# Guia Teórico - Programação Competitiva

***Equipe CalvO(cria) | CEFET-MG***

*Fabrício Augusto Lima Pereira, Luis Felipe Pessoa Lopes e Marcos Paulo Santos da Silva*

## Template

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define endl '\n'
#define s second
#define pb push_back;
#define endl "\n"
const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;

int main(){

return 0;
}
```

## Complexidade

| $N \leq$ | $\mathcal{O}$ "máximo" |
|---|---|
| 11 | $\mathcal{O}(n!)$ |
| 22 | $\mathcal{O}(2^n.n)$ |
| 100 | $\mathcal{O}(n^4)$ |
| 400 | $\mathcal{O}(n^3)$ |
| 2000 | $\mathcal{O}(n^2.\log_2(n))$ |
| $10^4$ | $\mathcal{O}(n^2)$ |
| $10^5$ | $\mathcal{O}(n.\log_2{}^2(n))$ |
| $10^6$ | $\mathcal{O}(n.\log_2(n))$ |
| $10^8$ | $\mathcal{O}(n)$ |
| $10^{18}$ | $\mathcal{O}(\log_2(n)), \mathcal{O}(1)$ |

# STL

## Vector

## next_permutation e prev_permutation

Para poder pegar a próxima permutação de um vetor.
Ex: {1,2,3} -> {1,3,2}. É possível combinar com um while do e coletar todas as possíveis permutações/anagramas (olhar tópico **Matemática - Combinatória - Permutação com repetição ou Anagrama**). Iniciar com sort.

```
next_permutation(vetor.begin(),vetor.end());
```

# Matemática

## Combinatória

### *Teorema fundamental da contagem*

Para saber todas as possíveis combinações de diversos itens.
Ex: Dispondo de 6 camisetas, 4 calças e 2 sapatos de quantas maneiras distintas é possível combinar tais roupas ?

R : quant. de camisas X quant. de calças X quant. de sapatos = total de possibilidades. 6 x 4 x 2 = 48

### *Fatorial ou Permutação Simples*

Ler um valor N. Calcular e escrever seu respectivo fatorial. Fatorial de N = N * (N-1) * (N-2) * (N-3) * ... * 1. Uma permutação simples se dá pelo fatorial da quantidade de caracteres SEM REPETIÇÃO.

```
int main(){
   int n,i,res;
   cin >> n;          //lê o valor de entrada
   if( n == 0){       // verifica se é 0 pois 0! = 1
       n = 1;
   }
   res = n;        //atribui o valor lido a resp
   for(i=n-1;i>=1;i--){  //mult. o valor a cada laço
      res *= i;
   }
   cout << res << endl;    //printa a resp. ex: 4! =
24
   return 0;
}
```

### *Soma de Fatoriais*

Código para verificar a soma de fatoriais para ter um determinado número. Ex: 8 = 3! + 2!. Codigo abaixo pra contar o num de fatoriais que é possível escrever.
Isso se dá pois 8 = 1! + 1! +1! + 1! + 1! + 1! + 1! + 1!
= 2! + 2! + 2! + 2!
= 3! + 2!

```
int main(){
   int n; cin >> n;    //numero lido
   vector<int> fat(11);    //vetor indo até 11!
   for(int i=1;i<=10;i++){  //add o valor ao vetor
       fat[i] = fat[i-1]*i;
   }
   int ans = 0;             //variável de resposta
   for(int i = 10; i>0; i--){  //guloso
       int at = n/fat[i];    //vê quantas vezes cabe
       ans += at;            //soma o num de vezes
```

```
        n -= at*fat[i];      //subtrai o num do n lido
    }
    cout << ans << endl;     //imprime a resp
    return 0;
}
```

### Permutação com repetição ou Anagrama

Ex: Quantos anagramas possíveis de se fazer com a palavra “programacao”?

R: $$P_n^{\alpha,\beta,\gamma} = \frac{n!}{\alpha!\beta!\gamma!}, \quad 11!/2!2!3! = 1163200$$

Código:

```
int main(){
    string s; getline(cin,s);
    int cont=0;
    sort(s.begin(),s.end());  //iniciar com sort
    do{
        cont++;
    }while (next_permutation(s.begin(),s.end()));
    cout<<cont<<endl;

    return 0;
}
```

### Arranjo

Nos arranjos, os agrupamentos dos elementos dependem da ordem e da natureza dos mesmos.

Para obter o arranjo simples de n elementos tomados, p a p (p ≤ n), utiliza-se a seguinte expressão:

$$A_{n,p} = \frac{n!}{(n-p)!}$$

```
// Calculo de fatorial com recursao
int fat(int n){
    if(n==0 || n==1) return 1;
    return n*fat(n-1);
}

int main(){
    int n; cin >> n;
    int p; cin >> p;
    int arranjo = fat(n)/fat(n-p);
    cout << arranjo << endl;
    return 0;
}
```

### Combinação

A combinação é definida como as diferentes formas de seleção de um grupo, tendo alguns ou todos os itens de um conjunto, sem que a ordem importe.

$$C_r^n = \binom{n}{r} = \frac{n!}{r! \cdot (n-r)!}$$

```
// Calculo de fatorial com recursao
int fat(int n){
    if(n==0 || n==1) return 1;
    return n*fat(n-1);
}

int main(){
    int n; cin >> n;
    int r; cin >> r;
    int arranjo = fat(n)/(fat(r)*fat(n-r));
    cout << arranjo << endl;
    return 0;
}
```

# Teoria dos Números

## MDC

O MDC é o maior número que divide dois ou mais números ao mesmo tempo. O máximo divisor comum, conhecido também pela sigla MDC, é o maior número que é divisor de dois ou mais números simultaneamente.

