ICPC Reference

Escuela Superior de Cómputo - IPN

Alberto Silva

Índice

1. Data Structures

1.1. AVL Tree

1.2. Kd tree

1.3. Quad tree

1.4. Binary Heap

1.5. Disjoint set union

1.6. Range Minimun Query

1.7. Sparse table

1.8. Fenwick tree (BIT)

1.9. Segment tree

1.10. Wavelet tree

1.11. Merge sort tree

1.12. Red black tree

1.13. Splay tree

1.14. Steiner tree

1.15. Treap

1.16. Heavy light decomposition

```python
# Note: π = lím_{n→∞} P_n/d
title = "Hello World"

sum = 0
for i in range(10):
  sum += i
  if sum >= 3:
  ->
  >>=
  print("HELLO")
```

2. Strings

2.1. Trie

2.2. Suffix array

2.3. Suffix Tree

2.4. Suffix automaton

2.5. Aho corasick

2.6. Z function

2.7. Knuth morris pratt

2.8. Palindromic tree

2.9. Manacher

2.10. Aritmetica modular

2.10.1. Inverso modular

```cpp
int inverse(int a, int m){
  int x, y ;
    if isPrime(m)return mod_pow(a,m-2,m);
  if(gcd( a, m, x, y ) != 1) return 0;
  return (x%m + m) % m;

*/
vector<lli> allinverse(lli p){
    vector<lli> ans(p);
    ans[1] = 1;
    for(lli i = 2;i<p:i++){
        ans[i] = p-(p/i)*ans[p%i]%p;
    }
    return ans;
}
```

2.10.2. Linear Congruence Equation

```cpp
bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &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;
}
```

2.10.3. Factorial modulo p

```cpp
vector<int>rem;
int CRT() {
    int prod = 1;
    for (int i = 0; i < nums.size(); i++)
        prod *= nums[i];
    int result = 0;
    for (int i = 0; i < nums.size(); i++) {
        int pp = prod / nums[i];
        result += rem[i] * inverse(pp, nums[i]) * pp;
    }
```

2.10.4. Chinese Remainder Theorem

```cpp
inline lli normalize(lli x, lli mod) { x %= mod; if (x < 0) x += mod; return x; }
vector<int> a;
vector<int> n;
lli LCM;
```

```
lli CRT(lli &ans){
    int t =a.size();
    ans = a[0];
    LCM = n[0];
    for(int i = 1; i < t; i++){
        int x1,d= gcd(LCM, n[i],x1,d);
        if((a[i] - ans) % d != 0) return 0;
        ans = normalize(ans + x1 * (a[i] - ans) / d % (n[i] / d) * LCM, LCM * n[i] / d);
        LCM = lcm(LCM, n[i]); // you can save time by replacing above LCM * n[i] /d by LCM = LCM * n[i] / d
    }
    return 1;
}
```

2.11. Cribas y primos

2.11.1. 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;
}
```

2.11.2. 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;
    }
```

2.11.3. 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;
}
```

2.11.4. 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];
}
```

2.11.5. 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];
```

```
        }
    }
```

2.11.6. 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];
    }
}
```

2.11.7. 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;
}
```

2.11.8. Prime factors of $n!$

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

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

2.11.9. Primaly test(miller rabin)

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

2.11.10. 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;
```

```cpp
}

/*
    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;
}
```

2.11.11. 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);
        }
    }
}
```

2.11.12. Numero de divisores hasta 10^{18}

```
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
}
```

2.12. Funciones multiplicativas

2.12.1. Función φ de Euler

```
//___________PHI de euler_____________//
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;
```

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

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

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:**

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

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

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.

2.13. Linear Algebra

2.13.1. Struct matrix

```cpp
typedef long long int lli;
template <typename T>
struct Matrix {
    vector < vector <T> > A;
    int r,c;
    Matrix(){
        this->r = 0;
        this->c = 0;
    }
    Matrix(int r,int c){
        this->r = r;
        this->c = c;
        A.assign(r , vector <T> (c));
    }

