Stormtroop3rs[C] ICPC Team Notebook 2017

Sumário

1 Template

1.1 Template C++

```
#include <bits/stdc++.h>
using namespace std;

#define rep(i, a, b) for(int i=(a); i<(b); i++)
#define pb push_back
#define mk make_pair
#define debug(x) cout<<__LINE__<<": "<<#x<<" = "<<x<<endl;
#define all(c) (c).begin(), (c).end()
#define F first
#define S second
#define UNIQUE(c) sort(all(c)); (c).resize(unique(all(c))-c.
    begin());
#define PI 3.14159265358979323846264338327950288841971

typedef long long ll;
typedef pair<int, int> ii;
const int INF = 0x3f3f3f3f;
const double EPS = 1e-9;
//(-1: x<y) (0: x==y) (1: x>y)
inline int cmp(double x, double y = 0, double tol = EPS){
  return ((x <= y+tol) ? (x+tol < y) ? -1:0:1); }

// STL Utils *
//Leitura de String:
```

```
#include <sstream>
istringstream ins; // ins.str(s); ins >> a >> b; //IN
ostringstream outs; // outs << a << b; s = outs.str(); //OUT

upper_bound(v.begin(), v.end(), key);//justafter the last
    element found
lower_bound(v.begin(), v.end(), key);//first element found, or,
     or case
//the element is not found, first position it could be inserted
     without
//violating the order (first element greater or equal key)
binary_search(v.begin(), v.end(), key);
next_permutation(v.begin(), v.end());

//Heap without priority change
priority_queue<ii,vector<ii>, greater<ii> > Q;
Q.push<ii(priority,key)>; //insert element
int u = Q.top().second; Q.pop(); //remove element

//Heap with priority change ( priorities in v[] )
struct comp{
  inline bool operator() (const int i, const int j) {
    return (v[i] != v[j]) ? (v[i] < v[j]) : (i < j);
  } };
set<int, comp> s;
s.erase(x); v[x]=new_priority; s.insert(x); //priority change
v[x] = priority; s.insert(x); //insert element
int u = *s.begin(); s.erase(*s.begin()); //remove element

remove(all(v), val); //removes all elements with val key
v.resize( remove(all(v), val) - v.begin()); // remove and
    resize the vector
reverse(all(v));
find(all(v), key); //returns first iterator to key
find_if(all(v), pred);//where pred is bool pred(T key)

//String Functions ( s is a string )
s.find(string pattern, int pos=0); // find a pattern in string
starting from position zero. returns index of first element if
pattern is found and a large number if not found
// substring de s comecando em pos e com tamanho tam
s.substr(int pos, int tam);

int main(){
  return 0;
}
```

2 Numerical algorithms

2.1 Triângulo de Pascal

```
/*Calcula os numeros binomiais (N,K) = N!/(K!(N-K)!)
(N,K) representa o numero de maneiras de criar um subconjunto
    de tamanho
K dado um conjunto de tamanho N.  A ordem dos elementos nao
    importa.*/

const int MAXN = 50;
long long C[MAXN][MAXN];
void calc_pascal() {
  memset(C, 0, sizeof(C));
  for (int i = 0; i < MAXN; ++i) {
    C[i][0] = C[i][i] = 1;
    for (int j = 1; j < i; ++j)
      C[i][j] = C[i-1][j-1] + C[i-1][j];
  }
}
//Pascal triangle elements:
C(33, 16) = 1.166.803.110 [int limit]
C(34, 17) = 2.333.606.220 [unsigned int limit]
C(66, 33) = 7.219.428.434.016.265.740 [int64_t limit]
C(67, 33) = 14.226.520.737.620.288.370 [uint64_t limit]
//Fatorial
12! = 479.001.600 [(unsigned) int limit]
20! = 2.432.902.008.176.640.000 [(unsigned) int64_t limit ]
```

2.2 GCD-LCM

```
ll gcd(ll a, ll b){return b==0 ? a:gcd(b, a%b);}
ll lcm(ll a, ll b){
  if (a and b) return abs(a)/gcd(a, b)*abs(y);
  else return abs(a | y);
}
```

2.3 Bezout Theorem

```
/* Aplicacoes:
Determinar a solucao da equacao a*x+b*y = gcd(a, b), onde a e b
     sao dois
numeros inteiros naturais.
Como chamar: egcd(a, b), Retorno: a tupla (gcd(a, b), x, y)
Determina tambem o Inverso Modular */
struct Triple{
  int d, x, y; // long long ?
  Triple (int q, int w, int e):d(q), x(w), y(e){}
};
Triple egcd (int a, int b){
  if (!b) return Triple(a, 1, 0);
  Triple q = egcd(b, a%b);
  return Triple(q.d, q.y, q.x - a/b * q.y);
}
int invMod (int a, int n){
  Triple t = egcd (a, n);
  if (t.d > 1) return 0;
```

```
  return (t.x%n+n)%n;
}
```

2.4 Teorema Chinês dos Restos

```
#define MAXN 1000
/* Aplicacoes:
Determinar x tal que x = a[i] (mod m[i]).
Exemplo: Para a[] = {1, 2, 3} e m[] = {5, 6, 7}
.: x = 206
Como chamar a funcao:
  1) Chamar a funcao crt()
Depende do Inverso Modular (Bezout ou Fermat)
Testado: URI 1831
*/
int n;
int a[MAXN], m[MAXN];
int crt(){
  int M = 1, x = 0;
  for(int i = 0; i < n; ++i)  M *= m[i];
  for(int i = 0; i < n; ++i)
    x += a[i]*invMod(M/m[i], m[i])*(M/m[i]);
  return (((x%M)+M)%M);
}
```

2.5 Crivo de Eratóstenes

```
bitset<10000005> bs;  vector<int> primos;
void crivo(ll limite = 10000000LL){ // calcula primos ate
    limite
  primos.clear();
  bs.set();
  bs[0] = bs[1] = 0;
  for(ll i = 2; i <= limite; i++)
    if (bs[i]){
      for(ll j = i*i; j <= limite; j += i)
        bs[j] = 0;
      primos.push_back(i);
    }
}
bool isPrime(ll N, ll limite){
  if (N <= limite) return bs[N];
  for(int i = 0; i < (int) primos.size(); i++)
    if (N%primos[i] == 0) return false;
  return true; }
```

2.6 Divisores de N

```
// Retorna todos os divisores naturais de N em O(sqrt(N))
vector<ll> divisores(ll N){
  vector<ll> divisor; divisor.clear();
  for(ll div = 1, k; div*div <= N; div++){
    if (N%div == 0){
      divisor.pb(div);
      k = N/div;
      if (k != div) divisor.pb(k);
    }
  }
  // sort(divisor.begin(), divisor.end()); // caso precise
      ordenado
  return divisor;
}
```

2.7 Funções com Números Primos (Crivo, Fatoração, PHI, etc)

```
map<int, int> factors;
//Encontra os fatores primos de N .: N = p1^e1 * ... *pi^ei
// factors armazena em first o fator primo e em segundo seu
    expoente
void primeFactors(ll N){
  factors.clear();
  while (N%2==0) ++factors[2], N>>=1;
  for(ll PF=3; PF*PF <= N; PF+=2){
    while (N%PF == 0)
      N /= PF, factors[PF]++;
  }
  if (N > 1) factors[N] = 1;
}
//Funcoess derivadas dos numeros primos
void NumberTheory(ll N){
  primeFactors(N);
  map<int, int> :: iterator f;//<fator primo, expoente>
  ll Totient=N, numDiv=1, sumDiv=1, numDiffPF, sumPF=0;
  numDiffPF = factors.size();
  for(f = factors.begin(); f != factors.end(); f++){
    ll PF = f->fst, power = f->snd;
    Totient -= Totient/PF;
    numDiv *= (power+1);
    sumDiv *= ((ll)pow((double)PF, power+1.0)-1)/(PF-1);
    sumPF += PF;
  }
  printf ("Totiente/Euler-Phi de N = %lld\n", Totient);
  printf ("qt de divisores de N = %lld\n", numDiv);
  printf ("soma dos divisores de N = %lld\n", sumDiv);
  printf ("qt de fatores primos distintos = %lld\n", numDiffPF)
      ;
  printf ("soma dos fatores primos = %lld\n", sumPF);
}

//Calcula Euler Phi para cada valor do intervalo [1, N]
#define MM 1000010
int phi[MM];
```

```
void crivo_euler_phi(int N){
  for(int i = 1; i <= N; i++)
    phi[i] = i;
  for(int i = 2; i <= N; i++)
    if (phi[i] == i){
      for(int k = i; k <= N; k+=i)
        phi[k] = (phi[k]/i)*(i-1);
    }
}

//Qtde de fatores primos distintos de cada valor do range [2,
    MAX_N]
#define MAX_N 10000000
int NDPF[MAX_N]; //
void NumDiffPrimeFactors(){
  memset(NDPF, 0, sizeof NDPF);
  for(int i = 2; i < MAX_N; i++)
    if (NDPF[i] == 0)
      for(int j = i; j < MAX_N; j += i)
        NDPF[j]++;
}
```

2.8 Exponenciação Modular Rápida

```
/* Calcula (B^P)%MOD em O(log(P))
Calcula o inverso modular de b(modulo mod) se mod for primo.
    basta
fazer invB = fastpow(b,mod-2,mod); */
int fastpow(int b, int p, int mod){
  int ret = 1;
  for(ll pow = b; p > 0; p >>= 1, pow = (pow*pow)%mod)
    if(p & 1) ret = (ret*pow)%mod;
  return ret;
}
```

2.9 Exponenciação de Matriz

```
const int M = 2; ll mod = 1e9+7;
int sz = 2;
ll mat[M][M], ans[M][M], tmp[M][M];

void mult(ll a[][M], ll b[][M]){
  rep(i, 0, sz)
    rep(j, 0, sz){
      tmp[i][j] = 0;
      rep(k, 0, sz)
        tmp[i][j] += a[i][k]*b[k][j];
      tmp[i][j] %= mod;
    }
  memcpy(a,tmp,sizeof tmp);
}
// ans = mat^n
void fastExp(ll ans[][M], ll n){
  // inicializar mat
  mat[0][0] = mat[0][1]  = mat[1][0] = 1;
  mat[1][1] = 0;
  // matriz identidade
  rep(i,0,sz) rep(j,0,sz) ans[i][j] = (i==j);
  while(n) {
    if (n & 1) mult(ans, mat);
    n >>= 1;
    mult(mat, mat);
  }
    //n-\'esino termo de fibonacci
  //cout << ans[1][0]*fib(1) + ans[1][1] * fib(0) << "\n";
}
```

2.10 Floyd Cycle Detection

```
// faz o mesmo que brent-cycle
ii floyd_cycle(int x){
  int t = f(x), h = f(f(x)), start = 0, lenght = 1;
  while (t != h){t = f(t); h = f(f(h)); }

  h = x;
  while (t != h){ t = f(t); h = f(h); start++; }

  t = start; h = f(t);
  while (t != h){ h = f(h); lenght++; }
  return ii(start, lenght);
}
```

2.11 Brent Cycle Detection

```
/*Dado uma sequencia f(x0), f(f(x0)), ..., f(f(...f(x0))), ela
    pode ser
ciclica. A funcao retorna o tamanho do ciclo e o valor que o
    inicia */
ii brent_cycle(int x){
    int p = 1, length = 1, t = x, start = 0;
    int h = f(x);
    while (t != h){
        if (p == length){ t = h; p *= 2; length = 0; }
        h = f(h);
        ++length;
    }
    t = h = x;
    for(int i = length; i != 0; --i)  h = f(h);
    while (t != h){
        t = f(t);  h = f(h);
        ++start;
    }
    return ii(start, length);
}
```

2.12 Romberg's method - Calcula Integral (UFS2010)

```
//Calcula a integral de f[a, b]
typedef long double ld;

ld f(double x){
  // return f(x)
}

ld romberg(ld a, ld b){
  ld R[16][16], div = (b-a)/2;
  R[0][0] = div*(f(a)+f(b));
  for(int n = 1; n <= 15; n++, div /= 2){
    R[n][0] = R[n-1][0]/2;
    for(ld sample = a+div; sample<b; sample += 2*div)
    R[n][0] += div*f(a+sample);
  }
  for(int m = 1; m <= 15; m++)
    for(int n = m; n <= 15; n++)
    R[n][m] = R[n][m-1]+1/(pow(4, m)-1)*(R[n][m-1]-R[n-1][m-1])
        ;
  return R[15][15];
}
```

2.13 Pollard's rho algorithm (UFS2010)

```
//Retorna um fator primo de N
ll f(ll val){return (val*val+pollard_r)%pollard_n; }
ll myabs(ll a){return a >= 0 ? a:-a; }
ll pollard(ll n){
  srand(unsigned(time(0)));
  pollard_n = n;
  long long d = 1;
  do {
    d = 1;
    pollard_r = rand()%n;
    long long x = 2, y = 2;
    while(d == 1)
      x = f(x), y = f(f(y)), d = __gcd(myabs(x-y), n);
  } while(d == n);
  return d;
}
```

2.14 Miller-Rabin's algorithm (UFS2010)

```
// Teste de primalidade
bool miller_rabin(int n, int base){
  if(n <= 1) return false;
  if(n%2 == 0) return n == 2;
  int s = 0, d = n-1;
  while(d%2 == 0) d /= 2, ++s;
  int base_d = fastpow(base, d, n);
  if(base_d == 1) return true;
  int base_2r = base_d;
  for(int i = 0; i < s; ++i){
    if(base_2r == 1) return false;
    if(base_2r == n-1) return true;
    base_2r = (long long)base_2r*base_2r%n;
  }
  return false;
}
bool isprime(int n){
  if(n == 2 || n == 7 || n == 61) return true;
  return miller_rabin(n, 2) && miller_rabin(n, 7) &&
      miller_rabin(n, 61);
}
```

2.15 Quantidade de dígitos de N! na base B

```
int NumOfDigitsInFactorial(int N, int B){
  double logFatN = 0;
  for (int i = 1; i <= N; i++)
    logFatN += log((double)i);
  int nd = floor(logFatN / log((double)B)) + 1;
  return nd;
}
```

2.16 Quantiade de zeros a direita de N! na base B

```
int NumOfTrailingZeros(int N, int B){
  int nfact = fatora(B);
  int zeros = INF;
    // para cada fator de B, aux representa qtas vezes
    // fator[i]^expoente[i] aparece na representacao de N!
  for (int i=0; i < nfact; i++){
    int soma = 0;
    int NN = N;
    while (NN){
      soma += NN/fator[i];
      NN /= fator[i];
    }
    int aux = soma / expoente[i];
    zeros = min(zeros, aux);
  }
  return zeros;
}
```

2.17 Baby Step Giant Step

```
/*Determinar o menor E tal que B^E = N (mod P), -1 se for
    impossivel
```

```
Requer: Bezout Theorem*/
ll bsgs(ll b, ll n, ll p){
  if (n == 1) return 0;
  map <ll, int> table;
  ll m = sqrt(p) + 1, pot = 1, pot2 = 1;
  for (int j = 0; j < m; ++j){
    if (pot == n) return j;
    table[(n * invMod(pot, p)) % p] = j;
    pot = (pot * b) % p;
  }
  for (int i = 0; i < m; ++i){
    if (table.find(pot2) != table.end())
      return i * m + table[pot2];
    pot2 = (pot * pot2) % p;
  }
  return -1;
}
```

2.18 Primos num intervalo

```
//Encontra os primos no intervalo [n,m]
vector<int> ret;
void primesBetween(int n, int m){
  ret.clear();
  vector<int> primes(m-n+1);
  for (int i = 0; i < m - n + 1; ++i)
    primes[i] = 0;
  for (int p = 2; p*p <= m; ++p){
    int less = (n / p)*p;
    for (int j = less; j <= m; j += p)
      if (j != p && j >= n)
        primes[j - n] = 1;
  }
  for (int i = 0; i < m - n + 1; ++i){
    if (primes[i] == 0 && n+i != 1){
      ret.push_back(n+i);
    }
  }
}
```

2.19 FFT

```
typedef complex<double> comp;

const double PI = 2*acos(0);

const int MAX_N = 1 << 20;
int rev[MAX_N];
comp roots[MAX_N];

void preCalc(int N, int BASE){
  for (int i = 1; i < N; ++i) rev[i] = (rev[i >> 1] >> 1) + ((i
      & 1) << (BASE - 1));
  int NN = N >> 1;
  roots[NN] = comp(1, 0);
  roots[NN + 1] = comp(cos(2 * PI / N), sin(2 * PI / N));
  for (int i = 2; i < NN; ++i) roots[NN + i] = roots[NN + i - 1
      ] * roots[NN + 1];
  for (int i = NN - 1; i > 0; --i) roots[i] = roots[2 * i];
}

void fft( vector<comp> &a, bool invert ){
  int N = a.size();
  if (invert) rep(i, 0, N) a[i] = conj(a[i]);
  rep(i, 0, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int k = 1; k < N; k *= 2){
    for (int i = 0; i < N; i += 2 * k){
      rep(j, 0, k){
        comp B = a[i + j + k] * roots[k + j];
        a[i + j + k] = a[i + j] - B;
        a[i + j] = a[i + j] + B;
      }
    }
  }
  if(invert) rep(i, 0, a.size()) a[i] /= N;
}

vector<comp> multiply_real( vector<comp> a, vector<comp> b,
    vector<comp> c ){
  int n = a.size();
  int m = b.size();

  int base = 0, N = 1;
  while (N < n+m-1) base++, N <<= 1;
  preCalc(N, base);

  a.resize(N, comp(0,0));
  c.resize(N);

  rep( i, 0, b.size() ) a[i] = comp( real(a[i]), real(b[i]) );
  fft(a, 0);
  rep( i, 0, N ) {
    int j = (N - i) & (N - 1);
    c[i] = (a[i] * a[i] - conj(a[j] * a[j])) * comp(0, -0.25);
  }
  fft(c, 1);
  return c;
}
```

