ACM ICPC Reference

University of São Paulo

May 13, 2015

Contents

1 Template

iniciodeprova.txt

```
###### Emacs config ######

(setq-default indent-tabs-mode nil)
(setq c-basic-offset 4)
(global-linum-mode 1)
(electric-indent-mode 1)
(global-unset-key "\C-z")
(global-unset-key "\C-x\C-z")
(global-hl-line-mode 1)
(set-face-background 'default "#202020")
(set-face-foreground 'default "White")
(set-face-background 'hl-line "#003399")

###### vimrc ######

filetype plugin on
filetype indent on
set number
syntax on
set shiftwidth=4
set tabstop=4
colorscheme evening

2b74 #include <bits/stdc++.h>
916e using namespace std;
916e
a947 #define debug(x...) fprintf(stderr,x)
f69c #define pb push_back
dd8c #define f(i,x,y) for(int i=x; i<y; i++)
40f4 #define quad(x) ((x)*(x))
a04e #define clr(x,y) memset(x,y,sizeof x)
2546 #define fst first
7c08 #define snd second
7c08
dd14 typedef pair<int,int> pii;
45e9 typedef long long ll;
a13c const int INF = 0x3f3f3f3f;
2b93 const ll LINF = 0x3f3f3f3f3f3f3f3fll;
```

../hashify.py

```
import hashlib,sys

m = hashlib.md5()
for line in sys.stdin.readlines():
    safe = line
    if line.find("//") != -1:
        line = line[:line.find("//")]
    trimmed = line.replace(" ","").replace("\n","").replace("\t","")
    m.update(trimmed)
    hash = m.hexdigest()[:4]
    print "%s %s"%(hash,safe),
```

2 Lista de bugs e recomendações

- Reler enunciado e pedir clarifications (evitar explicar o enunciado para outra pessoa).
- Verificar overflows.
- Ver se o ∞ é tão infinito quanto parece e se o *eps* é tão pequeno quanto o necessário.
- Comparação de ponto flutuante com tolerância.
- Verificar se o grafo pode ser desconexo.
- Verificar se pode haver self-loops, arestas com peso negativo ou ligando um mesmo par de vértices.
- Cuidar de casos com pontos coincidentes e pontos colineares.
- Igualdade dentro de if (a == b ao invés de a = b)
- Verificar trechos de código quase iguais ou copy-pasted.
- Verificar tamanho de vetores.
- Overflow em shift (1ll << 40 ao invés de 1 << 40)
- Não usar variáveis com nome min, max, next.
- Verificar inicialização de variáveis.
- Verificar casos extremos, muito pequenos ou muito grandes, caso zero.
- Cuidar de imprecisões ao subtrair números quase iguais.
- Tomar cuidado com resto de divisão envolvendo números negativos.
- Não comparar unsigned int (.size()) com int negativo.

3 Numbers and Number Theory

Extended Euclid's Algorithm (Bézout's Theorem)

```
d41d //O valor retornado eh gcd(a,b) = ax + by
a674 ll gcd_extended(ll a, ll b, ll &x, ll &y) {
c994     if (a == 0) { x = 0, y = 1; return b; }
7459     ll xx, yy, d = gcd_extended(b%a, a, xx, yy);
8d0e     x = yy-(b/a)*xx, y=xx;
5bdc     if (d < 0) {d = -d; x = -x; y = -y; }
9d45     return d;
1255 }
```

Chinese Remainder Theorem

Suppose $m_0, m_1, ..., m_{k-1}$ are positive integers that are pairwise coprime. Then, for any given sequence of integers $a_1, a_2, ..., a_{k-1}$, there exists an integer X solving the following system of simultaneous congruences.

$$X \equiv a_0 \pmod{m_0}$$

$$X \equiv a_1 \pmod{m_1}$$

$$\vdots$$

$$X \equiv a_{k-1} \pmod{m_{k-1}}$$

Furthermore, all solutions X of this system are congruent modulo the product, $M = m_0 m_1 ... m_{k-1}$. A possible solution for this system is:

$X = \sum_{i=0}^{k-1} \frac{M}{m_i} b_i a_i$, where b_i is a integer, such that: $\forall i, 0 \leq i \leq k-1, \frac{M}{m_i} b_i \equiv 1 \pmod{m_i}$.

OBS.: Observações sobre chinesResto.cpp.

- Não usa o algoritmo mostrado acima;
- Se o sistem tem solução, armazena em X o valor da menor solucao positiva e retorna true. Retorna false caso contrário;
- Complexidade : O(k * log(maxm[i]), sem usar mulmod;
- CUIDADO: l = lcm(m[0],m[1],...,m[k-1]) cuidado para não estourar long long;
- CUIDADO: os valores do vetor a[0...k-1] são alterados;
- CUIDADO: assume que k > 1;

```
9b9c ll gcd(ll a, ll b) { return a?gcd(b%a, a):abs(b); }
1c31 ll lcm(ll x, ll y) { return (x&&y) ? abs(x) / gcd(x,y) * abs(y): 0; }
1c31
1c31 //(a*b) % mod => O(log b)
839d ll mulmod(ll a, ll b, ll mod) {
28c3     if (b < 0) return mulmod(a, (b%mod + mod)%mod, mod);
d300     if (b == 0) return 0LL;
82f4     ll ans = (2LL * mulmod(a, b/2, mod)) % mod;
bec2     if (b%2 == 0) return ans;
18ca     return (ans + a) % mod;
12b8 }
12b8
12b8 //para td i, (0 <= i < k), x = a[i] ( mod m[i])
ce3b ll a[MAX], m[MAX];
9e85 bool chines_resto(ll &X, int k) {
9e85     //Assume que k >= 1
8818     ll d, z, w, l = m[0];
ad2e     X = (a[0] % m[0] + m[0]) % m[0];
d092     for (int i = 1; i < k; i++) {
e610             a[i] %= m[i];
aa7c             d = gcd_extended(l, m[i], z, w);
9043             if ( (a[i]-X) % d != 0) return false;
a3a9             X += l*z*((a[i]-X)/d); //Pode usar mulmod(), pra nao estourar ll
5d5c             l = lcm(l, m[i]);
062f             X = ((X%l) + l) % l;
bba8     }
01a9     return true;
ea7e }
```

Miller–Rabin (Primality test)

```
a288 llu llrand() { llu a = rand(); a<<= 32; a+= rand(); return a;}
67b7 int is_probably_prime(llu n) {
61d5     if (n <= 1) return 0;
2ecf     if (n <= 3) return 1;
a093     llu s = 0, d = n - 1;
0127     while (d % 2 == 0) {
028a         d/= 2; s++;
1c22     }
6cab     for (int k = 0; k < 64; k++) {
fc88         llu a = (llrand() % (n - 3)) + 2;
9d61         llu x = exp_mod(a, d, n);
e9cb         if (x != 1 && x != n-1) {
6e13             for (int r = 1; r < s; r++) {
1479                 x = mul_mod(x, x, n);
569b                 if (x == 1)
7ee2                     return 0;
74f4                 if (x == n-1)
344f                     break;
429d             }
c1fc             if (x != n-1)
85bd                 return 0;
03b9         }
abcb     }
8fad     return 1;
78e3 }
```

Pollard's Rho (Factorization)

```
295a llu rho(llu n) {
dd00     llu d, c = rand() % n, x = rand() % n, xx = x;
77b5     if (n % 2 == 0)
d711         return 2;
410c     do {
6200         x = (mul_mod(x, x, n) + c) % n;
72a6         xx = (mul_mod(xx, xx, n) + c) % n;
7ba8         xx = (mul_mod(xx, xx, n) + c) % n;
bf50         d = gcd(val_abs(x - xx), n);
07a4     } while (d == 1);
4ae0     return d;
0884 }
b528 map <llu,int> F;
6ac2 void factor(llu n) {
3fa3     if (n == 1)
aa26         return;
d6b5     if (is_probably_prime(n)) {
780e         F[n]++;
7609         return;
1f13     }
6468     llu d = rho(n);
0bcb     factor(d);
79c1     factor(n/d);
838b }
```

Brent's Algorithm (Cycle detection)

Let $x_0 \in S$ be an element of the finite set S and consider a function $f : S \to S$. Define

$$f_k(x) = \begin{cases} x, & k = 0 \\ f\big(f_{k-1}(x)\big), & k > 0 \end{cases}.$$

Clearly, there exists distinct numbers $i, j \in \mathbb{N}$, $i \neq j$, such that $f_i(x_0) = f_j(x_0)$.

Let $\mu \in \mathbb{N}$ be the least value such that there exists $j \in \mathbb{N} \setminus \{\mu\}$ such that $f_\mu(x_0) = f_j(x_0)$ and let $\lambda \in \mathbb{N}$ be the least value such that $f_\mu(x_0) = f_{\mu+\lambda}(x_0)$.

Given x_0 and f, this code computes μ and λ applying the operator f $\mathcal{O}(\mu + \lambda)$ times and storing at most a constant amount of elements from S.

Lucca's Theorem

For non-negative integers m and n and a prime p, the following congruence relation holds:

$$\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} \pmod{p},$$

where
$m = m_k p^k + m_{k-1} p^{k-1} + ... + m_1 p + m_0$
and
$n = n_k p^k + n_{k-1} p^{k-1} + ... + n_1 p + n_0$
are the base p expansions of m and n respectively.

Simpson's Rule for numerical integration

Simpson's rule is a method for numerical integration, the numerical approximation of definite integrals.

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

- Suppose that the interval $[a, b]$ is split up in n subintervals, with n an even number. Then, the composite Simpson's rule is given by:

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{h}{3}\left[f(x_0) + 2\sum_{j=1}^{\frac{n}{2}-1} f(x_{2j}) + 4\sum_{j=1}^{\frac{n}{2}} f(x_{2j-1}) + f(x_n)\right]$$

Where $x_j = a + jh$ for $j = 0, 1, ..., n-1, n$, with $h = (b-a)/n$ (in particular, $x_0 = a$ and $x_n = b$).
The above formula can also be written as:

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + ... + 4f(x_{n-1}) + f(x_n)\right]$$

Lagrange Interpolating Polynomial

The Lagrange interpolating polynomial is the polynomial $P(x)$ of degree $\leq (n-1)$ that passes through the n points $(x_1, y_1 = f(x_1)), (x_2, y_2 = f(x_2)), ..., (x_n, y_n = f(x_n))$, and is given by

$$P(x) = \sum_{i=1}^{n} y_i P_i(x),$$

where

$$P_i(x) = \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{x - x_k}{x_i - x_k}$$

Written explicitly,

$$\begin{aligned} P(x) = {} & \frac{(x-x_2)(x-x_3)...(x-x_n)}{(x_1-x_2)(x_1-x_3)...(x_1-x_n)} y_1 + \frac{(x-x_1)(x-x_3)...(x-x_n)}{(x_2-x_1)(x_2-x_3)...(x_2-x_n)} y_2 + ... \\ & + \frac{(x-x_1)(x-x_2)...(x-x_{n-1})}{(x_n-x_1)(x_n-x_2)...(x_n-x_{n-1})} y_n. \end{aligned}$$

Farey Sequence

Farey Sequence of order n is the sequence of completely reduced fractions between 0 and 1 which, when in lowest terms, have denominators less than or equal to n, arranged in order of increasing size.

$F_1 = \{0/1, 1/1\}$, $F_2 = \{0/1, 1/2, 1/1\}$, $F_3 = \{0/1, 1/3, 1/2, 2/3, 1/1\}, ...$

- From this, we can relate the lengths of F_n and F_{n-1} using Euler's totient function $\varphi(n)$:

$$|F_n| = |F_{n-1}| + \varphi(n)$$

- The asymptotic behaviour of $|F_n|$ is : $|F_n| \sim \frac{3n^2}{\pi^2}$.

```
d41d //gera a n-th sequencia de Farey
d41d //conjunto de numeros racionais irredutiveis a/b, 0<=a<=b<=n && gcd(a,b)=1.
d41d //Os valores sao armazenados em ordem cresecente em num[0..top-1]/den[0..top-1]
d41d
e3cd short num[10000000], den[10000000];
2e8e int top;
8781 void Farey_sequence(int n) {
6ab6    top = 0;
dea7    num[top] = 0, den[top] = 1;
7c2d    top++;
7c2d
12b6    build_sequence(0, 1, 1, 1, n);
12b6
d0f3    num[top] = 1, den[top] = 1;
c77b    top++;
6cc1 }
6cc1
d737 inline void build_sequence(int a1, int b1, int a2, int b2, int &n) {
877c    if (b1+b2 > n) return;
877c
0509    build_sequence(a1, b1, a1+a2, b1+b2, n);
```

```
c557    num[top] = a1+a2, den[top] = b1+b2;
278a    top++;
278a
8330    build_sequence(a1+a2, b1+b2, a2, b2, n);
f7c2 }
```

Euler's Totient Function

$$\varphi(n) = n\prod_{p|n}\left(1-\frac{1}{p}\right); \qquad \varphi(p.n) = \begin{cases}(p-1)\varphi(n), & p \nmid n \\ p\varphi(n), & p \mid n\end{cases};$$

$$\varphi(mn) = \varphi(m)\varphi(n).\frac{d}{\varphi(d)}, where\ d = gcd(m,n).\ Note\ the\ special\ cases.$$

$$\varphi(p^k) = p^k\left(1-\frac{1}{p}\right); \qquad \sum_{d|n}\varphi(d) = n;$$

Euler's theorem : if a and n are relatively prime then

$$a^{\varphi(n)} \equiv 1 \pmod n$$

Pillai's arithmetical function

$$P(n) := \sum_{k=1}^{n} gcd(k,n) \ \ (n \in \mathbb{N} := \{1,2,...\})$$

$$P(n) = \sum_{d|n} d\varphi(n/d) \ \ (n \in \mathbb{N})$$

For every prime power p^a, $(a \in \mathbb{N})$, $P(p^a) = (a+1)p^a - ap^{a-1}$.

