UNIVERSIDADE FEDERAL DE VIÇOSA
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
DEPARTAMENTO DE INFORMÁTICA

PROJETO FINAL DE CURSO

BIBLIOTECA PARA MATEMÁTICA SIMBÓLICA EM C++

Arthur Gonçalves do Carmo
Graduando em Ciência da Computação

Luiz Carlos de Abreu Albuquerque
(Orientador)

VIÇOSA – MINAS GERAIS
JUNHO – 2018

RESUMO

BIBLIOTECA PARA MATEMÁTICA SIMBÓLICA EM C++

Luiz Carlos de Abreu Albuquerque (Orientador)
Arthur Gonçalves do Carmo (Estudante)

RESUMO

Matemática simbólica é a área da computação que se preocupa com representar e manipular equações e expressões matemáticas de forma simbólica, em oposição aos métodos de manipulação por aproximação numérica. O objetivo do projeto é criar uma biblioteca para C++ contendo classes que representam simbolicamente alguns dos principais objetos matemáticos, como números inteiros e racionais, artimética modular, polinômios e expressões.

PALAVRAS-CHAVE

matemática simbólica; aritmética de precisão múltipla; expressões matemáticas;

ÁREA DE CONHECIMENTO

1.03.02.01- 8 - Matemática Simbólica

LINHA DE PESQUISA

DPI-040 – Algoritmos e Otimização Combinatória

1 – Introdução

A matemática simbólica é um campo já muito bem estudado na computação. A dificuldade de se criar métodos mais eficientes do que os já existentes levam a uma relativa ausência de opções para bibliotecas para aritmética de precisão múltipla e representação simbólica. O mesmo não pode ser dito, entretanto, sobre sistemas de manipulação algébrica simbólica, chamados CAS (*Computer Algebra System*), que são produtos de software completos para o mesmo fim, e possuem grande variedade, muitas vezes se diferenciando um do outro por trabalharem com campos bem distintos de aplicação da matemática.

É fácil notar que a matemática simbólica é mais interessante do ponto de vista teórico do que do prático. Para a maioria das aplicações, as aproximações em ponto flutuante são suficientemente precisas e, quando não são, é mais interessante a aritmética de precisão múltipla do que resultados simbólicos. As aplicações que se beneficiam da matemática simbólica são em sua maior parte da área da matemática e da física teórica. Além disso, a matemática simbólica pode ser considerada uma área pertinente aos limites da computação, e também uma forma de analisar a própria abordagem humana em relação à matemática.

1.2 – Objetivos

O objetivo geral deste trabalho é desenvolver uma biblioteca para matemática simbólica em C++, com classes para representar números inteiros, modulares e racionais, polinômios e expressões matemáticas, derivação e integração simbólicas.

Os objetivos específicos deste trabalho são:

- Estudar técnicas e algoritmos usados em computação simbólica
- Utilizar padrões de desenvolvimento para software livre

2 – Referencial Teórico

2.1 – Números de precisão múltipla

Os computadores modernos representam números inteiros como cadeias de bits, mais comumente 32 ou 64 bits, interpretadas como números inteiros em base 2. Essa abordagem tem a limitação de poder representar, em 64 bits, apenas números entre 0 e $2^{64} - 1$ (sem usar bit de sinal) ou entre -2^{63} e $2^{63} - 1$ (com bit de sinal). A ideia da aritmética de precisão múltipla é representar números números inteiros cujo tamanho é limitado apenas pela memória disponível no computador.

Uma excelente alternativa para trabalhar com números de precisão múltipla é a *GNU Multiple-Precision Library – GMP*, uma biblioteca escrita na linguagem C, altamente otimizada e que trabalha com números inteiros, racionais e ponto flutuante de precisão arbitrária.

É pertinente ao escopo do trabalho a criação de classes para representar números inteiros e racionais de precisão múltipla.

2.2 – Expressões algébricas

2.2.1 – Polinômios

Polinômios são expressões que consistem de variáveis e coeficientes, envolvendo apenas as operações de adição, subtração, multiplicação e expoentes inteiros não negativos. Um polinômio de grau N pode ser representado da forma:

$$\sum_{i=0}^{N} a_i x^i$$

Onde cada coeficiente a_i é um número real. É interessante que, para fins de representação, podemos considerar que a_i possa ser também um polinômio, o que nos leva a uma representação recursiva de polinômios com mais de uma variável:

$$\sum_{i=0}^{N}\sum_{j=0}^{M} a_{ij} x^i y^j = \sum_{i=0}^{N} c_i x^i, \quad \text{onde} \quad c_i = \sum_{j=0}^{M} a_{ij} y^j$$