3 Geometria 2D

3.1 Geometria 2D Library

```
const double EPS = 1e-9;
```

```
inline int cmp( double x, double y = 0, double tol = EPS){
  return ( (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1);
}

int sgn(ll x) { return x < 0 ? -1 : x > 0; }

struct point{
  double x, y;
  point(double x=0, double y=0): x(x), y(y){}
  point operator +(point q){ return point(x+q.x, y+q.y);}
  point operator -(point q){ return point(x-q.x, y-q.y);}
  point operator *(double t){ return point(x*t, y*t); }
  point operator /(double t){ return point(x/t, y/t); }
  point operator /(point q){ // divisao de num complexos
    return point(x*q.x+y*q.y, y*q.x-x*q.y)/q.norma2();
  }
  double norma2(){ return (*this)*(*this); } // |u|^2
  double operator *(point q){ return x*q.x+y*q.y; }
  double operator %(point q){ return x*q.y-y*q.x; }
  int cmp(point q) const{
    if(int t = ::cmp(x, q.x)) return t;
    return ::cmp(y, q.y);
  }
  bool operator == (point q) const{return cmp(q) == 0;};
  bool operator != (point q) const{return cmp(q) != 0;};
  bool operator < (point q) const{return cmp(q) < 0;};
};
ostream &operator<<(ostream &os, const point &p){
  os << "(" << p.x << "," << p.y << ")";
}
#define vec(a, b) (b-a)
typedef vector<point> polygon;

// Rotaciona um ponto em relacao a origem
point RotateCCW90(point p)  { return point(-p.y, p.x); }
point RotateCW90(point p) { return point(p.y, -p.x); }

// rotaciona um ponto P em A graus no sentido anti-horario.
// para rotacionar no sentido horario, basta A ser negativo
point RotateCCW(point p, double A){
  A = (A/180.0)*acos(-1.0); // convertendo para radianos
  return point(p.x*cos(A)-p.y*sin(A), p.x*sin(A)+p.y*cos(A));
}

// tamanho de um vetor
double abs(point u){
  return sqrt(u*u);
}


/*Decide se P->Q->R forma um giro a esquerda, direita ou se sao
colineares. Retorna 1 se R estiver a esquerda da reta PQ. -1 se
estiver a direta e 0 se forem colineares.*/
int ccw(point p, point q, point r){
  return cmp(vec(p, q)%(vec(p, r)));
}

//  Projeta o vetor A sobre a direcao do vetor B
point project(point a, point b){
  return b*((a*b)/(b*b));
}

// Retorna a projecao do ponto P sobre reta definida por [A,B]
point projectPointLine(point p, point a, point b){
  return p + project(p-a, b-a);
}

//  Retorna o angulo entre os vetores AB e AC
double arg(point B, point A, point C){
  point u = vec(A, B), v = vec(A, C);
  return atan2(u%v, u*v);
}


// verifica se x esta entre [l,r] *Tente evitar usar double!
bool is_in(double l, double r, double x) {
  if (l > r) swap(l, r);
  return (l <= x + EPS) && (x - EPS <= r);
}

//  Determina se P esta entre o segmento fechado [A,B],
    inclusive
bool between(point p, point a, point b) {
  if (ccw(p,a,b) != 0) return false;
  else return is_in(a.x, b.x, p.x) && is_in(a.y, b.y, p.x);
}

// (Opcao2) Determina se P esta entre o segmento fechado [A,B],
     inclusive
// Se vc nao entendeu a anterior ou se acha que ela pode ser o
    problema
// use essa. Se nao der certo, tente mudar a precisao usada em
    cmp() para 1e-15
bool between2 (point p, point a, point b){
  return cmp(abs(p-a) + abs(p-b), abs(a-b))==0;
}

// Distancia de ponto P para reta que passa por [A,B]
// armazena em C (por ref) o ponto projecao de P na reta
double distancePointLine(point p, point a, point b, point& c){
  c = projectPointLine(p,a,b);
  return fabs((b-a)%(p-a))/abs(a-b); // or abs(p-c);
}

// Distancia de ponto P ao segmento [A,B]
// armazena em C (por ref) o ponto de projecao de P em [A,B]
// se este ponto sair do segmento, eh retornado o mais proximo
double distancePointSeg(point p, point a, point b, point& c){
```

```
  if ((b-a)*(p-a) <= 0) { c = a; return abs(a-p); }
  if ((a-b)*(p-b) <= 0) { c = b; return abs(b-p); }

  c = projectPointLine(p,a,b);
  return fabs((b-a)%(p-a))/abs(a-b); // or abs(p-c);
}

// Distancia entre (x,y,z) e plano ax+by+cz=d
double distancePointPlane(double x, double y, double z, double
    a, double b, double c, double d){
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// Determina se os segmentos [A, B] e [C, D] se tocam
bool seg_intersect(point a, point b, point c, point d){
  int d1, d2, d3, d4;
  d1 = ccw(c, a, d);    d2 = ccw(c, b, d);
  d3 = ccw(a, c, b);    d4 = ccw(a, d, b);
  if (d1*d2 == -1 && d3*d4 == -1) return true;
  if (d1 == 0 && between(c, a, d)) return true;
  if (d2 == 0 && between(c, b, d)) return true;
  if (d3 == 0 && between(a, c, b)) return true;
  if (d4 == 0 && between(a, d, b)) return true;
  return false;
}

// Encontra a interseccao das retas (p-q) e (r-s) assumindo que
    existe
// apenas 1 intereccao
// Se for entre segmentos, verificar se interseptam primeiro
point line_intersect(point p, point q, point r, point s){
  point a = q - p, b = s - r, c = point(p % q, r % s);
  return point(point(a.x,b.x)%c, point(a.y,b.y)%c)/(a%b);
}


// determine if lines from a to b and c to d are parallel or
    collinear
bool LinesParallel(point a, point b, point c, point d){ // Nao
    testado
  return fabs((b-a) % (c-d)) < EPS;
}

bool LinesCollinear(point a, point b, point c, point d){ // Nao
    testado
  return LinesParallel(a, b, c, d)
      && fabs((a-b) % (a-c)) < EPS
      && fabs((c-d) % (c-a)) < EPS;
}



// Retas, equacao na forma: ax+by+c = 0; com (b=1)
struct line{
  double a, b, c;
  line(){}
  line(double m, point p){ // coeficiente e ponto sobre reta
    if (cmp(m) == 0) a = 0;
    else a = -m;
    b = 1;
    c = m*p.x-p.y;
  }
  line(point p1, point p2){ // dois pontos que estao na reta
    if (fabs(p1.x-p2.x) < EPS){
      a = 1;  b = 0;  c = -p1.x;
    }else{
      a = -(double)(p1.y-p2.y)/(p1.x-p2.x);
      b = 1;
      c = -(double)(a*p1.x)-p1.y;
    }
  }
  bool parallelTo(line s){//verdadeiro se forem paralelas
    return cmp(a, s.a) == 0 && cmp(b, s.b) == 0;
  }
  bool sameAs(line s){//verdadeiro se forem retas coincidentes
    return this->parallelTo(s) && cmp(c, s.c) == 0;
  }
  point intersectPoint(line s){
      point p;
      p.x = (s.b*c-b*s.c)/(s.a*b-a*s.b);
        if (fabs(b) > EPS) p.y = -(a*p.x+c)/b;
      else               p.y = -(s.a*p.x+s.c)/s.b;
      return p;
  }
};

//  Menor distancia entre dois pontos numa esfera de raio r
//  lat = [-90,90]; long = [-180,180]
double spherical_distance(double lt1, double lo1, double lt2,
    double lo2, double r){
  double pi = acos(-1);
  double a = pi*(lt1/180.0), b = pi*(lt2/180.0);
  double c = pi*((lo2-lo1)/180.0);
  return r*acos(sin(a)*sin(b) + cos(a)*cos(b)*cos(c));
}

// Heron's formula - area do triangulo(a,b,c) -1 se nao existe
double area_heron(double a, double b, double c) {
  if (a < b) swap(a, b);
  if (a < c) swap(a, c);
  if (b < c) swap(b, c);
  if (a > b+c) return -1;
  return sqrt((a+(b+c))*(c-(a-b))*(c+(a-b))*(a+(b-c))/16.0);
}

/*Dado dois pontos (A, B) de uma circunferencia e seu raio R,
    eh
possivel obter seus possiveis centros (C1 e C2). Para obter o
```

```
outro centro, basta inverter os paramentros */
bool circle2PtsRad(point a, point b, double r, point &c) {
  point aux = a - b;
  double d = aux * aux;
  double det = r * r/d - 0.25;
  if (det < 0.0) return false;
  double h = sqrt(det);
  c.x = (a.x + b.x) * 0.5 + (a.y - b.y) * h;
  c.y = (a.y + b.y) * 0.5 + (b.x - a.x) * h;
  return true;
}

//***[Inicio] Funcoes que usam numeros complexos para pontos***
typedef complex<double> cxpt;
struct circle {
  cxpt c; double r;
  circle(cxpt c, double r) : c(c),r(r){}
  circle(){}
};
double cross(const cxpt &a, const cxpt &b) {
  return imag(conj(a)*b);
}
double dot(const cxpt &a, const cxpt &b) {
  return real(conj(a)*b);
}

// Area da interseccao de dois circulos
double circ_inter_area(circle &a, circle &b) {
  double d = abs(b.c-a.c);
  if (d <= (b.r - a.r)) return a.r*a.r*M_PI;
  if (d <= (a.r - b.r)) return b.r*b.r*M_PI;
  if (d >= a.r + b.r) return 0;
  double A = acos((a.r*a.r+d*d-b.r*b.r)/(2*a.r*d));
  double B  = acos((b.r*b.r+d*d-a.r*a.r)/(2*b.r*d));
  return a.r*a.r*(A-0.5*sin(2*A))+b.r*b.r*(B-0.5*sin(2*B));
}

// Pontos de interseccao de dois circulos
// Intersects two circles and intersection points are in 'inter
    '
// -1-> outside, 0-> inside, 1-> tangent, 2-> 2 intersections
int circ_circ_inter(circle &a, circle &b, vector<cxpt> &inter)
    {
  double d2 = norm(b.c-a.c), rS = a.r+b.r, rD = a.r-b.r;
  if (d2 > rS*rS) return -1;
  if (d2 < rD*rD) return 0;
  double ca = 0.5*(1 + rS*rD/d2);
  cxpt z = cxpt(ca, sqrt((a.r*a.r/d2)-ca*ca));
  inter.push_back(a.c + (b.c-a.c)*z);
  if(abs(z.imag())>EPS)
    inter.push_back(a.c + (b.c-a.c)*conj(z));
  return inter.size();
}
```

```
// Line-circle intersection
// Intersects (infinite) line a-b with circle c
// Intersection points are in 'inter'
// 0 -> no intersection, 1 -> tangent, 2 -> two intersections
int line_circ_inter(cxpt a, cxpt b, circle c, vector<cxpt> &
    inter){
    c.c -= a; b -= a;
    cxpt m = b*real(c.c/b);
    double d2 = norm(m-c.c);
    if (d2 > c.r*c.r) return 0;
    double l = sqrt((c.r*c.r-d2)/norm(b));
    inter.push_back(a + m + l*b);
    if(abs(l)>EPS)
        inter.push_back(a + m - l*b);
    return inter.size();
}
//***[FIM] Funcoes que usam numeros complexos para pontos***
```

4 Polígonos 2D

4.1 Polígono 2D Library

```
/*Poligono eh representado como um array de pontos T[i] sao os
  vertices do poligono. Existe uma aresta que conecta T[i] com
      T[i+1],
  e T[size-1] com T[0]. Logo assume-se que T[0] != T[size-1]

  Poligono simples: Aquele em que as arestas nao se interceptam
Convexo: O angulo interno de T[i] com T[i-1] e T[i+1] <= 180
Concavo: Existe algum i que nao satisfaz a condicao anteriro
 */


/* Retorna a area com sinal de um poligono T
   Se area > 0, T esta listado na ordem CCW */
double signedArea(polygon& T){
  double area = 0;
  int n = T.size();
  if (n < 3) return 0;
  rep(i, 0, n)
    area += T[i] % T[(i+1)%n];
  return (area/2.0);
}

/* Retorna a area de um poligono T
   (pode ser concavo ou convexo) em O(N)*/
double poly_area(polygon& T){
  return fabs(signedArea(T));
}

/* Retorna a centroide de um poligono T em O(N)*/
point centroide(polygon &T){
  int n = T.size();
```

```
  double sgnArea = signedArea(T);

  point c = point(0,0);
  rep(i, 0, n){
    int k = (i+1)%n;
    c = c + (T[i]+T[k]) * (T[i]%T[k]);
  }
  c = c / (sgnArea * 6.0);
  return c;
}

/* Retorna o perimetro do poligono T. (pode n funcionar como
   esperado se o poligono for uma linha reta (caso degenerado))
      */
double poly_perimeter(polygon& T){
  double perimeter = 0;
  int n = T.size();
  if (n < 3) return 0;
  rep(i, 0, n)
    perimeter += abs(T[i] - T[(i+1)%n]);
  return perimeter;
}



// tests whether or not a given polygon (in CW or CCW order) is
    simple
bool isSimple(const polygon &p) { // nao testado
  for (int i = 0; i < p.size(); i++) {
    for (int k = i+1; k < p.size(); k++) {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if (i == l || j == k) continue;
      if (seg_intersect(p[i], p[j], p[k], p[l]))
        return false;
    }
  }
  return true;
}



//Retorna True se T for convexo. O(N)
bool isConvex(polygon& T){
  int n = T.size();
  if (n < 3)  return false;

  int giro = 0;
  rep(i, 0, n){ // encontra um giro valido
    int t = ccw(T[i],T[(i+1)%n], T[(i+2)%n]);
    if (t != 0) giro = t;
  }
  if (giro == 0)  return false; //todos pontos sao colineares
  rep(i, 0, n){
    int t = ccw(T[i], T[(i+1)%n], T[(i+2)%n]);
    if (t != 0 && t != giro) return false;
  }
  return true;
}

//  Determina se P pertence a T, funciona para convexo ou
    concavo
// -1 borda, 0 fora, 1 dentro. O(N)
int in_poly(point p, polygon& T){
  double a = 0; int N = T.size();
  rep(i, 0, N){
    if (between(p, T[i], T[(i+1)%N])) return -1;
    a += arg(T[i], p, T[(i+1)%N]);
  }
  return cmp(a) != 0;
}

//determina se P pertence a B, funciona APENAS para convexo
bool PointInConvexPolygon(point P, polygon &B){
  int ini = 1, fim = B.size()-2, mid, pos = -1;
  int giro = -1; // sentido horario
  while(ini<=fim){
    mid = (ini+fim)/2;
    int aux = ccw(B[0], B[mid], P);
    if(aux == giro){
      pos = mid;
      ini = mid+1;
    }else{
      fim = mid-1;
    }
  }
  if(pos == -1) return false;
  if( ccw(B[0], B[pos], P)!=giro*-1 &&
      ccw(B[0], B[pos+1], P)!=giro &&
      ccw(B[pos], B[pos+1], P)==giro) // giro || 0 na borda
    return true;
  return false;
}

//  Determina o poligono interseccao de P e Q
//  P e Q devem estar orientados anti-horario.
polygon poly_intersect(polygon& P, polygon& Q){
  int m = Q.size(), n = P.size();
  int a = 0, b = 0, aa = 0, ba = 0, inflag = 0;
  polygon R;
  while ((aa<n || ba<m) && aa<2*n && ba<2*m){
    point p1 = P[a], p2 = P[(a+1)%n], q1 = Q[b], q2 = Q[(b+1)%m
        ];
    point A = p2-p1, B = q2-q1;
    int cross=cmp(A%B), ha=ccw(p2, q2, p1),
        hb=ccw(q2, p2, q1);
    if (cross==0 && ccw(p1, q1, p2)==0 && cmp(A*B)<0){
```

```
      if (between(q1, p1, p2)) R.push_back(q1);
      if (between(q2, p1, p2)) R.push_back(q2);
      if (between(p1, q1, q2)) R.push_back(p1);
      if (between(p2, q1, q2)) R.push_back(p2);
      if (R.size() < 2) return polygon();
      inflag = 1; break;
    }
    else if (cross!=0 && seg_intersect(p1, p2, q1, q2)){
      if (inflag == 0) aa = ba = 0;
      R.push_back(line_intersect(p1, p2, q1, q2));
      inflag = (hb > 0) ? 1:-1;
    }
    if (cross==0 && hb<0 && ha<0) return R;
    bool t = cross==0 && hb==0 && ha==0;
    if (t?(inflag==1):(cross>=0)?(ha<=0):(hb>0)){
      if (inflag == -1) R.push_back(q2);
      ba++; b++; b %= m;
    }
    else {
      if (inflag == 1) R.push_back(p2);
      aa++; a++; a %= n;
    }
  }
  if (inflag == 0){
    if (in_poly(P[0], Q)) return P;
    if (in_poly(Q[0], P)) return Q;
  }
  R.erase(unique(all(R)), R.end());
  if (R.size() > 1 && R.front() == R.back()) R.pop_back();
  return R;
}
```