$P(n)$ is multiplicative, ie, for every integers a and b, such that $gcd(a,b) = 1$, $P(ab) = P(a)P(b)$.

Let $P_{altern}(n)$ be the alternating gcd-sum function. Let $n \in \mathbb{N}$ and write $n = 2^a m$, where $a \in \mathbb{N}_0 := \{0,1,2,....\}$ and $m \in \mathbb{N}$ is odd. Then

$$P_{altern}(n) := \sum_{k-1}^{n}(-1)^{k-1}gcd(k,n) = \begin{cases} n, & \text{if n is odd;} \\ -2^{a-1}aP(m) = -\frac{a}{a+2}P(n), & \text{if n is even.}\end{cases}$$

Gaussian Elimination

Can be easily adapted to integers mod P, fractions and long doubles.

```
b94a const int MAXN = 110;
b94a
6d0c typedef double Number;
c19b const Number EPS = 1e-9;
c19b
33d8 Number mat[MAXN][MAXN];
bdd8 int idx[MAXN];   // row index
09a1 int pivot[MAXN]; // pivot of row i
09a1
09a1 // Solves Ax = B, where A is a neq x nvar matrix and B is mat[*][nvar]
09a1 // Returns a vector of free variables (empty if system is defined,
09a1 // or {-1} if no solution exists)
09a1 // Reduces matrix to reduced echelon form
c993 vector<int> solve(int nvar, int neq) {
b764     for(int i = 0; i < neq; i++) idx[i] = i;
96b9     int currow = 0;
a12c     vector<int> freeVars;
3bf0     for(int col = 0; col < nvar; col++) {
43da         int pivotrow = -1;
e8ad         Number val = 0;
45fb         for(int row = currow; row < neq; row++) {
c4e9             if(fabs(mat[idx[row]][col]) > val + EPS) {
7188                 val = fabs(mat[idx[row]][col]);
ddf6                 pivotrow = row;
b465             }
0aab         }
f452         if(pivotrow == -1) { freeVars.push_back(col); continue; }
291f         swap(idx[currow], idx[pivotrow]);
d1de         pivot[currow] = col;
d898         for(int c = 0; c <= nvar; c++) {
2e81             if(c == col) continue;
558c             mat[idx[currow]][c] = mat[idx[currow]][c] / mat[idx[currow]][col];
e221         }
868e         mat[idx[currow]][col] = 1;
1320         for(int row = 0; row < neq; row++) {
d2ac             if(row == currow) continue;
f3af             Number k = mat[idx[row]][col] / mat[idx[currow]][col];
68fa             for(int c = 0; c <= nvar; c++)
ee55                 mat[idx[row]][c] -= k * mat[idx[currow]][c];
aae1         }
95f1         currow++;
f9ce     }
c884     for(int row = currow; row < neq; row++)
b5d4         if(mat[idx[row]][nvar] != 0) return vector<int>(1, -1);
e4b2     return freeVars;
19cf }
```

Fast Fourier Transform

If you are calculating the product of polynomials, don't forget to set the vector's size to **at least the sum of degrees** of both polynomials, regardless of whether you will use only the first few elements of the array.

```
6dba typedef complex<long double> Complex;
b9dc const long double PI = acos(-1.0L);
b9dc
b9dc // Computes the DFT of vector v if type = 1, or the IDFT if type = -1
6ca7 vector<Complex> FFT(vector<Complex> v, int type) {
7ca9     int n = v.size();
7e69     while(n&(n-1)) { v.push_back(0); n++; }
e1c7     int logn = __builtin_ctz(n);
```

```
4d07      vector<Complex> v2(n);
7df5      for(int i=0; i<n; i++) {
3e55          int mask = 0;
ba11          for(int j=0; j<logn; j++) if(i&(1<<j)) mask |= (1<<(logn - 1 - j));
a97d          v2[mask] = v[i];
5e4f      }
1a64      for(int s=0, m=2; s<logn; s++, m<<=1) {
48a4          Complex wm(cos(2.L * type * PI / m), sin(2.L * type * PI / m));
c108          for(int k=0; k<n; k+=m) {
7a46              Complex w = 1;
15e7              for(int j=0; 2*j<m; j++) {
2b20                  Complex t = w * v2[k + j + (m>>1)], u = v2[k + j];
1085                  v2[k + j] = u + t; v2[k + j + (m>>1)] = u - t;
8e81                  w *= wm;
66f2              }
a81b          }
c40c      }
b83f      if(type == -1) for(Complex &c: v2) c /= n;
a0e0      return v2;
5d65 }
```

Least Squares Method

Let $f, g_0, g_1, \ldots, g_{n-1}$ be given functions. The coefficients of the function $g(x) = a_0 g_0(x) + a_1 g_1(x) + \ldots + a_{n-1} g_{n-1}(x)$ that best fits $f(x)$ so that the inner product $< f-g, f-g >$ is minimum is given by the system of equations below.

$$\begin{bmatrix} <g_0,g_0> & <g_0,g_1> & \cdots & <g_0,g_{n-1}> \\ <g_1,g_0> & <g_1,g_1> & \cdots & <g_1,g_{n-1}> \\ \vdots & \vdots & \ddots & \vdots \\ <g_{n-1},g_0> & <g_{n-1},g_1> & \cdots & <g_{n-1},g_{n-1}> \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} <f,g_0> \\ <f,g_1> \\ \vdots \\ <f,g_{n-1}> \end{bmatrix}$$

In the linear discrete case when $g(x) = ax + b$, a and b can also be found using the following equations:

$$a = \frac{\sum_{i=1}^n x_i (y_i - \bar{y})}{\sum_{i=1}^n x_i (x_i - \bar{x})}, \qquad b = \bar{y} - a\bar{x}$$

where $\bar{x}$ and $\bar{y}$ are the mean value of x and y, respectively.

4 Combinatorics

Catalan Numbers

C_n is:

- The number of balanced expressions built from n pairs of parentheses.
- The number of paths in an $n \times n$ grid that stays on or below the diagonal.
- The number of words of size $2n$ over the alphabet $\Sigma = \{a, b\}$ having an equal number of a symbols and b symbols containing no prefix with more a symbols than b symbols.
- (starting with C_0): 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452...

It holds that:

$$C_0 = 1, C_{n+1} = \sum_{k=0}^{n} C_k C_{n-k}$$

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

Stirling Numbers of the First Kind

$\begin{bmatrix} n \\ k \end{bmatrix}$ is:

- For integers $n \geq k \geq 0$, $\begin{bmatrix} n \\ k \end{bmatrix}$ counts the number of permutations of n elements with exactly k cycles.
- The number of digraphs with n vertices and k cycles such that each vertex has in and out degree of 1.

It holds that:

$$\begin{bmatrix} n \\ 0 \end{bmatrix} = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}, \quad \begin{bmatrix} 0 \\ k \end{bmatrix} = \begin{cases} 1, & k = 0 \\ 0, & k \neq 0 \end{cases}; \quad \begin{bmatrix} n \\ k \end{bmatrix} = (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix};$$

$$\begin{bmatrix} n \\ 1 \end{bmatrix} = (n-1)!; \quad \begin{bmatrix} n \\ n-1 \end{bmatrix} = \binom{n}{2}; \quad \begin{bmatrix} n \\ n-2 \end{bmatrix} = \frac{1}{4}(3n-1)\binom{n}{3};$$

$$\begin{bmatrix} n \\ n-3 \end{bmatrix} = \binom{n}{2}\binom{n}{4}; \quad \begin{bmatrix} n \\ 2 \end{bmatrix} = (n-1)! H_{n-1}; \quad \begin{bmatrix} n \\ 3 \end{bmatrix} = \frac{1}{2}(n-1)!\left(H_{n-1}^2 - H_{n-1}^{(2)}\right);$$

$$H_n = \sum_{j=1}^{n} \frac{1}{j}, \quad H_n^{(k)} = \sum_{j=1}^{n} \frac{1}{j^k}; \quad \sum_{k=0}^{n} \begin{bmatrix} n \\ k \end{bmatrix} = n!; \quad \sum_{j=k}^{n} \begin{bmatrix} n \\ j \end{bmatrix}\binom{j}{k} = \begin{bmatrix} n+1 \\ k+1 \end{bmatrix};$$

Stirling Numbers of the Second Kind

$\left\{ n \atop k \right\}$ is the number of ways to partition an n-set into exactly k non-empty disjoint subsets up to a permutation of the sets among themselves. It holds that:

$$\left\{ n \atop 0 \right\} = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}, \quad \left\{ n \atop 1 \right\} = \left\{ n \atop n \right\} = 1$$

$$\left\{ n \atop k \right\} = \left\{ n-1 \atop k-1 \right\} + k\left\{ n-1 \atop k \right\}$$

$$\left\{ n \atop k \right\} \bmod 2 = \begin{cases} 0, & (n-k) \,\&\, \left\lfloor \frac{k-1}{2} \right\rfloor \neq 0 \\ 1, & \text{otherwise} \end{cases},$$

where & is the C bitwise "and" operator.

$$\left\{ n \atop 2 \right\} = 2^{n-1} - 1; \quad \left\{ n \atop n-1 \right\} = \binom{n}{2}; \quad \left\{ n \atop k \right\} = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j} j^n;$$

Bell Numbers

$\mathcal{B}_n$ is the number of equivalence relations on an n-set or, alternatively, the number of partitions of an n-set. It holds that:

$$\mathcal{B}_n = \sum_{k=0}^{n} \left\{ {n \atop k} \right\}; \qquad \mathcal{B}_{n+1} = \sum_{k=0}^{n} \binom{n}{k} \mathcal{B}_k; \qquad \mathcal{B}_n = \frac{1}{e} \sum_{k=0}^{\infty} \frac{k^n}{k!};$$

$$\mathcal{B}_{n+p} \equiv \mathcal{B}_n + \mathcal{B}_{n+1} \pmod p$$

- (starting with $\mathcal{B}_0$): 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437, 190899322, 1382958545, 10480142147, 82864869804, 682076806159, 5832742205057, 51724158235372, 474869816156751, 4506715738447323, 44152005855084346, 445958869294805289, 4638590332229999353, ...

Narayana Numbers

$N(n, k)$, $n = 1, 2, 3..., 1 \le k \le n$, is the number of expressions containing n pairs of parentheses which are correctly matched and which contain k distinct nestings. For instance, N(4, 2) = 6 as with four pairs of parentheses six sequences can be created which each contain two times the sub-pattern '()': ()((())) (())(()) (()(())) ((()())) ((())()) ((()))().

$$N(n, k) = 0, \;\; if \;\; k > n$$

$$N(n, 1) = N(n, n) = 1$$

$$N(n, k) = \frac{1}{n} \binom{n}{k} \binom{n}{k-1}$$

$$N(n, k) = \frac{n(n-1)}{(n-k)(n-k+1)} N(n-1, k)$$

$$N(n, 1) + N(n, 2) + N(n, 3) + ... + N(n, n) = C_n (Catalan \;\; Number)$$

The Twelvefold Way

Let A be a set of m balls and B be a set of n boxes. The following table provides methods to compute the number of equivalent functions $f : A \to B$ satisfying specific constraints.

Balls	Boxes	Any	Injective	Surjective
$\not\equiv$	$\not\equiv$	n^m	$\frac{n!}{(n-m)!}$	$n!\left\{ {m \atop n} \right\}$
$\not\equiv$	$\equiv$	$\sum_{k=0}^{n} \left\{ {m \atop k} \right\}$	$\delta_{m \leqslant n}$	$\left\{ {m \atop n} \right\}$
$\equiv$	$\not\equiv$	$\binom{m+n-1}{m}$	$\binom{n}{m}$	$\binom{m-1}{n-1}$
$\equiv$	$\equiv$	**(*)** $\sum_{k=0}^{n} p(m, k)$	$\delta_{m \leqslant n}$	**(**)** $p(m, n)$

(**) is a definition and both **(*)** and **(**)** are very hard to compute. So do not try to.

Partition (number theory)

A partition of a positive integer n, is a way of writing n as a sum of positive integers. Two sums that differ only in the order of their summands are considered to be the same partition. The number of partitions of n is given by the partition function $p(n)$. Ex: $p(4) = 5$ => {1+1+1+1} , {1+1+2} , {1+3} , {2+2} , {4}.