2.3 – *GNU Build System*[5]

O *GNU Build System* uma convenção para organização, construção (build) e instalação de pacotes de forma padronizada. Idealmente a instalação de um pacote que segue o *GNU Build System* em qualquer ambiente necessita apenas dos comandos *./configure*, *make*, e *make installI.*

3 – Metodologia

3.1 – Representação

3.1.1 – Números inteiros

Um número inteiro é representado como uma dupla $(\sum_{i=0}^{N-1} d_i b^i, S)$, onde *b* é a base é a base de representação, *N* é o número de dígitos do número, $0 \leqslant d_i < b$ é o *i*-ésimo dígito e *S* é sinal do número.

A transcrição dessa representação para uma linguagem de programação consiste em um arranjo onde cada posição armazena um dígito na base escolhida, e um valor booleano para armazenar o sinal. Nesse projeto, os números são representados em *little endian*, os dígitos mais significativos possuem índices maiores.

A base de representação escolhida para esse projeto é a base 10^9, porque bases que são potência de 10 facilitam a leitura e escrita dos números em base 10, e alguns dos algoritmos necessitam que o valor do quadrado da base (b^2) seja representável por um tipo básico da linguagem. Utilizando um sistema de 64 bits, temos $(10^9)^2 = 10^{18} < 2^{64} < 10^{20} = (10^{10})^2$.

Além disso, a classe ainda guarda a quantidade de dígitos (em base 10^9) do número.

Nome da classe ***num_z***.

3.1.2 – Números racionais

Os números racionais são representados como uma tripla (N_1, N_2, S), onde N_1 é um número inteiro não negativo, N_2 é um número inteiro positivo e *S* é o sinal do número.

No projeto, a classe de números racionais utiliza a classe de números inteiros para representar N_1 e N_2 e um byte para representar S. Os números são representados em sua forma irredutível. Nome da classe ***num_q***.

3.1.3 – Aritmética modular

Os números usados para aritmética modular são representados com uma dupla (N, B), onde *N* é um número inteiro e *B* é a base da aritmética. Um número *x mod B* é o resto da divisão de *x* por *B*.

No projeto, a classe de números modulares é uma classe *template* que recebe a base como *tipo* e possui um membro que é um número inteiro para armazenar *N*. Nome da classe ***num_zm<N>***.

3.1.4 – Tuplas de divisão inteira

O resultado da divisão inteira de um inteiro *M* por um inteiro *N*, resulta em um quociente *q* e um resto *r* de forma que *M* = *q***N* + *r*.

A forma usada para representar a dupla (*q*, *r*), é por meio de duas estruturas de duplas, chamadas ***div_tuple*** (usada para a divisão) e ***mod_tuple*** (usada para a operação de resto da divisão).

A estrutura ***div_tuple*** armazena o quociente e resto da divisão inteira de *M* por *N* de forma que o resto da divisão é sempre não negativo, enquanto a estrutura ***mod_tuple*** armazena o quociente e resto da divisão inteira de *M* por *N* de forma que o resto da divisão é zero ou possui o mesmo sinal de *N*.

3.2 – Manipulação

3.2.1 – Números inteiros

3.2.1.1 – Leitura e escrita

A classe lê e escreve números inteiros em base 2 (prefixado por 0b), base 10 e base 16 (prefixado por 0x) no formato *little endian* sem espaços ou pontuação entre os dígitos. Não há nenhuma forma de tratamento para entradas não formatadas corretamente.

3.2.1.2 – Operações e operadores

Os algoritmos usados para as operações aritméticas básicas são os descritos por Donald Knuth[1].

O símbolo (**-**) indica que chamada sem argumentos é aceita.

Onde especificado por *, a função aceita os seguintes tipos como argumento:

int
long long int
unsigned int
unsigned long long int
const char *
div_tuple
mod_tuple
num_z

