lucas(N/mod,R/mod)*Lucas(N%mod,R%mod))%mod;
  return (1ll*
  ↪ fact[N]*((invfact[N-R]*invfact[R])%mod))%mod;
}
/*
    Using calc() we can also
↪   calculate P(n,k)
↪   (permutations)
*/
lli permutation(int n,int k){
    return (1ll*fact[n]*
    ↪ invfact[n-k])%mod;
}


/*****************************************************************************
 *  Cayleys formula
 *  Computes all posibles trees
↪   whit n nodes
 *  Tested [Codeforces D - Sasha
↪   and Interesting Fact from
↪   Graph Theory]
*****************************************************************************
lli cayley(int n ,int k){
    if(n-k-1<0)
```

```
        return
        ↪   (1ll*k*mod_pow(n,mod-2))%mod;
    return
    ↪   (1ll*k*mod_pow(n,n-k-1))%mod;
}

int main(){

    return 0;
```

Distribute N items in m container
HOLA N+m-1$\binom{}{N}$