# Matemática

Representação Binária

Prof. Edson Alves
Faculdade UnB Gama

#### Representação em base decimal

ullet A representação de número n, em base decimal, consiste na concatenação de k+1 coeficientes  $c_i$  tais que

$$n = c_0 + c_1 \cdot 10 + c_2 \cdot 10^2 + \ldots + c_k \cdot 10^k$$

Por exemplo,

$$2507 = 7 + 0 \cdot 10 + 5 \cdot 10^2 + 2 \cdot 10^3$$

#### Representação em uma base arbitrária

ullet De forma geral, a representação de n em base b>1 é a concatenação de k+1 coeficientes  $a_j$  tais que

$$n=a_0+a_1\cdot b+a_2\cdot b^2+\ldots+a_k\cdot b^k$$

- A representação de qualquer inteiro n em base b é única
- ullet Esta representação R de n em base b pode ser obtida usando-se recursão e o algoritmo de Euclides: R(n)=R(q)b+r, onde  $n=bq+r, 0\leq r < b$

### Representação em base arbitrária

```
const string digits { "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ" };
string representation(int n, int b)
    string rep;
    do {
        rep.push_back(digits[n % b]);
        n /= b;
    } while (n);
    reverse(rep.begin(), rep.end());
    return rep;
```

#### Conversão entre bases

- A conversão de uma representação em base a para uma base b é, em geral, feita em duas etapas:
  - 1. conversão da base a para uma base pré-determinada (base 10 ou 2, por exemplo);
  - $\overline{\ 2.}$  conversão desta base pré-determinada para a base b.
- ullet A primeira etapa é realizada por meio da expansão da representação do número em base a
- ullet Esta expansão pode ser realizada em O(k) por meio do algoritmo de Horner
- A segunda é feita por meio da rotina de geração de representação já mencionada

## Conversão para base decimal

```
long long to_decimal(const string& rep, long long base)
{
    long long n = 0;
    for (auto c : rep)
    {
        n *= base;
        n += digits.find(c);
    }
    return n;
}
```

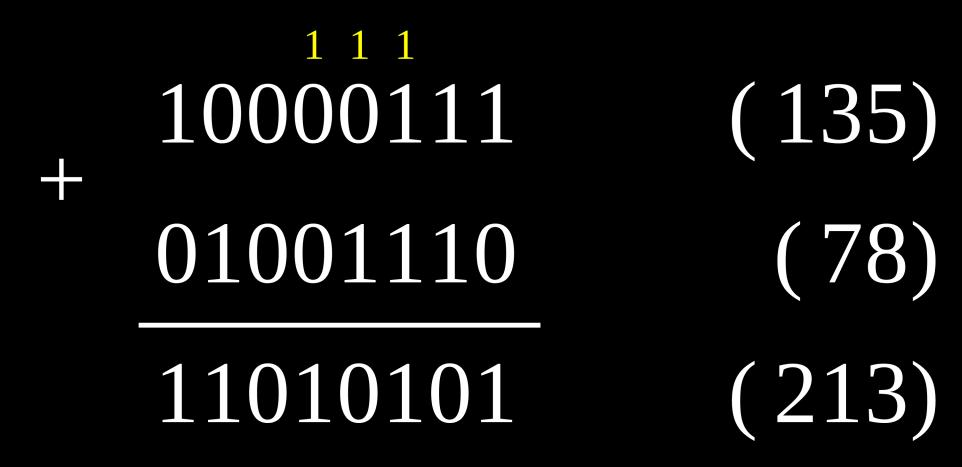
#### Representação em base binária

- ullet A base b=2 é a menor e mais simples dentre todas as bases positivas
- ullet Os únicos dois dígitos possíveis em R(n) são 0 e 1
- Internamente, os computadores armazenam números inteiros em sua representação binária
- É possível comparar diretamente dois números em base binária, sem a necessidade de convertê-los para a base decimal