    Matrix(int r,int c,const T &val){
        this->r = r;
        this->c = c;
        A.assign(r , vector <T> (c , val));
    }
    Matrix(int n){
        this->r = this->c = n;
        A.assign(n , vector <T> (n));
        for(int i=0;i<n;i++)
            A[i][i] = (T)1;
    }
    Matrix operator * (const Matrix<T> &B){
//      Matrix <T> C(r,B.c,0);
//      for(int i=0 ; i<r ; i++)
//          for(int j=0 ; j<B.c ; j++)
//              for(int k=0 ; k<c ; k++)
//                  C[i][j] = (C[i][j] + ( (long long )A[i][k] * (long long)B[k][j] ));
//      return C;
        Matrix<T> C(r,B.c,0);
        for(int i = 0;i<r;i++){
            for(int j = 0;j<B.c;j++){
                for(int k = 0;k<c;k++){
                    C[i][j] = (C[i][j] + ((lli)A[i][k] * (lli)B[k][j]));
                    if(C[i][j] >= 8ll*mod*mod)
                        C[i][j]%=mod;
                }
            }
        }
        for(int i = 0;i<r;i++)for(int j = 0;j<c;j++)C[i][j]%=mod;
        return C;
    }

    Matrix operator + (const Matrix<T> &B){
        assert(r == B.r);
        assert(c == B.c);
        Matrix <T> C(r,c,0);
        int i,j;
        for(i=0;i<r;i++)
            for(j=0;j<c;j++)
                C[i][j] = ((A[i][j] + B[i][j]));
        return C;
    }
    Matrix operator*(int & c) {
        Matrix<T> C(r, c);
        for(int i = 0; i < r; i++)
            for(int j = 0; j < c; j++)
                C[i][j] = A[i][j] * c;
        return C;
    }
    Matrix operator - (){
        Matrix <T> C(r,c,0);
        int i,j;
        for(i=0;i<r;i++)
            for(j=0;j<c;j++)
                C[i][j] = -A[i][j];
        return C;
    }
    Matrix operator - (const Matrix<T> &B){
        assert(r == B.r);
        assert(c == B.c);
        Matrix <T> C(r,c,0);
        int i,j;
```

```
        for(i=0;i<r;i++)
            for(j=0;j<c;j++)
                C[i][j] = A[i][j] - B[i][j];
        return C;
    }
    Matrix operator ^ (long long n){
        assert(r == c);
        int i,j;
        Matrix <T> C(r);
        Matrix <T> X(r,c,0);
        for(i=0;i<r;i++)
            for(j=0;j<c;j++)
                X[i][j] = A[i][j];
        while(n){
            if(n&1)
                C *= X;
            X *= X;
            n /= 2;
        }
        return C;
    }
    vector<T>& operator [] (int i){
        assert(i < r);
        assert(i >= 0);
        return A[i];
    }

    const vector<T>& operator [] (int i) const{
        assert(i < r);
        assert(i >= 0);
        return A[i];
    }

    friend ostream& operator << (ostream &out,const Matrix<T> &M){
        for (int i = 0; i < M.r; ++i) {
            for (int j = 0; j < M.c; ++j) {
                out << M[i][j] << " ";
            }
            out << '\n';
        }
        return out;
```

```
    }

    void operator *= (const Matrix<T> &B){
        (*this) = (*this)*B;
    }

    void operator += (const Matrix<T> &B){
        (*this) = (*this)+B;
    }


        for (int i=0; i<m; ++i)
```

2.13.2. Transpuesta

```
        else {
            adjoint = this->cofactorMatrix().transpose();
        }
        return adjoint;
    }

    //Transpuesta
    Matrix transpose(){
        Matrix <T> C(c,r);
```

2.13.3. Traza

```
        for(i=0;i<r;i++)
            for(j=0;j<c;j++)
                C[j][i] = A[i][j];

        return C;
    }
```

2.13.4. Gauss System of Linear Equationsn

```
                if (j != i && abs (temp[j][i]) > EPS)
                    for (int k=i+1; k<n; ++k)
                        temp[j][k] -= temp[i][k] * temp[j][i];
        }
```

```cpp
    return (int)det;
}

int gauss (vector<double> & ans) {
    Matrix<double> Temp(this->r,this->c);
    int n = (int) Temp.A.size();
    int m = (int) Temp[0].size() - 1;
    for(int i = 0;i<n;i++)
        for(int j = 0;j<n;j++)
            Temp[i][j] = (double)A[i][j];

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (fabs (Temp[i][col]) > fabs (Temp[sel][col]))
                sel = i;
        if (fabs (Temp[sel][col]) < EPS)
            continue;
        for (int i=col; i<m; ++i)
            swap (Temp[sel][i], Temp[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = Temp[i][col] / Temp[row][col];
                for (int j=col; j<m; ++j)
                    Temp[i][j] -= Temp[row][j] * c;
            }
        ++row;
    }
    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = Temp[where[i]][m] / Temp[where[i]][i];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * Temp[i][j];
        if (fabs (sum - Temp[i][m]) > EPS)
            return 0;
```

2.13.5. Gauss Determinant