4.2 Convex Hull

```
/*Encontra o convex hull de um conjunto de pontos;
pivot: Ponto base para a criacao do convex hull;
radial_lt(): Ordena os pontos em sentido anti-horario (ccw).
Input: Conjunto de pontos 2D;
Output: Conjunto de pontos do convex hull em sentido ccw;

(1)Se for preciso manter pontos colineares na borda do convex
hull, essa parte evita que eles sejam removidos;
*/

point pivot;
bool radial_lt(point a, point b){
  int R = ccw(pivot, a, b); // verifica se A esta antes de B
  if (R == 0) // pivot,a,b sao colineares
    return (pivot-a)*(pivot-a) < (pivot-b)*(pivot-b);
  else
    return (R == 1); // 1 se A vem antes de B, sentido ccw
}
vector<point> convexhull(vector<point> &T){
  //  Remove pontos duplicadados
  sort(T.begin(), T.end()); //ordena por x e por y
  T.resize( unique( T.begin(), T.end() ) - T.begin() );

  int tam = 0, k, n = T.size();
  vector<point> U; // convex hull

  int P0 = min_element(T.begin(), T.end() ) - T.begin();
  //nesse caso, pivot = ponto com menor x, depois menor y
  pivot = T[P0];
  swap(T[0], T[P0]);
  sort(++T.begin(), T.end(), radial_lt);

  /*(1)*/for(k=n-2; k>=0 && ccw(T[0],T[n-1],T[k])==0; k--);
  reverse((k+1)+all(T));

  // troque <= para < para manter pontos colineares
  for(int i = 0; i < n; i++){
    while (tam > 1 && ccw(U[tam-2], U[tam-1], T[i]) <= 0)
      U.pop_back(), tam--;
    U.pb(T[i]); tam++;
  }
  return U;
}
```

4.3 Minimum Enclosing Circle

```
//Finds a circle of the minimum area enclosing a 2D point set.
typedef pair<point, double> circle; // {ponto,raio}
bool in_circle(circle C, point p){ // ponto dentro de circulo?
  return cmp(abs(p-C.first), C.second) <= 0;
}
// menor circulo que engloba o triangulo (P,Q,R)
point circumcenter(point p, point q, point r){
  point a = p-r, b = q-r, c, ret;
  c = point(a*(p+r)/2, b*(q+r)/2);
  ret=point(c%point(a.y, b.y), point(a.x, b.x)%c)/(a%b);
  return ret;
}
circle spanning_circle(vector<point>& T){
  int n = T.size();
  random_shuffle(all(T));
  circle C(point(), -INF);
  rep(i, 0, n) if(!in_circle(C, T[i])){
    C = circle(T[i], 0);
    rep(j, 0, i) if (!in_circle(C, T[j])){
      C = circle((T[i]+T[j])/2, abs(T[i]-T[j])/2);
      rep(k, 0, j) if (!in_circle(C, T[k])){
        point O = circumcenter(T[i], T[j], T[k]);
        C = circle(O, abs(O-T[k]));
      }
    }
  }
```

```
  return C;
}
```

5 Geometria 3D

5.1 Geometria 3D Library

```
// Tem algumas funcoes que possuem implementacoes distintas,
    mas com
// mesmo objetivo. Maioria nao foi testada.

#define LINE 0
#define SEGMENT 1
#define RAY 2

int sgn(double x){
  return (x > EPS) - (x < -EPS);
}

#define vec(ini, fim) (fim - ini)
struct PT{
  double x, y, z;
  PT(){x = y = z = 0;}
  PT(double x,double y, double z):x(x), y(y), z(z){}
  PT operator + (PT q){return PT(x+q.x,y+q.y,z+q.z);}
  PT operator - (PT q){return PT(x-q.x,y-q.y,z-q.z);}
  PT operator * (double d){ return PT(x*d, y*d, z*d); }
  PT operator / (double d){ return PT(x/d, y/d, z/d); }
  double dist2() const {
    return x*x+y*y+z*z;
  }
  double dist() const{
    return sqrt(dist2());
  }
  bool operator == (const PT& a) const{
    return fabs(x - a.x) < EPS && fabs(y - a.y) < EPS && fabs(z
        - a.z) < EPS;
  }
};

double dot(PT A, PT B){ // produto escalar
  return A.x*B.x + A.y*B.y + A.z*B.z;
}
PT cross(PT A, PT B){ // produto vetorial
  return PT (A.y*B.z-A.z*B.y, A.z*B.x-A.x*B.z, A.x*B.y-A.y*B.x
      );
}


inline double det(double a,double b,double c,double d){
  return a*d - b*c;
}

inline double det(double a11,double a12,double a13,double a21,
    double a22, double a23, double a31, double a32,double a33){

  return a11*det(a22,a23,a32,a33) - a12*det(a21,a23,a31,a33) +
      a13*det(a21,a22,a31,a32);
}

inline double det(const PT& a,const PT& b,const PT& c){
  return det(a.x,a.y,a.z,b.x,b.y,b.z,c.x,c.y,c.z);
}


// tamanho do vetor A
double norma(PT A){
  return sqrt(dot(A, A));
}

// distancia^2 de (a->b)
double distSq(PT a, PT b){
  return dot(a-b, a-b);
}

// Projeta vetor A sobre o vetor B
PT project(PT A, PT B){ return B * dot(A, B) / dot(B, B); }

// Verifica se existe interseccao de segmentos
// (assumir que [A,B] e [C,D] sao coplanares)
bool seg_intersect(PT A, PT B, PT C, PT D){
  return cmp(dot(cross(A-B, C-B), cross(A-B, D-B))) <= 0 &&
    cmp(dot(cross(C-D, A-D), cross(C-D, B-D))) <= 0;
}



// square distance between point and line, ray or segment
double ptLineDistSq(PT s1, PT s2, PT p, int type){
  double pd2 = distSq(s1, s2);
  PT r;
  if(pd2 == 0)
    r = s1;
  else{
    double u = dot(p-s1, s2-s1) / pd2;
    r = s1 + (s2 - s1)*u;
    if(type != LINE && u < 0.0)
      r = s1;
    if(type == SEGMENT && u > 1.0)
      r = s2;
  }
  return distSq(r, p);
}


//  Distancia de ponto P ao segmento [A,B]
double dist_point_seg(PT P, PT A, PT B){
```

```
  PT PP = A + project(P-A, B-A);
  if (cmp(norma(A-PP) + norma(PP-B), norma(A-B)) == 0)
    return norma(P-PP);//distance point-line!
  else
    return min(norma(P-A), norma(P-B));
}


// Distance between lines ab and cd. TODO: Test this
double lineLineDistance(PT a, PT b, PT c, PT d) {
  PT v1 = b-a;
  PT v2 = d-c;
  PT cr = cross(v1, v2);
  if (dot(cr, cr) < EPS) {
    PT proj = v1*(dot(v1, c-a)/dot(v1, v1));
    return sqrt(dot(c-a-proj, c-a-proj));
  } else {
    PT n = cr/sqrt(dot(cr, cr));
    PT p = dot(n, c - a);
    return sqrt(dot(p, p));
  }
}



//  Menor distancia do segmento [A,B] ao segmento [C,D] (lento
    *)
#define dps dist_point_seg
double dist_seg_seg(PT A, PT B, PT C, PT D){
  PT E = project(A-D, cross(B-A, D-C));
  // distance between lines!
  if (seg_intersect(A, B, C+E, D+E)) {
    return norma(E);
  }else {
    double dA = dps(A,C,D), dB = dps(B,C,D);
    double dC = dps(C,A,B), dD = dps(D,A,B);
    return min(min(dA, dB), min(dC, dD));
  }
}

//  Menor distancia do segmento [A,B] ao segmento [C,D] (rapido
    *)
double dist_seg_seg2(PT A, PT B, PT C, PT D){
  PT u(B-A), v(D-C), w(A-C);
  double a = dot(u, u), b = dot(u, v);
  double c = dot(v, v), d = dot(u, w), e = dot(v, w);

  double DD = a*c - b*b ;
  double sc, sN, sD = DD;
  double tc, tN, tD = DD;
  if (DD < EPS){
    sN = 0, sD = 1, tN = e, tD = c ;
  }else{
    sN = (b*e - c*d);
    tN = (a*e - b*d);
    if (sN < 0){
      sN = 0, tN = e, tD = c;
    }else if(sN > sD){
      sN = sD, tN = e+b, tD = c;
    }
  }
  if (tN < 0){
    tN = 0 ;
    if (-d < 0) sN = 0 ;
    else if(-d > a) sN = sD;
    else{
      sN = -d;
      sD = a ;
    }
  }else if(tN > tD){
    tN = tD ;
    if((-d + b) < 0) sN = 0 ;
    else if(-d + b > a) sN = sD;
    else{
      sN = -d + b ;
      sD = a ;
    }
  }
  sc = fabs(sN) < EPS ? 0 : sN/sD;
  tc = fabs(tN) < EPS ? 0 : tN/tD;
  PT dP = w + (u*sc) - (v*tc);
  return norma(dP);
}

//  Distancia de Ponto a Triangulo, dps = dist_point_seg
double dist_point_tri(PT P, PT A, PT B, PT C){
  PT N = cross(B-A, C-A);
  PT PP = P - project(P-A, N);
  PT R1, R2, R3;
  R1 = cross(B-A, PP-A);
  R2 = cross(C-B, PP-B);
  R3 = cross(A-C, PP-C);

  if (cmp(dot(R1,R2))>=0 && cmp(dot(R2,R3))>=0 &&
      cmp(dot(R3,R1))>=0) {
    return norma(P-PP);
  }
  else {
    return min(dps(P,A,B), min(dps(P,B,C), dps(P,A,C)));
  }
}


// compute a, b, c, d such that all points lie on ax + by + cz
    = d. TODO: test this
double planeFromPts(PT p1, PT p2, PT p3, double& a, double& b,
    double& c, double& d) {
  PT normal = cross(p2-p1, p3-p1);
```

```
  a = normal.x; b = normal.y; c = normal.z;
  d = -a*p1.x-b*p1.y-c*p1.z;
}

// project point onto plane. TODO: test this
PT ptPlaneProj(PT p, double a, double b, double c, double d) {
  double l = (a*p.x+b*p.y+c*p.z+d)/(a*a+b*b+c*c);
  return PT(p.x-a*l, p.y-b*l, p.z-c*l);
}

// distance from point p to plane aX + bY + cZ + d = 0
double ptPlaneDist(PT p, double a, double b, double c, double d
    ){
  return fabs(a*p.x + b*p.y + c*p.z + d) / sqrt(a*a + b*b + c*c
      );
}

// distance between parallel planes aX + bY + cZ + d1 = 0 and
// aX + bY + cZ + d2 = 0
double planePlaneDist(double a, double b, double c, double d1,
    double d2){
  return fabs(d1 - d2) / sqrt(a*a + b*b + c*c);
}

// Volume de Tetraedro
double signedTetrahedronVol(PT A, PT B, PT C, PT D) {
  double A11 = A.x - B.x;
  double A12 = A.x - C.x;
  double A13 = A.x - D.x;
  double A21 = A.y - B.y;
  double A22 = A.y - C.y;
  double A23 = A.y - D.y;
  double A31 = A.z - B.z;
  double A32 = A.z - C.z;
  double A33 = A.z - D.z;
  double det =
    A11*A22*A33 + A12*A23*A31 +
    A13*A21*A32 - A11*A23*A32 -
    A12*A21*A33 - A13*A22*A31;
  return det / 6;
}

// Parameter is a vector of vectors of points - each interior
    vector
// represents the 3 points that make up 1 face, in any order.
// Note: The polyhedron must be convex, with all faces given as
     triangles.
double polyhedronVol(vector<vector<PT> > poly) {
  int i,j;
  PT cent(0,0,0);
  for (i=0; i<poly.size(); i++)
    for (j=0; j<3; j++)
      cent=cent+poly[i][j];
  cent=cent*(1.0/(poly.size()*3));
  double v=0;
  for (i=0; i<poly.size(); i++)
    v+=fabs(signedTetrahedronVol(cent,poly[i][0],poly[i][1],
        poly[i][2]));
  return v;
}
```

```
// Outras implementacoes [Usa struct PT]

struct line{ // reta definida por um ponto p e direcao v
  PT p, v;
  line(){};
  line(const PT& p,const PT& v):p(p),v(v){
    assert(!(v == PT()));
  }
  bool on(const PT& pt) const{
    return cross(pt - p, v) == PT();
  }
};
struct plane {
  PT n;
  double d;
  plane() : d(0) {}
  plane(const PT &p1, const PT &p2,
      const PT &p3) {
    n = cross(p2 - p1, p3 - p1);
    d = -dot(n, p1);
    assert(side(p1) == 0);
    assert(side(p2) == 0);
    assert(side(p3) == 0);
  }
  int side(const PT &p) const {
    return sgn(dot(n, p) + d);
  }
};

// interesecao de retas
int intersec(const line& l1, const line& l2,
    PT& res){
  assert(!(l1.v == PT()));
  assert(!(l2.v == PT()));
  if (cross(l1.v,l2.v) == PT()){
    if (cross(l1.v, l1.p - l2.p) == PT())
      return 2; // same
    return 0; // parallel
  }
  PT n = cross(l1.v,l2.v);
  PT p = l2.p - l1.p;
  if (sgn(dot(n,p)))
    return 0; // skew
  double t;
```

```
  if (sgn(n.x))
    t = (p.y * l2.v.z - p.z * l2.v.y) / n.x;
  else if (sgn(n.y))
    t = (p.z * l2.v.x - p.x * l2.v.z) / n.y;
  else if (sgn(n.z))
    t = (p.x * l2.v.y - p.y * l2.v.x) / n.z;
  else
    assert(false);
  res = l1.p + l1.v * t;
  assert(l1.on(res)); assert(l2.on(res));
  return 1; // intersects
}

// distancia entre 2 retas
double dist(const line& l1,const line& l2){
  PT ret = l1.p - l2.p;
  ret = ret - l1.v * (dot(l1.v,ret) / l1.v.dist2());
  PT tmp = l2.v - l1.v *
    (dot(l1.v,l2.v) / l1.v.dist2());
  if (sgn(tmp.dist2()))
    ret = ret - tmp * (dot(tmp,ret) / tmp.dist2());
  assert(fabs(dot(ret,l1.v)) < eps);
  assert(fabs(dot(ret,tmp)) < eps);
  assert(fabs(dot(ret,l2.v)) < eps);
  return ret.dist();
}

// Retorna os dois pontos mais proximos entre l1 e l2
void closest(const line& l1,const line& l2,
    PT& p1,PT& p2){
  if (cross(l1.v,l2.v) == PT()){
    p1 = l1.p;
    p2 = l2.p - l1.v *
      (dot(l1.v,l2.p - l1.p) / l1.v.dist2());
    return;
  }
  PT p = l2.p   - l1.p;
  double t1 = (
      dot(l1.v,p) * l2.v.dist2() -
      dot(l1.v,l2.v) * dot(l2.v,p)
        ) / cross(l1.v,l2.v).dist2();
  double t2 = (
      dot(l2.v,l1.v) * dot(l1.v,p) -
      dot(l2.v,p) * l1.v.dist2()
        ) / cross(l2.v,l1.v).dist2();
  p1 = l1.p + l1.v * t1;
  p2 = l2.p + l2.v * t2;
  assert(l1.on(p1));
  assert(l2.on(p2));
}

//retorna a intersecao de reta com plano [retorna 1 se
    intersecao for pt]
int cross(const line &l, const plane &pl,
    PT &res) {
  double d = dot(pl.n, l.v);
  if (sgn(d) == 0) {
    return (pl.side(l.p) == 0) ? 2 : 0;
  }
  double t = (-dot(pl.n, l.p) - pl.d) / d;
  res = l.p + l.v * t;
#ifdef DEBUG
  assert(pl.side(res) == 0);
#endif
  return 1;
}

bool cross(const plane& p1,const plane& p2,
    const plane& p3, PT& res){
  double d = det(p1.n,p2.n,p3.n);
  if (sgn(d) == 0) {
    return false;
  }
  PT px(p1.n.x, p2.n.x, p3.n.x);
  PT py(p1.n.y, p2.n.y, p3.n.y);
  PT pz(p1.n.z, p2.n.z, p3.n.z);
  PT p(-p1.d,-p2.d,-p3.d);
  res = PT(
      det(p,py,pz)/d,
      det(px,p,pz)/d,
      det(px,py,p)/d
    );
#ifdef DEBUG
  assert(p1.side(res) == 0);
  assert(p2.side(res) == 0);
  assert(p3.side(res) == 0);
#endif
  return true;
}

// retorna reta da intersecao de dois planos
int cross(const plane &p1, const plane &p2,
    line &res) {
  res.v = cross(p1.n, p2.n);
  if (res.v == PT()) {
    if ( (p1.n * (p1.d / p1.n.dist2())) ==
        (p2.n * (p2.d / p2.n.dist2())) )
      return 2;
    else
      return 0;
  }
  plane p3;
  p3.n = res.v;
  p3.d = 0;
  bool ret = cross(p1, p2, p3, res.p);
  assert(ret);
  assert(p1.side(res.p) == 0);
  assert(p2.side(res.p) == 0);
```

```
  return 1;
}

// testes
int main(){
  {
    line l;
    l.p = PT(1, 1, 1);
    l.v = PT(1, 0, -1);
    plane p(PT(10, 11, 12), PT(9, 8, 7), PT(1, 3, 2));
    PT res;
    assert(cross(l, p, res) == 1);
  }
  {
    plane p1(PT(1, 2, 3), PT(4, 5, 6), PT(-1, 5, -4));
    plane p2(PT(3, 2, 1), PT(6, 5, 4), PT(239, 17, -42));
    line l;
    assert(cross(p1, p2, l) == 1);
  }
  {
    plane p1(PT(1, 2, 3), PT(4, 5, 6), PT(-1, 5, -4));
    plane p2(PT(1, 2, 3), PT(7, 8, 9), PT(3, -1, 10));
    line l;
    assert(cross(p1, p2, l) == 2);
  }
  {
    plane p1(PT(1, 2, 3), PT(4, 5, 6), PT(-1, 5, -4));
    plane p2(PT(1, 2, 4), PT(4, 5, 7), PT(-1, 5, -3));
    line l;
    assert(cross(p1, p2, l) == 0);
  }

  line l1,l2;
  while (l1.p.load())
  {
    l1.v.load(); l1.v = l1.v - l1.p;
    l2.p.load();
    l2.v.load(); l2.v = l2.v - l2.p;
    if (l1.v == PT() || l2.v == PT()) continue;
    PT res;
    int cnt = intersec(l1,l2,res);
    double d = dist(l1,l2);
    if (fabs(d) < eps)
      assert(cnt >= 1);
    else
      assert(cnt == 0);
    PT p1,p2;
    closest(l1,l2,p1,p2);
    assert(fabs((p1-p2).dist() - d) < eps);
  }
  plane a(PT(1,0,0),PT(0,1,0),PT(0,0,1));
  plane b(PT(-1,0,0),PT(0,-1,0),PT(0,0,-1));
  line l;
  assert((cross(a,b,l))==0);
  return 0;
}
```

