Índice

Números

2^2	4	3^2	9	2!	2
2^3	8	3^3	27	3!	6
2^4	16	3^4	81	4!	24
2^5	32	3^5	243	5!	120
2^6	64	3^6	729	6!	720
2^7	128	3^7	2.187	7!	5.040
2^8	256	3^8	6.561	8!	40.320
2^9	512	3^9	19.683	9!	362.880
2^10	1.024	3^10	59.049	10!	3.628.800
2^11	2.048	3^11	177.147	11!	39.916.800
2^12	4.096	3^12	531.441	12!	[limite int] 479.001.600
2^13	8.192	3^13	1.594.323	13!	6.227.020.800
2^14	16.384	3^14	4.782.969	14!	87.178.291.200
2^15	32.768	3^15	14.348.907	15!	1.307.674.368.000
2^16	65.536	3^16	43.046.721	16!	20.922.789.888.000
2^17	131.072	3^17	129.140.163	17!	355.687.428.096.000
2^18	262.144	3^18	387.420.489	18!	6.402.373.705.728.000
2^19	524.288	3^19	1.162.261.467	19!	121.645.100.408.832.000
2^20	1.048.576	3^20	3.486.784.401	20!	2.432.902.008.176.640.000
					[limite unsigned long long]

Matemática: Cálculo e formulas

Somatórios de Progressões Geométricas:

Finita: $$S_n = \frac{a_1(q^n - 1)}{q - 1}$$ Infinita: $$S_\infty = \sum_{n=1}^{\infty} a_1 q^{n-1} = \frac{a_1}{1 - q}$$

Identities:

$$\sin x = \frac{1}{\csc x}, \qquad \cos x = \frac{1}{\sec x}, \qquad \sin 2x = 2\sin x \cos x,$$

$$\tan x = \frac{1}{\cot x}, \qquad \sin^2 x + \cos^2 x = 1, \qquad \cos 2x = \cos^2 x - \sin^2 x,$$

$$1 + \tan^2 x = \sec^2 x, \qquad 1 + \cot^2 x = \csc^2 x, \qquad \cos 2x = 1 - 2\sin^2 x,$$

$$\sin x = \cos\left(\tfrac{\pi}{2} - x\right), \qquad \sin x = \sin(\pi - x), \qquad \tan 2x = \frac{2\tan x}{1 - \tan^2 x},$$

$$\cos x = -\cos(\pi - x), \qquad \tan x = \cot\left(\tfrac{\pi}{2} - x\right), \qquad \sin 2x = \frac{2\tan x}{1 + \tan^2 x},$$

$$\cot x = -\cot(\pi - x), \qquad \csc x = \cot\tfrac{x}{2} - \cot x, \qquad \cos 2x = 2\cos^2 x - 1,$$

$$\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y, \qquad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$$

$$\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y, \qquad \cot 2x = \frac{\cot^2 x - 1}{2\cot x},$$

$$\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y},$$

$$\cot(x \pm y) = \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y},$$

$$\sin(x + y)\sin(x - y) = \sin^2 x - \sin^2 y,$$

$$\cos(x + y)\cos(x - y) = \cos^2 x - \sin^2 y.$$

θ	$\sin\theta$	$\cos\theta$	$\tan\theta$	
0	0	1	0	$\tan(x$
$\frac{\pi}{6}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{3}}{3}$	$\cot(x$
$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$	1	
$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	$\sqrt{3}$	$\sin(x$
$\frac{\pi}{2}$	1	0	∞	$\cos(x$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \qquad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}$$

Derivatives:

1. $\dfrac{d(cu)}{dx} = c\dfrac{du}{dx},$
2. $\dfrac{d(u+v)}{dx} = \dfrac{du}{dx} + \dfrac{dv}{dx},$
3. $\dfrac{d(uv)}{dx} = u\dfrac{dv}{dx} + v\dfrac{du}{dx},$
4. $\dfrac{d(u^n)}{dx} = nu^{n-1}\dfrac{du}{dx},$
5. $\dfrac{d(u/v)}{dx} = \dfrac{v\left(\frac{du}{dx}\right) - u\left(\frac{dv}{dx}\right)}{v^2},$
6. $\dfrac{d(e^{cu})}{dx} = ce^{cu}\dfrac{du}{dx},$
7. $\dfrac{d(c^u)}{dx} = (\ln c)c^u\dfrac{du}{dx},$
8. $\dfrac{d(\ln u)}{dx} = \dfrac{1}{u}\dfrac{du}{dx},$
9. $\dfrac{d(\sin u)}{dx} = \cos u\dfrac{du}{dx},$
10. $\dfrac{d(\cos u)}{dx} = -\sin u\dfrac{du}{dx},$
11. $\dfrac{d(\tan u)}{dx} = \sec^2 u\dfrac{du}{dx},$
12. $\dfrac{d(\cot u)}{dx} = \csc^2 u\dfrac{du}{dx},$
13. $\dfrac{d(\sec u)}{dx} = \tan u \sec u\dfrac{du}{dx},$
14. $\dfrac{d(\csc u)}{dx} = -\cot u \csc u\dfrac{du}{dx},$
15. $\dfrac{d(\arcsin u)}{dx} = \dfrac{1}{\sqrt{1-u^2}}\dfrac{du}{dx},$
16. $\dfrac{d(\arccos u)}{dx} = \dfrac{-1}{\sqrt{1-u^2}}\dfrac{du}{dx},$
17. $\dfrac{d(\arctan u)}{dx} = \dfrac{1}{1+u^2}\dfrac{du}{dx},$
18. $\dfrac{d(\operatorname{arccot} u)}{dx} = \dfrac{-1}{1+u^2}\dfrac{du}{dx},$
19. $\dfrac{d(\operatorname{arcsec} u)}{dx} = \dfrac{1}{u\sqrt{1-u^2}}\dfrac{du}{dx},$
20. $\dfrac{d(\operatorname{arccsc} u)}{dx} = \dfrac{-1}{u\sqrt{1-u^2}}\dfrac{du}{dx},$
21. $\dfrac{d(\sinh u)}{dx} = \cosh u\dfrac{du}{dx},$
22. $\dfrac{d(\cosh u)}{dx} = \sinh u\dfrac{du}{dx},$
23. $\dfrac{d(\tanh u)}{dx} = \operatorname{sech}^2 u\dfrac{du}{dx},$
24. $\dfrac{d(\coth u)}{dx} = -\operatorname{csch}^2 u\dfrac{du}{dx},$
25. $\dfrac{d(\operatorname{sech} u)}{dx} = -\operatorname{sech} u \tanh u\dfrac{du}{dx},$
26. $\dfrac{d(\operatorname{csch} u)}{dx} = -\operatorname{csch} u \coth u\dfrac{du}{dx},$
27. $\dfrac{d(\operatorname{arcsinh} u)}{dx} = \dfrac{1}{\sqrt{1+u^2}}\dfrac{du}{dx},$
28. $\dfrac{d(\operatorname{arccosh} u)}{dx} = \dfrac{1}{\sqrt{u^2-1}}\dfrac{du}{dx},$
29. $\dfrac{d(\operatorname{arctanh} u)}{dx} = \dfrac{1}{1-u^2}\dfrac{du}{dx},$
30. $\dfrac{d(\operatorname{arccoth} u)}{dx} = \dfrac{1}{u^2-1}\dfrac{du}{dx},$
31. $\dfrac{d(\operatorname{arcsech} u)}{dx} = \dfrac{-1}{u\sqrt{1-u^2}}\dfrac{du}{dx},$
32. $\dfrac{d(\operatorname{arccsch} u)}{dx} = \dfrac{-1}{|u|\sqrt{1+u^2}}\dfrac{du}{dx}.$

Integrals:

1. $\displaystyle\int cu\,dx = c\int u\,dx,$
2. $\displaystyle\int (u+v)\,dx = \int u\,dx + \int v\,dx,$
3. $\displaystyle\int x^n\,dx = \frac{1}{n+1}x^{n+1}, \quad n \neq -1,$
4. $\displaystyle\int \frac{1}{x}dx = \ln x,$
5. $\displaystyle\int e^x\,dx = e^x,$
6. $\displaystyle\int \frac{dx}{1+x^2} = \arctan x,$
7. $\displaystyle\int u\frac{dv}{dx}dx = uv - \int v\frac{du}{dx}dx,$
8. $\displaystyle\int \sin x\,dx = -\cos x,$
9. $\displaystyle\int \cos x\,dx = \sin x,$

10. $\int \tan x\,dx = -\ln|\cos x|,$ 11. $\int \cot x\,dx = \ln|\cos x|,$

12. $\int \sec x\,dx = \ln|\sec x + \tan x|,$ 13. $\int \csc x\,dx = \ln|\csc x + \cot x|,$

14. $\int \arcsin\frac{x}{a}dx = \arcsin\frac{x}{a} + \sqrt{a^2 - x^2}, \quad a > 0,$

15. $\int \arccos\frac{x}{a}dx = \arccos\frac{x}{a} - \sqrt{a^2 - x^2}, \quad a > 0,$

17. $\int \sin^2(ax)dx = \frac{1}{2a}\big(ax - \sin(ax)\cos(ax)\big),$

16. $\int \arctan\frac{x}{a}dx = x\arctan\frac{x}{a} - \frac{a}{2}\ln(a^2 + x^2), \quad a > 0,$

18. $\int \cos^2(ax)dx = \frac{1}{2a}\big(ax + \sin(ax)\cos(ax)\big),$

19. $\int \sec^2 x\,dx = \tan x,$ 20. $\int \csc^2 x\,dx = -\cot x,$

Matemática: Wilson's Theorem

A natural number n > 1 is prime if and only if

$$(n-1)! \equiv -1 \pmod{n}$$

Matemática: Pick's Theorem

Usefull in a lattice polygon (all vertice's coordinates are integers)

A = Area of polygon
i = number of lattice points in the interior of polygon
b = number of lattice points in the boundary of polygon

$$A = i + \frac{b}{2} - 1.$$

Matemática: Euler's Totient Function

In number theory, the **totient** $\varphi(n)$ of a positive integer *n* is defined to be the number of positive integers less than or equal to *n* that are coprime to *n*.

$$\varphi(n) = (p_1 - 1)p_1^{k_1-1} \cdots (p_r - 1)p_r^{k_r-1}.$$

Matemática: Catalão e Stirling

Catalan numbers. Catalan numbers are defined by the recurrence:

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

A closed formula for Catalan numbers is:

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

Stirling numbers of the first kind. These are the number of permutations of I_n with exactly k disjoint cycles. They obey the recurrence:

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

Stirling numbers of the second kind. These are the number of ways to partition I_n into exactly k sets. They obey the recurrence:

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

Matemática: Aritmética Modular

```
// maior divisor comum
int gcd(int x, int y) { return y ? gcd(y, x % y) : abs(x); }

// Euclides estendido: x,y tais que a*x + b*y = gcd(a,b)
pair<int,int> euclides(int a, int b) {
    if (b==0) return pair<int,int>(1,0);
    pair<int,int> rec = euclides(b, a%b);
    return pair<int,int>(rec.second, rec.first - (a/b)*rec.second);
}
//inverso modular: (a*inv(a,n))%n == 1
int invMod(int a, int n) { return euclides(a,n).first; }

// teorema chines do resto, retorna x tal que x = a[i] % p[i]
int chineseRemainder(int qtd, int *a, int *p) {
    int M = 1, x = 0;
    for (int i = 0 ; i < qtd ; i++) M*=p[i];
    for (int i = 0 ; i < qtd ; i++) x+=a[i]*invMod(M/p[i],p[i])*(M/p[i]);
    return (((x%M)+M)%M);
}
```

Matemática: Eliminação Gaussiana

```
//coeficientes das variáveis em mat e termos independentes em indep
void gaussianElimination(){
    int pivo = 0;
    int linha = 0, npivo;
    double fator;
    bool achou;

    while(pivo < n){
        achou = false;
        for(int i = linha; i < m; i++){
                if(cmpEPS(mat[i][pivo], 0) != 0){
                    achou = true;
                    npivo = i;
                    break;
                }
        }
        if(achou){
                if(npivo != linha){
                    for(int i = 0; i < n; i++)
                        swap(mat[npivo][i], mat[linha][i]);
                    swap(indep[npivo], indep[linha]);
                }
                for(int i = linha+1; i < m; i++){
                    if(cmpEPS(mat[i][pivo], 0) != 0){
                        fator = mat[i][pivo];
                        for(int j = pivo; j < n; j++){
                            mat[i][j] = mat[i][j]*mat[linha][pivo] - mat[linha][j]*fator;
                        }
                        indep[i] = indep[i]*mat[linha][pivo] - indep[linha]*fator;
                    }
                }
                linha++;
        }
        pivo++;
    }
    for(int i = linha; i < m;i++){
        if(cmpEPS(indep[i], 0) != 0){
            printf("Inconsistent data.\n");
            return;
        }
    }
    if(linha < n){
        printf("Multiple solutions.\n");
    }else{
        for(int i = linha-1; i >= 0; i--){
            indep[i] = (indep[i]/(mat[i][i]));

            for(int j = i -1; j >= 0; j--){
                indep[j] = indep[j] - mat[j][i]*indep[i];
            }
        }
    }
}
```

Matemática: Transformada Rápida de Fourier

```
#include <complex>

void inverterBits(int inv) {
    for (int i = 0; i < N; i++) {
        if (inv == -1) {
            ord[i] = i;
        } else {
            ord[i] = ordTotal[i]>>(15-logN);
        }
    }
}

void fft(int inv) {
    inverterBits(inv);

    complex<double> wPot;
    complex<double> wAtual;

    complex<double> segPart;
    complex<double> priPart;

    int m;
    for (int i = 1; i <= logN; i++) {
        m = (1<<i);

        wPot = exp(complex<double>(0,(inv*2*pi)/m));

        for (int k = 0; k < N; k+=m) {

            wAtual = complex<double>(1,0);
            for (int j = 0; j < m/2; j++) {
                segPart = wAtual*f[ord[k+j+m/2]];
                priPart = f[ord[k+j]];

                f[ord[k+j]] = priPart + segPart;
                f[ord[k+j+m/2]] = priPart - segPart;

                wAtual *= wPot;
            }

        }
    }
}

void invfft() {
    fft(-1);
    for (int i = 0; i < N; i++) {
        f[i] /= N;
    }
}

void preprocess() {
    N = 1<<15;
    logN = 15;
    for (int i = 0; i < N; i++) {
        ordTotal[i] = 0;
        for (int j = logN-1; j >= 0; j--) {
            if (i&(1 << j)) {
                ordTotal[i] += (1 << (logN-1-j));
            }
        }
    }
}

/*
Seguir seguintes passos:
  1) Guardar polinômio começando do termo de menor grau
  2) Polinômio final vai ter tamanho N = tam1+tam2-1
Ao usar FFT para multiplicar dois números:
  3) Aumentar os polinomios para menor potência de 2 maior ou igual a N (completa com
0s)
  4) Calcular FFT para cada número, multiplica resultados e calcula a inversa
  5) Usa código abaixo para achar o numero final
*/
void calcMult() {
    int carry = 0;
    mult[0] = valInt(real(f[ord[0]]))%10;
    for (int i = 1; i < Nverd; i++) {
        mult[i] = valInt(real(f[ord[i-1]]))/10 + valInt(real(f[ord[i]]))%10 + carry;
```

```
        carry = mult[i] / 10;
        mult[i] %= 10;
    }
    mult[Nverd] = valInt(real(f[ord[Nverd-1]]))/10+carry;

    int ini = Nverd;
    while (ini > 0 && mult[ini] == 0) {
        ini--;
    }
}
```