```cpp
            return 0;
        res = res * temp[i][i] % mod;
    }
    if (res < 0)
        res += mod;
    return static_cast<int>(res);
}
int detGauss(){
    assert(r == c);
    double det = 1;
    Matrix<double> temp(r);
    temp.r = r;
    temp.c = c;
    int n = r;
    for(int i = 0;i<n;i++)
        for(int j = 0;j<n;j++)
            temp[i][j] = (double)A[i][j];
    for (int i=0; i<n; ++i) {
        int k = i;
        for (int j=i+1; j<n; ++j)
            if (fabs (temp[j][i]) > fabs (temp[k][i]))
                k = j;
        if (abs (temp[k][i]) < EPS) {
            det = 0;
            break;
        }
        swap (temp[i], temp[k]);
        if (i != k)
            det = -det;
        det *= temp[i][i];
        for (int j=i+1; j<n; ++j)
            temp[i][j] /= temp[i][i];
```

2.13.6. Cofactors Matrix

```cpp
    }
    for(int i = 0;i<n;i++)
```

```
        for(int j = 0;j<n;j++)
            inverse[i][j] = temp[i][j+n];
    return true;
}
//Adjoint
Matrix<T> minor(int x, int y){
    Matrix<T> M(r-1, c-1);
    for(int i = 0; i < c-1; ++i)
        for(int j = 0; j < r-1; ++j)
            M[i][j] = A[i < x ? i : i+1][j < y ? j :
             j+1];
    return M;
}
T cofactor(int x, int y){
    T ans = minor(x, y).detGauss();
    if((x + y) % 2 == 1) ans *= -1;
    return ans;
}
```

2.13.7. Matriz inversa

```
    (*this) = (*this)-B;
}

void operator ^= (long long n){
    (*this)  =(*this)^n;
}
//Inverse
bool Inverse(Matrix<double> &inverse){
    if(this->detGauss() == 0)return false;
    int n = A[0].size();
    Matrix<double> temp(n,2*n);
    for(int i = 0;i<n;i++)
        for(int j = 0;j<n;j++)temp[i][j] =  A[i][j];
    Matrix<double> ident(n);
    for(int i = 0;i<n;i++)
        for(int j = n;j<2*n;j++)temp[i][j] =
         ident[i][j-n];
    int m = n*2;
    vector<int> where (m, -1);

    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (temp[i][col]) > abs
             (temp[sel][col]))
                sel = i;
        if (abs (temp[sel][col]) < EPS)
            continue;
        for (int i=col; i<m; ++i)
            swap (temp[sel][i], temp[row][i]);
        where[col] = row;
        double div = temp[row][col];
        for(int i = 0;i<m;i++)
            if(fabs(temp[row][i])>EPS)temp[row][i]
             /=div;
        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = temp[i][col] /
                 temp[row][col];
                for (int j=col; j<m; ++j)
                    temp[i][j] -= temp[row][j] * c;
            }
```

2.13.8. Adjoint Matrix

```
Matrix<T> cofactorMatrix(){
    Matrix<T> C(r, c);
    for(int i = 0; i < c; i++)
        for(int j = 0; j < r; j++)
            C[i][j] = cofactor(i, j);
    return C;
}
Matrix<T> Adjunta(){
    int n = A[0].size();
    Matrix<int> adjoint(n);
    Matrix<double> inverse(n);
    this->Inverse(inverse);
    int determinante = this->detGauss();
    if(determinante){
        for(int i = 0;i<n;i++)
            for(int j = 0;j<n;j++)
```

2.13.9. Recurrencias lineales

```
lli Linear_recurrence(vector<lli> C, vector<lli> init,lli
↪  n,bool constante){
    int k = C.size();
    Matrix<lli> T(k,k);
    Matrix<lli> first(k,1);
    for(int i = 0;i<k;i++)T[0][i] = C[i];
    for(int i = 0,col=1;i<k && col<k;i++,col++)
        T[col][i]=1;
    if(constante){
        for(int i = 0;i<k;i++)first[i][0]=init[(k-2)-i];
        first[k-1][0]=init[k-1];
    }
    else
        for(int i = 0;i<k;i++)first[i][0]=init[(k-1)-i];
    if(constante)
        T^=((n-k)+1);
    else
        T^=(n-k);
    Matrix<lli> sol = T*first;
    return sol[0][0];
}
```

2.13.10. 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

