



Kaio Christaldo
Fabricio Matsunaga

Algorithm <bigInt>

- BigInt é uma estrutura ou técnica usada para representar e manipular números inteiros que ultrapassam os limites dos tipos de dados nativos das linguagens de programação, como int ou long long. Ela permite trabalhar com números arbitrariamente grandes, que não cabem em variáveis convencionais.
- O conceito de BigInt surgiu da necessidade de cálculos precisos em áreas como criptografia, computação científica e algoritmos que envolvem números muito grandes, como fatoriais, combinações e problemas de programação competitiva.

Algorithm <bigInt>

- BigInt pode ser implementado como uma classe ou estrutura que armazena o número em partes (por exemplo, usando vetores ou strings) e implementa operações aritméticas básicas (adição, subtração, multiplicação, divisão) manualmente, dígito a dígito.
- Apesar de ser mais lento que os tipos primitivos, BigInt é fundamental quando é necessário ultrapassar o limite fixo de bits e garantir precisão total em números grandes.
- Vamos implementar: **soma, subtração, divisão, multiplicação, módulo e comparações**

Algorithm <bigInt>

Base da Implementação

No C++ é possível implementar com structs (recomendável) ou como classes (mais complexo)



```
1 #include <string>
2 #include <vector>
```



```
1 struct BigInt {
2     string number;
3
4     BigInt(string v) : number(v){ remove_zeros(); }
5     BigInt() : number("0"){ }
6
7     BigInt operator+(const BigInt& other) const;
8     BigInt operator/(const BigInt& other) const;
9     BigInt operator-(const BigInt &b) const;
10    BigInt operator*(int n) const;
11    BigInt operator*(const BigInt& outro) const;
12
13    bool operator<=(const BigInt &b) const;
14    bool operator>=(const BigInt &b) const;
15    bool operator<(const BigInt &b) const;
16    bool operator>(const BigInt &b) const;
17    bool operator==(const BigInt &b) const;
18    bool operator!=(const BigInt &b) const;
19
20    void remove_zeros();
21
22    friend ostream& operator << (ostream& os, const BigInt& b);
23 };
```

◀Soma▶

Algorithm <sum>

Como funciona?

A operação de soma acontece como em uma operação de soma de inteiros

A hand-drawn addition problem on a white background. The numbers 8728 and 5537 are stacked vertically, with a blue plus sign to the left of the second number. A horizontal blue line is drawn below the second number. The sum 14265 is written below the line. A vertical orange bar highlights the second column from the right (the tens column), which contains the digits 8, 5, and 4. Above the 8 and 5, and above the 4, is the digit 1, representing the carry. The digits are: 8 7 2 8 (top row), 5 5 3 7 (middle row), and 1 4 2 6 5 (bottom row).

```
1  BigInt BigInt::operator+(const BigInt& other) const {
2
3      string n1 = this->number;
4      string n2 = other.number;
5
6      string res = "";
7
8      int carry = 0;
9
10     if (n2.size() > n1.size())
11         swap(n1, n2);
12
13     int n = n1.size();
14
15     while ((int)n2.size() < n)
16         n2 = "0" + n2;
17
18     for (int i = n - 1; i >= 0; i--) {
19         int soma = (n1[i] - '0') + (n2[i] - '0') + carry;
20         carry = soma / 10;
21         res = char((soma % 10) + '0') + res;
22     }
23
24     if (carry) res = '1' + res;
25
26     return BigInt(res);
27 };
28
```

◀ **Divisão** ▶

Algorithm <div>

Como funciona?

É uma divisão longa (igual à que fazemos no papel, algarismo por algarismo).

- **Laço principal**

- Para cada dígito do dividendo (`this->number`):
- Constrói o número atual adicionando um dígito.
- Encontra o maior x tal que $\text{divisor} * x \leq \text{atual}$.
- Adiciona x ao resultado.
- Atualiza $\text{atual} = \text{atual} - \text{divisor} * x$ (o "resto" até agora).