6 Grafos

6.1 Topological Sort

```
vector<int> adj[MAXV];
void topo_sort(int N){
  queue<int> q; // priority_queue?
  vector<int> in_degree(N,0);

    rep(i, 0, N) rep(j, 0, adj[i].size())
  in_degree[ adj[i][j] ] ++;

  rep( i, 0, n ) if (in_degree[i] == 0) q.push(i);
  while( !q.empty() ){
    int u = q.front();  q.pop();
    rep(i, 0, adj[u].size()){
      int v = adj[u][i];
      if ( --in_degree[v] == 0 ) q.push(v);
    }
  }
}
```

6.2 Dijkstra

```
#define MAXV 100000
int dist[MAXV], pi[MAXV];// dist from s and pointer to parent
vector<ii> adj[MAXV]; //edge = {v, dist}
int dijkstra(int s, int t, int n){
  priority_queue<ii> pq;
  memset(pi, -1, sizeof pi);
  memset(dist, INF, sizeof dist); // INF eh suficiente?
  pq.push(ii(dist[s]=0, s));
  while (!pq.empty()){
    ii top = pq.top(); pq.pop();
    int u = top.second, d = -top.first;
    if (d != dist[u]) continue;
    if (u == t) break; // terminou antes
    rep(i,0,(int)adj[u].size()){
      int v = adj[u][i].F;
      int cost = adj[u][i].S;
      if (dist[v] > dist[u] + cost){
        dist[v] = dist[u] + cost;
        pi[v] = u;
        pq.push(ii(-dist[v], v));
      }
    }
  }
  return dist[t];
```

```
}
```

6.3 Floyd-Warshall

```
int adj[400][400], path[400][400];
int n, m; // #vertices, #arestas
// adj[u][v] = custo de {U->V}
// path[u][v] = k .: K vem logo apos U no caminho ate V
void read_graph(){
  memset(adj,INF,sizeof adj); // para menor caminho
  rep(i, 0, n) adj[i][i] = 0; // para menor caminho
  int u, v, w;
  rep(i, 0, m){
    cin >> u >> v >> w;
    adj[u][v] = w;
    path[u][v] = v;
  }
}
void floyd(){
  rep(k, 0, n)
    rep(i, 0, n)
      rep(j, 0, n)
        if(adj[i][k] + adj[k][j] < adj[i][j]){
          adj[i][j] = adj[i][k] + adj[k][j];
          path[i][j] = path[i][k];
        }
}
vector<int> findPath(int s, int d){
  vector<int> Path; Path.pb(s);
  while (s != d){ s = path[s][d]; Path.pb(s); }
  return Path;
}

/*Aplicacoes:
1-Encontrar o fecho transitivo (saber se U consegue visitar V)
.: adj[u][v] |= (adj[u][k] & adj[k][v]);
   (inicializar adj com 0)

2-Minimizar a maior aresta do caminho entre U e V
.: adj[u][v] = min(adj[u][v], max(adj[u][k], adj[k][v]));
   (inicializar adj com INF)

3-Maximizar a menor aresta do caminho entre U e V
.: adj[u][v] = max(adj[u][v], min(adj[u][k], adj[k][u]));
   (inicializar adj com -INF)*/
```

6.4 Bellman-Ford

```
#define MAXV 400
//Vertices indexados em 0.
int V, E;//#vertices, #arestas
vector<ii> adj[MAXV];
ll dist[MAXV];
//Menor custo de S para todos vertices em O(V^3)
//bellman() retorna FALSE se o grafo tem ciclo com custo
    negativo
//dist[v] contem o menor custo de s ate v.
bool bellman(int s){
  rep(i,0,V) dist[i] = INF;
  dist[s] = 0;
  rep(i,0,V-1)
    rep(u,0,V){
      rep(j,0,adj[u].size()){
        int v = adj[u][j].F, duv = adj[u][j].S;
        dist[v] = min(dist[v], dist[u] + duv);
      }
    }
  // verifica se tem ciclo com custo negativo
  rep(u,0,V)
    rep(j,0,adj[u].size()){
      int v = adj[u][j].F, duv = adj[u][j].S;
      if (dist[v] > dist[u] + duv) return false;
    }
  return true;
}
```

6.5 Vértices de Articulação e Pontes

```
#define MAXV 100000
vector<int> adj[MAXV];
int dfs_num[MAXV], dfs_low[MAXV], dfs_parent[MAXV];
int dfscounter, V, dfsRoot, rootChildren, ans;
int articulation[MAXV], articulations;
vector<ii> bridges;

void articulationPointAndBridge(int u){
  dfs_low[u] = dfs_num[u] = dfscounter++;
  rep(i,0,adj[u].size()){
    int v = adj[u][i];
    if (dfs_num[v] == -1){
      dfs_parent[v] = u;
      if (u == dfsRoot) rootChildren++;

      articulationPointAndBridge(v);
      if(dfs_low[v] >= dfs_num[u]) articulation[u] = true;
      if (dfs_low[v] > dfs_num[u]) bridges.pb(mk(u,v));

      dfs_low[u] = min(dfs_low[u], dfs_low[v]);
    }
    else if (v != dfs_parent[u])
      dfs_low[u] = min(dfs_low[u], dfs_num[v]);
  }
}

int main(){
```

```
  // read graph
  dfscounter = 0;
  rep(i,0,V){
    dfs_low[i] = dfs_parent[i] = articulation[i] = 0;
    dfs_num[i] = -1;
  }
  articulations = 0;
  bridges.clear();
  rep(i,0,V)
    if (dfs_num[i] == -1){
      dfsRoot = i; rootChildren = 0;
      articulationPointAndBridge(i);
      articulation[dfsRoot] = (rootChildren > 1);
    }
  printf("#articulations = %d\n",articulations);
  rep(i,0,V)
    if(articulation[i]) printf("Vertex %d\n", i);
  printf("#bridges = %d\n",bridges.size());
  rep(i,0,bridges.size())
    printf("Bridge %d<->%d\n", bridges[i].F, bridges[i].S);
  return 0;
}
```

6.6 Tarjan

```
#define MAXV 100000
vector<int> adj[MAXV]; int V;
int dfs_num[MAXV], dfs_low[MAXV], vis[MAXV], SCC[MAXV];
int dfsCounter, numSCC;
vector<int> S; //global variables

void tarjanSCC(int u){
  dfs_low[u] = dfs_num[u] = dfsCounter++; //dfs_low[u] <=
      dfs_num[u]
  S.push_back(u); //stores u in a vector based on order of
      visitation
  vis[u] = 1;
  rep(i,0,adj[u].size()){
    int v = adj[u][i];
    if (dfs_num[v] == -1)
      tarjanSCC(v);
    if (vis[v]) //condition for update
      dfs_low[u] = min(dfs_low[u], dfs_low[v]);
  }
  if (dfs_low[u] == dfs_num[u]){ //if this is a root (start) of
       an SCC
    while(42){
      int v = S.back(); S.pop_back();
      vis[v] = 0;
      SCC[v] = numSCC; //wich SCC this vertex belong
      if (u == v) break;
    }
    numSCC++;
  }
}

int main(){
  // read graph
  rep(i,0,V){
    dfs_num[i] = -1;
    dfs_low[i] = vis[i] = 0;
    SCC[i] = -i;
  }
  dfsCounter = numSCC = 0;
  rep(i,0,V)
    if(dfs_num[i] == -1)
      tarjanSCC(i);

  return 0;
}
```

6.7 Kosaraju

```
#define MAXV 100000
vector<int> adj[2][MAXV]; // adj[0][] original, adj[1][]
    transposto
vector<int> S, dfs_num;
int N, numSCC, SCC[MAXV];

void Kosaraju(int u, int t, int comp){
  Wdfs_num[u] = 1;
  if (t == 1) SCC[u] = comp;
  for (int j = 0; j < (int)adj[t][u].size(); j++) {
    int v = adj[t][u][j];
    if (dfs_num[v] == DFS_WHITE) Kosaraju(v, t, comp);
  }
  S.push_back(u);
}
void doit(){ // chamar na main
  S.clear();
  dfs_num.assign(N, DFS_WHITE);
  for (int i = 0; i < N; i++)
    if (dfs_num[i] == DFS_WHITE) Kosaraju(i, 0, -1);
    numSCC = 0;
  dfs_num.assign(N, DFS_WHITE);
  for (int i = N - 1; i >= 0; i--)
    if (dfs_num[ S[i] ] == DFS_WHITE) {
      Kosaraju(S[i], 1, numSCC);
      numSCC++;
    }
  printf("There are %d SCCs\n", numSCC);
}
```

6.8 2-Sat

```
// 2-sat - Codigo do problema X-Mart
// vertices indexado em 1
vector<int> adj[2*MAXV];
vector<int> radj[2*MAXV];
int seen[2 * MAXV], comp[2 * MAXV], order[2 * MAXV], ncomp,
    norder;
int N; // #variaveis
int n; // #vertices

#define NOT(x) ( (x<=N)? (x+N) : (x-N) )
#define quero 1
void add_edge(int a, int b, int opcao){
  if (a > b) swap(a, b);
  if (b == 0) return;
  if (a == 0){
    if (opcao == quero) adj[NOT(b)].pb(b);
    else                adj[b].pb(NOT(b));
  }else{ // normal...
    if (opcao == quero){
      adj[NOT(a)].pb(b);
      adj[NOT(b)].pb(a);
    }else{
      a = NOT(a); b = NOT(b);
      adj[NOT(a)].pb(b);
      adj[NOT(b)].pb(a);
    }
  }
}
void init(){
  rep(i, 0, n+1){
    adj[i].clear();
    radj[i].clear();
  }
}
void dfs1(int u) {
  seen[u] = 1;
  rep(i,0,adj[u].size()) if (!seen[adj[u][i]]) dfs1( adj[u][i]
      );
  order[norder++] = u;
}
void dfs2(int u) {
  seen[u] = 1;
  rep(i,0,radj[u].size()) if (!seen[radj[u][i]]) dfs2( radj[u][
      i] );
  comp[u] = ncomp;
}
void strongly_connected_components(){
  rep(v,1,n+1) rep(i,0,(int)adj[v].size()) radj[ adj[v][i] ].pb
      (v);

  norder = 0; memset(seen, 0, sizeof seen);
  rep(v,1,n+1) if (!seen[v]) dfs1(v);

  ncomp = 0;  memset(seen, 0, sizeof seen);
  for (int i = n-1, u = order[n-1]; i>=0; u = order[--i])
    if (!seen[u]) { dfs2(u);  ncomp++; }
}
bool sat2(){
  strongly_connected_components();
  rep(i,1,n+1) if (comp[i] == comp[NOT(i)]) return false;
  return true;
}
int main(){
  int Clientes;
  while (cin >> Clientes >> N){
    if (Clientes == 0 && N==0) break;
    n = 2*N;
    init();
    int u, v;
    rep(i, 0, Clientes){
      scanf("%d %d", &u, &v);
      add_edge(u, v, quero);
      scanf("%d %d", &u, &v);
      add_edge(u, v, !quero);
    }
    sat2() ? printf("yes\n") : else printf("no\n");
  }
  return 0;
}
```

6.9 LCA

```
/*Lowest Common Ancestor (LCA) entre dois vertices A, B de uma
arvore. LCA(A,B) = ancestral mais proximo de A e B.
O codigo abaixo tambem calcula a menor aresta do caminho entre
A e B.
Para saber quantas arestas tem entre A e B basta fazer:
  level[A]+level[B]-2*level[lca(A,B)] Pode-se modificar para
  retorna a distancia entre A e B.
Como usar:
(1) ler a arvore em E[] e W[], chamar doit(raiz),
passando a raiz da arvore. Indexar em 0 os vertices
(2) A funcao retorna o LCA e a menor aresta entre A e B.
*/

#define MAXV 101000
const int maxl = 20; // profundidade maxima 2^(maxl) > MAXV
int pai[MAXV][maxl+1]; // pai[v][i] = pai de v subindo 2^i
    arestas
int dist[MAXV][maxl+1]; // dist[v][i] = menor aresta de v
    subindo 2^i arestas
int level[MAXV]; // level[v] = #arestas de v ate a raiz

int N, M; // numero de vertices e arestas
vector<pair<int,int> > E[MAXV]; // {v,custo}

void dfs(int v, int p, int peso){
```

```
  level[v] = level[p] + 1;
  pai[v][0] = p;
  dist[v][0] = peso; // aresta de v->pai[v]
  for (int i = 1; i <= maxl; i++){
    pai[v][i] = pai[ pai[v][i-1] ][i-1]; // subindo 2^i arestas
    dist[v][i] = min(dist[v][i-1] , dist[ pai[v][i-1] ][i-1] );
        //(||)
  }
    rep(i, 0, E[v].size()){
    int viz = E[v][i].first;
    int cost = E[v][i].second;
    if (viz == p) continue;
    dfs(viz, v, cost);
  }
}

void doit(int root){
  level[root] = 0;
  for (int i = 0; i <= maxl; i++)
    pai[root][i] = root, dist[root][i] = INF;
  rep(i, 0, E[root].size()){
      int viz = E[root][i].first;
      int cost = E[root][i].second;
    dfs(viz, root, cost);
  }
}

pair<int,int> lca(int a, int b){
  int menor = INF; // valor da menor aresta do caminho a->b
  if (level[a] < level[b]) swap(a, b);

  for (int i = maxl; i >= 0; i--){
    if (level[ pai[a][i] ] >= level[b]){
      menor = min(menor, dist[a][i]);
      a = pai[a][i];
    }
  }
  if (a != b){
    for (int i = maxl; i >= 0; i--){
      if (pai[a][i] != pai[b][i]){
        menor = min(menor, min(dist[a][i], dist[b][i]));
        a = pai[a][i];    b = pai[b][i];
      }
    }
    //ultimo salto
    menor = min(menor, min(dist[a][0], dist[b][0]));
    a = pai[a][0];  b = pai[b][0];
  }
  return make_pair(a, menor);
}
```

6.10 Maximum Bipartite Matching

```
// Encontra o casamento bipartido maximo entre os vertices
// grupo X e Y. x = [0,X-1], y = [0,Y-1]
// match[y] = x - contem quem esta casado com y
// Teorema de Konig - Num grafo bipartido, o matching eh igual
    ao minimum vertex cover
// Complexidade O(nm)
#define MAXV 1000
vector<int> adj[MAXV];
int match[MAXV], V, X, Y;
bool vis[MAXV];

int aug(int v){
  if(vis[v])  return 0;
  vis[v] = true;
  rep(i,0,adj[v].size()){
    int r = adj[v][i];
    if (match[r] == -1 || aug(match[r])){
      match[r] = v; // augmenting path
      return 1;
    }
  }
  return 0;
}
int matching(int X, int Y){
  int V = X + Y;
  rep(i,0,V) match[i] = -1;
  int mcbm = 0;
  rep(i,0,X){
    rep(j,0,X) vis[j] = false;
    mcbm += aug(i);
  }
  return mcbm;
}
```