3.2.1.2.1 – Membros

Função	Descrição	Argumentos
(construtor)	Constroi objeto num_z	- *
(destrutor)	Destrutor padrão	-
operator=	Atribui o valor do número do lado direito e retorna uma referência para o objeto	*
lsd	Retorna o último dígito em base 10^9	-
abs	Retorna o valor absoluto do número	-
sign	Retorna 1 se o número é negativo, 0 caso contrário	-
negative	Retorna o negativo do valor absoluto do número	-
make_abs	Muda o número para seu valor absoluto e retorna uma referência para o objeto	-
flip_sign	Inverte o sinal do número e retorna uma referência para o objeto	-
make_negative	Muda o número para o negativo de seu valor absoluto e retorna uma referência para o objeto	-
operator- (unário)	Retorna o valor do número com sinal invertido	-
operator+ (unário)	Retorna o valor do número	-
operator++ (esq.)	Incrementa o número em um e retorna uma referência para o objeto	-
operator++ (dir.)	Retorna o valor do número incrementado em um	-
operator-- (esq.)	Decrementa o número em um e retorna uma referência para o objeto	-
operator-- (dir.)	Retorna o valor do número decrementado em um	-
operator+=	Soma ao número do lado direito e retorna uma referência para o objeto	*
operator-=	Subtrai o número do lado direito e retorna uma referência para o objeto	*
operator*=	Multiplica pelo número do lado direito e retorna uma referência para o objeto	*
operator/=	Recebe o quociente que produz resto não negativo da divisão pelo número do lado direito e retorna uma referência para o objeto	*
operator%=	Recebe o resto, com mesmo sinal do divisor, da divisão inteira pelo número do lado direito e retorna uma referência para o objeto	*

operator+ (binário)	Retorna o valor da soma com número do lado direito	*
operator- (binário)	Retorna o valor da subtração pelo número do lado direito	*
operator*	Retorna o valor do produto pelo número do lado direito	*
operator/	Retorna uma dupla, contendo o quociente e resto da divisão inteira pelo número do lado direito, com resto positivo	*
operator%	Retorna uma dupla, contendo o quociente e resto da divisão inteira pelo número do lado direito, resto com sinal do divisor	*
operator==	Retorna verdadeiro se os números forem iguais e falso, caso contrário	*
operator!=	Retorna falso se os números forem iguais e verdadeiro, caso contrário	*
operator<=	Retorna falso se o número do lado direito for menor, e verdadeiro caso contrário	*
operator>=	Retorna falso se o número do lado direito for maior, e verdadeiro caso contrário	*
operator<	Retorna verdadeiro se o número do lado direito for maior, e falso caso contrário	*
operator>	Retorna verdadeiro se o número do lado direito for menor, e falso caso contrário	*
pow	Retorna o valor da exponenciação pelo número passado como argumento	int
abs_eq	Compara igualdade em valor absoluto	num_z
abs_neq	Compara desigualdade em valor absoluto	num_z
abs_geq	Compara desigualdade maior ou igual em valor absoluto	num_z
abs_leq	Compara desigualdade menor ou igual em valor absoluto	num_z
abs_gt	Compara desigualdade maior que em valor absoluto	num_z
abs_lt	Compara desigualdade menor que em valor absoluto	num_z

3.2.1.2.2 – Não membros

Função	Descrição	Argumento
hex	Indica que o número deve ser escrito em hexadecimal	num_z
bin	Indica que o número deve ser escrito em binário	num_z
z_gcd	Retorna o valor do MDC(máximo divisor comum) dos dois argumentos	num_z, num_z
operator<<	Escrita na saída padrão	ostream, num_z
operator>>	Leitura na entrada padrão	istream, num_z

3.2.2 – Números Racionais

3.2.2.1 – Entrada e saída

A classe espera a entrada de dois números inteiros (entrada vista em 3.2.1.1) separados por um espaço em branco, o primeiro será o numerador e o segundo o denominador.

A saída imprime o numerador e o denominador, nesta ordem, separados por um caracter ***/***.

3.2.2.2 – Operações e operadores

O símbolo (**-**) indica que chamada sem argumentos é aceita.

Onde especificado por *, a função aceita os seguintes tipos como argumento:

long long int
num_z
num_q

Função	Descrição	Argumentos
(construtor)	Constroi objeto num_q	- *
numerator	Retorna valor do numerador, com sinal do número	-
denominator	Retorna o denominador, em valor absoluto	-
operator+ (unário)	Retorna o valor do número	-
operator- (unário)	Retorna o valor do número com sinal invertido	-
operator=	Atribui o valor do número do lado direito e retorna uma referência para o objeto	*
operator+=	Soma ao número do lado direito e retorna uma referência para o objeto	*
operator-=	Subtrai o número do lado direito e retorna uma referência para o objeto	*
operator*=	Multiplica pelo número do lado direito e retorna uma referência para o objeto	*
operator/=	Multiplica pelo inverso número do lado direito e retorna uma referência para o objeto	*
operator+ (binário)	Retorna o valor da soma com número do lado direito	*
operator- (binário)	Retorna o valor da subtração pelo número do lado direito	*