Geometria: Primitivas

```
#include <cstdio>
#include <algorithm>
#include <cmath>
#include <utility>

using namespace std;

const double EPS = 1E-8;

int cmpEPS(double a, double b){
    if(fabs(a-b) < EPS)return 0;
    return a < b ? -1 : 1;
}
struct Ponto {
    int x,y;

    Ponto(int x = 0, int y = 0) : x(x) , y(y) {}
    Ponto operator + (Ponto p){return Ponto(x+p.x, y+p.y);}
    Ponto operator - (Ponto p){return Ponto(x-p.x, y-p.y);}
    //cuidado com overflow daqui pra baixo
    int operator % (Ponto p){return x*p.y - y*p.x;}
    int operator * (Ponto p){ return x*p.x + y*p.y;}
    int norma2(){return x*x + y*y;}
    int dist(Ponto p){return (x-p.x)*(x-p.x) + (y-p.y)*(y-p.y);}
    Ponto operator / (int a){return Ponto(x/a, y/a);}
};


//verifica se o ponto p está no segmento p1-p2
bool pertenceSeg(Ponto p1, Ponto p2, Ponto p){
    int pvet;
    pvet = (p - p1)%(p2 - p1);
    if(pvet == 0){
        return p.x >= min(p1.x,p2.x) && p.x <= max(p1.x, p2.x)
            && p.y >= min(p1.y, p2.y) && p.y <= max(p1.y, p2.y);
    }else{
        return false;
    }
}
//verifica se os segmentos p1 e p2 se intersectam
bool intersectaSegmento(Ponto p1, Ponto p2, Ponto p3, Ponto p4){
    int d1,d2,d3,d4;

    d1 = (p1 - p3)%(p4 - p3);
    d2 = (p2 - p3)%(p4 - p3);
    d3 = (p3 - p1)%(p2-p1);
    d4 = (p4 - p1)%(p2-p1);

    if(((d1 < 0 && d2 > 0) || (d1 > 0 && d2 < 0)) &&
        ((d3 < 0 && d4 > 0) || (d3 > 0 && d4 < 0))){
        return true;
    }
    return pertenceSeg(p3,p4, p1) || pertenceSeg(p3,p4,p2) || pertenceSeg(p1,p2,p3)
        || pertenceSeg(p1,p2,p4);
}
//Calcula Distância do Ponto p ao segmento p1-p2
double distPontoSeg(Ponto p1, Ponto p2, Ponto p) {
    double dist;
    if ((p1-p2)*(p-p2) <= 0) {
        dist = sqrt((double)((p2-p)*(p2-p)));
    } else if ((p2-p1)*(p-p1) <= 0) {
        dist = sqrt((double)((p1-p)*(p1-p)));
    } else {
        dist = ((p2-p1)%(p-p1))/sqrt((double)((p2-p1)*(p2-p1)));
    }
    return fabs(dist);
}
//calcula 2*area do polígono p que tem tam lados
int calcArea2(Ponto *p, int tam){
    int res = 0;

    for(int i = 2; i < tam; i++){
        res += (p[i-1]-p[0])%(p[i]-p[0]);
    }
    return abs(res);
}
//Ponto deve ser de doubles, e os Pontos devem ser não colineares!
Ponto circumcenter(Ponto p, Ponto q, Ponto r) {
    Ponto a = p - r, b = q - r, c = Ponto(a * (p + r) / 2, b * (q + r) / 2);
    return Ponto(c % Ponto(a.y, b.y), Ponto(a.x, b.x) % c) / (a % b);
}
```

Geometria: Convex Hull

```
//compara pelo ângulo polar
Ponto pivot;
bool cmpPolar(Ponto p, Ponto q){
    int pvet = (p - pivot)%(q-pivot);
    if(pvet > 0){
        return true;
    }else if(pvet < 0){
        return false;
    }else{
        return (p-pivot)*(p-pivot) < (q-pivot)*(q-pivot);
    }
}
//encontra o invólucro  convexo no sentido anti-horário retirando pontos colineares
void convexHull(Ponto *pontos, int N, Ponto *hull, int &nConv) {
    int pivo;
    if(N > 0){
        pivo = 0;
        for(int i = 1; i < N; i++){
            if(pontos[i].y < pontos[pivo].y || (pontos[i].y == pontos[pivo].y &&
                pontos[i].x < pontos[pivo].x)){

                pivo = i;
            }
        }
        swap(pontos[0], pontos[pivo]);
        pivot = pontos[0];
        sort(pontos+1, pontos+N , cmpPolar);
        nConv = 0;
        hull[nConv++] = pontos[0];
        //lembre de trocar o  <= por cmpEPS  caso esteja usando doubles
        for(int i = 1; i < N; i++){
            while(nConv > 1 && (hull[nConv-1]-hull[nConv-2])%(pontos[i]-hull[nConv-1]) <=
0){
                nConv--;
            }
            hull[nConv++] = pontos[i];
        }
    }else{
        nConv = 0;
    }
}
```

Geometria: Ponto Polígono (Convexo)

```
//Determina se o ponto q pertence a um polígono convexo em O(log nConv)
//O polígono deve estar no  sentido anti-horário!
//O polígono está no array hull e tem nConv lados
bool insideConvex(Ponto *hull, int nConv, Ponto q) {
    int ini, med, fim;
    if(nConv < 3){
        return false;
    }
    ini = 1;
    fim = nConv-1;
    while(fim-ini > 1){
        med = (ini + fim)/2;

        if((hull[med]-hull[0])%(q - hull[0]) >= 0){
            ini = med;
        }else{
            fim = med;
        }
    }
    if( (hull[ini]-hull[0])%(q - hull[0]) >= 0 &&
        (hull[ini+1]-hull[ini])%(q - hull[ini]) >= 0 &&
            (hull[0]-hull[ini+1])%(q - hull[ini+1]) >= 0){
        return true;
    }else{
        return false;
    }
}
```

Geometria: Ponto Polígono (Qualquer)

```
//verifica se o ponto P pertence ao polígono poli que possui np lados
bool pertencePoli(Ponto *poli, int np, Ponto p){
    bool ehpar = true;
    for(int i = 0, j = np-1; i < np; i++){
        if(pertenceSeg(poli[j], poli[i], p)){
            return true;
        }
        if(poli[i].y > p.y && poli[j].y <= p.y){
            if((poli[i] - poli[j])%(p - poli[j]) < 0){
```

```
            ehpar = !ehpar;
        }
    }else if(poli[j].y > p.y && poli[i].y <= p.y){
        if((poli[j] - poli[i])%(p - poli[i]) < 0){
            ehpar = !ehpar;
        }
    }
    j = i;
  }
  return !ehpar;
}
```

Geometria: Closest Pair of Points

```
//Encontra o Par de pontos mais próximos em nlog^2(n)
//Entrada no array de pontos
Ponto *pontos;
int *yInd;
int INF;
bool cmpX(Ponto a, Ponto b){
  return a.x < b.x;
}
bool cmpY(int a, int b){
  return pontos[a].y < pontos[b].y;
}
int findMin(int ini, int fim){
  int ret = INF;
  intbarreira,temp, pos;
  if(fim - ini + 1 <= 3){
    for(int i = ini; i <= fim; i++){
      for(int j = i+1; j <= fim; j++){
        temp = pontos[i].dist(pontos[j]);
        if(ret > temp){
          ret = temp;
        }
      }
    }
  }else{
    barreira = pontos[(ini + fim)/2].x;
    ret = findMin(ini, (ini + fim)/2);
    temp = findMin((ini + fim)/2 + 1, fim);
    if(ret > temp)ret = temp;
    pos = 0;
    for(int i = ini; i <= fim; i++){
      if(abs(pontos[i].x - barreira) <= ret){
        yInd[pos++] = i;
      }
    }
    sort(yInd, yInd + pos,cmpY);
    for(int i = 0; i < pos; i++){
      for(int j = 1; j <= 7 && i + j < pos; j++){
        temp = pontos[yInd[i]].dist(pontos[yInd[i+j]]);
        if(temp < ret) ret = temp;
      }
    }
  }
  return ret;
}
```