6.11 Hopcroft Karp - Maximum Bipartite Matching

```
/*Encontra o casamento bipartido maximo de forma eficiente
1) Chamar a funcao init(L,R) L: #vertices da esquerda, R: #
    vertices da direita
2) Usar addEdge(Li,Ri) para adicionar a aresta Li -> Ri
3) maxMatching() retorna o casamento maximo
matching[]: matching[Rj] -> armazena Li
Complexidade do algoritmo:
O(sqrt(n) * m)    */
#define MAXN1 3010
#define MAXN2 3010
#define MAXM 6020
int n1, n2, edges, last[MAXN1], prev[MAXM], head[MAXM];
int matching[MAXN2], dist[MAXN1], Q[MAXN1];
bool used[MAXN1], vis[MAXN1];

void init(int L, int R){
  n1 = L, n2 = R;
```

```
  edges = 0;
  fill(last, last + n1, -1);
}
void addEdge(int u, int v){
  head[edges] = v;
  prev[edges] = last[u];
  last[u] = edges++;
}
void bfs(){
  fill(dist, dist + n1, -1);
  int sizeQ = 0;
  for (int u = 0; u < n1; ++u){
    if (!used[u]){
      Q[sizeQ++] = u;
      dist[u] = 0;
    }
  }
  for (int i = 0; i < sizeQ; i++){
    int u1 = Q[i];
    for (int e = last[u1]; e >= 0; e = prev[e]){
      int u2 = matching[head[e]];
      if (u2 >= 0 && dist[u2] < 0){
        dist[u2] = dist[u1] + 1;
        Q[sizeQ++] = u2;
      }
    }
  }
}
bool dfs(int u1){
  vis[u1] = true;
  for (int e = last[u1]; e >= 0; e = prev[e]){
    int v = head[e];
    int u2 = matching[v];
    if (u2 < 0 || !vis[u2] && dist[u2] == dist[u1] + 1 && dfs(u
        2)){
      matching[v] = u1;
      used[u1] = true;
      return true;
    }
  }
  return false;
}
int maxMatching(){
  fill(used, used + n1, false);
  fill(matching, matching + n2, -1);
  for (int res = 0;;){
    bfs();
    fill(vis, vis + n1, false);
    int f = 0;
    for (int u = 0; u < n1; ++u)
      if (!used[u] && dfs(u)) ++f;
    if (!f) return res;
    res += f;
  }
}
```

6.12 Network Flow (lento)

```
#define MAXV 250
vector<int> edge[MAXV];
int cap[MAXV][MAXV];
bool vis[MAXV];

void init() {
  rep(i,0,MAXV) edge[i].clear();
  memset(cap, 0, sizeof cap);
}

void add(int a, int b, int cap_ab, int cap_ba){
  edge[a].pb(b), edge[b].pb(a);
  cap[a][b] += cap_ab, cap[b][a] += cap_ba;
}

int dfs(int src, int snk, int fl){
  if(vis[src]) return 0;
  if(snk == src) return fl;
  vis[src] = 1;
  rep(i,0,edge[src].size()){
    int v = edge[src][i];
    int x = min(fl, cap[src][v]);
    if(x > 0){
      x = dfs(v, snk, x);
      if( !x ) continue;
      cap[src][v] -= x;
      cap[v][src] += x;
      return x;
    }
  }
  return 0;
}

int flow(int src, int snk) {
  int ret = 0;
  while( 42 ) {
    memset(vis, 0, sizeof vis);
    int delta = dfs(src, snk, 1<<30);
    if(!delta) break;
    ret += delta;
  }
  return ret;
}
```

6.13 Network Flow - Dinic

```
/*
  Grafo indexado em 1
```

```
    Inicializar maxN, maxE.
    Chamar init( ) com #nos e source e sink
    montar o grafo chamando add(a,b,c1,c2), sendo c1 cap. de a
        ->b e c2 cap. de b->a
*/
#define FOR(i,a,b) for(int i=a; i<=b; i++)
#define SET(c, v) memset(c, v, sizeof c)
const int maxN = 5000;
const int maxE = 70000;
const int inf = 1000000005;

int nnode, nedge, src, snk;
int Q[ maxN ], pro[ maxN ], fin[ maxN ], dist[ maxN ];
int flow[ maxE ], cap[ maxE ], to[ maxE ], next[ maxE ];

void init( int _nnode, int _src, int _snk ) {
  nnode = _nnode, nedge = 0, src = _src, snk = _snk;
  FOR(i,1,nnode) fin[i] = -1;
}

void add( int a, int b, int c1, int c2 ) {
  to[nedge]=b, cap[nedge]=c1, flow[nedge]=0, next[nedge]=fin[a
      ], fin[a]=nedge++;
  to[nedge]=a, cap[nedge]=c2, flow[nedge]=0, next[nedge]=fin[b
      ], fin[b]=nedge++;
}

bool bfs() {
  SET( dist, -1 );
  dist[src] = 0;
  int st = 0, en = 0;
  Q[en++] = src;
  while( st < en ) {
    int u = Q[ st++ ];
    for(int e = fin[u]; e >= 0; e = next[e] ) {
      int v = to[e];
      if( flow[e] < cap[e] && dist[v] == -1 ) {
        dist[v] = dist[u] + 1;
        Q[en++] = v;
      }
    }
  }
  return dist[snk] != -1;
}

int dfs(int u, int fl) {
  if( u == snk ) return fl;
  for( int& e = pro[u]; e >= 0; e = next[e] ) {
    int v = to[e];
    if( flow[e] < cap[e] && dist[v] == dist[u]+1 ) {
      int x = dfs( v, min( cap[e] - flow[e] , fl ) );
      if( x > 0 ) {
        flow[ e ] += x, flow[ e^1 ] -= x;
        return x;
      }
    }
  }
  return 0;
}

LL dinic() {
  LL ret = 0;
  while( bfs() ) {
    FOR(i, 1, nnode) pro[i] = fin[i];
    while( 42 ) {
      int delta = dfs( src, inf );
      if( !delta ) break;
      ret += delta;
    }
  }
  return ret;
}
```

6.14 Min Cost Max Flow

```
// Criar o grafo chamando MCMF g(V), onde g eh o grafo e V a
    qtde de vertices (indexado em 0)
// chamar g.add(u,v,cap,cost) para add a aresta u->v, se for
    bidirecional, chamar denovo .: g.add(v,u,cap,cost)
struct MCMF {
  typedef int ctype;
  enum { MAXN = 550, INF = INT_MAX };
  struct Edge { int x, y; ctype cap, cost; };
  vector<Edge> E;       vector<int> adj[MAXN];
  int N, prev[MAXN];    ctype dist[MAXN], phi[MAXN];

  MCMF(int NN) : N(NN) {}

  void add(int x, int y, ctype cap, ctype cost) {  // cost >= 0
    Edge e1={x,y,cap,cost}, e2={y,x,0,-cost};
    adj[e1.x].push_back(E.size()); E.push_back(e1);
    adj[e2.x].push_back(E.size()); E.push_back(e2);
  }

  void mcmf(int s, int t, ctype &flowVal, ctype &flowCost) {
    int x;
    flowVal = flowCost = 0;  memset(phi, 0, sizeof(phi));
    while (true) {
      for (x = 0; x < N; x++) prev[x] = -1;
      for (x = 0; x < N; x++) dist[x] = INF;
      dist[s] = prev[s] = 0;

      set< pair<ctype, int> > Q;
      Q.insert(make_pair(dist[s], s));
      while (!Q.empty()) {
        x = Q.begin()->second; Q.erase(Q.begin());
        // x is a vertex
```

```
        // FOREACH(it, adj[x]) { // adj[] stores the index of a
             edge
        for (vector<int> :: iterator it = adj[x].begin(); it !=
             adj[x].end(); it++){
          const Edge &e = E[*it];
          if (e.cap <= 0) continue;
          ctype cc = e.cost + phi[x] - phi[e.y];
                                  // ***
          if (dist[x] + cc < dist[e.y]) {
            Q.erase(make_pair(dist[e.y], e.y));
            dist[e.y] = dist[x] + cc;
            prev[e.y] = *it;
            Q.insert(make_pair(dist[e.y], e.y));
          }
        }
      }
      if (prev[t] == -1) break;

      ctype z = INF;
      for (x = t; x != s; x = E[prev[x]].x) z = min(z, E[prev[x
          ]].cap);
      for (x = t; x != s; x = E[prev[x]].x)
        { E[prev[x]].cap -= z; E[prev[x]^1].cap += z; }
      flowVal += z;
      flowCost += z * (dist[t] - phi[s] + phi[t]);
      for (x = 0; x < N; x++) if (prev[x] != -1) phi[x] += dist
          [x];      // ***
    }
  }
};
```

6.15 Min Cost Max Flow (Stefano)

```
#define MAX_V 2003
#define MAX_E 2*3003

/*
   Inicializar MAX_V e MAX_E corretamente
   chamar init(_V) com a qtde de vertices (indexado em 0)
   mesmo que seja bidirecional, add as arestas duas vezes no
       main()
   Complexiade (?)
 */

typedef int cap_type;
typedef long long cost_type;
const cost_type INF = LLONG_MAX;

int V,E,prev[MAX_V],last[MAX_V],to[MAX_E],next[MAX_E];
bool visited[MAX_V];
cap_type flowVal, cap[MAX_E];
cost_type flowCost,cost[MAX_E],dist[MAX_V],pot[MAX_V];

void init(int _V){
  memset(last,-1,sizeof(last));
  V = _V; E = 0;
}

void add_edge(int u, int v, cap_type _cap, cost_type _cost){
  to[E] = v, cap[E] = _cap;
  cost[E] = _cost, next[E] = last[u];
  last[u] = E++;
  to[E] = u, cap[E] = 0;
  cost[E] = -_cost, next[E] = last[v];
  last[v] = E++;
}

// only if there is initial negative cycle
void BellmanFord(int s, int t){
  bool stop = false;
  for(int i = 0;i<V;++i) dist[i] = INF;
  dist[s] = 0;

  for(int i = 1;i<=V && !stop;++i){
    stop = true;

    for(int j = 0;j<E;++j){
      int u = to[j^1], v = to[j];

      if(cap[j]>0 && dist[u]!=INF && dist[u]+cost[j]<dist[v]){
        stop = false;
        dist[v] = dist[u]+cost[j];
      }
    }
  }

  for(int i = 0;i<V;++i) if (dist[i]!=INF) pot[i] = dist[i];
}

void mcmf(int s, int t){
  flowVal = flowCost = 0;
  memset(pot,0,sizeof(pot));

  BellmanFord(s,t);

  while(true){
    memset(prev,-1,sizeof(prev));
    memset(visited,false,sizeof(visited));
    for(int i = 0;i<V;++i) dist[i] = INF;

    priority_queue< pair<cost_type, int> > Q;
    Q.push(make_pair(0,s));
    dist[s] = prev[s] = 0;

    while(!Q.empty()){
      int aux = Q.top().second;
      Q.pop();
```

```
      if(visited[aux]) continue;
      visited[aux] = true;

      for(int e = last[aux];e!=-1;e = next[e]){
        if(cap[e]<=0) continue;
        cost_type new_dist = dist[aux]+cost[e]+pot[aux]-pot[to[
            e]];
        if(new_dist<dist[to[e]]){
          dist[to[e]] = new_dist;
          prev[to[e]] = e;
          Q.push(make_pair(-new_dist,to[e]));
        }
      }
    }

    if (prev[t]==-1) break;

    cap_type f = cap[prev[t]];
    for(int i = t;i!=s;i = to[prev[i]^1]) f = min(f,cap[prev[i
        ]]);
    for(int i = t;i!=s;i = to[prev[i]^1]){
      cap[prev[i]] -= f;
      cap[prev[i]^1] += f;
    }

    flowVal += f;
    flowCost += f*(dist[t]-pot[s]+pot[t]);

    for(int i = 0;i<V;++i) if (prev[i]!=-1) pot[i] += dist[i];
  }
}
```

6.16 Tree Isomorphism

```
// Determina se duas arvores com indices distintos sao iguais
const int MAXV = 100010;
int N;
vector <int> A[MAXV], NA[MAXV], B[MAXV], NB[MAXV];
/* Armazenar em N o numero de vertices de cada arvore,
   salvar em A e em B as duas arvores indexadas em 1
   Antes de ler as arvores, limpar A[], NA[], B[], NB[]
*/
bool treeIsomorphism(){
  rep(i, 1, N+1){
    rep(j, 0, A[i].size()) NA[i].pb( A[A[i][j]].size() );
    sort(NA[i].begin(), NA[i].end());

    rep(j, 0, B[i].size()) NB[i].pb( B[B[i][j]].size() );
    sort(NB[i].begin(), NB[i].end());
  }
  sort(NA + 1, NA + N + 1);
  sort(NB + 1, NB + N + 1);
  rep(i, 1, N+1){
    if(NA[i] != NB[i])
      return false;
  }
  return true;
}
```

6.17 Stoer Wagner - Minimum Cut

```
/*
   Retorna o corte minimo do grafo
   (Conjunto de arestas que se removido, desconecta o grafo)
Input: n = #vertices, g[i][j] = custo da aresta (i->j)
Output: Retorna o corte minimo
Complexidade: O(N^3)
 */
// Maximum number of vertices in the graph
#define NN 101
// Maximum edge weight (MAXW * NN * NN must fit into an int)
#define MAXW 110

// Adjacency matrix and some internal arrays
int g[NN][NN], v[NN], w[NN], na[NN], n;
bool a[NN];
int stoer_wagner() {
  // init the remaining vertex set
  for( int i = 0; i < n; i++ ) v[i] = i;
  // run Stoer-Wagner
  int best = MAXW * n * n;
  while( n > 1 ) {
    // initialize the set A and vertex weights
    a[v[0]] = true;
    for( int i = 1; i < n; i++ ){
      a[v[i]] = false;
      na[i - 1] = i;
      w[i] = g[v[0]][v[i]];
    }
    // add the other vertices
    int prev = v[0];
    for( int i = 1; i < n; i++ ){
      // find the most tightly connected non-A vertex
      int zj = -1;
      for( int j = 1; j < n; j++ )
        if( !a[v[j]] && ( zj < 0 || w[j] > w[zj] ) )
          zj = j;
      // add it to A
      a[v[zj]] = true;
      // last vertex?
      if( i == n - 1 ){
        // remember the cut weight
        best = min(best,w[zj]);

        // merge prev and v[zj]
```

```
        for( int j = 0; j < n; j++ )
          g[v[j]][prev] = g[prev][v[j]] += g[v[zj]][v[j]];
        v[zj] = v[--n];
        break;
      }
      prev = v[zj];
      // update the weights of its neighbours
      for( int j = 1; j < n; j++ ) if( !a[v[j]] )
        w[j] += g[v[zj]][v[j]];
    }
  }
  return best;
}
```

6.18 Erdos Gallai

```
// Determina se existe um grafo tal que b[i] eh o grau do i-
    esimo vertice
// vertices indexado em 1. Apenas armazenar em b[1...N] e
    chamar EGL()
long long b[100005], n;
long long dmax, dmin, dsum, num_degs[100005];

bool basic_graphical_tests(){ //Sort and perform some simple
    tests on the sequence
  int p = n;
  memset(num_degs, 0, (n+1)*sizeof(long long));

  dmax = dsum = n = 0;
  dmin = p;
  for(int d=1; d<=p; d++){
    if(b[d] < 0 || b[d] >= p)
      return false;
    else if(b[d] > 0){
      if(dmax < b[d]) dmax = b[d];
      if(dmin > b[d]) dmin = b[d];
      dsum = dsum+b[d];
      n++;
      num_degs[b[d]]++;
    }
  }
  if(dsum%2 || dsum > n*(n-1)) return false;
  return true;
}
bool EGL() {
  long long k,sum_deg,sum_nj,sum_jnj,run_size;

  if( !basic_graphical_tests()) return false;
  if(n == 0 || 4*dmin*n >= (dmax+dmin+1)*(dmax+dmin+1)) return
      true;

  k = sum_deg = sum_nj = sum_jnj = 0;
  for(int dk = dmax; dk >= dmin; dk--){
    if(dk < k+1) return true;

    if(num_degs[dk] > 0){
      run_size = num_degs[dk];
      if(dk < k+run_size)
        run_size = dk-k;
      sum_deg += run_size * dk;

      for(int v=0; v<run_size; v++){
        sum_nj += num_degs[k+v];
        sum_jnj += (k+v) * num_degs[k+v];
      }
      k += run_size;
      if(sum_deg > k*(n-1) - k*sum_nj + sum_jnj)
        return false;
    }
  }
  return true;
}
```

6.19 Stable Marriage

```
/*Seja um conjunto de m homens e n mulheres, onde cada pessoa
  tem uma preferencia por outra de sexo oposto. O algoritmo
  produz o casamento estavel de cada homem com uma mulher.
Estavel:
- Cada homem se casara com uma mulher diferente (n >= m)
- Dois casais H1M1 e H2M2 nao serao instaveis.
Dois casais H1M1 e H2M2 sao instaveis se:
- H1 prefere M2 ao inves de M1, e
- M1 prefere H2 ao inves de H1.
Entrada
(1) m = #homens, n = #mulheres
(2) R[x][y] = i, i: eh a ordem de preferencia do homem y pela
    mulher x
Obs.: Quanto maior o valor de i menor eh a preferencia do homem
     y pela mulher x

(3) L[x][i] = y : A mulher y eh a i-esima preferencia do homem
    x
Obs.: 0 <= i <= n-1, quanto menor o valor de i maior eh a
    preferencia
do homem x pela mulher y
Saida
L2R[i]: a mulher do homem i (sempre entre 0 e n-1)
R2L[j]: o homem da mulher j (-1 se a mulher for solteira)

Complexidade O(m^2)
 */

#define MAXM 1000
#define MAXW 1000
int L[MAXM][MAXW];
```

```
int R[MAXW][MAXM];
int L2R[MAXM], R2L[MAXW];
int m, n;
int p[MAXM];

void stableMarriage (){
  static int p[MAXM];
  memset(R2L, -1, sizeof(R2L));
  memset(p, 0, sizeof(p));
  for (int i = 0; i < m; ++i){
    int man = i;
    while (man >= 0){
      int wom;
      while (42){
        wom = L[man][p[man]++];
        if (R2L[wom] < 0 || R[wom][man] > R[wom][R2L[wom]])
          break;
      }
      int hubby = R2L[wom];
      R2L[L2R[man] = wom] = man;
      man = hubby;
    }
  }
}
```

