Aritmética e Álgebra

Prof. Rafael Alves Bonfim de Queiroz
rafael.queiroz@ufop.edu.br

Conteúdo

- A ligação entre habilidades de programação e matemática é bem estabelecida
- Os primeiros computadores foram construídos por matemáticos para acelerar cálculos
- Pascal construiu uma máquina de somar baseada em engrenagens mecânicas em 1645
- Cientistas da computação pioneiros como Turing e von Neumann tiveram realizações iguais ou até maiores em matemática pura

Aritmética de Máquina

- Toda linguagem de programação inclui um tipo de dado inteiro que suporta as quatro operações aritméticas básicas: adição, subtração, multiplicação e divisão
- Essas operações geralmente são mapeadas quase diretamente para instruções aritméticas no nível de hardware e, portanto, a faixa de tamanho de inteiros depende do processador subjacente
- Computadores de 32 bits suporta inteiros aproximadamente na faixa $\pm 2^{31}$
- A maioria das linguagens de programação suporta tipos de dados inteiros *long* ou mesmo *long long*, que geralmente definem inteiros de 64 bits ou até de 128 bits

- Inteiros convencionais de 32 bits são normalmente representados usando quatro bytes contíguos, com *longs* de 64 bits sendo matrizes de oito bytes
- Imagens digitais são frequentemente representadas como matrizes de cores de byte único (ou seja, 256 níveis de cinza) para eficiência de espaço
- Inteiros positivos são representados como números binários positivos
- Números negativos geralmente usam uma representação mais sofisticada, como complemento de dois

Bibliotecas de Inteiros

- Biblioteca C/C++ *stdlib.h* inclui cálculo de valor absoluto e números aleatórios, enquanto *math.h* inclui tetos, pisos, raízes quadradas e exponenciais
- As classes inteiras para C++ e Java são ainda mais poderosas
  - A classe GNU g++ *Integer* e a classe *java.math Big Integer* fornecem suporte para inteiros de alta precisão

Inteiros de alta precisão

- Representar números inteiros realmente enormes requer encadear dígitos juntos
  - *Matrizes de Dígitos*
    - Elemento inicial do vetor representa o dígito menos significativo
    - Manter um contador com o comprimento do número em dígitos pode ajudar na eficiência minimizando as operações que não afetam o resultado
  - *Listas Encadeadas de Dígitos*
    - Se não houver limite superior no comprimento dos números

Tipo de dados: Bignum

```
#define MAXDIGITS       100             /* maximum length bignum */

#define PLUS            1               /* positive sign bit */
#define MINUS           -1              /* negative sign bit */

typedef struct {
        char digits[MAXDIGITS];         /* represent the number */
        int signbit;                    /* PLUS or MINUS */
        int lastdigit;                  /* index of high-order digit */
} bignum;
```

- Observe que cada dígito (0-9) é representado usando um caractere de byte único

Bases Numéricas e Conversão

- A representação de dígitos de um dado número-raiz é uma função de qual base numérica é usada
  - **Binário** – (Base 2) números são compostos pelos dígitos 0 e 1
    - Eles fornecem o representação inteira usada em computadores, porque esses dígitos mapeiam naturalmente para estados ligado/desligado ou alto/baixo
  - **Octal** – (Base 8) úteis como abreviação para facilitar a leitura de números binários, uma vez que os bits podem ser lidos à direita em grupos de três
    - Exemplo: $10111001_2 = 371_8 = 249_{10}$

Bases Numéricas e Conversão

- **Decimal** - Usamos números de base 10 porque aprendemos a contar até dez nos dedos
  - Os antigos maias usavam um sistema numérico de base 20, presumivelmente porque contavam nos dedos das mãos e dos pés
- **Hexadecimal** - Números de base 16 - dígitos 0, 1, 2, $\cdots$, 9, "A" a "F"

Números Reais

- Aritémtica de ponto flutuante tem precisão limitada
- Grande parte da matemática depende da *continuidade* dos reais, o fato de que sempre existe um número c entre a e b se $a < b$
  - Isso não é verdade em números reais como eles são representados em um computador
- Muitos algoritmos dependem de uma suposição de computação *exata*
  - Isso não é verdade para números reais como eles são representados em um computador
- A associatividade da adição garante que $(a + b) + c = a + (b + c)$
  - Infelizmente, isso não é necessariamente verdade em aritmética computacional devido a erros de arredondamento

Tipos de números: Inteiros

- Estes são os números de contagem, $-\infty$, $\cdots$, -2, -1, 0, 1, 2, $\cdots$, ∞
- Os subconjuntos dos inteiros incluem os números naturais (inteiros a partir de 0) e os inteiros positivos (aqueles que começam em 1)
- Um aspecto limitante dos inteiros é que existem lacunas entre eles