Geometria: Minimum Enclosing Circle

```
typedef pair<Ponto,double> circle;
bool in_circle(circle C, Ponto p){
  return cmpEPS(C.first.dist(p) , C.second) <= 0;
}
//Ponto deve ser de doubles!
circle spanning_circle(Ponto *T, int n) {
  random_shuffle(T, T + n);
  circle C(Ponto(), 0);
  for (int i = 0; i < n; i++) if (!in_circle(C, T[i])) {
    C = circle(T[i], 0);
    for (int j = 0; j < i; j++) if (!in_circle(C, T[j])) {
      C = circle((T[i] + T[j]) / 2, T[i].dist(T[j]) / 2);
      for (int k = 0; k < j; k++) if (!in_circle(C, T[k])) {
        Ponto o = circumcenter(T[i], T[j], T[k]);
        C = circle(o, T[k].dist(o));
      }
    }
  }
  return C;
}
```

Geometria: Kd-Tree

```
//Kd-Tree, tem um array de indices e um array de pontos
bool cmpX(int a, int b){
    return pontos[a].x < pontos[b].x;
}
bool cmpY(int a, int b){
    return pontos[a].y < pontos[b].y;
}
void buildKdTree(int no, int left, int right, bool par){
    if(left > right){
        kdTree[no] = -1;
    }else if(left == right){
        kdTree[no] = indices[left];
        folha[no] = true;
    }else{
        folha[no] = false;
        if(par)
            nth_element(indices+left, indices + (left+right)/2, indices+right+1, cmpX);
        else
            nth_element(indices+left, indices + (left+right)/2, indices+right+1, cmpY);

        kdTree[no] = indices[(left+right)/2];
        buildKdTree(2*no,left, (left+right)/2-1, !par);
        buildKdTree(2*no+1, (left+right)/2 + 1, right, !par);
    }
}
int xq1,xq2,yq1,yq2;
bool get(int no, int mx, int Mx, int my, int My, bool par){
    if(kdTree[no] == -1)return false;
    if(Mx < xq1 || mx > xq2 || My < yq1 || my > yq2){
        return false;
    }
    if(mx >= xq1 && Mx <= xq2 && my >= yq1 && My <= yq2){
        return true;
    }
    bool ret = pontos[kdTree[no]].x >= xq1 && pontos[kdTree[no]].x <= xq2 &&
            pontos[kdTree[no]].y >= yq1 && pontos[kdTree[no]].y <= yq2;
    if(ret)
        return true;
    if(par && !folha[no]){
        return get(2*no, mx,pontos[kdTree[no]].x, my,My,!par) ||
            get(2*no+1, pontos[kdTree[no]].x, Mx, my,My,!par);
    }else if(!folha[no]){
        return get(2*no, mx, Mx, my, pontos[kdTree[no]].y, !par) ||
            get(2*no+1, mx, Mx, pontos[kdTree[no]].y, My, !par);
    }else
        return false;
}
```

Geometria: RANGE Tree com Fractional Cascading

```
//RANGE Tree com Fractional Cascading, NÃO FUNCIONA COM PONTOS COINCIDENTES
bool cmpX(int a, int b){
    return (x[a] < x[b]) || (x[a] == x[b] && y[a] < y[b]);
}
bool cmpY(int a, int b){
    return (y[a] < y[b]) || (y[a] == y[b] && x[a] < x[b]);
}
void initializeRangeTree(int np){
    for(int i = 0; i < np; i++){
        indX[i] = indY[i] = i;
    }

    sort(indX, indX + np, cmpX);
    sort(indY, indY + np, cmpY);

    for(int i = 0; i < np; i++)
        indice[i][0] = indY[i];
}
void buildFractionalCascading(int ini, int fim, int level){
    if(ini != fim){
        int med = (ini + fim)/2;
        int p1 = ini, p2 = med+1;
        for(int i = ini; i <= fim; i++){
            if((x[indice[i][level]] < x[indX[med]]) || (x[indice[i][level]] == x[indX[med]] &&
                y[indice[i][level]] <= y[indX[med]])){
                indice[p1++][level+1] = indice[i][level];
            }else{
                indice[p2++][level+1] = indice[i][level];
            }
        }
        p1 = ini;
        p2 = med+1;
```

```
        for(int i = ini; i <= fim; i++){
            while(p1 <= med && cmpY(indice[p1][level+1], indice[i][level])){
                p1++;
            }
            while(p2 <= fim && cmpY(indice[p2][level+1], indice[i][level])){
                p2++;
            }
            left[i][level] = p1;
            right[i][level] = p2;
        }
        buildFractionalCascading(ini, med, level+1);
        buildFractionalCascading(med+1,fim,level+1);
    }
}
long long xmq,xMq,ymq, yMq;
void add(int ini, int fim, int iniY, int fimY, int level){
    if(xMq < x[indX[ini]] || xmq > x[indX[fim]]){return;}
    if(iniY > fimY || iniY > fim){return false;}
    if(x[indX[ini]] >= xmq && x[indX[fim]] <= xMq){
        if(fimY > fim){
            fimY = fim;
        }
        if(y[indice[fimY][level]] > yMq){
            fimY--;
        }
        if(iniY <= fimY){
            //do anything
        }
        return;
    }
    add(ini, (ini+fim)/2, left[iniY][level], (fimY > fim) ? fimY : left[fimY][level], level+1);
    add((ini+fim)/2 + 1, fim,  right[iniY][level], (fimY > fim) ? fimY : right[fimY][level],
level+1);
}
int findY1(long long yy, int n){
    int ini = 0;
    int fim = n-1;
    int med;
    if(y[indY[fim]] < yy)return fim+1;
    while(ini != fim){
        med = (ini + fim)/2;
        if(y[indY[med]] >= yy){
            fim = med;
        }else{
            ini = med+1;
        }
    }
    return ini;
}
int findY2(long long yy, int n){
    int ini = 0;
    int fim = n-1;
    int med;
    if(y[indY[0]] > yy){
        return -1;
    }
    while(ini != fim){
        med = (ini + fim+1)/2;
        if(y[indY[med]] <= yy){
            ini = med;
        }else{
            fim = med-1;
        }
    }
    return ini;
}
```

Strings: KMP

```
///// KMP: acha e imprime as ocorrências do padrão no texto [não usar com padrão vazio]
int func[10100]; // tamanho máximo de "padrao"
void KMP (char *padrao, char *texto) {
    int k = -1; func[0] = -1;
    for (int i = 1 ; padrao[i] ; i++) {
        while (k > -1 && padrao[k+1] != padrao[i]) k = func[k];
        if (padrao[k+1] == padrao[i]) k++;
        func[i] = k;
    }
    k = -1;    // posicao alinhada no padrao
    for (int i = 0 ; texto[i] ; i++) {
        while (k > -1 && padrao[k+1] != texto[i]) k = func[k];
        if (padrao[k+1] == texto[i]) k++;
        if (!padrao[k+1]) {
            printf("%d\n",i-k);  // encontrou uma das ocorrencias
            k = func[k];
        }
    }
}
```