Algorithm <div>

Implementação

A implementação é simples, no entanto é necessário também criar alguns operadores como requisitos:

- Subtração, Multiplicação, Comparação Menor Igual (\leq)
- Função para remover zeros a esquerda



```
1 BigInt operator/(const BigInt& other) const;
2 BigInt operator-(const BigInt &b) const;
3 BigInt operator*(int n) const;
4 BigInt operator*(const BigInt& outro) const;
5 bool operator<=(const BigInt &b) const;
6 void remove_zeros();
```



```
1 BigInt BigInt::operator/(const BigInt &divisor) const {
2     BigInt atual, resultado;
3     for (char digito : this->number) {
4         atual.number += digito;
5         atual.remove_zeros();
6
7         int x = 0;
8         while (divisor * (x + 1) <= atual)
9             x++;
10
11         resultado.number += (x + '0');
12         atual = atual - (divisor * x);
13     }
14
15     resultado.remove_zeros();
16     return resultado;
17 }
```

«Comparações»

Algorithm <cond>

As condicionais são interessantes e necessárias quando precisamos de verificar igualdade ou maior e menor destes interios gigantes.



```
1  bool operator<=(const BigInt &b) const;  
2  bool operator>=(const BigInt &b) const;  
3  bool operator<(const BigInt &b) const;  
4  bool operator>(const BigInt &b) const;  
5  bool operator==(const BigInt &b) const;  
6  bool operator!=(const BigInt &b) const;
```

Algorithm <cond>

Implementação

Todos os operadores são implementados apartir do operador **menor**(<) e **igual**(==) usando a ideia da tabelinha.

```
1 bool BigInt::operator<(const BigInt &b) const {
2     if (number.size() < b.number.size()) return true;
3     if (number.size() > b.number.size()) return false;
4     return number < b.number;
5 }
```

Operador desejado	Expressão equivalente usando < e ==
a < b	a < b
a <= b	!(a < b)
a > b	b < a
a >= b	!(a < b)
a == b	a == b
a != b	!(a == b)

Apresentação Problema Motivador

beecrowd | 1161



Soma de Fatoriais

Adaptado por Neilor Tonin, URI  Brasil

Timelimit: 1

Leia dois valores inteiros M e N indefinidamente. A cada leitura, calcule e escreva a soma dos fatoriais de cada um dos valores lidos. Utilize uma variável apropriada, pois cálculo pode resultar em um valor com mais de 15 dígitos.

Entrada

O arquivo de entrada contém vários casos de teste. Cada caso contém dois números inteiros M ($0 \leq M \leq 20$) e N ($0 \leq N \leq 20$). O fim da entrada é determinado por eof.

Saída

Para cada caso de teste de entrada, seu programa deve imprimir uma única linha, contendo um número que é a soma de ambos os fatoriais (de M e N).

Exemplo de Entrada

```
4 4
0 0
0 2
```

Exemplo de Saída

```
48
2
3
```

**1161 – Soma
de Fatoriais**

Apresentação Problema Motivador

beecrowd | 1161



Soma de Fatoriais

Adaptado por Neilor Tonin, URI  Brasil

Timelimit: 1

Leia dois valores inteiros M e N indefinidamente. A cada leitura, calcule e escreva a soma dos fatoriais de cada um dos valores lidos. Utilize uma variável apropriada, pois cálculo pode resultar em um valor com mais de 15 dígitos.

Entrada

O arquivo de entrada contém vários casos de teste. Cada caso contém dois números inteiros M ($0 \leq M \leq 20$) e N ($0 \leq N \leq 20$). O fim da entrada é determinado por eof.

Saída

Para cada caso de teste de entrada, seu programa deve imprimir uma única linha, contendo um número que é a soma de ambos os fatoriais (de M e N).

Exemplo de Entrada

```
4 4
0 0
0 2
```

Exemplo de Saída

```
48
2
3
```

**1161 – Soma
de Fatoriais**

Resolução do Problema Motivador

1161 – Soma de Fatoriais

Desafio:

- Implementar solução para somar fatoriais acima de 21!

A resolução estará disponível no Drive. Tente resolver por conta própria e, se precisar, compare com a solução! 😊

◀Subtração▶

◀BigInt▶

- Estrutura para representar números inteiros muito grandes
- Usada quando long long (64 bits) não é suficiente
- Importante para aplicações como criptografia, astronomia, e competições de programação
- C++ não possui suporte nativo — é preciso implementar manualmente

«BigInt»

Representamos o número como uma string ou vetor de dígitos



```
1 string digitos; // Armazenado invertido: "123" representa o número 321
2 bool negative; // Sinal
```

Por que invertido? Facilita operações de soma e subtração

◀BigInt – Subtração▶ Ideia Geral

- Igual à subtração manual (de trás para frente)
- Controla o empréstimo quando dígito de cima é menor
- Verifica qual número é maior para manter sinal correto

