

# Manipulação de bits

Representação binária

---

Prof. Edson Alves - UnB/FGA

2018



# Representação binária

---

# Representação em base arbitrária

- A representação de número  $n$ , em base decimal, consiste na concatenação dos coeficientes  $c_i$  tal que  $n = \sum_i c_i 10^i$
- Em particular, temos

$$2507 = 2 \cdot 10^3 + 5 \cdot 10^2 + 0 \cdot 10^1 + 7 \cdot 10^0$$

- De forma geral, a representação de  $n$  em base  $b > 1$  é a concatenação dos coeficientes  $a_j$  tal que  $n = \sum_j a_j b^j$
- A representação em base  $b$  é única
- 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  $n = bq + r, 0 \leq r < b$

# Implementação da rotina que computa $R(n)$

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const string digits { "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ" };
6
7 string representation(int n, int b)
8 {
9     string rep;
10
11     do {
12         rep.push_back(digits[n % b]);
13         n /= b;
14     } while (n);
15
16     reverse(rep.begin(), rep.end());
17
18     return rep;
19 }
20
```

# Implementação da rotina que computa $R(n)$

```
21 int main()
22 {
23     int n, base;
24
25     // 2 <= base <= 36
26     cin >> n >> base;
27
28     cout << representation(n, base) << '\n';
29
30     return 0;
31 }
```

# Representação em base binária

- A base  $b = 2$  é a menor e mais simples dentre todas as bases positivas
- 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: uma vez alinhados o número de dígitos (com zero à esquerda, se necessário), a comparação é a mesma da comparação lexicográfica de strings
- 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

$$\begin{array}{r} \phantom{+} \phantom{0000} \overset{1}{\phantom{0}} \overset{1}{\phantom{0}} \overset{1}{\phantom{0}} \\ 10000111 \quad (135) \\ + 01001110 \quad (78) \\ \hline 11010101 \quad (213) \end{array}$$



# 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* (variáveis **long long** ocupam 64 *bits*)
- Em geral, uma variável do tipo **int** ocupam o mesmo espaço em memória que uma palavra do processador
- Esta limitação de espaço pode levar ao *overflow*: quando o limite é atingido, os *bits* que excedem o tamanho máximo “transbordam”, ficando apenas aqueles dentro do limite
- O *overflow* pode levar a resultados inesperados, e deve ser tratado com cuidado e atenção

## Visualização do overflow em variáveis de 8 bits

$$\begin{array}{rcl} & 11001000 & (200) \\ + & 01100100 & (100) \\ \hline & 00101100 & (44) \end{array}$$

## Representação binária de números negativos

- Para representar número negativos, utiliza-se o fato de que  $n + (-n) = 0$
- Assim, a representação de  $(-n)$  seria um número tal que, somado com  $n$ , daria resto zero
- Devido ao *overflow*, tal número existe e é denominado complemento de dois de  $n$
- Por exemplo, em variáveis de 8 *bits* de tamanho, o complemento de dois de 77 é 179, pois  $77 + 179 = 256 = 0$
- O complemento de dois pode ser obtido diretamente, sem necessidade de uma subtração: 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

$$\begin{array}{r} \sim 01001101 \quad (77) \\ \hline 10110010 \quad (178) \\ + \\ \quad \quad \quad 1 \quad (1) \\ \hline 10110011 \quad (-77) \end{array}$$

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.