- The number of partitions of n in which the greatest part is m is equal to the number of partitions of n into m parts.
- The number of partitions of n in which all parts are 1 or 2 (or, equivalently, the number of partitions of n into 1 or 2 parts) is $\lfloor \frac{n}{2} + 1 \rfloor$.
- (starting with $p(0) = 1$): 1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42, 56, 77, 101, 135, 176, 231, 297, 385, 490, 627...
- Summation to calculate $p(n)$ for the first n elements in $O(n\sqrt{n})$ time. $p(n) = \sum_k (-1)^{k-1} p(n - \frac{k(3k-1)}{2})$, where the summation is over all **nonzero** integers k (positive and negative) and p(m) is taken to be 0 if $m < 0$.

$p(5k + 4) \equiv 0 \pmod 5$
$p(7k + 5) \equiv 0 \pmod 7$
$p(11k + 6) \equiv 0 \pmod{11}$
$p(11^3.13.k + 237) \equiv 0 \pmod{13}$

Derangement (Desarranjo)

A derangement is a permutation of the elements of a set such that none of the elements appear in their original position.

Suppose that there are n persons numbered $1, 2, \ldots, n$. Let there be n hats also numbered $1, 2, \ldots, n$. We have to find the number of ways in which no one gets the hat having same number as his/her number. Let us assume that first person takes the hat i. There are $n - 1$ ways for the first person to choose the number i. Now there are 2 options:

- Person i takes the hat of 1. Now the problem reduces to $n - 2$ persons and $n - 2$ hats.
- Person i does not take the hat 1. This case is equivalent to solving the problem with $n - 1$ persons $n - 1$ hats (each of the remaining $n - 1$ people has precisely 1 forbidden choice from among the remaining $n - 1$ hats).

From this, the following relation is derived:

$$d_n = (n - 1) * (d_{n-1} + d_{n-2})$$

$$d_1 = 0$$

$$d_2 = 1$$

Starting with $n = 0$, the numbers of derangements of n are: 1, 0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 1334961, 14684570, 176214841, 2290792932.

---

5 Geometry 2D

Point Structure

```
271c inline int cmp(double x, double y = 0, double tol = eps) {
c9c4    return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
4c40 }
4c40
13ed struct point {
5753    double x, y;
0188    point(double x = 0, double y = 0): x(x), y(y) {}
a930    point operator +(point q) { return point(x + q.x, y + q.y); }
4fac    point operator -(point q) { return point(x - q.x, y - q.y); }
9684    point operator *(double t) { return point(x * t, y * t); }
e00a    point operator /(double t) { return point(x / t, y / t); }
08fb    double operator *(point q) {return x * q.x + y * q.y;}//a*b = |a||b|cos(ang)
ed7d    double operator %(point q) {return x * q.y - y * q.x;}//a%b = |a||b|sin(ang)
a117    double polar() { return ((y > -eps) ? atan2(y,x) : 2*Pi + atan2(y,x)); }
370c    double mod() { return sqrt(x * x + y * y); }
92cf    double mod2() { return (x * x + y * y); }
cc80    point rotate(double t) {return point(x*cos(t)-y*sin(t), x*sin(t)+y*cos(t));}
1b1c    int cmp(point q) const {
0db4            if (int t = ::cmp(x, q.x)) return t;
425f            return ::cmp(y, q.y);
fc93    }
ed60    bool operator ==(point q) const { return cmp(q) == 0; }
30cd    bool operator !=(point q) const { return cmp(q) != 0; }
bf2c    bool operator < (point q) const { return cmp(q) < 0; }
b780    static point pivot;
9948 };
522c point point::pivot;
ddfe typedef vector<point> polygon;
```

Auxiliary Functions

```
160a double abs(point p) { return hypot(p.x, p.y); }
ab7c double arg(point p) { return atan2(p.y, p.x); }
ab7c
27db inline int ccw(point p, point q, point r) {
69a1    return cmp((p - r) % (q - r));
34b1 }
34b1
34b1 //Projeta o vetor v sobre o vetor u (cuidado precisao)
06d3 point proj(point v, point u) {
6a85    return u*((u*v) / (u*u));
126a }
126a
126a //\angle(p,q,r)| e o menor angulo entre os vetores u(p-q) e v(r-q)
126a // p->q->r virar pra esquerda => angle(p,q,r) < 0
6c5c inline double angle(point p, point q, point r) {
bf72    point u = p - q, v = r - q;
bc40    return atan2(u % v, u * v);
615a }
615a
615a //Decide se q esta sobre o segmento fechado pr.
a582 bool between(point p, point q, point r) {
51cd    return ccw(p, q, r) == 0 && cmp((p - q) * (r - q)) <= 0;
9480 }
9480
9480 //Decide se os segmentos fechados pq e rs tem pontos em comum
cab9 bool seg_intersect(point p, point q, point r, point s) {
0ee2    point A = q - p, B = s - r, C = r - p, D = s - q;
e529    int a = cmp(A % C) + 2 * cmp(A % D);
9307    int b = cmp(B % C) + 2 * cmp(B % D);
5a00    if (a == 3 || a == -3 || b == 3 || b == -3) return false;
a282    if (a || b || p == r || p == s || q == r || q == s) return true;
e435    int t = (p < r) + (p < s) + (q < r) + (q < s);
c388    return t != 0 && t != 4;
b193 }
b193
b193 // Calcula a distancia do ponto r ao segmento pq.
f311 double seg_distance(point p, point q, point r) {
3285    point A = r - q, B = r - p, C = q - p;
b348    double a = A * A, b = B * B, c = C * C;
3b09    if (cmp(b, a + c) >= 0) return sqrt(a);
d2b9    else if (cmp(a, b + c) >= 0) return sqrt(b);
f5b7    else return fabs(A % B) / sqrt(c);
ed51 }
ed51
ed51 // Classifica o ponto p em relacao ao poligono T.
ed51 // Retorna 0, -1 ou 1 dependendo se p esta no exterior, na fronteira
ed51 // ou no interior de T, respectivamente.
cf13 int in_poly(point p, polygon& T) {
5047    double a = 0; int N = T.size();
7f2c    for (int i = 0; i < N; i++) {
6ea3            if (between(T[i], p, T[(i+1) % N])) return -1;
0935            a += angle(T[i], p, T[(i+1) % N]);
8850    }
294c    return cmp(a) != 0;
8a8d }
8a8d
8a8d //Encontra o ponto de intersecao das retas pq e rs.
550d point line_intersect(point p, point q, point r, point s) {
d6fd    point a = q - p, b = s - r, c = point(p % q, r % s);
49b7    return point(point(a.x, b.x) % c, point(a.y, b.y) % c) / (a % b);
ae66 }
ae66
ae66 // Calcula a area orientada do poligono T.
ae66 // Se o poligono P estiver em setido anti-horario, poly_area(P) > 0,
ae66 // e <0 caso contrario
78f3 double poly_area(polygon& T) {
fe30    double s = 0; int n = T.size();
f7e8    for (int i = 0; i < n; i++)
23db            s += T[i] % T[(i+1) % n];
ccd3    return s / 2;
549a }
549a
```

```
549a //Calcula o incentro de um triangulo
585a point incenter(point p, point q, point r) {
f167     double a = (p-q).mod(), b = (p-r).mod(), c = (q-r).mod();
3b2d     return (r*a + q*b + p*c) / (a + b + c);
7752 }
7752
7752 //Centro de massa de um poligono
3df1 point centro_massa(polygon p) {
e7c6     double x=0., y=0., area = poly_area(p);
9823     p.push_back(p[0]);
6227     for (int i = 0; i < p.size()-1; i++) {
308e             x += (p[i].x + p[i+1].x) * (p[i] % p[i+1]);
5bd2             y += (p[i].y + p[i+1].y) * (p[i] % p[i+1]);
48e2     }
bbb9     return point(x/(6*area), y/(6*area));
efc0 }
```

Convex Hull

```
d41d // Comparacao radial.
d41d // Obs: suponha tds pontos no vetor p[] = p[]-pivot, (pivot = min_elemento(p))
d41d // tds ptos do novo p[] estarao no 1 e 4 quadrante ordenado no sentido anti-hor
f7bf bool radial_lt(point p, point q) {
1cb2     point P = p - point::pivot, Q = q - point::pivot;
69c0     double R = P % Q;
f7e9     if (cmp(R)) return R > 0;
ea84     return cmp(P * P, Q * Q) < 0;
119c }
119c
119c // Determina o fecho convexo de um conjunto de pontos no plano.
119c // Destroi a lista de pontos T.
5b79 polygon convex_hull(vector<point>& T) {
548d     int j = 0, k, n = T.size(); polygon U(n);
76fb     point::pivot = *min_element(all(T));
bfee     sort(all(T), radial_lt);
4a7a     for (k = n-2; k >= 0 && ccw(T[0], T[n-1], T[k]) == 0; k--);
6a93     reverse((k+1) + all(T));
6a93
b5fc     for (int i = 0; i < n; i++) {
b5fc             // troque o >= por > para manter pontos colineares
a64e             while (j > 1 && ccw(U[j-1], U[j-2], T[i]) >= 0) j--;
426e             U[j++] = T[i];
bd98     }
df01     U.erase(j + all(U));
519d     return U;
e39f }
```

Convex Polygon Intersection

```
d41d // Determina o poligono intersecao dos dois poligonos convexos P e Q.
d41d // Tanto P quanto Q devem estar orientados positivamente.(anti-horario)
32b5 polygon poly_intersect(polygon& P, polygon& Q) {
e0a6     int m = Q.size(), n = P.size();
8d65     int a = 0, b = 0, aa = 0, ba = 0, inflag = 0;
1f0c     polygon R;
2546     while ((aa < n || ba < m) && aa < 2*n && ba < 2*m) {
377b             point p1 = P[a], p2 = P[(a+1) % n], q1 = Q[b], q2 = Q[(b+1) % m];
261f             point A = p2 - p1, B = q2 - q1;
43f8             int cross = cmp(A % B), ha = ccw(p2, q2, p1), hb = ccw(q2, p2, q1);
240f             if (cross == 0 && ccw(p1, q1, p2) == 0 && cmp(A * B) < 0) {
22de                     if (between(p1, q1, p2)) R.push_back(q1);
7f4b                     if (between(p1, q2, p2)) R.push_back(q2);
1839                     if (between(q1, p1, q2)) R.push_back(p1);
52b9                     if (between(q1, p2, q2)) R.push_back(p2);
4868                     if (R.size() < 2) return polygon();
ff0c                     inflag = 1; break;
2afe             } else if (cross != 0 && seg_intersect(p1, p2, q1, q2)) {
816a                     if (inflag == 0) aa = ba = 0;
d2e3                     R.push_back(line_intersect(p1, p2, q1, q2));
5766                     inflag = (hb > 0) ? 1 : -1;
83e3             }
95ef             if (cross == 0 && hb < 0 && ha < 0) return R;
440c             bool t = cross == 0 && hb == 0 && ha == 0;
68f7             if (t ? (inflag == 1) : (cross >= 0) ? (ha <= 0) : (hb > 0)) {
8d2f                     if (inflag == -1) R.push_back(q2);
2740                     ba++; b++; b %= m;
74ba             } else {
2fa0                     if (inflag == 1) R.push_back(p2);
1d34                     aa++; a++; a %= n;
1d34
09a0             }
c759     }
c27d     if (inflag == 0) {
8d7f             if (in_poly(P[0], Q)) return P;
913d             if (in_poly(Q[0], P)) return Q;
9a7e     }
993f     R.erase(unique(all(R)), R.end());
35c1     if (R.size() > 1 && R.front() == R.back()) R.pop_back();
41e9     return R;
113c }
```

Polygon-Line Intersection

```
d41d //retorna tds trechos da reta que passa por s(s.a != s.b) coberta
d41d //por p(simple polygon)
d41d //tempo:O(n log n), memoria: O(n), onde n = p.size() TODO testar
abcb vector &lt;segment&gt; cobertura(segment s, polygon p) {
c8be     if (cmp(poly_area(p)) < 0) reverse(all(p)); //p deve esta no sentido anti-hr
5ec6     int n = (int)p.size();
```

```
5ec6
ed2a    polygon q;
a951    p.pb(p[0]), p.pb(p[1]), p.pb(p[2]);
592f    for (int i = 1; i <= n; i++) {
a782            point u = p[i], v = p[i+1];
73c9            int d0 = ccw(s.a,s.b,p[i-1]), d1 = ccw(s.a,s.b,p[i]);
3fc2            int d2 = ccw(s.a,s.b,p[i+1]), d3 = ccw(s.a,s.b,p[i+2]);
3fc2
3630            if (d1 == d2) continue;
3630
14b9            if (d1 * d2 == -1)
aab9                    q.pb(line_intersect(s.a, s.b, u, v));
02c0            else if (d1 == 0 && (d0*d2 == -1 || ccw(p[i-1], p[i], p[i+1]) > 0))
82d2                    q.pb(line_intersect(s.a, s.b, u, v));
6957            else if (d2 == 0 && ccw(p[i], p[i+1], p[i+2]) > 0 && d1*d3 >= 0)
e155                    q.pb(line_intersect(s.a, s.b, u, v));
84d6    }
c42e    sort(all(q));
d4d8    vector &lt;segment&gt; seg;
50e6    for (int i = 0; i < q.size(); i += 2) seg.pb( segment(q[i], q[i+1]));
50e6
fa88    return seg;
6c84 }
```