6.20 Hungarian Max Bipartite Matching with Cost

```
/*Encontra o casamento bipartido maximo/minimo com peso nas
    arestas
  Como usar:

  Criar o grafo:
  Hungarian G(L, R, ehMaximo)
  L = #vertices a esquerda
  R = #vertices a direita
  ehMaximo = variavel booleana que indica se eh casamento
      maximo ou minimo

  Adicionar arestas:
  G.add_edge(x,y,peso)
  x = vertice da esquerda no intervalo [0,L-1]
  y = vertice da direita no intervalo [0,R-1]
  peso = custo da aresta
obs: tomar cuidado com multiplas arestas.

Resultado:
match_value = soma dos pesos dos casamentos
pairs = quantidade de pares (x-y) casados
xy[x] = vertice y casado com x
yx[y] = vertice x casado com y

Complexidade do algoritmo: O(n^3)
Problemas resolvidos: SCITIES (SPOJ)
*/

struct Hungarian{
  enum{ MAXN = 150, INF = 0x3f3f3f3f };
  int cost[MAXN][MAXN];
  int xy[MAXN], yx[MAXN];
  bool S[MAXN], T[MAXN];
  int lx[MAXN], ly[MAXN], slack[MAXN], slackx[MAXN], prev[MAXN
      ];
  int match_value, pairs;
  bool ehMaximo;
  int n;

  Hungarian(int L, int R, bool _ehMaximo=true){
    n = max(L, R);
    ehMaximo = _ehMaximo;
    if (ehMaximo) memset(cost, 0, sizeof cost);
    else      memset(cost, INF, sizeof cost);

  }

  void add_edge(int x, int y, int peso){
    if (!ehMaximo) peso *= (-1);
    cost[x][y] = peso;
  }

  int solve(){
    match_value = 0;
    pairs = 0;
    memset(xy, -1, sizeof(xy));
    memset(yx, -1, sizeof(yx));
    init_labels();
    augment();
    for (int x = 0; x < n; ++x)
      match_value += cost[x][xy[x]];
    return match_value;
  }

  void init_labels (){
    memset(lx, 0, sizeof(lx));
    memset(ly, 0, sizeof(ly));
    for (int x = 0; x < n; ++x)
      for (int y = 0; y < n; ++y)
        lx[x] = max(lx[x], cost[x][y]);
  }

  void augment (){
    if (pairs == n) return;
    int x, y, root;
    int q[MAXN], wr = 0, rd = 0;
    memset(S, false, sizeof(S));
    memset(T, false, sizeof(T));
    memset(prev, -1, sizeof(prev));
    for (x = 0; x < n; ++x)
```

```
    if (xy[x] == -1){
      q[wr++] = root = x;
      prev[x] = -2;
      S[x] = true;
      break;
    }
  for (y = 0; y < n; ++y){
    slack[y] = lx[root] + ly[y] - cost[root][y];
    slackx[y] = root;
  }
  while (true){
    while (rd < wr){
      x = q[rd++];
      for (y = 0; y < n; ++y)
        if (cost[x][y] == lx[x] + ly[y] && !T[y]){
          if (yx[y] == -1)
            break;
          T[y] = true;
          q[wr++] = yx[y];
          add(yx[y], x);
        }
      if (y < n)
        break;
    }
    if (y < n) break;
    update_labels();
    wr = rd = 0;
    for (y = 0; y < n; ++y)
      if (!T[y] && slack[y] == 0){
        if (yx[y] == -1){
          x = slackx[y];
          break;
        }
        else{
          T[y] = true;
          if (!S[yx[y]]){
            q[wr++] = yx[y];
            add(yx[y], slackx[y]);
          }
        }
      }
    if (y < n) break;
  }
  if (y < n){
    ++pairs;
    for (int cx = x, cy = y, ty; cx != -2; cx = prev[cx], cy
        = ty){
      ty = xy[cx];
      yx[cy] = cx;
      xy[cx] = cy;
    }
    augment();
  }
}
```

```
  void add (int x, int prevx){
    S[x] = true;
    prev[x] = prevx;
    for (int y = 0; y < n; ++y)
      if (lx[x] + ly[y] - cost[x][y] < slack[y]){
        slack[y] = lx[x] + ly[y] - cost[x][y];
        slackx[y] = x;
      }
  }

  void update_labels (){
    int x, y, delta = INF;
    for (y = 0; y < n; ++y)
      if (!T[y])
        delta = min(delta, slack[y]);
    for (x = 0; x < n; ++x)
      if (S[x])
        lx[x] -= delta;
    for (y = 0; y < n; ++y)
      if (T[y])
        ly[y] += delta;
    for (y = 0; y < n; ++y)
      if (!T[y])
        slack[y] -= delta;
  }

  int casouComX(int x){
    return xy[x];
  }

  int casouComY(int y){
    return yx[y];
  }
};

// O codigo abaixo resolve o problema scities (Spoj)
int main(){
  int casos; cin >> casos;
  while (casos--){
    int L, R;
    cin >> L >> R;
    Hungarian G(L, R, true);

    int x, y, w, aux[L][R];
    memset(aux, 0, sizeof aux);
    while (scanf("%d %d %d", &x, &y, &w) != EOF){
      if (x==0 && y==0 && w==0) break;
      aux[x-1][y-1] += w;
    }
    for (int x = 0; x < L; x++){
      for (int y = 0; y < R; y++){
        if (aux[x][y] != 0){
          G.add_edge(x,y, aux[x][y]);
```

```
        }
      }
    }
    printf("%d\n", G.solve());
  }
  return 0;
}
```

7 Estruturas de Dados

7.1 BIT

```
ll bit[MAXN]; int N;
void update(int i, ll v){
  for (; i <= N; i += i&-i)
    bit[i] += v;
}
ll query(int i){
  ll sum=0;
  for(; i; i -= i&-i)
    sum += BIT[i];
  return sum;
}
```

7.2 BIT 2D

```
ll bit[MAXL][MAXC]; int L, C;
void update(int x, int y, int v){
  for (; x <= L; x += x&-x)
    for (int yy = y; yy <= C; yy += yy&-yy)
      bit[x][yy] += v;
}
ll query(int x, int y){
  ll sum = 0;
  for (; x; x -= x&-x)
    for (int yy = y; yy; yy -= yy&-yy)
      sum += bit[x][yy];
  return sum;
}
```

7.3 RMQ

```
#define MAXN 100000
int A[MAXN], T[4*MAXN];
int N; // #number of elements in A[]

void build(int no, int a, int b){
  if (a == b){
      T[no] = a;
    }else{
      int m = (a + b)/2;
      build(2*no, a, m);
      build(2*no+1, m+1, b);
      int p1 = T[2*no], p2 = T[2*no+1];
      T[no] = (A[p1] <= A[p2]) ? p1 : p2;
    }
}
// Faz A[i] = v
void update(int no, int a, int b, int i, int v){
  if (a > i || b < i) return;
    if (a == i && b == i){
    A[i] = v;
    T[no] = i;
    return;
  }
  int m = (a + b)/2;
  update(2*no, a, m, i, v);
  update(2*no+1, m+1, b, i, v);
  int p1 = T[2*no], p2 = T[2*no+1];
  T[no] = (A[p1] <= A[p2]) ? p1 : p2;
}
// Retorna o idx do menor valor do intervalo [i,j]
int query(int no, int a, int b, int i, int j){
  if (a > j || b < i) return -1;
    if (a >= i && b <= j) return T[no];
    int m = (a + b)/2;
  int p1 = query(2*no, a, m, i, j);
  int p2 = query(2*no+1, m+1, b, i, j);
    if (p1 == -1) return p2;
    if (p2 == -1) return p1;
    return (A[p1] <= A[p2]) ? p1 : p2;
}
```

7.4 RMQ-Aplicações

```
//1. Determinar a maior area dado pelo tamanho do segmento [i,j
    ] e o menor valor contido neste segmento
long long greater_area(int i, int j){
  if(i > j) return -1;
  if(i == j) return A[i];
  int idx = rmq(i, j);
  ll total = (long long)A[idx]*(j - i + 1);
  ll left = greater_area(i, idx-1), right = greater_area(idx+1,
       j);
  return max(total, max(left, right));
}

//2. Determinar os dois menores valores de um intervalo
pair<int, int> min_par(int i, int j){
  int idx = rmq(i, j);
  int left = rmq(i, idx-1), right = rmq(idx+1, j);
  return make_pair(A[idx], min(A[left], A[right]));
}
```

```
//3. Determinar qual o no mais a esquerda na arvore tem valor
    maior que K, e adicionar K a ele.
void update(int no, int b, int e, int K){
  if (b == e){ A[b] += K; T[no] = b; return; }
  int m = (b+e)/2;
  if (A[T[no*2]] >= K) update(no*2, b, m, K);
  else if(A[T[no*2 + 1] >= K) update(no*2+1, m+1, e, K);
  if (A[T[2*no]] >= A[T[2*no + 1]]) T[no] = T[2*no];
  else T[no] = T[2*no + 1];
}
```

7.5 Seg Tree com Lazy

```
ll v[MAXN], tree[4*MAXN], lazy[4*MAXN];

void build_tree(int no, int a, int b){
  if(a == b){
    tree[no] = v[a];
    return;
  }
  int m = (a + b)/2;
  build_tree(2*no, a, m);
  build_tree(2*no+1, m+1,b);
  tree[no] = tree[no*2] + tree[no*2+1];
}
void propagate(int no, int a, int b){
  if(lazy[no]){
    tree[no] += (b - a + 1) * lazy[no];
    if(a != b){
      lazy[2*no] += lazy[no];
      lazy[2*no+1] += lazy[no];
    }
    lazy[no] = 0;
  }
}
void update(int no, int a, int b, int i, int j, ll value){
  propagate(no, a, b);
  if(a > j || b < i)return;
  if(a >= i && b <= j){
    lazy[no] += value;
    propagate(no, a, b);
    return;
  }
  int m = (a + b)/2;
  update( 2*no ,  a,  m, i, j, value);
  update(2*no+1, m+1, b, i, j, value);
  tree[no] = tree[no*2] + tree[no*2+1];
}

ll query(int no, int a, int b, int i, int j){
  propagate(no, a, b);
  if(a > j || b < i) return 0;
  if(a >= i && b <= j) return tree[no];
  int m = (a + b)/2;
  ll q1 = query(2*no, a, m, i, j);
  ll q2 = query(2*no+1, m+1, b, i, j);
  return q1+q2;
}
```

7.6 Union-Find

```
#define MAXV 100000
int V, pai[MAXV], rank[MAXV], size[MAXV];
void init(){
  rep(i,0,V)
    pai[i] = i, rank[i] = 0, size[i] = 1;
}
void find(int v){
  if(v != pai[v]) pai[v] = find(pai[v]);
  return pai[v];
}
void join(int u, int v){
  u = find(u); v = find(v);
  if(u == v) return;

  if(rank[u] < rank[v]) swap(u, v);
  pai[v] = u; // add v na arvore de u
  size[u] += size[v];
  if (rank[u] == rank[v]) rank[u]++;
}
bool same_set(int u, int v){
  return find(u) == find(v);
}
```

7.7 Treap

```
#include <bits/stdc++.h>
using namespace std;

#define rep(i, a, b) for(int i=a; i<b; i++)

typedef struct node{
  int prior,size;
  int val;//value stored in the array
  int sum;//whatever info you want to maintain in segtree for
      each node
  int lazy;//whatever lazy update you want to do
  int rev;
  struct node *l,*r;
}node;
typedef node* pnode;
int sz(pnode t){
  return t?t->size:0;
}
```

```
void upd_sz(pnode t){
  if(t)t->size=sz(t->l)+1+sz(t->r);
}
void lazy(pnode t){
  if(!t || t->lazy==-1)return;
  t->val=t->lazy;//operation of lazy
  t->sum=t->lazy*sz(t);
  if(t->l)t->l->lazy=t->lazy;//propagate lazy
  if(t->r)t->r->lazy=t->lazy;
  t->lazy=-1;
}
void reset(pnode t){
  if(t)t->sum = t->val;//no need to reset lazy coz when we call
       this lazy would itself be propagated
}
void combine(pnode& t,pnode l,pnode r){//combining two ranges
    of segtree
  if(!l || !r)return void(t = l?l:r);
  t->sum = l->sum + r->sum;
}
void operation(pnode t){//operation of segtree
  if(!t)return;
  reset(t);//reset the value of current node assuming it now
      represents a single element of the array
  lazy(t->l);lazy(t->r);//imp:propagate lazy before combining t
      ->l,t->r;
  combine(t,t->l,t);
  combine(t,t,t->r);
}
void push(pnode t){
  if(!t || !t->rev) return;
  t->rev=false;
  swap(t->l, t->r);
  if(t->l)  t->l->rev ^= true;
  if(t->r)  t->r->rev ^= true;
}
void split(pnode t,pnode &l,pnode &r,int pos,int add=0){
  if(!t)return void(l=r=NULL);
  push(t); lazy(t);
  int curr_pos = add + sz(t->l);
  if(curr_pos<=pos)//element at pos goes to left subtree(l)
    split(t->r,t->r,r,pos,curr_pos+1),l=t;
  else
    split(t->l,l,t->l,pos,add),r=t;
  upd_sz(t);
  operation(t);
}
void merge(pnode &t,pnode l,pnode r){ //l->leftarray,r->
    rightarray,t->resulting array
  push(l);push(r);lazy(l);lazy(r);
  if(!l || !r) t = l?l:r;
  else if(l->prior>r->prior)merge(l->r,l->r,r),t=l;
  else    merge(r->l,l,r->l),t=r;
  upd_sz(t);
  operation(t);
}
pnode init(int val){
  pnode ret = new node;
  ret->prior=rand();ret->size=1;
  ret->val=val;
  ret->sum=val;ret->lazy=-1; ret->rev=0;
  ret->l=NULL, ret->r=NULL;
  return ret;
}
int range_query(pnode t,int l,int r){//[l,r]
  pnode L,mid,R;
  split(t,L,mid,l-1);
  split(mid,t,R,r-l);//note: r-l!!
  int ans = t->sum;
  merge(mid,L,t);
  merge(t,mid,R);
  return ans;
}
void range_update(pnode t,int l,int r,int val){//[l,r]
  pnode L,mid,R;
  split(t,L,mid,l-1);
  split(mid,t,R,r-l);//note: r-l!!
  t->lazy=val; //lazy_update
  merge(mid,L,t);
  merge(t,mid,R);
}
void reverse(pnode t, int l, int r){
  pnode L, mid, R;
  split(t, L, mid, l-1);
  split(mid, mid, R, r-l);
  mid->rev^=true;
  merge(t, L, mid);
  merge(t, t, R);
}
void output(pnode t) {
  if (!t)  return;
  push (t);
  lazy(t);
  output (t->l);
  printf ("%d ", t->val);
  output (t->r);
}

int valor(int val){
  return val&1?0:1;
}

int main(){
  int P, Q;
  while(scanf("%d %d", &P, &Q) !=EOF){
    pnode tree=NULL, T1=NULL, T2=NULL, T3=NULL; int val;
    rep(i, 0, P){
      scanf("%d", &val);
```

```
      split(tree, T1, T2, i);
      merge(T1, T1, init(valor(val)));
      merge(tree, T1, T2);
    }
    while(Q--){ }
  }
}
```