```
    while(t--){
        cin>>n>>m>>k;
        Matrix<lli> Kirchof(n);
        for(int i = 0;i<m;i++){
            cin>>a>>b;
            a--;
            b--;
            Kirchof[a][b] = Kirchof[b][a] = 1;
            Kirchof[a][a]++;
            Kirchof[b][b]++;
        }
        for(int i =0;i<n;i++)
            Kirchof[i][i] =
            ↪  (1ll*n*k%mod-Kirchof[i][i]+mod)%mod;
        lli ans = 1;
        ans = ans*(mod_pow(1ll*k*n%mod*k%mod*n%mod,mod-2));
        lli determinante =Kirchof.det();
        ans  = ans*(mod_pow(determinante,k))%mod;
        cout<<ans<<endl;
    }
```

2.14. Metodos numericos

2.14.1. 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;
}
```

2.15. Combinatorics

2.15.1. 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 M = 1e6;
const lli mod = 986444681;
vector<lli> fact(M+1, 1), inv(M+1, 1), invfact(M+1, 1);
lli ncr(lli n, lli r){
  if(r < 0 || r > n) return 0;
  return fact[n] * invfact[r] % mod * invfact[n - r] % mod;
}
void calc(int m){
     for(int i = 2; i <= M; ++i){
    fact[i] = (lli)fact[i-1] * i % mod;
    inv[i] = mod - (lli)inv[mod % i] * (mod / i) % mod;
    invfact[i] = (lli)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 fact[n] * invfact[r] % mod * invfact[n - r] % 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;
}

//   Computes C(N,R)%p
lli power(lli x, lli y, lli p) {
    lli res = 1;
    x = x % p;
    while (y > 0) {
        if (y & 1)
            res = (res*x) % p;
        y = y>>1;
        x = (x*x) % p;
    }
    return res;
}