Polygon Triangulation

```
395e #define N_VERTEX 10000
80e5 int orelha[N_VERTEX], prox[N_VERTEX], prev[N_VERTEX];
80e5
80e5 // verifica se p[prev[id]~p[id]~p[prox[id]] forma uma orelha p
eef9 bool eh_orelha(polygon &p, int id) {
9db8    int n = p.size();
c0f0    point a = p[prev[id]], b = p[prox[id]];
a8dc    if (ccw(a, p[id], b) <= 0) return false;
a8dc
5a7a    for (int i = 0; i < n; i++) {
038b            int j = ((i+1<n)?(i+1):(i+1-n)); //j = (i+1)%n
2b13            if (i == prev[id] || i == prox[id] ||
1567                     j == prev[id] || j == prox[id]) continue;
c9eb            if (seg_intersect(p[i], p[j], a, b))
b43f                    return false;
4cf0    }
cac8    return true;
b1e3 }
b1e3 //Complexidade O(n^2)
b1e3 //assume q o poligono eh simples com 3 ou mais vertices, sem pontos repitidos
e82d void triangulacao(polygon &p) {
aea2    int n = p.size(), id = 0;
2911    if (cmp(poly_area(p)) < 0) reverse(all(p)); //deixa p no sentido anti-hor
2911
9b58    for (int i = 0; i < n; i++) {
e616            prev[i] = ((i==0)?(n-1):(i-1));
fd13            prox[i] = ((i+1<n)?(i+1):(i+1-n));
fd13
aa38            orelha[i] = eh_orelha(p, i);
0401    }
08d2    while (n > 3) {
229e            while (!orelha[id]) id = prox[id];
229e
229e            //triangulo p[id], p[prev[id]], p[prox[id]]
229e            //diagonal inserida => prev[id] <-> prox[id]
05eb            printf("%2d %2d %2d\n", id, prox[id], prev[id]);
05eb
0bed            int ant = prev[id], next = prox[id];
41b5            prox[ant] = next;
e844            prev[next] = ant;
a6df            orelha[ant] = eh_orelha(p, ant);
85d8            orelha[next] = eh_orelha(p, next);
c9f0            n--;
37ca            id = prox[id];
2202    }
2202    //triangulo p[id], p[prox[id]], p[prox[prox[id]]]
b944    printf("%2d %2d %2d\n", id, prox[id], prox[prox[id]]);
7fe3 }
```

Closest Pair

```
904d #define inf 1e20
35c2 #define N_PTS 300010
1db4 #define LOG_PTS 25
6b34 point X[N_PTS], Y[N_PTS], Yrl[LOG_PTS][N_PTS];
6b34
c8e3 inline bool cmpy(const point &a, const point &b) {
4705    return (cmp(a.y,b.y) < 0 || (cmp(a.y,b.y) == 0 && cmp(a.x,b.x) < 0));
6f5b }
6f5b
6f5b //Retorna o quadrado da menor distancia
31d9 inline double divide_conquer(int n, int cont, point X[], point Y[]) {
37f0    if (n <= 1) return inf;
dfc0    if (n == 2) return (X[0]-X[1]).mod2();
dfc0
6568    int left = n/2; int right = n-left;
f57e    int l = 0, r = left;
a0d9    point mid = (X[left-1] + X[left])/2;
264f    for (int i = 0; i < n; i++) {
8b4b            if (Y[i] < mid) Yrl[cont][l++] = Y[i];
4faa            else Yrl[cont][r++] = Y[i];
323f    }
1eb9    double resp = inf;
6fb0    resp = min(resp, divide_conquer(left, cont+1, X, Yrl[cont]));
c40a    resp = min(resp, divide_conquer(right, cont+1, X+left, Yrl[cont]+left));
c40a
f209    for (int i = 0; i < n; i++) {
5430            if (cmp(abs(Y[i].x-mid.x), resp) > 0) continue;
```

```
6e13            for (int j = max(0, i-8); j < i; j++)
bad4                    resp = min(resp, (Y[i]-Y[j]).mod2());
3bfb    }
0ccc    return resp;
9b71 }
9b71
9b71
ce21 double closest_pair(vector <point> &p) {
cb73    for (int i = p.size()-1; i >= 0; i--) X[i] = Y[i] = p[i];
88d6    sort(X, X+p.size());
fffe    sort(Y, Y+p.size(), cmpy);
3ad4    return sqrt(divide_conquer(p.size(), 0, &X[0], &Y[0]));
9554 }
```

Crosses Half-Plane

```
d41d // Retorna a interseccao de um poligono simples com um semiplano
d41d // TODO cuidado qdo o poligono nao for convexo
d41d // sumpoem q o semiplano eh o lado esquerdo, indo de p1 para p2
d41d // p pode estar em qlqr direcao( poligono de retorno esta no msm sentido que p)
3755 polygon halfplane(polygon& p, line& semiplano) {
c316    polygon q;
3b60    point p1 = semiplano.first, p2 = semiplano.second;
3b60
3b60    // Sequencia poligono convexo exaustiva, para determinar se o semiplano.
4981    int n = p.size();
f194    for (int i = 0; i < n; i++) {
9a0e            double c = (p2-p1) % (p[i]-p1);
88e0            double d = (p2-p1) % (p[(i+1)%n]-p1);
7689            if (cmp(c) >= 0) q.push_back(p[i]);
9df1            if (cmp(c * d) < 0)
1d83                    q.push_back(line_intersect(p1, p2, p[i], p[(i+1)%n]));
c519    }
1645    return q;
9943 }
```

N segments Intersection

```
67e3 typedef ll ptype;
0bfd struct segment {
dfe5    point a, b;
fb83    segment(point a=point(0,0), point b=point(0,0)): a(a), b(b) {}
72c2    double interpolate(ptype x) {
7e34            if (a.x == b.x) return min(a.y, b.y);
08e1        return a.y + (double)(b.y - a.y) / (b.x - a.x) * (x - a.x);
5f79    }
8abb };
8abb
fd2a struct event {
a86d    ptype x;
f8b1    int tp, id;
56d0    event(ptype x=0, int tp=0, int id=0): x(x), tp(tp), id(id) {}
d428    bool operator<(const event &e) const {
0551            return x < e.x || (x == e.x && tp > e.tp);
5ae4    }
c7e3 };
c7e3
60c0 vector&lt;segment&gt; T;
dc88 struct CMP {
c5a6    static ptype x;
bc5e    bool operator()(const int &i, const int &j) {
1a89            return T[i].interpolate(x) < T[j].interpolate(x) - eps;
276a     }
e3c1 };
49a9 ptype CMP::x = 0;
ab9c set<int, CMP> S;
ab9c
083a #define auto set<int>::iterator
8737 auto prev(auto it) {return it == S.begin() ? S.end() : --it;}
e3b4 auto next(auto it) {return it == S.end() ? S.end() : ++it;}
1985 bool null(auto it) {return it == S.end();}
1985
4fcf bool intersect(const int &i, const int &j) {
fee1    return seg_intersect(T[i].a, T[i].b, T[j].a, T[j].b);
12fd }
12fd
6be6 pii segment_intersect (const vector&lt;segment&gt; &p) {
7136    T = p;
f399    int n = (int) T.size();
9c01    vector<event> e;
b204    f(i, 0, n) {
f7e9            e.push_back (event (min (T[i].a.x, T[i].b.x), +1, i));
4839            e.push_back (event (max (T[i].a.x, T[i].b.x), -1, i));
f1b8    }
bec0    sort(all(e));
bec0
3058    S.clear();
66a2     f(i, 0, e.size()) {
d61e        CMP::x = e[i].x;
323b        int& id = e[i].id;
323b
38f1        if (e[i].tp == +1) { // insert segment
cf62            auto nid = S.lower_bound(id);
08b9            auto pid = prev(nid);
68c6            if (!null(nid) && intersect(*nid, id)) return mp((*nid), id);
ae7e            if (!null(pid) && intersect(*pid, id)) return mp((*pid), id);
a7da            S.insert(nid, id);
15b5        }
c496        else { // remove segment
9bf4            auto cid = S.lower_bound(id);
2465            auto pid = prev(cid), nid = next(cid);
af6f            if (!null(pid) && !null(nid) && intersect(*pid, *nid))
db87                return mp((*pid), (*nid));
3472            S.erase(cid);
07d6        }
```

```
c2b5     }
85a3    return mp(-1,-1);
5edd }
```

Circle Structure

```
aa81 struct circle {
5235    point c; double r;
0ba6    circle(point c = point(0,0), double r=0.0):c(c), r(r) {}
639a    bool inside(point &e) { return cmp((e-c).mod(), r) <= 0; }
9757 };
```

Circle Intersection

```
d41d //assume que ha pelo menos 1 ponto de interseccao
d41d //cuidado com circulos iguais, sem intersecao (um dentro do outro)
9b67 pair<point, point> interseccao(circle a, circle b) {
4979    if (cmp(a.r, b.r) < 0) swap(a, b);
99d7    double R = a.r, r = b.r, d = (b.c-a.c).mod();
ee39    double x1 = (R*R - r*r + d*d) / (2*d);
9881    double h = 0.0;
f8cf    if (cmp(R*R - x1*x1) > 0) h = sqrt(R*R - x1*x1);
2cb8    point v = ((b.c-a.c)/d) * R;
48ac    return mp(a.c + v.rotate(h/R, x1/R), a.c + v.rotate(-h/R, x1/R));
ee26 }
```

Circle Union

```
f244 void increment(double ini, double fim, double &Area, double &Perim, circle &q) {
5456     double teta = (cmp(fim,ini)>=0) ? (2*Pi-fim+ini) : (ini-fim);
7959     Perim += teta * q.r;
6dde     Area += 0.5*teta*quad(q.r) - 0.5*sin(teta)*quad(q.r);
f25f     point a = (point(cos(fim), sin(fim)) * q.r) + q.c;
557d     point b = (point(cos(ini), sin(ini)) * q.r) + q.c;
7f51     Area += 0.5*(a % b);
e7cf }
e7cf //Calcula a area e o perimetro em O(n^2*logn)
e7cf //Consome memoria O(n)
b0a6 double area_uniao_circle(vector <circle> &T) {
f354     vector <circle> p;
ee7b     vector < pair<double, double> > seg;
554f     point e, a, b;
2ff7     double Area = 0, Perim = 0, teta, ini, fim;
2ff7
2ff7     //remove circles repitidos, com area nula, ou circle dentro de outro circle
c3ee     f(i, 0, T.size()) {
2fcf         bool check = cmp(T[i].r) > 0;
ff4a         for (int j = 0; j < T.size() && check; j++) {
80cd             if (T[i]==T[j]) check = (i>=j);
fe16             else if (cmp((T[i].c-T[j].c).mod()+T[i].r, T[j].r) <= 0)
2b8c                 check = false;
081c         }
d825         if (check) p.pb(T[i]);
88a4     }
88a4
4d74     for (int i = 0; i < p.size(); i++) {
7e41         seg.clear();
8b21         for (int j = 0; j < p.size(); j++) if (i != j) {
8b21             //p[i] e p[j] nao tem 2 pontos em comum
ae3c             if ((p[i].c-p[j].c).mod() > p[i].r + p[j].r -eps) continue;
ae3c
a1cf             pair <point, point> inter = interseccao2(p[i], p[j]);
8514             teta = angle(inter.first, p[i].c, inter.second);
9d66             if (teta > eps) {
1354                 e = (inter.first - p[i].c).rotate(teta/2) + p[i].c;
dbe0                 if (p[j].inside(e) == false) swap( inter.first, inter.second);
607c             }
b7d1             else {
7470                 e = (inter.first - p[i].c).rotate((2*Pi+teta)/2) + p[i].c;
0581                 if (p[j].inside(e) == false) swap( inter.first, inter.second);
029f             }
029f
35e2             ini = (inter.first - p[i].c).polar();
59b4             fim = (inter.second - p[i].c).polar();
59b4
3298             a = (point(cos(ini), sin(ini)) * p[i].r) + p[i].c;
4514             b = (point(cos(fim), sin(fim)) * p[i].r) + p[i].c;
4514
65d8             if (ini > fim) {
f064                 seg.pb( mp(ini, 2*Pi));
64e4                 seg.pb( mp(0, fim));
7453             }
d4d3             else seg.pb( mp(ini, fim));
a2ca         }
a489         sort(all(seg));
7592         if ((int) seg.size() == 0) {
13e7             Perim += 2*Pi*p[i].r;
74ac             Area += Pi*quad(p[i].r);
bf78             continue;
240b         }
2bf4         fim = seg[0].second;
6d26         for (int j = 1; j < seg.size(); j++) {
accc             if (cmp(fim, seg[j].first) < 0) {
4654                 increment(seg[j].first, fim, Area, Perim, p[i]);
79eb             }
8c91             fim = max(fim, seg[j].second);
dfa2         }
b486         increment(seg[0].first, fim, Area, Perim, p[i]);
6444     }
8849     return Area;
08f5 }
```