```
// Calcular mdc entre 2 numeros
int mdc(int num1, int num2)
{
    if(num1%num2 == 0)
        return num2;
    else
        return mdc(num2, num1%num2);
}
```

## Primo

Um número é classificado como primo se ele é maior do que um e é divisível apenas por um e por ele mesmo. Apenas números naturais são classificados como primos.

```
int main(){
    ll n; cin >> n;
    int primo = 1;
    for(int i=2; i*i <= n; i++){
        if(n % i == 0) primo = 0;
    }
    if(primo == 1) cout << "sim" << endl;
    else cout << "nao" << endl;

    return 0;
}
```

```
import java.util.*;
import java.math.BigInteger;

class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNext()) {
                        int N = sc.nextInt();
System.out.printf("%d is ", N);
            BigInteger BN = BigInteger.valueOf(N);
                                String R = new
StringBuffer(BN.toString()).reverse().toString();
            int RN = Integer.parseInt(R);
            BigInteger BRN = BigInteger.valueOf(RN);
                    if (!BN.isProbablePrime(10)) //
certainty 10 is enough
                System.out.println("not prime.");
                            else if ((N != RN) &&
BRN.isProbablePrime(10))
                System.out.println("emirp.");
            else
                System.out.println("prime.");
        }
    }
}
```

## Logarítmo

Calcular o log e poder selecionar a base e o número:

```
double log(double base, double x)
{
   return std::log(x) / std::log(base);
}
```

## MMC

Dados dois ou mais números, o MMC é o menor dos múltiplos que esses números possuem em comum. O mínimo múltiplo comum (MMC) entre números inteiros é o menor número, também inteiro, que é múltiplo de todos esses números ao mesmo tempo.

```
int mmc(int num1, int num2) {

    int mmc, aux, i;

    for (i = 1; i <= num2; i++) {
        aux = num1 * i;
        if ((aux % num2) == 0) {
            mmc = aux;
            i = num2 + 1;
        }
    }
    return mmc;
```

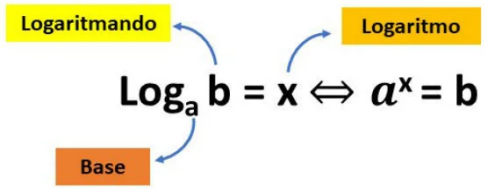
}

### Consequência da definição dos logaritmos

- O logaritmo de qualquer base, cujo logaritmando seja igual a 1, o resultado será igual a 0, ou seja, $\log_a 1 = 0$. Por exemplo, $\log_9 1 = 0$, pois $9^0 = 1$.
- Quando o logaritmando é igual a base, o logaritmo será igual a 1, assim, $\log_a a = 1$. Por exemplo, $\log_5 5 = 1$, pois $5^1 = 5$
- Quando o logaritmo de **a** na base **a** possui uma potência m, ele será igual ao expoente m, ou seja $\log_a a^m = m$, pois usando a definição $a^m = a^m$. Por exemplo, $\log_3 3^5 = 5$.
- Quando dois logaritmos com a mesma base são iguais, os logaritmandos também serão iguais,ou seja, $\log_a b = \log_a c \Leftrightarrow b = c$.
- A potência de base a e expoente $\log_a b$ será igual a b, ou seja $a^{\log_a b} = b$.

### Propriedades dos Logaritmos

- **Logaritmo de um produto**: O logaritmo de um produto é igual a soma de seus logaritmos: $\text{Log}_a (b.c) = \text{Log}_a b + \log_a c$
- **Logaritmo de um quociente**: O logaritmo de um quociente é igual a diferença dos logaritmos: $\text{Log}_a \left(\frac{b}{c}\right)$ $= \text{Log}_a b - \text{Log}_a c$
- **Logaritmo de uma potência**: O logaritmo de uma potência é igual ao produto dessa potência pelo logaritmo: $\text{Log}_a b^m = m \cdot \text{Log}_a b$
- **Mudança de base:** Podemos mudar a base de um logaritmo usando a seguinte relação: $\log_b c = \frac{\log_a c}{\log_a b}$




## Possíveis erros

**Compilation Error:** O programa não compilou. Deve-se verificar se a linguagem de programação foi escolhida corretamente e verificar possíveis erros

**Runtime Error:** O programa compilou corretamente, entretanto teve um erro em tempo de execução. Deve-se verificar a solução na busca de possíveis erros de programação que dão origem a acesso inválidos de memória, estouros de pilha, falhas de segmentação,etc.

**Wrong Answer:** O programa compilou corretamente, entretanto não forneceu a saída correta para todos os casos. Deve-se verificar se o algoritmo proposto funciona para todos os casos, revisar o problema na busca de possíveis condições não detectadas anteriormente.

**Presentation Error:** O programa compilou e executou sem erros, além disso forneceu a resposta correta, entretanto possui algum erro de formatação da saída. Deve-se verificar se a saída está no formato especificado pela questão.

**Time Limit Exceeded:** O programa demorou mais para terminar do que o permitido pelos juízes. Deve-se verificar a possibilidade do programa não estar parando em determinado caso. Deve-se verificar a complexidade do algoritmo e verificar se o algoritmo proposto é rápido o suficiente para o problema proposto

**Accepted:** Rodou subiu.