lli modInverse(lli n, lli p) {
```

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

3. Facts

3.1. Números de Catalán

están definidos por la recurrencia:

$$C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1}$$

3.2. Números de Stirling de primera clase

son el número de permutaciones de n elementos con exactamente k ciclos disjuntos.

$$\begin{bmatrix} n \\ k \end{bmatrix} = (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$$

3.3. Números de Stirling de segunda clase

son el número de particionar un conjunto de n elementos en k subconjuntos no vacíos.

$$\begin{Bmatrix} n \\ k \end{Bmatrix} >= k\begin{Bmatrix} n-1 \\ k \end{Bmatrix} + \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix}$$

Además:

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j} j^n$$

3.4. Números de Bell

cuentan el número de formas de dividir n elementos en subconjuntos.

$$\mathcal{B}_{n+1} = \sum_{k=0}^{n} \binom{n}{k} \mathcal{B}_k$$

x	0	1	2	3	4	5	6	7	8	9	10
$\mathcal{B}_x$	1	1	2	5	15	52	203	877	4.140	21.147	115.975

3.5. Derangement

permutación que no deja ningún elemento en su lugar original

$$!n = (n-1)(!(n-1)+!(n-2)); !1 = 0, !2 = 1$$

$$!n = n! \sum_{k=0}^{n} \frac{(-1)^k}{k!}$$

3.6. Números armónicos

$$H_n = \sum_{k=1}^{n} \frac{1}{k}$$

$$\frac{1}{2n+1} < H_n - \ln n - \gamma < \frac{1}{2n}$$

$$\gamma = 0.577215664901532860606512090082402431042159335\ldots$$

3.7. Número de Fibonacci

$f_0 = 0,\ f_1 = 1$:

$$f_n = \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^n - \frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2})^n$$

$$f_{n+1} = f_n * 2 - f_{n-2}$$

$$f_0 + f_1 + f_2 + \cdots + f_n = f_{n+2} - 1$$

$$f_0 - f_1 + f_2 - \cdots + (-1)^n f_n = (-1)^n f_{n-1} - 1$$

$$f_1 + f_3 + f_5 + \cdots + f_{2n-1} = f_{2n}$$

$$f_0 + f_2 + f_4 + \cdots + f_{2n} = f_{2n+1} - 1$$

$$f_0^2 + f_1^2 + f_2^2 + \cdots + f_n^2 = f_n f_{n+1}$$

$$f_1 f_2 + f_2 f_3 + f_3 f_4 + \cdots + f_{2n-1} f_n = f_{2n}^2$$

$$f_1 f_2 + f_2 f_3 + f_3 f_4 + \cdots + f_{2n} f_{2n+1} = f_{2n+1}^2 - 1$$

$$k \geq 1 \Rightarrow f_{n+k} = f_k f_{n+1} + f_{k-1} f_n \forall n \geq 0$$

Identidad de Cassini: $f_{n+1} fn - 1 - f_n^2 = (-1)^n$

$$f_{n+1}^2 + f_n^2 = f_{2n+1}$$

$$f_{n+2}^2 - f_n^2 = f_{2n+2}$$

$$f_{n+2}^2 - f_{n+1}^2 = f_n f_{n+3}$$

$$f_{n+2}^3 - f_{n+1}^3 - f_n^3 = f_{3n+3}$$

$$mcd(f_n, f_m) = f_{mcd(n,m)}$$

$$f_{n+1} = \sum_{j=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n-j}{j}$$

$$f_{3n} = \sum_{j=0}^{n} \binom{n}{j} 2^j f_j$$

El último dígito de cada número se repite periódicamente cada 60 números. Los dos últimos, cada 300; a partir de ahí, se repiten cada $15{*}10^{n-1}$ números.

3.8. Sumas de combinatorios

$$\sum_{i=n}^{m} \binom{i}{n} = \binom{m+1}{n+1}$$

$$\sum_{i=0}^{k} \binom{n}{i}\binom{m}{k-i} = \binom{n+m}{k}$$

3.9. Funciones generatrices

Una lista de funciones generatrices para secuencias útiles:

$(1, 1, 1, 1, 1, 1, \ldots)$	$\frac{1}{1-z}$
$(1, -1, 1, -1, 1, -1, \ldots)$	$\frac{1}{1+z}$
$(1, 0, 1, 0, 1, 0, \ldots)$	$\frac{1}{1-z^2}$
$(1, 0, \ldots, 0, 1, 0, 1, 0, \ldots, 0, 1, 0, \ldots)$	$\frac{1}{1-z^2}$
$(1, 2, 3, 4, 5, 6, \ldots)$	$\frac{1}{(1-z)^2}$
$(1, \binom{m+1}{m}, \binom{m+2}{m}, \binom{m+3}{m}, \ldots)$	$\frac{1}{(1-z)^{m+1}}$
$(1, c, \binom{c+1}{2}, \binom{c+2}{3}, \ldots)$	$\frac{1}{(1-z)^c}$
$(1, c, c^2, c^3, \ldots)$	$\frac{1}{1-cz}$
$(0, 1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots)$	$\ln \frac{1}{1-z}$

Truco de manipulación:

$$\frac{1}{1-z}G(z) = \sum_n \sum_{k\leq n} g_k z^n$$

3.10. The twelvefold way

¿Cuántas funciones $f\colon N \to X$ hay?

N	X	Any f	Injective	Surjective
dist.	dist.	x^n	$(x)_n$	$x!\left\{ {n \atop x} \right\}$
indist.	dist.	$\binom{x+n-1}{n}$	$\binom{x}{n}$	$\binom{n-1}{n-x}$
dist.	indist.	$\left\{ {n \atop 1} \right\} + \ldots + \left\{ {n \atop x} \right\}$	$[n \leq x]$	$\left\{ {n \atop k} \right\}$
indist.	indist.	$p_1(n) + \ldots p_x(n)$	$[n \leq x]$	$p_x(n)$

Where $\binom{a}{b} = \frac{1}{b!}(a)_b$ and $p_x(n)$ is the number of ways to partition the integer n using x summands.

3.11. Teorema de Euler

si un grafo conexo, plano es dibujado sobre un plano sin intersección de aristas, y siendo v el número de vértices, e el de aristas y f la cantidad de caras (regiones conectadas por aristas, incluyendo la región externa e infinita), entonces

$$v - e + f = 2$$

3.12. Burnside's Lemma

Si X es un conjunto finito y G es un grupo de permutaciones que actúa sobre X, sean $S_x = \{g \in G : g * x = x\}$ y $Fix(g) = \{x \in X : g * x = x\}$. Entonces el número de órbitas está dado por:

$$N = \frac{1}{|G|}\sum_{x\in X}|S_x| = \frac{1}{|G|}\sum_{g\in G}|Fix(g)|$$

3.13. Ángulo entre dos vectores

Sea α el ángulo entre $\vec{a}$ y $\vec{b}$:

$$\cos\alpha = \frac{\vec{a}\cdot\vec{b}}{\|\vec{a}\|\|\vec{b}\|}$$

3.14. Proyección de un vector

Proyección de $\vec{a}$ sobre $\vec{b}$:

$$\mathrm{proy}_{\vec{b}}\vec{a} = (\frac{\vec{a}\cdot\vec{b}}{\vec{b}\cdot\vec{b}})\vec{b}$$