Tipos de números: Números racionais e irracionais

- **Números Racionais**
  - Estes são os números que podem ser expressos como a razão de dois inteiros, ou seja, c é racional se $c = a/b$ para inteiros a e b
  - Todo inteiro pode ser representado por um racional, a saber, $c/1$.
  - Os números racionais são sinônimos de frações, desde que incluamos as frações impróprias a/b onde $a > b$.
  - Há sempre um número racional entre quaisquer dois racionais x e y ($(x + y)/2$ é um bom exemplo)
- **Números Irracionais**
  - Existem muitos números interessantes que não são números racionais
  - Exemplos incluem $\pi = 3{,}1415926\cdots$, $2 = 1{,}41421\cdots$ e $e = 2{,}71828\cdots$

Lidando com Números Reais

- A representação interna de números de ponto flutuante varia de computador para computador, linguagem para linguagem e compilador para compilador
- Existe um padrão IEEE para aritmética de ponto flutuante que um número crescente dos fornecedores aderem, mas você deve sempre esperar problemas em cálculos que exigem precisão muito alta
- Números de ponto flutuante são representados em notação científica, ou seja, $a \times 2^c$, com um número limitado de bits atribuídos à mantissa a e ao expoente c
- Operar com dois números com expoentes muito diferentes geralmente resulta em estouro ou erros de **underflow**, pois a mantissa não possui bits suficientes para armazenar a resposta
- Tais problemas são a fonte de muitas dificuldades com erros de arredondamento

- O problema mais importante ocorre no teste de igualdade de números reais, já que geralmente há lixo nos bits de baixa ordem da mantissa para tornar esses testes sem sentido
- Nunca teste se um float é igual a zero, ou qualquer outro float
- Em vez disso, teste se está dentro de um valor de ϵ mais ou menos do alvo
- Muitos problemas pedirão que você mostre uma resposta para um determinado número de dígitos de precisão à direita do ponto decimal
  - Aqui devemos distinguir entre *arredondamento* e *truncado*
  - O truncamento é exemplificado pela função floor, que converte um número real de um inteiro cortando a parte fracionária
  - Arredondamento é usado para obter um valor mais preciso para o dígito menos significativo
  - Para arredondar um número de X para k dígitos decimais, use a fórmula

$$round(X, k) = floor(10^k X + (1/2))/10^k$$

Frações

- Números racionais exatos x/y são melhores representados por pares de inteiros x, y, onde x é o numerador e y é o denominador da fração
- As operações aritméticas básicas sobre racionais $c = x_1/y_1$ e $d = x_2/y_2$ são fáceis de programar:
  - **Adição** - Devemos encontrar um denominador comum antes de adicionar frações:

$$c + d = \frac{x_1 y_2 + x_2 y_1}{y_1 y_2}$$

  - **Subtração** - O mesmo que adição, pois $c - d = c + (-1) \times d$:

$$c - d = \frac{x_1 y_2 - x_2 y_1}{y_1 y_2}$$

Frações

- **Multiplicação** – Como a multiplicação é uma adição repetida:

$$c \times d = \frac{x_1 x_2}{y_1 y_2}$$

- **Divisão** – Para dividir frações você multiplica pelo inverso do denominador:

$$c/d = \frac{x_1}{y_1} \times \frac{y_2}{x_2} = \frac{x_1 y_2}{y_1 x_2}$$

- A implementação sem atenção dessas operações leva a um perigo significativo de *overflows*
- É importante reduzir frações à sua representação mais simples, ou seja, substituir 2/4 por 1/2
- O algoritmo de Euclides para o cálculo do mdc (máximo divisor comum) é eficiente, muito simples de programar

Decimais

- A representação decimal dos números reais é apenas um caso especial dos números racionais
- Um número decimal representa a soma de dois números; a parte inteira à esquerda do vírgula decimal e a parte fracionária à direita da vírgula
- Assim, a fração representação dos primeiros cinco dígitos decimais de π é

$$3{,}1415 = (3/1) + (1415/10000)$$

  - O denominador da parte fracionária é 10^{i+1} se o dígito diferente de zero mais à direita estiver i casas à direita da vírgula

Manipulando Polinômios

- $P(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_i x^i + \cdots$, onde x é a variável e c_i é o coeficiente do i-ésimo termo x_i
- O grau de um polinômio é o maior i tal que c_i é diferente de zero
- A representação mais natural para um polinômio de grau n é como uma matriz de $n + 1$ coeficientes c_0 a c_n
- Avaliação:
  - Calcular $P(x)$ para algum dado x pode ser feito facilmente por força bruta, ou seja, computando cada termo $c_i \cdot n$ independentemente e somando-os
  - O problema é que isso vai custar $O(n^2)$ multiplicações onde $O(n)$ é suficiente
  - O segredo é notar que $x^i = x^{i-1}x$, então se calcularmos os termos do menor grau ao grau mais alto, podemos acompanhar o poder atual de x, e fugir com duas multiplicações por termo ($x^{i-1}x$, e então $c_i \cdot x^i$)
  - Alternativamente, pode-se empregar a regra de Horner, uma maneira ainda mais esperta de fazer o mesmo trabalho:
    $a_n x^n + a_{n-1} x^{n-1} + \cdot + a_0 = ((a_n x + a_{n-1})x + \cdot)x + a_0$