Minimum Spanning Circle

```
d41d //Calcula o circuncentro de um triangulo
22e1 point circumcenter(point p, point q, point r) {
3732    point a = p-r, b = q-r, c = point(a*(p + r) / 2, b*(q + r) / 2);
0f79    return point(c % point(a.y, b.y), point(a.x, b.x) % c) / (a % b);
0250 }
0250
c95d circle spanning_circle(vector<point>& T) {
7645    int n = T.size();
018e    random_shuffle(all(T));
cc09    circle C(point(), -INFINITY);
a542    for (int i = 0; i < n; i++) if (!C.inside(T[i])) {
e1c1            C = circle(T[i], 0);
335f            for (int j = 0; j < i; j++) if (!C.inside(T[j])) {
069f                    C = circle((T[i] + T[j]) / 2, abs(T[i] - T[j]) / 2);
5c05                    for (int k = 0; k < j; k++) if (!C.inside(T[k])) {
6abf                            point o = circumcenter(T[i], T[j], T[k]);
a55e                            C = circle(o, abs(o - T[k]));
9d60                    }
50bc            }
a82f    }
09a6    return C;
49db }
```

Circle-Polygon Intersection

```
d41d //area da intersecao de um triangulo a,b,S.c com circulo S
6673 double area(point a, point b, circle S) {
6421    double aa=(S.c-a).mod(), bb=(S.c-b).mod(), cc=seg_distance(a, b, S.c);
91ca    if (cmp(aa,S.r) <= 0 && cmp(bb,S.r) <= 0) return 0.5*fabs((a-S.c)%(b-S.c));
a8b9    if (cmp(cc,S.r) >= 0) return 0.5*fabs(S.r * S.r * angle(a, S.c, b));
a8b9
3da3    if (cmp(aa, bb) > 0) {swap(a, b); swap(aa,bb); }
eadd    double A=(a-b).mod2(), B=2*((a-b)*(b-S.c)), C=(b-S.c).mod2()-S.r*S.r;
ffb6    double t = ((cmp(B*B-4*A*C)==0)?0.0: sqrt(B*B-4*A*C));
726d    double x1 = 0.5*(-B-t)/A, x2 = 0.5*(-B+t)/A;
6c18    point p1 = a*x1 + b*(1-x1), p2 = a*x2 + b*(1-x2);
6c18
5fcf    if (cmp(aa, S.r) < 0) return area(a, p1, S) + area(p1, b, S);
18c2    return area(a, p2, S) + area(p2, p1, S) + area(p1, b, S);
7c63 }
7c63
7c63 //area da intersecao de poligono(simples qlqr) com circulo S O(n)
64ab double area(polygon &T, circle S) {
643c    double ans=0.0;
2a56    int n = (int)T.size();
65d0    for (int i = 0; i < n; i++)
ef35            ans += (area(T[i], T[(i+1)%n], S) * ccw(T[i], T[(i+1)%n], S.c));
6b33    return fabs(ans);
bd68 }
```

6 Geometry 2D - Integer

Geometry Integer

```
bd39 bool seg_intersect1d (ll l1, ll r1, ll l2, ll r2) {
5fd8    if (l1 > r1) swap(l1, r1);
4e39    if (l2 > r2) swap(l2, r2);
15ac    return max(l1, l2) <= min(r1, r2);
d57a }
d57a
d57a //Decide se os segmentos fechados pq e rs tem pontos em comum
ee01 bool seg_intersect(point p, point q, point r, point s) {
3992    return seg_intersect1d(p.x, q.x, r.x, s.x)
e388            && seg_intersect1d(p.y, q.y, r.y, s.y)
2acd            && ccw(p, q, r) * ccw(p, q, s) <= 0
5180            && ccw(r, s, p) * ccw(r, s, q) <= 0;
4842 }
4842
4842 ////////////////////////////////////////////////////////////////////////////
4842 // Classifica o ponto p em relacao ao poligono T.
4842 //
4842 // Retorna 0, -1 ou 1 dependendo se p esta no exterior, na fronteira
4842 // ou no interior de T, respectivamente.
8390 int in_poly(point p, polygon& T) {
2696    int n = T.size(), cnt = 0;
17f8    for (int i = 0, j = n-1; i < n; j = i++) {
3065            if (between(T[i], p, T[j])) return -1;
6ba3            if ((T[i].y > p.y) != (T[j].y > p.y)) {
786a                    cnt ^= (ccw(p, T[j], T[i])*(T[j].y-T[i].y) > 0);
198d            }
38bc    }
78eb    return cnt;
a580 }
a580
a580
a580 //TESTAR XXX
a580 ////////////////////////////////////////////////////////////////////////////
a580 // Returns a list of points on the convex hull in counter-clockwise order.
a580 // Note: the last point in the returned list is the same as the first one.
a580 // Se T.size()==1 => H.size()==1
a580 //trocar ccw(H[k-1], T[i], H[k-2]) <= 0 por ccw(...) < 0,
a580 //para incluir pts colineares
a580 // Monotone hull O(nlogn)
ebb2 polygon convex_hull(vector<point> T) {
18f8    int n = T.size(), k = 0;
c585    polygon H(2*n);
c585
0d3d    sort(T.begin(), T.end());
0d3d    // Build lower hull
```

```
8045     for (int i = 0; i < n; i++) {
50b0             while (k >= 2 && ccw(H[k-1], T[i], H[k-2]) <= 0) k--;
8c5e             H[k++] = T[i];
608e     }
608e     // Build upper hull
0201     for (int i = n-2, t = k+1; i >= 0; i--) {
1007             while (k >= t && ccw(H[k-1], T[i], H[k-2]) <= 0) k--;
b5d8             H[k++] = T[i];
9df1     }
94cb     H.resize(k);
7a34     return H;
2e5b }
```

7 Geometry 3D

Geometry 3D

```
a616 #define vetor point
a616
a616 // FORMULAS.
a616 //  vetores a,b; a*b = a.mod()*b.mod()*cos( angulo entre a e b) =>
a616 // a*b = |a|*|b|*cos(t)
a616 //  vetores a,b; (a^b).mod() = a.mod()*b.mod()*sin( angulo entre a e b)
a616
45c8 inline int cmp(ld x, ld y = 0, ld tol = eps) {
2939    return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
426f }
858c struct point {
148a    ld x, y, z;
a276    point(ld x = 0, ld y = 0, ld z = 0): x(x), y(y), z(z) {}
78a6    point operator +(point q) { return point(x + q.x, y + q.y, z + q.z); }
d092    point operator -(point q) { return point(x - q.x, y - q.y, z - q.z); }
55b5    point operator *(ld t) { return point(x * t, y * t, z * t); }
0646    point operator /(ld t) { return point(x / t, y / t, z / t); }
0286    point operator ^(point q) {
b5e0            return point(y*q.z - z*q.y, z*q.x - x*q.z, x*q.y - y*q.x); }
10a6    ld operator *(point q) { return x * q.x + y * q.y + z * q.z; }
44b7    ld mod() { return sqrt(x * x + y * y + z * z); }
8344    ld mod2() { return x * x + y * y + z * z; }
feaf    point projecao(vetor u) { return (*this) * ((*this)*u) / ((*this)*(*this)); }
feaf
5eb8    int cmp(point q) const {
ff5c            if (int t = ::cmp(x, q.x)) return t;
82ce            if (int t = ::cmp(y, q.y)) return t;
d86a            return ::cmp(z, q.z);
940d    }
9b26    bool operator ==(point q) const { return cmp(q) == 0; }
b245    bool operator !=(point q) const { return cmp(q) != 0; }
ce46    bool operator < (point q) const { return cmp(q) < 0; }
bb0f };
bb0f
bb0f // RETAS, SEMIRETAS, SEGMENTOS E TRIANGULOS
e08c struct reta {
d6d7    point a, b;//  <--a---b--->
4b8c    reta(point A=point(0,0,0), point B=point(0,0,0)): a(A), b(B) { }
4b8c
4b8c    //verifica se o ponto p esta na reta ab
7880    bool belongs(point p) {
44f8            return cmp(((a-p)^(b-p)).mod()) == 0;
b5fd    }
4442 };
6e35 struct semireta {
479c    point a, b; //  |a---b--->
2e96    semireta(point A=point(0,0,0), point B=point(0,0,0)): a(A), b(B) { }
bc3c };
79a8 struct segmento {
3de6    point a, b; //  |a---b|
de90    segmento(point A=point(0,0,0), point B=point(0,0,0)): a(A), b(B) { }
df01    bool between(point p) {
7581            return cmp(((a-p)^(b-p)).mod()) == 0 && cmp((a-p) * (b-p)) <= 0;
b6cc    }
3bfe };
b44d struct triangulo {
af47    point a, b, c;
f643    triangulo(point A, point B, point C): a(A), b(B), c(C) { }
6b62    ld area() { return 0.5*((b-a)^(c-a)).mod(); }
6b62
6b62    //retorna o ponto que eh a projecao de p no plano abc
1780    point projecao(point p) {
f9f6            vetor w = (b-a)^(c-a);
1f9f            return p - w.projecao(p-a);
2b63    }
2b63    //verifica se p esta dentro de abc
2b63    // se retornar true, entao a,b,c,p sao coplanares
1ba8    bool inside(point p) {
ef7a            return cmp(((p-a)^(b-a)).mod() +
dc8e                                    ((p-b)^(c-b)).mod() +
f81e                                    ((p-c)^(a-c)).mod() -
6d1e                                    ((b-a)^(c-a)).mod()) == 0;
0366    }
1ea3 };
1ea3
1ea3 //Produto misto
a2a0 ld produto_misto(point p, point q, point r) {
56b2    return (p^q)*r;
8db1 }
8db1 //Volume do tetraedro pqrs
059b ld volume(point p, point q, point r, point s) {
e035    return fabs(produto_misto(q-p, r-p, s-p)) / 6.0;
11d5 }
11d5
11d5 // DISTANCIA ENTRE OBJETOS GEOMETRICOS
4985 ld distancia(point p, reta r) {
ee85    vetor v = r.b-r.a, w = p-r.a;
2f7c    return (v^w).mod() / v.mod();
4a60 }
8dea ld distancia(point p, semireta s) {
```

```
3c72     vetor v = s.b-s.a, w = p-s.a;
341c     if (cmp(v*w) <= 0) return (p-s.a).mod();
b99b     return (v^w).mod() / v.mod();
1027 }
cdef ld distancia(point p, segmento s) {
7700     point proj = s.a + (s.b-s.a).projecao(p-s.a);
cb69     if (segmento(s.a,s.b).between(proj))
8d5f             return (p-proj).mod();
ad83     return min((p-s.a).mod(), (p-s.b).mod());
55c4 }
1268 ld distancia(point p, triangulo T) {
a756     point proj = T.projecao(p);
c71d     if (T.inside(proj)) return (p-proj).mod();
0e8c     return min(     distancia(p, segmento(T.a, T.b)),
c3f9                             min(distancia(p, segmento(T.b, T.c)),
cf4b                                     distancia(p, segmento(T.c, T.a))));
4622 }
3f36 ld distancia(reta r, reta s) {
65cd     vetor u = r.b-r.a, v = s.b-s.a, w = s.a-r.a;
4b58     ld a = u*u, b = u*v, c = v*v, d = u*w, e = v*w;
4a12     ld D = a*c - b*b, sc, tc;
23fb     if (D < eps) {
6ee7             sc = 0;
b72c             tc = (b > c) ? d/b : e/c;
cce1     } else {
bd44             sc = (b*e - c*d) / D;
9c8d             tc = (a*e - b*d) / D;
5d82     }
d414     vetor dP = w + (u * sc) - (v * tc);
4733     return dP.mod();
f55a }
8eae ld distancia(segmento X, segmento Y) {
e75f     point p = X.a, q = X.b;
397b     point r = Y.a, s = Y.b;
0da3     if (p == q) return distancia(p, Y);
4849     if (r == s) return distancia(r, X);
1f96     if (cmp(((p-q)^(s-r)).mod()) == 0)
9028             return min( min(distancia(p,Y),distancia(q,Y)),
cf88                                     min(distancia(p,Y),distancia(q,Y)));
6402     vetor v = q-p, u = s-r, t = (r-p);
185c     ld b = ((t*v)*(v*u) - (t*u)*(v*v)) / ((u*u)*(v*v) - (u*v)*(v*u));
b645     ld a = (b*(u*v) + t*v) / (v*v);
4231     if (cmp(a) >= 0 && cmp(a,1.0) <= 0 && cmp(b) >= 0 && cmp(b,1.0) <= 0)
ff9e             return ((p+v*a) - (r+u*b)).mod();
39c6     point ini = ((cmp(a) < 0)?p:q);
78e4     point fim = ((cmp(b) < 0)?r:s);
5fdb     return (ini-fim).mod();
320d }
320d
320d //Calcula o centro da esfera circunscrita de uma piramide triangular
fd93 point circumsphere(point p, point q, point r, point s) {
6645     point a = q-p, b = r-p, c = s-p;
c16b     return p + ((a^b)*c.mod2() + (c^a)*b.mod2() + (b^c)*a.mod2()) / (a*(b^c)*2);
39fe }
39fe
39fe //Calcula o circuncentro de um triangulo no espaco
c9a7 point circumcenter(point p, point q, point r) {
1653     point a = (q-p)^((q-p)^(r-p)), b = (r-p)^((q-p)^(r-p)); ld t;
7118     if (fabs(a.x) < eps) t = (r.x-q.x)/2/b.x;
a779     else if (fabs(a.y) < eps) t = (r.y-q.y)/2/b.y;
05bf     else if (fabs(a.z) < eps) t = (r.z-q.z)/2/b.z;
26be     else {
e1e1             t = a.x*(r.y-q.y) - a.y*(r.x-q.x);
d5fe             t = t / (2*a.y*b.x - 2*a.x*b.y);
4a16     }
fc14     return (p+q)/2 + a*t;
9023 }
9023
9023 //Verifica se T[a], T[b], T[c] eh face do convex hull
9023 //OBS.: Cuidado com mais de 3 pontos coplanares
1da8 bool ishullface(vector <point> &T, int a, int b, int c) {//TODO testar
4015     int n = (int)T.size(), pos = 0, neg = 0;
9d39     for (int i = 0; i < n; i++) {
814e             ld pm = produto_misto(T[b]-T[a], T[c]-T[a], T[i]-T[a]);
1e07             if (cmp(pm) < 0) neg++;
89c3             if (cmp(pm) > 0) pos++;
b08d     }
3002     return (neg*pos == 0);
9d7d }
```

8 Graph

Erdős–Gallai theorem

A sequence of non-negative integers $d_1 \geq \cdots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + \cdots + d_n$ is even and

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$$

holds for $1 \leq k \leq n$.

