1ª Lista de Exercícios

Estrutura de Dados

Prof. Hamilton José Brumatto

Tipos Abstratos de Dados

1. Retas
   As retas e outros elementos geométricos são bons exemplos de aplicação de estrutura de dados. Uma reta é definida por uma equação: $ax + by + cz = d$, ou seja, os coeficientes a, b, c e d definem unicamente uma reta. Vamos nos ater a uma reta no plano, neste caso z deixa de ser uma variável, e podemos nos restringir $ax + by = c$, neste caso uma estrutura contendo a, b e c definem uma reta no plano. Em especial, se todos os termos a, b e c forem multiplicados pelo mesmo escalar, chegamos a uma nova parametrização da mesma reta. Podemos, então, multiplicar por $1/c$, assim ficamos com $(a/c)x + (b/c)y = 1$, e sem perda de generalidade, podemos chamar esta reta de "$ax + by = 1$", o que resulta que apenas com a e b podemos descrever uma reta. Este tema é explorado em geometria computacional, mas vamos ficar somente com isto.
   Vamos, então, construir uma estrutura de dados com os parâmetros a e b para representar a reta. Queremos algumas operações para a reta, implemente-as:

   - C

   ```
   typedef struct reta_s {
     double a, b;
   } reta;

   void criar(reta *r, double a, double b); // deve criar a reta ax + by = 1
   void criar(reta *r, double a, double b, double c); // deve criar a reta $(a/c)x + (b/c)y = 1
   int horizontal(reta r); // retorna 1 se r.a == 0, ou 0 caso contrário
   int vertical(reta r); // retorna 1 se r.b == 0, ou 0 caso contrário
   int paralela(reta r, reta s); // retorna 1 se (r.a==s.a== 0, ou r.b==s.b==0, ou r.a/r.b==s.a/s.b)
                                 // ou 0 caso contrário
   ```

   - C++

   ```
   class reta {
   private:
    double a, b;
   public:
    reta(); // deve criar a reta x = 0
    reta(a,b); // deve criar a reta ax + by = 1
    reta(a,b,c); // deve criar a reta (a/c)x + (b/c)y = 1
    bool horizontal(); // retorna true se a == 0, false caso contrário
    bool vertical(); // retorna true se b == 0, false caso contrário
    bool paralela(reta r); // retorna true se (a == r.a == 0, ou b == r.b == 0, ou a/b==r.a/r.b)
                           // false caso contrário.
   };
   ```

2. Números complexos. Conforme apresentado em sala, vamos fazer a implementação dos números complexos e suas operações:

Atribuição	$(a + bi) \rightarrow (a + bi)$
Simétrico	$-(a + bi) \rightarrow (-a - bi)$
Conjugado	$(a + bi) \rightarrow (a - bi)$
Soma	$(a + bi) + (c + di) = (a + c) + (b + d)i$
Subtração	$(a + bi) - (c + di) = (a - c) + (b - d)i$
Multiplicação	$(a + bi) \times (c + di) = (ab - cd) + (ad + bc)i$
Divisão*	$\dfrac{a + bi}{c + di} = \dfrac{(ac + bd) + (-ad + bc)i}{c^2 + d^2}$
Módulo	$\lvert a + bi \rvert = \sqrt{a^2 + b^2}$
Real	$(a + bi) \rightarrow a$
Imaginário	$(a + bi) \rightarrow b$

   - C

   ```
   typedef struct complexo_s {
     double real, imag;
   } complexo;

   void criar(complexo *c, double a, double b); // c = real + imag i
   ```

```
void atribuir(complexo *c, complexo z); // c = z
void simetrico(complexo *c, complexo z); // c = -z
void conjugado(complexo *c, complexo z); // c = z^* --> a.real = z.real; c.imag = -z.imag
void soma(complexo *c, complexo a, complexo b); // c = a + b
void subtracao(complexo *c, complexo a, complexo b); // c = a - b
void multiplicacao(complexo *c, complexo a, complexo b); // c = a * b
void divisao(complexo *c, complexo a, complexo b); // c = a / b
double modulo(complexo c); // return |c|
double real(complexo c); // return a.real
double imag(complexo c); // return b.imag
\begin{verbatim}
```

