UNIVASF

UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO

Algoritmos e Estrutura de Dados

Professor: Marcelo Santos Linder

E-mail: marcelo.linder@univasf.edu.br

Ementa

Noções de abstração de dados. Pilhas, filas, listas, árvores binárias e árvores AVL: alocação estática e dinâmica e algoritmos de inserção, remoção e consulta. Algoritmos recursivos. Grafos. Tabelas de hash: alocação estática e dinâmica e algoritmos de inserção, remoção e consulta. Introdução à complexidade de algoritmos, com análise de complexidade no tempo e no espaço.

UNIVASF

Referências Bibliográficas

BIBLIOGRAFIA BÁSICA:

1. AARON M.T.; LANGSAM, Y.; MOSHA, J.A. Estruturas de Dados Usando C. Pearson Education, 2005.
2. CORMEN T.H.; LEISERSON C.E.; RIVEST, R.L. Algoritmos - Teoria e Prática. 3ª ed. Elsevier, 2012.
3. SEDGEWICK, R. Algorithms in C. 3ª ed. Pearson Education, 1998.

BIBLIOGRAFIA COMPLEMENTAR:

1. WIRTH, N. Algoritmos e Estruturas de Dados. LTC, 1999.
2. ZIVIANI, N. Projeto de Algoritmos - com implementação em Pascal e C. Cengage Learning, 3ª Edição 2010.

Forma de Avaliação

- A avaliação será realizada mediante a aplicação de três provas práticas.

A média do aluno na disciplina será calculada através do computo da média aritmética obtida com base nas notas do discente.

Informações Gerais

- Material de apoio
  - **Videoaulas, PDF's dos slides utilizados, datas de avaliações, grupo do WhatsApp e demais informações referentes à disciplina encontram-se nos links a seguir.**

UNIVASF

Informações Gerais

Canal no YouTube

Grupo no Whatsapp da disciplina

Página do professor

http://univasf.edu.br/~marcelo.linder/

Noções de abstração de dados

UNIVASF

Programa

Um programa pode ser visto como a especificação formal da solução de um problema. N.Wirth expressa em sua equação

$$\textit{programa} = \textit{algoritmo} + \textit{estruturas de dados}$$

onde: o algoritmo contém a lógica do programa e os dados são organizados em estruturas de dados.

UNIVASF

Estruturas de Dados

A qualidade da solução de um problema depende, entre outros fatores, da forma como estão organizados os dados relevantes.

- Encontrar o número do telefone de um contato em uma agenda armazenada em um caderno na forma de uma lista.
- Representação interna de uma string:

4	G	A	T	O
G	A	T	O	\0

Toda uma classe de modelos desenvolveu-se, ao longo do tempo, com o objetivo de viabilizar o processamento de dados.

UNIVASF

Estruturas de Dados

Estruturas de dados são formas genéricas de se estruturar informação de modo a serem registradas e processadas pelo computador.

Ex.:

- **vetores;**
- **lista ordenada;**
- **árvores;**
- **grafos, etc.**

Contudo, estas só adquirem significado quando associadas a um conjunto de **operações**, que visam, de um modo geral, manipulá-las (algoritmos).

UNIVASF

Tipos Abstratos de Dados

Embora os termos “tipo de dado”, “estrutura de dados” e “tipo abstrato de dados” em essência se refiram aos mesmos objetos formais, eles são usados em contextos próprios.

O termo “tipo de dado” é usado no contexto de uma linguagem de programação (tipos primitivos) e está associado a um método de interpretar um padrão de bits.

UNIVASF

Tipos Abstratos de Dados

O termo “tipo abstrato de dado” (TAD) denota um modelo junto com um conjunto de operações definidas sobre o modelo. (tipo + operações válidas).

Por fim, “estrutura de dados” é uma forma concreta de se implementar um TAD, ou seja, uma representação computacional do modelo matemático em questão.

UNIVASF

Tipos Abstratos de Dados

Para viabilizar a implementação de tipos abstratos de dados vamos nos valer das estruturas em C. Para exemplificar este conceito, definiremos um TAD RACIONAL.

Um número racional pode ser expresso como o quociente de dois inteiros. Definiremos a operação de criação e multiplicação de números racionais.