Strings: Aho-Corasick

```
///// Aho-corasick: lista as ocorrencias de varios padrões em um texto grande
// comentada: possivel otimizacao para problema decisão: padrao[i] aparece?
struct No {
    int fail;
    vector< pair<int,int> > out; // num e tamanho do padrao
    //bool marc;  // p/ decisao
    map<char, int> lista;
    int next; // aponta para o próximo sufixo que tenha out.size > 0
};
No arvore[1000003]; // quantida maxima de nos
//bool encontrado[1005]; // quantidade maxima de padroes, p/ decisao
int qtdNos, qtdPadroes;

// Função para inicializar
void inic() {
    arvore[0].fail = -1;
    arvore[0].lista.clear();
    arvore[0].out.clear();
    arvore[0].next = -1;
    qtdNos = 1;
    qtdPadroes = 0;
    //arvore[0].marc = false; // p/ decisao
    //memset(encontrado, false, sizeof(encontrado)); // p/ decisao
}

// Funcao para adicionar um padrao
void adicionar(char *padrao) {
    int no = 0, len = 0;
    for (int i = 0 ; padrao[i] ; i++, len++) {
        if (arvore[no].lista.find(padrao[i]) == arvore[no].lista.end()) {
            arvore[qtdNos].lista.clear(); arvore[qtdNos].out.clear();
            //arvore[qtdNos].marc = false; // p/ decisao
            arvore[no].lista[padrao[i]] = qtdNos;
            no = qtdNos++;
        } else no = arvore[no].lista[padrao[i]];
    }
    arvore[no].out.push_back(pair<int,int>(qtdPadroes++,len));
}

// Ativar Aho-corasick, ajustando funcoes de falha
```

```
void ativar() {
    int no,v,f,w;
    queue<int> fila;
    for (map<char,int>::iterator it = arvore[0].lista.begin() ; it != arvore[0].lista.end() ; it++) {
        arvore[no = it->second].fail = 0;
        arvore[no].next = arvore[0].out.size() ? 0 : -1;
        fila.push(no);
    }
    while (!fila.empty()) {
        no = fila.front(); fila.pop();
        for (map<char,int>::iterator it=arvore[no].lista.begin(); it!=arvore[no].lista.end(); it++){
            char c = it->first;
            v = it->second;
            fila.push(v);
            f = arvore[no].fail;
            while (arvore[f].lista.find(c) == arvore[f].lista.end()) {
                if (f == 0) { arvore[0].lista[c] = 0; break; }
                f = arvore[f].fail;
            }
            w = arvore[f].lista[c];
            arvore[v].fail = w;
            arvore[v].next = arvore[w].out.size() ? w : arvore[w].next;
        }
    }
}

// Buscar padroes no aho-corasik
void buscar(char *input) {
    int v, no = 0;
    for (int i = 0 ; input[i] ; i++) {
        while (arvore[no].lista.find(input[i]) == arvore[no].lista.end()) {
            if (no == 0) { arvore[0].lista[input[i]] = 0; break; }
            no = arvore[no].fail;
        }
        v = no = arvore[no].lista[input[i]];
        // marcar os encontrados
        while (v != -1 /* && !arvore[v].marc */ ) { // p/ decisao
            //arvore[v].marc = true; // p/ decisao: nao continua a lista
            for (int k = 0 ; k < arvore[v].out.size() ; k++) {
                //encontrado[arvore[v].out[k].first] = true; // p/ decisao
                printf("Padrao %d na posicao %d\n", arvore[v].out[k].first, i-
arvore[v].out[k].second+1);
            }
            v = arvore[v].next;
        }
    }
    // for (int i = 0 ; i < qtdPadroes ; i++) printf("%s\n", encontrado[i]?"y":"n"); // p/ decisão
}
```

Strings: Array de Sufixo (mais LCP e substrings distintas)

```
char input[50100];
int array[50100];
int val[50100], novoVal[50100];

int inc, len;
bool cmp(const int & a, const int & b) {
    return (a+inc < len ? val[a+inc] : -1) < (b+inc < len ? val[b+inc] : -1);
}

void criarArraySufixo() {
    len = strlen(input);
    for (int i = 0 ; i < len ; i++) {
        array[i] = i;
        val[i] = input[i];
    }

    inc = 0;
    sort(array, array+len, cmp);
    for (inc = 1 ; (inc>>1) < len ; inc<<=1) {
        int i,j;
        bool mudou = false;
        for (i = 0 ; i < len-1 ; i++) { // se tiver um cara no ultimo intervalo, ele é o ultimo já
            j = i+1;
            while (j < len && val[array[i]] == val[array[j]]) j++;
            if (j > i+1) {
                mudou = true;
                sort(array+i, array+j, cmp);
            }
            i = j-1;
        }
        if (!mudou) break;
        novoVal[array[0]] = 0;
        for (int i = 1 ; i < len ; i++) {
            novoVal[array[i]] = novoVal[array[i-1]];
            if (val[array[i]] > val[array[i-1]] || cmp(array[i-1],array[i])) novoVal[array[i]]++;
        }
        for (int i = 0 ; i < len ; i++) val[i] = novoVal[i];
    }
}

int rank[50100];
int height[50100];
void criarArrayHeight() {
    for (int i = 0 ; i < len ; i++) rank[array[i]] = i;
    int h = 0;
    for (int i = 0 ; i < len ; i++) {
        if (rank[i] > 0) {
            int j = array[rank[i]-1];
            while (i+h < len && j+h < len && input[i+h] == input[j+h]) h++;
            height[rank[i]] = h;
            if (h > 0) h--;
        }
    }
}

int calcSubstringDistintas(){
    int total = len - array[0];
    int ancestral;
    for (int i = 1 ; i < len; i++) {
        ancestral = height[i];
        total += (len-array[i])-height[i];
    }
    return total;
}
```