#### Representação em base binária

- Para isso, uma vez alinhados o número de dígitos (com zeros à esquerda, se necessário), vale a comparação lexicográfica
- Do mesmo modo, é possível somar diretamente dois números em base binária
- Uma vez alinhados, a soma de dígitos distintos resulta em 1; a soma de dois zeros é 0; a soma de dois uns resulta em 0 e um novo 1 é adicionado à próxima posição (vai um, carry)

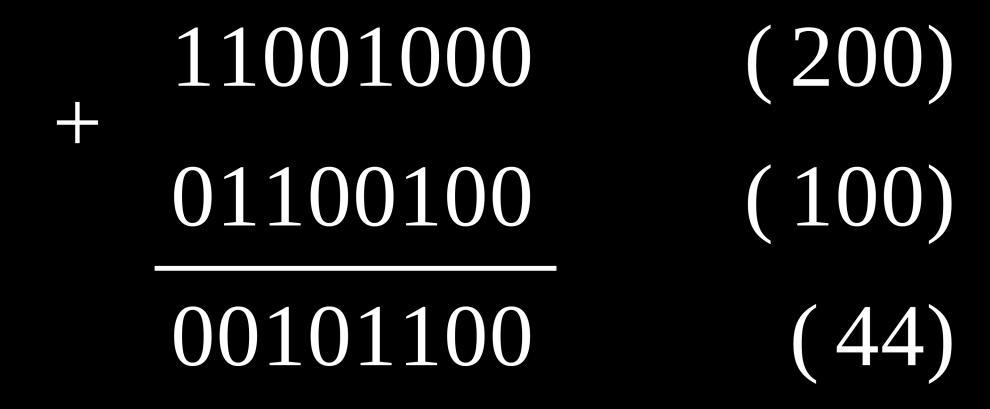
### Visualização da soma em base binária



#### Overflow

- Nas linguagens de programação, o número de bits usados na representação de inteiros é limitado
- Por exemplo, em C/C++, variáveis do tipo int ocupam, em geral, 32 bits (o mesmo espaço em memória que uma palavra do processador)
- Variáveis long long, em geral, ocupam 64 bits
- Esta limitação de espaço pode levar ao *overflow*: quando o limite é atingido, os *bits* que excedem o tamanho máximo "transbordam", ficando apenas aqueles que se encontram dentro do limite de espaço
- O overflow pode levar a resultados inesperados, e deve ser tratado com cuidado e atenção

## Visualização do *overflow* em variáveis de 8 *bits*



### Representação binária de números negativos

- ullet Para representar número negativos, utiliza-se o fato de que n+(-n)=0
- ullet Assim, a representação de -n seria um número tal que, somado com n, daria resto zero
- ullet Devido ao  $\emph{overflow}$ , tal número existe e é denominado complemento de dois de n
- ullet Por exemplo, em variáveis de 8  $\it bits$  de tamanho, o complemento de dois de 77 é 179, pois 77+179=256=0

### Representação binária de números negativos

- O complemento de dois pode ser obtido diretamente, sem necessidade de uma subtração
- ullet Basta inverter os *bits* da representação binária de n e somar um ao resultado
- Desta maneira, o bit mais significativo diferencia os números positivos (zero) dos negativos (um)

## Visualização do complemento de dois de $77\,$

$$\sim 01001101$$
 (77)  
 $10110010$  (178)  
 $+$  10110011 (-77)

#### **Problemas**

- AtCoder
  - 1. ABC 044D Digit Sum
- Codeforces
  - 1. 258A Little Elephant and Bits
  - 2. <u>1338B Captain Flint and a Long Voyage</u>
- OJ
  - 1. 343 What Base is This?
  - 2. 355 The Bases are Loaded
  - 3. <u>11185 Ternary</u>

#### Referências

- 1. **HALIM**, Felix; **HALIM**, Steve. *Competitive Programming 3*, 2010.
- 2. LAAKSONEN, Antti. Competitive Programmer's Handbook, 2018.
- 3. SKIENA, Steven S.; REVILLA, Miguel A. Programming Challenges, 2003.