Stable Marriage

```
a511 #define maxn 512
a511
5e20 int W[maxn][maxn], M[maxn][maxn];
68c0 int quem[maxn]; //quem[i] = homem q a mulher i casa
6589 int ja[maxn]; //ja[i] = mulher que o cara i casa
2baa int topo[maxn];
7fc7 int n;
7fc7
fc77 bool prefere (int m, int novo, int velho){
802c      for (int i = 0; i < n; i++){
07a0          if (W[m][i] == novo) return true;
```

```
13ac         if (W[m][i] == velho) return false;
b334     }
4aa7 }
4aa7
4aa7
b9be void solve (){
bb77    bool flag = true;
1b4d    while (1){
07f4            flag = true;
2b4e            for (int i = 0; i < n; i++) if (ja[i]==-1){
6ad6                    flag = false;
b929                    int m = M[i][topo[i]++];
45ef                    if (quem[m] == -1){
0227                            quem[m] = i; ja[i] = m;
203f                    }
feff                    else if (prefere (m, i, quem[m])){
3321                            ja[quem[m]] = -1, quem[m] = i; ja[i] = m;
3a7c                    }
2021             }
78d3             if (flag) break;
e453     }
5e2b }
5e2b
17bb int main (){
a3d0     int t; scanf("%d", &t);
cd6d     int aux;
6172     while (t--){
feda         scanf("%d", &n);
19d4         for (int i = 0; i < n; i++) {
52db            scanf("%d", &aux);
a790             for (int j = 0; j < n; j++){
eb1d                 scanf("%d", &M[i][j]), M[i][j]--;
2bab              }
e7ef         }
3296         for (int i = 0; i < n; i++){
153e            scanf("%d", &aux);
b9ad             for (int j = 0; j < n; j++){
d607                 scanf("%d", &W[i][j]), W[i][j]--;
cda2             }
2213         }
0a5d         memset (quem, -1, sizeof (quem));
5c21         memset (topo, 0, sizeof (topo));
26cb         memset (ja, -1, sizeof (ja));
b19c         solve();
7b3f         for (int i = 0; i < n; i++) printf("%d %d\n", i+1, quem[i]+1);
5840     }
5840
dec7     return 0;
f516 }
```

Dinic maxflow

```
d41d // Dinic maxflow, min(O(EV^2),O(maxflow*E)(?)) worst case
d41d // O(E*min(V^2/3,sqrt(E))) for unit caps (O(E*sqrt(V)) if bipartite)
d41d
469e typedef ll FTYPE;      // define as needed
469e
1adb const int MAXV = ; // maximo numero de vertices
1b9b const FTYPE FINF = LINF; // infinite flow
1b9b
3922 struct Edge {
b25e     int to;
7173     FTYPE cap;
47ea     Edge(int t, FTYPE c) { to = t; cap = c; }
d4ff };
d4ff
bcdd vector<int> adj[MAXV];
5509 vector<Edge> edge;
dbcd int ptr[MAXV], dinic_dist[MAXV];
dbcd
dbcd // Inserts an edge u->v with capacity c
6358 inline void add_edge(int u,int v,FTYPE c) {
0a1b     adj[u].push_back(edge.size());
a469     edge.push_back(Edge(v,c));
2e59     adj[v].push_back(edge.size());
6571     edge.push_back(Edge(u,0)); // modify to Edge(u,c) if graph is non-directed
66ca }
66ca
31a3 bool dinic_bfs(int _s,int _t) {
c9a4     memset(dinic_dist,-1,sizeof(dinic_dist));
fac6     dinic_dist[_s] = 0;
ecd3     queue<int> q;
cc20     q.push(_s);
5524     while(!q.empty() && dinic_dist[_t] == -1) {
b2fc         int v = q.front();
3a8a         q.pop();
1e8a         for(size_t a=0;a<adj[v].size();++a) {
c984             int ind = adj[v][a];
2aa0             int nxt = edge[ind].to;
8b60             if(dinic_dist[nxt] == -1 && edge[ind].cap) {
3c55                 dinic_dist[nxt] = dinic_dist[v] + 1;
3322                 q.push(nxt);
9840             }
eb49         }
febd     }
c28a     return dinic_dist[_t] != -1;
0f3a }
0f3a
dfab FTYPE dinic_dfs(int v,int _t,FTYPE flow) {
bd70     if(v == _t) return flow;
4b3f     for(int &a = ptr[v];a<(int)adj[v].size();++a) {
31a5         int ind = adj[v][a];
6de4         int nxt = edge[ind].to;
1b53         if(dinic_dist[nxt] == dinic_dist[v] + 1 && edge[ind].cap) {
b77c             FTYPE got = dinic_dfs(nxt,_t,min(flow,edge[ind].cap));
```

```
b722            if(got) {
c013                edge[ind].cap -= got;
d24b                edge[ind^1].cap += got;
8152                return got;
ba4c            }
0545        }
d9ee    }
bc36    return 0;
c298 }
c298
1110 FTYPE dinic(int _s,int _t) {
f3b1    FTYPE ret = 0, got;
351d    while(dinic_bfs(_s,_t)) {
49e8        memset(ptr,0,sizeof(ptr));
3d97        while((got = dinic_dfs(_s,_t,FINF))) ret += got;
c875    }
da59    return ret;
aeed }
aeed
aeed // Clears dinic structure
46d7 inline void dinic_clear() {
0db6    for(int a=0;a<MAXV;++a) adj[a].clear();
e3fc    edge.clear();
38fc }
```

Min-cost maxflow

```
d41d // Min-cost Max-flow ( O(V*E + V^2*MAXFLOW) )
d41d
d97f typedef int FTYPE; // type of flow
e6a7 typedef int CTYPE; // type of cost
4551 typedef pair<FTYPE,CTYPE> pfc; // pair<flow,cost>
4551
1c31 const int MAXV = ; // maximum number of vertices
54b0 const CTYPE CINF = INF; // infinite cost
b6ba const FTYPE FINF = INF; // infinite flow
b6ba
b844 void operator+=(pfc &p1,pfc &p2) { p1.first+=p2.first; p1.second+=p2.second; }
b844
ba37 struct Edge {
09c7    int to;
45f8    FTYPE cap;
4417    CTYPE cost;
2771    Edge(int a,FTYPE cp,CTYPE ct) { to = a; cap = cp; cost = ct; }
7093 };
7093
96f2 vector<int> adj[MAXV];
5234 vector<Edge> edge;
c32a int V; // number of vertices (don't forget to set!)
c32a
c32a // Inserts an edge u->v with capacity c and cost cst
4864 inline void add_edge(int u,int v,FTYPE c,CTYPE cst) {
c4cf    adj[u].push_back(edge.size());
aa3f    edge.push_back(Edge(v,c,cst));
787c    adj[v].push_back(edge.size());
f5c6    edge.push_back(Edge(u,0,-cst));
d0d9 }
d0d9
a5df FTYPE flow[MAXV];
0045 CTYPE dist[MAXV], pot[MAXV];
bac0 int prv[MAXV], e_ind[MAXV];
ded6 bool foi[MAXV];
ded6
f9d1 void bellman_ford(int _s) {
f123    for(int a=0;a<V;++a) dist[a] = CINF;
7647    dist[_s] = 0;
6410    for(int st=0;st<V;++st) {
1719        for(int v=0;v<V;++v) {
d96c            for(size_t a=0;a<adj[v].size();++a) {
ea3c                int ind = adj[v][a];
4429                int nxt = edge[ind].to;
6092                if(!edge[ind].cap) continue;
cec9                dist[nxt] = min(dist[nxt],dist[v] + edge[ind].cost);
2f1a            }
411c        }
0098    }
3046 }
3046
6793 pfc dijkstra(int _s,int _t) { // O(V^2)
405b    for(int a=0;a<V;++a) {
e94d        dist[a] = CINF;
1f2b        foi[a] = 0;
8d6c    }
ec7b    dist[_s] = 0;
6d38    flow[_s] = FINF;
8599    while(1) {
30a8        int v;
5370        CTYPE d = CINF;
ce7a        for(int a=0;a<V;++a) {
9ada            if(foi[a] || dist[a] >= d) continue;
8afc            d = dist[a];
3b54            v = a;
9de4        }
610c        if(d == CINF) break;
1a34        foi[v] = 1;
31f8        for(size_t a=0;a<adj[v].size();++a) {
71fb            int ind = adj[v][a];
e26d            int nxt = edge[ind].to;
3669            if(!edge[ind].cap || dist[nxt] <= dist[v] +
07af               edge[ind].cost + pot[v] - pot[nxt]) continue;
5228            dist[nxt] = dist[v] + edge[ind].cost + pot[v] - pot[nxt];
f290            prv[nxt] = v;
2565            e_ind[nxt] = ind;
0577            flow[nxt] = min(flow[v],edge[ind].cap);
6aaf        }
43c0    }
c20c    if(dist[_t] == CINF) return pfc(FINF,CINF);
```

```
efac     for(int a=0;a<V;++a) pot[a] += dist[a];
de63     pfc ret(flow[_t],0);
9208     for(int cur = _t; cur != _s; cur = prv[cur]) {
e8c6         int ind = e_ind[cur];
3e32         edge[ind].cap -= flow[_t];
5464         edge[ind^1].cap += flow[_t];
f53a         ret.second += flow[_t] * edge[ind].cost; // careful with overflow!
1c33     }
3620     return ret;
fc6b }
fc6b
fc6b // Returns a pair (max-flow, min-cost)
c9d9 pfc mcmf(int _s,int _t) {
b783     pfc ret(0,0), got;
24bc     bellman_ford(_s);
4f45     for(int a=0;a<V;++a) pot[a] = dist[a];
8e0a     while( (got = dijkstra(_s,_t)).first != FINF ) ret += got;
0045     return ret;
0241 }
0241
0241 // Clears mcmf structure
8377 inline void mcmf_clear() {
e477     edge.clear();
1245     for(int a=0;a<V;++a) adj[a].clear();
8892 }
```

Finding one solution for 2SAT

1. Construct implication graph and contract it (it will be a skew-symmetric DAG)
2. Consider the vertices in topsort order. If it was not assigned a value, set it's value to false. This causes all of the terms in the complementary component to be set to true.

Due to the topological ordering, when a term x is set to false, all terms that lead to it via a chain of implications will themselves already have been set to false. Symmetrically, when a term is set to true, all terms that can be reached from it via a chain of implications will already have been set to true. Therefore, the truth assignment constructed by this procedure satisfies the given formula.