Grafos: AVL

```
struct No {
    int val, esq, dir, bal, filhos;
    No (int val = 0) : val(val), esq(-1), dir(-1), bal(0), filhos(0) {}
};

No nos[200100];
int raiz, qtdNos, proximoNo;

int getTam(int no) { return no == -1 ? 0 : 1 + nos[no].filhos; }
void ajeitar(int no) { nos[no].filhos = getTam(nos[no].esq) + getTam(nos[no].dir); }

pair<int,bool> rotacaoDireita(int no, bool mudouAltura) {
    int esq = nos[no].esq;
    if (nos[esq].bal <= 0) { // rotacao simples para a direita
        nos[no].esq = nos[esq].dir;
        nos[esq].dir = no;
        nos[no].bal  = nos[esq].bal == 0 ? -1 : 0;
        nos[esq].bal = nos[esq].bal == 0 ?  1 : 0;
        ajeitar(no);
        ajeitar(esq); // preserve essa ordem!
        return pair<int, bool>(esq, mudouAltura);
    } else { // rotacao dupla para a direita
        int v = nos[esq].dir;
        nos[esq].dir = nos[v].esq;
        nos[no].esq = nos[v].dir;
        nos[v].esq = esq;
        nos[v].dir = no;
        nos[no].bal  = nos[v].bal == -1 ?  1:0;
        nos[esq].bal = nos[v].bal ==  1 ? -1:0;
        nos[v].bal = 0;
        ajeitar(no);
        ajeitar(esq);
        ajeitar(v); // preserve essa ordem!
        return pair<int, bool>(v, mudouAltura);
    }
}

pair<int,bool> rotacaoEsquerda(int no, bool mudouAltura) {
    int dir = nos[no].dir;
    if (nos[dir].bal >= 0) { // rotacao simples para a direita
        nos[no].dir = nos[dir].esq;
        nos[dir].esq = no;
        nos[no].bal  = nos[dir].bal == 0 ?  1 : 0;
        nos[dir].bal = nos[dir].bal == 0 ? -1 : 0;
        ajeitar(no);
        ajeitar(dir); // preserve essa ordem!
        return pair<int, bool>(dir, mudouAltura);
    } else { // rotacao dupla para a direita
        int v = nos[dir].esq;
        nos[dir].esq = nos[v].dir;
        nos[no].dir = nos[v].esq;
        nos[v].dir = dir;
        nos[v].esq = no;
        nos[no].bal  = nos[v].bal ==  1 ? -1:0;
        nos[dir].bal = nos[v].bal == -1 ?  1:0;
        nos[v].bal = 0;
        ajeitar(no);
        ajeitar(dir);
        ajeitar(v); // preserve essa ordem!
        return pair<int, bool>(v, mudouAltura);
    }
}

pair<int, bool> inserir(int no, int val) {
    if (no == -1) {
        nos[proximoNo] = No(val);
        qtdNos++;
        return pair<int, bool>(proximoNo++, true);
    }

    if (val < nos[no].val) { // insere na esquerda
        pair<int, bool> ret = inserir(nos[no].esq, val);
        nos[no].esq = ret.first;
        ajeitar(no);
        if (!ret.second) return pair<int, bool>(no, false);
        else if (--nos[no].bal != -2) return pair<int, bool>(no, nos[no].bal == -1);
        else return rotacaoDireita(no, false);

    } else if (val > nos[no].val) { // insere na direita
        pair<int, bool> ret = inserir(nos[no].dir, val);
        nos[no].dir = ret.first;
```

```
        ajeitar(no);
        if (!ret.second) return pair<int, bool>(no, false);
        else if (++nos[no].bal != 2) return pair<int, bool>(no, nos[no].bal == 1);
        else return rotacaoEsquerda(no, false);

    } else { // valor repetido, esse código ignora a inserção
        return pair<int, bool> (no, false);
    }
}

pair<int, bool> remover(int no, int val) {
    if (no == -1) {
        qtdNos++;
        return pair<int, bool>(-1, false); // nao encontrado, ignora remoção
    }
    if (val < nos[no].val) { // remove na esquerda
        pair<int, bool> ret = remover(nos[no].esq, val);
        nos[no].esq = ret.first;
        ajeitar(no);
        if (!ret.second) return pair<int, bool>(no, false);
        else if (++nos[no].bal != 2) return pair<int, bool>(no, nos[no].bal == 0);
        else return rotacaoEsquerda(no, true);

    } else { // remove na direita, ou ele proprio
        if (val == nos[no].val) { // remove aqui (libera posição "no" ou "sucessor")
            if (nos[no].dir == -1) return pair<int, bool> (nos[no].esq, true);
            if (nos[no].esq == -1) return pair<int, bool> (nos[no].dir, true);
            int sucessor = nos[no].dir;
            while (nos[sucessor].esq != -1) sucessor = nos[sucessor].esq;
            val = nos[no].val = nos[sucessor].val;
        }
        pair<int, bool> ret = remover(nos[no].dir, val);
        nos[no].dir = ret.first;
        ajeitar(no);
        if (!ret.second) return pair<int, bool>(no, false);
        else if (--nos[no].bal != -2) return pair<int, bool>(no, nos[no].bal == 0);
        else return rotacaoDireita(no, true);
    }
}

// inserir(val): Insere um valor na AVL
void inserir(int val) { raiz = inserir(raiz, val).first; }

// buscarKesimo(k): Busca o kesimo elemento da arvore. 0 <= k < qtdNos
int buscarKesimo(int k, int no = raiz, int a = 0, int b = qtdNos-1) {
    int pos = a + getTam(nos[no].esq);
    if (k < pos)      return buscarKesimo(k, nos[no].esq, a, pos-1);
    else if (k > pos) return buscarKesimo(k, nos[no].dir, pos+1, b);
    else return nos[no].val;
}

// menoresQue(x): Retorna a quantidade de elementos menores que x
int menoresQue(int x, int no = raiz) {
    if (no == -1) return 0;
    if (x < nos[no].val) return menoresQue(x, nos[no].esq);
    if (x == nos[no].val) return getTam(nos[no].esq);
    return menoresQue(x, nos[no].dir) + getTam(nos[no].esq) + 1;
}

// remover(val): Remove val da árvore, se ele existir
void remover(int val) {
    raiz = remover(raiz, val).first;
    qtdNos--;
}

// chame esse método para limpar a AVL e poder começar a usar
void inicializar() {
    raiz = -1;
    qtdNos = 0;
    proximoNo = 0;
}
```

Grafos: Componentes fortemente conexos

```
int dfs(int no){
    int minimo,w, temp;
    minimo = dfs_number[no] = contador++;
    pilha.push(no);

    for(int i = 0; i < grau[no]; i++){
        w = adj[no][i];
        if(dfs_number[w] == -1){
            temp = dfs(w);
            if(temp < minimo)
                minimo = temp;
        }else if(scc[w] == -1){
            if(dfs_number[w] < minimo)
                minimo = dfs_number[w];
        }
    }
    if(minimo == dfs_number[no]){
        nscc++;
        while(pilha.top() != no){
            scc[pilha.top()] = nscc;
            pilha.pop();
        }
        scc[pilha.top()] = nscc;
        pilha.pop();
    }
    return minimo;
}
```

Grafos: Componentes vértice-biconexos

```
//Componentes Vértice-Biconexos, inicializa dfs_number com -1
void generateBC(int no){
    while(pilha.top() != no){
        //pilha.top() eh mais um vertice
        pilha.pop();
    }
    //no eh outro
    pilha.pop();
}
int dfs(int no){
    int minimo, nextAr, temp, w;
    minimo = dfs_number[no] = contador++;
    pilha.push(no);
    nextAr = list[no];
    while(nextAr != -1){
        w = dest[nextAr];
        if(dfs_number[w] == -1){
            temp = dfs(w);
            if(temp < minimo)
                minimo = temp;
            if(temp >= dfs_number[no]){
                nbc++;
                pilha.push(no);
                generateBC(w);
            }
        }else if(dfs_number[w] < minimo){
            minimo = dfs_number[w];
        }
        nextAr = next[nextAr];
    }
    return minimo;
}
```

Grafos: Pontes

```
//Lembrar caso especial da raiz em ponto de articulacao
int dfs(int u) {
    num[u] = nextNum++;
    int minimo = num[u];

    int a = listas[u];
    int v, temp;
    while (a != -1) {

        v = ar[a][1];

        if (num[v] == -1) {

            paiAr[v] = a;
            temp = dfs(v);

            if (temp < minimo) {
```

```
                minimo = temp;
            }

        } else if (paiAr[u] != (a^1)) {
            if (num[v] < minimo) {
                minimo = num[v];
            }
        }

        a = prox[a];
    }

    if (paiAr[u] != -1 && minimo > num[ar[paiAr[u]][0]]) {
        res[qtdRes++] = paiAr[u];
    }

    return minimo;
}
```

Grafos: Caminho euleriano

```
//Lembrar de ver se o grafo pode ser desconexo!!!!!
//Ligar os vertices de grau impar em caso de caminho
void cicloEuleriano(int u) {
    int v,a;
    while (nextAr[u] < qtdAdj[u]) {
        a = listas[u][nextAr[u]];

        if (!markAr[a]) {
            markAr[a] = true;

            if (ar[a][0] == u) {
                v = ar[a][1];
            } else {
                v = ar[a][0];
            }

            nextAr[u]++;
            cicloEuleriano(v);

            cam[tamCam++] = a;
        } else {
            nextAr[u]++;
        }
    }
}
```

Grafos: Corte mínimo (Stoer-Wagner)

```
int corteFase() {
    memset(somas, 0, sizeof(somas));
    memset(estaConj, false, sizeof(estaConj));

    int u, v;
    int qtdConj = 0;
    while (qtdConj < qtdVert) {
        u = -1;
        for (int i = 0; i < N; i++) {
            if (existe[i] && !estaConj[i]) {
                if (u == -1 || somas[i] > somas[u]) {
                    u = i;
                }
            }
        }

        conj[qtdConj++] = u;
        estaConj[u] = true;
        for (int i = 0; i < N; i++) {
            if (existe[i] && !estaConj[i]) {
                somas[i] += matriz[u][i];
            }
        }
    }

    //merge
    u = conj[qtdConj-2];
    v = conj[qtdConj-1];
    int corte = matriz[u][v];
    for (int i = 0; i < N; i++) {
        if (existe[i] && i != u) {
            corte += matriz[v][i];

            matriz[u][i] += matriz[v][i];
            matriz[i][u] += matriz[i][v];
        }
```

```
    }
    existe[v] = false;
    qtdVert--;

    return corte;
}

int minCut() {
    int menor = 0x63636363;
    if (N == 1) {
        menor = 0;
    }

    memset(existe, true, sizeof(existe));
    qtdVert = N;
    int temp;
    while (qtdVert > 1) {
        temp = corteFase();

        if (temp < menor) {
            menor = temp;
        }
    }

    return menor;
}
```

