

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 gcd(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

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

A árvore é contruída inicialmente zerada e para relizar a inserção dos itens deve ser feita a soma ao índice 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 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;
    }
};

```

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

int knapsack(int w, int wt[], int val[], int n){
    for(int i=0; i<=n; i++){
        for(int j=0; j<=w; j++){
            if(i==0 || j==0)
                mt[i][j]=0;
            else 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];
}
```

Geometria Computacional

Ponto

Todas as notações de double podem ser adaptadas a depender do problema

- Classe Ponto

```
template<typename T>
struct Point{
    T x;
    T y;

    bool operator <(const Point& a){
        return x==a.x ? y<a.y : x<a.x;
    }
    bool operator ==(const Point& a){
        return x==a.x && y==a.y;
    }
    double distance(Point a){
        return hypot(x-a.x, y-a.y);
    }
};
```

Reta

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

- Classe reta

```
template<typename T>
struct Line{
    T a;
    T b;
    T c;

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

    T fy(T y){
        return -(b*y+c)/a;
    }
};
```

Criação de uma reta a partir de dois pontos

```
Line new_line(Point p, Point q){
    Line ans;
    ans.a = p.y-q.y;
    ans.b = q.x-p.x;
    ans.c = p.x*p.y-q.x*p.y;

    return ans;
};
```

Círculo

- Classe círculo

As funções definidas abaixo se referem ao arco, 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));
    }
};
```