Adição/Subtração de polinômios

- Adicionar e subtrair polinômios é ainda mais fácil do que as mesmas operações em inteiros longos, uma vez que não há empréstimo ou carregamento
- Basta adicionar ou subtrair os coeficientes dos termos i de todo i de zero a grau máximo

Multiplicação e Divisão

- **Multiplicação**
  - O produto dos polinômios $P(x)$ e $Q(x)$ é a soma do produto de cada par de termos, onde cada termo vem de um polinômio diferente: $P(x) \times Q(x) = \sum_{i=0}^{degree(P)} \sum_{j=0}^{degree(Q)} (c_i c_j) x^{i+j}$
  - Essa operação de todos contra todos é chamada de convolução
  - Outras convoluções incluem multiplicação de inteiros (todos os dígitos contra todos os dígitos) e correspondência de strings (todas as posições possíveis da string padrão contra todas as posições de texto possíveis)
  - A transformada rápida de Fourier (FFT) que calcula convoluções em tempo $O(n \log n)$ em vez de $O(n^2)$
- **Divisão**
  - Dividir polinômios é um negócio complicado, já que os polinômios não são fechados sob divisão
  - Note que $1/x$ pode ou não ser pensado como um polinômio, já que é x^{-1}
  - $2x/(x^2 + 1)$ é uma função racional

Manipulação de polinômios

- Às vezes, os polinômios são *esparsos*, o que significa que tem muitos coeficientes $c_i = 0$
- Polinômios suficientemente esparsos devem ser representados como listas encadeadas de pares coeficiente/grau
- Polinômios multivariados são definidos em mais de uma variável
- O polinômio bivariado $f(x, y)$ pode ser representado por uma matriz C de coeficientes, tal que $C[i][j]$ é o coeficiente de $x^i y^j$

Obtenção de raízes de polinômio

- Dado um polinômio $P(x)$ e um número alvo t, o problema de encontrar raízes é identificar qualquer ou todos os x tais que $P(x) = t$
- Se $P(x)$ for um polinômio de primeiro grau, a raiz é simplesmente $x = (t - a_0)/a_1$, onde a_i é o coeficiente de x_i em $P(x)$
- Se $P(x)$ for um polinômio de segundo grau, então a equação quadrática se aplica:

$$x = \frac{-a_1 \pm \sqrt{a_1^2 - 4a_2(a_0 - t)}}{2a_2}$$

- Existem equações mais complicadas fórmulas para resolver polinômios de terceiro e quarto graus
- Não existe forma fechada para as raízes de equações de quinto grau ou de grau superior

Obtenção de raízes de polinômio

- Além de equações quadráticas, métodos numéricos são normalmente usados
- Há uma variedade de algoritmos de busca de raízes, incluindo os métodos de Bisseção e de Newton
- A ideia básica para encontar uma raiz é a da busca binária
  - Suponha que uma função $f(x)$ seja monotonicamente crescente entre l e u, o que significa que $f(i) \leq f(j)$ para todo $l \leq i \leq j \leq u$
  - Agora, suponha que queremos encontrar o x tal que $f(x) = t$
  - Podemos comparar $f((l + u)/2)$ com t
  - Se $t < f((l + u)/2)$, então a raiz está entre l e $(l + u)/2$; se não, está entre $(l + u)/2$ e u
  - Repete-se a análise até atingir um critério de convergência

Logaritmos

- Um logaritmo é simplesmente uma função exponencial inversa. Dizer que $b^x = y$ é equivalente a dizer que $x = \log_b y$
- O termo b é conhecido como a base do logaritmo
- O *natural* log, geralmente denotado $\ln x$, é um logaritmo de base $e = 2,71828\cdots$
  - O inverso de $\ln x$ é a função exponencial $\exp x = e^x$
  - Assim, compondo essas funções, obtemos $\exp(\ln x) = x$
- A base 10 ou *logaritmo comum*, geralmente denotado $\log x$
- Os logaritmos ainda são úteis para multiplicação, particularmente para exponenciação
- $\log_a(xy) = \log_a x + \log_a y \Rightarrow \log_a n^b = b \log_a y$
  $a^b = \exp(\ln(a^b)) = \exp(b \ln a)$
- Converter o logaritmo de uma base para outra: $\log_a b = \frac{\log_c b}{\log_c a}$

Bibliotecas matemáticas reais

Bibliotecas matemáticas em C/C++

```
#include <math.h>           /* include the math library */

double floor(double x);     /* chop off fractional part of x */
double ceil (double x);     /* raise x to next largest integer */
double fabs(double x);      /* compute the absolute value of x */

double sqrt(double x);      /* compute square roots */
double exp(double x);       /* compute e^x */
double log(double x);       /* compute the base-e logarithm */
double log10(double x);     /* compute the base-10 logarithm */
double pow(double x, double y);   /* compute x^y */
```

Bibliotecas matemáticas em Java

- A classe java **java.lang.Math** tem todas essas funções, mais obviamente uma função round para levar um real ao inteiro mais próximo

Referências

(Capítulo 5) Skiena, Steven S; Revilla, Miguel A. Programming challenges: the programming contest training manual. New York: Springer, 2003.