«BigInt – Subtração» Trecho de Código



```
1  BigInt &operator-=(BigInt&a,const BigInt &b){
2      if(a < b)
3          throw("UNDERFLOW");
4      int n = Length(a), m = Length(b);
5      int i, t = 0, s;
6      for (i = 0; i < n;i++){
7          if(i < m)
8              s = a.digits[i] - b.digits[i]+ t;
9          else
10             s = a.digits[i]+ t;
11             if(s < 0)
12                 s += 10,
13                 t = -1;
14             else
15                 t = 0;
16             a.digits[i] = s;
17         }
18         while(n > 1 && a.digits[n - 1] == 0)
19             a.digits.pop_back(),
20             n--;
21         return a;
22     }
```



```
1  BigInt operator-(const BigInt& a,const BigInt&b){
2      BigInt temp;
3      temp = a;
4      temp -= b;
5      return temp;
6  }
```

◀ **Multiplicação** ▶

◀BigInt – Multiplicação▶ Ideia Geral

- Usa a técnica da "armação" (multiplicação manual)
- Multiplica cada dígito de A por cada dígito de B
- Soma os resultados parciais deslocados

«BigInt – Multiplicação» Trecho de Código

```
1  BigInt &operator*=(BigInt &a, const BigInt &b)
2  {
3      if(Null(a) || Null(b)){
4          a = BigInt();
5          return a;
6      }
7      int n = a.digits.size(), m = b.digits.size();
8      vector<int> v(n + m, 0);
9      for (int i = 0; i < n; i++)
10         for (int j = 0; j < m; j++){
11             v[i + j] += (a.digits[i] ) * (b.digits[j]);
12         }
13     n += m;
14     a.digits.resize(v.size());
15     for (int s, i = 0, t = 0; i < n; i++)
16     {
17         s = t + v[i];
18         v[i] = s % 10;
19         t = s / 10;
20         a.digits[i] = v[i] ;
21     }
22     for (int i = n - 1; i >= 1 && !v[i]; i--)
23         a.digits.pop_back();
24     return a;
25 }
```

```
1  BigInt operator*(const BigInt&a, const BigInt&b){
2      BigInt temp;
3      temp = a;
4      temp *= b;
5      return temp;
6  }
```

◀ **Comparações** ▶

◀BigInt – Comparações▶ Ideia Geral

- **principais comparações:**
 - **Igualdade (==)**
 - **Maior (>)**
 - **Maior igual (>=)**
 - **Menor (<)**
 - **Menor igual (<=)**
- **Leva em conta:**
 - **Sinal**
 - **Tamanho**
 - **Algarismos de maior peso**

◀ BigInt – Comparações ▶

Trecho de Código



```
1  bool operator==(const BigInt &a, const BigInt &b){  
2      return a.digits == b.digits;  
3  }  
4  
5  bool operator!=(const BigInt &a, const BigInt &b){  
6      return !(a == b);  
7  }
```

«BigInt – Comparações»

Trecho de Código



```
1  bool operator<(const BigInt&a,const BigInt&b){
2      int n = Length(a), m = Length(b);
3      if(n != m)
4          return n < m;
5      while(n--)
6          if(a.digits[n] != b.digits[n])
7              return a.digits[n] < b.digits[n];
8      return false;
9  }
10
11 bool operator>(const BigInt&a,const BigInt&b){
12     return b < a;
13 }
14 bool operator>=(const BigInt&a,const BigInt&b){
15     return !(a < b);
16 }
17 bool operator<=(const BigInt&a,const BigInt&b){
18     return !(a > b);
19 }
20
```

Resolução do Problema Motivador

A resolução estará disponível no Drive. Tente resolver por conta própria e, se precisar, compare com a solução! 😊

Lista de Exercícios

1237 – Comparação de Substring

2087 – Conjuntos Bons e Ruins



Se tiver alguma dúvida ou dificuldade na resolução de algum exercício, sinta-se à vontade para perguntar! 😊

Referências

[1] GEEKSFORGEEKS. Introduction to Trie - Data Structure and Algorithm Tutorials. GeeksforGeeks, [s. l.], [s. d.]. Disponível em: <https://www.geeksforgeeks.org/dsa/introduction-to-trie-data-structure-and-algorithm-tutorials/>. Acesso em: 13 jun. 2025.

[2] SPOJ. ADAINDEX - Ada and Indexing. Sphere Online Judge, [s. l.], [s. d.]. Disponível em: <https://www.spoj.com/problems/ADAINDEX/>. Acesso em: 13 jun. 2025.