Introdução

Template

```
#include <bits/stdc++.h>
using namespace std;
#define DEBUG true
#define coutd if(DEBUG) cout
#define debugf if(DEBUG) printf
#define ff first
#define ss second
#define len(x) int(x.size())
#define all(x) x.begin(), x.end()
#define pb push_back
using ll = long long;
using ii = pair<int, int>;
using vi = vector<int>;
const double PI = acos(-1);
const double EPS = 1e-9;

int main(){
    ios_base::sync_with_stdio(0);


}
```

Matemática

Vetor de primos até N

```
const int MAX = 50000; // Números primos até MAX
vector<int> primes(){
    bitset<MAX> sieve;
    vector<int> ps;

    sieve.set();            // Todos são "potencialmente" primos
    sieve[1] = false;       // 1 não é primo

    for (int i = 2; i <= MAX; ++i){
        if (sieve[i]){
            // i é primo
            ps.push_back(i);

            for (int j = 2 * i; j <= N; j += i)
                sieve[j] = false;
        }
    }

    return ps;
}
```

Maior Divisor Comum

Dados dois inteiros a e b, o maior divisor comum (MDC) de a e b (notamos d = (a, b) é o inteiro não-negativo d tal que d divide a e d divide b; se c divide a e c divide b, então c divide d.

```
long long gdc(long long a, long long b){
    long long rest;
    do{
        rest = a%b;
        a=b;
        b=rest;
    }while(rest!=0);
    return a;
}
```

Complexidade O(log a +log b)

Menor Múltiplo Comum

Sejam a e b dois inteiros. O menor múltiplo comum (MMC) de a e b (notamos m = [a,b]) é o inteiro m tal que a divide m e b divide m;

```
long long lcm(long long a, long long b){
    return (a/gcd(a, b))*b;
}
```

Complexidade O(log a + log b)

Veja que, na implementação acima, a divisão é feita antes do produto: esta ordem pode evitar overflow em alguns casos.

Número de Divisores

A fatoração de um número n também permite computar o número de divisores deste número: basta fazer o produto de todos os expoentes da fatoração, somados cada um de uma unidade. Veja o código abaixo.

```
long long number_of_divisors(long long n)
{
    long long res = 0;

    for (long long i=1; i*i <= n; ++i)
    {
        if (n%i == 0)
            res += (i == n/i ? 1 : 2);
    }

    return res;
}
```

Complexidade O(sqrt n)

Estrutura de dados

Fenwick Tree [RSQ em log(n) soma pontual]

Resolve queries do tipo RSQ de 1 a n (1-indexed) em O(log n).

A arvore é contruida inicialmente zerada e para relizar a inserção dos itens deve ser feita a soma ao indice i do valor x

```
template <typename T>
class BITree {
private:
    vector<T> ts;
    size_t N;

    int LSB(int n) { return n&(-n); }

    T RSQ(int i){
        T sum = 0;

        while(i>=1){
            sum+=ts[i];
            i-=LSB(i);
        }
        return sum;
    }
public:
    BITree(size_t n) : ts(n+1, 0), N(n) {};

    T RSQ(int i, int j){
        return RSQ(j) - RSQ(i-1);
    }

    void add(size_t i, const T& x){
        if(i==0) return;
        while(i<=N){
            ts[i]+=x;
            i+=LSB(i);
        }
    }
};
```

Fenwick Tree [range update em log(n) point query]

```
template <typename T>
class BITree {
        private:
                vector<T> ts;
                size_t N;

                int LSB(int n) { return n&(-n); }

                void add(size_t i, ll x){
                        while (i <= N) {
                                ts[i] += x;
                                i += LSB(i);
                        }
                }
                T RSQ(int i){
                        T sum = 0;

                        while(i>=1){
                                sum+=ts[i];
                                i-=LSB(i);
                        }
                        return sum;
                }
        public:
                BITree(size_t n) : ts(n+1, 0), N(n) {};

                ll value_at(int i) { return RSQ(i); }

                void range_add(size_t i, size_t j, T x){
                        add(i, x);
                        add(j+1, -x);
                }
};
```