Stoer-Wagner algorithm for global mincut

```
d41d //   Algoritmo de Stoer-Wagner
d41d //   Encontra o corte minimo em um grafo
d41d //   Complexidade: O(N^3)
d41d
431d const int INF = 0x3f3f3f3f;
abbd const int MAXN = 110;
abbd
29e0 int adj[MAXN][MAXN], cost[MAXN][MAXN];
fb4f int deg[MAXN], sum[MAXN], ord[MAXN];
d93b bool foi[MAXN], out[MAXN];
3a33 int n,m;
3a33
b8b0 int StoerWagner() {
b942     int ret = INF;
3405     for(int siz=n;siz>1;--siz) {
26ba         for(int a=0;a<n;++a) {
7a96             sum[a] = 0;
7113             foi[a] = out[a];
5bae         }
daae         int i=0;
d6f7         while(1) {
13a0             int v,d=-1;
9ed3             for(int a=0;a<n;++a) {
d5a2                 if(!foi[a] && sum[a] > d) {
ea58                     d = sum[a];
4484                     v = a;
10ae                 }
296a             }
be90             if(d==-1) break;
49f4             foi[v] = 1;
5dac             ord[i++] = v;
7cc7             for(int a=0;a<deg[v];++a) {
a55e                 int nxt = adj[v][a];
563c                 if(!foi[nxt]) sum[nxt] += cost[v][nxt];
eea1             }
89ee         }
8607         int s = ord[siz-2], t = ord[siz-1];
10c1         ret = min(ret, sum[t]);
4d8d         out[t] = 1;
679e         for(int a=0;a<deg[t];++a) {
6290             int nxt = adj[t][a];
06ef             if(s==nxt) continue;
f771             if(cost[s][nxt]==0) {
1773                 adj[s][deg[s]++] = nxt;
94fc                 adj[nxt][deg[nxt]++] = s;
5c51             }
4c12             cost[s][nxt] += cost[t][nxt];
51ce             cost[nxt][s] += cost[nxt][t];
6fcb         }
d8c9     }
30d8     return ret;
e9e7 }
```

Maximum matching in generic graph

Edmonds' algorithm: $O(n^3)$

```
d41d // Edmonds' Blossom Algorithm O(N^3)
d41d // Finds maximum matching in generic graphs
d41d
50ac const int MAXN = ;     // maximo numero de vertices
50ac
5285 int n;                 // numero de vertices
9124 vector<int> adj[MAXN]; // lista de adj
```

```
2a7d int match[MAXN];       // match[i] eh o par de i. -1 se nao tem par
fe16 int p[MAXN], base[MAXN] , q[MAXN];
cd46 bool used[MAXN], blossom[MAXN];
cd46
3229 int lca ( int a, int b ) {
e3c9     bool used [MAXN] = {0};
0c59     for ( ;; ) {
eb63         a = base [a];
a684         used [a] = true;
1ad4         if ( match[a] == -1 )  break;
8cdd         a = p[match[a]];
79f3     }
1750     for ( ;; ) {
0912         b = base[b];
1b7d         if ( used[b] )  return b;
a0f5         b = p[match[b]];
0a76     }
c88a }
c88a
4072 void mark_path ( int v, int b, int children ) {
a48d     while ( base[v] != b ) {
f79f         blossom[base[v]] = blossom[base[match[v]]] = true;
3c8d         p[v] = children;
5ab4         children = match[v];
efc3         v = p[match[v]];
0ac9     }
d657 }
d657
d969 int find_path ( int root ) {
c0ba     memset ( used, 0 , sizeof used );
2939     memset ( p, - 1 , sizeof p );
7022     for ( int i = 0 ; i < n ; ++i )
450a         base[i] = i;
b9c4     used[root] = true;
044c     int qh = 0 , qt = 0;
6cd2     q[qt++] = root;
b731     while ( qh < qt ) {
7ab6         int v = q[qh++];
e2fd         for ( size_t i = 0; i < adj[v].size(); ++i ) {
8a3c             int to = adj[v][i];
0cab             if ( base[v] == base[to] || match[v] == to )  continue;
ab27             if ( to == root || match[to] != -1 && p[match[to]] != -1 ) {
95e1                 int curbase = lca ( v, to );
680e                 memset ( blossom, 0 , sizeof blossom );
7930                 mark_path ( v, curbase, to );
e755                 mark_path ( to, curbase, v );
7d5b                 for ( int i = 0 ; i < n ; ++i ) {
f4b3                     if ( blossom[base[i]] ) {
9269                         base[i] = curbase;
fa64                         if ( !used[i] ) {
d079                             used[i] = true;
0583                             q[qt++] = i;
d224                         }
5acb                     }
f979                 }
b476             }
398f             else if ( p[to] == -1 ) {
6af0                 p[to] = v;
b064                 if ( match[to] == -1 )
1172                     return to;
7e68                 to = match[to];
ec93                 used[to] = true;
82f8                 q[qt++] = to;
333d             }
1362         }
f167     }
120f     return -1;
38fd }
38fd
b5b7 int main() {
b5b7     // ler grafo
326e     memset ( match, -1 , sizeof match );
326e     // otimizacao: comeca com um matching parcial guloso
c118     for ( int i = 0 ; i < n ; ++i ) {
f9fa         if ( match[i] == -1 ) {
7257             for ( size_t j = 0 ; j < adj[i].size() ; ++j ) {
0c38                 if ( match[ adj[i][j] ] == -1 ) {
877f                     match[ adj[i][j] ] = i;
9af3                     match[i] = adj[i][j];
6223                     break;
527f                 }
95ab             }
6d85         }
8796     }
5da4     for ( int i = 0 ; i < n ; ++ i ) {
d98e         if ( match[i] == - 1 ) {
8ba9             int v = find_path(i);
039e             while ( v != -1 ) {
55b1                 int pv = p[v], ppv = match[pv];
5729                 match[v] = pv, match[pv] = v;
ab71                 v = ppv;
b34d             }
be6f         }
2db7     }
2db7     // ...
110d }
```

Tarjan's algorithm for Strongly Connected Components

```
d41d // Precisa de adj[MAXN][MAXN], deg[MAXN] (lista de adjacencias)
1972 int cmp[MAXN];                      // Em qual componente esta o vertice i
830e int ind[MAXN],low[MAXN],cnt,cont;  // Zerar vetor ind com -1 !!!  <------
4e54 bool inStack[MAXN];
45c6 stack<int> st;
c06f
bf13 void tarjan(int v) {
```

```
b69c     ind[v] = low[v] = cnt++;
5387     st.push(v);
471e     inStack[v] = 1;
3b9d     for(int a=0;a<deg[v];++a) {
d9cf         int nxt = adj[v][a];
329e         if(ind[nxt] == -1) {
65c3             tarjan(nxt);
1da7             low[v] = min(low[v],low[nxt]);
9394         }
0cdb         else if(inStack[nxt]) low[v] = min(low[v],ind[nxt]);
6ad9     }
5a5c     if(ind[v] == low[v]) {
6e2f         int vv;
ff4b         do {
c01c             vv = st.top();
a0cd             st.pop();
43d7             inStack[vv] = 0;
01d6             cmp[vv] = cont;
b008         } while(vv != v);
8e76         ++cont;
c759     }
dac1 }
```

Articulation point and Bridges

```
7d7d int ind[MAXN],low[MAXN]; // Zerar ind com -1! <-----------------
e083 int cnt;                 // Zerar!             <-----------------
a811
a811 // Chamar com prev = -1
b954 void bap_dfs(int v,int prev) {
634d     ind[v] = low[v] = cnt++;
3095     int cont = 0;
757a     bool flag = 0;
1a10     for(int a=0;a<adj[v].size();++a) {
f611         int nxt = adj[v][a];
8aae         if(ind[nxt] == -1) {
ca72             ++cont;
da43             bap_dfs(nxt,v);
cba9             low[v] = min(low[v],low[nxt]);
9627             if(low[nxt] >= ind[v]) flag = 1;
4ce6             if(low[nxt] == ind[nxt]) {
4ce6                 // v-nxt eh uma ponte
339f             }
a92f         }
acae         else if(nxt != prev) low[v] = min(low[v],ind[nxt]);
3be5     }
5642     if(prev == -1) {
cc0a         if(cont > 1) {
cc0a             // v eh um ponto de articulacao
162a         }
c2ee     }
9663     else if(flag) {
9663         // v eh um ponto de articulacao
1c41     }
85ab }
```

9 Strings

Aho-Corasick

```
1ec0 #define MAXS 1000
0176 #define MAXT 100000
631f #define MAX 100000
ed2d #define cc  52
ed2d
b5eb int T[MAX], term[MAX], sig[MAX][cc], cnt;
b5eb
da6a void add (char s[MAXS], int id){
afc3     int x = 0, n = strlen(s);
f085     for (int i = 0; i < n; i++){
b391         int c = s[i]-'A';
d26c         if (sig[x][c] == 0) term[cnt] = 0, sig[x][c] = cnt++;
8816         x = sig[x][c];
404a     }
ceb6     term[x] = 1;
0749 }
0749
e6be void aho (){
a46c     queue <int> Q;
0498     for (int i = 0; i < cc; i++){
d60a         int v = sig[0][i];
601a         if (v) Q.push (v), T[v] = 0;
db6c     }
a687     while (!Q.empty()){
7125         int u = Q.front(); Q.pop();
b451         for (int i = 0; i < cc; i++){
6ce2             int x = sig[u][i];
1aff             if (x == 0) continue;
90ab             int v = T[u];
ed7d             while (sig[v][i] == 0 && v != 0) v = T[v];
eae4             int y = sig[v][i];
b5dc             Q.push(x), T[x] = y, term[x] |= term[y];
44a0         }
e3ee     }
1532 }
```

Manacher's algorithm for finding longest palindromic substring

```
8eac char s[200000];
b90a int n;
b90a // Encontrar palindromos - inicializa d1 e d2 com zeros, e eles guadram
b90a // o numero de palindromos centrados na posicao i (d1[i] e d2[i])
```

```
b90a // impar
0245 int d1[200000], d2[200000];
6103 void imp(){
adee    int l=0, r=-1;
e50f    for (int i=0; i<n; ++i) {
96d1        int k = (i>r ? 0 : min (d1[l+r-i], r-i)) + 1;
b185        while (i+k < n && i-k >= 0 && s[i+k] == s[i-k])  ++k;
7404        d1[i] = --k;
86ca        if (i+k > r)
7b9c            l = i-k,  r = i+k;
e5c3    }
9a4e }
9a4e // par
61d8 void par(){
f943    int l=0, r=-1;
5ec6    for (int i=0; i<n; ++i) {
1a46        int k = (i>r ? 0 : min (d2[l+r-i+1], r-i+1)) + 1;
4712        while (i+k-1 < n && i-k >= 0 && s[i+k-1] == s[i-k])  ++k;
b698        d2[i] = --k;
1f61        if (i+k-1 > r)
7e38            l = i-k,  r = i+k-1;
7a76    }
d167 }
```

Suffix array

```
e9bf #define MAXN 1000005
e9bf
0bf5 int n,t;  //n es el tamano de la cadena
b7a0 int p[MAXN],r[MAXN],h[MAXN];
b7a0 //p es el inverso del suffix array, no usa indices del suffix array ordenado,
b7a0 //p[i] eh a ordem do sufixo que comeca na posicao i
b7a0 //h el el tamano del lcp entre el i-esimo y el i+1-esimo elemento de
b7a0 //suffix array ordenado
b7a0 //r eh o vetor inverso
c70a string s;
c70a
759d void fix_index(int *b, int *e) {
e78c     int pkm1, pk, np, i, d, m;
f2ec     pkm1 = p[*b + t];
fbe8     m = e - b; d = 0;
dcaa     np = b - r;
b346     for(i = 0; i < m; i++) {
128a         if (((pk = p[*b+t]) != pkm1) && !(np <= pkm1 && pk < np+m)) {
fbac             pkm1 = pk;
53fc             d = i;
bd68         }
eb4b         p[*(b++)] = np + d;
9d6a     }
94a0 }
978e bool comp(int i, int j) {
7bfc     return p[i + t] < p[j + t];
4948 }
16f8 void suff_arr() {
38ec     int i, j, bc[256];
bc0a     t = 1;
4b4b     for(i = 0; i < 256; i++) bc[i] = 0;  //alfabeto
4b4b
53a1     for(i = 0; i < n; i++) ++bc[int(s[i])]; //counting sort inicial del alfabeto
53a1
b4b7     for(i = 1; i < 256; i++) bc[i] += bc[i - 1];
9ac6     for(i = 0; i < n; i++) r[--bc[int(s[i])]] = i;
e803     for(i = n - 1; i >= 0; i--) p[i] = bc[int(s[i])];
8552     for(t = 1; t < n; t *= 2) {
0c03         for(i = 0, j = 1; i < n; i = j++) {
c9e6             while(j < n && p[r[j]] == p[r[i]]) ++j;
2227             if (j - i > 1) {
1631                 sort(r + i, r + j, comp);
40ba                 fix_index(r + i, r + j);
6b94             }
399c         }
c661     }
a7d9 }
4163 void lcp() {
264b     int tam = 0, i, j;
dca2     for(i = 0; i < n; i++) if (p[i] > 0) {
cde0         j = r[p[i] - 1];
9c7d         while(s[i + tam] == s[j + tam]) ++tam;
07c5         h[p[i] - 1] = tam;
8107         if (tam > 0) --tam;
dfb9     }
1c1e     h[n - 1] = 0;
3f74 }
6aa9 int main(void){
6aa9     //suff_arr();
6aa9     //lcp();
6aa9     //
c9c7     cin >> s;
b858     s += '$';  //un caracter menor a todos para que no afecte el resultado
04ef     n = s.size();
e889     suff_arr();
e84b     for (int i = 0; i < n; i++){
995d         printf("%d ", p[i]);
3362     }
5cc9     printf("\n\n");
493f     lcp();
f31c     for (int i = 0; i < n; i++){
b6d1         printf("%d ", h[i]);
f49d     }
00f3     printf("\n");
1b8d     return 0;
0486 }
```

Knuth-Morris-Pratt algorithm

```
33c2 class KMP {
5f1e private:
ab34     string pattern;
245b     int len;
93f5 public:
8dad     vector<int> f;
2b5b     KMP(string p) {
7fbd         pattern = p;
7186         len = p.size();
3e31         f.resize(len+2);
dc81         f[0] = f[1] = 0;
89d1         for(int a=2;a<=len;++a) {
6c79             int now = f[a-1];
3ad8             while(1) {
48f3                 if(p[now] == p[a-1]) {
d73c                     f[a] = now+1;
63e2                     break;
11d9                 }
0562                 if(now==0) {
c0b7                     f[a] = 0;
51a9                     break;
f206                 }
4675                 now = f[now];
15df             }
2e00         }
3f8e     }
3f8e     //returns a vector of indices with the beginning of each match
56a6     vector<int> match(string text) {
d8c1         vector<int> ret;
0744         int size = text.size();
c9ce         int i=0,j=0;
54c5         while(j<size) {
4846             if(text[j] == pattern[i]) {
6d09                 ++i; ++j;
bb49                 if(i==len) {
4c96                     ret.push_back(j-len);
2a3b                     i = f[i];
25d0                 }
eb70             }
f16c             else if(i>0) i = f[i];
1f75             else j++;
938a         }
49a4         return ret;
a285     }
abdb };
```