- C++

```
class strutcture complexo {
private:
 double real, imag;

public:
 complexo(); // 0 + 0i
 complexo(double real, double imag); // real + imag i
 complexo conjugado(); return a - bi
 complexo operator=(complexo c); // real + imag i = c; return real + imag i
 void operator-(); // -a -bi; return -a -bi
 void operator+(complexo c); // return (real+imag i) + c
 void operator-(complexo c); // return (real+imag i) - c
 void operator*(complexo c); // return (real+imag i) * c
 void operator/(complexo c); // return (real+imag i)/c
 double modulo(); // return raiz(real^2 + imag^2)
 double real(); // return real
 double imag(); // return imag
};
```

3. Números racionais
O conjunto numérico $\mathcal{Q}$ dos racionais difere dos $\mathcal{R}$ reais, pois todos os elementos deste conjunto pode ser escrito como uma fração de dois inteiros. Então, a forma mais exata de representar qualquer elemento deste conjunto é na forma de uma fração. Para tanto, estrutura de dados pode utilizar este fato.
Um detalhe importante é que um valor racional pode assumir diversas formas, por exemplo: $\frac{1}{2} = \frac{2}{4} = \frac{5}{10} = \ldots$ Existe, entre estas uma forma que é a irredutível. Por exemplo, na representação $\frac{2}{4}$, podemos simplificar dividindo o numerador e denominador por 2, e chegamos a $\frac{1}{2}$. De uma forma simples uma representação é redutível se existe um Máximo Divisor Comum (veja no final da lista o algoritmo de Euclides para achar o MDC) entre o numerador e denominador diferente de 1. Neste caso dividimos ambos pelo MDC e chegamos à forma irredutível. Em especial, podemos considera a forma irredutível do racional 0, como sendo: $\frac{0}{1}$. Construa uma implementação para números racionais, e funções que realizam;

- Iniciação: cria um número com numerador e denominador.
- Redução: transforma o número em sua forma irredutível.
- Comparação: **int** compara(racional num1, racional num2); $\rightarrow$ retorna 1 se $num1 < num2$, -1 se $num1 > num2$ ou 0 se $num1 = num2$.
- Construa um vetor de números racionais.
- Faça uma rotina que busque o maior número no vetor.
- C

```
typedef struct racional_s {
 int num, den; // den != 0 sempre
} racional;

void criar(racional *r, int num, int den); // r = num/den --> reduzir
void reduzir(racional *r); // r --> em sua forma irredutível.
int compara(racional r, racional s; // return -1 se se r > s, 0 se r = s, ou 1 se r < s
```

- C++

```
class racional {
private:
  int num, den;
public:
  racional(); // 0/1
  racional(int num, int den); // num/den --> reduzir
  void reduzir(); // (num/MDC(num,den)) / (den/MDC(num,dem))
  bool operator<(racional r); // return true se < r; false caso contrário
  bool operator==(racional r); // return true se == r; false caso contrário
  bool operator>(racional r); // return true se > r; false caso contrário
};
```

Apoio: Algoritmo de Euclides para encontrar o MDC: Este algoritmo consiste em pegar o resto da divisão entre os dois números que queremos achar o MDC e sucessivamente o resto até que este seja 0. O MDC é o último resto não nulo. Por exemplo: 48 e 30.

$48|30 = 18 \rightarrow 30|18 = 12 \rightarrow 18|12 = 6 \rightarrow 12|6 = 0$

Conclui-se que o MDC de 48 e 30 é 6.

```
int mdc(int a, int b) {
  if(a < b) { // troca se a < b
      a^=b;
      b^=a;
      a^=b;
  }
  int r = a%b
  while(r > 0) {
   a = b;
   b = r;
   r = a%b
  }
  return b;
}
```