Fenwick Tree bidimensional

```
template<typename T>
class BITree2D{
private:
    size_t rows;
    size_t columns;
    vector<vector<T>> ft;

    int LSB(T n) { return n&(-n); }

    T RSQ(int y, int x){
        T sum = 0;

        for(int i=y; i>0; i-=LSB(i))
            for(int j=x; j>0; j-=LSB(j))
                sum+=ft[i][j];

        return sum;
    }
public:
    BITree2D(size_t y, size_t x) : ft(y+1, vector<T>(x+1, 0)), rows(y),
columns(x) {}

    // Y e X sao as coordenadas do ponto superior
    // x e y sao as coordenadas do ponto inferior
    // RSQ vai retornar a soma do quadrado formado pelos pontos
    T RSQ(int y, int x, int Y, int X){
        return RSQ(Y, X) - RSQ(Y, x-1) - RSQ(y-1, X) + RSQ(y-1, x-1);
    }

    void add(int y, int x, T v){
        for(int i=y; i<=rows; i+=LSB(i))
            for(int j=x; j<=columns; j+=LSB(j))
                ft[i][j] += v;
    }

};
```

Fenwick Tree - Multiplicação query product em O(log n) e point update

- Árvore faz a atualização de um índice do vetor multiplicando-o por uma constante k
- Toda a classe é feita em long long por se tratar de multiplicação entre inteiros

```
class BITree{
private:
    size_t N;
    vector<long long> ft;
    vector<int> sz; // 1 se o indice i for 0

    int LSB(const long long& n) { return n&(-n); }

    long long RPQ(int i){
        long long prod = 1;
        while(i){
            prod*=ft[i];
            i-=LSB(i);
        }
        return prod;
    }

    // Funções rsq para o vetor com zeros
    int RSQ(int i){
        int sum = 0;
        while(i){
            sum+=sz[i];
            i-=LSB(i);
        }
        return sum;
    }

    int RSQ(int i, int j){
        return RSQ(j)-RSQ(i-1);
    }

    void multiply(int i, long long k){
        while(i <= N){
            ft[i] *= k;
            i+=LSB(i);
        }
    }

    void add(int i, int k){
        while(i <= N){
            sz[i] += k;
            i+=LSB(i);
        }
    }
```

```
public:

    BITree(int n) : N(n), ft(n+1, 1), sz(n+1, 0) { }

    long long RPQ(int i, int j){
        long long p = RPQ(j)/RPQ(i-1);
        int z = RSQ(i, j);

        if(z) return 0;
        else return p;
    }

    void update(int i, const long long& k){
        if(k)
            multiply(i, k);
        else if (RSQ(i, i)==0)
            add(i, 1);
    }

};
```

Geometria Computacional

Ponto

Todas as notações ultilizam double, o que pode ser adaptado dependendo da questão.

Classe Ponto

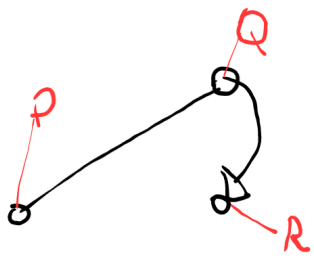
```
const double EPS = 1e-9;
struct Point{
    double x;
    double y;

    Point(double a, double b) : x(a), y(b) {} ;
    Point() : x(0), y(0) {} ;

    bool operator <(const Point& a){
        if (fabs(x-a.x)<EPS) // Se x==x
            return (y<a.y+EPS) and (y<a.y-EPS); //y<y
        else
            return (x<a.x+EPS) and (x<a.x-EPS); // x<x
    }
    bool operator ==(const Point& a){
        return fabs(x-a.x)<EPS && fabs(y-a.y)<EPS;
    }
    double distance(Point a){
        return hypot(x-a.x, y-a.y);
    }

};
```