operator*	Retorna o valor do produto pelo número do lado direito	*
operator/	Retorna o valor do produto pelo inverso do número do lado direito	*
pow	Retorna o valor da exponenciação pelo número passado como argumento	int
abs	Retorna o valor absoluto do número	-
sign	Retorna 1 se o número é negativo, 0 caso contrário	-
negative	Retorna o negativo do valor absoluto do número	-
make_abs	Muda o número para seu valor absoluto e retorna uma referência para o objeto	-
flip_sign	Inverte o sinal do número e retorna uma referência para o objeto	-
make_negative	Muda o número para o negativo de seu valor absoluto e retorna uma referência para o objeto	-
inverse	Retorna o inverso multiplicativo do número	-
operator==	Retorna verdadeiro se os números forem iguais e falso, caso contrário	*
operator!=	Retorna falso se os números forem iguais e verdadeiro, caso contrário	*
operator<=	Retorna falso se o número do lado direito for menor, e verdadeiro caso contrário	*
operator>=	Retorna falso se o número do lado direito for maior, e verdadeiro caso contrário	*
operator<	Retorna verdadeiro se o número do lado direito for maior, e falso caso contrário	*
operator>	Retorna verdadeiro se o número do lado direito for menor, e falso caso contrário	*

3.2.3 – Aritmética Modular

3.2.3.1 – Entrada e saída

Funciona exatamente como um número inteiro (3.2.1.1).

3.2.3.2 – Operações e operadores

Todas as operações são realizadas dentro da aritmética modular.

O símbolo (**-**) indica que chamada sem argumentos é aceita.
Onde especificado por *, a função aceita os seguintes tipos como argumento:

int
long long int
unsigned int
unsigned long long int
const char *
div_tuple
mod_tuple
num_z

Função	Descrição	Argumentos
(construtor)	Constroi objeto num_zm	- *
operator=	Atribui o valor do número do lado direito e retorna uma referência para o objeto	*
raw_value	Retorna o valor como um número inteiro	-
operator- (unário)	Retorna o valor do número com sinal invertido	-
operator+ (unário)	Retorna o valor do número	-
operator++ (esq.)	Incrementa o número em um e retorna uma referência para o objeto	-
operator++ (dir.)	Retorna o valor do número incrementado em um	-
operator-- (esq.)	Decrementa o número em um e retorna uma referência para o objeto	-
operator-- (dir.)	Retorna o valor do número decrementado em um	-
operator+=	Soma ao número do lado direito e retorna uma referência para o objeto	*
operator-=	Subtrai o número do lado direito e retorna uma referência para o objeto	*
operator*=	Multiplica pelo número do lado direito e retorna uma referência para o objeto	*

operator/=	Multiplica pelo inverso multiplicativo do número do lado direito e retorna uma referência para o objeto	*
operator+ (binário)	Retorna o valor da soma com número do lado direito	*
operator- (binário)	Retorna o valor da subtração pelo número do lado direito	*
operator*	Retorna o valor do produto pelo número do lado direito	*
operator/	Retorna o valor do produto pelo inverso multiplicativo do número do lado direito	*
valid	Retorna verdadeiro se o número é válido, falso caso contrário	-
has_inverse	Retorna verdadeiro se tem inverso multiplicativo, falso caso contrário	-
inverse	Retorna o inverso multiplicativo, se houver. Retorna lixo caso contrário	-
operator==	Retorna verdadeiro se os números forem iguais e falso, caso contrário	*
operator!=	Retorna falso se os números forem iguais e verdadeiro, caso contrário	*
operator<=	Retorna falso se o número do lado direito for menor, e verdadeiro caso contrário	*
operator>=	Retorna falso se o número do lado direito for maior, e verdadeiro caso contrário	*
operator<	Retorna verdadeiro se o número do lado direito for maior, e falso caso contrário	*
operator>	Retorna verdadeiro se o número do lado direito for menor, e falso caso contrário	*
pow	Retorna o valor da exponenciação pelo número passado como argumento	int

Referências

[1] KNUTH, D. **The Art Of Computer Programming Volume 2 Seminumerical Algorithms**: 1998.

[2] LISKA, R. et al. **Computer Algebra, Algorithms, Systems and Applications**: 1999.

[3] COHEN, J. S. **Computer Algebra and Symbolic Computation: Elementary Algorithms**: 2002.

[4] COHEN, J. S. **Computer Algebra and Symbolic Computation: Mathematical Methods**: 2003.

[5] http://www.gnu.org/software/automake/manual/html_node/GNU-Build-System.html#GNU-Build-System