Grafos: Blossom (Edmonds)

```
int n;
int grau[MAXV];
int adj[MAXV][MAXV];
int match[MAXV];
int root[MAXV];
int pai[MAXV];
int rep[MAXV];
int tam[MAXV];
int base[MAXV];
int ponteFirst[MAXV];
int ponteSecond[MAXV];
int status[MAXV];
bool visitado[MAXV];
int fila[MAXV];
int findSet(int no){
    if(rep[no] != no){
        rep[no] = findSet(rep[no]);
    }
    return rep[no];
}
void unir(int a, int b){
    if(tam[a] > tam[b]){
        rep[b] = a;
    }else{
        rep[a] = b;
    }
    if(tam[a] == tam[b]){
        tam[b]++;
    }
}
int findLCA(int a, int b){
    int baseA = base[findSet(a)];
    int retorno;
    if(visitado[baseA]){
        retorno =  baseA;
    }else{
        visitado[baseA] = true;
        if(b != -1){
            retorno = findLCA(b, pai[baseA]);
        }else{
            retorno = findLCA(pai[baseA], b);
        }
        visitado[baseA] = false;
    }
    return retorno;
}
void aumenta(int u, int v){
    if(u != v){
        if(status[u] == 0){
            aumenta(pai[pai[u]], v);
            match[pai[u]] = pai[pai[u]];
            match[pai[pai[u]]] = pai[u];
        }else{
            aumenta(ponteFirst[u], match[u]);
```

```
            aumenta(ponteSecond[u], v);
            match[ponteFirst[u]] = ponteSecond[u];
            match[ponteSecond[u]] = ponteFirst[u];
        }
    }
}
bool edmonds(){
    int ini, fim, atual, w, repW, lca, repAtual;
    ini = fim = 0;
    for(int i = 0; i < n; i++){
        rep[i] = base[i] = i;
        tam[i] = 1;
        pai[i] = -1;

        if(match[i] == -1){
            fila[fim++] = i;
            root[i] = i;
            status[i] = 0;
        }else{
            status[i] = -1;
        }
    }
    while(ini != fim){
        atual = fila[ini++];
        for(int i = 0; i < grau[atual]; i++){
            w = adj[atual][i];
            if(status[w] == -1){
                //adiciona w a árvore
                status[w] = 1;
                status[match[w]] = 0;
                fila[fim++] = match[w];
                root[match[w]] = root[w] = root[atual];
                pai[w] = atual;
                pai[match[w]] = w;
            }else{
                repW = base[findSet(w)];
                repAtual = base[findSet(atual)];
                if(status[repW] == 0 && root[repW] != root[repAtual]){
                    aumenta(atual, root[atual]);
                    aumenta(w,root[w]);
                    match[atual] = w;
                    match[w] = atual;
                    return true;
                }else if(status[repW] == 0 && repW != repAtual){
                    lca = findLCA(w,atual);
                    for(int j = repW; j != lca; j = base[findSet(pai[j])]){
                        unir(findSet(lca), findSet(j));
                        if(status[j] == 1){
                            fila[fim++] = j;
                            ponteFirst[j] = w;
                            ponteSecond[j] = atual;
                        }
                        base[findSet(lca)] = lca;
                    }
                    for(int j = repAtual; j != lca; j = base[findSet(pai[j])]){
                        unir(findSet(lca), findSet(j));
                        if(status[j] == 1){
                            fila[fim++] = j;
                            ponteFirst[j] = atual;
                            ponteSecond[j] = w;
                        }
                        base[findSet(lca)] = lca;
                    }
                }
            }
        }
    }
    return false;
}
```

Grafos: Minimum Mean Weight Cycle (Karp)

```
//Minimum Mean Weight Cycle, deve ser usado em um SCC
int dist[N][N];
void karp(){
    int arAtual, tot,w;
    tot = total[scc[source]];
    for(int i = 0; i < n; i++)
        for(int k = 0;  k <= tot;k++)
            dist[i][k] = INF;
    dist[source][0] = 0;
    for(int k = 1; k <= tot; k++){
        for(int i = 0; i < n; i++){
```

```
            if(scc[i] == scc[source]){
                arAtual = list[i];
                while(arAtual != -1){
                    w = dest[arAtual];
                    if(scc[w] == scc[source]){
                        if(dist[i][k-1] + peso[arAtual] < dist[w][k]){
                            dist[w][k] = dist[i][k-1] + peso[arAtual];
                        }
                    }
                    arAtual = next[arAtual];
                }
            }
        }
    }
    double maior, temp;
    for(int i = 0;  i< n; i++){
        if(scc[i] != scc[source])continue;
        maior = 0;
        for(int k = 0; k < tot; k++){
            temp = (dist[i][tot]-dist[i][k])/(double)(tot-k);
            if(temp > maior)
                maior = temp;
        }
        if(minimumMeanCycle > maior){
            minimumMeanCycle = maior;
        }
    }
}
```

Grafos: LCA

```
int calc(int s, int t) {
    //level comeca de 0 (na raiz)
    int log;
    if (level[s] > level[t]) {
        swap(s,t);
    }
    if (level[s] < level[t]) {
        for (log = 1; (1 << log) <= level[t] ; log++);
        log--;
        for (int i = log ; i >= 0 ; i--) {
            if (level[t] - (1<<i) >= level[s]) {
                t = pai[t][i];
            }
        }
    }

    if (s != t) {
        for (log = 1 ; (1 << log) <= level[t] ; log++);
        log--;
        for (int i = log ; i >= 0 ; i--) {
            if (pai[s][i] != pai[t][i]) {
                t = pai[t][i];
                s = pai[s][i];
            }
        }

        return pai[s][0];
    } else {
        return s;
    }
}
//lembrar de chamar antes das queries XD
void lca() {
    //pai[v][0] começa com o pai da árvore e o resto eh -1
    for (int x = 1 ; 1 << x < N ; x++) {
        for (int v = 0 ; v < N ; v++) {
            if (pai[v][x-1] != -1 && pai[pai[v][x-1]][x-1] != -1) {
                pai[v][x] = pai[pai[v][x-1]][x-1];
            }
        }
    }
}
```

Grafos: Max Flow $O(V^2E)$ (Dinic)

```
int last_edge[MAXV], cur_edge[MAXV], dist[MAXV];
int prev_edge[MAXE], cap[MAXE], flow[MAXE], adj[MAXE];
int nedges;

void d_init() {
    nedges = 0;
    memset(last_edge, -1, sizeof last_edge);
}
```

```
void d_edge(int v, int w, int capacity, bool r = false) {
    prev_edge[nedges] = last_edge[v];
    cap[nedges] = capacity;
    adj[nedges] = w;
    flow[nedges] = 0;
    last_edge[v] = nedges++;

    if(!r) d_edge(w, v, 0, true);
}

bool d_auxflow(int source, int sink) {
    queue<int> q;
    q.push(source);

    memset(dist, -1, sizeof dist);
    dist[source] = 0;
    memcpy(cur_edge, last_edge, sizeof last_edge);

    while(!q.empty()) {
        int v = q.front(); q.pop();
        for(int i = last_edge[v]; i != -1; i = prev_edge[i]) {
            if(cap[i] - flow[i] == 0) continue;

            if(dist[adj[i]] == -1) {
                dist[adj[i]] = dist[v] + 1;
                q.push(adj[i]);

                if(adj[i] == sink) return true;
            }
        }
    }

    return false;
}

inline int rev(int i) { return i ^ 1; }

int d_augmenting(int v, int sink, int c) {
    if(v == sink) return c;

    for(int& i = cur_edge[v]; i != -1; i = prev_edge[i]) {
        if(cap[i] - flow[i] == 0 || dist[adj[i]] != dist[v] + 1)
            continue;

        int val;
        if(val = d_augmenting(adj[i], sink, min(c, cap[i] - flow[i]))) {
            flow[i] += val;
            flow[rev(i)] -= val;
            return val;
        }
    }

    return 0;
}
int dinic(int source, int sink) {
    int ret = 0;
    while(d_auxflow(source, sink)) {
        int flow;
        while(flow = d_augmenting(source, sink, 0x3f3f3f3f))
            ret += flow;
    }

    return ret;
}
```

