Matemática

Bases numéricas: representação binária

Prof. Edson Alves

Faculdade UnB Gama

Representação binária

Representação em base decimal

Definição

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

- 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
- Esta representação $R(n)$ 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 representação em base arbitrária em C++

```cpp
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);
  2. conversão desta base pré-determinada para a base b.
- A primeira etapa é realizada por meio da expansão da representação do número em base a
- Esta expansão pode ser realizada em $O(k)$ por meio do algoritmo de Horner
- A segunda é feita por meio da rotina que obtém a representação

Conversão para base decimal

```cpp
const string digits { "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ" };

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

- 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
- 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

$$
\begin{array}{rrr}
 & {\color{blue}1\,1\,1} & \\
 & \texttt{10000111} & (135) \\
+ & \texttt{01001110} & (78) \\
\hline
 & \texttt{11010101} & (213)
\end{array}
$$

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* (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*

```
   11001000      (200)
+
   01100100      (100)
  ----------
   00101100       (44)
```

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

```
~ 01001101        (77)
 -----------
   10110010      (178)
+
          1        (1)
 ___________
   10110011      (-77)
```

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.