Hash

```
dc9a typedef char HType;
dc9a
0c12 const int P1 = 31, P2 = 37, MOD = (int)1e9 + 7;
0c12
ec30 struct Hash {
d084     ll h1, h2;
0d28     Hash(ll a = 0, ll b = 0) { h1 = a; h2 = b; }
0f7b     void append(HType c) {
b917         h1 = (P1*h1 + c) % MOD;
afbc         h2 = (P2*h2 + c) % MOD;
ac5b     }
547d     bool operator== (Hash that) const {
3369         return h1 == that.h1 && h2 == that.h2;
8aaa     }
f1f5     bool operator!= (Hash that) const {
674f         return h1 != that.h1 || h2 != that.h2;
6710     }
ba95     Hash operator*  (Hash that) const {
e26c         return Hash((h1*that.h1)%MOD, (h2*that.h2)%MOD);
1895     }
b83d     Hash operator-  (Hash that) const {
c53d         return Hash( (h1 - that.h1 + MOD)%MOD, (h2 - that.h2 + MOD)%MOD);
bb50     }
56f7 };
56f7
2b89 Hash pot[MAXN];
2b89
5df8 vector<Hash> build_hash(int n, HType *v) {
bd84     pot[0] = Hash(1,1);
c2d0     vector<Hash> ret;
d0df     Hash acc;
8ace     for(int i = 0; i < n; i++) {
6231         acc.append(v[i]);
ab47         ret.push_back(acc);
ecc9         if(i > 0) pot[i] = pot[i-1] * Hash(P1, P2);
4d55     }
6b7f     return ret;
26eb }
26eb
aa1a inline Hash get_hash(int l, int r, vector<Hash> &hashv) {
bda8     if(l == 0) return hashv[r];
65e1     return hashv[r] - hashv[l-1] * pot[r-l+1];
0e4b }
```

Z-algorithm

```
d41d // Z-algorithm, O(N)
d41d // Builds array z such that z[i] = size of longest prefix substring
d41d // starting at index i
dfb2 vector<int> Z(string s) {
297a     vector<int> z(1,s.size());
d7d5     int l=0,r=0;
57ec     for(int a=1;a<(int)s.size();++a) {
e35e         if(r < a) {
```

```
ae49             l = r = a;
8690             while(r<(int)s.size() && s[r] == s[r-l]) ++r;
ea6e             z.push_back(r-l);
e7b0             r--;
9141         }
4f00         else if(z[a-l] < r-a+1) z.push_back(min<int>(z[a-l],s.size()-a));
cae0         else {
d3a9             l = a;
552c             while(r<(int)s.size() && s[r] == s[r-l]) ++r;
657f             z.push_back(r-l);
0258             r--;
7bee         }
f580     }
f2fc     return z;
c455 }
```

10 Data structures

Static Range Minimum Query

```
3dd1 int rmq[LOG][MAXN];
8592 int v[MAXN];
dc31 int n;
dc31
dc31 // Builds RMQ structure for array v of size n in O(n*log(n))
8f76 void build_rmq() {
babe     for(int i = 0; i < n; i++)
ed78         rmq[0][i] = v[i];
1a88     for(int log = 1; log < LOG; ++log)
8012         for(int i = 0; i < n; i++)
7f02             rmq[log][i] = min(rmq[log-1][i],
5a08                               rmq[log-1][min(n-1, i + (1<<(log-1)))]);
b394 }
b394 // l e r inclusives
3fde int get_rmq(int l,int r) {
6e47     int len = r - l + 1;
ef17     int bit = 31 - __builtin_clz(len);
173e     return min(rmq[bit][l], rmq[bit][r - (1<<bit) + 1]);
11eb }
```

2D Longest Increasing Subsequence

```
6cdc struct ponto{
9745     int x,y;
c458     bool operator <(const ponto& p) const { return x<p.x; }
0343 };
0343
1d06 ponto v[MAXN];
b9bf set<ponto> m[MAXN];
54ac set<ponto> :: iterator it;
bde8 int n, t;
bde8
5fe8 void insere(int p, ponto c){
ffbb     if(p>t)t++;
9fd2     it = m[p].lower_bound(c);
6696     vector< ponto > tira;
de4f     for(;it!=m[p].end();it++){
1b6e         if(it->y >= c.y) tira.pb(*it);
c573         else break;
0390     }
b739     for(int j=0;j<sz(tira);j++) m[p].erase(tira[j]);
55fc     m[p].insert(c);
a393 }
a393
cf14 bool cabe(int p, int i){
2c2c     if(p==0) return false;
d3ba     if(p==t+1) return true;
a479     it = m[p].lower_bound(v[i]);
a0ee     if(it == m[p].begin()) return true;
584d     it--;
7dcf     return v[i].y <= it->y;
d9e1 }
d9e1
5c14 int lis(){
6c55     t=0;
7bdd     f (i, 0, n) m[i].clear();
7c8a     for(int i=0;i<n;i++){
985b         int l=0,r=t+1, j;
41ae         while(l!=r){
0941             j = (l+r)/2;
4e53             if(!cabe(j,i)) l=j+1;
e33a             else r=j;
7916         }
ff35         insere(l,v[i]);
927a     }
f8a9     return t;
8fb5 }
```

Treap

```
d41d // Supports insertion, deletion, querying kth-element and finding element
d41d // Keeps duplicate elements as different nodes
d41d
10aa template <class T> class Treap {
d02a     struct Node {
a39f         T val;
2112         int h,cnt;
bfcc         Node *l, *r;
588c         Node(T val2) {
ed02             val = val2;
1501             h = rand();
e0eb             cnt = 1;
```

```
cd33            l = r = NULL;
b66b        }
cb40    };
0efa    Node *root;
6510    inline Node* newNode(T val) { return new Node(val); }
2612    inline void refresh(Node *node) {
e761        if(node == NULL) return;
d4ef        node->cnt = (node->l == NULL ? 0 : node->l->cnt) +
b26d            (node->r == NULL ? 0 : node->r->cnt) + 1;
f737    }
74a9    void _insert(Node *&node, T val) {
205f        if(node == NULL) {
2a7b            node = newNode(val);
1885            return;
993f        }
731b        if(val <= node->val) {
1f9e            _insert(node->l, val);
b3c9            if(node->l->h > node->h) {
7762                Node *aux = node->l;
b75b                node->l = aux->r;
aca0                aux->r = node;
cc9c                node = aux;
ed45                refresh(node->r);
7741                refresh(node);
f999            }
203f            else refresh(node);
6a29        }
19fb        else {
930c            _insert(node->r, val);
60c7            if(node->r->h > node->h) {
0531                Node *aux = node->r;
da12                node->r = aux->l;
1663                aux->l = node;
dcd3                node = aux;
325a                refresh(node->l);
757d                refresh(node);
a155            }
721a            else refresh(node);
a9d0        }
43b5    }
6cd0    Node* merge(Node *L,Node *R) {
f742        if(L == NULL) return R;
4efc        if(R == NULL) return L;
79b4        if(L->h < R->h) {
1704            L->r = merge(L->r,R);
5c1f            refresh(L);
5e00            return L;
ca41        }
02b9        else {
0301            R->l = merge(L,R->l);
9863            refresh(R);
7961            return R;
9103        }
a81a    }
a81a    // not used. splits node into two trees a(<=val) and b(>val)
```

```
314d    void split(T val, Node *node, Node *&a, Node *&b) {
05c5        if(node == NULL) {
5cdb            a = b = NULL;
069b            return;
f1a9        }
0f67        Node *aux;
496e        if(val > node->val) {
44c0            split(val,node->r,aux,b);
c4a9            node->r = aux;
2145            a = node;
d30c            refresh(a);
64c5        }
4bca        else {
6a72            split(val,node->l,a,aux);
372d            node->l = aux;
426d            b = node;
f8ae            refresh(b);
cfb4        }
79c4    }
79c4    // erases a single appearence of val
42f8    void _erase(Node *&node, T val) {
f6ee        if(node == NULL) return;
3a82        if(node->val > val) _erase(node->l, val);
70b2        else if(node->val < val) _erase(node->r, val);
bdea        else node = merge(node->l,node->r);
323c        refresh(node);
8fb2    }
8fb2    // 0-indexed (not safe if element doesnt exist)
9fa7    int _kth(Node *node,int k) {
3a77        int ql = (node->l == NULL ? 0 : node->l->cnt);
efe4        if(k < ql) return _kth(node->l,k);
7072        if(k == ql) return node->val;
209f        k -= ql + 1;
61e3        return _kth(node->r,k);
1509    }
1509    // returns position (0-indexed) of element 'val'. -1 if it doesn't exist
560f    int _find(Node *node, T val) {
5713        if(node == NULL) return -1;
d526        if(node->val == val) return (node->l == NULL ? 0 : node->l->cnt);
d502        else if(node->val > val) return _find(node->l,val);
2893        else {
7cec            int pos = _find(node->r,val);
53a4            if(pos == -1) return -1;
76f9            return 1 + (node->l == NULL ? 0 : node->l->cnt) + pos;
509e        }
d04b    }
d07b    void _clear(Node *&node) {
3224        if(node == NULL) return;
7ad7        _clear(node->l); _clear(node->r);
6120        delete node;
c349        node = NULL;
46e2    }
bdb0 public:
06c9    Treap() { root = NULL; }
0b2f    void insert(T val) { _insert(root,val); }
```

```
547c     int kth(int k) { return _kth(root,k); }
4978     int size() { return root == NULL ? 0 : root->cnt; }
7544     void clear() { _clear(root); }
6986     void erase(T val) { _erase(root,val); }
ac7e     int find(T val) { return _find(root,val); }
a350 };
```

KD-Tree

```
be52 struct point {
e3bf     ll x, y, z;
a0fa     point(ll x=0, ll y=0, ll z=0): x(x), y(y), z(z) {}
d25b     point operator-(point q) { return point(x-q.x, y-q.y, z-q.z); }
b9f7     ll operator*(point q) { return x*q.x + y*q.y + z*q.z; }
cc29 };
1152 typedef vector<point> polygon;
1152
149d struct KDTreeNode {
5e32     point p;
8257     int level;
6a86     KDTreeNode *below, *above;
6a86
dcaf     KDTreeNode (const point& q, int levl) {
e3e5         p = q;
d2e9         level = levl;
7b48         below = above = 0;
2fe5     }
d31e     ~KDTreeNode() { delete below, above; }
d31e
f38d     int diff (const point& pt) {
777c         switch (level) {
ce32         case 0: return pt.x - p.x;
0982         case 1: return pt.y - p.y;
26c3         case 2: return pt.z - p.z;
74d7         }
d403         return 0;
1a75     }
c8c3     ll distSq (point& q) { return (p-q)*(p-q); }
c8c3
41fd     int rangeCount (point& pt, ll K) {
9124         int count = (distSq(pt) < K*K) ? 1 : 0;
d0f8         int d = diff(pt);
598d         if (-d <= K && above != 0)
b820             count += above->rangeCount(pt, K);
799c         if (d <= K && below != 0)
406c             count += below->rangeCount(pt, K);
d9c3         return count;
a157     }
375a };
375a
1a75 class KDTree {
7d28 public:
106f     polygon P;
130c     KDTreeNode *root;
a4ad     int dimention;
f095     KDTree() {}
91b1     KDTree(polygon &poly, int D) {
7cf8         P = poly;
14e0         dimention = D;
5d49         root = 0;
3061         build();
e8ab     }
2035     ~KDTree() { delete root; }
2035
2035     //count the number of pairs that has a distance less than K
fee1     ll countPairs(ll K) {
890b         ll count = 0;
f4d3         f(i, 0, P.size())
f807             count += root->rangeCount(P[i], K) - 1;
e5bf         return count;
8eb5     }
8eb5
f1ed protected:
544a     void build() {
c821         random_shuffle(all(P));
b93b         f(i, 0, P.size()) {
a894             root = insert(root, P[i], -1);
937f         }
23aa     }
f1db     KDTreeNode *insert(KDTreeNode* t, const point& pt, int parentLevel) {
f750         if (t == 0) {
9ba3             t = new KDTreeNode (pt, (parentLevel+1) % dimention);
ee91             return t;
e0eb         } else {
d4e8             int d = t->diff(pt);
247d             if (d <= 0) t->below = insert (t->below, pt, t->level);
36e9             else t->above = insert (t->above, pt, t->level);
8ced         }
5747         return t;
dd9f     }
ca38 };
ca38
d70e int main() {
1a3c     int n, k;
20fc     point e;
2efb     polygon p;
e672     while (cin >> n >> k && n+k) {
8293         p.clear();
0fbb         f(i, 0, n) {
9912             cin >> e.x >> e.y >> e.z;
7ce5             p.pb(e);
faf5         }
e75f         KDTree tree(p, 3);
6ada         cout << tree.countPairs(k) / 2LL << endl;
ae2c     }
77c1     return 0;
c2ab }
```