Geral: Big Int

```
#include <sstream>
const int DIG = 4;
const int BASE = 10000; // BASE**3 < 2**51
const int TAM = 2048;
struct bigint {
    int v[TAM], n;
    bigint(int x = 0): n(1) {
        memset(v, 0, sizeof(v));
        v[n++] = x; fix();
    }
    bigint(char *s): n(1) {
        memset(v, 0, sizeof(v));
        int sign = 1;
        while (*s && !isdigit(*s)) if (*s++ == '-') sign *= -1;
        char *t = strdup(s), *p = t + strlen(t);
        while (p > t) {
            *p = 0; p = max(t, p - DIG);
            sscanf(p, "%d", &v[n]);
            v[n++] *= sign;
        }
        free(t); fix();
    }
    bigint& fix(int m = 0) {
        n = max(m, n);
        int sign = 0;
        for (int i = 1, e = 0; i <= n || e && (n = i); i++) {
            v[i] += e; e = v[i] / BASE; v[i] %= BASE;
            if (v[i]) sign = (v[i] > 0) ? 1 : -1;
        }
        for (int i = n - 1; i > 0; i--)
            if (v[i] * sign < 0) { v[i] += sign * BASE; v[i+1] -= sign; }
        while (n && !v[n]) n--;
        return *this;
    }
    int cmp(const bigint& x = 0) const {
        int i = max(n, x.n), t = 0;
        while (1) if ((t = ::cmp(v[i], x.v[i])) || i-- == 0) return t;
    }
    bool operator <(const bigint& x) const { return cmp(x) < 0; }
    bool operator ==(const bigint& x) const {return cmp(x)== 0; }
    bool operator !=(const bigint& x) const { return cmp(x) != 0; }
    operator string() const {
        ostringstream s; s << v[n];
        for (int i = n - 1; i > 0; i--) {
            s.width(DIG); s.fill('0'); s << abs(v[i]);
        }
        return s.str();
    }
    friend ostream& operator <<(ostream& o, const bigint& x) {
        return o << (string) x;
    }
    bigint& operator +=(const bigint& x) {
        for (int i = 1; i <= x.n; i++) v[i] += x.v[i];
        return fix(x.n);
    }
    bigint operator +(const bigint& x) { return bigint(*this) += x; }
    bigint& operator -=(const bigint& x) {
        for (int i = 1; i <= x.n; i++) v[i] -= x.v[i];
        return fix(x.n);
    }
    bigint operator -(const bigint& x) { return bigint(*this) -= x; }
    bigint operator -() { bigint r = 0; return r -= *this; }
    void ams(const bigint& x, int m, int b) {//*this += (x*m) << b;
        for (int i = 1, e = 0; (i <= x.n || e) && (n = i + b); i++) {
            v[i+b] += x.v[i] * m + e;
            e = v[i+b] / BASE; v[i+b] %= BASE;
        }
    }
    bigint operator *(const bigint& x) const {
        bigint r;
        for (int i = 1; i <= n; i++) r.ams(x, v[i], i-1);
        return r;
    }
    bigint& operator *=(const bigint& x) {return *this =*this * x;}
    // cmp(x / y) == cmp(x) * cmp(y); cmp(x % y) == cmp(x);
    bigint div(const bigint& x) {
        if (x == 0) return 0;
        bigint q; q.n = max(n - x.n + 1, 0);
        int d = x.v[x.n] * BASE + x.v[x.n-1];
        for (int i = q.n; i > 0; i--) {
            int j = x.n + i - 1;
```

```
            q.v[i] = int((v[j] * double(BASE) + v[j-1]) / d);
            ams(x, -q.v[i], i-1);
            if (i == 1 || j == 1) break;
            v[j-1] += BASE * v[j]; v[j] = 0;
        }
        fix(x.n); return q.fix();
    }
    bigint& operator /=(const bigint& x) { return *this = div(x); }
    bigint& operator %=(const bigint& x) { div(x); return *this; }
    bigint operator /(const bigint& x) {return bigint(*this).div(x);}
    bigint operator %(const bigint& x) { return bigint(*this) %= x; }
    bigint pow(int x) {
        if (x < 0) return (*this == 1 || *this == -1) ? pow(-x) : 0;
        bigint r = 1;
        for (int i = 0; i < x; i++) r *= *this;
        return r;
    }
    bigint root(int x) {
        if (cmp() == 0 || cmp() < 0 && x % 2 == 0) return 0;
        if (*this == 1 || x == 1) return *this;
        if (cmp() < 0) return -(-*this).root(x);
        bigint a = 1, d = *this;
        while (d != 1) {
            bigint b = a + (d /= 2);
            if (cmp(b.pow(x)) >= 0) { d += 1; a = b; }
        }
        return a;
    }
};
```

Geral: Stable Marriage

```
bool ocupado(int f, int m) {
    if (garota[m] == -1) {
        return false;
    } else {
        return prefMasc[m][f] > prefMasc[m][garota[m]];
    }
}

void stableMarriage() {
    prox = 2;
    garotaAtual = 1;
    int m, f;
    while (garotaAtual <= N) { //garotaAtual
        //printf("antes while %d\n", garotaAtual);
        while (ocupado(garotaAtual, prefFem[garotaAtual][p[garotaAtual]])) {
            p[garotaAtual]++;
        }

        m = prefFem[garotaAtual][p[garotaAtual]];

        f = garotaAtual;
        if (garota[m] != -1) {
            garotaAtual = garota[m];
        } else {
            garotaAtual = prox++;
        }

        garota[m] = f;
    }
}
```

Geral: Simplex

```
const double EPS = 1e-9;
typedef long double T;
typedef vector<T> VT;
vector<VT> A;
VT b,c,res;
VI kt,N;
int m;
inline void pivot(int k,int l,int e){
   int x=kt[l]; T p=A[l][e];
   REP(i,k) A[l][i]/=p; b[l]/=p; N[e]=0;
   REP(i,m) if (i!=l) b[i]-=A[i][e]*b[l],A[i][x]=A[i][e]*-A[l][x];
   REP(j,k) if (N[j]){
      c[j]-=c[e]*A[l][j];
      REP(i,m) if (i!=l) A[i][j]-=A[i][e]*A[l][j];
   }
   kt[l]=e; N[x]=1; c[x]=c[e]*-A[l][x];
}

VT doit(int k){
   VT res; T best;
   while (1){
      int e=-1,l=-1; REP(i,k) if (N[i] && c[i]>EPS) {e=i; break;}
      if (e==-1) break;
      REP(i,m) if (A[i][e]>EPS && (l==-1 || best>b[i]/A[i][e]))
         best=b[ l=i ]/A[i][e];
      if (l==-1) /*ilimitado*/ return VT();
      pivot(k,l,e);
   }
   res.resize(k,0); REP(i,m) res[kt[i]]=b[i];
   return res;
}

VT simplex(vector<VT> &AA,VT &bb,VT &cc){
   int n=AA[0].size(),k;
   m=AA.size(); k=n+m+1; kt.resize(m); b=bb; c=cc; c.resize(n+m);
   A=AA; REP(i,m){ A[i].resize(k); A[i][n+i]=1; A[i][k-1]=-1; kt[i]=n+i;}
   N=VI(k,1); REP(i,m) N[kt[i]]=0;
   int pos=min_element(ALL(b))-b.begin();
   if (b[pos]<-EPS){
      c=VT(k,0); c[k-1]=-1; pivot(k,pos,k-1); res=doit(k);
      if (res[k-1]>EPS) /*impossivel*/ return VT();
      REP(i,m) if (kt[i]==k-1)
         REP(j,k-1) if (N[j] && (A[i][j]<-EPS || EPS<A[i][j])){
            pivot(k,i,j); break;
         }
      c=cc; c.resize(k,0); REP(i,m) REP(j,k) if (N[j]) c[j]-=c[kt[i]]*A[i][j];
   }
   res=doit(k-1); if (!res.empty()) res.resize(n);
   return res;
}
```