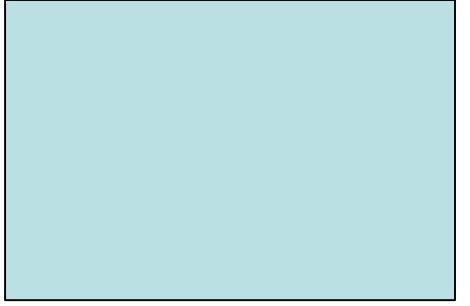
UNIVASF

Tipos Abstratos de Dados

```
typedef struct
{
   int num;
   int den;
}RACIONAL;
void criar_racional (int, int, RACIONAL *);
void multiplicar_racionais (RACIONAL *,
   RACIONAL *, RACIONAL *);
```

UNIVASF

```
void criar_racional (int n, int d, RACIONAL *r)
{
   r->num = n;
   (*r).den = d;
}
void multiplicar_racionais (RACIONAL *a, RACIONAL
    *b, RACIONAL *c)
{
   c->num = a->num * b->num;
   c->den = a->den * b->den;
}
```



Tipos Abstratos de Dados - Exercício

Implemente as operações de soma e verificação de equivalência entre elementos do TAD RACIONAL, definido anteriormente.

UNIVASF

Tipos Abstratos de Dados - Exercício

```
void somar_racionais (RACIONAL *a,
RACIONAL *b, RACIONAL *c)
{
   c->num = a->num * b->den + b->num * a->den;
   c->den = a->den * b->den;
}
int equivalencia_racionais (RACIONAL *a, RACIONAL *b)
{
   return a->num * b->den == a->den * b->num;
}
```

UNIVASF

```
main() {
   int n1, n2;
   char op;
   RACIONAL r1, r2, r3;
   printf ("A operacao: \n");
   scanf ("%d/%d %c ", &n1, &n2, &op);
   criar_racional (n1, n2, &r1);
   scanf ("%d/%d", &n1, &n2);
   criar_racional (n1, n2, &r2);
   switch (op) {
      case '+':  somar_racionais(&r1, &r2, &r3);
           break;
      case '*':  multiplicar_racionais(&r1, &r2, &r3);
           break;
      case '=':  printf ("%s",equivalencia_racionais(&r1, &r2)?"iguais":"diferentes");
           exit();
   }
   printf ("= %d/%d", r3.num, r3.den);
}
```

Tipos Abstratos de Dados

A forma apresentada para manipulação de racionais é satisfatória?

Não.

Por que?

Devido ao limite de representatividade dos inteiros. Os racionais deveriam ser representados em seus números mínimos.

Uma rotina conhecida como algoritmo de Euclides pode ser usada para reduzir qualquer fração da forma numerador/ denominador a seus termos mínimos.

UNIVASF

Tipos Abstratos de Dados

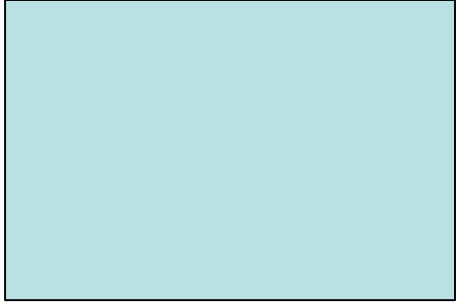
Essa rotina pode ser descrita como:

1. Seja a o maior entre o numerador e o denominador e b o menor.
2. Divida a por b, encontrando um quociente q e um resto r (isto é, $a = q * b + r$).
3. Defina $a = b$ e $b = r$.
4. Repita os passos 2 e 3 até que b seja igual a 0.
5. Divida tanto o numerador quanto o denominador pelo valor de a.

Implemente esta rotina em C.

UNIVASF

```
void reduzir_racional (RACIONAL *inrat,
RACIONAL *outrat)
{
   int a, b, rem;
   if (inrat->num > inrat->den)
   {
      a = inrat->num;
      b = inrat->den;
   }
   else
   {
      a = inrat->den;
      b = inrat->num;
   }
```



```
  while (b)
  {
    rem = a%b;
    a = b;
    b = rem;
  }
  outrat->num = inrat->num/a;
  outrat->den = inrat->den/a;
}
```