7.8 Seg Tree 2D

```
struct node{
  int qt;
  int f1, f2, f3, f4;
};

node new_node(){
  node ret;
  ret.qt=ret.f1=ret.f2=ret.f3=ret.f4=0;
  return ret;
}

vector <node> tree;
int cnt = 0;

bool inRange(int x1, int x2, int y1, int y2, int a1, int a2,
    int b1, int b2){
  if(x2 < x1 || y2 < y1) return false;
  if(x2 < a1 || x1 > a2) return false;
  if(y2 < b1 || y1 > b2) return false;
  return true;
}

void update(int no, int x1, int x2, int y1, int y2, int a1, int
     a2, int b1, int b2, int val){
  if(no == cnt) tree[cnt++] = new_node();

  if(x1>=a1 && x2<=a2 && y1>=b1 && y2<=b2){
    tree[no].qt = val;
    return;
  }

  int f1=0, f2=0, f3=0, f4=0;
  if(inRange(x1, (x1+x2)/2, y1, (y1+y2)/2, a1, a2, b1, b2)){
    if(!tree[no].f1) tree[no].f1 = cnt;
    update(tree[no].f1, x1, (x1+x2)/2, y1, (y1+y2)/2, a1, a2, b
        1, b2, val);
  }
  if(inRange(x1, (x1+x2)/2, (y1+y2)/2+1, y2, a1, a2, b1, b2)){
    if(!tree[no].f2) tree[no].f2 = cnt;
    update(tree[no].f2, x1, (x1+x2)/2, (y1+y2)/2+1, y2, a1, a2,
         b1, b2, val);
  }
  if(inRange((x1+x2)/2+1, x2, y1, (y1+y2)/2, a1, a2, b1, b2)){
    if(!tree[no].f3) tree[no].f3 = cnt;
    update(tree[no].f3, (x1+x2)/2+1, x2, y1, (y1+y2)/2, a1, a2,
         b1, b2, val);
  }
  if(inRange((x1+x2)/2+1, x2, (y1+y2)/2+1, y2, a1, a2, b1, b2))
      {
    if(!tree[no].f4) tree[no].f4 = cnt;
    update(tree[no].f4, (x1+x2)/2+1, x2, (y1+y2)/2+1, y2, a1, a
        2, b1, b2, val);
  }

  if(tree[no].f1) f1=tree[tree[no].f1].qt;
  if(tree[no].f2) f2=tree[tree[no].f2].qt;
  if(tree[no].f3) f3=tree[tree[no].f3].qt;
  if(tree[no].f4) f4=tree[tree[no].f4].qt;

  tree[no].qt=f1+f2+f3+f4;
}

int query(int no, int x1, int x2, int y1, int y2, int a1, int a
    2, int b1, int b2){
  if(!inRange(x1, x2, y1, y2, a1, a2, b1, b2) || no>=cnt ||
      tree[no].qt==0) return 0;

  if(x1>=a1 && x2<=a2 && y1>=b1 && y2<=b2) return tree[no].qt;

  int f1=0, f2=0, f3=0, f4=0;
  if(tree[no].f1) f1 = query(tree[no].f1, x1, (x1+x2)/2, y1, (y
      1+y2)/2, a1, a2, b1, b2);
  if(tree[no].f2) f2 = query(tree[no].f2, x1, (x1+x2)/2, (y1+y2
      )/2+1, y2, a1, a2, b1, b2);
  if(tree[no].f3) f3 = query(tree[no].f3, (x1+x2)/2+1, x2, y1,
      (y1+y2)/2, a1, a2, b1, b2);
  if(tree[no].f4) f4 = query(tree[no].f4, (x1+x2)/2+1, x2, (y1+
      y2)/2+1, y2, a1, a2, b1, b2);

  return f1+f2+f3+f4;

}

void erase(){
  tree.clear();
  vector <node> xua;
  swap(tree, xua);
  tree.resize(1000010);
  cnt = 0;
}
```

7.9 Polyce

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
```

```cpp
#include <ext/rope>

using namespace std;
using namespace __gnu_cxx;
using namespace __gnu_pbds;

typedef tree <int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> ordered_set;

int main(){

  ordered_set T; ordered_set :: iterator it;
  T.insert(); T.erase(); int k = T.find_by_order(); int kk = T.
      order_of_key();

  rope <int> v; rope <int> :: iterator it;
  v.push_back();
  rope <int> cur = v.substr(l, r - l + 1);
  for( it = cur.mutable_begin(); it != cur.mutable_end(); it++
      ) cout << *it << " ";
  v.erase(l, r - l + 1);
  v.insert(v.mutable_begin(), cur);

}
```

7.10 KD2

```cpp
struct point{
  int x, y, z;
  point(int x=0, int y=0, int z=0): x(x), y(y), z(z) {}
  point operator-(point q){
    return point(x-q.x, y-q.y, z-q.z);
  }
  int operator*(point q){
    return x*q.x + y*q.y + z*q.z;
  }
};

typedef vector<point> polygon;
priority_queue <double> vans;
int NN, CC, KK, DD;

struct KDTreeNode{
  point p;
  int level;
  KDTreeNode *below, *above;
  KDTreeNode (const point& q, int levl){
    p = q;
    level = levl;
    below = above = 0;
  }
  ~KDTreeNode(){
    delete below, above;
  }
  int diff (const point& pt){
    switch (level)
    {
      case 0:
        return pt.x - p.x;
      case 1:
        return pt.y - p.y;
      case 2:
        return pt.z - p.z;
    }
    return 0;
  }
  ll distSq (point& q){
    return (p-q)*(p-q);
  }
  int rangeCount (point& pt, ll K){
    int count = (distSq(pt) <= K*K) ? 1 : 0;
    if(count) vans.push(-sqrt(distSq(pt)));

    int d = diff(pt);
    if (~d <= K && above != 0)
      count += above->rangeCount(pt, K);
    if (d <= K && below != 0)
      count += below->rangeCount(pt, K);
    return count;
  }
};

class KDTree {
  public:
    polygon P;
    KDTreeNode *root;
    int dimention;
    KDTree() {}
    KDTree(polygon &poly, int D){
      P = poly;
      dimention = D;
      root = 0;
      build();
    }
    ~KDTree(){
      delete root;
    }
    //count the number of pairs that has a distance less than K
    ll countPairs(ll K){
      ll count = 0;
      rep(i, 0, P.size())
        count += root->rangeCount(P[i], K) - 1;
      return count;
    }
  protected:
    void build(){
      //random_shuffle(all(P));
```

```
      rep(i, 0, P.size()){
        root = insert(root, P[i], -1);
      }
    }
    KDTreeNode *insert(KDTreeNode* t, const point& pt, int
        parentLevel){
      if (t == 0){
        t = new KDTreeNode (pt, (parentLevel+1) % dimention);
        return t;
      }
      else{
        int d = t->diff(pt);
        if (d <= 0) t->below = insert (t->below, pt, t->level);
        else t->above = insert (t->above, pt, t->level);
      }
      return t;
    }
};

int main(){
  point e; e.z=0;
  polygon p;
  set <ii> st;

  while(scanf("%d %d %d %d", &NN, &CC, &KK, &DD)!=EOF){
    p.clear(); KK=min(NN, KK); st.clear();

    rep(i, 0, NN){
      scanf("%d %d", &e.x, &e.y);
      st.insert(mp(e.x, e.y));
      p.pb(e);
    }

    KDTree tree(p, 2);
    int ans=0;
    rep(i, 0, CC){
      scanf("%d %d", &e.x, &e.y);
      if(st.count(mp(e.x, e.y))) continue;

      ll at=0;
      rep(i, 0, 30){
        at=ll(1)<<i; while(!vans.empty()) vans.pop();
        int aux=tree.root->rangeCount(e, at);
        if(aux>=KK) break;
      }
      double sum=0.0;
      rep(i, 0, KK){
        sum+=-vans.top();
        vans.pop();
      }
      if(sum>=DD) ans++;
    }
    printf("%d\n", ans);
  }
  return 0;
}
```

8 Strings

8.1 KMP

```
/*obs: A funcao strstr (char* text, char* pattern) da
    biblioteca <cstring> implementa KMP (C-ANSI). A funcao
    retorna a primeira ocorrencia do padrao no texto, KMP
    retorna todas.
nres -> O numero de ocorrencias do padrao no texto
res[] -> posicoes das nres ocorrencias do padrao no texto
Complexidade do algoritmo: O(n+m)*/
#define MAXN 100001
int pi[MAXN], res[MAXN], nres;
void kmp(string text, string pattern){
  nres = 0;
  pi[0] = -1;
  rep(i, 1, pattern.size()){
        pi[i] = pi[i-1];
        while (pi[i] >= 0 && pattern[pi[i]+1] != pattern[i])
            pi[i] = pi[pi[i]];
        if (pattern[pi[i]+1] == pattern[i])
            ++pi[i];
    }
  int k = -1;  //k+1 eh o tamanho do match atual
    rep(i, 0, text.size()){
        while (k >= 0 && pattern[k+1] != text[i])
            k = pi[k];
        if (pattern[k+1] == text[i])
            ++k;
        if (k+1 == pattern.size()){
            res[nres++] = i-k;
            k = pi[k];
        }
    }
}
```

8.2 Aho Corasick

```
const int cc = 26;
const int MAX = 100;

int cnt;
int sig[MAX][cc];
int term[MAX];
int T[MAX];
int v[MAX];

inline int C( char c ){
```

```
  return c - '0';
}

void add( string s, int id ){
  int x = 0;
  rep( i, 0, s.size() ){
    int c = C( s[i] );
    if( !sig[x][c] ){
      term[cnt] = 0;
      sig[x][c] = cnt++;
    }
    x = sig[x][c];
  }
  term[x] = 1;
  v[id] = x;
}

void aho(){
  queue < int > q;
  rep( i, 0, cc ){
    int x = sig[0][i];
    if( !x ) continue;
    q.push( x );
    T[x] = 0;
  }
  while( !q.empty() ){
    int u = q.front();
    q.pop();
    rep( i, 0, cc ){
      int x = sig[u][i];
      if( !x ) continue;
      int v = T[u];
      while( v && !sig[v][i] ) v = T[v];
      v = sig[v][i];
      T[x] = v;
      term[x] += term[v];
      q.push( x );
    }
  }
}

// Conta a quantidade de palavras de exatamente l caracteres
    que se pode formar com um determinado alfabeto, dado que
    algumas palavras s o "proibidas"

int mod = 1e9+7;
ll pd[100][MAX];

ll solve( int pos, int no ){
  if( pos == 0 ) return 1;
  if( pd[pos][no] != -1 ) return pd[pos][no];
  ll ans = 0;
  rep( i, 0, cc ){
    int v = no;
    while( v && !sig[v][i] ) v = T[v];
    v = sig[v][i];
    if( term[v] ) continue;
    ans = ( ans + solve( pos-1, v ) )%mod;
  }
  return pd[pos][no] = ans;
}

void Qttd_de_Palavras(){

  while( 1 ){
    memset( sig, 0, sizeof sig );
    memset( pd, -1, sizeof pd );
    cnt = 1;
    int l = readInt();
    if( !l ) break;
    int n = readInt();
    string pattern;
    rep( i, 0, n ){
      cin >> pattern;
      add( pattern, i );
    }
    aho();
    ll ans = 0;
    rep( i, 1, l+1 )
      ans = ( ans + solve( i, 0 ) )%mod;
    printf("%d\n", ans );
  }
}

// Verifica quais padres ocorreram em um texto

int alc[MAX];

void busca( string s ){
  int x = 0;
  rep( i, 0, s.size() ){
    int c = C( s[i] );
    while( x && !sig[x][c] ) x = T[x];
    x = sig[x][c];
    alc[x] = 1;
  }
}

void Ql_Ocorreu(){
  string pattern, text;
  while( getline(cin, text )){
    if( text == "*" ) break;
    memset( sig, 0, sizeof sig );
    memset( alc, 0, sizeof alc );
    cnt = 1;
    int n;
    cin >> n;
    rep( i, 0, n ){
```

```
      cin >> pattern;
      add( pattern, i );
    }
    aho();
    busca( text );
    for( int i = cnt-1; i >= 0; i-- ){
      if( alc[i] ) alc[ T[i] ] = 1;
    }
    rep( i, 0, n ){
      int u = v[i];
      if( alc[u] ) printf("Ocorreu\n");
      else printf("Nao ocorreu\n");
    }
  }
}

// Total de ocorrencias de cada padro em uma string, mesmo com
    sufixos iguais
ll busca2( string s ){
  ll x = 0, cont = 0;
  rep( i, 0, s.size() ){
    int c = C( s[i] );
    while( x && !sig[x][c] ) x = T[x];
    x = sig[x][c];
    cont += term[x];
  }
  return cont;
}

void Qnts_vezes_Ocorreu(){
  string text, pattern;
  while( cin >> text ){
    if( text == "*" ) break;
    memset( sig, 0, sizeof sig );
    cnt = 1;
    int n = readInt();
    rep( i, 0, n ){
      cin >> pattern;
      add( pattern, i );
    }
    aho();
    rep( i, 1, 10 ) debug( T[i] )
    cout << busca2( text ) << endl;
  }
}

//Encontra a primeira ocorrencia de cada padro em uma string
void busca3( string s ){
  int x = 0;
  rep( i, 0, s.size() ){
    int c = C( s[i] );
    while( x && !sig[x][c] ) x = T[x];
    x = sig[x][c];
    if( !alc[x] ) alc[x] = i+1;
  }
}

void Onde_Ocorreu(){
  string pattern, text;
  int tam[1000];
  while( cin >> text ){
    if( text == "*" ) break;
    memset( sig, 0, sizeof sig );
    memset( alc, 0, sizeof alc );
    cnt = 1;
    int n;cin >> n;
    rep( i, 0, n ){
      cin >> pattern;
      tam[i] = pattern.size();
      add( pattern, i );
    }

    aho();
    busca3( text );
    for( int i = cnt-1; i >= 0; i-- ){
      alc[ T[i] ] = min( alc[i], alc[ T[i] ] );
    }
    rep( i, 0, n ){
      int u = v[i];
      if( alc[u] != INF ){
        int k = alc[u] - tam[i] + 1;
        printf("De %d a %d\n", k, alc[u] );
      }
      else
        printf("Nao ocorreu\n");
    }
  }
}
```

8.3 Suffix Array

```
#define MAX 100010
#define MAX_N 100010
char T[MAX_N];
ll n;
int RA[MAX_N], tempRA[MAX_N];
int SA[MAX_N], tempSA[MAX_N];
int c[MAX_N];
int Phi[MAX_N], PLCP[MAX_N], LCP[MAX_N];

void countingSort(int k){
  int i, sum, maxi = max((ll)300, n);
  memset(c, 0, sizeof c);
  for (i = 0; i < n; i++)
    c[i + k < n ? RA[i + k] : 0]++;
  for (i = sum = 0; i < maxi; i++) {
    int t = c[i]; c[i] = sum; sum += t;
```

```
  }
  for (i = 0; i < n; i++)
    tempSA[c[SA[i]+k < n ? RA[SA[i]+k] : 0]++] = SA[i];
  for (i = 0; i < n; i++)
    SA[i] = tempSA[i];
}

void constructSA() {

  int i, k, r;
  for (i = 0; i < n; i++) RA[i] = T[i];
  for (i = 0; i < n; i++) SA[i] = i;

  for (k = 1; k < n; k <<= 1) {
    countingSort(k);
    countingSort(0);
    tempRA[SA[0]] = r = 0;
    for (i = 1; i < n; i++)
      tempRA[SA[i]] = (RA[SA[i]] == RA[SA[i-1]] && RA[SA[i]+k]
          == RA[SA[i-1]+k]) ? r : ++r;
    for (i = 0; i < n; i++)
      RA[i] = tempRA[i];
    if (RA[SA[n-1]] == n-1) break;
  }
}

void computeLCP() {

  int i, L;
  Phi[SA[0]] = -1;
  for (i = 1; i < n; i++)
    Phi[SA[i]] = SA[i-1];
  for (i = L = 0; i < n; i++) {
    if (Phi[i] == -1) { PLCP[i] = 0; continue; }
    while (T[i + L] == T[Phi[i] + L]) L++;
    PLCP[i] = L;
    L = max(L-1, 0);
  }
  for (i = 0; i < n; i++){
    LCP[i] = PLCP[SA[i]];
  }
}
int main(){
  // concatenar $ no final
}
```

8.4 Suffix Array (Gugu)

```
#include <bits/stdc++.h>
//LIFE IS NOT A PROBLEM TO BE SOLVED
using namespace std;
#define rep(i,a,b) for(int i = int(a); i < int(b) ; i++)
typedef long long int ll;

const int MAX = 100010;
int gap, tam, sa[MAX], pos[MAX], lcp[MAX], tmp[MAX];

bool sufixCmp(int i, int j){
  if(pos[i]!=pos[j]) return pos[i]<pos[j];
  i+=gap, j+=gap;
  return (i<tam && j<tam) ? pos[i]<pos[j] : i>j;
}
void buildSA(char s[]){
  tam=strlen(s);
  for(int i=0; i<tam; i++)
    sa[i]=i, pos[i]=s[i], tmp[i]=0;

  for(gap=1;;gap*=2){
    sort(sa, sa+tam, sufixCmp);
    tmp[0]=0;
    for(int i=0; i<tam-1; i++)
      tmp[i+1] = tmp[i] + sufixCmp(sa[i], sa[i+1]);
    for(int i=0; i<tam; i++)
      pos[sa[i]]=tmp[i];
    if(tmp[tam-1] == tam-1) break;
  }
}
ll buildLCP(char s[]){
  ll sum=0;
  for(int i=0, k=0; i<tam; i++){
    if(pos[i]==tam-1) continue;
    for(int j=sa[pos[i]+1]; s[i+k]==s[j+k];) k++;
    lcp[pos[i]+1] = k;
    sum+=k;
    if(k>0) k--;
  }
  return sum;
}
void PrintAll(char s[]){
  printf("SA\ttam\tLCP\tSuffix\n");
  rep(i, 0, tam)
    printf("%2d\t%2d\t%2d\t%s\n", sa[i], tam-sa[i], lcp[i], s +
        sa[i]);
}
ll num_subs(ll m){
  return (ll)tam*(tam+1)/2 - m;
}
ll num_subsrn(){
  ll ret=0;
  rep(i, 1, tam)
    if(lcp[i]>lcp[i-1])
      ret+=lcp[i]-lcp[i-1];
  return ret;
}
void printans(char s[], int n){
  int maior=0, id=-1;
  rep(i, 0, tam)
```

```
    if(lcp[i]>n  && lcp[i]>maior)
      maior=lcp[i], id=i;
  if(id==-1) printf("*");
  else
    rep(i, sa[id], sa[id]+maior)
      printf("%c", s[i]);
  printf("\n");
}

char s[MAX];
int main(){
  while(1){
    scanf("%s", s);
    if(s[0]=='*') break;

    buildSA(s);
    ll m=buildLCP(s);