Discriminante entre 3 pontos



- dis=0 caso R pertença ao plano
- dis=1 caso R esteja no semiplano à **esquerda** da reta
- dis=-1 caso R esteja no semiplano à **direita** da reta

```
int dis(Point p, Point q, Point r){
    double ans = (p.x*q.y+p.y*r.x+q.x*r.y)-(r.x*q.y+r.y*p.x+q.x*p.y);
    if(fabs(ans)<EPS) return 0;
    else if(ans>0) return 1;
    else return -1;
}
```

Vetores

Classe

A representação de vetores é igual a de ponto, porém com diferentes funções

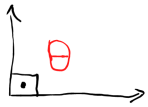
```
struct Vector{
    double x;
    double y;
    Vector(double a, double b) : x(a), y(b) {};
    Vector(): x(1), y(1) {};

    //Apenas se a classe ponto estiver emplementada
    // Vetor de A apontando para B
    Vector(Point a, Point b){
        x = b.x-a.x;
        y = b.y-a.y;
    };

    double size(){
        return hypot(x, y);
    }
    double ang(){
        return atan2(y, x);
    }
    void rotate(double theta){
        double r = size();
        double w = ang();
        x = r*cos(w-theta);
        y = r*sin(w-theta);
    }
    void rotate(Point c, double theta){
        x-=c.x; y-=c.y;
        rotate(theta);
        x+=c.x; y+=c.y;
    }
};
```

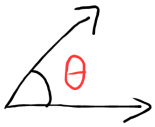
Produto interno entre 2 vetores

- produto interno=0 Vetores ortogonais



- produto interno=0 Ângulo obtuo

- produto interno=0 Ângulo agudo



```
double inter_prod(Vector u, Vector v){
    return u.x*v.x+u.y*v.y;
}
```

Angulo entre 2 vetores

```
const double EPS = 1e-9;
const double PI = acos(-1);

double ang(Vector u, Vector v){
    double p = inter_prod(u, v);
    if(fabs(p)<EPS) return PI;
    else return acos(p/(u.size()*v.size()));
}
```

Reta

Classe

A equação da reta mostrada abaixo se refere a ax+by+c=0

- Classe reta

```
struct Line{
    double a;
    double b;
    double c;

    Line() : a(1), b(1), c(1) {};
    //Apenas se a classe ponto estiver emplementada
    Line(Point p, Point q){
        a = p.y-q.y;
        b = q.x-p.x;
        c = p.x*p.y-q.x*p.y;
    }

    double fx(double x){
        return -(a*x+c)/b;
    }

    double fy(double y){
        return -(b*y*c)/a;
    }

};
```

Círculo

- Classe círculo

As funções definidas abaixo se referem ao argo, corda, setor e segmento do círculo respectivamente.

```
const double PI = acos(-1);

struct Circle{
    double r;
    double x;
    double y;

    double arc(double ang){
        return ang*r;
    }

    double chord(double ang){
        return 2*PI*sin(ang/2);
    }

    double sector(double ang){
        return ang*r*r/2;
    }

    double segment(double ang){
        double s = sector(ang);
        double c = chord(ang);
        return sqrt((s-r)*(s-r)*(s-c));
    }

};
```

Programação dinâmica

Knapsack

Problema clássico da mochila com pesos e valores dos itens, no código abaixo w é o peso que a pessoa pode carregar, wt é um array de pesos de tamanho n, val é um array de valores de tamanho n, e n é o tamanho dos arrays.

- A matriz do knapSack foi inicializada globalmente para poupar tempo de processamento inicializando os valores zerados
- Com esse código é necessário declarar os valores máximos que a matriz pode assumir como constantes

```
const int MAXN = 20000; // Número máximo de ítens
const int MAXW = 20000; // Número máximo de peso carregado
int mt[MAXN+1][MAXW+1]; // Matríz global é iniciada com zeros

int knapSack(int w, int wt[], int val[], int n){
    for(int i=1; i<=n; i++){
        for(int j=1; j<=w; j++){
            if(wt[i-1]<=j)
                mt[i][j] = max(val[i-1]+mt[i-1][j-wt[i-1]], mt[i-1][j]);
            else
                mt[i][j] = mt[i-1][j];
        }
    }
    return mt[n][w];
}
```