    PrintAll(s); //printa sa, lcp, suffixs
    //printf("%lld\n", num_subs(m)); //numero de substrings nao
        repetidas
    //printf("%lld\n", num_subsrn()); //numero de substrings
        que se repete
    //printans(s, 2); //maior substring de tamanho maior ou
        igual a n que se repete
  }
}
```

8.5 Rolling Hash

```
//Permite encontrar um hash de uma substring de S. precompute O
    (n), my_hash O(1)
#define NN 1000006
const ll mod = 1e9+7; //modulo do hash
const ll x = 33; //num. primo > que o maior caracter de S.
ll H[NN], X[NN];
ll V(char c){ return c-'A'; }
ll my_hash(int i, int j){
  ll ret = H[j];
  if(!i) return ret;
  return ((ret-(H[i-1]*X[j-i+1])%mod)+mod)%mod;
}
void precompute(string s){
  X[0] = 1;
  rep(i, 1, NN)
    X[i] = (X[i-1]*x)%mod;
  H[0] = V(s[0]);
  rep(i, 1, s.size())
    H[i] = ((H[i-1]*x)%mod + V(s[i]))%mod;
}
```

8.6 Longest Commom Prefix with Hash

```
// Longest Commom Prefix between S[i..] and S[j..]
int lcp(int i, int j, int tam){
  int lo = 0, hi = tam, ans;
  while(lo <= hi){
    int mid = (lo+hi)/2;
    if(my_hash(i, i+mid-1) == my_hash(j, j+mid-1)){
      ans = mid;
      lo = mid+1;
    }
    else hi = mid-1;
  }
  return ans;
}
```

8.7 Minimum Lexicographic Rotation

```
/*Retorna a menor string lexicografica de s
Necessario my_hash() e lcp()*/
string min_lex_rot(string s){
  int t = s.size();
  precompute(s); // hashing
  s += s;
  int idx = 0;
  for(int i = 1; i < t; i++){
    // tam do prefix comum
    int len = lcp(i, idx, t);
    if (s[i+len] < s[idx+len])
      idx = i;
  }
  return s.substr(idx, t);
}
```

8.8 Longest Palindrome (Manacher algorithm)

```
string preProcess(string s) {
  int n = s.length();
  if (n == 0) return "^$";

  string ret = "^";
  for (int i = 0; i < n; i++)
    ret += "#" + s.substr(i, 1);
  ret += "#$";
  return ret;
}

string longestPalindrome(string s){
  L=C=s.size();
  string T = preProcess(s);
  int n = T.length();
  int *P = new int[n];
  int C = 0, R = 0;
```

```
  for (int i = 1; i < n-1; i++) {
    int i_mirror = 2*C-i;
    P[i] = (R > i) ? min(R-i, P[i_mirror]) : 0;
    while (T[i + 1 + P[i]] == T[i - 1 - P[i]])
      P[i]++;
    if (i + P[i] > R) {
      C = i;
      R = i + P[i];
    }
  }

  int maxLen = 0;
  int centerIndex = 0;

  for (int i = 1; i < n-1; i++) {
    if(!P[i]) continue;
    if (P[i] > maxLen) {
      maxLen = P[i];
      centerIndex = i;
    }
  }
  delete[] P;
  return s.substr((centerIndex - 1 - maxLen)/2, maxLen);
}
```

8.9 Autômato de Sufixos

```
struct state{
  int len, link;
  int next[26];
};

const int MAXN = 200020;
state st[2*MAXN]; //vetor que armazena os estados
int sz; //contador do numero de estados
int last; //numero do estado que corresponde ao texto todo

void sa_init(){
  sz = 1;
  last = 0;
  st[0].len = 0;
  st[0].link = -1;
  rep(i, 0, 26) st[0].next[i]=0;
  //limpa o mapeamento de transicoes
}

void sa_extend (int c, ll &ans){
  int cur = sz++; // novo estado a ser criado
  st[cur].len = st[last].len + 1;
  rep(i, 0, 26) st[cur].next[i]=0;
  int p; // variavel que itera sobre os estados terminais
  for(p=last; p!=-1 && !st[p].next[c]; p=st[p].link){
    st[p].next[c] = cur;
  }
  if(p == -1){// nao ocorreu transicao c nos estados terminais
    st[cur].link = 0;
    ans+=st[cur].len;
  }else{ // ocorreu transicao c no estado p
    int q = st[p].next[c];
    if(st[p].len + 1 == st[q].len){
      st[cur].link = q;
    }else{
      int clone = sz++; // criacao do vertice clone de q
      st[clone].len = st[p].len + 1;
      rep(i, 0, 26) st[clone].next[i]=st[q].next[i];
      st[clone].link = st[q].link;
      for(; p!=-1 && st[p].next[c]==q; p=st[p].link){//
          atualizacao das transicoes c
        st[p].next[c] = clone;
      }
      st[q].link = st[cur].link = clone;
    }
    ans+=st[cur].len-st[st[cur].link].len;
  }
  // atualizacao do estado que corresponde ao texto
  last = cur;
}

bool busca_automato(int m, string p){
  int i, pos=0;
  for( i=0 ; i<m ; i++ ){
    if( st[pos].next[p[i]]==0 ){
      return false;
    }else{
      pos=st[pos].next[p[i]];
    }
  }
  return true;
}


int maior_tamanho_em_comum( string s, string t ){
  ll nothing=0;
  // Constroi o automato com o primeiro texto
  sa_init();
  for(int i=0; i<(int)s.size(); i++) sa_extend (s[i]-'a',
      nothing);
  int estado = 0, tamanho = 0, maior = 0;
  // Passando pelos caracteres do segundo texto
  for(int i=0; i<(int)t.size(); ++i){
    while( estado && ! st[estado].next[t[i]-'a'] ){
      estado = st[estado].link;
      tamanho = st[estado].len;
    }
    if(st[estado].next[t[i]-'a']){
      estado = st[estado].next[t[i]-'a'];
      tamanho++;
```

```
    }
    if(tamanho > maior){
      maior = tamanho;
    }
  }
  return maior;
}

int main(){
  char s[MAXN];
  char p[MAXN];
  while(gets(s)){
    sa_init();
    int tam=strlen(s); ll ans=0;
    rep(i, 0, tam){
      sa_extend(s[i]-'a', ans);
    }
    gets(p);
    printf("%d\n", maior_tamanho_em_comum(s, p));
  }
  return 0;
}
```

9 PD

9.1 Soma acumulada 2D

```
/*Retorna o somatorio dos elementos de uma submatriz em O(1)
Submatriz definida por canto superior esquerdo [a,b] e  canto
    inferior
direito (x,y) .: a <= x && b <= y
*/

#define MAXN 1000
int N, M; // linhas colunas
long long V[MAXN+2][MAXN+2]; // matriz da entrada
long long S[MAXN+2][MAXN+2]; // matriz com as somas acumuladas

// precomputa as somas em O(N*M)
void precal(){
  rep(x, 0, N)
    rep(y, 0, M){
      S[x][y] = V[x][y];
      if (x>0) S[x][y] += S[x-1][y];
      if (y>0) S[x][y] += S[x][y-1];
      if (x>0 && y>0) S[x][y] -= S[x-1][y-1];
    }
}
// retorna a soma da submatriz
long long sum(int a, int b, int x, int y){
  long long soma = S[x][y];
  if (a>0) soma -= S[a-1][y];
  if (b>0) soma -= S[x][b-1];
  if (a>0 && b>0) soma += S[a-1][b-1];
  return soma;
}
```

9.2 Knuth Optimization

```
int N, B, C, yep, save[MAXN][MAXN], sav[MAXN];
ll n[MAXN], mc[MAXN][MAXN], se[MAXN], sd[MAXN], pd[MAXN][MAXN];

ll solve(int i, int k){
  if(i==N) return 0;
  if(k==1) return pd[i][k] = mc[i][N-1];
  if(pd[i][k]!=-1) return pd[i][k];

  ll ret=LINF; int ini=i, fim=N-k+1, best = -1;
  if(i && save[i-1][k]) ini=save[i-1][k];
  if(save[i][k-1]) fim=save[i][k-1]+1;

  rep(l, ini, fim){
    ll aux = solve(l+1, k-1)+mc[i][l];
    if(ret>aux){
      best=l;
      ret=aux;
    }
  }
  save[i][k]=best;
  return pd[i][k]=ret;
}

int main()
{
  rep(i, 0, N) scanf("%lld", &n[i]);

  se[0]=n[0];
  rep(i, 1, N) se[i]=se[i-1]+n[i];

  sd[N-1]=n[N-1];
  for(int i=N-2; i>=0; i--) sd[i]=sd[i+1]+n[i];

  rep(i, 1, N) pd[0][i]=pd[0][i-1]+se[i-1];
  for(int i=N-2; i>=0; i--) pd[N-1][i]=pd[N-1][i+1]+sd[i+1];

  rep(i, 1, N){
    rep(j, i+1, N)
      pd[i][j]=pd[i-1][j]-n[i-1]*(j-i+1);
  }
  for(int i=N-2; i>=0; i--){
    for(int j=i-1; j>=0; j--)
      pd[i][j]=pd[i+1][j]-n[i+1]*(i-j+1);
  }

  rep(i, 0, N){
    if(pd[i][i+1] < pd[i+1][i]) mc[i][i+1]=pd[i][i+1], save[i][
```

```
        i+1]=i+1;
    else            mc[i][i+1]=pd[i+1][i], save[i][i+1]=i;
    rep(j, i+2, N){
      int ini=save[i][j-1];
      mc[i][j]=pd[i][ini]+pd[j][ini], save[i][j]=ini;
      rep(k, ini+1, j+1){
        ll a=pd[i][k]+pd[j][k];
        if(mc[i][j]<=a) break;
        mc[i][j]=a;
        save[i][j]=k;
      }
    }
    rep(j, 0, N+1){
      pd[i][j]=-1,
      save[i][j]=0;
    }
  }

  rep(j, 0, N+1) pd[N][j]=-1, save[N][j]=0;

  solve();

  return 0;

}
```

9.3 Convex Hull Trick

```
bool bad(int l1,int l2,int l3){
  return (B[l3]-B[l1])*(M[l1]-M[l2])<(B[l2]-B[l1])*(M[l1]-M[l3
      ]);
}
void add(long long m,long long b){
  M.push_back(m);
  B.push_back(b);
  while (M.size()>=3&&bad(M.size()-3,M.size()-2,M.size()-1)){
    M.erase(M.end()-2);
    B.erase(B.end()-2);
  }
}
long long query(long long x){
  if (pointer>=M.size())
    pointer=M.size()-1;
  while (pointer<M.size()-1&&
    M[pointer+1]*x+B[pointer+1]<M[pointer]*x+B[pointer])
    pointer++;
  return M[pointer]*x+B[pointer];
}

struct hux{
  int a, b, id;
};
bool my_sort(hux a, hux b){
  return a.b!=b.b ? a.b > b.b : a.a > b.a;
}

const ll LINF = 1LL << 52;
const double EPS = 1e-9;
const int MAXV = 100010;

double intersept(hux a, hux b){
  return double(b.b-a.b)/(a.a-b.a);
}

vector < pair< double, double > > convex_hux( const vector <hux
    > &v ){

  int p=0, n=v.size(), bestai=v[0].a;
  double cross=0.0;
  pair<double, int> aux;

  priority_queue < pair<double, int> > pq;
  vector < pair< double, double > > ret(n+1, mp(-1, -1));

  pq.push(mp(cross, p));
  ret[v[p].id].F=cross,
  ret[v[p].id].S=LINF;

  rep(i, 1, n){

    aux=pq.top();
    cross=aux.F,
    p=aux.S;

    if(v[i].a <= bestai) continue; bestai=v[i].a;

    double new_cross = intersept(v[i], v[p]);
    while( new_cross <= cross + EPS ){
      pq.pop();
      ret[v[p].id]=mp(-1.0, -1.0);

      aux=pq.top();
      cross=aux.F,
      p=aux.S;

      new_cross = intersept(v[i], v[p]);
    }

    pq.push(mp(new_cross, i));
    ret[v[p].id].S=new_cross;
    ret[v[i].id].F=new_cross;
    ret[v[i].id].S=LINF;

  }

  //rep(i, 0, n) cout << ret[i].F << " " << ret[i].S << "\n";
```

```
  return ret;

}
```

9.4 Longest Increasing Subsequence

```
//Maior subsequencia crescente
#define MAX_N 100
int vet[MAX_N], P[MAX_N], N;
void reconstruct_print(int end){
  int x = end;
  stack<int> s;
  while(P[x] >= 0){
    s.push(vet[x]);
    x = P[x];
  }
  printf("%d", vet[x]);
  while(!s.empty()){
    printf(", %d", s.top());
    s.pop();
  }
}
int lis(){
  int L[MAX_N], L_id[MAX_N];
  int li = 0, lf = 0; //lis ini, lis end
  rep(i, 0, N){
    int pos = lower_bound(L, L+li, vet[i]) - L;
    L[pos] = vet[i];
    L_id[pos] = i;
    P[i] = pos ? L_id[pos - 1] : -1;
    if (pos + 1 > li){
      li = pos + 1;
      lf = i;
    }
  }
  reconstruct_print(lf);
  return li;
}
```

9.5 Kadane 1D

```
//  Encontra maior soma contigua positiva num vetor em O(N)
//  [s,f] contem o intervalo de maior soma
int Kadane1D(int vet[], int N, int &s, int &f){
  int ret = -INF, sum, saux;
  sum = s = f = saux = 0;
  rep(i, 0, N){
    sum += vet[i];
    if(sum > ret){ ret = sum; s = saux; f = i; }
    if(sum < 0) { sum = 0; saux = i+1; }
  }
  return ret;
}
```

9.6 Kadane 2D

```
//maior soma de uma sub-matriz a partir de valores positivos
//[x1,y1]=upper-left, [x2,y2]=bottom-right
int L, C, pd[MAX_L], mat[MAX_L][MAX_C];
int x1, y1, x2, y2;
int Kadane2D(){
  int ret = 0, aux;
  rep(left, 0, C){
    rep(i, 0, L) pd[i] = 0;
    rep(right, left, C){
      rep(i, 0, L) pd[i] += mat[i][right];
      int sum = aux = 0;
      rep(i, 0, L){ // Kadane1D
        sum += pd[i];
        if(sum>ret)
          ret=sum, x1=aux, y1=left, x2=i, y2=right;
        if(sum < 0) sum = 0, aux = i+1;
      }
    }
  }
  return ret;
}
```

9.7 Merge Sort com num de Inversoes

```
void merge(int* arr, int size1, int size2, ll &inversions) {
  int temp[size1+size2+2];
  int ptr1=0, ptr2=0;

  while (ptr1+ptr2 < size1+size2) {
    if (ptr1 < size1 && arr[ptr1] <= arr[size1+ptr2] || ptr1 <
        size1 && ptr2 >= size2)
      temp[ptr1+ptr2] = arr[ptr1++];

    if (ptr2 < size2 && arr[size1+ptr2] < arr[ptr1] || ptr2 <
        size2 && ptr1 >= size1) {
      temp[ptr1+ptr2] = arr[size1+ptr2++];
      inversions += size1-ptr1;
    }
  }

  for (int i=0; i < size1+size2; i++)
    arr[i] = temp[i];
}

void mergeSort(int* arr, int size, ll &inversions) {
  if (size == 1) return;

  int size1 = size/2, size2 = size-size1;
```

```
  mergeSort(arr, size1, inversions);
  mergeSort(arr+size1, size2, inversions);
  merge(arr, size1, size2, inversions);
}
```

9.8 Quick Sort

```
//No main, chamar quicksort(array, 0, tam-1);
int partition(int s[], int l, int h) {
  int i, p, firsthigh;
  p = h;
  firsthigh = l;
  for (i=l; i < h; i++)
    if (s[i] < s[p]){
      swap(s[i], s[firsthigh]);
      firsthigh++;
    }
  swap(s[i], s[firsthigh]);
  return firsthigh;
}
void quicksort(int s[], int l, int h){
  int p;
  if ((h-l) > 0){
    p = partition(s,l,h);
    quicksort(s,l,p-1);
    quicksort(s,p+1,h);
  }
}
```

10 Miscelânia

10.1 Calendário

```
// Routines for performing computations on dates.  In these
    routines,
// months are expressed as integers from 1 to 12, days are
    expressed
// as integers from 1 to 31, and years are expressed as 4-digit
// integers.
string dayOfWeek[] = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat",
     "Sun"};

// converts Gregorian date to integer (Julian day number)
int dateToInt (int m, int d, int y){
  return
  1461 * (y + 4800 + (m - 14) / 12) / 4 +
  367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
  3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
  d - 32075;
}
// converts integer (Julian day number) to Gregorian date:
    month/day/year
void intToDate (int jd, int &m, int &d, int &y){
  int x, n, i, j;
  x = jd + 68569;
  n = 4 * x / 146097;
  x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001;
  x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447;
  d = x - 2447 * j / 80;
  x = j / 11;
  m = j + 2 - 12 * x;
  y = 100 * (n - 49) + i + x;
}
// converts integer (Julian day number) to day of week
string intToDay (int jd){
  return dayOfWeek[jd % 7];
}
int main (){
  int jd = dateToInt (3, 24, 2004);
  int m, d, y;
  intToDate (jd, m, d, y);
  string day = intToDay (jd);
  // expected output:
  //    2453089
  //    3/24/2004
  //    Wed
  cout << jd << endl
  << m << "/" << d << "/" << y << endl
  << day << endl;
}
```